Защита информации в компьютерных системах

Колесников Дмитрий Геннадьевич

http://web-protect.net/

Вступление. Информация как объект защиты

Развитие современных информационных технологий сопровождается ростом числа компьютерных преступлений и связанных с ними хищений информации, а также материальных потерь. По результатам одного исследования около 58% опрошенных пострадали от компьютерных взломов за последний год. Примерно 18% опрошенных из этого числа заявляют, что потеряли более миллиона долларов в ходе нападений, более 66% потерпели убытки в размере 50 тыс. долларов. Свыше 22% атак были нацелены на промышленные секреты или документы, представляющие интерес прежде всего для конкурентов.

Федеральным законом "Об информации, информатизации и защите информации" определено, что информационные ресурсы, т.е. отдельные документы или массивы документов, в том числе и в информационных системах, являясь объектом отношений физических, юридических лиц и государства, подлежат обязательному учету и защите, как всякое материальное имущество собственника. При этом собственнику предоставляется право самостоятельно в пределах своей компетенции устанавливать режим защиты информационных ресурсов и доступа к ним.

Закон также устанавливает, что "конфиденциальной информацией считается такая документированная информация, доступ к которой ограничивается в соответствии с законодательством Российской Федерации". При этом федеральный закон может содержать прямую норму, согласно которой какие-либо сведения относятся к категории конфиденциальных или доступ к ним ограничивается. Так, Федеральный закон "Об информации, информатизации и защите информации" напрямую относит к категории конфиденциальной информации персональные данные (информацию о гражданах). Закон "О банках и банковской деятельности в РФ" ограничивает доступ к сведениям по операциям, счетам и вкладам клиентов и корреспондентов банков (статья 25).

Однако не ко всем сведениям, составляющим конфиденциальную информацию, применима прямая норма. Иногда законодательно определяются только признаки, которым должны удовлетворять эти сведения. Это в частности относится к служебной и коммерческой тайне, признаки которых определяются Гражданским кодексом РФ (статья 139):

- соответствующая информация неизвестна третьим лицам;
- к ней нет свободного доступа на законном основании;
- меры по обеспечению ее конфиденциальности принимает собственник информации.

В настоящее время отсутствует какая-либо универсальная методика, позволяющая четко соотносить ту или иную информацию к категории коммерческой тайны. Исходить можно только из принципа экономической выгоды и безопасности предприятия - чрезмерная "засекреченность" приводит к необоснованному удорожанию необходимых мер по защите информации и не способствует развитию бизнеса. Тогда как широкая открытость может привести к большим финансовым потерям или разглашению тайны. Законом "О коммерческой тайне" права по отнесению информации к категории коммерческой тайны представлены руководителю юридического лица.

Федеральный закон "Об информации, информатизации и защите информации", определяя нормы, согласно которых сведения относятся к категории конфиденциальных, устанавливает и цели защиты информации:

- предотвращение утечки, хищения, искажения, подделки информации;
- предотвращение несанкционированного уничтожения и блокирования информации;
- сохранение государственной тайны, конфиденциальности документированной информации.

Стандарты и рекомендации, рассмотренные выше, образуют базис понятий, на котором строятся все работы по обеспечению информационной безопасности. В то же время эти документы ориентированы в первую очередь на производителей и "оценщиков" систем и в гораздо меньшей степени - на пользователей.

Стандарты и рекомендации статичны, причем статичны, по крайней мере, в двух аспектах. Во-первых, они не учитывают постоянной перестройки защищаемых систем и их окружения. Во-вторых, они не содержат практических рекомендаций по формированию режима безопасности. Информационную безопасность нельзя купить, ее приходится каждодневно поддерживать, взаимодействуя при этом не только и не столько с компьютерами, сколько с людьми.

Таким образом, стандарты и рекомендации не дают ответов на два главных, с практической точки зрения, вопроса:

1. Как приобретать (комплектовать) информационную систему масштаба предприятия, чтобы ее можно было сделать безопасной?
2. Как практически сформировать режим безопасности и поддерживать его в условиях постоянно меняющегося окружения и структуры самой системы?

Иными словами, стандарты и рекомендации являются лишь отправной точкой на длинном и сложном пути защиты информационных систем организаций.

Для поддержания режима информационной безопасности особенно важны аппаратно-программные меры, поскольку основная угроза компьютерным системам исходит от самих этих систем (сбои оборудования, ошибки программного обеспечения, промахи пользователей и администраторов и т.п.).

Аппаратно-программные способы защиты - это и есть основная тема данного курса. А весь курс будет состоять из пяти крупных разделов:

1. Информационная безопасность компьютерных систем
2. Криптосистемы

3. Идентификация и аутентификация
4. Электронная цифровая подпись
5. Защита от удаленных атак через Internet

В завершении вступительной части не могу не сказать:

Не существует абсолютной защиты. Всякая защита измеряется временем взлома.

Информационная безопасность

Быстро развивающиеся компьютерные информационные технологии вносят заметные изменения в нашу жизнь. Информация стала товаром, который можно приобрести, продать, обменять. При этом стоимость информации часто в сотни раз превосходит стоимость компьютерной системы, в которой она хранится.

От степени безопасности информационных технологий в настоящее время зависит благополучие, а порой и жизнь многих людей. Такова плата за усложнение и повсеместное распространение автоматизированных систем обработки информации.

Под ***информационной безопасностью*** понимается защищенность информационной системы от случайного или преднамеренного вмешательства, наносящего ущерб владельцам или пользователям информации.

На практике важнейшими являются три аспекта информационной безопасности:

- ***доступность*** (возможность за разумное время получить требуемую информационную услугу);
- ***целостность*** (актуальность и непротиворечивость информации, ее защищенность от разрушения и несанкционированного изменения);
- ***конфиденциальность*** (защита от несанкционированного прочтения).

Нарушения доступности, целостности и конфиденциальности информации могут быть вызваны различными опасными воздействиями на информационные компьютерные системы.

Основные угрозы информационной безопасности

Современная информационная система представляет собой сложную систему, состоящую из большого числа компонентов различной степени автономности, которые связаны между собой и обмениваются данными. Практически каждый компонент может подвергнуться внешнему воздействию или выйти из строя. Компоненты автоматизированной информационной системы можно разбить на следующие группы:

- ***аппаратные средства*** - компьютеры и их составные части (процессоры, мониторы, терминалы, периферийные устройства - дисководы, принтеры, контроллеры, кабели, линии связи и т.д.);
- ***программное обеспечение*** - приобретенные программы, исходные, объектные, загрузочные модули; операционные системы и системные программы (компиляторы, компоновщики и др.), утилиты, диагностические программы и т.д.;
- ***данные*** - хранимые временно и постоянно, на магнитных носителях, печатные, архивы, системные журналы и т.д.;
- ***персонал*** - обслуживающий персонал и пользователи.

Опасные воздействия на компьютерную информационную систему можно подразделить на случайные и преднамеренные. Анализ опыта проектирования, изготовления и эксплуатации информационных систем показывает, что информация подвергается различным случайным воздействиям на всех этапах цикла жизни системы. Причинами ***случайных воздействий*** при эксплуатации могут быть:

- аварийные ситуации из-за стихийных бедствий и отключений электропитания;
- отказы и сбои аппаратуры;
- ошибки в программном обеспечении;
- ошибки в работе персонала;
- помехи в линиях связи из-за воздействий внешней среды.

Преднамеренные воздействия - это целенаправленные действия нарушителя. В качестве нарушителя могут выступать служащий, посетитель, конкурент, наемник. Действия нарушителя могут быть обусловлены разными мотивами:

- недовольством служащего своей карьерой;
- взяткой;
- любопытством;
- конкурентной борьбой;
- стремлением самоутвердиться любой ценой.

Можно составить гипотетическую модель потенциального нарушителя:

- квалификация нарушителя на уровне разработчика данной системы;
- нарушителем может быть как постороннее лицо, так и законный пользователь системы;
- нарушителю известна информация о принципах работы системы;
- нарушитель выбирает наиболее слабое звено в защите.

Наиболее распространенным и многообразным видом компьютерных нарушений является ***несанкционированный доступ*** (НСД). НСД использует любую ошибку в системе защиты и возможен при нерациональном выборе средств защиты, их некорректной установке и настройке.

Проведем классификацию каналов НСД, по которым можно осуществить хищение, изменение или уничтожение информации:

- Через человека:
  - хищение носителей информации;
  - чтение информации с экрана или клавиатуры;
  - чтение информации из распечатки.
- Через программу:
  - перехват паролей;
  - дешифровка зашифрованной информации;
  - копирование информации с носителя.
- Через аппаратуру:

- подключение специально разработанных аппаратных средств, обеспечивающих доступ к информации;
- перехват побочных электромагнитных излучений от аппаратуры, линий связи, сетей электропитания и т.д.

Особо следует остановиться на угрозах, которым могут подвергаться компьютерные сети. Основная особенность любой компьютерной сети состоит в том, что ее компоненты распределены в пространстве. Связь между узлами сети осуществляется физически с помощью сетевых линий и программно с помощью механизма сообщений. При этом управляющие сообщения и данные, пересылаемые между узлами сети, передаются в виде пакетов обмена. Компьютерные сети характерны тем, что против них предпринимают так называемые ***удаленные атаки***. Нарушитель может находиться за тысячи километров от атакуемого объекта, при этом нападению может подвергаться не только конкретный компьютер, но и информация, передающаяся по сетевым каналам связи.

Обеспечение информационной безопасности

Формирование режима информационной безопасности - проблема комплексная. Меры по ее решению можно подразделить на пять уровней:

1. законодательный (законы, нормативные акты, стандарты и т.п.);
2. морально-этический (всевозможные нормы поведения, несоблюдение которых ведет к падению престижа конкретного человека или целой организации);
3. административный (действия общего характера, предпринимаемые руководством организации);
4. физический (механические, электро- и электронно-механические препятствия на возможных путях проникновения потенциальных нарушителей);
5. аппаратно-программный (электронные устройства и специальные программы защиты информации).

Единая совокупность всех этих мер, направленных на противодействие угрозам безопасности с целью сведения к минимуму возможности ущерба, образуют ***систему защиты***.

Надежная система защиты должна соответствовать следующим принципам:

- Стоимость средств защиты должна быть меньше, чем размеры возможного ущерба.
- Каждый пользователь должен иметь минимальный набор привилегий, необходимый для работы.
- Защита тем более эффективна, чем проще пользователю с ней работать.

- Возможность отключения в экстренных случаях.
- Специалисты, имеющие отношение к системе защиты должны полностью представлять себе принципы ее функционирования и в случае возникновения затруднительных ситуаций адекватно на них реагировать.
- Под защитой должна находиться вся система обработки информации.
- Разработчики системы защиты, не должны быть в числе тех, кого эта система будет контролировать.
- Система защиты должна предоставлять доказательства корректности своей работы.
- Лица, занимающиеся обеспечением информационной безопасности, должны нести личную ответственность.
- Объекты защиты целесообразно разделять на группы так, чтобы нарушение защиты в одной из групп не влияло на безопасность других.
- Надежная система защиты должна быть полностью протестирована и согласована.
- Защита становится более эффективной и гибкой, если она допускает изменение своих параметров со стороны администратора.
- Система защиты должна разрабатываться, исходя из предположения, что пользователи будут совершать серьезные ошибки и, вообще, имеют наихудшие намерения.
- Наиболее важные и критические решения должны приниматься человеком.
- Существование механизмов защиты должно быть по возможности скрыто от пользователей, работа которых находится под контролем.

Аппаратно-программные средства защиты информации

Несмотря на то, что современные ОС для персональных компьютеров, такие, как Windows 2000, Windows XP и Windows NT, имеют собственные подсистемы защиты, актуальность создания дополнительных средств защиты сохраняется. Дело в том, что большинство систем не способны защитить данные, находящиеся за их пределами, например при сетевом информационном обмене.

Аппаратно-программные средства защиты информации можно разбить на пять групп:

1. Системы идентификации (распознавания) и аутентификации (проверки подлинности) пользователей.
2. Системы шифрования дисковых данных.
3. Системы шифрования данных, передаваемых по сетям.

4. Системы аутентификации электронных данных.
5. Средства управления криптографическими ключами.

1. Системы идентификации и аутентификации пользователей

Применяются для ограничения доступа случайных и незаконных пользователей к ресурсам компьютерной системы. Общий алгоритм работы таких систем заключается в том, чтобы получить от пользователя информацию, удостоверяющую его личность, проверить ее подлинность и затем предоставить (или не предоставить) этому пользователю возможность работы с системой.

При построении этих систем возникает проблема выбора информации, на основе которой осуществляются процедуры идентификации и аутентификации пользователя. Можно выделить следующие типы:

- секретная информация, которой обладает пользователь (пароль, секретный ключ, персональный идентификатор и т.п.); пользователь должен запомнить эту информацию или же для нее могут быть применены специальные средства хранения;
- физиологические параметры человека (отпечатки пальцев, рисунок радужной оболочки глаза и т.п.) или особенности поведения (особенности работы на клавиатуре и т.п.).

Системы, основанные на первом типе информации, считаются ***традиционными***. Системы, использующие второй тип информации, называют ***биометрическими***. Следует отметить наметившуюся тенденцию опережающего развития биометрических систем идентификации.

2. Системы шифрования дисковых данных

Чтобы сделать информацию бесполезной для противника, используется совокупность методов преобразования данных, называемая ***криптографией*** [от греч. ***kryptos*** - скрытый и ***grapho*** - пишу].

Системы шифрования могут осуществлять криптографические преобразования данных на уровне файлов или на уровне дисков. К программам первого типа можно отнести архиваторы типа ARJ и RAR, которые позволяют использовать криптографические методы для защиты архивных файлов. Примером систем второго типа может служить программа шифрования Diskreet, входящая в состав популярного программного пакета Norton Utilities, Best Crypt.

Другим классификационным признаком систем шифрования дисковых данных является способ их функционирования. По способу функционирования системы шифрования дисковых данных делят на два класса:

- системы "прозрачного" шифрования;
- системы, специально вызываемые для осуществления шифрования.

В системах прозрачного шифрования (шифрования "на лету") криптографические преобразования осуществляются в режиме реального времени, незаметно для пользователя. Например, пользователь записывает подготовленный в текстовом редакторе

документ на защищаемый диск, а система защиты в процессе записи выполняет его шифрование.

Системы второго класса обычно представляют собой утилиты, которые необходимо специально вызывать для выполнения шифрования. К ним относятся, например, архиваторы со встроенными средствами парольной защиты.

Большинство систем, предлагающих установить пароль на документ, не шифрует информацию, а только обеспечивает запрос пароля при доступе к документу. К таким системам относится MS Office, 1С и многие другие.

3. Системы шифрования данных, передаваемых по сетям

Различают два основных способа шифрования: канальное шифрование и оконечное (абонентское) шифрование.

В случае ***канального шифрования*** защищается вся информация, передаваемая по каналу связи, включая служебную. Этот способ шифрования обладает следующим достоинством - встраивание процедур шифрования на канальный уровень позволяет использовать аппаратные средства, что способствует повышению производительности системы. Однако у данного подхода имеются и существенные недостатки:

- шифрование служебных данных осложняет механизм маршрутизации сетевых пакетов и требует расшифрования данных в устройствах промежуточной коммуникации (шлюзах, ретрансляторах и т.п.);
- шифрование служебной информации может привести к появлению статистических закономерностей в шифрованных данных, что влияет на надежность защиты и накладывает ограничения на использование криптографических алгоритмов.

Оконечное (абонентское) шифрование позволяет обеспечить конфиденциальность данных, передаваемых между двумя абонентами. В этом случае защищается только содержание сообщений, вся служебная информация остается открытой. Недостатком является возможность анализировать информацию о структуре обмена сообщениями, например об отправителе и получателе, о времени и условиях передачи данных, а также об объеме передаваемых данных.

4. Системы аутентификации электронных данных

При обмене данными по сетям возникает проблема аутентификации автора документа и самого документа, т.е. установление подлинности автора и проверка отсутствия изменений в полученном документе. Для аутентификации данных применяют код аутентификации сообщения (имитовставку) или электронную подпись.

Имитовставка вырабатывается из открытых данных посредством специального преобразования шифрования с использованием секретного ключа и передается по каналу связи в конце зашифрованных данных. Имитовставка проверяется получателем, владеющим секретным ключом, путем повторения процедуры, выполненной ранее отправителем, над полученными открытыми данными.

Электронная цифровая подпись представляет собой относительно небольшое количество дополнительной аутентифицирующей информации, передаваемой вместе с подписываемым текстом. Отправитель формирует цифровую подпись, используя секретный ключ отправителя. Получатель проверяет подпись, используя открытый ключ отправителя.

Таким образом, для реализации имитовставки используются принципы симметричного шифрования, а для реализации электронной подписи - асимметричного. Подробнее эти две системы шифрования будем изучать позже.

5. Средства управления криптографическими ключами

Безопасность любой криптосистемы определяется используемыми криптографическими ключами. В случае ненадежного управления ключами злоумышленник может завладеть ключевой информацией и получить полный доступ ко всей информации в системе или сети.

Различают следующие виды функций управления ключами: генерация, хранение, и распределение ключей.

Способы ***генерации ключей*** для симметричных и асимметричных криптосистем различны. Для генерации ключей симметричных криптосистем используются аппаратные и программные средства генерации случайных чисел. Генерация ключей для асимметричных криптосистем более сложна, так как ключи должны обладать определенными математическими свойствами. Подробнее на этом вопросе остановимся при изучении симметричных и асимметричных криптосистем.

Функция ***хранения*** предполагает организацию безопасного хранения, учета и удаления ключевой информации. Для обеспечения безопасного хранения ключей применяют их шифрование с помощью других ключей. Такой подход приводит к концепции иерархии ключей. В иерархию ключей обычно входит главный ключ (т.е. мастер-ключ), ключ шифрования ключей и ключ шифрования данных. Следует отметить, что генерация и хранение мастер-ключа является критическим вопросом криптозащиты.

Распределение - самый ответственный процесс в управлении ключами. Этот процесс должен гарантировать скрытность распределяемых ключей, а также быть оперативным и точным. Между пользователями сети ключи распределяют двумя способами:

- с помощью прямого обмена сеансовыми ключами;
- используя один или несколько центров распределения ключей.

Перечень документов

1. О ГОСУДАРСТВЕННОЙ ТАЙНЕ. Закон Российской Федерации от 21 июля 1993 года № 5485-1 (в ред. Федерального закона от 6 октября 1997 года № 131-ФЗ).

2. ОБ ИНФОРМАЦИИ, ИНФОРМАТИЗАЦИИ И ЗАЩИТЕ ИНФОРМАЦИИ. Федеральный закон Российской Федерации от 20 февраля 1995 года № 24-ФЗ. Принят Государственной Думой 25 января 1995 года.
3. О ПРАВОВОЙ ОХРАНЕ ПРОГРАММ ДЛЯ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И БАЗ ДАННЫХ. Закон Российской Федерации от 23 фентября 1992 года № 3524-1.
4. ОБ ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ. Федеральный закон Российской Федерации от 10 января 2002 года № 1-ФЗ.
5. ОБ АВТОРСКОМ ПРАВЕ И СМЕЖНЫХ ПРАВАХ. Закон Российской Федерации от 9 июля 1993 года № 5351-1.
6. О ФЕДЕРАЛЬНЫХ ОРГАНАХ ПРАВИТЕЛЬСТВЕННОЙ СВЯЗИ И ИНФОРМАЦИИ. Закон Российской Федерации (в ред. Указа Президента РФ от 24.12.1993 № 2288; Федерального закона от 07.11.2000 № 135-ФЗ.
7. Положение об аккредитации испытательных лабораторий и органов по сертификации средств защиты информации по требованиям безопасности информации / Государственная техническая комиссия при Президенте Российской Федерации.
8. Инструкция о порядке маркирования сертификатов соответствия, их копий и сертификационных средств защиты информации / Государственная техническая комиссия при Президенте Российской Федерации.
9. Положение по аттестации объектов информатизации по требованиям безопасности информации / Государственная техническая комиссия при Президенте Российской Федерации.
10. Положение о сертификации средств защиты информации по требованиям безопасности информации: с дополнениями в соответствии с Постановлением Правительства Российской Федерации от 26 июня 1995 года № 608 "О сертификации средств защиты информации" / Государственная техническая комиссия при Президенте Российской Федерации.
11. Положение о государственном лицензировании деятельности в области защиты информации / Государственная техническая комиссия при Президенте Российской Федерации.
12. Автоматизированные системы. Защита от несанкционированного доступа к информации. Классификация автоматизированных систем и требования по защите информации: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.
13. Концепция защиты средств вычислительной техники и автоматизированных систем от несанкционированного доступа к информации: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.
14. Средства вычислительной техники. Межсетевые экраны. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.
15. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.
16. Защита информации. Специальные защитные знаки. Классификация и общие требования: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.

17. Защита от несанкционированного доступа к информации. Термины и определения: Руководящий документ / Государственная техническая комиссия при Президенте Российской Федерации.

Критерии оценки безопасности компьютерных систем."Оранжевая книга" США

С 1983 по 1988 год Министерство обороны США и Национальный комитет компьютерной безопасности разработали систему стандартов в области компьютерной безопасности, которая включает более десяти документов. Этот список возглавляют "Критерии оценки безопасности компьютерных систем", которые по цвету обложки чаще называют "Оранжевой книгой". В 1995 году Национальный центр компьютерной безопасности США опубликовал "Пояснения к критериям безопасности компьютерных систем", объединившие все имеющиеся на тот момент дополнения и разъяснения к "Оранжевой книге".

В "Оранжевой книге" ***надежная система*** определяется как "система, использующая достаточные аппаратные и программные средства, чтобы обеспечить одновременную обработку информации разной степени секретности группой пользователей без нарушения прав доступа".

Надежность систем оценивается по двум основным критериям:

- ***Политика безопасности*** - набор законов, правил и норм поведения, определяющих, как организация обрабатывает, защищает и распространяет информацию. В частности, правила определяют, в каких случаях пользователь имеет право оперировать с определенными наборами данных. Чем надежнее система, тем строже и многообразнее должна быть политика безопасности. В зависимости от сформулированной политики можно выбирать конкретные механизмы, обеспечивающие безопасность системы. Политика безопасности - это активный компонент защиты, включающий в себя анализ возможных угроз и выбор мер противодействия.
- ***Гарантированность*** - мера доверия, которая может быть оказана архитектуре и реализации системы. Гарантированность можно определить тестированием системы в целом и ее компонентов. Гарантированность показывает, насколько корректны механизмы, отвечающие за проведение в жизнь политики безопасности. Гарантированность можно считать пассивным компонентом защиты, надзирающим за самими защитниками.

Важным средством обеспечения безопасности является механизм ***подотчетности*** (протоколирования). Надежная система должна фиксировать все события, касающиеся безопасности. Ведение протоколов должно дополняться ***аудитом***, то есть анализом регистрационной информации.

При оценке степени гарантированности, с которой систему можно считать надежной, центральной является концепция надежной вычислительной базы. ***Вычислительная база*** - это совокупность защитных механизмов компьютерной системы (включая аппаратное и программное обеспечение), отвечающих за проведение в жизнь политики безопасности. Надежность вычислительной базы определяется исключительно ее реализацией и корректностью исходных данных, которые вводит административный персонал (например, это могут быть данные о степени благонадежности пользователей).

Основное назначение надежной вычислительной базы - выполнять функции ***монитора обращений***, то есть контролировать допустимость выполнения субъектами определенных операций над объектами. Каждое обращение пользователя к программам или данным проверяется на предмет согласованности со списком действий, допустимых для пользователя.

От монитора обращений требуется выполнение трех свойств:

- ***Изолированность***. Монитор должен быть защищен от отслеживания своей работы;
- ***Полнота***. Монитор должен вызываться при каждом обращении, не должно быть способов его обхода;
- ***Верифицируемость***. Монитор должен быть компактным, чтобы его можно было проанализировать и протестировать, будучи уверенным в полноте тестирования.

Основные элементы политики безопасности

Согласно "Оранжевой книге", политика безопасности должна включать в себя по крайней мере следующие элементы:

- произвольное управление доступом;
- безопасность повторного использования объектов;
- метки безопасности;
- принудительное управление доступом.

Рассмотрим перечисленные элементы подробнее.

Произвольное управление доступом

Произвольное управление доступом - это метод ограничения доступа к объектам, основанный на учете личности субъекта или группы, в которую субъект входит. Произвольность управления состоит в том, что некоторое лицо (обычно владелец объекта) может по своему усмотрению давать другим субъектам или отбирать у них права доступа к объекту.

Текущее состояние прав доступа при произвольном управлении описывается матрицей, в строках которой перечислены субъекты, а в столбцах - объекты. В клетках, расположенных на пересечении строк и столбцов, записываются способы доступа, допустимые для субъекта по отношению к объекту, например: чтение, запись, выполнение, возможность передачи прав другим субъектам и т.п.

Очевидно, прямолинейное представление подобной матрицы невозможно (поскольку она очень велика), да и не нужно (поскольку она разрежена, то есть большинство клеток в ней пусты). В операционных системах более компактное представление матрицы доступа основывается или на структурировании совокупности субъектов (владелец/группа/прочие в ОС UNIX), или на механизме списков управления доступом, то есть на представлении матрицы по столбцам, когда для каждого объекта перечисляются субъекты вместе с их правами доступа. За счет использования метасимволов можно компактно описывать группы субъектов, удерживая тем самым размеры списков управления доступом в разумных рамках.

Большинство операционных систем и систем управления базами данных реализуют именно произвольное управление доступом. Главное его достоинство - гибкость, главные недостатки - рассредоточенность управления и сложность централизованного контроля, а также оторванность прав доступа от данных, что позволяет копировать секретную информацию в общедоступные файлы.

Безопасность повторного использования объектов

Безопасность повторного использования объектов - важное на практике дополнение средств управления доступом, предохраняющее от случайного или преднамеренного извлечения секретной информации из "мусора". Безопасность повторного использования должна гарантироваться для областей оперативной памяти, в частности для буферов с образами экрана, расшифрованными паролями и т.п., для дисковых блоков и магнитных носителей в целом.

Важно обратить внимание на следующий момент. Поскольку информация о субъектах также представляет собой объект, необходимо позаботиться о безопасности "повторного использования субъектов". Когда пользователь покидает организацию, следует не только лишить его возможности входа в систему, но и запретить доступ ко всем объектам. В противном случае новый сотрудник может получить ранее использовавшийся идентификатор, а с ним и все права своего предшественника.

Современные интеллектуальные периферийные устройства усложняют обеспечение безопасности повторного использования объектов. Действительно, принтер может буферизовать несколько страниц документа, которые останутся в памяти даже после окончания печати. Необходимо предпринять специальные меры, чтобы "вытолкнуть" их оттуда.

Впрочем, иногда организации защищаются от повторного использования слишком ревностно - путем уничтожения магнитных носителей. На практике заведомо достаточно троекратной записи случайных последовательностей бит.

Метки безопасности

Для реализации принудительного управления доступом с субъектами и объектами используются ***метки безопасности***. Метка субъекта описывает его благонадежность, метка объекта - степень закрытости содержащейся в нем информации.

Согласно "Оранжевой книге", метки безопасности состоят из двух частей: уровня секретности и списка категорий. Уровни секретности, поддерживаемые системой, образуют упорядоченное множество, которое может выглядеть, например, так:

- совершенно секретно;
- секретно;
- конфиденциально;
- несекретно.

Категории образуют неупорядоченный набор. Их назначение - описать предметную область, к которой относятся данные. В военной области каждая категория может соответствовать, например, определенному виду вооружений. Механизм категорий позволяет разделить информацию "по отсекам", что способствует лучшей защищенности. Субъект не может получить доступ к "чужим" категориям, даже если его уровень

благонадежности - "совершенно секретно". Специалист по танкам не узнает тактико-технические данные самолетов.

Главная проблема, которую необходимо решать в связи с метками, - это обеспечение их ***целостности***. Во-первых, не должно быть непомеченных субъектов и объектов, иначе в меточной безопасности появятся легко используемые бреши. Во-вторых, при любых операциях с данными метки должны оставаться правильными. В особенности это относится к экспорту и импорту данных. Например, печатный документ должен открываться заголовком, содержащим текстовое и/или графическое представление метки безопасности. Аналогично, при передаче файла по каналу связи должна передаваться и ассоциированная с ним метка, причем в таком виде, чтобы удаленная система могла ее протрактовать, несмотря на возможные различия в уровнях секретности и наборе категорий.

Метки безопасности субъектов более подвижны, чем метки объектов. Субъект может в течение сеанса работы с системой изменять свою метку, естественно, не выходя за предопределенные для него рамки. Иными словами, он может сознательно занижать свой уровень благонадежности, чтобы уменьшить вероятность непреднамеренной ошибки. Вообще, принцип минимизации привилегий - весьма разумное средство защиты.

Принудительное управление доступом

Принудительное управление доступом основано на сопоставлении меток безопасности субъекта и объекта.

Субъект может читать информацию из объекта, если уровень секретности субъекта не ниже, чем у объекта, а все категории, перечисленные в метке безопасности объекта, присутствуют в метке субъекта. В таком случае говорят, что метка субъекта доминирует над меткой объекта. Смысл сформулированного правила понятен - читать можно только то, что положено.

Субъект может записывать информацию в объект, если метка безопасности объекта доминирует над меткой субъекта. В частности, "конфиденциальный" субъект может писать в секретные файлы, но не может - в несекретные (разумеется, должны также выполняться ограничения на набор категорий). На первый взгляд, подобное ограничение может показаться странным, однако оно вполне разумно. Ни при каких операциях уровень секретности информации не должен понижаться, хотя обратный процесс вполне возможен. Посторонний человек может случайно узнать секретные сведения и сообщить их куда следует, однако лицо, допущенное к работе с секретными документами, не имеет права раскрывать их содержание простому смертному.

Описанный способ управления доступом называется принудительным, поскольку он не зависит от воли субъектов (даже системных администраторов). После того как зафиксированы метки безопасности субъектов и объектов, оказываются зафиксированными и права доступа. В терминах принудительного управления нельзя выразить предложение "разрешить доступ к объекту X еще и для пользователя Y". Конечно, можно изменить метку безопасности пользователя Y, но тогда он, скорее всего, получит доступ ко многим дополнительным объектам, а не только к X.

Принудительное управление доступом реализовано во многих вариантах операционных систем и СУБД, отличающихся повышенными мерами безопасности. В частности, такие варианты существуют для SunOS и СУБД Ingres. Независимо от

практического использования принципы принудительного управления являются удобным методологическим базисом для начальной классификации информации и распределения прав доступа. Удобнее мыслить в терминах уровней секретности и категорий, чем заполнять неструктурированную матрицу доступа.

Классы безопасности

"Критерии оценки безопасности компьютерных систем" Министерства обороны США открыли путь к ранжированию информационных систем по степени надежности. В "Оранжевой книге" определяется четыре уровня надежности (безопасности) - D, C, B и A. Уровень D предназначен для систем, признанных неудовлетворительными. В настоящее время он пуст, и ситуация едва ли когда-нибудь изменится. По мере перехода от уровня C к A к надежности систем предъявляются все более жесткие требования. Уровни C и B подразделяются на классы (C1, C2, B1, B2, B3) с постепенным возрастанием надежности. Таким образом, всего имеется шесть классов безопасности - C1, C2, B1, B2, B3, A1. Чтобы система в результате процедуры сертификации могла быть отнесена к некоторому классу, ее политика безопасности и гарантированность должны удовлетворять приводимым ниже требованиям. Поскольку при переходе к каждому следующему классу требования только добавляются, будем говорить лишь о том новом, что присуще данному классу, группируя требования в согласии с предшествующим изложением.

Итак, ниже следуют критерии оценки надежных компьютерных систем.

Требования к политике безопасности

Требования к политике безопасности, проводимой системой, подразделяются в соответствии с основными направлениями политики, предусматриваемыми "Оранжевой книгой".

Произвольное управление доступом:

Класс C1 - вычислительная база должна управлять доступом именованных пользователей к именованным объектам. Механизм управления (права для владельца/группы/прочих, списки управления доступом) должен позволять специфицировать разделение файлов между индивидами и/или группами.

Класс C2 - в дополнение к C1, права доступа должны гранулироваться с точностью до пользователя. Механизм управления должен ограничивать распространение прав доступа - только авторизованный пользователь, например владелец объекта, может предоставлять права доступа другим пользователям. Все объекты должны подвергаться контролю доступа.

Класс B3 - в дополнение к C2, обязательно должны использоваться списки управления доступом с указанием разрешенных режимов. Должна быть возможность явного указания пользователей или их групп, доступ которых к объекту запрещен.

*(**Примечание.** Поскольку классы B1 и B2 не упоминаются, требования к ним в плане добровольного управления доступом те же, что и для C2. Аналогично, требования к классу A1 те же, что и для B3.)*

Повторное использование объектов:

Класс C2 - при выделении хранимого объекта из пула ресурсов вычислительной базы необходимо ликвидировать все следы предыдущих использований.

Метки безопасности:

Класс B1 - вычислительная база должна управлять метками безопасности, связанными с каждым субъектом и хранимым объектом. Метки являются основой функционирования механизма принудительного управления доступом. При импорте непомеченной информации соответствующий уровень секретности должен запрашиваться у авторизованного пользователя и все такие действия следует протоколировать.

Класс B2 - в дополнение к B1, помечаться должны все ресурсы системы, например ПЗУ, прямо или косвенно доступные субъектам.

Целостность меток безопасности:

Класс B1 - метки должны адекватно отражать уровни секретности субъектов и объектов. При экспорте информации метки должны преобразовываться в точное и однозначно трактуемое внешнее представление, сопровождающее данные. Каждое устройство ввода/вывода (в том числе коммуникационный канал) должно трактоваться как одноуровневое или многоуровневое. Все изменения трактовки и ассоциированных уровней секретности должны протоколироваться.

Класс B2 - в дополнение к B1, вычислительная база должна немедленно извещать терминального пользователя об изменении его метки безопасности. Пользователь может запросить информацию о своей метке. База должна поддерживать присваивание всем подключенным физическим устройствам минимального и максимального уровня секретности. Эти уровни должны использоваться при проведении в жизнь ограничений, налагаемых физической конфигурацией системы, например расположением устройств.

Принудительное управление доступом:

Класс B1 - вычислительная база должна обеспечить проведение в жизнь принудительного управления доступом всех субъектов ко всем хранимым объектам. Субъектам и объектам должны быть присвоены метки безопасности, являющиеся комбинацией упорядоченных уровней секретности, а также категорий. Метки являются основой принудительного управления доступом. Надежная вычислительная база должна поддерживать по крайней мере два уровня секретности.

Вычислительная база должна контролировать идентификационную и аутентификационную информацию. При создании новых субъектов, например процессов, их метки безопасности не должны доминировать над меткой породившего их пользователя.

Класс B2 - в дополнение к B1, все ресурсы системы (в том числе ПЗУ, устройства ввода/вывода) должны иметь метки безопасности и служить объектами принудительного управления доступом.

Требования к подотчетности

Идентификация и аутентификация:

Класс C1 - пользователи должны идентифицировать себя, прежде чем выполнять какие-либо иные действия, контролируемые вычислительной базой. Для аутентификации должен использоваться какой-либо защитный механизм, например пароли. Аутентификационная информация должна быть защищена от несанкционированного доступа.

Класс C2 - в дополнение к C1, каждый пользователь системы должен уникальным образом идентифицироваться. Каждое регистрируемое действие должно связываться с конкретным пользователем.

Класс B1 - в дополнение к C2, вычислительная база должна поддерживать метки безопасности пользователей.

Предоставление надежного пути:

Класс B2 - вычислительная база должна поддерживать надежный коммуникационный путь к себе для пользователя, выполняющего операции начальной идентификации и аутентификации. Инициатива в общении по этому пути должна исходить исключительно от пользователя.

Класс B3 - в дополнение к B2, коммуникационный путь может формироваться по запросу, исходящему как от пользователя, так и от самой базы. Надежный путь может использоваться для начальной идентификации и аутентификации, для изменения текущей метки безопасности пользователя и т.п. Общение по надежному пути должно быть логически отделено и изолировано от других информационных потоков.

Аудит:

Класс C2 - вычислительная база должна создавать, поддерживать и защищать журнал регистрационной информации, относящейся к доступу к объектам, контролируемым базой. Должна быть возможность регистрации следующих событий:

- использование механизма идентификации и аутентификации;
- внесение объектов в адресное пространство пользователя, например открытие файла, запуск программы;
- удаление объектов;
- действия системных операторов, системных администраторов, администраторов безопасности;
- другие события, затрагивающие информационную безопасность.

Каждая регистрационная запись должна включать следующие поля:

- дата и время события;
- идентификатор пользователя;
- тип события;
- результат действия (успех или неудача).

Для событий идентификации/аутентификации регистрируется также идентификатор устройства, например терминала. Для действий с объектами регистрируются имена объектов.

Системный администратор может выбирать набор регистрируемых событий для каждого пользователя.

Класс B1 - в дополнение к C2, должны регистрироваться операции выдачи на печать и ассоциированные внешние представления меток безопасности. При операциях с объектами, помимо имен, регистрируются их метки безопасности. Набор регистрируемых событий может различаться в зависимости от уровня секретности объектов.

Класс B2 - в дополнение к B1, должна быть возможность регистрировать события, связанные с организацией тайных каналов с памятью.

Класс B3 - в дополнение к B2, должна быть возможность регистрации появления или накопления событий, несущих угрозу политике безопасности системы. Администратор безопасности должен немедленно извещаться о попытках нарушения политики безопасности, а система, в случае продолжения попыток, должна пресекать их наименее болезненным способом.

Требования к гарантированности

Архитектура системы:

Класс C1 - вычислительная база должна поддерживать область для собственного выполнения, защищенную от внешних воздействий, в частности от изменения команд и/или данных, и от попыток слежения за ходом работы. Ресурсы, контролируемые базой, могут составлять определенное подмножество всех субъектов и объектов системы.

Класс C2 - в дополнение к C1, вычислительная база должна изолировать защищаемые ресурсы в той мере, как это диктуется требованиями контроля доступа и подотчетности.

Класс B1 - в дополнение к C2, вычислительная база должна обеспечивать взаимную изоляцию процессов путем разделения их адресных пространств.

Класс B2 - в дополнение к B1, вычислительная база должна быть внутренне структурирована на хорошо определенные, относительно независимые модули. Вычислительная база должна эффективно использовать имеющееся оборудование для отделения элементов, критически важных с точки зрения защиты, от прочих компонентов системы. Модули базы должны проектироваться с учетом принципа минимизации привилегий. Для защиты логически раздельных хранимых объектов должны использоваться аппаратные средства, например сегментация. Должен быть полностью определен пользовательский интерфейс с вычислительной базой.

Класс B3 - в дополнение к B2, вычислительная база должна быть спроектирована и структурирована таким образом, чтобы использовать полный и концептуально простой защитный механизм. Этот механизм должен играть центральную роль во внутренней структуризации вычислительной базы и всей системы. База должна активно использовать разделение данных по уровням. Значительные инженерные усилия должны быть

направлены на уменьшение сложности вычислительной базы и на вынесение из нее модулей, не являющихся критически важными с точки зрения защиты.

Целостность системы:

Класс C1 - должны быть в наличии аппаратные и/или программные средства, позволяющие периодически проверять корректность функционирования аппаратных и микропрограммных компонентов вычислительной базы.

Анализ тайных каналов передачи информации:

Класс B2 - системный архитектор должен тщательно проанализировать возможности по организации тайных каналов с памятью и оценить максимальную пропускную способность каждого выявленного канала.

Класс B3 - в дополнение к B2, аналогичная процедура должна быть проделана для временных каналов.

Класс A1 - в дополнение к B3, для анализа должны использоваться формальные методы.

Надежное администрирование:

Класс B2 - система должна поддерживать разделение функций оператора и администратора.

Класс B3 - в дополнение к B2, должна быть специфицирована роль администратора безопасности. Получить права администратора безопасности можно только после выполнения явных, протоколируемых действий. Не относящиеся к защите действия администратора безопасности должны быть по возможности ограничены.

Надежное восстановление:

Класс B3 - должны существовать процедуры и/или механизмы, позволяющие произвести восстановление после сбоя или иного нарушения работы без ослабления защиты.

Тестирование:

Класс C1 - защитные механизмы должны быть протестированы на предмет соответствия их поведения системной документации. Тестирование должно подтвердить, что у неавторизованного пользователя нет очевидных способов обойти или разрушить средства защиты вычислительной базы.

Класс C2 - в дополнение к C1, тестирование должно подтвердить отсутствие очевидных недостатков в механизмах изоляции ресурсов и защиты регистрационной информации.

Класс B1 - в дополнение к C2, группа специалистов, полностью понимающих конкретную реализацию вычислительной базы, должна подвергнуть описание архитектуры, исходные и объектные коды тщательному анализу и тестированию. Цель

должна состоять в выявлении всех дефектов архитектуры и реализации, позволяющих субъекту без должной авторизации читать, изменять, удалять информацию или приводить базу в состояние, когда она перестает обслуживать запросы других субъектов. Все выявленные недостатки должны быть исправлены или нейтрализованы, после чего база подвергается повторному тестированию, чтобы убедиться в отсутствии прежних или приобретении новых недостатков.

Класс B2 - в дополнение к B1, должна быть продемонстрирована относительная устойчивость вычислительной базы к попыткам проникновения.

Класс B3 - в дополнение к B2, должна быть продемонстрирована устойчивость вычислительной базы к попыткам проникновения.

Класс A1 - в дополнение к B3, тестирование должно продемонстрировать, что реализация вычислительной базы соответствует формальным спецификациям верхнего уровня.

Основу тестирования средств защиты от проникновения в систему должно составлять наличие спецификаций на исходные тексты.

Верификация спецификаций архитектуры:

Класс B1 - должна существовать неформальная или формальная модель политики безопасности, поддерживаемой вычислительной базой. Модель должна соответствовать основным посылкам политики безопасности на протяжении всего жизненного цикла системы.

Класс B2 - в дополнение к B1, модель политики безопасности должна быть формальной. Для вычислительной базы должны существовать описательные спецификации верхнего уровня, точно и полно определяющие ее интерфейс.

Класс B3 - в дополнение к B2, должны быть приведены убедительные аргументы соответствия между спецификациями и моделью.

Класс A1 - в дополнение к B3, помимо описательных должны быть представлены формальные спецификации верхнего уровня, относящиеся к аппаратным и/или микропрограммным элементам, составляющим интерфейс вычислительной базы. Комбинация формальных и неформальных методов должна подтвердить соответствие между спецификациями и моделью. Должны использоваться современные методы формальной спецификации и верификации систем, доступные Национальному центру компьютерной безопасности США.

Конфигурационное управление:

Класс B2 - в процессе разработки и сопровождения вычислительной базы должна использоваться система конфигурационного управления, обеспечивающая контроль за изменениями в описательных спецификациях верхнего уровня, иных архитектурных данных, реализационной документации, исходных текстах, работающей версии объектного кода, тестовых данных и документации. Конфигурационное управление должно обеспечивать соответствие друг другу всех аспектов текущей версии вычислительной базы. Должны предоставляться средства генерации новых версий базы по

исходным текстам и средства для сравнения версий, чтобы убедиться в том, что произведены только запланированные изменения.

Класс A1 - в дополнение к B2, механизм конфигурационного управления должен распространяться на весь жизненный цикл и все компоненты системы, имеющие отношение к обеспечению безопасности, включая спецификации и документацию. Для защиты эталонной копии материалов, использующихся для генерации надежной вычислительной базы, должна использоваться комбинация физических, административных и технических мер.

Надежное распространение:

Класс A1 - должна поддерживаться целостность соответствия между эталонными данными, описывающими текущую версию вычислительной базы, и эталонной копией текстов этой версии. Должны существовать процедуры, подтверждающие соответствие между поставляемыми клиентам аппаратными и программными компонентами и эталонной копией.

Требования к документации

Руководство пользователя по средствам безопасности:

Класс C1 - отдельный фрагмент документации (глава, том) должен описывать защитные механизмы, предоставляемые вычислительной базой, и их взаимодействие между собой, содержать рекомендации по их использованию.

Руководство администратора по средствам безопасности:

Класс C1 - руководство должно содержать сведения о функциях и привилегиях, которыми управляет системный администратор посредством механизмов безопасности.

Класс C2 - в дополнение к C1, должны описываться процедуры обработки регистрационной информации и управления файлами с такой информацией, а также структура записей для каждого типа регистрируемых событий.

Класс B1 - в дополнение к C2, руководство должно описывать функции оператора и администратора, затрагивающие безопасность, в том числе действия по изменению характеристик пользователей. Должны быть представлены рекомендации по согласованному и эффективному использованию средств безопасности, их взаимодействию друг с другом, по безопасной генерации новых версий вычислительной базы.

Класс B2 - в дополнение к B1, должны быть указаны модули вычислительной базы, содержащие механизмы проверки обращений. Должна быть описана процедура безопасной генерации новой версии базы после внесения изменений в исходные тексты.

Класс B3 - в дополнение к B2, должна быть описана процедура, обеспечивающая безопасность начального запуска системы и возобновления ее работы после сбоя.

Тестовая документация:

Класс C1 - разработчик системы должен представить экспертному совету документ, содержащий план тестов, процедуры прогона тестов и результаты тестов.

Класс B2 - в дополнение к C1, тесты должны подтверждать действенность мер по уменьшению пропускной способности тайных каналов передачи информации.

Класс A1 - в дополнение к B2, должно быть описано соответствие между формальными спецификациями верхнего уровня и исходными текстами.

Описание архитектуры:

Класс C1 - должны быть описаны подход к безопасности, используемый производителем, и применение этого подхода при реализации вычислительной базы. Если база состоит из нескольких модулей, должен быть описан интерфейс между ними.

Класс B1 - в дополнение к C1, должно быть представлено неформальное или формальное описание модели политики безопасности, проводимой в жизнь вычислительной базой. Необходимо наличие аргументов в пользу достаточности избранной модели для реализации политики безопасности. Должны быть описаны защитные механизмы базы и их место в модели.

Класс B2 - в дополнение к B1, модель политики безопасности должна быть формальной и доказательной. Должно быть показано, что описательные спецификации верхнего уровня точно отражают интерфейс вычислительной базы. Должно быть показано, как база реализует концепцию монитора обращений, почему она устойчива к попыткам отслеживания ее работы, почему ее нельзя обойти и почему она реализована корректно. Должна быть описана структура базы, чтобы облегчить ее тестирование и проверку соблюдения принципа минимизации привилегий. Документация должна содержать результаты анализа тайных каналов передачи информации и описание мер протоколирования, помогающих выявлять каналы с памятью.

Класс B3 - в дополнение к B2, должно быть неформально продемонстрировано соответствие между описательными спецификациями верхнего уровня и реализацией вычислительной базы.

Класс A1 - в дополнение к B3, должно быть неформально продемонстрировано соответствие между формальными спецификациями верхнего уровня и реализацией вычислительной базы.

Таковы, согласно "Оранжевой книге", требования к классам безопасности информационных систем.

Криптографические средства защиты (шифрование) информации

Простые криптосистемы

Криптографические методы являются наиболее эффективными средствами защиты информации в автоматизированных системах (АС). А при передаче информации по протяженным линиям связи они являются единственным реальным средством предотвращения несанкционированного доступа.

Любой криптографический метод характеризуется такими показателями, как ***стойкость*** и ***трудоемкость***:

- ***Стойкость метода*** - это тот минимальный объем зашифрованного текста, статистическим анализом которого можно вскрыть исходный текст. Таким образом стойкость шифра определяет допустимый объем информации, зашифровываемый при использовании одного ключа.
- ***Трудоемкость метода*** - определяется числом элементарных операций, необходимых для шифрования одного символа исходного текста.

Основные требования к криптографическому закрытию информации в АС

1. Сложность и стойкость криптографического закрытия данных должны выбираться в зависимости от объема и степени секретности данных.
2. Надежность закрытия должна быть такой, чтобы секретность не нарушалась даже в том случае, когда злоумышленнику становится известен метод шифрования.
3. Метод закрытия, набор используемых ключей и механизм их распределения не должны быть слишком сложными.
4. Выполнение процедур прямого и обратного преобразований должно быть формальным. Эти процедуры не должны зависеть от длины сообщений.
5. Ошибки, возникающие в процессе преобразования не должны распространяться по системе.
6. Вносимая процедурами защиты избыточность должна быть минимальной.

Классификация основных методов криптографического закрытия информации

I. Шифрование
   1. Подстановка (замена)
      a. Одноалфавитная
      b. Многоалфавитная одноконтурная обыкновенная
      c. Многоалфавитная одноконтурная монофоническая
      d. Многоалфавитная многоконтурная

2. Перестановка
   a. Простая
   b. Усложненная по таблице
   c. Усложненная по маршрутам

3. Гаммирование
   a. С конечной короткой гаммой
   b. С конечной длинной гаммой
   c. С бесконечной гаммой
4. Аналитические преобразования
   a. Матричные
   b. По особым зависимостям
5. Комбинированные
   a. Подстановка+перестановка
   b. Подстановка+гаммирование
   c. Перестановка+гаммирование
   d. Гаммирование+гаммирование

II. Кодирование
   1. Смысловое
      a. По специальным таблицам
   2. Символьное
      a. По кодовому алфавиту

III. Другие виды
   1. Рассечение-разнесение
      a. Смысловое
      b. Механическое
   2. Сжатие-расширение

Начиная разговор о шифровании, определимся с терминологией на примере фильма "Семнадцать мгновений весны": Юстас - зашифровывает, Алекс - расшифровывает, а старина Мюллер - дешифрует сообщение.

Шифрование методом замены (подстановки)

Наиболее простой метод шифрования. Символы шифруемого текста заменяются другими символами, взятыми из одного алфавита (одноалфавитная замена) или нескольких алфавитов (многоалфавитная подстановка).

Одноалфавитная подстановка

Простейшая подстановка - прямая замена символов шифруемого сообщения другими буквами того же самого или другого алфавита.

Примеры таблиц замены:

```
А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я
М Л Д О Т В А Ч К Е Ж Х Щ Ф Ц Э Г Б Я Ъ Ш Ы З И Ь Н Ю У П С Р Й

А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я
Q W E R T Y U I O P [ ] A S D F G H J K L Z X C V B N M < > @ %
```

Стойкость метода простой замены низкая. Зашифрованный текст имеет те же самые статистические характеристики, что и исходный, поэтому зная стандартные частоты появления символов в том языке, на котором написано сообщение, и подбирая по частотам появления символы в зашифрованном сообщении, можно восстановить таблицу замены. Для этого требуется лишь достаточно длинный зашифрованный текст, для того, чтобы получить достоверные оценки частот появления символов. Поэтому простую замену используют лишь в том случае, когда шифруемое сообщение достаточно коротко!

Стойкость метода равна 20 - 30, трудоемкость определяется поиском символа в таблице замены. Для снижения трудоемкости при шифровании таблица замены сортируется по шифруемым символам, а для расшифровки формируется таблица дешифрования, которая получается из таблицы замены сортировкой по заменяющим символам.

Многоалфавитная замена повышает стойкость шифра.

Многоалфавитная одноконтурная обыкновенная подстановка

Для замены символов используются несколько алфавитов, причем смена алфавитов проводится последовательно и циклически: первый символ заменяется на соответствующий символ первого алфавита, второй - из второго алфавита, и т.д. пока не будут исчерпаны все алфавиты. После этого использование алфавитов повторяется.

Рассмотрим шифрование с помощью ***таблицы Вижинера*** - квадратной матрицы с n^2 элементами, где n - число символов используемого алфавита. В первой строке матрицы содержится исходный алфавит, каждая следующая строка получается из предыдущей циклическим сдвигом влево на один символ.

Таблица Вижинера для русского алфавита:

```
А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я
Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А
В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б
Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В
Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г
Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д
Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е
З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж
И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З
Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И
К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й
Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К
М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л
Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М
О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н
П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О
Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П
С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р
Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С
У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т
Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У
Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф
Ц Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х
Ч Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц
```

```
Ш Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч
Щ Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш
Ь Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ
Ы Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь
Ъ Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы
Э Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ
Ю Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э
Я А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю
```

Для шифрования необходимо задать ключ - слово с неповторяющимися символами. Таблицу замены получают следующим образом: строку "Символы шифруемого текста" формируют из первой строки матрицы Вижинера, а строки из раздела "Заменяющие символы" образуются из строк матрицы Вижинера, первые символы которых совпадают с символами ключевого слова.

При шифровании и дешифровании нет необходимости держать в памяти всю матрицу Вижинера, поскольку используя свойства циклического сдвига, можно легко вычислить любую строку матрицы по ее номеру и первой строке.

При шифровании символы из первой строки заменяются символами остальных строк по правилу

```
a(1,i) -> a(k,i),
```

где `k` - номер используемой для шифрования строки.

Используя свойства циклического сдвига влево элементы `k`-ой строки можно выразить через элементы первой строки

```
                a(1,i+k-1), если i<=n-k+1
a(k,i)=
                a(1,i-n+k-1), если i>n-k+1
```

При дешифровании производится обратная замена

```
a(k,i) -> a(1,i).
```

Поэтому необходимо решить следующую задачу: пусть очередной дешифруемый символ в тексте - `a(1,j)` и для дешифрования используется `k`-я строка матрицы Вижинера. Необходимо найти в `k`-ой строке номер элемента, равного `a(1,j)`. Очевидно,

```
                a(k,j-k+1), если j>=k
a(1,j)=
                a(k,n-k+j+1), если j<k
```

Таким образом при дешифровании по `k`-ой строке матрицы Вижинера символа из зашифрованного текста, значение которого равно `a(1,j)`, проводится обратная подстановка

```
                a(1,j-k+1), если j>=k
a(1,j) ->
                a(1,n-k+j+1), если j<k
```

Стойкость метода равна стойкости метода подстановки, умноженной на количество используемых при шифровании алфавитов, т.е. на длину ключевого слова и равна `20*L`, где `L` - длина ключевого слова.

С целью повышения стойкости шифрования предлагаются следующие усовершенствования таблицы Вижинера:

1. Во всех (кроме первой) строках таблицы буквы располагаются в произвольном порядке.
2. В качестве ключа используются случайные последовательности чисел, которые задают номера используемых строк матрицы Вижинера для шифрования.

Многоалфавитная одноконтурная монофоническая подстановка

В монофонической подстановке количество и состав алфавитов выбирается таким образом, чтобы частоты появления всех символов в зашифрованном тексте были одинаковыми. При таком положении затрудняется криптоанализ зашифрованного текста с помощью его статистической обработки. Выравнивание частот появления символов достигается за счет того, что для часто встречающихся символов исходного текста предусматривается большее число заменяющих символов, чем для редко встречающихся.

Пример таблицы монофонической замены:

```
А Б В Г Д Е Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я _
Ф Н ( Щ И Г Е R А Д Ы ~ @ S Л Я Ж ^ С Ш М Б Q П Т Х Ю Ъ Р } \ _ #
* Н У Щ D + Е R = Д Ц Й Ч [ В Ь ) О & { М Б Q П Т Х Ю Ъ Р } \ _ <
Л Н ( Щ И ] Е R % Д Ы ~ @ G / Я Э З " Ш М Б Q П Т Х Ю Ъ Р } \ _ W
Ф Н У Щ D К Е R А Д Ц Й Ч S + Ь Ж ^ С { М Б Q П Т Х Ю Ъ Р } \ _ V
```

Шифрование проводится так же, как и при простой подстановке, с той лишь разницей, что после шифрования каждого символа соответствующий ему столбец алфавитов циклически сдвигается вверх на одну позицию. Таким образом, столбцы алфавитов как бы образуют независимые друг от друга кольца, поворачиваемые вверх на один знак каждый раз после шифрования соответствующего знака исходного текста.

Многоалфавитная многоконтурная подстановка

Многоконтурная подстановка заключается в том, что для шифрования используются несколько наборов (контуров) алфавитов, используемых циклически, причем каждый контур в общем случае имеет свой индивидуальный период применения. Частным случаем многоконтурной полиалфавитной подстановки является замена по таблице Вижинера, если для шифрования используется несколько ключей, каждый из которых имеет свой период применения.

Общая модель шифрования подстановкой может быть представлена в следующем виде:

$$t_ш = t_о + w \bmod (k-1),$$

где $t_ш$ - символ зашифрованного текста,
$t_о$ - символ исходного текста,

w - целое число в диапазоне 0 – (k-1),
k - число символов используемого алфавита.

Если w фиксировано, то формула описывает одноалфавитную подстановку, если w выбирается из последовательности w_1 ,w_2 ,...,w_n, то получается многоалфавитная подстановка с периодом n.

Если в многоалфавитной подстановке n > m (где m - число знаков шифруемого текста) и любая последовательность w_i, i=1,2,...,n используется только один раз, то такой шифр является теоретически нераскрываемым. Такой шифр получил название шифра Вермэна.

Стойкость простой многоалфавитной подстановки оценивается величиной 20*n, где n - число различных алфавитов, используемых для замены. Усложнение многоалфавитной подстановки существенно повышает ее стойкость. Монофоническая подстановка может быть весьма стойкой (и даже теоретически нераскрываемой), однако строго монофоническую подстановку реализовать на практике трудно, а любые отклонения от монофоничности снижают реальную стойкость шифра.

Шифрование методом перестановки

При шифровании перестановкой символы шифруемого текста переставляются по определенным правилам внутри шифруемого блока этого текста.

Простая перестановка

Выбирается размер блока шифрования в n столбцов и m строк и ключевая последовательность, которая формируется из натурального ряда чисел 1,2,...,n случайной перестановкой.

Шифрование проводится в следующем порядке:

1. Шифруемый текст записывается последовательными строками под числами ключевой последовательности, образуя блок шифрования размером n*m.
2. Зашифрованный текст выписывается колонками в порядке возрастания номеров колонок, задаваемых ключевой последовательностью.
3. Заполняется новый блок и т.д.

Например, зашифруем текст

ГРУЗИТЕ_АПЕЛЬСИНЫ_БОЧКАХ

блоком размером 8*3 и ключом 5-8-1-3-7-4-6-2.

Таблица простой перестановки будет иметь вид:

			Ключ				
5	8	1	3	7	4	6	2
Г	Р	У	З	И	Т	Е	_

А	П	Е	Л	Ь	С	И	Н
Ы	_	Б	О	Ч	К	А	Х

Зашифрованное сообщение:

УЕБ_НХЗЛОЕСЛГАЫЕИАИЬЧРП_

Расшифрование выполняется в следующем порядке:

1. Из зашифрованного текста выделяется блок символов размером `n*m`.
2. Этот блок разбивается на `n` групп по `m` символов.
3. Символы записываются в те столбцы таблицы перестановки, номера которых совпадают с номерами групп в блоке. Расшифрованный текст читается по строкам таблицы перестановки.
4. Выделяется новый блок символов и т.д.

Перестановка, усложненная по таблице

При усложнении перестановки по таблицам для повышения стойкости шифра в таблицу перестановки вводятся неиспользуемые клетки таблицы. Количество и расположение неиспользуемых элементов является дополнительным ключом шифрования.

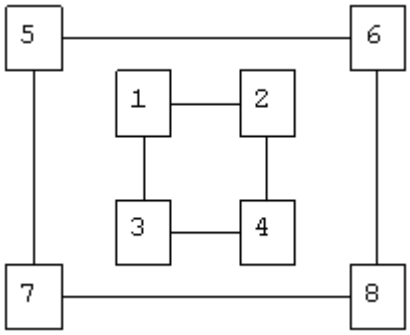
При шифровании текста в неиспользуемые элементы не заносятся символы текста и в зашифрованный текст из них не записываются никакие символы - они просто пропускаются. При расшифровке символы зашифрованного текста также не заносятся в неиспользуемые элементы.

Для дальнейшего увеличения криптостойкости шифра можно в процессе шифрования менять ключи, размеры таблицы перестановки, количество и расположение неиспользуемых элементов по некоторому алгоритму, причем этот алгоритм становится дополнительным ключом шифра.

Перестановка, усложненная по маршрутам

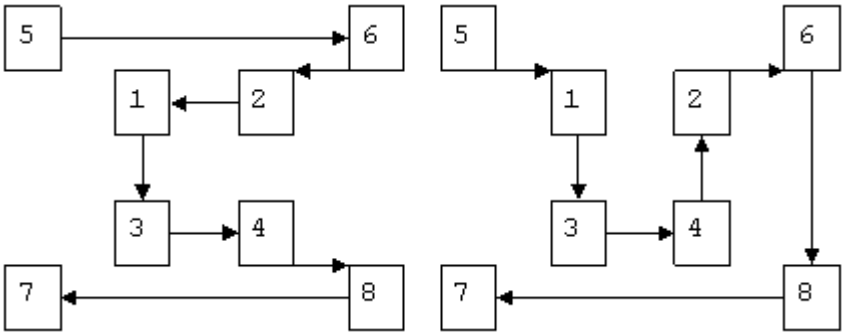
Высокую стойкость шифрования можно обеспечить усложнением перестановок по маршрутам типа гамильтоновских. При этом для записи символов шифруемого текста используются вершины некоторого гиперкуба, а знаки зашифрованного текста считываются по маршрутам Гамильтона, причем используются несколько различных маршрутов. Для примера рассмотрим шифрование по маршрутам Гамильтона при `n=3`.

Струкрура трехмерного гиперкуба:



Номера вершин куба определяют последовательность его заполнения символами шифруемого текста при формировании блока. В общем случае n-мерный гиперкуб имеет n^2 вершин.

Маршруты Гамильтона имеют вид:



Последовательность перестановок символов в шифруемом блоке для первой схемы 5-6-2-1-3-4-8-7, а для второй 5-1-3-4-2-6-8-7. Аналогично можно получить последовательность перестановок для других маршрутов: 5-7-3-1-2-6-8-4, 5-6-8-7-3-1-2-4, 5-1-2-4-3-7-8-6 и т.д.

Размерность гиперкуба, количество вид выбираемых маршрутов Гамильтона составляют секретный ключ метода.

Стойкость простой перестановки однозначно определяется размерами используемой матрицы перестановки. Например, при использовании матрицы 16*16 число возможных перестановок достигает 1.4E26. Такое число вариантов невозможно перебрать даже с использованием ЭВМ. Стойкость усложненных перестановок еще выше. Однако следует иметь в виду, что при шифровании перестановкой полностью сохраняются вероятностные характеристики исходного текста, что облегчает криптоанализ.

Шифрование методом гаммирования

Суть метода состоит в том, что символы шифруемого текста последовательно складываются с символами некоторой специальной последовательности, называемой гаммой. Иногда такой метод представляют как наложение гаммы на исходный текст, поэтому он получил название "***гаммирование***".

Наложение гаммы можно осуществить несколькими способами, например по формуле

$$t_ш = t_о \text{ XOR } t_г,$$

где $t_ш$, $t_о$, $t_г$ - ASCII коды соответственно зашифрованного символа, исходного символа и гаммы,
XOR - побитовая операция "исключающее или".

Расшифрование текста проводится по той же формуле:

$$t_о = t_ш \text{ XOR } t_г.$$

Последовательность гаммы удобно формировать с помощью датчика псевдослучайных чисел (ПСЧ).

Стойкость гаммирования однозначно определяется длиной периода гаммы. При использовании современных ПСЧ реальным становится использование бесконечной гаммы, что приводит к бесконечной теоретической стойкости зашифрованного текста.

Шифрование с помощью аналитических преобразований

Достаточно надежное закрытие информации может обеспечить использование при шифровании некоторых аналитических преобразований. Например, можно использовать методы алгебры матриц - в частности умножение матрицы на вектор.

В качестве ключа задается квадратная матрица ||a|| размера n*n. Исходный текст разбивается на блоки длиной n символов. Каждый блок рассматривается как n-мерный вектор. А процесс шифрования блока заключается в получении нового n-мерного вектора (зашифрованного блока) как результата умножения матрицы ||a|| на исходный вектор.

Расшифрование текста происходит с помощью такого же преобразования, только с помощью матрицы, обратной ||a||. Очевидно, что ключевая матрица ||a|| должна быть невырожденной.

Комбинированные методы шифрования

Достаточно эффективным средством повышения стойкости шифрования является комбинированное использование нескольких различных способов шифрования, т.е. последовательное шифрование исходного текста с помощью двух или более методов.

Стойкость комбинированного шифрования S не ниже произведения стойкостей используемых способов

$$S >= S_1*S_2*...*S_k$$

Если какой-либо способ шифрования при независимом применении может обеспечить стойкость не ниже S, то комбинировать его с другими способами целесообразно лишь при выполнении условия

$$R > R_1+R_2+ ... +R_k,$$

где R_i - трудоемкость i-го способа, используемого при комбинированном шифровании, R - трудоемкость того способа, который обеспечивает стойкость не ниже S.

Организационные проблемы криптозащиты

Рассмотренные значения стойкости шифров являются потенциальными величинами. Они могут быть реализованы при строгом соблюдении правил использования криптографических средств защиты.

Основные правила криптозащиты:

1. Сохранение в тайне ключей.
2. Исключение дублирования.
3. Достаточно частая смена ключей.

Под дублированием здесь понимается повторное шифрование одного и того же отрывка текста с использованием тех же ключей (например, если при первом шифровании произошел сбой). Нарушение этого правила резко снижает надежность шифрования, так как исходный текст может быть восстановлен с помощью статистического анализа двух вариантов зашифрованного текста.

Важнейшим правилом криптозащиты является достаточно частая смена ключей. Причем частота может определяться исходя из длительности использования ключа или исходя из объема зашифрованного текста. При этом смена ключей по временному графику является защитной мерой против возможного их хищения, смена после шифрования определенного объема текста - от раскрытия шифра статистическими методами.

Нельзя допускать злоумышленнику возможности направить в систему ряд специально подобранных сообщений и получать их в зашифрованном виде. Такого взлома не может выдержать ни одна криптосистема!

Важными аспектами организации криптозащиты являются выбор способа закрытия, распределение ключей и доставка их в места пользования (механизм распределения ключей).

Выбор способа защиты тесно связан с трудоемкостью метода шифрования, степенью секретности закрываемых данных, стойкостью метода и объемом шифруемой информации.

Один из принципов криптографии является предположение о несекретности метода закрытия информации. Предполагается, что необходимая надежность закрытия обеспечивается только за счет сохранения в тайне ключей. Отсюда вытекает принципиальная важность формирования ключей, распределения их и доставка в пункты назначения. Основными правилами механизма распределения ключей являются:

1. Ключи должны выбираться случайно.
2. Выбранные ключи должны распределяться таким образом, чтобы не было закономерностей в изменении ключей от пользователя к пользователю.

3. Должна быть обеспечена тайна ключей на всех этапах функционирования системы. Ключи должны передаваться по линиям связи, почте или курьерами в зашифрованном виде с помощью другого ключа. На практике часто образуется иерархия ключей шифрования, в которой ключи нижнего уровня при пересылке шифруются с помощью ключей верхнего уровня. Ключ в вершине иерархии не шифруется, а задается и хранится у доверенного лица, рассылается пользователям курьерами. Чем ниже уровень ключа, тем чаще он меняется и рассылается по линиям связи. Подобная схема шифрования ключей часто используется в сетях.

Литература

1. Герасименко В.А., Размахнин М.К. "Криптографические методы в автоматизированных системах" Зарубежная радиоэлектроника,1982,N8
2. Сяо Д., Керр Д., С.Мэдник "Защита ЭВМ",М.,Мир,1982 - рассмотрены практически все аспекты защиты информации в вычислительных и автоматизированных системах.
3. Хоффман Л.Дж. Современные методы защиты информации. М.,Сов.радио, 1980 - наиболее полная и серьезная монография по теории и практике защиты информации
4. Джефф П.Р. Электроника,1973,т.46,N1 - шифрование данных методом гаммирования.
5. Уолкер Б.Дж., Блейк Я.Ф. Безопасность ЭВМ и организация их защиты.- М.:Связь,1980 - рассматриваются криптографические методы защиты информации, защита данных в файлах или базах данных.
6. Мануильямс Ф.Ж.,Слоан Н.Дж. ТИИЭР,1976,т.64,N12 - детально описаны способы формирования и свойства псевдослучайных последовательностей, которые используются при шифровании данных методом гаммирования.

Стандарт шифрования данных Data Encryption Standard

В 1977 году Национальное бюро Стандартов США (NBS) опубликовало стандарт шифрования данных ***Data Encryption Standard (DES)***, предназначенный для использования в государственных и правительственных учреждениях США для защиты от несанкционированного доступа важной, но несекретной информации. Алгоритм, положенный в основу стандарта, распространялся достаточно быстро, и уже в 1980 году был одобрен ANSI. С этого момента DES превращается в стандарт не только по названию (Data Encryption Standard), но и фактически. Появляются программное обеспечение и специализированные микроЭВМ, предназначенные для шифрования/расшифрования информации в сетях передачи данных и на магнитных носителях. К настоящему времени DES является наиболее распространенным алгоритмом, используемым в системах защиты коммерческой информации. Более того реализация алгоритма DES в таких системах является просто признаком хорошего тона! За примерами далеко ходить не надо. Программа DISKREET из пакета Norton Utilities, предназначенная для создания зашифрованных разделов на диске, использует именно алгоритм DES. "Собственный алгоритм шифрования" отличается от DES только числом итераций при шифровании. Почему же DES добился такой популярности?

Основные ***достоинства алгоритма DES***:

- используется только один ключ длиной 56 битов;
- зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
- относительная простота алгоритма обеспечивает высокую скорость обработки информации;
- достаточно высокая стойкость алгоритма.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности (несмотря на кажущуюся очевидность, так делается далеко не всегда. Позже мы рассмотрим шифры, в которых шифрование и расшифрование осуществляются по разным алгоритмам).

Процесс шифрования заключается в начальной перестановке битов 64-битового блока, шестнадцати циклах шифрования и, наконец, обратной перестановки битов (рис.1).

Рис.1. Обобщенная схема шифрования в алгоритме DES

Необходимо сразу же отметить, что ВСЕ таблицы, приведенные в данной статье, являются СТАНДАРТНЫМИ, а следовательно должны включаться в вашу реализацию алгоритма в неизменном виде. Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс расшифровки путем подбора ключа. Структура алгоритма DES приведена на рис.2.

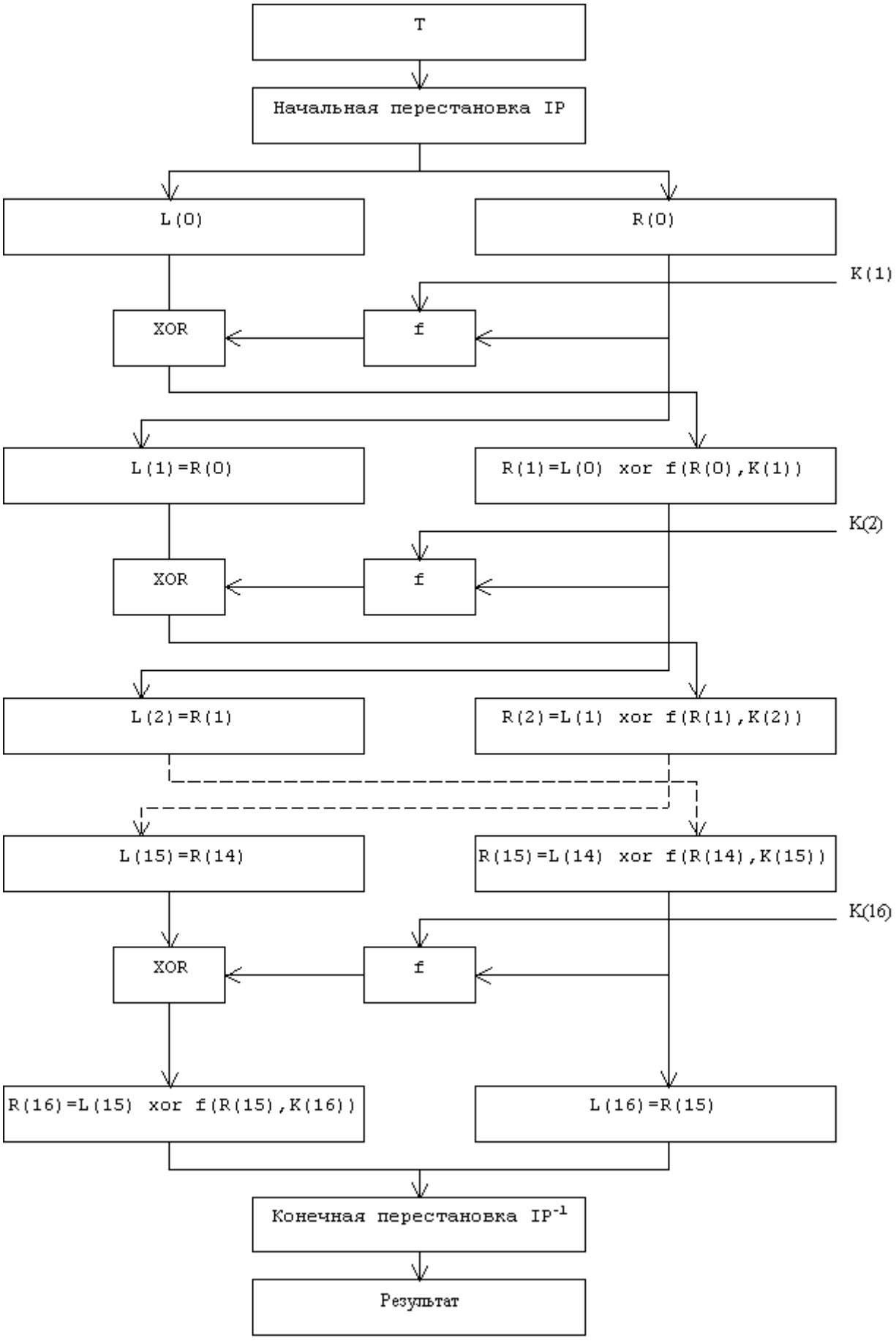


Рис.2. Структура алгоритма шифрования DES

Пусть из файла считан очередной 8-байтовый блок T, который преобразуется с помощью матрицы начальной перестановки IP (табл.1) следующим образом: бит 58 блока T становится битом 1, бит 50 - битом 2 и т.д., что даст в результате: T(0) = IP(T).

Полученная последовательность битов T(0) разделяется на две последовательности по 32 бита каждая: L(0) - левые или старшие биты, R(0) - правые или младшие биты.

Таблица 1: ***Матрица начальной перестановки IP***

58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

Затем выполняется шифрование, состоящее из 16 итераций. Результат i-й итерации описывается следующими формулами:

$$L(i) = R(i-1)$$

$$R(i) = L(i-1) \text{ xor } f(R(i-1), K(i)) ,$$

где xor - операция ИСКЛЮЧАЮЩЕЕ ИЛИ.

Функция f называется функцией шифрования. Ее аргументы - это 32-битовая последовательность R(i-1), полученная на (i-1)-ой итерации, и 48-битовый ключ K(i), который является результатом преобразования 64-битового ключа K. Подробно функция шифрования и алгоритм получения ключей K(i) описаны ниже.

На 16-й итерации получают последовательности R(16) и L(16) (без перестановки), которые конкатенируют в 64-битовую последовательность R(16)L(16).

Затем позиции битов этой последовательности переставляют в соответствии с матрицей IP^{-1} (табл.2).

Таблица 2: ***Матрица обратной перестановки IP^{-1}***

40	08	48	16	56	24	64	32
39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	41	09	49	17	57	25

Матрицы IP^{-1} и IP соотносятся следующим образом: значение 1-го элемента матрицы IP^{-1} равно 40, а значение 40-го элемента матрицы IP равно 1, значение 2-го элемента матрицы IP^{-1} равно 8, а значение 8-го элемента матрицы IP равно 2 и т.д.

Процесс расшифрования данных является инверсным по отношению к процессу шифрования. Все действия должны быть выполнены в обратном порядке. Это означает, что расшифровываемые данные сначала переставляются в соответствии с матрицей IP^{-1}, а затем над последовательностью бит R(16)L(16) выполняются те же действия, что и в процессе шифрования, но в обратном порядке.

Итеративный процесс расшифрования может быть описан следующими формулами:

$$R(i-1) = L(i),\ i = 1, 2, ..., 16;$$

$$L(i-1) = R(i) \text{ xor } f(L(i), K(i)),\ i = 1, 2, ..., 16 .$$

На 16-й итерации получают последовательности L(0) и R(0), которые конкатенируют в 64-битовую последовательность L(0)R(0).

Затем позиции битов этой последовательности переставляют в соответствии с матрицей IP. Результат такой перестановки - исходная 64-битовая последовательность.

Теперь рассмотрим функцию шифрования f(R(i-1),K(i)). Схематически она показана на рис. 3.

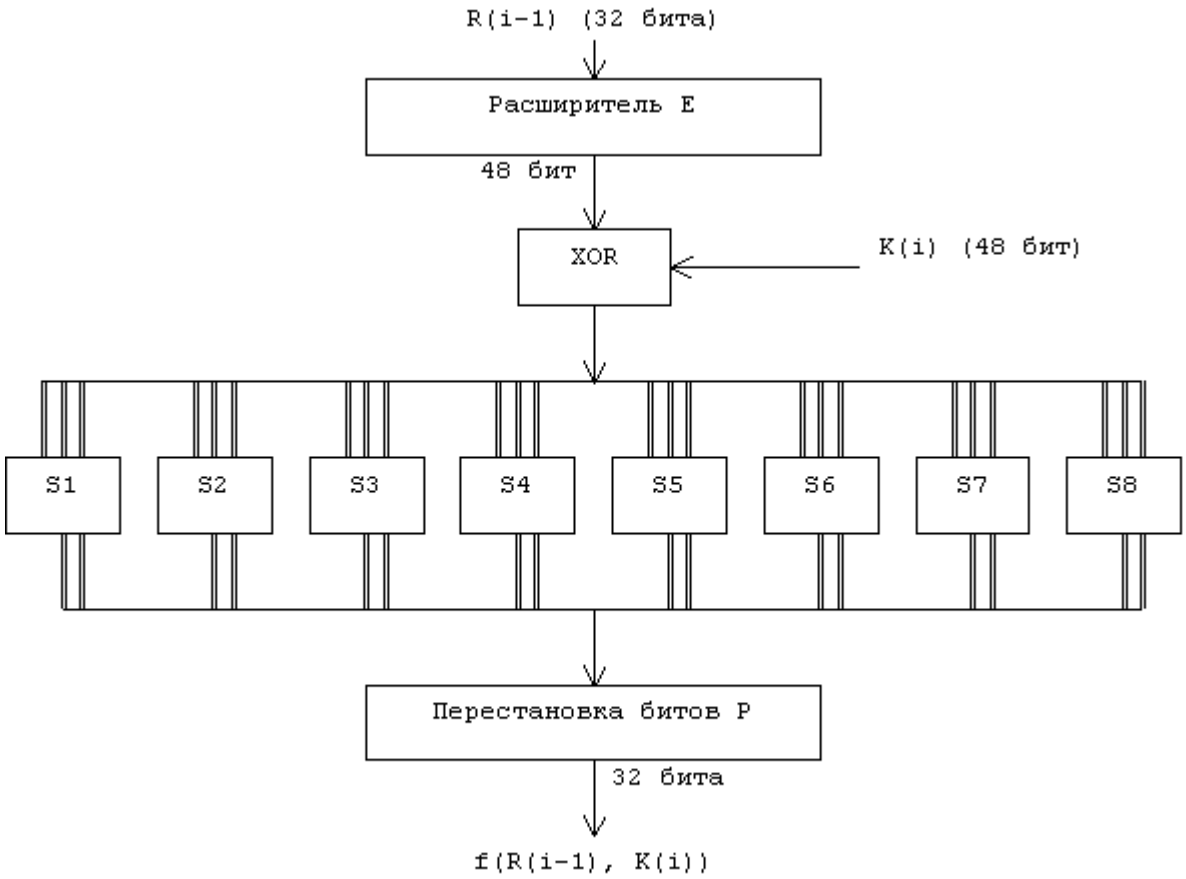


Рис.3. Вычисление функции f(R(i-1), K(i))

Для вычисления значения функции f используются следующие функции-матрицы:

- E - расширение 32-битовой последовательности до 48-битовой,

- S1, S2, ... , S8 - преобразование 6-битового блока в 4-битовый,
- P - перестановка бит в 32-битовой последовательности.

Функция расширения E определяется табл.3. В соответствии с этой таблицей первые 3 бита E(R(i-1)) - это биты 32, 1 и 2, а последние - 31, 32 и 1.

Таблица 3:***Функция расширения E***

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

Результат функции E(R(i-1)) есть 48-битовая последовательность, которая складывается по модулю 2 (операция xor) с 48-битовым ключом K(i). Получается 48-битовая последовательность, которая разбивается на восемь 6-битовых блоков B(1)B(2)B(3)B(4)B(5)B(6)B(7)B(8). То есть:

$$E(R(i-1)) \text{ xor } K(i) = B(1)B(2)...B(8) .$$

Функции S1, S2, ... , S8 определяются табл.4.

Таблица 4

Функции преобразования S1, S2, ..., S8

		Номер столбца															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Номер строки	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S1																	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S2																	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S3																	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S4																	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S5																	

	0	12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11	
	1	10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8	S6
	2	9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6	
	3	4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13	
	0	4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1	
	1	13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6	S7
	2	1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2	
	3	6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12	
	0	13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7	
	1	1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2	S8
	2	7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8	
	3	2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11	

К табл.4. требуются дополнительные пояснения. Пусть на вход функции-матрицы Sj поступает 6-битовый блок B(j) = b1b2b3b4b5b6, тогда двухбитовое число b1b6 указывает номер строки матрицы, а b2b3b4b5 - номер столбца. Результатом Sj(B(j)) будет 4-битовый элемент, расположенный на пересечении указанных строки и столбца.

Например, B(1)=011011. Тогда S1(B(1)) расположен на пересечении строки 1 и столбца 13. В столбце 13 строки 1 задано значение 5. Значит, S1(011011)=0101.

Применив операцию выбора к каждому из 6-битовых блоков B(1), B(2), ..., B(8), получаем 32-битовую последовательность S1(B(1))S2(B(2))S3(B(3))...S8(B(8)).

Наконец, для получения результата функции шифрования надо переставить биты этой последовательности. Для этого применяется функция перестановки P (табл.5). Во входной последовательности биты перестанавливаются так, чтобы бит 16 стал битом 1, а бит 7 - битом 2 и т.д.

Таблица 5:***Функция перестановки P***

16	07	20	21
29	12	28	17
01	15	23	26
05	18	31	10
02	08	24	14
32	27	03	09
19	13	30	06
22	11	04	25

Таким образом,

$$f(R(i-1), K(i)) = P(S1(B(1)),...S8(B(8)))$$

Чтобы завершить описание алгоритма шифрования данных, осталось привести алгоритм получения 48-битовых ключей K(i), i=1...16. На каждой итерации используется новое значение ключа K(i), которое вычисляется из начального ключа K. K представляет собой 64-битовый блок с восемью битами контроля по четности, расположенными в позициях 8,16,24,32,40,48,56,64.

Для удаления контрольных битов и перестановки остальных используется функция G первоначальной подготовки ключа (табл.6).

Таблица 6

Матрица G первоначальной подготовки ключа

57	49	41	33	25	17	09
01	58	50	42	34	26	18
10	02	59	51	43	35	27
19	11	03	60	52	44	36
63	55	47	39	31	23	15
07	62	54	46	38	30	22
14	06	61	53	45	37	29
21	13	05	28	20	12	04

Результат преобразования G(K) разбивается на два 28-битовых блока C(0) и D(0), причем C(0) будет состоять из битов 57, 49, ..., 44, 36 ключа K, а D(0) будет состоять из битов 63, 55, ..., 12, 4 ключа K. После определения C(0) и D(0) рекурсивно определяются C(i) и D(i), i=1...16. Для этого применяют циклический сдвиг влево на один или два бита в зависимости от номера итерации, как показано в табл.7.

Таблица 7

Таблица сдвигов для вычисления ключа

Номер итерации	Сдвиг (бит)
01	1
02	1
03	2
04	2
05	2
06	2
07	2
08	2
09	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Полученное значение вновь "перемешивается" в соответствии с матрицей H (табл.8).

Таблица 8:***Матрица H завершающей обработки ключа***

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Ключ K(i) будет состоять из битов 14, 17, ..., 29, 32 последовательности C(i)D(i). Таким образом:

$$K(i) = H(C(i)D(i))$$

Блок-схема алгоритма вычисления ключа приведена на рис.4.

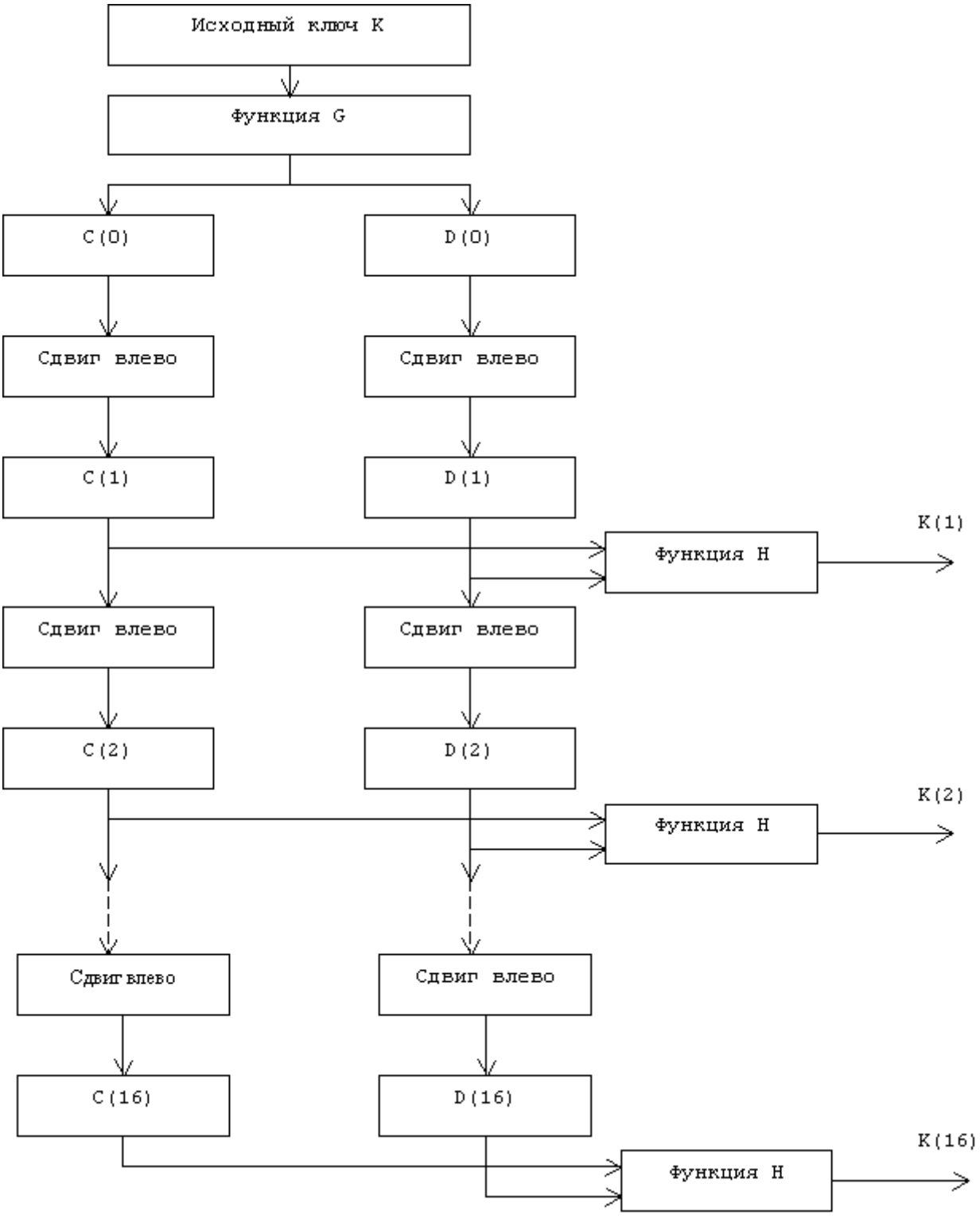


Рис.4. Блок-схема алгоритма вычисления ключа K(i)

Восстановление исходного текста осуществляется по этому алгоритму, но вначале вы используете ключ K(15), затем - K(14) и так далее. Теперь вам должно быть понятно, почему автор настойчиво рекомендует использовать приведенные матрицы. Если вы начнете самовольничать, вы, должно быть, получите очень секретный шифр, но вы сами не сможете его потом раскрыть!

Режимы работы алгоритма DES

Для наиболее полного удовлетворения всем требованиям, предъявляемым к коммерческим системам шифрования, реализованы несколько режимов работы алгоритма DES. Наиболее широкое распространение получили режимы:

- электронный шифроблокнот (Electronic Codebook) - ECB;
- цепочка цифровых блоков (Cipher Block Chaining) - CBC;
- цифровая обратная связь (Cipher Feedback) - CFB;
- внешняя обратная связь (Output Feedback) - OFB.

Кстати, если вы написали программу защиты данных и хотите дать полноценную рекламу, то автор рекомендует прямо указывать, какие из режимов поддерживает ваше детище. Это, вообще говоря, признак хорошего тона на рынке программного обеспечения средств защиты. Нет смысла раскрывать весь алгоритм, вы просто указываете : DES-CBC или DES-CFB и, как говорится, "умный догадается, а дураку и не надо".

Давайте рассмотрим перечисленные выше режимы

DES-ECB

В этом режиме исходный файл M разбивается на 64-битовые блоки (по 8 байтов): M = M(1)M(2)...M(n). Каждый из этих блоков кодируется независимо с использованием одного и того же ключа шифрования (рис.5). Основное достоинство этого алгоритма - простота реализации. Недостаток - относительно слабая устойчивость против квалифицированных криптоаналитиков.

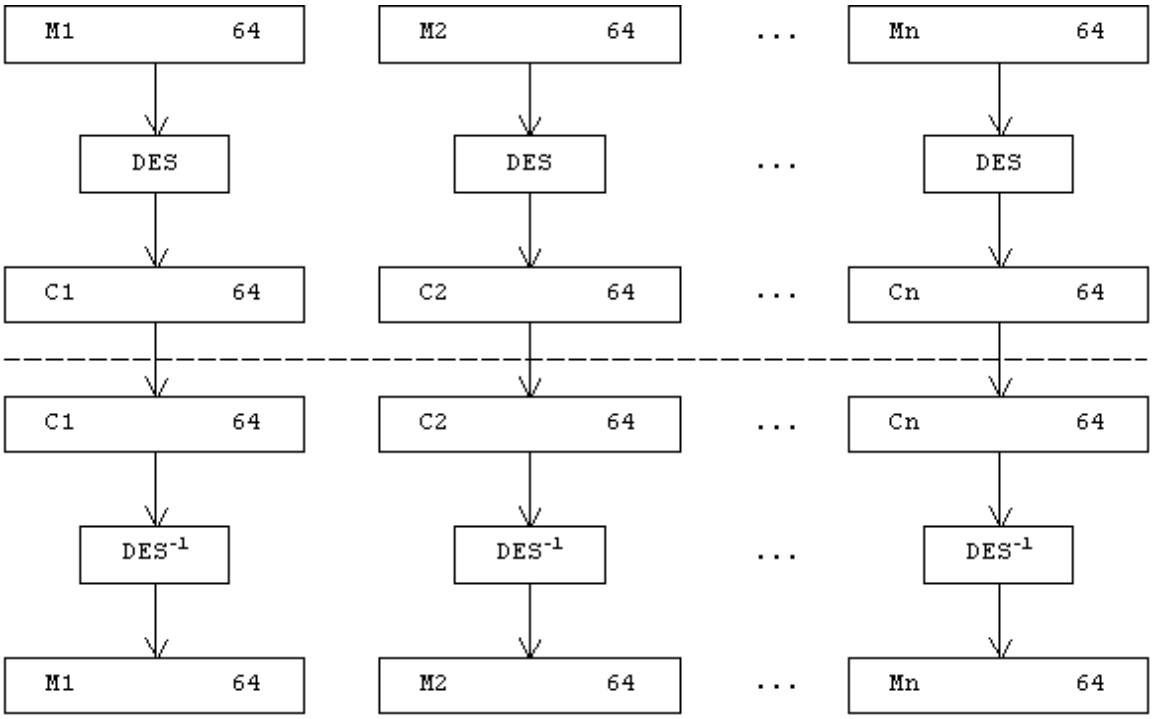


Рис.5. Работа алгоритма DES в режиме ***ECB***

В частности, не рекомендуется использовать данный режим работы для шифрования EXE файлов, потому что первый же блок - заголовок файла, является вполне удачным началом для взлома всего шифра.

В то же время следует признать, что этот режим в силу своей простой реализации наиболее популярен среди любительских разработок.

DES-CBC

В этом режиме исходный файл M также, как и в режиме ECB, разбивается на 64-битовые блоки: M = M(1)M(2)...M(n). Первый блок M(1) складывается по модулю 2 с 64-битовым начальным вектором IV, который меняется ежедневно и держится в секрете. Полученная сумма затем шифруется с использованием ключа DES, известного и отправителю, и получателю информации. Полученный 64-битовый блок шифртекста C(1) складывается по модулю 2 со вторым блоком исходного текста, результат шифруется и получается второй 64-битовый блок шифртекста C(2) и т.д. Процедура повторяется до тех пор, пока не будут обработаны все блоки исходного текста (рис.6).

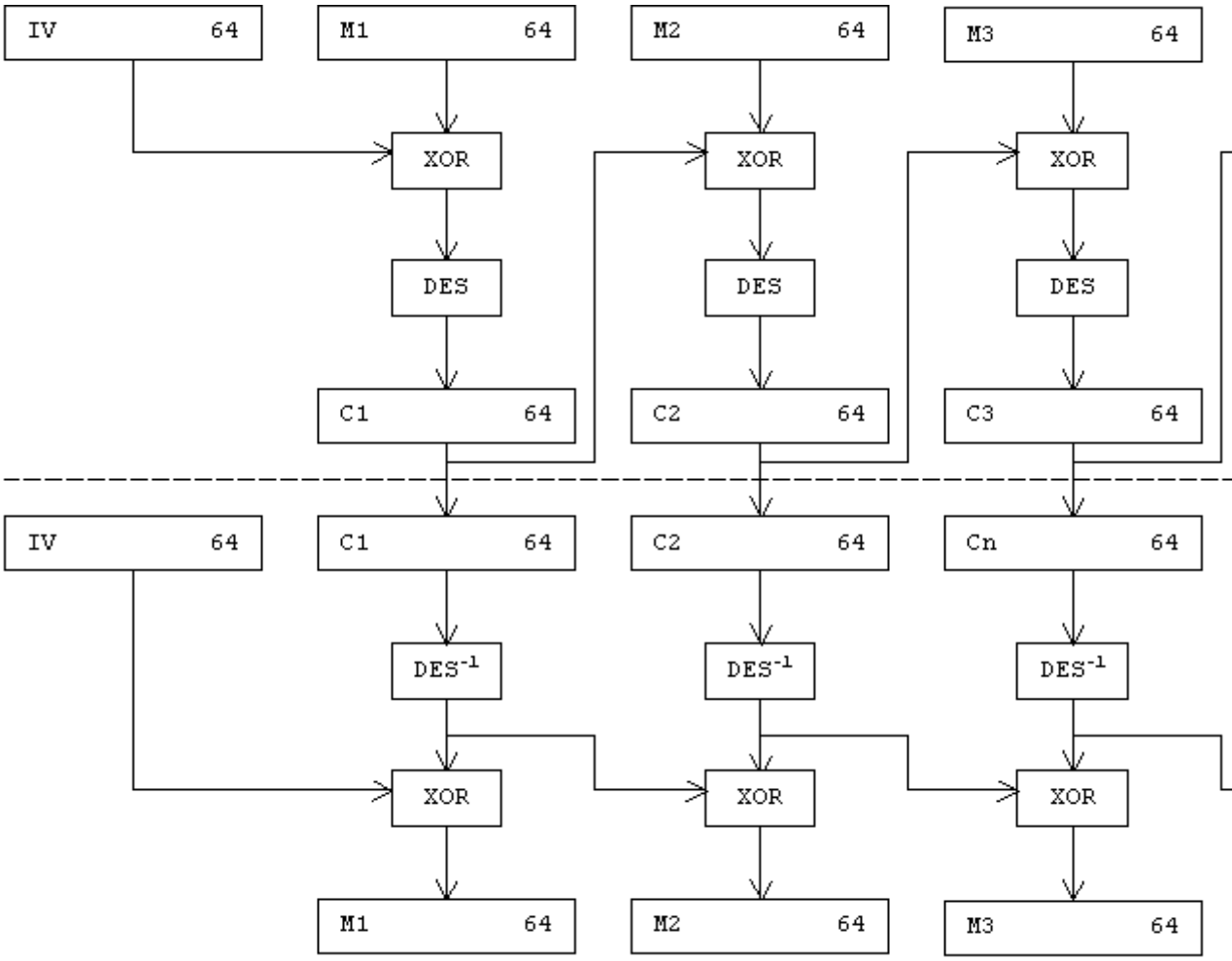


Рис.6. Работа алгоритма в режиме ***CBC***

Таким образом для всех i = 1...n блок шифртекста C(i) определяется следующим образом:

C(i) = DES(M(i) xor C(i-1)),

C(0) = IV - начальное значение шифра, равное начальному вектору.

Расшифрование выполняется следующим образом:

$M(i) = C(i-1) \text{ xor } DES^{-1}(C(i))$,

C(0) = IV - начальное значение шифра, равное начальному вектору.

Прелесть данного режима состоит в том, что он не позволяет накапливаться ошибкам при передаче. Блок M(i) является функцией только C(i-1) и C(i). Поэтому ошибка при передаче приведет к потере только двух блоков исходного текста.

DES-CFB

В этом режиме размер блока может отличаться от 64. Исходный файл M считывается последовательными t-битовыми блоками (t <= 64): M = M(1)M(2)...M(n) (остаток дописывается нулями или пробелами).

64-битовый сдвиговый регистр (входной блок) вначале содержит вектор инициализации IV, выравненный по правому краю. Для каждого сеанса шифрования используется новый IV.

Для всех i = 1...n блок шифртекста C(i) определяется следующим образом:

$$C(i) = M(i) \text{ xor } P(i-1) ,$$

где P(i-1) - старшие t битов операции DES(C(i-1)), причем C(0)=IV.

Обновление сдвигового регистра осуществляется путем удаления его старших t битов и дописывания справа C(i).

Восстановление зашифрованных данных также не представляет труда: P(i-1) и C(i) вычисляются аналогичным образом и

$$M(i) = C(i) \text{ xor } P(i-1) .$$

Блок-схема режима ***CFB*** приведена на рис.7.

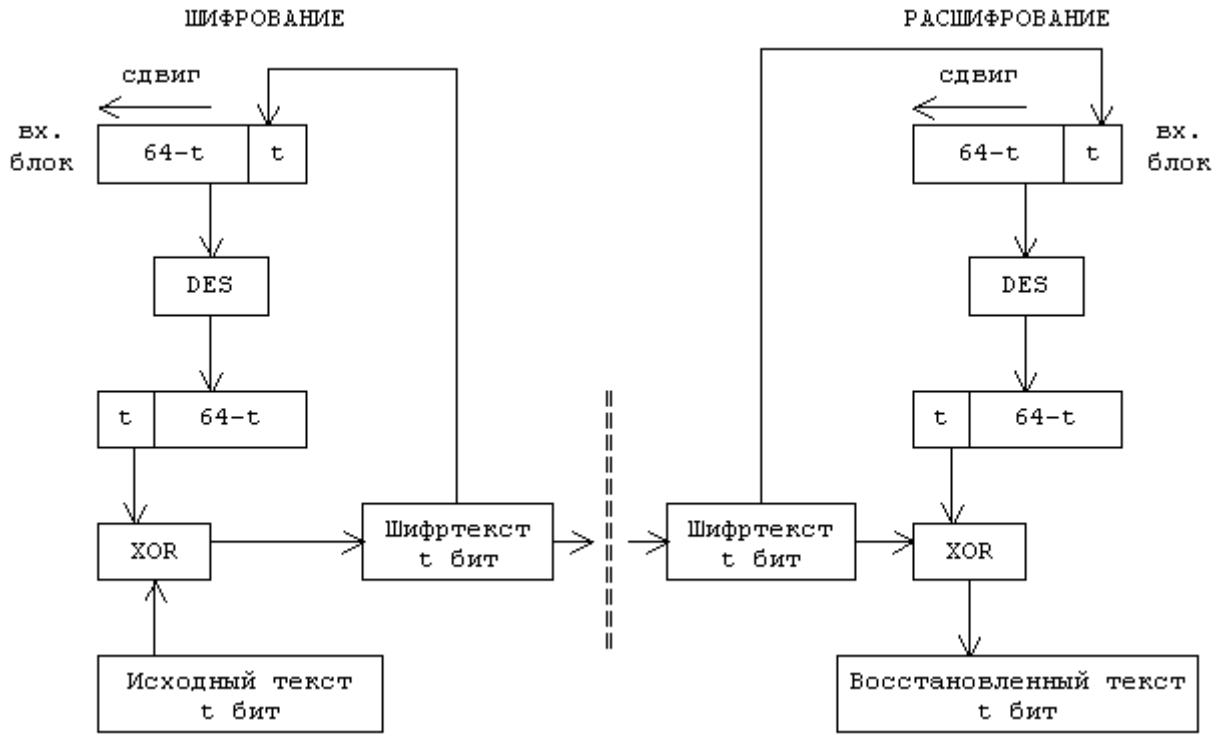


Рис.7. Работа алгоритма DES в режиме CFB

DES-OFB

Режим ***OFB*** очень похож на режим CFB.

Отличие от режима CFB состоит только в методе обновления сдвигового регистра. В данном случае это осуществляется путем удаления его старших t битов и дописывания справа P(i-1) (рис.8).

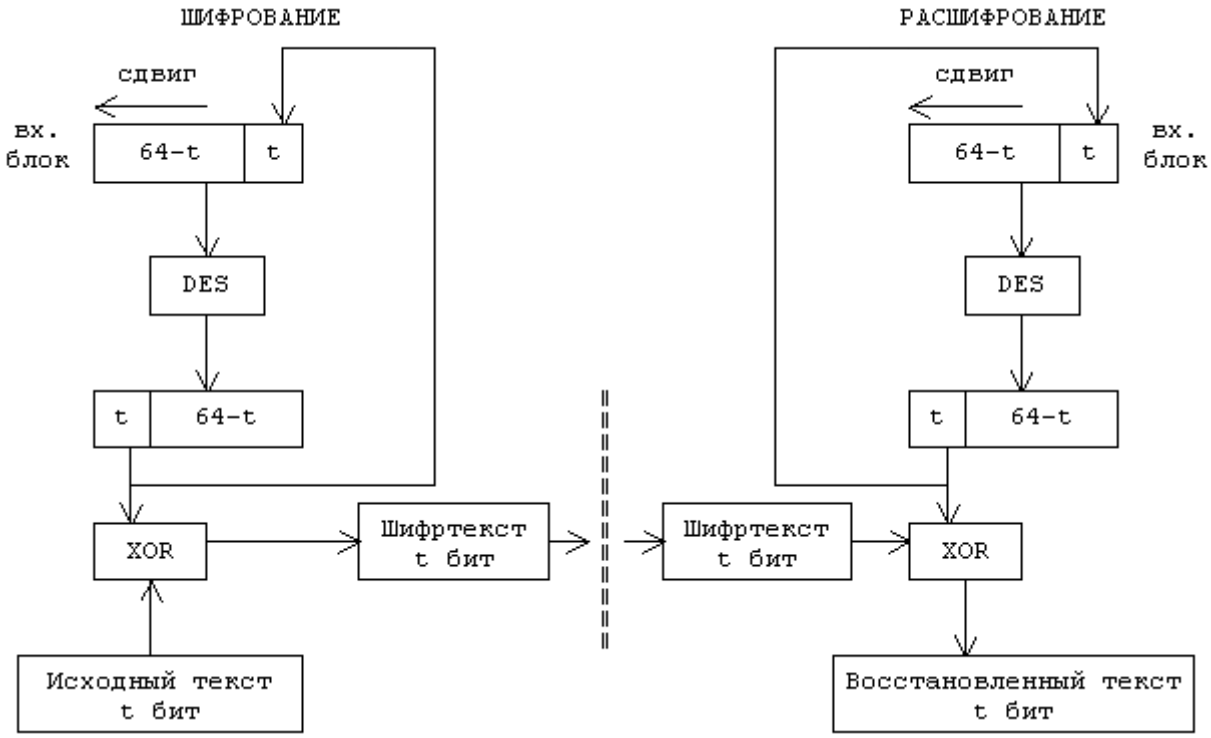


Рис.8. Блок-схема алгоритма DES в режиме OFB

Каждому из рассмотренных режимов свойственны свои достоинства и недостатки, что обусловливает области их применения.

Режим ***ECB*** хорошо подходит для шифрования ключей. Режимы CBC и CFB пригодны для аутентификации данных. Режим CFB, кроме того, предназначен для шифрования отдельных символов. Режим OFB нередко используется в спутниковых системах связи.

Алгоритм шифрования данных IDEA

Алгоритм ***IDEA*** (***International Data Encryption Algorithm***) является блочным шифром. Он оперирует 64-битовыми блоками открытого текста. Несомненным достоинством алгоритма IDEA является то, что его ключ имеет длину 128 бит. Один и тот же алгоритм используется и для шифрования, и для дешифрования.

Первая версия алгоритма IDEA была предложена в 1990 г., ее авторы - Х.Лей и Дж.Мэсси. Первоначальное алгоритм назывался ***PES*** (Proposed Encryption Standard). Улучшенный вариант этого алгоритма, разработанный в 1991 г., получил название ***IPES*** (Improved Proposed Encryption Standard). В 1992 г. IPES изменил свое имя на IDEA. Алгоритм IDEA использует при шифровании процессы смешивания и рассеивания, которые легко реализуются аппаратными и программными средствами.

В IDEA используются следующие математические операции:

- поразрядное сложение по модулю 2 (операция "исключающее ИЛИ"); операция обозначается как (+);
- сложение беззнаковых целых по модулю 2^{16}; операция обозначается как [+];
- умножение беззнаковых целых по модулю (2^{16}+1), причем блок из 16 нулей рассматривается как 2^{16}; операция обозначается как (·).

Все операции выполняются над 16-битовыми субблоками.

Эти три операции несовместимы в том смысле, что:

- никакая пара из этих трех операций не удовлетворяет ассоциативному закону, например a[+](b(+)c)#(a[+]b)(+)c;
- никакая пара из этих трех операций не удовлетворяет дистрибутивному закону, например a[+](b(·)c)#(a[+]b)(·)(a[+]c).

Комбинирование этих трех операций обеспечивает комплексное преобразование входных данных, существенно затрудняя крипто-анализ IDEA по сравнению с DES, который базируется исключительно на операции "исключающее ИЛИ".

Общая схема алгоритма IDEA приведена на рис.1. 64-битовый блок данных делится на четыре 16-битовых субблока. Эти четыре субблока становятся входом в первый цикл алгоритма. Всего выполняется восемь циклов. Между циклами второй и третий субблоки меняются местами. В каждом цикле выполняется следующая последовательность операций:

1. (·) - умножение субблока X_1 и первого подключа.
2. [+] - сложение субблока X_2 и второго подключа.
3. [+] - сложение субблока X_3 и третьего подключа.
4. (·) - умножение субблока X_4 и четвертого подключа.
5. (+) - сложение результатов шагов 1 и 3.
6. (+) - сложение результатов шагов 2 и 4.
7. (·) - умножение результата шага 5 и пятого подключа.
8. [+] - сложение результатов шагов 6 и 7.
9. (·) - умножение результата шага 8 и шестого подключа.

10. [+] - сложение результатов шагов 7 и 9.
11. (+) - сложение результатов шагов 1 и 9.
12. (+) - сложение результатов шагов 3 и 9.
13. (+) - сложение результатов шагов 2 и 10.
14. (+) - сложение результатов шагов 4 и 10.

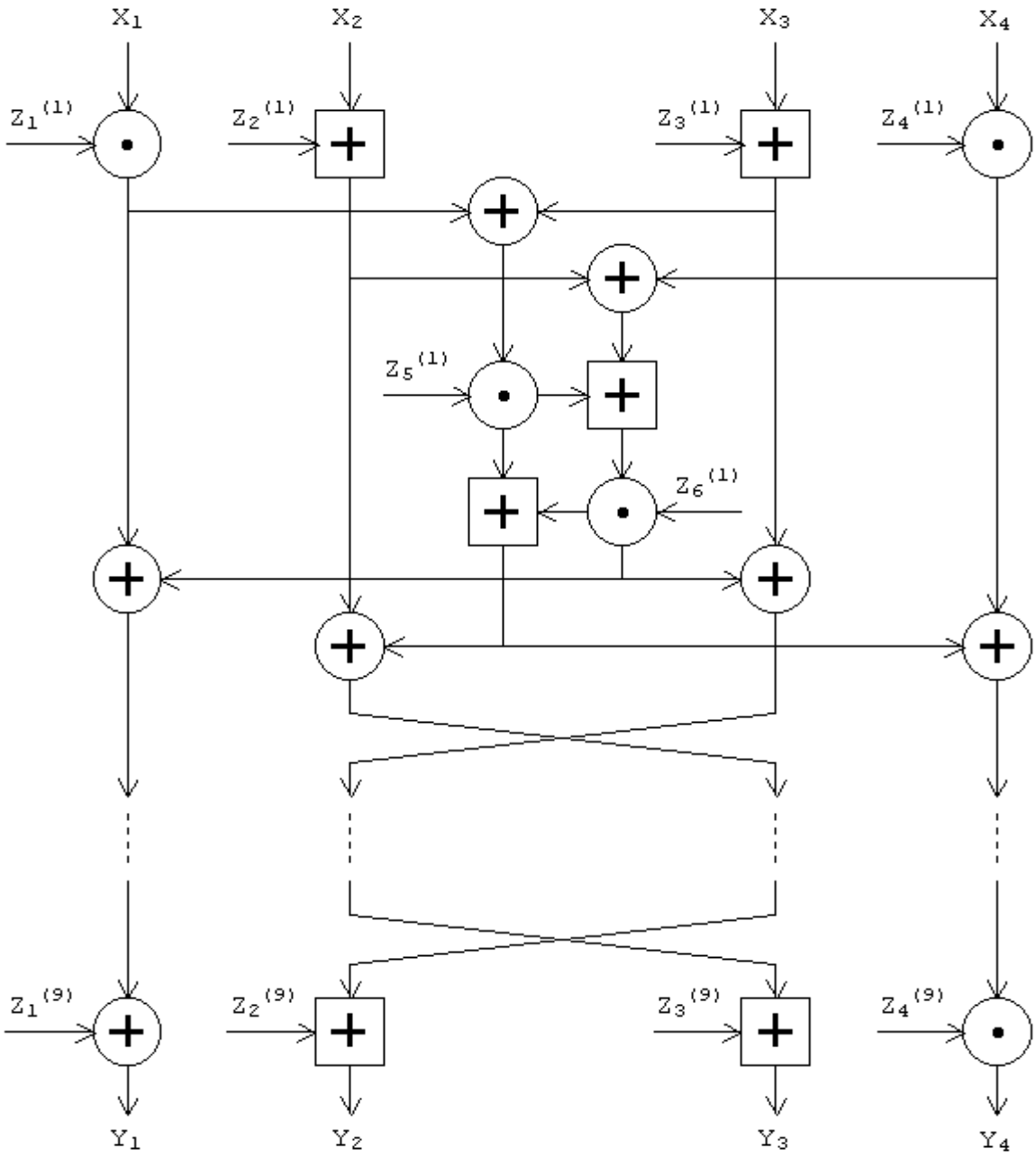


Рис.1. Схема алгоритма IDEA (режим шифрования)

Выходом цикла являются четыре субблока, которые получаются как результаты выполнения шагов 11, 12, 13 и 14. В завершение цикла второй и третий субблоки меняются местами (за исключением последнего цикла). В результате формируется вход для следующего цикла.

После восьмого цикла осуществляется заключительное преобразование выхода:

1. (·) - умножение субблока X_1 и первого подключа.
2. [+] - сложение субблока X_2 и второго подключа.
3. [+] - сложение субблока X_3 и третьего подключа.
4. (·) - умножение субблока X_4 и четвертого подключа.

Полученные четыре субблока $Y_1...Y_4$ объединяют в блок шифртекста.

Создание подключей $Z_1...Z_6$ также относительно несложно. Алгоритм использует всего 52 подключа (по шесть для каждого из восьми циклов и еще четыре для преобразования выхода). Сначала 128-битовый ключ делится на восемь 16-битовых подключей. Это - первые восемь подключей для алгоритма (шесть подключей - для первого цикла и первые два подключа - для второго). Затем 128-битовый ключ циклически сдвигается влево на 25 бит и снова делится на восемь подключей (четыре подключа - для второго цикла и четыре подключа - для третьего). Ключ снова циклически сдвигается влево на 25 бит для получения следующих восьми подключей и т.д., пока выполнение алгоритма не завершится.

Дешифрование осуществляется аналогичным образом, за исключением того, что порядок использования подключей становится обратным, причем ряд подключей дешифрования являются или аддитивными (-x), или мультипликативными (1/x) обратными величинами подключей шифрования (табл.1).

Таблица 1

Подключи шифрования и дешифрования алгоритма IDEA

Цикл	Подключи шифрования	Подключи дешифрования
1	$Z_1^{(1)}\ Z_2^{(1)}\ Z_3^{(1)}\ Z_4^{(1)}\ Z_5^{(1)}\ Z_6^{(1)}$	$Z_1^{(9)-1}\ -Z_2^{(9)}\ -Z_3^{(9)}\ Z_4^{(9)-1}\ Z_5^{(8)}\ Z_6^{(8)}$
2	$Z_1^{(2)}\ Z_2^{(2)}\ Z_3^{(2)}\ Z_4^{(2)}\ Z_5^{(2)}\ Z_6^{(2)}$	$Z_1^{(8)-1}\ -Z_3^{(8)}\ -Z_2^{(8)}\ Z_4^{(8)-1}\ Z_5^{(7)}\ Z_6^{(7)}$
3	$Z_1^{(3)}\ Z_2^{(3)}\ Z_3^{(3)}\ Z_4^{(3)}\ Z_5^{(3)}\ Z_6^{(3)}$	$Z_1^{(7)-1}\ -Z_2^{(7)}\ -Z_3^{(7)}\ Z_4^{(7)-1}\ Z_5^{(6)}\ Z_6^{(6)}$
4	$Z_1^{(4)}\ Z_2^{(4)}\ Z_3^{(4)}\ Z_4^{(4)}\ Z_5^{(4)}\ Z_6^{(4)}$	$Z_1^{(6)-1}\ -Z_3^{(6)}\ -Z_2^{(6)}\ Z_4^{(6)-1}\ Z_5^{(5)}\ Z_6^{(5)}$
5	$Z_1^{(5)}\ Z_2^{(5)}\ Z_3^{(5)}\ Z_4^{(5)}\ Z_5^{(5)}\ Z_6^{(5)}$	$Z_1^{(5)-1}\ -Z_2^{(5)}\ -Z_3^{(5)}\ Z_4^{(5)-1}\ Z_5^{(4)}\ Z_6^{(4)}$
6	$Z_1^{(6)}\ Z_2^{(6)}\ Z_3^{(6)}\ Z_4^{(6)}\ Z_5^{(6)}\ Z_6^{(6)}$	$Z_1^{(4)-1}\ -Z_3^{(4)}\ -Z_2^{(4)}\ Z_4^{(4)-1}\ Z_5^{(3)}\ Z_6^{(3)}$
7	$Z_1^{(7)}\ Z_2^{(7)}\ Z_3^{(7)}\ Z_4^{(7)}\ Z_5^{(7)}\ Z_6^{(7)}$	$Z_1^{(3)-1}\ -Z_2^{(3)}\ -Z_3^{(3)}\ Z_4^{(3)-1}\ Z_5^{(2)}\ Z_6^{(2)}$
8	$Z_1^{(8)}\ Z_2^{(8)}\ Z_3^{(8)}\ Z_4^{(8)}\ Z_5^{(8)}\ Z_6^{(8)}$	$Z_1^{(2)-1}\ -Z_3^{(2)}\ -Z_2^{(2)}\ Z_4^{(2)-1}\ Z_5^{(1)}\ Z_6^{(1)}$
Преобра-зование выхода	$Z_1^{(9)}\ Z_2^{(9)}\ Z_3^{(9)}\ Z_4^{(9)}$	$Z_1^{(1)-1}\ -Z_2^{(1)}\ -Z_3^{(1)}\ Z_4^{(1)-1}$

Для реализации алгоритма IDEA было принято соглашение, что мультипликативная обратная величина (1/x) от 0 равна 0.

Алгоритм IDEA обладает рядом преимуществ перед алгоритмом DES. Он зачительно безопаснее алгоритма DES, поскольку 128-битовый ключ алгоритма IDEA вдвое больше ключа DES. Внутренняя структура алгоритма IDEA обеспечивает лучшую устойчивость к ***криптоанализ***у. Существующие программные реализации примерно вдвое быстрее реализаций алгоритма DES. Алгоритм IDEA запатентован в Европе и США.

Отечественный стандарт шифрования данных

В нашей стране установлен единый алгоритм криптографического представления данных для систем обработки информации в сетях ЭВМ, отдельных вычислительных комплексов и ЭВМ, который определяется ***ГОСТ 28147-89***.

Этот алгоритм криптографического преобразования данных представляет собой 64-битовый блочный алгоритм с 256-битовым ключом, предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации.

При описании алгоритма используются следующие обозначения:

L и R - последовательности битов;
LR - конкатенация последовательностей L и R, в которой биты последовательности R следуют за битами последовательности L;
(+) - поразрядное сложение по модулю 2 (операция "исключающее ИЛИ");
[+] - сложение 32-разрядных чисел по модулю 2^{32};
{+} - сложение 32-разрядных чисел по модулю 2^{32}-1.

Числа суммируются по следующему правилу:

A [+] B = A + B, если A + B < 2^{32},
A [+] B = A + B - 2^{32}, если A + B >= 2^{32}.
A {+} B = A + B , если A + B < 2^32 - 1,
A {+} B = A + B - (2^32 - 1), если A + B >= 2^32 - 1.

Алгоритм предусматривает четыре режима работы:

- простая замена;
- гаммирование;
- гаммирование с обратной связью;
- выработка имитовставки.

В любом случае для шифрования данных используется 256-битовый ключ K, который представляется в виде восьми 32-битовых подключей K_i:

$K = K_7K_6K_5K_4K_3K_2K_1K_0$.

Расшифрование выполняется по тому же ключу, что и шифрование, но этот процесс является инверсией процесса шифрования данных.

Режим простой замены

Первый и самый простой режим - ***замена***. Данные, подлежащие шифрованию, разбивают на 64-битовые блоки. Процедура шифрования блока открытых данных T_0 включает 32 цикла (j=1...32).

Блок T_0 разделяется на две последовательности по 32 бита: B(0)A(0), где B(0) - левые или старшие биты, A(0) - правые или младшие биты.

Эти последовательности вводят в накопители N_1 и N_2 перед началом первого цикла шифрования.

Первый цикл (j=1) процедуры шифрования 64-битового блока данных описывается следующими формулами:

$$\begin{cases} A(1) = f\ (\ A(0)\ [+]\ K_0\ )\ (+)\ B(0), \\ B(1)=A(0). \end{cases}$$

Здесь A(1) - заполнение накопителя N_1 после 1-го цикла шифрования

$$\begin{cases} A(i) = f\ (\ A(i-1)\ [+]\ X(j)\ )\ (+)\ B(i-1), \\ B(i) = A(i-1), \text{ если } i=25, 26,..., 31;\ j=32-i \end{cases}$$

$$\begin{cases} A(32) = A(31), \\ B(32) = f\ (\ A(31)\ [+]\ X(0)\ )\ (+)\ B(31), \end{cases}$$

Здесь i обозначает номер итерации (i = 1, 2,..., 32).

Функция f называется функцией шифрования. Ее аргументом является сумма по модулю 2^{32} числа A(i), полученного на предыдущем шаге итерации, и числа X(j) ключа (размерность каждого из этих чисел равна 32 знакам).

Функция шифрования включает две операции над полученной 32-разрядной суммой. Первая операция называется подстановкой К. Блок подстановки К состоит из 8 узлов замены К(1) ... К(8) с памятью 64 бит каждый. Поступающий на блок подстановки 32-разрядный вектор разбивается на 8 последовательно идущих 4-х разрядных векторов, каждый из которых преобразуется в 4-х разрядный вектор соответствующим узлом замены, представляющим собой таблицу из 16 целых чисел в диапазоне 0...15.

Входной вектор определяет адрес строки в таблице, число из которой является выходным вектором. Затем 4-х разрядные выходные векторы последовательно объединяются в 32-разрядный вектор. Таблицы блока подстановки К содержит ключевые элементы, общие для сети ЭВМ и редко изменяемые.

Вторая операция - циклический сдвиг влево 32-разрядного вектора, полученного в результате подстановки К. 64-разрядный блок зашифрованных данных $T_ш$ представляется в виде $T_ш$=A(32)B(32).

Остальные блоки открытых данных в режиме простой замены зашифровываются аналогично.

Следует иметь в виду, что режим простой замены допустимо использовать для шифрования данных только в ограниченных случаях. К этим случаям относится выработка ключа и зашифрование его с обеспечением

имитозащиты (защиты от навязывания ложных данных) для передачи по каналам связи или хранения в памяти ЭВМ.

Режим гаммирования

Открытые данные, разбитые на 64-разрядные блоки T(i) (i=1, 2,..., m, где m определяется обьемом шифруемых данных), зашифровываются в режиме гаммирования путем поразрядного сложения по модулю 2 с гаммой шифра $Г_ш$, которая вырабатывается блоками по 64 бит, то есть $Г_ш$ = (Г(1),Г(2),...,Г(i),...,Г(m)).

Число двоичных разрядов в блоке T(m) может быть меньше 64, при этом неиспользованная для шифрования часть гаммы шифра из блока Г(m) отбрасывается.

Уравнение зашифрования данных в режиме гаммирования может быть представлено в следующем виде:

$$Ш(i) = A\ (Y(i-1)\ [+]\ C2,\ Z(i-1)\ \{+\}\ C1)\ (+)\ T(i) = Г(i)\ (+)\ T(i)\ .$$

Здесь Ш(i) - 64-разрядный блок зашифрованного текста,
A - функция шифрования в режиме простой замены (аргументами этой функции являются два 32-разрядных числа),
C1 и C2 - константы, заданные в ГОСТ 28147-89,
Y(i) и Z(i) - величины, которые определяются итерационно по мере формирования гаммы следующим образом:
(Y(0), Z(0)) = A(S), где S - 64-разрядная двоичная последовательность (синхропосылка);
(Y(i), Z(i)) = (Y(i-1) [+] C2, Z(i-1) {+} C1) для i = 1, 2,...,m.

Расшифрование данных возможно только при наличии синхропосылки, которая не является секретным элементом шифра и может храниться в памяти ЭВМ или передаваться по каналам связи вместе с зашированными данными.

Режим гаммирования с обратной связью

Режим ***гаммирования*** с обратной связью очень похож на режим гаммирования. Как в и режиме гаммирования открытые данные, разбитые на 64-разрядные блоки T(i) (i=1, 2,..., m , где m определяется обьемом шифруемых данных), зашифровываются путем поразрядного сложения по модулю 2 с гаммой шифра $Г_ш$, которая вырабатывается блоками по 64 бит:

$$Г_ш = (Г(1),Г(2),...,Г(i),...,Г(m)).$$

Число двоичных разрядов в блоке T(m) может быть меньше 64, при этом неиспользованная для шифрования часть гаммы шифра из блока Г(m) отбрасывается.

Уравнение зашифрования данных в режиме гаммирования с обратной связью может быть представлено в следующем виде :

Ш(1) = A(S) (+) T(1) = Г(1) (+) T(1),
Ш(i) = A(Ш(i-1)) (+) T(i) = Г(i) (+) T(i), для i = 2,3,...,m.

Здесь Ш(i) - 64-разрядный блок зашифрованного текста, A - функция шифрования в режиме простой замены. Аргументом функции на первом шаге итеративного алгоритма является 64-разрядная синхропосылка, а на всех последующих - предыдущий блок зашифрованных данных Ш(i-1).

Выработки имитовставки

Процесс выработки ***имитовстаки*** единообразен для любого из режимов шифрования данных.

Имитовставка - это блок из p бит (имитовставка Ир), который вырабатывается либо перед шифрованием всего сообщения, либо параллельно с шифрованием по блокам. Первые блоки открытых данных, которые участвуют в выработке имитовставки, могут содержать служебную информацию (например, адресную часть, время, синхропосылку) и не зашифровываться. Значение параметра p (число двоичных разрядов в имитовставке) определяется криптографическими требованиями с учетом того, что вероятность навязывания ложных помех равна 1/2^p.

Для получения имитовставки открытые данные представляются в виде 64-разрядных блоков T(i) (i = 1, 2,..., m , где m определяется объемом шифруемых данных). Первый блок открытых данных T(1) подвергается преобразованию, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены. Причем в качестве ключа для выработки имитовставки используется ключ, по которому шифруются данные.

Полученное после 16 циклов работы 64-разрядное число суммируется по модулю 2 со вторым блоком открытых данных T(2). Результат суммирования снова подвергается преобразованию, соответствующему первым 16 циклам алгоритма зашифрования в режиме простой замены. Полученное 64-разрядное число суммируется по модулю 2 с третьим блоком открытых данных T(3) и т.д. Последний блок T(m) при необходимости дополненный до полного 64-разрядного блока нулями, суммируется по модулю 2 с результатом работы на шаге m-1, после чего зашифровывается в режиме простой замены по первым 16 циклам работы алгоритма. Из полученного 64-разрядного числа выбирается отрезок Ир длиной p бит.

Имитовставка Ир передается по каналу связи или в память ЭВМ после зашифрованных данных. Поступившие зашифрованные данные расшифровываются, и из полученных блоков открытых данных T(i) вырабатывается имитовставка Ир', которая затем сравнивается с имитовставкой Ир, полученной из канала связи или из памяти ЭВМ. В случае несовпадения имитовставок все расшифрованные данные считают ложными.

Концепция криптосистемы с открытым ключом

Эффективными системами криптографической защиты являются криптосистемы с открытым ключом, называемые также асимметричными криптосистемами. В таких системах для зашифрования данных используется один ключ, а для расшифрования - другой (отсюда и название - асимметричные).

Первый ключ является открытым и может быть опубликован для использования всеми пользователями системы, которые зашифровывают данные. Расшифрование данных с помощью открытого ключа невозможно. Для расшифрования данных получатель зашифрованной информации использует второй ключ, который является секретным. Разумеется, ключ расшифрования не может быть определен из ключа зашифрования.

Однонаправленные функции

Вся концепция криптосистем с открытым ключом основана на применении однонаправленных функций (one way functions). Однако, точное определение этого класса функций с математической точки зрения дать достаточно сложно. Неформально однонаправленную функцию можно определить следующим образом.

Пусть X и Y - произвольные множества. Функция

$$f(X) \rightarrow Y,$$

является однонаправленной, если для всех x, входящих в X, легко вычислить функцию f(x), и в то же время для большинства y, входящих в Y, получить любое значение x, входящее в X, такое что f(x) = y достаточно сложно (при этом полагают, что существует, по крайней мере, одно такое значение x).

К сожалению, в настящее время математика не в состоянии дать нам ответ на вопрос, существуют ли таковые функции вообще или же это только красивая гипотеза. Тем не менее пытливым умам удалось обнаружить несколько зависимостей, которые могут быть использованы (и используются!) в качестве однонаправленных. Основной критерий причисления функции к классу однонаправленных очень прост - отсутствие эффективных алгоритмов обратного преобразования.

Простейший пример однонаправленной функции - целочисленное умножение. В самом деле, вычислить произведение двух очень больших целых чисел (имеется в виду, с помощью ЭВМ, а не вручную) достаточно легко, но даже самый мощный компьютер с наилучшими известными на сегодняшний день алгоритмами не в состоянии факторизовать (разделить на сомножители) двухсотзначное число, которое является произведением двух сопоставимых по длине простых чисел.

Необходимо отметить, что любая однонаправленная функция (ОНФ) отнесена к этому классу как бы условно. Как показала практика, как только алгоритм получает достаточно широкое распространение, сразу же у определенных групп лиц возникает желание найти обратную функцию, и, поскольку это желание подкрепляется солидными денежными призами, не уверенные в себе кандидаты в ОНФ оказываются на помойке. Так что, господа, у вас есть возможность отличиться и посрамить апологетов буржуйских коммерческих систем шифрования.

Но вернемся к теме. Вторым важным классом функций, используемых в практике построения систем с открытым ключом, являются так называемые однонаправленные функции с черным ходом (trap door one way function). Для порядка введем определение.

Функция

$$f(X) \rightarrow Y$$

относится к классу однонаправленных функций с черным ходом в том случае, если она является однонаправленной и, кроме того, возможно эффективное вычисление инверсной функции, если известен "черный ход" (или, говоря по-русски, секретная строка, число или другая информация, ассоциирующаяся с данной функцией).

Завершая наше "теоретическое" введение, еще раз отметим, что любители математики и особенно "высшей арифметики" - теории чисел - имеют непочатый фронт работ как в области поиска новых однонаправленных функций, так и в области отыскания эффективных алгоритмов обратных преобразований.

Система распределения ключей Диффи-Хеллмана

В традиционных криптографических системах каждая пара пользователей применяет один и тот же секретный ключ для шифрования и расшифровки сообщений. Это означает, что необходим надежный способ передачи ключа от одного пользователя к другому. Если пользователи меняют ключ достаточно часто, его доставка превращается в серьезную проблему. И более того, в традиционной криптографической системе просто невозможно передать информацию новому пользователю системы до тех пор, пока ему не будет по надежному каналу связи передан секретный ключ. И если спецслужбы как-то выходят из этой ситуации, то для коммерческих приложений это никуда не годится. И пытливая инженерная мысль нашла выход - была создана система распределения открытых ключей (public-key distribution system), позволяющая своим пользователям обмениваться секретными ключами по незащищенным каналам связи.

Первой системой такого рода стала система Диффи-Хеллмана, разработанная в 1976 году, построенная на задаче о дискретном логарифмировании.

Предположим, что два пользователя, Алекс и Юстас, применяющие традиционную криптосистему, желают связаться друг с другом. Это означает, что они должны прийти к соглашению относительно ключа K, которым будут шифроваться сообщения. Давайте посмотрим, как система Диффи-Хелмана позволит обменяться ключом.

Пусть N - некоторое большое целое число, а G - другое целое, такое что

$$1 <= G <= N-1.$$

Рассмотрим процедуру обмена ключами по шагам.

1. Вначале Алекс и Юстас достигают соглашения о значениях N и G (как правило, эти значения являются стандартными для всех пользователей системы).
2. Затем Алекс выбирает некоторое большое целое число X и вычисляет XX = G^X MOD N. Аналогичным образом Юстас выбирает число Y и вычисляет

YY = G^Y MOD N.
После этого Алекс и Юстас обмениваются значениями XX и YY. (Мы считаем, что все данные, которые передаются по каналу связи, могут быть перехвачены злоумышленником - стариной Мюллером). Числа X и Y Алекс и Юстас хранят в секрете.

3. Получив от Юстаса число YY, Алекс вычисляет K(1) = YY^X MOD N, а Юстас - K(2) = XX^Y MOD N.

Но (!)

$$YY^X \text{ MOD } N = G^{(X*Y)} \text{ MOD } N = XX^Y \text{ MOD } N,$$

а следовательно,

$$K(1) = K(2) = K.$$

Это значение K и является ключом, который используется для шифрования сообщений.

А что же старина Мюллер? Злоумышленник, перехвативший G, N, XX и YY, тоже должен определить значение ключа K. Очевидный путь для решения задачи состоит в вычислении значения X по G, N, XX или, по крайней мере, некоторого X', такого что

$$G^{X'} \text{ MOD } N = X,$$

поскольку в этом случае

$$YY^{X'} \text{ MOD } N = K.$$

Однако это и есть задача дискретного логарифмирования в чистом виде, которая считается неразрешимой.

Система Диффи-Хеллмана позволяет двум пользователям прийти к соглашению относительно общего секретного ключа. Однако система никак не влияет на то, как потом будет шифроваться сама информация. И если Алекс хочет передать Юстасу секретное сообщение M, то после установления ключа по Диффи-Хеллману может быть использована любая система шифрования.

Но системы с открытым ключом создавались не только и даже не столько для решения задачи распределения ключей. При грамотном подходе возможно эффективное их использование для шифрования информации. Ведь, по определению, система с открытым ключом отличается тем, что тот, кто знает ключ для шифрования, не может дешифровать текст за практически приемлемое время.

Рассмотрим, как же используются системы с открытым ключом.

Пользователь Алекс имеет в своем распоряжении два алгоритма: E для шифрования и D для расшифровки сообщений. При этом алгоритм E делается общедоступным, например, через использование каталога ключей, а алгоритм D хранится Алексом в секрете. Если Юстас или даже старина Мюллер хочет послать Алексу сообщение, он ищет в каталоге ключей алгоритм E и использует его для шифрования передаваемой информации. А вот расшифровать сообщение сможет только Алекс, поскольку алгоритм D есть только у него. Очевидно, что E и D должны удовлетворять условию:

$$D(E(M)) = M,$$

для любого сообщения M.

И снова, как и для традиционных криптосхем, требуется получить эффективные алгоритмы E и D. При этом необходимо, чтобы алгоритм E представлял собой функцию с черным ходом, то есть знание алгоритма E не должно быть достаточным для реализации D.

Системы с открытым ключом могут быть реализованы только в том случае, если подобрана однонаправленная функция с черным ходом. При этом необходимо постоянно помнить, что доказательства однонаправленности не существует. А из этого, в свою очередь, следует, что при выборе кандидатов в однонаправленные функции следует соблюдать известную осторожность, подкрепленную результатами тщательного тестирования.

Система RSA

В настоящее время наиболее развитым методом криптографической защиты информации с известным ключом является RSA, названный так по начальным буквам фамилий его изобретателей (Rivest, Shamir и Adleman). Перед тем как приступить к изложению концепции метода RSA, необходимо определить некоторые термины.

Под простым числом будем понимать такое число, которое делится только на 1 и на само себя. Взаимно простыми числами будем называть такие числа, которые не имеют ни одного общего делителя, кроме 1.

Чтобы использовать алгоритм RSA надо сначало сгенерировать открытый и секретный ключи, выполнив следующие шаги:

1. Выберем два очень больших простых числа p и q.
2. Определим n как результат умножения p на q (n = p*q).
3. Выберем большое случайное число, которое назовем d. Это число должно быть взаимно простым с результатом умножения (p-1) * (q-1).
4. Определим такое число e, для которого является истинным следующее соотношение: (e * d) mod ((p-1) * (q-1)) = 1.
5. Назовем открытым ключем числа e и n, а секретным ключем числа d и n.

Теперь, чтобы зашифровать данные по известному ключу {e,n}, необходимо сделать следующее:

- разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа M(i) = 0, 1,..., n-1;
- зашифровать текст, рассматриваемый как последовательность чисел M(i), по формуле:
  C(i) = (M(i)^e) mod n.

Чтобы расшифровать эти данные используя секретный ключ {d,n}, необходимо выполнить следующие вычисления: M(i) = (C(i)^d) mod n. В результате будет получено множество чисел M(i), которые представляют собой исходный текст.

Приведем простой пример использования метода RSA для шифрования сообщения "CAB". Для простоты будем использовать очень маленькие числа (на практике используются намного большие числа).

1. Выберем p = 3 и q = 11.
2. Определим n = 3*11 = 33.
3. Найдем (p-1) * (q-1) = 20. Следовательно в качестве d выберем любое число, которое является взаимно простым с 20, например d = 3.
4. Выберем число e. В качестве такого числа может быть взято любое число, для которого удовлетворяется соотношение (e*3) mod 20 = 1, например 7.
5. Представим шифруемое сообщение как последовательность целых чисел в диапазоне 0...32. Пусть буква A изображается числом 1, буква B - числом 2, а буква C - числом 3. Тогда сообщение можно представить в виде последовательности чисел 3 1 2.
   Зашифруем сообщение, используя ключ {7,33}:
   C1 = (3^7) mod 33 = 2187 mod 33 = 9,
   C2 = (1^7) mod 33 = 1 mod 33 = 1,
   C3 = (2^7) mod 33 = 128 mod 33 = 29.
6. Попытаемся расшифровать сообщение {9,1,29}, полученное в результате зашифрования по известному ключу на основе секретного ключа {3,33}:
   M1 = (9^3) mod 33 = 729 mod 33 = 3,
   M2 = (1^3) mod 33 = 1 mod 33 = 1,
   M3 = (29^3) mod 33 = 24389 mod 33 = 2.
   Таким образом, в результате расшифрования сообщения получено исходное сообщение "CAB".

Криптостойкость алгоритма RSA основывается на предположении, что исключительно трудно определить секретный ключ по известному, поскольку для этого необходимо решить задачу о существовании делителей целого числа. Данная задача является NP - полной. Известные точные алгоритмы для решения данной задачи имеют экспоненциальную оценку вычислительной сложности, следствием чего является невозможность получения точных решений для задач большой и даже средней размерности. Более того, сам вопрос существования эффективных алгоритмов решения NP - полных задач является до настоящего времени открытым. В связи с этим для чисел, состоящих из 200 цифр (а именно такие числа рекомендуется использовать), традиционные методы требуют выполнения огромного числа операций (около 10^{23}).

Оценки сложности задачи ДИСКРЕТНОГО ЛОГАРИФМИРОВАНИЯ в зависимости от длины двоичной записи простого числа P (при правильном его выборе) приведены в таблице:

Длина P (в битах)	**Сложность определения ключа x**	**Память используемая алгоритмом (в битах)**	**Время решения задачи на компьюре типа 10^9 оп/с**
128	$2*10^{12}$	$7*10^6$	Несколько минут
200	10^{16}	10^8	Несколько месяцев
256	$9*10^{17}$	10^9	Несколько десятков лет
512	$4*10^{24}$	$3*10^{12}$	Более 100 лет непрерывной работы

1024	10^{34}	10^{17}
1500	10^{41}	$8*10^{20}$
2000	$7*10^{47}$	10^{24}
2200	10^{50}	10^{25}

Все асимметричные криптосистемы пытаются взломать путем прямого перебора ключей. Поэтому в асимметричных криптосистемах используют длинные ключи. Для обеспечения эквивалентного уровня защиты ключ асимметричной криптосистемы должен быть гораздо длиннее ключа симметричной криптосистемы. Это сразу же сказывается на вычислительных ресурсах, требуемых для шифрования. Брюс Шнейер в книге "Прикладная криптография: протоколы, алгоритмы и исходный текст на С" приводит следующие данные об эквивалентных длинах ключей.

Длина симметричного ключа (в битах)	**Длина открытого ключа (в битах)**
56	384
64	512
80	768
112	1792
128	2304

Для того чтобы избежать низкой скорости алгоритмов асимметричного шифрования, генерируется временный симметричный ключ для каждого сообщения и только он шифруется асимметричными алгоритмами. Само сообщение шифруется с использованием этого временного сеансового ключа. Затем этот сеансовый ключ шифруется с помощью открытого асимметричного ключа получателя и асимметричного алгоритма шифрования. После этого этот зашифрованный сеансовый ключ вместе с зашифрованным сообщением передается получателю. Получатель использует тот же самый асимметричный алгоритм шифрования и свой секретный ключ для расшифровки сеансового ключа, а полученный сеансовый ключ используется для расшифровки самого сообщения.

В асимметричных криптосистемах важно, чтобы сеансовые и асимметричные ключи были сопоставимы в отношении уровня безопасности, который они обеспечивают. Если используется короткий сеансовый ключ (например, DES), то не имеет значения, насколько велики асимметричные ключи. Хакеры будут атаковать не их, а сеансовые ключи. Асимметричные открытые ключи уязвимы к атакам прямым перебором отчасти из-за того, что их тяжело заменить. Если атакующий узнает секретный асимметричный ключ, то будет скомпрометирован не только текущее, но и все последующие взаимодействия между отправителем и получателем.

Pretty Good Privacy (PGP) (довольно хорошая секретность)

Одной из реализаций асимметричных криптосистем является разработанная Филиппом Циммерманном программа PGP. Эта программа отличается превосходно продуманным и чрезвычайно мощным механизмом обработки ключей. Популярность и бесплатное распространение сделали PGP фактически стандартом для электронной переписки во всем мире.

Программа PGP широко доступна в сети. В связи с ограничениями на экспорт криптографической продукции версия 5.0 запрещена к экспорту, но по старой доброй традиции была немедленно проэкспортирована из США.

В четвертой лабораторной работе вам необходимо с помощью программы PGP создать двух User-ов и обменяться сообщениями. В зашифрованном виде, в зашифрованном виде с электронной подписью.

Электронная подпись в системах с открытым ключом

Пусть пользователь N1 должен передать сообщение m пользователю N2 так, чтобы пользователь N2 в случае надобности мог доказать, что сообщение послано пользователем N1.

Для этого пользователь N1 разрабатывает систему шифрации

$$E1(D1(X)) = X$$

справедливую для любых X, которая будет использоваться для его аутентификации. Он рассылает ключ E1 как общий его электронной подписи, в том числе и пользователю N2.

Получив от пользователя N2 общий ключ шифрования E2 открытой системы

$$E2(D2(X)) = X,$$

пользователь N1 вычисляет сигнатуру сообщения

$$S = D1(m),$$

затем шифрует ее открытым ключом E2

$$C = E2(S)$$

и передает его пользователю N2, который расшифровывает сообщение процедурой

$$S = D2(C)$$
$$m^* = E1(S).$$

Теперь при возникновении спорной ситуации пользователь N1 должен представить арбитру свой личный ключ подписи D1, удовлетворяющий соотношению

$$E1(D1(X)) = X.$$

Арбитру необходимо лишь проверить, что

$$D1(m^*) = S,$$

и следовательно, сообщение мог послать только пользователь N1. Кроме того и пользователь N2 не может исказить сообщение m, т.к. для доказательства в суде ему необходимо изменить и сигнатуру S, а для этого ему нужно знать личный ключ подписи пользователя N1, а его то и нет!

О "двуличии" в алгоритмах цифровой подписи

Анализируя ту или иную схему ЭЦП, обычно ставят вопрос так: "Можно ли быстро подобрать два различных (осмысленных) сообщения, которые будут иметь одинаковые ЭЦП". Ответ здесь обычно отрицательный - трудно это сделать. Поставим вопрос по другому, а именно: "Можно ли, имея два сообщения, подобрать секретные ключи так, чтобы подписи совпадали?". Оказывается, что сделать это чрезвычайно просто!

Вот пример действий злоумышленника.

Он может:

1. Подготовить две платежки: на 10000000 руб. (**m_1**); на 3 руб. (**m_2**).
2. Выбрать секретный ключ X_1 и рассчитать ключ X_2.
3. Зарегистрировать открытые ключи, соответствующие секретным.
4. Отправить в банк требование **m_1** с подписью на X_1 (**m_1c_1**).
5. Дождаться выполнения банком поручения.
6. Предъявить банку претензию, состоящую в том, что он якобы посылал требование о переводе 3-х рублей (**m_2c_2**), а не 10000000 (**m_1c_1**), а то, что кто-то подобрал текст сообщения, не изменяющий ЭЦП - дело не его. Пусть платит банк, удостоверяющий центр, страховая компания - кто угодно, только верните мои деньги!

И главное, что придется вернуть!

В чем же источник успеха такой атаки? Дело в том, что Федеральный Закон "Об электронной цифровой подписи" допускает множественность ЭЦП для одного лица (статья 4, п.2). Именно поэтому злоумышленник получает реальную возможность подбирать не сообщение, а ключ! Более того - не исключен вариант сговора двух лиц - т.е. ЭЦП оказывается уязвимой, если секретный ключ известен хоть кому-нибудь! А если сговорятся два центра по сертификации подписей?

Тем не менее, не все так страшно.

Приведенный пример никак не дискредитирует собственно криптостойкость ЭЦП. Он показывает возможную уязвимость при неправильном применении механизмов ЭЦП. В этой связи особое внимание должно быть уделено способам построения защищенного документооборота на базе ЭЦП.

Для того чтобы лучше понять проблему, упомяну еще одну ассоциацию. Если резистор 1МОм зашунтировать резистором 1 Ом, то общее сопротивление будет порядка

1 Ом. Известно, что по открытому ключу восстановить закрытый очень сложно (1МОм). По закрытому получить открытый очень просто.

Из всего сказанного вытекают следствия:

1. В Интернете не должно быть анонимности.
2. Кроме асимметричной криптографии в электронном документообороте необходимо применять и симметричные методы.
3. Вопросы применения ЭЦП в документообороте должны быть нормативно закреплены в ФЗ "Об электронном документообороте".

Электронная цифровая подпись

1. Проблема аутентификации данных и электронная цифровая подпись

При обмене электронными документами по сети связи существенно снижаются затраты на обработку и хранение документов, убыстряется их поиск. Но при этом возникает проблема аутентификации автора документа и самого документа, т.е. установления подлинности автора и отсутствия изменений в полученном документе. В обычной (бумажной) информатике эти проблемы решаются за счет того, что информация в документе и рукописная подпись автора жестко связаны с физическим носителем (бумагой). В электронных документах на машинных носителях такой связи нет.

Целью аутентификации электронных документов является их защита от возможных видов злоумышленных действий, к которым относятся:

- ***активный перехват*** - нарушитель, подключившийся к сети, перехватывает документы (файлы) и изменяет их;
- ***маскарад*** - абонент С посылает документ абоненту В от имени абонента А;
- ***ренегатство*** - абонент А заявляет, что не посылал сообщения абоненту В, хотя на самом деле послал;
- ***подмена*** - абонент В изменяет или формирует новый документ и заявляет, что получил его от абонента А;
- ***повтор*** - абонент С повторяет ранее переданный документ, который абонент А посылал абоненту В.

Эти виды злоумышленных действий могут нанести существенный ущерб банковским и коммерческим структурам, государственным предприятиям и организациям, а также частным лицам, применяющим в своей деятельности компьютерные информационные технологии.

При обработке документов в электронной форме совершенно непригодны традиционные способы установления подлинности по рукописной подписи и оттиску печати на бумажном документе. Принципиально новым решением является электронная цифровая подпись (***ЭЦП***).

Электронная цифровая подпись используется для аутентификации текстов, передаваемых по телекоммуникационным каналам. Функционально она аналогична обычной рукописной подписи и обладает ее основными достоинствами:

- удостоверяет, что подписанный текст исходит от лица, поставившего подпись;
- не дает самому этому лицу возможности отказаться от обязательств, связанных с подписанным текстом;
- гарантирует целостность подписанного текста.

Цифровая подпись представляет собой относительно небольшое количество дополнительной цифровой информации, передаваемой вместе с подписываемым текстом.

Система ЭЦП включает две процедуры: 1) процедуру постановки подписи; 2) процедуру проверки подписи. В процедуре постановки подписи используется секретный ключ отправителя сообщения, в процедуре проверки подписи - открытый ключ отправителя.

При формировании ЭЦП отправитель прежде всего вычисляет ***хэш-функцию*** `h(M)` подписываемого текста `M`. Вычисленное значение хэш-функции `h(M)` представляет собой один короткий блок информации `m`, характеризующий весь текст `M` в целом. Затем число `m` шифруется секретным ключом отправителя. Получаемая при этом пара чисел представляет собой ЭЦП для данного текста `M`.

При проверке ЭЦП получатель сообщения снова вычисляет хэш-функцию `m = h(M)` принятого по каналу текста `M`, после чего при помощи открытого ключа отправителя проверяет, соответствует ли полученная подпись вычисленному значению `m` хэш-функции.

Принципиальным моментом в системе ***ЭЦП*** является невозможность подделки ЭЦП пользователя без знания его секретного ключа подписывания.

В качестве подписываемого документа может быть использован любой файл. Подписанный файл создается из неподписанного путем добавления в него одной или более электронных подписей.

Каждая подпись содержит следующую информацию:

- дату подписи;
- срок окончания действия ключа данной подписи;
- информацию о лице, подписавшем файл (Ф.И.0., должность, краткое наименование фирмы);
- идентификатор подписавшего (имя открытого ключа);
- собственно цифровую подпись.

2. Однонаправленные хэш-функции

Хэш-функция (англ. ***hash*** - мелко измельчать и перемешивать) предназначена для сжатия подписываемого документа до нескольких десятков или сотен бит. Хэш-функция `h(·)` принимает в качестве аргумента сообщение (документ) `M` произвольной длины и возвращает хэш-значение `h(M)=H` фиксированной длины. Обычно хэшированная информация является сжатым двоичным представлением основного сообщения произвольной длины. Следует отметить, что значение хэш-функции `h(M)` сложным образом зависит от документа `M` и не позволяет восстановить сам документ `M`.

Хэш-функция должна удовлетворять целому ряду условий:

1. хэш-функция должна быть чувствительна к всевозможным изменениям в тексте `M`, таким как вставки, выбросы, перестановки и т.п.;
2. хэш-функция должна обладать свойством необратимости, то есть задача подбора документа `M'`, который обладал бы требуемым значением хэш-функции, должна быть вычислительно неразрешима;

3. вероятность того, что значения хэш-функций двух различных документов (вне зависимости от их длин) совпадут, должна быть ничтожно мала.

Большинство хэш-функций строится на основе однонаправленной функции $f(\cdot)$, которая образует выходное значение длиной n при задании двух входных значений длиной n. Этими входами являются блок исходного текста M, и хэш-значение H_{i-1} предыдущего блока текста (рис.1).

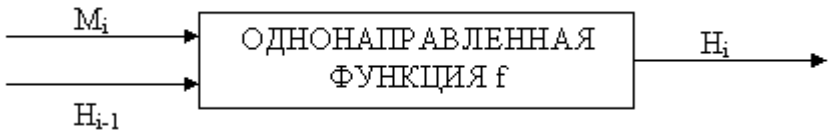


Рис.1. Построение однонаправленной хэш-функции

$$H_i = f(M_i, H_{i-1}) .$$

Хэш-значение, вычисляемое при вводе последнего блока текста, становится хэш-значением всего сообщения M.

В результате однонаправленная хэш-функция всегда формирует выход фиксированной длины n (независимо от длины входного текста).

Основы построения хэш-функций

Общепринятым принципом построения хэш-функций является ***итеративная последовательная схема***. По этой методики ядром алгоритма является преобразование k бит в n бит. Величина n - разрядность результата хэш-функции, а k - произвольное число, большее n. Базовое преобразование должно обладать всеми свойствами хэш-функции т.е. необратимостью и невозможностью инвариантного изменения входных данных.

Хэширование производится с помощью промежуточной вспомогательной переменной разрядностью в n бит. В качестве ее начального значения выбирается произвольное известное всем сторонам значение, например, 0.

Входные данные разбиваются на блоки по $(k-n)$ бит. На каждой итерации хэширования со значением промежуточной величины, полученной на предыдущей итерации, объединяется очередная $(k-n)$-битная порция входных данных, и над получившимся k-битным блоком производится базовое преобразование. В результате весь входной текст оказывается "перемешанным" с начальным значением вспомогательной величины. Из-за характера преобразования базовую функцию часто называют ***сжимающей***. Значение вспомогательной величины после финальной итерации поступает на выход хэш-функции (рис.2). Иногда над получившимся значением производят дополнительные преобразования. Но в том случае, если сжимающая функция спроектирована с достаточной степенью стойкости, эти преобразования излишни.

При проектировании хэш-функции по итеративной схеме возникают два взаимосвязанных вопроса: как поступать с данными, не кратными числу $(k-n)$, и как добавлять в хэш-сумму длину документа, если это требуется. Есть два варианта решения этих вопросов. В первом варианте в начало документа перед хэшированием добавляется поле фиксированной длины (например, 32 бита), в котором в двоичном виде записывается

исходная длина текста. Затем объединенный блок данных дополняется нулями до ближайшего кратного (k-n) бит размера. Во втором варианте документ дополняется справа одним битом "1", а затем до кратного (k-n) бит размера битами "0". В этом варианте необходимость в поле длины отпадает - никакие два разных документа после выравнивания по границе порций не станут одинаковыми.

Кроме более популярных однопроходных алгоритмов хэширования существуют и многопроходные алгоритмы. В этом случае входной блок данных на этапе расширения неоднократно повторяется, а уже затем дополняется до ближайшей границы порции.

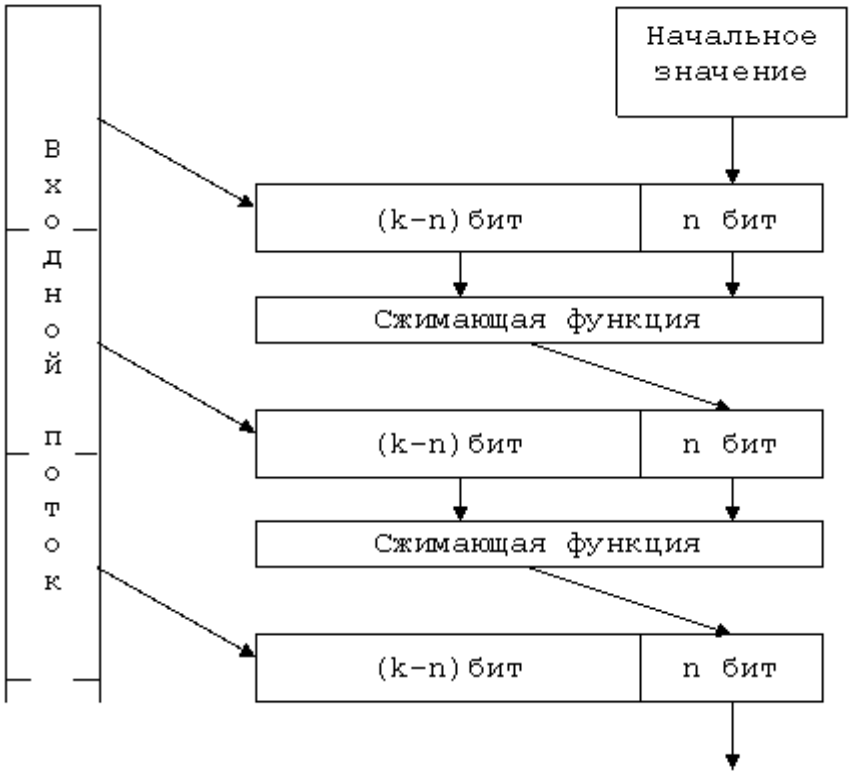


Рис.2. Итерактивная хэш-функция

Однонаправленные хэш-функции на основе симметричных блочных алгоритмов

Однонаправленную хэш-функцию можно построить, используя симметричный блочный алгоритм. Наиболее очевидный подход состоит в том, чтобы шифровать сообщение M посредством блочного алгоритма в режиме CBC или CFB с помощью фиксированного ключа и некоторого вектора инициализации IV. Последний блок шифртекста можно рассматривать в качестве хэш-значения сообщения M. При таком подходе не всегда возможно построить безопасную однонаправленную хэш-функцию, но всегда можно получить код аутентификации сообщения MAC (Message Authentication Code).

Более безопасный вариант хэш-функции можно получить, используя блок сообщения в качестве ключа, предыдущее хэш-значение - в качестве входа, а текущее хэш-значение - в качестве выхода. Реальные хэш-функции проектируются еще более сложными. Длина блока обычно определяется длиной ключа, а длина хэш-значения совпадает с длиной блока.

Поскольку большинство блочных алгоритмов являются 64-битовыми, некоторые схемы хэширования проектируют так, чтобы хэш-значение имело длину, равную двойной длине блока.

Если принять, что получаемая хэш-функция корректна, безопасность схемы хэширования базируется на безопасности лежащего в ее основе блочного алгоритма. Схема хэширования, у которой длина хэш-значения равна длине блока, показана на рис.3. Ее работа описывается выражениями:

$$H_0 = I_н,$$
$$H_i = E_A(B) \oplus C,$$

где $\oplus$ - сложение по модулю 2 (исключающее ИЛИ); $I_н$ - некоторое случайное начальное значение; A, B, C могут принимать значения M_i, H_{i-1}, $(M_i \oplus H_{i-1})$ или быть константами.

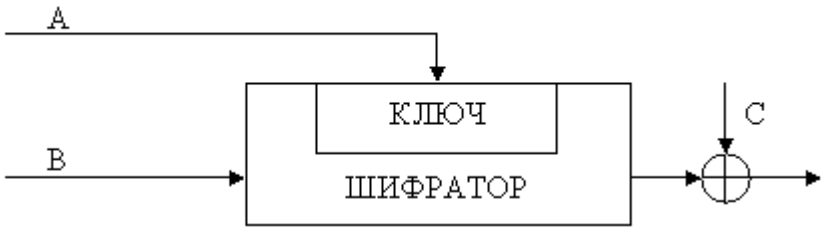


Рис.3. Обобщенная схема формирования хэш-функции

Сообщение M разбивается на блоки M_i принятой длины, которые обрабатываются поочередно.

Три различные переменные A, B, C могут принимать одно из четырех возможных значений, поэтому в принципе можно получить 64 варианта общей схемы этого типа. Из них 52 варианта являются либо тривиально слабыми, либо небезопасными. Остальные 12 схем безопасного хэширования, у которых длина хэш-значения равна длине блока перечислены в табл.1.

Таблица 1

Номер схемы	Функция хэширования
1	$H_i = E_{Hi-1}(M_i) \oplus M_i$
2	$H_i = E_{Hi-1}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
3	$H_i = E_{Hi-1}(M_i) \oplus M_i \oplus H_{i-1}$
4	$H_i = E_{Hi-1}(M_i \oplus H_{i-1}) \oplus M_i$
5	$H_i = E_{Mi}(H_{i-1}) \oplus H_{i-1}$
6	$H_i = E_{Mi}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
7	$H_i = E_{Mi}(H_{i-1}) \oplus M_i \oplus H_{i-1}$
8	$H_i = E_{Mi}(M_i \oplus H_{i-1}) \oplus H_{i-1}$
9	$H_i = E_{Mi \oplus Hi-1}(M_i) \oplus M_i$

10	$H_i = E_{Mi \oplus Hi-1}(H_{i-1}) \oplus H_{i-1}$
11	$H_i = E_{Mi \oplus Hi-1}(M_i) \oplus H_{i-1}$
12	$H_i = E_{Mi \oplus Hi-1}(H_{i-1}) \oplus M_i$

Первые четыре схемы хэширования, являющиеся безопасными при всех атаках, приведены на рис.4.

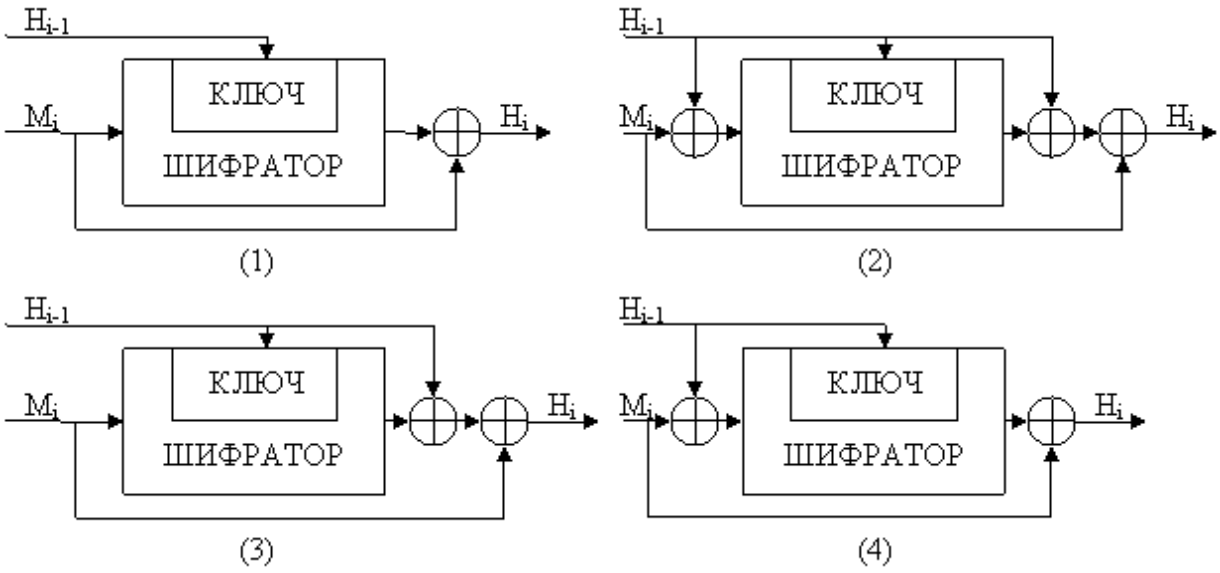


Рис.4. Четыре схемы безопасного хэширования

Недостатком хэш-функций, спроектированных на основе блочных алгоритмов, является несколько заниженная скорость работы. Дело в том, что ту же самую стойкость относительно двух основных требований к хэш-функции можно обеспечить за гораздо меньшее количество операций над входными данными. Но для этого алгоритм необходимо изначально проектировать специально, исходя из тандема требований (стойкость, скорость). Далее рассмотрены три самостоятельных алгоритма криптостойкого хэширования, получивших наибольшее распространение на сегодняшний день.

Алгоритм MD5

Алгоритм ***MD5*** (Message Digest №5) разработан Роналдом Риверсом. MD5 использует 4 многократно повторяющиеся преобразования над тремя 32-битными величинами U, V и W:

```
f(U,V,W)=(U AND V) OR ((NOT U) AND W)
g(U,V,W)=(U AND W) OR (V AND (NOT W))
h(U,V,W)=U XOR V XOR W
k(U,V,W)=V XOR (U OR (NOT W)).
```

В алгоритме используются следующие константы:

- начальные константы промежуточных величин -

```
H[0]=67452301₁₆,        H[1]=EFCDAB89₁₆,        H[2]=98BADCFE₁₆,
H[3]=10325476₁₆;
```

- константы сложения в раундах -

```
y[j]=HIGHEST_32_BITS(ABS(SIN(j+1))) j=0...63,
```

где функция `HIGHEST_32_BITS(X)` отделяет 32 самых старших бита из двоичной записи дробного числа `X`, а операнд `SIN(j+1)` считается взятым в радианах;

- массив порядка выбора ячеек в раундах -

```
z[0...63] = (0, 1,  2, 3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
13, 14, 15,
            1, 6, 11, 0,  5, 10, 15,  4,  9, 14,  3,  8, 13,
2,  7, 12,
            5, 8, 11, 4,  1,  4,  7, 10, 13,  0,  3,  6,  9,
12, 15,  2,
             0, 7, 14, 5, 12,  3, 10,  1,  8, 15,  6, 13,  4,
11,  2,  9);
```

- массив величины битовых циклических сдвигов влево -

```
s[0...63] = (7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7,
12, 17, 22,
            5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,
9, 14, 20,
            4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4,
11, 16, 23,
             6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6,
10, 15, 21).
```

На первоначальном этапе входной блок данных дополняется одним битом "1". Затем к нему добавляется такое количество битов "0", чтобы остаток от деления блока на 512 составлял 448. Наконец, к блоку добавляется 64-битная величина, хранящая первоначальную длину документа. Получившийся входной поток имеет длину кратную 512 битам.

Каждый 512-битный блок, представленный в виде 16 32-битных значений `X[0]...X[15]`, проходит через сжимающую функцию, которая перемешивает его со вспомогательным блоком `(H[0],H[1],H[2],H[3])`:

```
(A,B,C,D) = (H[0],H[1],H[2],H[3])
цикл по j от 0 до 15
       T = (A + f(B,C,D) + x[z[j]] + y[j]) ROL s[j]
       (A,B,C,D) = (D,B+T,B,C)
конец_цикла
цикл по j от 16 до 31
       T = (A + g(B,C,D) + x[z[j]] + y[j]) ROL s[j]
       (A,B,C,D) = (D,B+T,B,C)
конец_цикла
цикл по j от 32 до 47
       T = (A + h(B,C,D) + x[z[j]] + y[j]) ROL s[j]
       (A,B,C,D) = (D,B+T,B,C)
конец_цикла
```

```
цикл по j от 48 до 63
       T = (A + k(B,C,D) + x[z[j]] + y[j]) ROL s[j]
       (A,B,C,D) = (D,B+T,B,C)
конец_цикла
(H[0],H[1],H[2],H[3]) = (H[0]+A,H[1]+B,H[2]+C,H[3]+D)
```

После того, как все 512-битные блоки прошли через процедуру перемешивания, временные переменные `H[0],H[1],H[2],H[3]`, а 128-битное значение подается на выход хэш-функции.

Алгоритм MD5, основанный на предыдущей разработке Роналда Риверса MD4, был призван дать еще больший запас прочности к криптоатакам. MD5 очень похож на MD4. Отличие состоит в простейших изменениях в алгоритмах наложения и в том, что в MD4 48 проходов основного преобразования, а в MD5 - 64. Несмотря на большую популярность, MD4 "медленно, но верно" был взломан. Сначала появились публикации об атаках на упрощенный алгоритм. Затем было заявлено о возможности найти два входных блока сжимающей функции MD4, которые порождают одинаковый выход. Наконец, в 1995 году было показано, что найти коллизию, т.е. "хэш-двойник" к произвольному документу, можно менее чем за минуту, а добиться "осмысленности" фальшивого документа (т.е. наличия в нем только ASCII-символов с определенными "разумными" законами расположения) - всего лишь за несколько дней.

Алгоритм безопасного хэширования SHA

Алгоритм безопасного хэширования ***SHA*** (***Secure Hash Algorithm***) разработан НИСТ и АНБ США в рамках стандарта безопасного хэширования SHS (Secure Hash Standard) в 1992 г. Алгоритм хэширования SHA предназначен для использования совместно с алгоритмом цифровой подписи DSA.

При вводе сообщения M произвольной длины менее 2^{64} бит алгоритм SHA вырабатывает 160-битовое выходное сообщение, называемое дайджестом сообщения MD (Message Digest). Затем этот дайджест сообщения используется в качестве входа алгоритма DSA, который вычисляет цифровую подпись сообщения M. Формирование цифровой подписи для дайджеста сообщения, а не для самого сообщения повышает эффективность процесса подписания, поскольку дайджест сообщения обычно намного короче самого сообщения.

Такой же дайджест сообщения должен вычисляться пользователем, проверяющим полученную подпись, при этом в качестве входа в алгоритм SHA используется полученное сообщение M.

Алгоритм хэширования SHA назван безопасным, потому что он спроектирован таким образом, чтобы было вычислительно невозможно восстановить сообщение, соответствующее данному дайджесту, а также найти два различных сообщения, которые дадут одинаковый дайджест. Любое изменение сообщения при передаче с очень большой вероятностью вызовет изменение дайджеста, и принятая цифровая подпись не пройдет проверку.

Рассмотрим подробнее работу алгоритма хэширования SHA. Прежде всего исходное сообщение M дополняют так, чтобы оно стало кратным 512 битам. Дополнительная набивка сообщения выполняется следующим образом: сначала добавляется единица, затем следуют столько нулей, сколько необходимо для получения сообщения, которое на

64 бита короче, чем кратное 512, и наконец добавляют 64-битовое представление длины исходного сообщения.

Инициализируется пять 32-битовых переменных в виде:

```
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
E = 0xC3D2E1F0
```

Затем начинается главный цикл алгоритма. В нем обрабатывается по 512 бит сообщения поочередно для всех 512-битовых блоков, имеющихся в сообщении. Первые пять переменных A, B, C, D, E копируются в другие переменные a, b, c, d, e:

```
a = A, b = B, c = C, d = D, e = E
```

Главный цикл содержит четыре цикла по 20 операций каждый. Каждая операция реализует нелинейную функцию от трех из пяти переменных a, b, c, d, e, а затем производит сдвиг и сложение.

Алгоритм SHA имеет следующий набор нелинейных функций:

$f_t(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ для t = 0...19,
$f_t(X, Y, Z) = X \oplus Y \oplus Z$ для t = 20...39,
$f_t(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$ для t = 40...59,
$f_t(X, Y, Z) = X \oplus Y \oplus Z$ для t = 60...79,

где t - номер операции.

В алгоритме используются также четыре константы:

K_t = 0x5A827999 для t = 0...19,
K_t = 0x6ED9EBA1 для t = 20...39,
K_t = 0x8F1BBCDC для t = 40...59,
K_t = 0xCA62C1D6 для t = 60...79.

Блок сообщения преобразуется из шестнадцати 32-битовых слов ($M_0 \ldots M_{15}$) в восемьдесят 32-битовых слов ($W_0 \ldots W_{79}$) с помощью следующего алгоритма:

$W_t = M_t$ для t = 0...15,
$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1$ для t = 16...79,

где t - номер операции, W_t - t-й субблок расширенного сообщения, <<< S - циклический сдвиг влево на S бит.

С учетом введенных обозначений главный цикл из восьмидесяти операций можно описать так:

```
цикл по t от 0 до 79
       TEMP = (a <<< 5) + f_t (b, c, d) + e + W_t + K_t
       e = d
       d = c
       c = (b <<< 30)
```

```
        b = a
        a = TEMP
конец_цикла
```

Схема выполнения одной операции показана на рис.5.

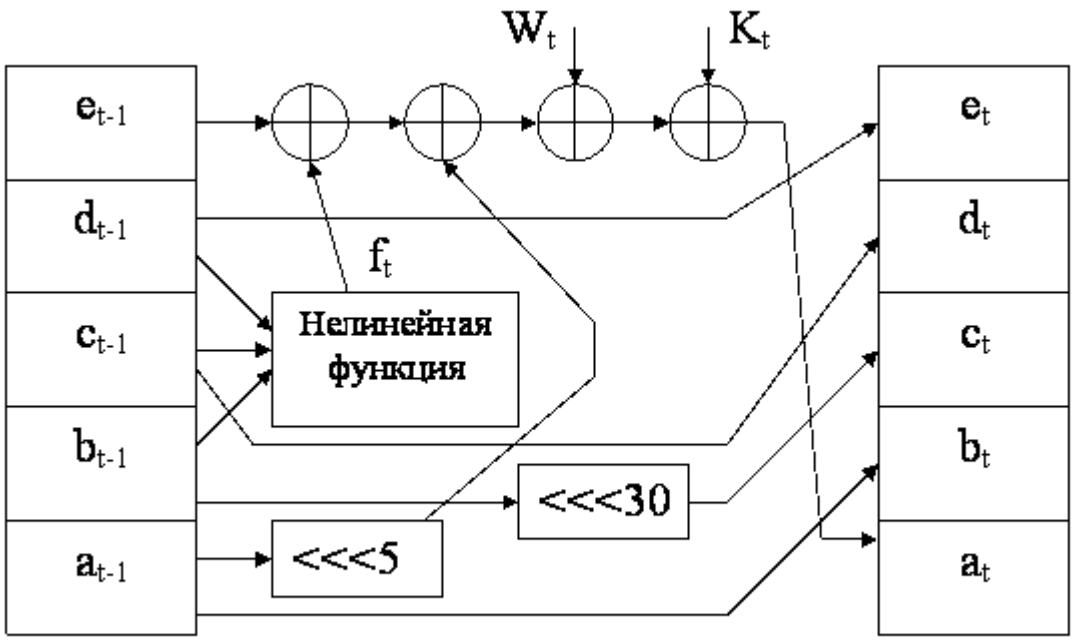


Рис.5. Схема выполнения одной операции алгоритма SHA

После окончания главного цикла значения `a, b, c, d, e` складываются с `A, B, C, D, E` соответственно, и алгоритм приступает к обработке следующего 512-битового блока данных. Окончательный выход формируется в виде конкатенации значений `A, B, C, D, E`.

Отличия SHA от MD5 состоят в следующем:

- SHA выдает 160-битовое хэш-значение, поэтому он более устойчив к атакам полного перебора и атакам "дня рождения", чем MD5, формирующий 128-битовые хэш-значения.
- Сжимающая функция SHA состоит из 80 шагов, а не из 64 как в MD5.
- Расширение входных данных производится не простым их повторение в другом порядке, а рекуррентной формулой.
- Усложнен процесс перемешивания

Отечественный стандарт хэш-функции

Российский стандарт ***ГОСТ Р 34.11-94*** определяет алгоритм и процедуру вычисления хэш-функции для любых последовательностей двоичных символов, применяемых в криптографических методах обработки и защиты информации. Этот стандарт базируется на блочном алгоритме шифрования ГОСТ 28147-89, хотя в принципе можно было бы использовать и другои блочный алгоритм шифрования с 64-битовым блоком и 256-битовым ключом.

Данная хэш-функция формирует 256-битовое хэш-значение.

Функция сжатия $H_i = f(M_i, H_{i-1})$ (оба операнда M_i и H_{i-1} являются 256-битовыми величинами) определяется следующим образом:

1. Генерируются 4 ключа шифрования K_j , $j = 1...4$, путем линейного смешивания M_i , H_{i-1} и некоторых констант C_j .
2. Каждый ключ K_j используют для шифрования 64-битовых подслов h_j слова H_{i-1} в режиме простой замены: $S_i = E_{Kj}(h_j)$. Результирующая последовательность S_4, S_3, S_2, S_1 длиной 256 бит запоминается во временной переменной S.
3. Значение H_i является сложной, хотя и линейной функцией смешивания S, M_i, H_{i-1}.

При вычислении окончательного хэш-значения сообщения M учитываются значения трех связанных между собой переменных:

H_n - хэш-значение последнего блока сообщения;
Z - значение контрольной суммы, получаемой при сложении по модулю 2 всех блоков сообщения;
L - длина сообщения.

Эти три переменные и дополненный последний блок M' сообщения объединяются в окончательное хэш-значение следующим образом:

$$H = f (Z \oplus M', f (L, f(M', H_n))).$$

Данная хэш-функция определена стандартом ГОСТ Р 34.11-94 для использования совместно с российским стандартом электронной цифровой подписи.

3. Алгоритмы электронной цифровой подписи

Технология применения системы ЭЦП предполагает наличие сети абонентов, посылающих друг другу подписанные электронные документы. Для каждого абонента генерируется пара ключей: секретный и открытый. Секретный ключ хранится абонентом в тайне и используется им для формирования ЭЦП. Открытый ключ известен всем другим пользователям и предназначен для проверки ЭЦП получателем подписанного электронного документа. Иначе говоря, открытый ключ является необходимым инструментом, позволяющим проверить подлинность электронного документа и автора подписи. Открытый ключ не позволяет вычислить секретный ключ.

Для генерации пары ключей (секретного и открытого) в алгоритмах ЭЦП, как и в асимметричных системах шифрования, используются разные математические схемы, основанные на применении однонаправленных функции. Эти схемы разделяются на две группы. В основе такого разделения лежат известные сложные вычислительные задачи:

- задача факторизации (разложения на множители) больших целых чисел;
- задача дискретного логарифмирования.

Алгоритм цифровой подписи RSA

Первой и наиболее известной во всем мире конкретной системой ЭЦП стала система ***RSA***, математическая схема которой была разработана в 1977 г. в Массачуссетском технологическом институте США.

Сначала необходимо вычислить пару ключей (секретный ключ и открытый ключ). Для этого отправитель (автор) электронных документов вычисляет два больших простых числа P и Q, затем находит их произведение

```
N = P * Q
```

и значение функции

```
φ (N) = (P-1)(Q-1).
```

Далее отправитель вычисляет число E из условий:

```
E ≤φ (N), НОД (E, φ (N)) = 1
```

и число D из условий:

```
D < N, E*D ≡ 1 (mod φ (N)).
```

Пара чисел (E, N) является открытым ключом. Эту пару чисел автор передает партнерам по переписке для проверки его цифровых подписей. Число D сохраняется автором как секретный ключ для подписывания.

Обобщенная схема формирования и проверки цифровой подписи RSA показана на рис.6.

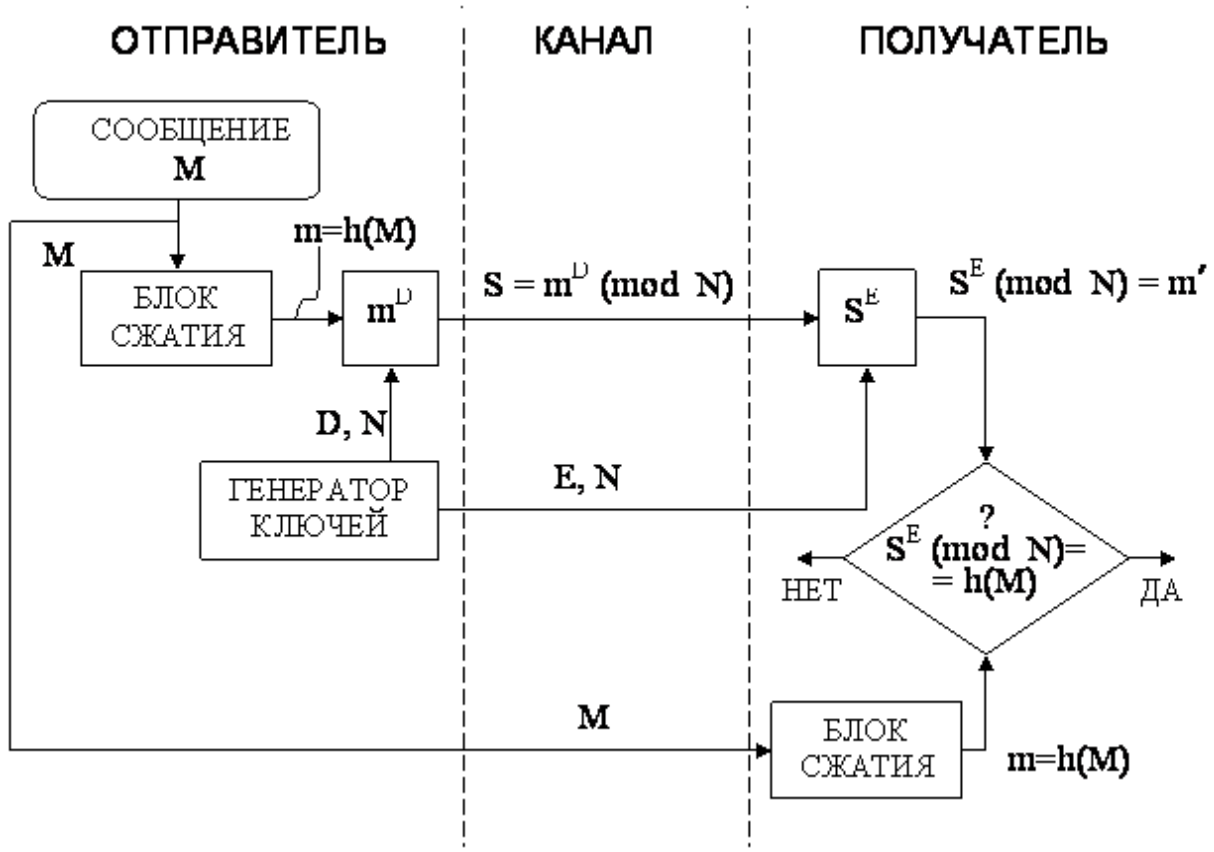


Рис.6. Обобщённая схема цифровой подписи RSA

Допустим, что отправитель хочет подписать сообщение M перед его отправкой. Сначала сообщение M (блок информации, файл, таблица) сжимают с помощью хэш-функции h(·) в целое число m:

$$m = h(M).$$

Затем вычисляют цифровую подпись S под электронным документом M, используя хэш-значение m и секретный ключ D:

$$S = m^D \pmod N.$$

Пара (M,S) передается партнеру-получателю как электронный документ M, подписанный цифровой подписью S, причем подпись S сформирована обладателем секретного ключа D.

После приема пары (M,S) получатель вычисляет хэш-значение сообидения M двумя разными способами. Прежде всего он восстанавливает хэш-значение m', применяя криптографическое преобразование подписи S с использованием открытого ключа E:

$$m' = S^E \pmod N.$$

Кроме того, он находит результат хэширования принятого сообщения M с помощью такой же хэш-функции h(·):

$$m = h(M).$$

Если соблюдается равенство вычисленных значений, т.е.

$$S^E \ (\mathrm{mod}\ N) = h\ (M),$$

то получатель признает пару (M,S) подлинной. Доказано, что только обладатель секретного ключа D может сформировать цифровую подпись S по документу M, а определить секретное число D по открытому числу E не легче, чем разложить модуль N на множители.

Кроме того, можно строго математически доказать, что результат проверки цифровой подписи S будет положительным только в том случае, если при вычислении S был использован секретный ключ D, соответствующий открытому ключу E. Поэтому открытый ключ E иногда называют "идентификатором" подписавшего.

Недостатки алгоритма цифровой подписи RSA.

1. При вычислении модуля N, ключей E и D для системы цифровой подписи RSA необходимо проверять большое количество дополнительных условий, что сделать практически трудно. Невыполнение любого из этих условий делает возможным фальсификацию цифровой подписи со стороны того, кто обнаружит такое невыполнение. При подписании важных документов нельзя допускать такую возможность даже теоретически.
2. Для обеспечения криптостойкости цифровой подписи RSA по отношению к попыткам фальсификации на уровне, например, национального стандарта США на шифрование информации (алгоритм DES), т.е. 10^{18}, необходимо использовать при вычислениях N, D и E целые числа не менее 2^{512} (или около 10^{154}) каждое, что требует больших вычислительных затрат, превышающих на 20...30% вычислительные затраты других алгоритмов цифровой подписи при сохранении того же уровня криптостойкости.
3. Цифровая подпись RSA уязвима к так называемой мультипликативной атаке. Иначе говоря, алгоритм цифровой подписи RSA позволяет злоумышленнику без знания секретного кпюча D сформировать подписи под теми документами, у которых результат хэширования можно вычислить как произведение результатов хэширования уже подписанных документов.

Пример. Допустим, что злоумышленник может сконструировать три сообщения M_1, M_2, M_3, у которых хэш-значения

$$m_1 = h\ (M_1),\ m_2 = h\ (M_2),\ m_3 = h\ (M_3)\ ,$$

причем

$$m_3 = m_1 * m_2\ (\mathrm{mod}\ N)\ .$$

Допустим также, что для двух сообщений M_1 и M_2 получены законные подписи

$$S_1 = m_1^D\ (\mathrm{mod}\ N)$$
$$S_2 = m_2^D\ (\mathrm{mod}\ N)\ .$$

Тогда злоумышленник может легко вычислить подпись S_3 для документа M_3, даже не зная секретного ключа D:

$$S_3 = S_1 * S_2 \pmod N.$$

Действительно,

$$S_1 * S_2 \pmod N = m_1^D * m_2^D \pmod N = (m_1 m_2)^D \pmod N = m_3^D \pmod N = S_3 .$$

Более надежный и удобный для реализации на персональных компьютерах алгоритм цифровой подписи был разработан в 1984 г. американцем арабского происхождения Тахером Эль Гамалем. В 1991 г. НИСТ США обосновал перед комиссией Конгресса США выбор алгоритма цифровой подписи Эль Гамаля в качестве основы для национального стандарта.

Алгоритм цифровой подписи Эль Гамаля (EGSA)

Название EGSA происходит от слов E_ Gama_ Signature Algorithm (алгоритм цифровой подписи Эль Гамаля). Идея EGSA основана на том, что для обоснования практической невозможности фальсификации цифровой подписи может быть использована более сложная вычислительная задача, чем разложение на множители большого целого числа,- задача дискретного логарифмирования. Кроме того, Эль Гамалю удалось избежать явной слабости алгоритма цифровой подписи RSA, связанной с возможностью подделки цифровой подписи под некоторыми сообщениями без определения секретного ключа.

Рассмотрим подробнее алгоритм цифровой подписи ***Эль-Гамаля***. Для того чтобы генерировать пару ключей (открытый ключ - секретный ключ), сначала выбирают некоторое большое простое целое число P и большое целое число G, причем $G < P$. Отправитель и получатель подписанного документа используют при вычислениях одинаковые большие целые числа P (~10^{308} или ~2^{1024}) и G (~10^{154} или ~2^{512}), которые не являются секретными.

Отправитель выбирает случайное целое число X, $1 < X \le (P-1)$, и вычисляет

$$Y = G^X \bmod P .$$

Число Y является открытым ключом, используемым для проверки подписи отправителя. Число Y открыто передается всем потенциальным получателям документов.

Число X является секретным ключом отправителя для подписывания документов и должно храниться в секрете.

Для того чтобы подписать сообщение M, сначала отправитель хэширует его с помощью хэш-функции $h(\cdot)$ в целое число m:

$$m = h(M),\ 1 < m < (P-1) ,$$

и генерирует случайное целое число K, $1 < K < (P-1)$, такое, что K и $(P-1)$ являются взаимно простыми. Затем отправитель вычисляет целое число a:

$$a = G^K \bmod P$$

и, применяя расширенный алгоритм Евклида, вычисляет с помощью секретного ключа X целое число b из уравнения

$$m = X * a + K * b \pmod{(P-1)} .$$

Пара чисел (a,b) образует цифровую подпись S:

$$S=(a,b)\ ,$$

проставляемую под документом M.

Тройка чисел (M,a,b) передается получателю, в то время как пара чисел (X,K) держится в секрете.

После приема подписанного сообщения (M,a,b) получатель должен проверить, соответствует ли подпись S=(a,b) сообщению M. Для этого получатель сначала вычисляет по принятому сообщению M число

$$m = h(M)\ ,$$

т.е. хэширует принятое сообщение M.

Затем получатель вычисляет значение

$$A = Y^a\ a^b\ (\mathrm{mod}\ P)$$

и признает сообщение M подлинным, только если

$$A = G^m\ (\mathrm{mod}\ P)\ .$$

Иначе говоря, получатель проверяет справедливость соотношения

$$Y^a\ a^b\ (\mathrm{mod}\ P) = G^m\ (\mathrm{mod}\ P)\ .$$

Можно строго математически доказать, что последнее равенство будет выполняться тогда, и только тогда, когда подпись S=(a,b) под документом M получена с помощью именно того секретного ключа X, из которого был получен открытый ключ Y. Таким образом, можно надежно удостовериться, что отправителем сообщения M был обладатель именно данного секретного ключа X, не раскрывая при этом сам ключ, и что отправитель подписал именно этот конкретный документ M.

Следует отметить, что выполнение каждой подписи по методу Эль Гамаля требует нового значения K, причем это значение должно выбираться случайным образом. Если нарушитель раскроет когда-либо значение K, повторно используемое отправителем, то он сможет раскрыть секретный ключ X отправителя.

Пример. Выберем: числа $P = 11$, $G = 2$ и секретный ключ $X = 8$. Вычисляем значение открытого ключа:

$$Y = G^X \bmod P = 2^8 \bmod 11 = 3\ .$$

Предположим, что исходное сообщение M характеризуется хэш-значением $m = 5$.

Для того чтобы вычислить цифровую подпись для сообщения M, имеющего хэш-значение $m = 5$, сначала выберем случайное целое число $K = 9$. Убедимся, что числа K и (P-1) являются взаимно простыми. Действительно, НОД (9,10) = 1. Далее вычисляем элементы a и b подписи:

$$a = G^K \bmod P = 2^9 \bmod 11 = 6\ ,$$

элемент b определяем, используя расширенный алгоритм Евклида:

$$m = X * a + K * b \ (mod(P-1)).$$

При $m = 5, a = 6, X = 8, K = 9, P = 11$ получаем

$$5 = 8 * 6 + 9 * b \ (mod\ 10)$$

или

$$9 * b = -43 \ (mod\ 10) .$$

Решение: $b = 3$. Цифровая подпись представляет собой пару: $a = 6, b = 3$. Далее отправитель передает подписанное сообщение. Приняв подписанное сообщение и открытый ключ $Y = 3$, получатель вычисляет хэш-значение для сообщения $M: m = 5$, а затем вычисляет два числа:

$$Y^a a^b \ (mod\ P) = 3^6 * 6^3 \ (mod\ 11) = 10 \ (mod\ 11);$$
$$G^m \ (mod\ P) = 2^5 \ (mod\ 11) = 10 \ (mod\ 11).$$

Так как эти два целых числа равны, принятое получателем сообщение признается подлинным.

Следует отметить, что схема Эль Гамаля является характерным примером подхода, который допускает пересылку сообщения M в открытой форме вместе с присоединенным аутентификатором (a,b). В таких случаях процедура установления подлинности принятого сообщения состоит в проверке соответствия аутентификатора сообщению.

Схема цифровой подписи Эль Гамаля имеет ряд преимуществ по сравнению со схемой цифровой подписи RSA:

1. При заданном уровне стойкости алгоритма цифровой подписи целые числа, участвующие в вычислениях, имеют запись на 25% короче, что уменьшает сложность вычислений почти в два раза и позволяет заметно сократить объем используемой памяти.
2. При выборе модуля P достаточно проверить, что это число является простым и что у числа $(P-1)$ имеется большой простой множитель (т.е. всего два достаточно просто проверяемых условия).
3. Процедура формирования подписи по схеме Эль Гамаля не позволяет вычислять цифровые подписи под новыми сообщениями без знания секретного ключа (как в RSA).

Однако алгоритм цифровой подписи Эль Гамаля имеет и некоторые недостатки по сравнению со схемой подписи RSA. В частности, длина цифровой подписи получается в 1,5 раза больше, что, в свою очередь, увеличивает время ее вычисления.

Алгоритм цифровой подписи DSA

Алгоритм цифровой подписи ***DSA*** (***Digital Signature Algorithm***) предложен в 1991 г. в НИСТ США для использования в стандарте цифровой подписи DSS (Digital Signature Standard). Алгоритм DSA является развитием алгоритмов цифровой подписи Эль Гамаля и К.Шнорра.

Отправитель и получатель электронного документа используют при вычислении большие целые числа: G и P - простые числа, L бит каждое ($512 \le L \le 1024$); q -

простое число длиной 160 бит (делитель числа (P-1)). Числа G, P, q являются открытыми и могут быть общими для всех пользователей сети.

Отправитель выбирает случайное целое число X, 1 < X < q. Число X является секретным ключом отправителя для формирования электронной цифровой подписи.

Затем отправитель вычисляет значение

$$Y = G^X \bmod P.$$

Число Y является открытым ключом для проверки подписи отправителя и передается всем получателям документов.

Этот алгоритм также предусматривает использование односторонней функции хэширования h(·). В стандарте DSS определен алгоритм безопасного хэширования SHA (Secure Hash Algorithm).

Для того чтобы подписать документ M, отправитель хэширует его в целое хэш-значение m:

$$m = h(M),\ 1<m<q\ ,$$

затем генерирует случайное целое число K, 1< K< q, и вычисляет число r:

$$r = (G^K \bmod P) \bmod q\ .$$

Затем отправитель вычисляет с помощью секретного ключа X целое число s:

$$s = ((m + r * X)/K) \bmod q\ .$$

Пара чисел (r,s) образует цифровую подпись

$$S = (r,s)$$

под документом M.

Таким образом, подписанное сообщение представляет собой тройку чисел (M,r,s).

Получатель подписанного сообщения (M,r,s) проверяет выполнение условий

$$0 < r < q,\ 0 < s < q$$

и отвергает подпись, если хотя бы одно из этих условий не выполнено. Затем получатель вычисляет значение

$$w = (1/s) \bmod q\ ,$$

хэш-значение

$$m = h(M)$$

и числа

$$u_1 = (m * w) \bmod q\ ,$$
$$u_2 = (r * w) \bmod q\ .$$

Далее получатель с помощью открытого ключа Y вычисляет значение

$$v = ((G^{u_1} * Y^{u_2}) \bmod P) \bmod q$$

и проверяет выполнение условия

$$v = r\ .$$

Если условие v = r выполняется, тогда подпись S=(r,s) под документом M признается получателем подлинной.

Можно строго математически доказать, что последнее равенство будет выполняться тогда, и только тогда, когда подпись S=(r,s) под документом M получена с помощью именно того секретного ключа X, из которого был получен открытый ключ Y. Таким образом, можно надежно удостовериться, что отправитель сообщения владеет именно данным секретным ключом X (не раскрывая при этом значения ключа X) и что отправитель подписал именно данный документ M.

По сравнению с алгоритмом цифровой подписи Эль Гамаля алгоритм DSA имеет следующие основные преимущества:

1. При любом допустимом уровне стойкости, т.е. при любой паре чисел G и P (от 512 до 1024 бит), числа q, X, r, s имеют длину по 160 бит, сокращая длину подписи до 320 бит.
2. Большинство операций с числами K, r, s, X при вычислении подписи производится по модулю числа q длиной 160 бит, что сокращает время вычисления подписи.
3. При проверке подписи большинство операций с числами u_1, u_2, v, w также производится по модулю числа q длиной 160 бит, что сокращает объем памяти и время вычисления.

Недостатком алгоритма DSA является то, что при подписывании и при проверке подписи приходится выполнять сложные операции деления по модулю q:

$$s = ((m + rX)/K) \pmod q, \quad w = (1/s) \pmod q ,$$

что не позволяет получать максимальное быстродействие.

Следует отметить, что реальное исполнение алгоритма DSA может быть ускорено с помощью выполнения предварительных вычислений. Заметим, что значение r не зависит от сообщения M и его хэш-значения m. Можно заранее создать строку случайных значений K и затем для каждого из этих значений вычислить значения r. Можно также заранее вычислить обратные значения K^{-1} для каждого из значений K. Затем, при поступлении сообщения M, можно вычислить значение s для данных значений r и K^{-1}. Эти предварительные вычисления значительно ускоряют работу алгоритма DSA.

Отечественный стандарт цифровой подписи

Отечественный стандарт цифровой подписи обозначается как ***ГОСТ Р 34.10-94. Алгоритм цифровой подписи, определяемый этим стандартом, концептуально близок к алгоритму DSA. В нем используются следующие параметры:***

p - большое простое число длиной от 509 до 512 бит либо от 1020 до 1024 бит;
q - простой сомножитель числа (p-1), имеющий длину 254...256 бит;
a - любое число, меньшее (p-1), причем такое, что $a^q \bmod p = 1$;
x - некоторое число, меньшее q;
$y = a^x \bmod p$.

Кроме того, этот алгоритм использует однонаправленную хэш-функцию $H(x)$. *Стандарт ГОСТ Р 34.11-94 определяет хэш-функцию, основанную на использовании стандартного симметричного алгоритма ГОСТ 28147-89.*

Первые три параметра p, q, a *являются открытыми и могут быть общими для всех пользователей сети. Число* x *является секретным ключом. Число* y *является открытым ключом. Чтобы подписать некоторое сообщение* m, *а затем проверить подпись, выполняются следующие шаги.*

1. *Пользователь А генерирует случайное число* k, *причем* $k<q$.
2. *Пользователь А вычисляет значения*

$$r = (a^k \bmod p) \bmod p ,$$
$$s = (x * r + k\ (H(m))) \bmod p .$$

Если $H(m) \bmod q = 0$, *то значение* $H(m) \bmod q$ *принимают равным единице. Если* $r=0$, *то выбирают другое значение* k *и начинают снова. Цифровая подпись представляет собой два числа:*

$$r \bmod 2^{256} \text{ и } s \bmod 2^{256} .$$

Пользователь А отправляет эти числа пользователю В.

4. *Пользователь В проверяет полученную подпись, вычисляя*

$$v = H(m)^{q-2} \bmod q ,$$
$$z_1 = (s * v) \bmod q ,$$
$$z_2 = ((q-r) * v) \bmod q ,$$
$$u = ((a^{z_1} * y^{z_2}) \bmod p) \bmod p .$$

Если $u = r$, *то подпись считается верной.*

Различие между этим алгоритмом и алгоритмом DSA заключается в том, что в DSA

$$s = (k^{-1}\ (x * r + (H(m)))) \bmod q ,$$

что приводит к другому уравнению верификации.

Следует также отметить, что в отечественном стандарте ЭЦП параметр q *имеет длину 256 бит. Западных криптографов вполне устраивает* q *длиной примерно 160 бит. Различие в значениях параметра* q *является отражением стремления разработчиков отечественного стандарта к получению более безопасной подписи.*

Этот стандарт вступил в действие с начала 1995 г.

Литература

1. *Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. Под ред. В.Ф. Шаньгина. - 2-е изд., перераб. и доп. - М.: Радио и связь, 2001. - 376 с.: ил.*

2. ***Конеев И.Р., Беляев А.В. Информационная безопасность предприятия. - СПб.: БХВ-Петербург, 2003.***

Генерация личных ключей

Шифрование с использованием открытых ключей успешно, только если владелец личного ключа может контролировать его: всегда иметь под рукой, но не раскрывать никому. Существует несколько способов достичь этого:

- генерировать личный ключ;
- хранить личный ключ в файлах и смарт-картах;
- хранить личный ключ на сервере ;
- использовать сильную аутентификацию и слабые пароли.

Еще до возникновения проблемы защиты личных ключей возникает проблема их генерации.

Если используется алгоритм с ключом на основе возведения в степень по модулю (Диффи-Хелмана или DSS), то это просто вопрос выбора подходящего большого случайного числа.

Если же используется алгорит RSA, то необходимо выбрать два больших простых числа. И это само по себе проблема. Математически оперировать с большими простыми числами непросто, и именно поэтому они обеспечивают такую хорошую защиту. Но качество защиты зависит от выбора чисел, которые действительно являются простыми.

Стандартный подход к выбору простых чисел состоит в конструировании случайного числа, которое может быть простым, с последующей математической проверкой. Число отбрасывается, если в результате проверки не оказывается простым. К сожалению, нет практичных математических способов проверки, которые бы позволили точно сказать, что число действительно простое. Чтобы добиться этого, надо провести полное разложение числа на сомножители, а это не должно быть практически реализуемой вычислительной задачей для тех чисел, которые используются в качестве RSA-ключей. Иначе можно было бы самим использовать такие вычисления для взлома RSA-ключей.

Для обнаружения возможных сомножителей числа делается несколько проверок различными способами. Если такие сомножители не обнаружены, то делается вывод о том, что число - простое. Использование нескольких различных проверок снижает вероятность ошибки, ведь если в RSA-ключе есть дополнительные сомножители, то взломщику будет легче взломать его.

Первым шагом в конструировании простого числа является генерация случайного двоичного числа желаемого размера. После этого старший и младший биты устанавливаются в значение единица. Установка старшего бита в единицу гарантирует, что число будет иметь желаемый размер. Установка младшего бита в единицу гарантирует нечетность числа, ведь четное число явно не будет простым.

После построения такого числа осуществляется проверка его простоты. Обычно проверяется его делимость на "малые" простые числа (меньшие 2000).

На следующем шаге проводится вероятностный тест простоты числа. Существует нескоьлко вероятностных тестов, и все они работают путем выбора случайного числа, относительно которого затем осуществляется систематическая проверка простого числа.

Наиболее эффективен тест Миллера-Рабина (Miller-Rabin test). Прогоняя алгоритм Миллера-Рабина четыре раза, можно добиться чрезвычайно высокой степени уверенности в том, что число является простым.

По сути такой же подход используется для генерации RSA-ключей в пакете PGP. Сначала PGP конструирует простое число. Для проверки наличия у числа сомножителей PGP использует таблицу первых ста простых чисел (до 541). PGP случайным образом выбирает из таблицы значения и вычисляет остатки, выполняя поиск сомножителей в остатках. Наконец, PGP четыре раза прогоняет алгоритм вероятностной проверки на простоту числа. Эта операция выполняется в пакете PGP два раза, в результате чего оказываются сгенерированными два простых числа, которые и выбираются в качестве составляющих RSA-ключа.

Защита от копирования

Системы защиты от копирования можно разделить на следующие группы:

- привязка к дискете;
- привязка к компьютеру;
- привязка к ключу;
- опрос справочников;
- ограничение использования ПО.

В мировой практике существуют следующие способы распространения программ:

- ***FreeWare*** (свободно с сохранением прав за автором);
- ***ShareWare*** (2-4 недели опробовать, потом или не использовать или оплатить);
- ***CriptWare*** (две версии: демо+зашифрованная рабочая).

Большинство программ распространяется по принципу ***AS IS*** (как есть), общепринятым в международной компьютерной практике. Это означает, что за проблемы, возникающие в процессе эксплуатации программы, разработчик и распространитель ответственности не несут.

Защита программного продукта от несанкционированного копирования - актуальная задача в связи с сохранением коммерческих и авторских прав фирм и разработчиков. По сведениям зарубежных специалистов, экономический ущерб от "пиратского" копирования программного обеспечения составляет миллиарды долларов. Точные потери установить невозможно из-за отсутствия полных сведений о числе "пиратских" копий; считается, что с каждой программы их делается от 2 до 15. В Росии 95% используемого софта "пиратское", оставшиеся 5% - FreeWare.

С точки зрения профессионального программиста термин "защита от копирования" для IBM PC, работающей под управлением MS DOS/Windows, достаточно условен, так как практически всегда возможно переписать информацию, находящуюся на дискете или на жестком диске. Другое дело, что после этого программа может не выполняться. Таким образом, без санкции разработчика или фирмы-распространителя невозможно получить работоспособный программный продукт. То есть, фактически, "защита от копирования" - это создание средств, дающих возможность "защиты от несанкционированного выполнения".

Одной из распространенных технологий защиты от копирования, является создание особо определяемых дискет. Их особенность заключается в том, что на дискете создается специально организованная метка, которая используется как признак ее дистрибутивности. Функцию контроля метки выполняет специальная часть защищаемой программы. После копирования средствами OS защищаемого диска будет скопирована вся информация, за исключением метки. При выполнении программы ее контролирующая часть установит, что диск не дистрибутивный, и прервет выполнение программы. Тем самым программа как бы "привязывается" к своей дискете. Для создания метки применяются программные и аппаратные средства, а также их комбинирование.

Другой способ предотвращения незаконного использования программ и данных заключается в хранении информации в кодированном, зашифрованном виде. В этом случае без знания ключа работа с информацией невозможна.

Третий способ использовать ключи подключаемые к COM, LPT или USB портам.

Теперь давайте рассмотрим, что можно противопоставить "корсарам", рыщущим на волнах рынка software-продуктов.

Привязка к дискете

Для начала несколько слов об устройстве дискеты. Двухстороняя дискета 3.5" в дисководе 3.5". Скорость передачи данных дисководом 250 Кбит (DD) 500 Кбит (HD).

720	Kb	80	дорожек	9 секторов
800	Kb	80	дорожек	10 секторов
820	Kb	82	дорожки	10 секторов
830	Kb	83	дорожки	10 секторов (может не поддерживаться дисководом)
1.44	Mb	80	дорожек	18 секторов
1.52	Mb	80	дорожек	19 секторов
1.6	Mb	80	дорожек	20 секторов
1.68	Mb	80	дорожек	21 сектор (*)
1.72	Mb	82	дорожки	21 сектор (*)
1.74	Mb	83	дорожки	21 сектор (*) (может не поддерживаться дисководом)

(*) - При форматировании использовать Interlive(чередование секторов)=2, работает медленнее.

Файловая система на дискете FAT12.

Перестановка в нумерации секторов

При подготовке новой дискеты к работе она форматируется, т. е. определяется количество дорожек, длина сектора, количество секторов на дорожке, нумерация секторов (то есть формируется ID маркер) и производятся другие операции. Если эти действия осуществляются с установкой параметров (длины сектора, нумерации секторов, величины межсекторного промежутка и др.), отличных от принятых по умолчанию для системы MS DOS, то такой процесс будем называть нестандартным форматированием дискеты.

Один из методов зашиты от копирования основывается на перестановке номеров секторов на дорожке, то есть вместо обычной последовательности 1,2,3,4,5,6,7,8,9 вводится, например, 1,5,3,7,9,8,6,2,4.

При выполнении программы на скопированной дискете ее контролирующая часть определяет порядок следования секторов на заданной дорожке. Так как дискета, на которую осуществлено копирование, была отформатирована обычными средствами, то нумерация секторов установлена последовательная.

В дальнейшем контролирующая часть программы осуществляет сравнение вычисленного порядка следования секторов с установленным на дистрибутивной дискете. И так как порядок номеров не совпадает, то выполнение программы прекращается.

Введение одинаковых номеров секторов на дорожке

Другой схемой защиты, основанной на идее нестандартного форматирования, является способ, при котором часть секторов на определенной дорожке нумеруется

одинаково. Например, 1,2,3,3,3,6,7,8,9. В эти сектора записываются некоторые различные данные.

Контролирующая часть защищаемой программы должна определить, имеется ли на дискете несколько секторов с одинаковым номером. Для этого она посылает запрос чтения данных из сектора, номер которого повторяется (в нашем примере из сектора 3). В этом случае контроллер НГМД при выводе головки чтения/записи на сектор 3 может считать любой сектор с данным номером. Этот процесс повторяется заданное количество раз подряд, и очередные считанные данные сравниваются с полученными ранее. В случае их различия делается вывод о наличии на дискете секторов с одинаковым номером. Если же за заданное количество повторений цикла N различия в данных не найдено, то делается вывод о единственности сектора с номером 3.

Так как дискета, на которую осуществлено копирование, была отформатирована обычными средствами, то нумерация секторов на всех дорожках устанавливается последовательной. Контролирующая часть защищаемой программы организует проверку на наличие нескольких секторов с одинаковыми номером. И так как каждый номер сектора на заданной дорожке присутствует только один раз, то выполнение программы прекращается.

Введение межсекторных связей

В одном из методов защиты от копирования используется модификация изложенного выше способа, применяющего организацию нескольких секторов с одинаковым номером на заданной дорожке.

Суть метода в следующем. На выделенной дорожке дистрибутивной дискеты путем применения специальной программы организуется несколько секторов с одинаковым номером и в них записываются некоторые различные данные. Пусть, например, номера на 79 дорожке заданы как 2,2,1,4,5,6,7,8,9 и в первом секторе с номером 2 записано "C", а во втором секторе 2 - "B". Берется еще одна дорожка, например, 78, и сектор на ней, например, 1-ый, и в него записывается некоторая информация, пусть "A". В контролирующей части защищаемой программы организуется чтение сектора 1 на 78 дорожке и сразу же запрос чтения сектора 2 на 79 дорожке. Тем самым всегда обеспечивается переход к чтению информации из второго сектора с номером 2 на дорожке 79, то есть мы прочитаем биграмму "AB" с дистрибутивной дискеты.

Пусть было произведено "пиратское" копирование дискеты с защищаемой программой на обычно отформатированную дискету. Контролирующая часть программы считывает 1-ый сектор с 78 дорожки. Далее идет запрос чтения сектора 2 на 79 дорожке. Так как сектора на "пиратской" дискете пронумерованы стандартно, то есть 1,2,3,4,5,6,7,8,9, то после запроса сектора 2 будет получено значение "C". Следовательно, будет прочитана биграмма "AC".

Таким образом, контролирующая часть защищаемой программы устанавливает, что если дискета не дистрибутивная, то прочитывается биграмма "AC", означающая, что считан сектор 2 с 79 дорожки, который находится на ином физическом расстоянии от начала дорожки по сравнению с расстоянием для сектора 2 на дистрибутивной дискете, когда считывается биграмма "AB". Программа прерывается.

Изменение длины секторов

Еще одна схема защиты, основанная на методе нестандартного форматирования, использует изменение длины сектора. Напомним, что стандартная длина сектора, с которой работает MS DOS по умолчанию, - 512 байт, при этом на дорожке размещается 9 секторов. В процессе специального форматирования дискеты на заданной дорожке длина секторов устанавливается либо 128, либо 256 байт.

Введенное изменение длины секторов на заданной дорожке определяется контролирующей частью защищаемой программы. Это осуществляется, например, путем обращения к сектору с максимальным номером, либо проверкой того, что длина секторов на указанной дорожке 128 (или 256) байт.

Стандартно отформатированная дискета, на которую проведено "пиратское" копирование защищаемой программы, имеет длину секторов на всех дорожках 512 байт (по 9 секторов на дорожке). В ходе выполнения защищаемой программы ее контролирующая часть осуществляет проверку длины секторов на заданной дорожке. Если контролирующая часть использует способ обращения к сектору с максимальным номером (большим 9), то такой сектор найден не будет (так как их только 9) и выполнение программы прекращается.

Аналогично работает контролирующая часть и в способе, в котором осуществляется проверка, что длина секторов 128 (или 256) байт.

Изменение межсекторных промежутков

Таблица базы дисков, с помощью которой частично управляется работа НГМД, содержит параметр, определяющий величину межсекторного промежутка. Выбирается дорожка на дискете, которая специальной программой форматируется заново с измененным значением межсекторного промежутка в таблице базы дисков, что ведет к нестандартному расположению секторов.

Контролирующая часть защищаемой программы при обращении к выбранной дорожке производит ее анализ и делает вывод, является ли дискета дистрибутивной.

Использование дополнительной дорожки

При этом способе защиты используется дополнительная, 81-ая дорожка на дистрибутивной дискете. Заметим, что на гибком диске имеется место для размещения еще трех дорожек, сверх принятых 80. Дисковод позволяет вести на них запись данных. Поэтому на дополнительной 81-ой дорожке на дистрибутивной дискете хранится информация для контролирующей части защищаемой программы.

При копировании на стандартно отформатированную дискету дополнительно введенная 81-ая дорожка воспринята не будет, что и обнаруживается контролирующей частью защищаемой программы.

Ведение логических дефектов в заданный сектор

Сначала необходимо сказать несколько слов о физическом и логическом строении дорожки на гибком диске. Как известно, начало дорожек на дискете отмечено индексным отверстием. Каждый сектор на дорожке имеет две части: секторный ID маркер и собственно данные. ID маркер имеет шесть байт и дополнительные байты, которые

идентифицируют его для контроллера накопителя на гибких магнитных дисках (НГМД) как секторный маркер, а не как данные. Эти шесть байт следующие:

1-ый. Номер дорожки с 0 по 79 включительно.

2-ой. Номер стороны или номер головки. Верхняя сторона обозначается 0, нижняя - 1.

3-ий. Номер сектора на треке с 1 по 18 включительно.

4-ый. Размер сектора в байтах: 0,1,2 или 3 обозначают соответственно, 128-,256-,512- или 1024- байта в секторе. По умолчанию формат DOS имеет 512 байт в секторе.

5-ый и 6-ой. CRC байты для контроля при помощи циклического избыточного кода возможных искажений.

Четыре первых байта секторного ID маркера обозначаются как CHRN. CRC-байты фактически содержат 16-битную контрольную сумму CHRN-байт, которая вычисляется, когда сектор записывается. Когда же сектор читается, контрольная сумма перевычисляется для прочитанных из CHRN данных и сравнивается с данными из байтов CRC. Любое расхождение вызывает CRC-ошибку ID маркера.

Следующими за ID маркером идут несколько меточных байт маркера начала данных (BOD) и 512 байт собственно данных. Еще два проверочных байта CRC следуют за данными. Они содержат контрольную сумму 512 байт данных, вычисленную при записи сектора. Расхождение между контрольными суммами, записанной в CRC байтах и вычисленной при чтении, определяется как CRC-ошибка данных. После CRC-байтов первого сектора имеется межсекторный промежуток, затем идут другие сектора и межсекторные промежутки.

Теперь вам будет понятна идея метода защиты, основанного на использовании CRC-ошибки данных. Ее возникновение достигается следующим образом. Стандартная операция записи данных в сектор включает шесть шагов:

1. Включение мотора НГМД и короткое ожидание его разгона.
2. Выполнение операции поиска заданного сектора и ожидание прерывания, указывающего на завершение данной операции.
3. Инициализация микросхемы прямого доступа в память (DMA) 8237 для пересылки данных из памяти.
4. Посылка команды записи микросхеме контроллера НГМД 765 и ожидание прерывания, указывающего на окончание пересылки данных.
5. Получение информации о статусе контроллера НГМД.
6. Выключение мотора.

На дистрибутивной дискете при выполнении операции записи в определенный сектор во время четвертого шага изложенной выше процедуры производится кратковременное отключение мотора НГМД, что приводит к искажению записываемых данных. Контрольная же сумма данных в CRCбайтах будет иметь значение, которое было получено по исходным передаваемым данным. Так что при чтении данных из этого сектора и подсчета их контрольной суммы полученное значение, будет отличаться от содержащегося в CRC-байтах. Таким образом, контролирующая часть защищаемой

программы при попытке чтения заданного сектора обнаружит CRC-ошибку данных и определит дискету как дистрибутивную.

При копировании ранее искаженные данные на заданном секторе воспринимаются как истинные и по ним подсчитывается соответствующая контрольная сумма. Следовательно, при чтении данного сектора на скопированной дискете CRC-ошибка данных не возникает, и контролирующая часть защищаемой программы дискету отвергает.

Изменение параметров дисковода

Одной из схем создания защиты от копирования используется другая скорость вращения диска. Стандартная скорость вращения - 300 об/мин. Если ее понизить, скажем, до 280 об/мин, в то время как данные для записи передаются с прежней скоростью, то это понижение увеличивает плотность хранения данных на дискете, и на каждой дорожке образуется место для дополнительного десятого сектора. Это обстоятельство является основой защиты, так как работа с информацией, записанной на дискете, вращающейся с новой скоростью, возможна только при соответствующей модификации параметров дисковода.

Технология "ослабленных" битов

Способом записи информации на дискету является ее представление в виде 0 или 1. Данный метод защиты использует запись некоторого участка дистрибутивной дискеты с неопределенным уровнем сигнала. Таким образом, получается участок "ослабленных" битов. Данные с этого участка при их чтении несколько раз подряд будут восприниматься различным образом ввиду того, что сигнал может преобразовываться только в два дискретных значения - 0 или 1.

При копировании дистрибутивной дискеты участок "неопределенных" данных примет какое-то фиксированное значение.

Таким образом, контролирующая часть защищаемой программы организовывает чтение указанного участка несколько раз подряд. Если получаются одинаковые данные при всех запросах чтения, следовательно, дискета не дистрибутивная, и программа прерывается.

Физическая маркировка дискеты

Одной из наиболее известных технологий защиты от копирования является физическая пометка дискеты лазерным лучом путем ее прожигания. Такой дефект приводит к искажению данных в секторе и возникновению CRC ошибки данных при его чтении. При копировании пробивка не воспроизводится на принимающей дискете, хотя данные переписываются. Таким образом, CRC-ошибка данных при чтении сектора с копии не возникает.

Контролирующая часть защищаемой программы, производя чтение соответствующего сектора, это устанавливает и производит действия, предусмотренные разработчиком (прерывание выполнения программы, ее уничтожение и т.п.).

Проверка присутствия пробивки может производиться контролирующей частью программы и другим, более надежным способом. Она производит попытку записи в место, где должно находиться отверстие. Неудача - явный признак того, что дискета дистрибутивная. Если же запись производится успешно, следовательно, поверхность в данном месте не повреждена и дискета не дистрибутивная.

Рассмотренная технология применяется фирмой Vault Corporation в средстве PROLOK и его дальнейших модификациях.

Применение физического защитного устройства

Многие аппаратно - программные способы защиты основаны на том, что в компьютер добавляется специальное физическое защитное устройство. (Оно подключается, например, к порту ввода-вывода.) При запуске защищаемой программы ее контролирующая часть обращается к этому дополнительному устройству, проверяя его присутствие. Если оно не найдено (некоторые устройства еще и формируют код ответа, который затем анализируется), то производится останов программы, либо какие-то иные действия (например, уничтожение информации на дискете или "винчестере").

Общий принцип работы компьютера в этом случае следующий. После запроса на выполнение защищаемой программы происходит ее загрузка в оперативную память и инициализация контролирующей части. На физическое устройство защиты, подсоединенное к компьютеру через порт ввода-вывода (либо другим способом), посылается запрос. В ответ формируется код, посылаемый через микропроцессор в оперативную память для распознавания контролирующей частью программы. В зависимости от правильности кода ответа программа либо прерывается, либо выполняется.

Один из наиболее распространненых в России защитных устройств это Hasp by "Алладин". Это небольшое устройство, подключаемое к параллельному порту компьютера. Существует несколько разновидностей ключей с памятью, с часами. Обращение к HASP происходит исключительно через API ключа и, к тому же, через одну и ту же точку. Соответственно, туда ставится подпрограмма - эмулятор ключа. Все прочие навороты не имеют никакого смысла. Кроме того, возможно копирование самого ключа или аппаратная эмуляция.

"Привязка" к компьютеру.

Как Вы уже возможно заметили, "привязка" программы к дистрибутивной дискете путем описанных ранее способов защиты несет в себе большое неудобство для пользователя, связанное с необходимостью работы только со вставленной в дисковод оригинальной дискетой. Гораздо удобнее иметь необходимый программный продукт записанным на винчестере. Поэтому необходимо, чтобы контролирующая часть защищаемой программы (КЧЗП) "запомнила" свой компьютер и потом при запуске сравнивала имеющиеся характеристики с характеристиками "родного" компьютера. В случае их расхождения можно считать, что программа незаконно скопирована, и прервать ее выполнение. Для этого надо найти какие-то параметры, которые бы индивидуально характеризовали каждую вычислительную систему. На самом деле это весьма нетривиальная задача, поскольку открытая архитектура построения компьютеров IBM PC подразумевает их обезличенность.

Рассмотрим, что все же можно предложить для КЧЗП в качестве характеристик, которые могли бы проверяться при работе защищаемой программы.

Физические дефекты винчестера

При работе жесткого диска (винчестера) возможно возникновение сбойных участков на магнитной поверхности. Такие дефектные места могут иметься даже на совершенно новом винчестере. Номера этих сбойных участков помещаются в FAT (их признак - код FF7). При инсталляции защищаемой программы на винчестер в ее контролирующую часть записываются их адреса. В процессе выполнения программы осуществляется сравнение адресов сбойных участков, записанных в КЧЗП и в FAT. В случае запуска незаконно скопированной программы будет обнаружено расхождение сравниваемых адресов (первые не будут составлять подмножество вторых) и произойдет выполнение аварийных действий, предусмотренных для такого случая.

Развитием этой идеи является метод, в котором некоторые исправные кластеры помечаются как сбойные, и в них помещается информация для КЧЗП. (При копировании такие кластеры не передаются.)

Дата создания BIOS

Можно попытаться сузить класс компьютеров, на которых возможно функционирование незаконно скопированной программы. Это достигается, например, путем введения проверки даты создания BIOS, которая записана в ПЗУ каждого компьютера. Эта дата заносится в КЧЗП, и в процессе выполнения защищаемой программы осуществляется сравнение дат создания BIOS, записанных в КЧЗП и ПЗУ компьютера. В случае если защищаемая программа будет незаконно скопирована и установлена на компьютер другой серии, то КЧЗП это обнаружит и будут выполнены аварийные действия.

Дата создания BIOS записана в ПЗУ по адресу FFFF:0005 и занимает 8 байтов.

Версия используемой OS

Так как версия операционной системы, с которой работает пользователь на данном компьютере, не меняется на протяжении достаточно длительного времени, то ее также можно использовать в качестве параметра для организации проверки в КЧЗП. Этот контроль желательно проводить в комплексе с другими методами защиты, например, дополнительно к проверке даты создания BIOS, что еще более суживает класс машин, на которых может работать защищаемая программа.

Серийный номер диска

При форматировании диска на него создается так называемый серийный номер, его несложно изменить с помощью программ типа DiskEdit.

Тип компьютера

Для обеспечения работы защищаемой программы только на компьютере одного клона надо, чтобы она могла определять его тип. Такая информация содержится в байте, расположенном по адресу FFFF:000E в ROM BIOS со следующей кодировкой:

PC - FF; XT - FE,FB; PCjr - FD; AT - FC; PS/2 - FC,FA,F8;

PC-совместимый - F9.

Конфигурация системы и типы составляющих ее устройств

В развитие идеи, изложенной выше, рассмотрим способ, при котором КЧЗП в качестве параметра использует всю конфигурацию системы. Конечно надо учитывать, что этот способ будет давать ошибки и отказывать в выполнении законной копии программы, если какие либо составляющие вычислительной системы будут изменены (например, отключен принтер), либо наоборот законная и незаконная копии программы будут работать на системах с одинаковой конфигурацией.

Рассмотрим способы определения параметров системы, которые используются затем в КЧЗП.

Для IBM AT параметры системы хранятся в так называемой ***CMOS памяти***. Она имеет 64 однобайтовых регистра, пронумерованных от 00 до 3Fh. Назначения некоторых регистров представлены в табл. 1.

Таблица 1

Назначение некоторых регистров CMOS памяти

РЕГИСТР	НАЗНАЧЕНИЕ
00h	Секунды
01h	Second Alarm
02h	Минуты
03h	Minute Alarm
04h	Часы
05h	Hour Alarm
06h	День недели
07h	День месяца
08h	Месяц
09h	Год
10h	Типы НГМД A: и B: (0000 - не установлен; 0001 - 360Кб; 0010 - 1.2Мб; 0011 - 1.44Мб)
2h	Типы НЖМД C: и D: - для AT; D: - для PS/2
14h	Байт установленной периферии
15h-16h	Размер памяти на основной системной плате (0100h - 256Кб; 0200h - 512Кб; 0280h - 640Кб)
17h-18h	Размер памяти канала ввода/вывода

30h-31h	Размер дополнительной памяти сверх 1Майта

Регистр 10h содержит информацию о НГМД: его биты 7-4 описывают накопитель A, а биты 3-0 - накопитель B. Аналогично сведения о НЖМД помещаются в регистр 12h.

Информация о периферии хранится в байте 14h CMOS памяти. Для ее получения надо сначала записать номер регистра (в данном случае 14h) в порт с адресом 70h, а затем прочитать содержимое регистра через порт 71h. (Для записи какого-то значения в регистр CMOS памяти надо занести адрес регистра в порт с адресом 70h, а значение - в порт 71h.). Значения битов регистра 14h представлены в табл. 2.

Таблица 2

Назначение битов 14h-го регистра CMOS памяти

БИТ	СОДЕРЖИМОЕ	ЗНАЧЕНИЕ
7-6	00/01	Один/два НГМД
5-4	11/01/10	Видеорежим (монохромный/цветной 40*25/цветной 80*25)
3-2	Не спользуется	
1	1	Имеется математический сопроцессор
0	0/1	Нет/имеется НГМД

Компьютер может быть снабжен различными типами дисководов для гибких дисков и винчестеров. Рассмотрим с помощью каких средств можно получить информацию об их характеристиках кроме доступа к CMOS памяти.

Прерывание 21h DOS компьютера AT имеет функции 32h и 36h, связанные с определением характеристик установленных накопителей. Функция 36h сообщает текущие сведения о доступном пространстве на диске, номер которого загружается в регистр DL. Она возвращает:

- число секторов на один кластер в регистре AX;
- число незанятых кластеров - в BX;
- число байтов в одном секторе - в CX;
- число кластеров на диске - в DX.

Теперь легко узнать, каков объем диска, номер которого помещался в регистр DL. Для этого достаточно вычислить произведение значений регистров AX,CX,DX.

Функция 32h, позволяет получить таблицу с параметрами накопителя, номер которого загружен в регистр DL. Ее адрес будет содержаться в регистрах DX:BX. Подробное описание байтов этой таблицы можно найти в любом справочнике по прерываниям.

Состав установленного оборудования проверяется при загрузке, и результат проверки помещается в регистр статуса. Этот регистр занимает два байта, начиная с адреса 0040:0010, и в табл. 3 представлены значения его битов.

Для доступа к этому регистру кроме прямого обращения по адресу можно воспользоваться прерыванием 11h BIOS, которое возвращает два байта его значений в регистре AX.

В языке Turbo C имеется специальная функция BIOSEQUIP из библиотеки стандартных функций <bios.h>, которая возвращает целое число, описывающее оборудование, входящее в систему. Возвращаемое значение интерпретируется набором битовых полей, как представлено в табл. 4.

Таблица 3

Регистр состава установленного оборудования

БИТ	СОДЕРЖИМОЕ	ЗНАЧЕНИЕ
0	0/1	Нет/есть НГМД
1	0/1	Нет/есть математический сопроцессор 80x87
2-3	11	Оперативная память 64 Кбайта (в AT не используется и всегда равна 11)
4-5	11/01/10	Начальный видеорежим (монохромный/цветной 40*25/ цветной 80*25)
6-7	00/01/10/11	Число НГМД, если бит 0 =1 (соответственно 1,2,3,4)
8		XT/AT не используется (наличие микросхемы DMA)
9-11		Число адаптеров коммуникации RS232
12	1	Есть игровой адаптер (в AT не используется)
13		XT/AT не используется
14-15		Число присоединенных принтеров

Еще одно прерывание BIOS - 15h через функцию C0h позволяет получить адрес в ПЗУ, определяемый регистрами ES:BX, по которому находится табл. 4.

Таблица 4

БАЙТЫ	ЗНАЧЕНИЕ
0-1	Число байтов в таблице
2	Код модели компьютера (см. выше)
3	Различие между AT и XT/286 (подмодель)
4	Номер ревизии BIOS
5	80h - 3-й канал DMA, используется BIOS 40h - второй контроллер прерываний i8259 установлен 20h - таймер реального времени установлен 10h - int15h/AH=4Fh вызывается перед int 9h

	8h - допустимо ожидание внешнего события 4h - расширение BIOS размещено в 640 Кбайтах 2h - шиной является Micro Channel вместо шины ISA 1h - резерв Замечание: 1/10/86 XT BIOS возвращает некорректное значение 5-го байта

Для определения типа дисплея надо проверить бит номер 1 байта, находящегося по адресу 0040:0087. Когда этот бит равен 1 - подсоединяется монохромный дисплей, а когда он равен 0 - цветной.

Получение инженерной информации жесткого диска

Контроллер IDE, SCSI имеет специальную команду выдачи информации о подключенном устройстве. Программа IdeInfo выдает блок 512 байт информации о жестком диске, если в системе есть контроллер IDE и жесткий диск. Информация содержит параметры диска и его серийный номер. Некоторые OS не дают доступ к этой информации.

Опрос справочников

Еще один способ защиты от несанкционированного использования информации называется опросом справочников. Он не обеспечивает стойкой, долговременной защиты программного обеспечения, но может быть легко реализован и очень удобен в работе. При его применении пользователь не обязан работать с дистрибутивной дискетой, можно и с ее копиями. Работа программ, защищенных на основе этого метода, и их копий делится на два этапа. Первый: получение от пользователя информации из некоторого источника, называемого справочником, и сравнение ее с эталонной. (То есть программа проверяет, действительно ли введенная информация находится в справочнике.) Второй: работа самой программы или ее копии. При этом необходимо сделать все возможное, чтобы текст справочника был доступен только законному пользователю, так как доступ к справочнику в этом методе снимает все преграды для работы программы. Описанная возможность работы с копиями, недоступная при реализации других методов защиты данных, заставляет изготовителей программного обеспечения применять этот способ в своих разработках.

В простейшем случае справочником является обычный текстовый файл, который пользователь располагает либо на дискете, либо на жестком диске, либо имеет его в качестве распечатки. Его можно сделать скрытым, если он расположен на магнитном носителе. Вместо текстового файла можно использовать служебную информацию, которая характеризует определенный класс машин. Однако в этом случае придется преодолевать дополнительные сложности при доступе к такой информации.

Сравнение введенной информации (пароля) с соответствующей информацией из текста справочника может быть осуществлено двумя способами:

1. Путем поиска информации в текстовом файле, расположенном на магнитном носителе, по указанным координатам и сравнения ее с введенной. Например, программа выдает запрос: "Введите пятое слово третьего абзаца седьмой страницы." Пользователь

вводит слово. Программа считывает из текста справочника пятое слово третьего абзаца седьмой страницы и сравнивает его с введенным.

2. Путем сравнения слова из определенной части текста, представленного распечаткой, с данными, содержащимися в защищающей программе. Этот случай является как бы обратным к случаю 1. Программа содержит информацию, что в тексте справочника такие-то слова находятся в таких-то местах (например, слово "программа" является пятым словом третьего абзаца седьмой страницы, слово "справочник" - первое слово второго абзаца пятой страницы и т.д). В ответ на запрос программы, описанный в первом способе, пользователь в тексте справочника по указанным координатам ищет слово и вводит его. Программа сравнивает полученное слово с информацией, которой она располагает.

При небольшом по объему справочнике, а также малом запасе опрашиваемых парольных слов целесообразно список всех парольных слов разместить совершенно открыто в качестве файла на специальной дискете. Это позволит пользователю не держать в памяти информацию о том, какое слово где находится, и не тратить время на поиски нужного слова в справочнике, а снимать защиту самым элементарным образом, опробуя каждое слово из созданного списка. Но если эта дискета попадет в руки к постороннему лицу, то оно будет благодарно вам за возможность просто ознакомиться с вашими секретами.

Однако просмотр перед каждым запуском программы списка паролей и опробование каждого из них занимает довольно много времени. К тому же в таких системах часто возникают ошибки из-за неоднозначности решения: выделять или нет заголовок как отдельный абзац. Например, взяв пятое слово из третьего абзаца в качестве пароля, вы не можете запустить программу, так как автор выделил заголовок в отдельный абзац и требуемое слово находится не в третьем, а во втором абзаце текста, если рассматривать его без заголовка.

Из-за указанных недостатков описанный выше механизм чаще всего используются в качестве составной части многопрофильных систем защиты от копирования.

Введение ограничений на использование программного обеспечения

В одном из способов защиты программного обеспечения от незаконного использования применяются ограничения по:

- времени его эксплуатации;
- количеству запусков;
- его перемещению для использования на других машинах.

Иногда изготовители программного обеспечения накладывают ограничения на срок его работы у определенного пользователя, то есть как бы выдают временную лицензию, по истечении срока которой происходит либо самоуничтожение программы, либо она перестает работать. Идея, лежащая в основе этого метода, достаточно прозрачна: нужно только получить текущую дату из таймера, а затем, сравнив с "лицензионным" сроком использования, либо разрешить работу программы, либо запретить. Однако и недостаток данного метода защиты ясен: можно просто изменить дату, установленную в таймере, и тогда программа будет выполняться незаконно.

Еще одним способом ограничить использование программного обеспечения является введение счетчика запусков. Обычно такой счетчик устанавливают в:

- CMOS памяти;
- теле программы или в специальных числовых файлах;
- загрузочном секторе диска(вместо информации о версии DOS);
- FATe (в первом элементе таблицы, обычно заполненном кодом FFF);
- резервных секторах;
- сбойных секторах;
- системном реестре;
- за концом одного из файлов на локальном диске.

Уменьшая значение счетчика при каждом запуске, контролирующая часть программы следит за ним и при достижении нулевого значения выполняет предусмотренные разработчиком действия.

Защита CD от копирования

В настоящее время существует большое количество специальных форматов данных, используемых для записи информации на компакт-диск. К ним относятся не только формат для аудиоданных (CD Digital Audio - CD-DA) и формат, применяемый для хранения произвольной информации в общепринятом для современных компьютерных систем виде (DATA CD), но также специфические форматы, позволяющие создавать фотоколлекции (Kodak CD, CD-G), хранить видеоинформацию в доступном к воспроизведению виде (CD-I), сохранять специфическую текстовую информацию наряду с аудиоданными (CD-TEXT) и другие. Предтечей всех этих форматов является обычный аудиодиск. Развитие других форматов было связано исключительно с грандиозным скачком технологий, произошедших вскоре после внедрения аудиодисков в серийное производство. Фирма Philips, как разработчик базового стандарта записи аудиодисков, была вынужден признать необходимость разработок принципиально нового подхода к проблеме записи структурированных данных на компакт диск. Более того, в связи с существованием на мировом рынке целого ряда аппаратных платформ, работающих на существенно отличающихся операционных системах, была произведена попытка унификации формата записи данных на компакт-диск. Так возникли весьма экзотические форматы записи, в некоторых случаях необходимые для написания игр и мультимедиа на базе игровых консолей (Amiga CD32, Atari Jaguar, Sony Playstation), в других – для расширения возможностей мультимедийного подхода в компьютерных и бытовых технологиях в принципе (Video-CD, CD-I, CD-XA, CD-TEXT, CD-G). Следует отметить одну немаловажную деталь – все эти форматы являются адаптацией базового формата для записи аудиодисков.

Таким образом, на рынке за рекордно малый промежуток времени возникло гигантское количество компакт-дисков, несущих разнообразную информацию, как по содержанию, так и по фактической стоимости. Одновременно появилась проблема нелегального распространения подобного рода данных появилась.

За 15 лет развития медиа- и компьютерной индустрии были попытки защитить такого рода интеллектуальную собственность и весьма успешные. Однако говорить о тех методах, которые были применены в то время для реализации защиты, как о методах, которые можно предположительно использовать не в заводских условиях, просто не имеет смысла. Защита дисков тех лет базировалась исключительно на манипуляциях с покрытием дисков, что было в принципе невозможно сделать, не имея пресс-станка для тиражирования дисков заводским способом.

Но теперь ситуация изменилась. Не так давно на рынке появились устройства записи дисков, позволяющие производить достаточно сложные манипуляции как с данными, так и с физикой процесса записи. И, что особенно важно, эти устройства стали вполне доступными в ценовом плане и не требуют специального оборудования.

Рассмотрим методы защиты информации на CD:

Метод 1. Защита информации путем нарушения некоторых управляющих служебных сигналов, записанных на диск синхронно с данными.

Метод 2. Защита информации путем записи на заранее подготовленный носитель, поверхность которого содержит ряд неустранимых дефектов, не мешающих чтению, однако кардинально мешающих перезаписи диска.

Метод 3. Защита информации, базирующаяся на изменении файловой системы, используемой при записи. Следует сказать, что, несмотря на то, что этот метод защиты несколько менее универсален, он тоже позволяет эффективно защитить данные от возможности нелегального копирования диска целиком, т.к. возможна гибридная реализация методов 1-3.

Рассмотрим подробнее каждый из перечисленных методов.

Метод 1.

При записи абсолютно всех типов данных на компакт-диск синхронно с блоками данных формируется и записывается ряд управляющих цифровых сигналов. Подобная запись в подавляющем большинстве случаев делается аппаратно и означает, что при этом устройство при помощи внутреннего генератора формирует управляющие последовательности без непосредственного участия программы-копировщика и помещает их в конце каждого блока данных. Такие последовательности принято называть субканалами. Субканалов всего восемь и их принято нумеровать строчными английскими буквами P,Q,R,S,T,U,V,W.

Компакт-диски, записанные в стандарте CD-DA, используют лишь два субканала – P-субканал, который является по сути дела стробирующим при передаче данных от инициатора к устройству, и Q-субканал, в котором записывалась информация о тайм-коде, статусе устройства, кодах аппаратного корректора ошибок, работающего по схеме Соломона-Рида, а также некоторая другая. Для дальнейшего использования было зарезервировано еще 6 субканалов – R-W, которые на компакт-диск записываются, но фактически не используются. За период развития формата CD-DA в прочие форматы было сделано лишь несколько удачных попыток использования R-W субканалов. Например, в R-субканале при записи диска в форматах CD-G и CD-TEXT записывается некоторая пользовательская информация о копирайтах и авторстве для каждого трека. В редких случаях информацию из R-W субканалов используют тестовые программы для оценки производительности того или иного устройства чтения/записи дисков. Практически, на сегодняшний день ординарный компакт-диск, содержащий информацию произвольного типа, несет в себе 75% незадействованных субканалов.

Подобная ситуация позволяет особым образом защитить компакт-диск. В процессе записи субканалов отдельно от данных формируются полностью заполненные блоки, причем такая запись отнюдь не нарушает никаких договоренностей и стандартов записи данных на компакт диск, но дополняет. В области неиспользованных субканалов записывается дополнительная управляющая информация, которая неявно связана с данными субканалов P и Q. Программа для записи защищенных подобным образом дисков использует данные Q-субканала для формирования W-субканала. Данные, которые будут записаны в W-субканал, есть по сути закодированные симметричным алгоритмом соответствующие управляющие данные. Первичный ключ для кодирования формируется на основе данных, записанных в служебных областях на компакт-диске. Параллельно вводиться дополнительный псевдостробирующий R-субканал. Суть введения последнего заключается в том, что часть записанных на диск данных помещаются в область с верным Q-субканалом, но с ошибочным основным стробом. При чтении записанного таким образом компакт-диска эти данные будут просто проигнорированы, не вызвав сообщения об ошибке. Однако, при использовании программы, способной корректно читать подобные диски, перед принятием решения о том, насколько эффективна читаемая информация, будет произведена дизъюнкция реального и псевдостробирующих каналов, что обеспечит корректное и полное чтение данных во всем объеме.

Смысл введения кодированного W-субканала заключается в следующем. Большинство программ при копировании диск-диск не читают область субканалов непосредственно, но используют встроенные генераторы либо полагаются на возможности самого устройства. При попытке перезаписи диска с использованием подобных алгоритмов W и R субканалы потеряются. Программа, которую нелегально скопировали, перед запуском или в процессе своей инсталляции проверит ультраструктуру субканалов носителя, с которого произведен запуск и в том случае, если декодирование субканала с программно полученным ключом не даст контрольного результата, просто не запуститься.

Данный вариант защиты еще более эффективен при совместном использовании с методом 2.

Метод 2.

Данные должны быть записаны на диск, содержащий “плохие” для чтения области. Подобные области не должны мешать чтению данных ни одним из принятых для данного формата методов. Попытки чтения этих областей при копировании компакт-диска должны закончиться негативно и прервать процесс копирования.

Существует целый ряд устройств для записи компакт-дисков, поддерживающих команды управления мощностью лазера и скоростью вращения вала привода. К ним относятся Plextor, Matsashita, Plasmon и некоторые модели Teac. Создание сбойной области на диске сводиться к процедуре хаотичного варьирования этих параметров в процессе записи при помощи SCSI-команд Optical Power Calibration и Set Shaft Spd [1,2]. Необходимым и достаточным условием последующего успешного чтения данных, записанных подобным образом, есть точный мониторинг сбойных зон в момент записи. Это означает, что записывающая программа варьирует параметры лазера до записи, после чего, при достижении нормальных условий записи на поверхность диска в данном месте, записывает эффективные данные. Причем данные Q-субканала, отвечающие за позиционирование следующего не сбойного сектора, формируются внутри пишущей программы, а не самим устройством и записываются отдельно.

Таким образом, используя этот алгоритм, мы можем получить защищенный диск с практически любыми данными.

Недостаток этого метода состоит в ощутимом уменьшении общего объема данных, которые возможно записать при повышении степени защиты диска. Однако преимущество метода в том, что ни одно устройство копирования не сможет сделать копию такого диска в режимах TAO (Track-at-Once), SAO (Session-at-Once) и RAW. Конечно пользователь может нелегально скопировать данные без дублирования структуры диска, но легко реализуемая программная проверка носителя, откуда запущена программа, не даст возможности для запуска программы не с оригинального компакт-диска.

Метод 3.

Запись данных на любой носитель всегда делается структурировано. Метод построения базовых структур для упорядочения информации на носителях принято называть файловой системой. Этот метод определяет такие параметры, как размер апертуры чтения/записи (что иногда ошибочно называют длинной сектора), способ формирования директориальных записей и таблицы размещения, синхронизационные данные и коды контрольных сумм.

Запись данных на компакт-диск производиться с использованием файловой системы ***CDFS*** (Compact Disc File System). При этом в служебной области формируется таблица размещения данных, содержащая векторы начала данных (дорожек или файлов) и длины.

Суть данного метода защиты сводиться к использованию нестандартной файловой системы при абсолютно стандартной записи таблицы размещения. При записи совокупности данных на диск пишущая программа формирует таблицу и записывает ее в соответствующую часть служебной области. При этом запись о размере данных остается равной нулю, а первый вектор данный указывает на область, в которой в стандартном ***CDFS***-формате записан блок данных, соответствующих специфической программе-загрузчику. Собственно данные пишутся после этого блока уже в формате защищенной файловой системы.

При попытке копирования такого диска стандартная программа чтения определит, что диск заполнен, но суммарная длина всех файлов близка к нулю. И не сможет выполнить копирование ни в одном режиме, кроме режима RAW. С другой стороны, при запуске с такого диска программа-загрузчик, которая начнет работать автоматически, корректно прочитает данные из областей с нестандартной файловой системой, после чего приложение, записанное на диске, запуститься. Использование этого метода возможно вместе с методами 1 и 2, что еще в большей степени повышает надежность защиты.

Описанные методы позволяют осуществить защиту от попыток копирования с использованием режимов TAO, SAO и RAW. Однако следует помнить, что существует еще и метод снятия копии с матрицы-оригинала компакт-диска в заводских условиях, когда делается полная и точная физическая копия на пресс-станке, а потом тиражируется в нужных пиратам объемах. Но это уже скорее проблема в решении общих вопросов обеспечения безопасности на фирме-производителе. Поэтому мы считаем, что разработанные нами методы должны существенно снизить возможность нелегального копирования и тиражирования данных с компакт-диска.

Сравнение различных защит.

Данная таблица позволят в наглядном виде не только получить основные параметры всех защитных систем, но и провести анализ их свойств. Сильной защитой можно назвать ту, которая совсем не вскрыта (пока не вскрыта). Перспективной можно считать ту, которая вскрыта, но вскрыта каким-то одним способом. Такая защита имеет перспективу стать трудно вскрываемой, если ее разработчики смогут усилить тот или иной слабый блок. И слабой будем считать ту защиту, которая взломана всеми тремя известными способами, что говорит о чрезвычайно низкой защитной функции.

Наименование защиты	***Фирма-производитель\страна***	***Способ защиты***	***Необходимость в специальной аппаратуре для защиты***	***Защита небольших партий на CD-R\RW***	***Способы взлома***
Cd-Cops	Link Data Security	Анализ физических характеристи	НЕТ	НЕТ	Существует несколько "кряков"*

		к CD. Без установки меток			
LaserLock	MLS LaserLock International	Нанесение не копируемых меток	ДА	НЕТ	Эмуляция**, побитовое копирование***, "кряк"
StarForce	ProtectionTechnology (Россия)	Анализ физических характеристик CD. Без установки меток	НЕТ	ДА	Защита пока не вскрыта****
SafeDisk	Macrovision Corporation	Нанесение не копируемых меток	ДА	НЕТ	Эмуляция, побитовое копирование, "кряк"
SecuRom	Sony	Нанесение не копируемых меток	ДА	НЕТ	Эмуляция, побитовое копирование, "кряк"
TAGES	Thomson & MPO	Анализ физических характеристик CD. Без установки меток	НЕТ	НЕТ	Эмуляция, "кряк"

*Под термином "кряк" здесь понимается внешняя программа, способная дезавуировать защиту. При данном способе в код защищенной программы вносятся изменения.

**Данный вид программ эмулирует лазерные метки. При данном подходе в код вскрываемой программы не вносится изменений

***Наиболее распространенный способ копирования, смысл которого состоит в использовании специальных побитовых копировщиков, наподобие CloneCD. Данный тип защиты может работать как сам по себе, так и вместе с "кряком".

****справедливости ради стоит отметить, что прецедент вскрытия есть, но стал он возможен только благодаря тому, что пираты получили доступ к незащищенному коду приложения, после чего был сделан кряк.

Краткий справочник по методам взлома и способам защиты от них

(1) Взлом копированием и эмулированием

Побитовое копирование

Суть атаки заключается в том, что пользователь (не всегда злоумышленник) пытается скопировать имеющийся у него диск с целью создания копии (для личного использования или для тиража).

Для осуществления подобной атаки могут использоваться различные программы, зачастую входящие в поставку устройств CD-R/RW. Это и официальный Easy CD Creator, и полуофициальные (полухакерские) CloneCD и BlindRead:

Защита должна уметь противодействовать данному виду взлома, так как с него обычно и начинается взлом, поскольку копировщиков способных скопировать диски с примитивной защитой великое множество.

Способы обороны: существуют два способа противодействия взлому. Первый заключается в том, что на диск записывается определенная метка, которая не копируется обычными средствами (например, создается нестабильный сегмент, который не читается носителем, а раз не читается, то и скопированным быть также не может). К сожалению, данный способ не всегда устойчив, поскольку уже есть программы "продвинутого" копирования (те же CloneCD и BlindRead), которые способны пропускать подобные места (замещать нестабильные области произвольными данными) и проводить копирование до конца. Второй способ основывается на том, что ничего никуда записывать не надо, а надо лишь определенным образом запоминать физические характеристики диска, которые просто невозможно воспроизвести любым копированием, точнее диск сам по себе копируется, но уже с другой физической структурой. Соответственно, пользователь может спокойно клонировать диски, но ключевым будет тот, который был официально куплен.

Эмулирование

Данный подход позволяет формировать виртуальные драйверы устройств и имитировать обращение к диску. Это уже чистой воды взлом, поскольку для нормальной работы вскрытого приложения в систему инсталлируется специальный драйвер, который имитирует обращение к не копируемой метке на диске и возвращает вскрытой программе именно те данные, которые она ожидает "увидеть". Подобный способ довольно часто применяется на первых порах, когда хакеру известен способ получения некопируемой метки на диске, но ему не очень хочется разбираться с программой методом дизассемблирования.

Противодействием может служить работа с устройствами записи\чтения на низком уровне, когда невозможно перехватить вызовы к оборудованию. Здесь нужно еще внести одно пояснение: для того, чтобы защищенному приложению обратиться к CD, и проверить его на наличие не копируемой метки, необходимо воспользоваться одной из функций чтения\записи, которые предоставляет сама Windows. Хакерами уже наработан ряд механизмов, позволяющих перехватывать стандартные обращения к функциям Windows, а раз можно перехватить сообщение, значит целиком можно имитировать чтение, целиком заменяя стандартные вызовы на собственные. Как говорилось выше, противодействием данному способу взлома может быть только обращение к накопителю не через стандартные вызовы.

(2) Взлом программного модуля

Это следующий уровень взлома. В том случае если не удалось скопировать приложение, а способ его защиты также неизвестен, то хакер переходит на следующий уровень взлома - на исследование логики самой программы, с той целью, чтобы, проанализировав весь код приложения, выделить блок защиты и деактивировать его.

Взлом программ осуществляется двумя основными способами. Это отладка и дизассемблирование.

Отладка - это специальный режим, создаваемый специальным приложением - отладчиком, который позволяет по шагам исполнять любое приложение, передавая ему всю среду и делая все так, как будто приложение работает только с системой, а сам отладчик невидим. Механизмами отладки пользуются все, а не только хакеры, поскольку это единственный способ для разработчика узнать, почему его детище работает неправильно. Естественно, что любую благую идею можно использовать и во зло. Чем и пользуются хакеры, анализируя код приложения в поиске модуля защиты.

Это так называемый, пошаговый режим исполнения, или, иными словами интерактивный. А есть еще и второй - дизассемблирование - это способ преобразования исполняемых модулей в язык программирования, понятный человеку - Ассемблер. В этом случае хакер получает распечатку того, что делает приложение. Правда распечатка может быть очень и очень длинной, но никто и не говорил, что защиты снимать легко.

Хакеры активно пользуются обоими механизмами взлома, поскольку иногда приложение проще пройти по шагам, а иногда проще получить листинг и проанализировать его.

Давайте теперь рассмотрим основные методы взлома и противодействия ему

Отладчики

Отладчиков существует великое множество: от отладчиков, являющихся частью среды разработки, до сторонних эмулирующих отладчиков, которые полностью "погружают" отлаживаемое приложение в аналитическую среду, давая разработчику (или хакеру) полную статистику о том что и как делает приложение. С другой же стороны, подобный отладчик настолько четко имитирует среду, что приложение, исполняясь под ним, считает, что работает с системой напрямую (типичный пример подобного отладчика - SoftIce).

Противодействие

Способов противодействия существует великое множество. Это именно способы противодействия поскольку основная их задача сделать работу отладчика либо совсем невозможной, либо максимально трудоемкой. Опишем основные способы противодействия:

Замусоривание кода программы. Способ, при котором в программу вносятся специальные функции и вызовы, которые выполняют сложные действия, обращаются к накопителям, но по факту ничего не делают. Типичный способ обмана. Хакера нужно отвлечь, создав ответвление, которое и будет привлекать внимание сложными вызовами и

содержать в себе сложные и большие вычисления. Хакер рано или поздно поймет, что его обманывают, но время будет потеряно.

Использование мультипоточности. Тоже эффективный способ защиты, использующий возможности Windows по параллельному исполнению функций. Любое приложение может идти как линейно, то есть инструкция за инструкцией, и легко читаться отладчиком, а может разбивать на несколько потоков, исполняемых одновременно, естественно, в этом случае, нет никакого разговора о линейности кода, а раз нет линейности, то анализ здесь трудноосуществим. Как правило, создание 5-6 и более потоков существенно усложняет жизнь хакеру. А если потоки еще и шифруются, то хакер надолго завязнет, пытаясь вскрыть приложение.

Подавление изменения операционной среды - программа сама несколько раз перенастраивает среду окружения, либо вообще отказывается работать в измененной среде. Не все отладчики способны на 100% имитировать среду системы, и если "подопытное" приложение будет менять настройки среды, то рано или поздно "неправильный" отладчик может дать сбой

Противодействие постановке контрольных точек. Специальный механизм, поддерживаемы микропроцессором, при помощи которого, можно исследовать не всь программу, начиная с начал, а, например, только начиная с середины. Для этого в середине программы ставят специальный вызов (точка вызова - BreakPoint), который передает управление отладчику. Недостаток способа кроется в том, что для осуществления прерывания в код исследуемого приложения надо внести изменение. А если приложение время от времени проверяет себя на наличие контрольных точек, то сделать подобное будет весьма и весьма непросто.

Изменений определенных регистров процессора, на которые отладчики неадекватно реагируют. Также как и со средой. Отладчик тоже программа и тоже пользуется и операционной системой и процессором, который один на всех. Так если менять определенные регистры микропроцессора, которые отладчик не может эмулировать, то можно существенно "подорвать" его здоровье.

Дизассемблеры и дамперы

Про дизассемблер сказано было выше, а вот про дампер можно добавить то, что это практически тот же дизассемблер, только транслирует он не файл, находящийся на диске в Ассемблерный код, а содержимое оперативной памяти на тот момент, когда приложение начало нормально исполняться (то есть, пройдены все защиты). Это один из коварных средств взлома, при котором хакеру не надо бороться с механизмами противодействующими отладке, он лишь ждет, когда приложение закончит все проверки на легальность запуска, проверяя метки на диске, и начинает нормальную работу. В этот момент дампер и снимает "чистенький" код без примесей. К всеобщей радости не все защиты могут просто так себя раскрыть! И об этом ниже:

Шифрование. Самый простой и эффективный способ противодействия. Подразумевает, что определенная часть кода никогда не появляется в свободном виде. Код дешифруется только перед передачей ему управления. То есть вся программа или ее часть находится в зашифрованном виде, а расшифровывается только перед тем как исполниться. Соответственно, чтобы проанализировать ее код надо воспользоваться отладчиком, а его работу можно очень и очень осложнить (см. выше)!

Шифрование и дешифрование (динамическое изменение кода). Более продвинутый способ шифрования, который не просто дешифрует часть кода при исполнении, но и шифрует его обратно, как только он был исполнен. При такой защите хакеру придется проводить все время с отладчиком, и взлом защиты затянется на очень и очень долгое время.

Использование виртуальных машин. Еще одна модернизация шифрования. Способ заключается в том, чтобы не просто шифровать и дешифровать целые фрагменты кода целиком, а делать это покомандно, подобно тому, как действет отладчик или виртуальная машина: взять код, преобразовать в машинный и передать на исполнение, и так пока весь модуль не будет исполнен. Этот способ гораздо эффективнее предыдущих, так как функции приложения вообще никогда не бывают открытыми для хакера. Естественно, что его трудно реализовать, но реализовав, можно оградить себя от посягательств любых хакеров. Но в этом способе кроется недостаток - производительность, ведь на подобное транслирование требуется много времени, и, соответственно, способ хорош для защиты только для критических участков кода.

Дополнительные способы противодействия

Здесь уже идет чистое описание всяких возможностей по противодействию. Даются общие вводные, ведь защита может быть эффективной только тогда, когда каждый ее модуль написан на совесть с использованием различных ухищрений. То есть все рецепты, о которых говорилось выше, должны в той или иной форме присутствовать в любой системе.

Использовать для хранения данных защиты системные ресурсы Windows: дополнительную память, выделяемую для параметров окон и локальные хранилища потоков. Суть способа состоит в нестандартном использовании стандартных областей, скажем, хранить ключи, пароли: и т.п., совсем не там, где их будут искать при взломе в первую очередь.

Использовать операции сравнения нестандартными способами, во избежание их явного присутствия. Для сравнения есть определенные инструкции микропроцессора, о которых знают и разработчики и хакеры. А если попытаться использовать нестандартные виды сравнения, то можно слегка запутать хакера, ожидающего стандартного ответа.

Избегать обращений к переменным, относящимся к защите напрямую. То есть использовать любые косвенные способы доступа к специальным областям.

Использовать метод "зеркалирования" событий, то есть применять нестандартные действия на стандартные вызовы. Об этом говорилось выше.

Использовать для шифрования надежные, проверенные временем алгоритмы и т. д.

Здесь перечислены только основные подходы, даже не основные, а общеизвестные. А об оригинальных разработках мы узнаем позже, как только хакеры смогут взломать очередную уникальную защиту.

Защиту от копирования аудио-компакт-дисков Key2Audio, которую недавно внедрил ряд звукозаписывающих компаний, можно взломать при помощи обыкновенной ручки-маркера.

Это выяснили некие анонимные исследователи, распространившие информацию о своём открытии в Интернете, сообщает Reuters.

Технология Key2Audio, разработанная компанией Sony, заключается в том, что на компакт-диск, содержащий музыкальные композиции, записывается дополнительный трек с неверными цифровыми данными. Этот трек, как правило, находится на внешнем круге диска. Персональные компьютеры устроены таким образом, что сначала они считывают именно информационные треки. Так как данные на защитном треке являются ошибочными, компьютер будет безуспешно пытаться их считать и не сможет воспроизвести записанную на том же диске музыку.

Это ограничение распространяется как на компьютеры PC, так и на Macintosh (некоторые машины этой платформы при использовании таких дисков зависают), а также на некоторые модели портативных и автомобильных проигрывателей. Обычные же домашние аудиоустройства проигрывают такие диски без проблем.

Обойти эту защиту оказалось достаточно просто: если "ложный трек" закрасить обычным маркером, то остальное содержимое защищённого диска легко может быть прочитано компьютером и, следовательно, скопировано на жёсткий диск или другой носитель информации

По материалам статей:
Павел Ткачев, Александр Синицкий, Павел Хлызов, Владимир Горчаков, Сергей Карловский
УДК 638.235.231 "ИСПОЛЬЗОВАНИЕ ПРОГРАММНО-АППАРАТНЫХ СРЕДСТВ ДЛЯ ЗАЩИТЫ ИНФОРМАЦИИ НА КОМПАКТ-ДИСКЕ ОТ НЕЛЕГАЛЬНОГО КОПИРОВАНИЯ И ТИРАЖИРОВАНИЯ"

Новичков Александр
Анализ рынка средств защиты программного обеспечения от несанкционированного копирования.

Литература:

1. International standard. Audio recording – compact disk digital audio system. CEI/ IEC 60908. February 1999.

2. National Standard of Accredited Standards Committee X3. Working draft. Project X3T9.2-375D. Revision 10L. September 1993.

3. American National Standards Institute, National Committee on Interface Technology Standards T10. Working draft. Project T10/1363-D. Revision 01. March 2000.

4. American National Standards Institute, National Committee on Interface Technology Standards T10. Project 333-2000. May 2000.

Защиты от несанкционированного доступа

Существует притча о самом надежном способе хранения информации: Информация должна быть в одном экземпляре на компьютере, который находится в бронированном сейфе, отключенный от всех сетей и обесточенный.

Понятно, что работать с такой информацией, мягко говоря, неудобно. В то же время хочется защитить программы и данные от несанкционированного доступа (НСД). А чтобы доступ был санкционированным, нужно определиться, кому что можно, а что нельзя.

Для этого нужно:

1. разбить на классы информацию, хранящуюся и обрабатывающуюся в компьютере;
2. разбить на классы пользователей этой информации;
3. поставить полученные классы информации и пользователей в определенное соответствие друг другу.

Доступ пользователей к различным классам информации должен осуществляться согласно системе паролей, в качестве которой могут выступать:

- обычные пароли;
- настоящие замки и ключи;
- специальные тесты идентификации пользователей;
- специальные алгоритмы идентификации ПЭВМ, дискеты, программного обеспечения.

Системы защиты информации от НСД обеспечивают выполнение следующих функций:

1. идентификация, т.е. присвоение уникальных признаков - идентификаторов, по которым в дальнейшем система производит аутентификацию;
2. аутентификация, т.е. установление подлинности на основе сравнения с эталонными идентификаторами;
3. разграничение доступа пользователей к ПЭВМ;
4. разграничение доступа пользователей по операциям над ресурсами (программы, данные и т.д.);
5. администрирование:
    a. определение прав доступа к защищаемым ресурсам,
    b. обработка регистрационных журналов,
    c. установка системы защиты на ПЭВМ,
    d. снятие системы защиты с ПЭВМ;
6. регистрация событий:
    a. входа пользователя в систему,
    b. выхода пользователя из системы,
    c. нарушения прав доступа;
7. реакция на попытки НСД;
8. контроль целостности и работоспособности систем защиты;

9. обеспечение информационной безопасности при проведении ремонтно-профилактических работ;
10. обеспечение информационной безопасности в аварийных ситуациях.

Права пользователей по доступу к программам и данным описывают таблицы, на основе которых и производится контроль и разграничение доступа к ресурсам. Доступ должен контролироваться программными средствами защиты. Если запрашиваемый доступ не соответствует имеющемуся в таблице прав доступа, то системы защиты регистрирует факт НСД и инициализирует соответствующую реакцию.

Идентификация и аутентификация пользователя

Прежде чем получить доступ к ресурсам, пользователь должен пройти процесс представления компьютерной системе, который включает две стадии:

- ***идентификацию*** - пользователь сообщает системе по ее запросу свое имя (идентификатор);
- ***аутентификацию*** - пользователь подтверждает идентификацию, вводя в систему уникальную, не известную другим пользователям информацию о себе (например, пароль).

Для проведения процедур идентификации и аутентификации пользователя необходимо наличие:

- программы аутентификации;
- уникальной информации о пользователе.

Различают две формы хранения информации о пользователе: внешняя (например, пластиковая карта или голова пользователя) и внутренняя (например, запись в базе данных). Естественно, что информация, хранящаяся в голове, и информация в базе данных должны быть семантически тождественны. Беда с жадным братом Али-Бабы Касимом приключилась именно из-за несовпадения внешней и внутренней форм: сим-сим не тождественен гороху, рису и т.д.

Рассмотрим структуры данных и протоколы идентификации и аутентификации пользователя.

Практически любому ключевому носителю информации, используемому для опознания, соответствует следующая структура данных о пользователе:

- ID_i - неизменный идентификатор i-го пользователя, который является аналогом имени и используется для идентификации пользователя;
- K_i - аутентифицирующая информация пользователя, которая может изменяться и служит для аутентификации (например, пароль $P_i = K_i$).

Так для носителей типа пластиковых карт выделяется неизменяемая информация ID_i и объект в файловой структуре карты, содержащий K_i.

Совокупную информацию в ключевом носителе можно назвать первичной аутентифицирующей информацией i-го пользователя. Очевидно, что внутренний аутентифицирующий объект не должен существовать в системе длительное время

(больше времени работы конкретного пользователя). Например, Вы ввели пароль, который программа аутентификации занесла в переменную для сравнения с хранящимися в базе данных. Эта переменная должна быть обнулена не позже, чем Вы закончите свой сеанс. Для длительного хранения следует использовать данные в защищенной форме.

Рассмотрим две типовые схемы идентификации и аутентификации.

Схема 1. В компьютерной системе хранится:

Номер пользователя	Информация для идентификации	Информация для аутентификации
1	ID_1	E_1
2	ID_2	E_2
...	...	...
n	ID_n	E_n

Здесь $E_i = F(ID_i, K_i)$, где "невосстановимость" K_i оценивается некоторой пороговой трудоемкостью T_0 решения задачи восстановления K_i по E_i и ID_i. Кроме того для пары K_i и K_j возможно совпадение соответствующих значений E. В связи с этим вероятность ***ложной аутентификации*** пользователей не должна быть больше некоторого порогового значения P_0. На практике задают $T_0 = 10^{20}...10^{30}$, $P_0 = 10^{-7}...10^{-9}$.

Протокол идентификации и аутентификации (для схемы 1).

1. Пользователь предъявляет свой идентификатор ID.
2. Если существует i = 1...n, для которого $ID = ID_i$, то пользователь идентификацию прошел успешно. Иначе пользователь не допускается к работе.
3. Модуль аутентификации запрашивает у пользователя его аутентификатор K.
4. Вычисляется значение E = F(ID, K).
5. Если $E = E_i$, то аутентификация прошла успешно. Иначе пользователь не допускается к работе.

Схема 2 (модифицированная). В компьютерной системе хранится:

Номер пользователя	Информация для идентификации	Информация для аутентификации
1	ID_1, S_1	E_1
2	ID_2, S_2	E_2
...	...	...
n	ID_n, S_n	E_n

Здесь $E_i = F(S_i, K_i)$, где S - случайный вектор, задаваемый при создании идентификатора пользователя; F - функция, которая обладает свойством "невосстановимости" значения K_i по E_i и S_i.

Протокол идентификации и аутентификации (для схемы 2).

1. Пользователь предъявляет свой идентификатор ID.

2. Если существует i = 1...n, для которого ID = ID_i, то пользователь идентификацию прошел успешно. Иначе пользователь не допускается к работе.
3. По идентификатору ID выделяется вектор S.
4. Модуль аутентификации запрашивает у пользователя его аутентификатор K.
5. Вычисляется значение E = F(S, K).
6. Если E = E_i, то аутентификация прошла успешно. Иначе пользователь не допускается к работе.

Вторая схема аутентификации применяется в ОС UNIX. В качестве идентификатора используется имя пользователя (запрошенное по Login), в качестве аутентификатора - пароль пользователя (запрошенный по Password). Функция F представляет собой алгоритм шифрования DES. Эталоны для идентификации и аутентификации содержатся в файле Etc/passwd.

Следует отметить, что необходимым требованием устойчивости схем идентификации и аутентификации к восстановлению информации K_i является случайный равновероятный выбор K_i из множества возможных значений.

Простейший метод применения пароля основан на сравнении представленного пароля с исходным значением, хранящимся в памяти. Если значения совпадают, то пароль считается подлинным, а пользователь - законным. Перед пересылкой по незащищенному каналу пароль должен шифроваться. Если злоумышленник каким-либо способом все же узнает пароль и идентификационный номер законного пользователя, он получит доступ в систему.

Лучше вместо открытой формы пароля P пересылать его отображение, получаемое с использованием односторонней функции f(P). Это преобразование должно гарантировать невозможность раскрытия пароля по его отображению. Так противник наталкивается на неразрешимую числовую задачу.

Например, функция f может быть определена следующим образом:

$$f(P) = E_P(ID) ,$$

где P - пароль, ID - идентификатор, E_P - процедура шифрования, выполняемая с использованием пароля в качестве ключа.

На практике пароль состоит из нескольких букв. Но короткий пароль уязвим к атаке полного перебора. Для того, чтобы предотвратить такую атаку, функцию f определяют иначе:

$$f(P) = E_{P+K}(ID) ,$$

где K - ключ (таблетка Toch-memory, USB-ключ и т.п.)

Процедуры идентификации и аутентификации пользователя могут базироваться не только на секретной информации, которой обладает пользователь (пароль, секретный ключ, персональный идентификатор и т.п.). В последнее время все большее распространение получает биометрическая идентификация и аутентификация, позволяющая уверенно идентифицировать потенциального пользователя путем измерения физиологических параметров и характеристик человека, особенностей его поведения.

Основные достоинства ***биометрических методов*** идентификации и аутентификации:

- высокая степень достоверности идентификации по биометрических признакам из-за их уникальности;
- неотделимость биометрических признаков от дееспособной личности;
- трудность фальсификации биометрических признаков.

В качестве биометрических признаков, которые могут быть использованы для идентификации потенциального пользователя, используются:

- узор радужной оболочки и сетчатки глаз;
- отпечатки пальцев;
- геометрическая форма руки;
- форма и размеры лица;
- термограмма лица;
- форма ушей;
- особенности голоса;
- ДНК;
- биомеханические характеристики рукописной подписи;
- биомеханические характеристики "клавиатурного почерка".

При регистрации пользователь должен продемонстрировать один или несколько раз свои характерные биометрические признаки. Эти признаки (известные как подлинные) регистрируются системой как контрольный "образ" законного пользователя. Этот образ пользователя хранится в электронной форме и используется для проверки идентичности каждого, кто выдает себя за соответствующего законного пользователя.

Системы идентификации по узору радужной оболочки и сетчатки глаз могут быть разделены на два класса:

- использующие рисунок радужной оболочки глаза;
- использующие рисунок кровеносных сосудов сетчатки глаза.

Поскольку вероятность повторения данных параметров равна 10^{-78}, эти системы являются наиболее надежными среди всех биометрических систем. Такие средства применяются, например, в США в зонах военных и оборонных объектов.

Системы идентификации по отпечаткам пальцев являются самыми распространенными. Одна из основных причин широкого распространения таких систем заключается в наличии больших банков данных по отпечаткам пальцев. Основными пользователями таких систем во всем мире являются полиция, различные государственные организации и некоторые банки.

Системы идентификации по геометрической форме руки используют сканеры формы руки, обычно устанавливаемые на стенах. Следует отметить, что подавляющее большинство пользователей предпочитают системы именно этого типа.

Системы идентификации по лицу и голосу являются наиболее доступными из-за их дешевизны, поскольку большинство современных компьютеров имеют видео- и аудиосредства. Системы данного класса широко применяются при удаленной идентификации в телекоммуникационных сетях.

Системы идентификации по динамике рукописной подписи учитывают интенсивность каждого усилия подписывающегося, частотные характеристики написания каждого элемента подписи и начертания подписи в целом.

Системы идентификации по биомеханическим характеристикам "клавиатурного почерка" основываются на том, что моменты нажатия и отпускания клавиш при наборе текста на клавиатуре существенно различаются у разных пользователей. Этот динамический ритм набора ("клавиатурный почерк") позволяет построить достаточно надежные средства идентификации.

Следует отметить, что применение биометрических параметров при идентификации субъектов доступа автоматизированных систем пока не получило надлежащего нормативно-правового обеспечения, в частности в виде стандартов. Поэтому применение систем биометрической идентификации допускается только в системах, обрабатывающих и хранящих персональные данные, составляющие коммерческую и служебную тайну.

Взаимная проверка подлинности пользователей

Обычно стороны, вступающие в информационный обмен, нуждаются во взаимной аутентификации. Этот процесс выполняется в начале сеанса связи.

Для проверки подлинности применяют следующие способы:

- механизм запроса-ответа;
- механизм отметки времени ("временной штемпель").

Механизм запроса-ответа. Если пользователь А хочет быть уверен, что сообщения, получаемые им от пользователя В, не являются ложными, он включает в посылаемое для В сообщение непредсказуемый элемент - запрос X (например, некоторое случайное число). При ответе пользователь В должен выполнить над этим числом некоторую заранее оговоренную операцию (например, вычислить некоторую функцию f(X)). Это невозможно осуществить заранее, так как пользователю В неизвестно, какое случайное число X придет в запросе. Получив ответ с результатом действий В, пользователь А может быть уверен, что В - подлинный. Недостаток этого метода - возможность установления закономерности между запросом и ответом.

Механизм отметки времени подразумевает регистрацию времени для каждого сообщения. В этом случае каждый пользователь сети может определить насколько "устарело" пришедшее сообщение и не принимать его, поскольку оно может быть ложным.

В обоих случаях для защиты механизма контроля следует применять шифрование, чтобы быть уверенным, что ответ послан не злоумышленником.

При использовании отметок времени возникает проблема *допустимого временного интервала задержки* для подтверждения подлинности сеанса. Ведь сообщение с "временным штемпелем" в принципе не может быть передано мгновенно. Кроме того, компьютерные часы получателя и отправителя не могут быть абсолютно синхронизированы.

Для взаимной проверки подлинности обычно используют ***процедуру "рукопожатия"***, которая базируется на указанных выше механизмах и заключается во взаимной проверке ключей, используемых сторонами. Иначе говоря, стороны признают друг друга законными партнерами, если докажут друг другу, что обладают правильными ключами. Процедуру "рукопожатия" применяют в компьютерных сетях при организации связи между пользователями, пользователем и хост-компьютером, между хост-компьютерами и т.д.

В качестве примера рассмотрим процедуру "рукопожатия" для двух пользователей A и B. Пусть применяется симметричная криптосистема. Пользователи A и B разделяют один и тот же секретный ключ K_{AB}.

- Пользователь A инициирует "рукопожатие", отправляя пользователю B свой идентификатор ID_A в открытой форме.
- Пользователь B, получив идентификатор ID_A, находит в базе данных секретный ключ K_{AB} и вводит его в свою криптосистему.
- Тем временем пользователь A генерирует случайную последовательность S с помощью псевдослучайного генератора PG и отправляет ее пользователю B в виде криптограммы $E_{KAB}(S)$.
- Пользователь B расшифровывает эту криптограмму и раскрывает исходный вид последовательности S.
- Затем оба пользователя преобразуют последовательность S, используя одностороннюю функцию f.
- Пользователь B шифрует сообщение f(S) и отправляет криптограмму $E_{KAB}(f(S))$ пользователю A.
- Наконец, пользователь A расшифровывает эту криптограмму и сравнивает полученное сообщение f'(S) с исходным f(S). Если эти сообщения равны, то пользователь A признает подлинность пользователя B.

Пользователь A проверяет подлинность пользователя B таким же способом. Обе эти процедуры образуют процедуру "рукопожатия", которая обычно выполняется в самом начале любого сеанса связи между любыми двумя сторонами в компьютерных сетях.

Достоинством модели "рукопожатия" является то, что ни один из участников связи не получает никакой секретной информации во время процедуры подтверждения подлинности.

Иногда пользователи хотят иметь непрерывную проверку подлинности отправителей в течение всего сеанса связи. Рассмотрим один из простейших способов непрерывной проверки подлинности.

Чтобы отправить сообщение M, пользователь A передает криптограмму $E_K(ID_A, M)$. Получатель расшифровывает ее и раскрывает пару (ID_A, M). Если принятый идентификатор ID_A совпадает с хранимым, получатель принимает во внимание это сообщение.

Вместо идентификаторов можно использовать секретные пароли, которые подготовлены заранее и известны обеим сторонам. Продолжение: Протоколы идентификации с нулевой передачей знаний

Литература

1. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. Под ред. В.Ф. Шаньгина. - 2-е изд., перераб. и доп. - М.:Радио и связь, 2001. - 376 с.: ил.

Протоколы идентификации с нулевой передачей знаний

Широкое распространение смарт-карт (интеллектуальных карт) для разнообразных коммерческих, гражданских и военных применений (кредитные карты, карты социального страхования, карты доступа в охраняемые помещения, компьютерные пароли и ключи и т.д.) потребовало обеспечение безопасности идентификации таких карт и их владельцев. Во многих приложениях главная проблема заключается в том, чтобы при предъявлении смарт-карты оперативно обнаружить обман и отказать обманщику в допуске, ответе и обслуживании.

Для безопасного использования смарт-карт разработаны протоколы идентификации с нулевой передачей знаний. Секретный ключ владельца карты становится неотъемлемым признаком его личности. Доказательство знания этого секретного ключа с нулевой передачей этого знания служит доказательством подлинности личности владельца карты.

Схему идентификации с нулевой передачей знаний предложили в 1986 г. У.Фейге, А.Фиат и А.Шамир. Она является наиболее известным доказательством идентичности с нулевой передачей конфиденциальной информации.

Рассмотрим сначала упрощенный вариант схемы идентификации с нулевой передачей знаний для более четкого выявления ее основной концепции.

Но прежде всего определимся с терминологией.

Пусть a, b, d, g $\in$ **Z** (множество целых чисел), n $\in$ **N** (множество натуральных чисел, т.е. положительных целых чисел).

Определение 1. Число a *сравнимо* с b по модулю n, если a и b при делении на n дают одинаковые остатки, т.е.

$$a \bmod n = b \bmod n.$$

Принятое обозначение:

$$a \equiv b \ (\bmod n) .$$

Определение 2. Число d называют *обратным* к a по модулю n, если произведение a*d при делении на n дает в остатке 1, т.е.

$$a*d \bmod n = 1 .$$

Принятое обозначение:

$$b = a^{-1} \ (\bmod n) .$$

Примите к сведению, что целое число a^{-1} (mod n) существует тогда и только тогда, когда a является взаимно простым с n, т.е. имеет с модулем n наибольший общий делитель, равный единице.

Тех, кому интересно, почему это действительно так, отсылаю к расширенному алгоритму Евклида. В Интернете вы найдете не один сайт, посвященный этой теме.

Определение 3. Число g называют *квадратным корнем* из a по модулю n, если произведение g*g при делении на n дает в остатке a, т.е.

$$g*g \bmod n = a .$$

Принятое обозначение:

$$g = \mathrm{sqrt}\ (\ a\ )\ (\ \mathrm{mod}\ n\ ) .$$

Упрощенная схема идентификации с нулевой передачей знаний

Прежде всего выбирается значение модуля n, который является произведением двух больших простых чисел. Модуль n должен иметь длину 512...1024 бит. Это значение n представляют группе пользователей, которым придется доказывать свою подлинность. В процессе идентификации участвуют две стороны:

- сторона A, доказывающая свою подлинность;
- сторона B, проверяющая подлинность A.

Для того, чтобы сгенерировать открытый и секретный ключи для стороны A, доверенный арбитр (Центр) выбирает такое число V, что сравнение

$$x^2 \equiv V\ (\ \mathrm{mod}\ n\ )$$

имеет решение. Число V при этом называют *квадратичным вычетом по модулю n*.

Кроме того должно существовать целое число

$$V^{-1}\ (\ \mathrm{mod}\ n\ ) .$$

Выбранное значение V является открытым ключом для A. Затем вычисляют наименьшее значение S, для которого

$$S \equiv \mathrm{sqrt}\ (V^{-1})(\ \mathrm{mod}\ n\ ).$$

Это значение S является секретным ключом для A.

Теперь можно приступить к выполнению протокола идентификации.

1. Сторона A выбирает некоторое случайное число r, r < n. Затем она вычисляет

$$x = r^2 \bmod n$$

и отправляет x стороне B.

2. Сторона B посылает A случайный бит b.

3. Если b = 0, тогда A отправляет r стороне B. Если b = 1, то A отправляет стороне B

$$y = r * S \bmod n .$$

4. Если b = 0, сторона B проверяет, что

$$x = r^2 \bmod n ,$$

чтобы убедиться, что A знает sqrt (x). Если b = 1, сторона B проверяет,что

$$x = y^2 * V \bmod n ,$$

чтобы быть уверенной, что A знает sqrt (V^{-1}).

Эти шаги образуют один цикл протокола, называемый *аккредитацией*. Стороны A и B повторяют этот цикл t раз при разных случайных значениях r и b до тех пор, пока B не убедится, что A знает значение S.

Если сторона A не знает значения S, она может выбрать такое значение r, которое позволит ей обмануть сторону B, если B отправит ей b = 0, либо A может выбрать такое r, которое позволит обмануть B, если B отправит ей b = 1. Но этого невозможно сделать в обоих случаях. Вероятность того, что A обманет B в одном цикле, составляет 1/2. Вероятность обмануть B в t циклах равна $(1/2)^t$.

Для того чтобы этот протокол работал, сторона A никогда не должна повторно использовать значение r. Если A поступила бы таким образом, а сторона B отправила бы стороне A на шаге 2 другой случайный бит b, то B имела бы оба ответа A. После этого B может вычислить значение S, и для A все закончено.

Параллельная схема идентификации с нулевой передачей знаний

Параллельная схема идентификации позволяет увеличить число аккредитаций, выполняемых за один цикл, и тем самым уменьшить длительность процесса идентификации.

Как и в предыдущем случае, сначала генерируется число n как произведение двух больших чисел. Для того, чтобы сгенерировать открытый и секретный ключи для стороны A, сначала выбирают K различных чисел V_1, V_2, ..., V_K, где каждое V_i , является квадратичным вычетом по модулю n. Иначе говоря, выбирают значение V, таким, что сравнение

$$x^2 \equiv V_i \ (\bmod n)$$

имеет решение и существует V_i^{-1} (mod n). Полученная строка V_1, V_2, ..., V_K является открытым ключом.

Затем вычисляют такие наименьшие значения S_i, что

$$S_i = \text{sqrt} (V_i^{-1}) \ (\bmod n) .$$

Эта строка S_1, S_2, ..., S_K является секретным ключом стороны A.

Протокол процесса идентификации имеет следующий вид:

1. Сторона А выбирает некоторое случайное число r, r<n. Затем она вычисляет

$$x = r^2 \bmod n$$

и посылает x стороне В.

2. Сторона В отправляет стороне А некоторую случайную двоичную строку из К бит:

$$b_1, b_2, ..., b_K.$$

3. Сторона А вычисляет

$$y = r * (S_1^{b_1} * S_2^{b_2} * ... * S_K^{b_K}) \bmod n .$$

Перемножаются только те значения S_i, для которых $b_i = 1$. Например, если $b_1 = 1$, то сомножитель S_1 входит в произведение, если же $b_1 = 0$, то S_1 не входит в произведение, и т.д. Вычисленное значение y отправляется стороне В.

4. Сторона В проверяет, что

$$x = y^2 * (V_1^{b_1} * V_2^{b_2} * ... * V_K^{b_K}) \bmod n .$$

Фактически сторона В перемножает только те значения V_i, для которых $b_i = 1$. Стороны А и В повторяют этот протокол t раз, пока В не убедится, что А знает S_1, S_2, ..., S_K .

Вероятность того, что А может обмануть В, равна $(1/2)^{Kt}$. Авторы рекомендуют в качестве контрольного значения брать вероятность обмана В равной $(1/2)^{20}$ при К = 5 и t = 4.

Пример. Рассмотрим работу этого протокола для небольших числовых значений. Если n = 35 (n - произведение двух простых чисел 5 и 7), то возможные квадратичные вычеты будут следующими:

1: $x^2 \equiv 1 \pmod{35}$ имеет решения: x = 1, 6, 29, 34;

4: $x^2 \equiv 4 \pmod{35}$ имеет решения: x = 2, 12, 23, 33;

9: $x^2 \equiv 9 \pmod{35}$ имеет решения: x = 3, 17, 18, 32;

11: $x^2 \equiv 11 \pmod{35}$ имеет решения: x = 9, 16, 19, 26;

14: $x^2 \equiv 14 \pmod{35}$ имеет решения: x = 7, 28;

15: $x^2 \equiv 15 \pmod{35}$ имеет решения: x = 15, 20;

16: $x^2 \equiv 16 \pmod{35}$ имеет решения: x = 4, 11, 24, 31;

21: $x^2 \equiv 21 \pmod{35}$ имеет решения: x = 14, 21;

25: $x^2 \equiv 25 \pmod{35}$ имеет решения: x = 5, 30;

29: $x^2 \equiv 29 \pmod{35}$ имеет решения: x = 8, 13, 22, 27;

30: $x^2 \equiv 30 \pmod{35}$ имеет решения: x = 10, 25.

Заметим, что 14, 15, 21, 25 и 30 не имеют обратных значений по модулю 35, потому что они не являются взаимно простыми с 35.

Составим таблицу квадратичных вычетов по модулю 35, обратных к ним значений по модулю 35 и их квадратных корней.

V	V^{-1}	S = sqrt (V^{-1})
1	1	1
4	9 *	3
9	4	2
11	16	4
16	11	9 **
29	29	8
Пояснения: * (4 * 9) mod 35 = 1; ** (9 * 9) mod 35 = 11		

Итак, сторона A получает открытый ключ, состоящий из K = 4 значений V:

[4, 11, 16, 29] .

Соответствующий секретный ключ, состоящий из K = 4 значений S:

[3, 4, 9, 8] .

Рассмотрим один цикл протокола.

1. Сторона A выбирает некоторое случайное число r = 16, вычисляет

$$x = 16^2 \bmod 35 = 11$$

и посылает это значение x стороне B.

2. Сторона B отправляет стороне A некоторую случайную двоичную строку

[1, 1, 0, 1] .

3. Сторона A вычисляет значение

$$y = r * (S_1^{b_1} * S_2^{b_2} * ... * S_K^{b_K}) \bmod n = 16 * (3^1 * 4^1 * 9^0 * 8^1) \bmod 35 = 31$$

и отправляет это значение y стороне B.

4. Сторона B проверяет, что

$$x = y^2 * (V_1^{b_1} * V_2^{b_2} * ... * V_K^{b_K}) \bmod n = 31^2 * (4^1 * 11^1 * 16^0 * 29^1) \bmod 35 = 11 .$$

Стороны A и B повторяют этот протокол t раз, каждый раз с разным случайным числом r, пока сторона B не будет удовлетворена.

При малых значениях величин, как в данном примере, не достигается настоящей безопасности. Но если n представляет собой число длиной 512 бит и более, сторона B не сможет узнать ничего о секретном ключе стороны A, кроме того факта, что сторона A знает этот ключ.

В этот протокол можно включить идентификационную информацию.

Пусть I - некоторая двоичная строка, представляющая идентификационную информацию о владельце карты (имя, адрес, персональный идентификационный номер, физическое описание) и о карте (дата окончания действия и т.п.). Эту информацию I формируют в Центре выдачи интеллектуальных карт по заявке пользователя A.

Далее используют одностороннюю функцию $f(\cdot)$ для вычисления f(I, j), где j - некоторое двоичное число, сцепляемое со строкой I. Вычисляют значения

$$V_j = f(I, j)$$

для небольших значений j, отбирают K разных значений j, для которых V_j являются квадратичными вычетами по модулю n. Затем для отобранных квадратичных вычетов V_j вычисляют наименьшие квадратные корни из V_j^{-1} (mod n). Совокупность из K значений V_j образует открытый ключ, а совокупность из K значений S_j - секретный ключ пользователя A.

Схема идентификации Гиллоу - Куискуотера

Алгоритм идентификации с нулевой передачей знания, разработанный Л.Гиллоу и Ж.Куискуотером, имеет несколько лучшие характеристики, чем предыдущая схема идентификации. В этом алгоритме обмены между сторонами A и B и аккредитации в каждом обмене доведены до абсолютного минимума - для каждого доказательства требуется только один обмен с одной аккредитацией. Однако обьем требуемых вычислении для этого алгоритма больше, чем для схемы Фейге-Фиата-Шамира.

Пусть сторона A - интеллектуальная карточка, которая должна доказать свою подлинность проверяющей стороне B. Идентификационная информация стороны A представляет собой битовую строку I, которая включает имя владельца карточки, срок действия, номер банковского счета и др. Фактически идентификационные данные могут занимать достаточно длинную строку, и тогда их хэшируют к значению I.

Строка I является аналогом открытого ключа. Другой открытой информацией, которую используют все карты, участвующие в данном приложении, являются модуль n и показатель степени V. Модуль n является произведением двух секретных простых чисел.

Секретным ключом стороны A является величина G, выбираемая таким образом. чтобы выполнялось соотношение

$$I * G^V \equiv 1 \pmod{n} .$$

Сторона A отправляет стороне B свои идентификационные данные I. Далее ей нужно доказать стороне B, что эти идентификационные данные принадлежат именно ей. Чтобы добиться этого, сторона A должна убедить сторону B, что ей известно значение G.

Вот протокол доказательства подлинности A без передачи стороне B значения G:

1. Сторона A выбирает случайное целое r, такое, что $1 < r \leq n - 1$. Она вычисляет

$$T = r^V \bmod n$$

и отправляет это значение стороне B.

2. Сторона B выбирает случайное целое d, такое, что $1 < d \leq n - 1$, и отправляет это значение d стороне A.

3. Сторона A вычисляет

$$D = r * G^d \bmod n$$

и отправляет это значение стороне B.

4. Сторона B вычисляет значение

$$T' = D^V I^d \bmod n .$$

Если $T \equiv T' \pmod n$, то проверка подлинности успешно завершена.

Математические выкладки, использованные в этом протоколе не очень сложны:

$$T' = D^V * I^d = (r * G^d)^V * I^d = r^V * G^{dV} * I^d = r^V * (I * G^V)^d = r^V \equiv T \pmod n .$$

поскольку G вычислялось таким образом, чтобы выполнялось соотношение

$$I*G^V \equiv 1 \pmod n$$

Компьютерная графология

Письмо, написанное обыкновенной ручкой, доносит до читателя в своеобразии завитушек букв что-то личностное. Профессиональный графолог, продравшись сквозь частокол закорючек, может многое рассказать об их авторе. А что компьютер может рассказать о человеке, стучащем по клавишам клавиатуры?

Проанализируем подход профессионального графолога к исследованию почерка на предмет поиска каких-либо аналогий к возможному исследованию клавиатурного почерка.

В 1930 г. в СССР вышла одна из первых и наиболее фундаментальных работ по графологии "Строение почерка и характер". Ее автор Д.М.Зуев-Инсаров очень убедительно продемонстрировал не только возможные результаты графологической экспертизы (определение пола, возраста, образования, рода занятий), но и достаточно внимания уделил именно экспериментальным основаниям графологии. Приведенная им таблица классификации формальных особенностей почерка по Л.Клагесу содержит следующие основные характеристики:

1. сила движения,
2. динамичность,
3. напряженность,
4. содержательность,
5. вытянутость букв,
6. наклон,
7. связеобразование,
8. степень связности букв в слове,
9. направление строки,
10. расположение текста,
11. быстрота,
12. способ держания ручки,
13. орудие письма,
14. равномерность,
15. соразмерность,
16. ритм,
17. выразительность.

Ясно, что в нашем случае многие из перечисленных характеристик становятся совершенно бессмысленными благодаря наличию стандартных клавиатер и драйверов к ним. Однако, упраздняя ряд характеристик, копмпьютер позволяет получить другие характеристики, ранее недоступные, в частности:

1. скорость набора различных слов относительно друг друга,
2. относительные интервалы между нажатиями клавиш, принадлежащих разным полям клавиатуры.

При этом новые характеристики даже более информативны по отношению к старым. Напрмер, характеристика "скорость набора различных слов относительно друг друга" при грамотно поставлееном тесте позволяет определить чуть ли неоднозначно сферу интересов тестируемого.

Ясно, что при наборе слов типа: программа, водород, реакция, соединение, манекен, строчка, выкройка, одежда, экскаватор и т.д., среднее время набора символов в словах "водород, реакция, соединение" у химика будет намного ниже, чем в остальных. Это понятно, потому что данные слова химику уже приходилось набирать на клавиатере и не один раз. Профессиональный модельер среди перечисленных слов лучше справится с набором "выкройка, одежда" и т.п.

Подобный подход можно использовать при создании программы "Детектор лжи". Мышечная память руки не контролируется сознанием и обязательно будет проявлять себя.

Исследования почерка могут способствовать диагностированию больного, т.к. характеристики почерка соответствующим образом изменяются во время болезни, приближаясь к нормальному по мере выздоровления. Можно предположить, что в будущем результаты анализа клавиатурного почерка будут играть не последнюю роль в постановке предварительного диагноза компьютером своему пользователю.

Таблица особенностей клавиатурного почерка:

Характеристика	**Формула**
Скорость набора тестового текста	$Tm = T/n$ где T - время набора, n - число символов
Скорость набора каждого слова тестового текста	$Ts[i] = T[i]/k[i]$ где T[i] - время набора i-того слова текста, k[i] - число символов i-того слова текста
Средний временной промежуток между словами	$T\text{п} = (T - \sum_{i=1}^{i=L} T[i])/L$ где L - число слов текста
Степень связности набора (вычисляется после исключения грубых ошибок)	$S = SQRT\dfrac{(t[j] - M)^2}{n-1}$ где t[j] - время между набором j и j+1 символа в слове тестового текста (пробелы исключаются) $M = \sum_{i=1}^{i=L} T[i])/n$
Общий рисунок почерка	$dX[i] = t[i] - t[i+1]$ для всех i.

Защита исходных текстов и двоичного кода

Крупные производители тиражного программного обеспечения отказались от защиты своих продуктов, сделав ставку на их массовое распространение. Пусть лицензионные копии приобретает лишь несколько процентов пользователей, но, если число этих копий измеряется миллионами, даже считанные проценты выливаются в солидную сумму, с лихвой окупающую разработку. Однако для небольших коллективов и индивидуальных программистов такая тактика неприемлема. Им необходимо предотвратить "пиратское" копирование своего продукта.

Предоставление исходных текстов часто является обязательным условием заказчика и закрепляется в контракте. Той же цели добиваются сторонники популярного движения открытых исходных текстов, ратующие за их свободное распространение. Кроме того, исходные тексты могут банальным образом украсть. Поэтому, стоит внимательно отнестись к организации технологической защиты интеллектуальной собственности.

Противодействие изучению исходных текстов

Вообще говоря, открытость исходных текстов - понятие расплывчатое. Вопрос: "Является ли бинарная программа в 1 Кбайт более открытой, чем миллион строк исходников без адекватной инфраструктуры и документации?" В самом деле, мало иметь исходный текст - в нем еще предстоит разобраться. Достаточно удалить все или, по крайней мере, большую часть комментариев, дать переменным и функциям бессмысленные, ничего не говорящие имена, как в программе не разберется и сам автор. А наличие и полнота комментариев к исходному тексту в контракте, как правило, не оговаривается. Получается, что контракт может быть формально выполнен, а предоставленный заказчику исходный текст практически бесполезен. "Практически" не означает "полностью": такой простой прием не позволит обеспечить абсолютную защиту.

Воспрепятствовать анализу можно отделением, абстрагированием алгоритма от языка реализации. Например, реализовать критичные для раскрытия компоненты на машине Тьюринга, а ее поддержку обеспечить на целевом языке. Уровней абстракции может быть несколько - чем их больше, тем труднее осуществлять анализ. Помимо машины Тьюринга для этой цели подходят стрелка Пирса, сети Петри и т.д. Такой подход дает превосходный результат, но требует глубоких математических знаний и значительных накладных расходов - программировать на машине Тьюринга намного сложнее, чем на ассемблере. Отдельный вопрос - эффективность полученной программы. Для многих проектов это неприемлемо, поэтому приходится использовать другие приемы.

Динамическое ветвление

Программу, состоящую из нескольких сотен тысяч строк, невозможно рассматривать как простую совокупность команд. На таком уровне детализации "за деревьями леса не видно". Сначала необходимо проанализировать взаимосвязь отдельных функций друг с другом, выделить интересующие фрагменты и лишь затем изучать реализацию самих функций. Чем выше степень дробления программы, тем труднее анализ. Элементарные функции, состоящие из десятка строк, сами по себе дают мало информации. Для формирования целостной картины необходимо рассмотреть

вызывающий их код, поднимаясь по иерархии вызовов до тех пор, пока не удастся реконструировать весь алгоритм целиком или, по крайней мере, охватить его ключевой фрагмент. Чтобы построить дерево вызовов, нужно уметь отслеживать перекрестные ссылки в обоих направлениях: определять, какой функции передается управление данным вызовом, и, наоборот, находить все вызовы, передающие управление данной функции.

Этому легко воспрепятствовать, воспользовавшись динамическим ветвлением - т.е. вычислением адреса перехода непосредственно перед передачей управления. Если функция возвращает не результат своей работы, а указатель на следующую выполняемую функцию (результат работы можно возвратить через аргументы, переданные по ссылке), то статический анализ не позволит определить порядок выполнения программы и построить дерево вызовов станет невозможно. Попутно исчезнет масса дублируемого кода, в частности, станет ненужной проверка результата завершения на корректность - в случае возникновения ошибки функция сама передаст управление нужной ветви программы.

Здесь стоит сделать одну оговорку. Язык Си не позволяет объявить функцию, возвращающую указатель на функции - это объявление рекурсивное. Приходится объявлять функцию, возвращающую бестиповой указатель void *. Аналогично - если функция в ходе своей работы вызывает какие-то другие функции, лучше делать это не напрямую, а передавать указатели на вызываемые функции как аргументы. Такой подход не только препятствует анализу, но еще и увеличивает гибкость программы, облегчая повторное использование старого кода в новых проектах - при должной культуре программирования каждая функция представляет "вещь в себе" и не привязана ко всем остальным.

Контекстная зависимость

Рассмотрим теперь другой прием, когда алгоритм работы большинства функций зависит от флага - глобальной переменной в пределах одного модуля. Если флаги изменяются в зависимости от результата, возвращаемого функцией, автоматически возникает контекстная зависимость (не слишком сильная, но это лучше, чем ничего). Работа одной функции становится зависимой от других, вызванных до нее. Анализ одной, отдельно взятой функции, становится многовариантным. Чтобы определить, что конкретно она делает, необходимо знать значение флагов, а значит - иметь представление о том, какие функции выполнялись до этого и какие именно данные они обрабатывали.

Опять замечание: в многопоточной среде глобальный флаг можно использовать только в том случае, если все потоки явно синхронизованы; в противном случае необходимо предоставить каждому потоку свой собственный экземпляр флага. Глобальный флаг требует особого внимания, малейшая небрежность приводит к трудноуловимым ошибкам. Разобраться в программе с множеством одновременно выполняемых потоков, манипулирующих с одной переменной, невероятно трудно; малейшая невнимательность или излишняя торопливость приводят к ошибкам анализа, затрудняющим понимание сути алгоритма.

Хуки

Хуки ("крючья") - изящный, но ныне практически забытый прием программирования. Его суть заключается в совмещении нескольких разнотипных данных в одном аргументе. Например, если значение аргумента по модулю меньше 0x400000, функция считает его непосредственным значением, в противном случае - указателем на

функцию, результат выполнения которой следует поставить на его место. Это увеличивает гибкость программы, но и затрудняет ее анализ, не позволяя быстро определить, происходит ли передача переменной по значению или по указателю. Указатели на переменную, в свою очередь, становятся неотличимы от указателей на функцию. Разумеется, отказ от строгой типизации может приводить к ошибкам, но вероятность их появления в тщательно продуманной программе невелика.

Снова замечание: хуки отрицательно сказываются на переносимости программ, поскольку представление указателей имеет свои особенности на каждой аппаратной платформе. Поэтому использовать их следует только в тех случаях, когда переносимость не требуется, или когда на всех выбранных платформах представление указателей унифицировано.

Существует еще множество других способов запутать того, кто пытается проанализировать исходный текст. Например, можно использовать совершенно корректные с точки зрения языка, но необычные конструкции, ставящие в тупик незнакомого с ними человека. Классический пример - перестановка индекса и имени массива в Си. С точки зрения языка, выражения "buff[666]" и "666[buf]" абсолютно равнозначны, но всякий ли об этом помнит?

Препроцессор Си также имеет свои тонкости. Возможно, самая популярная из них заключается в чувствительности к пробелам при объявлении макроса: "#define x(a,b) a+b" создаст макрос x(a,b), заменяющийся суммой своих аргументов, но "#define x (a,b) a+b" создаст макрос x, заменяющийся последовательностью "(a,b) a+b". Если не обратить внимание на лишний пробел, можно получить совсем не тот результат.

В той или иной степени эти, а также другие приемы используются в большинстве свободно распространяемых "открытых текстов". Последнее словосочетание заключено в кавычки для придания ему ироничного оттенка: одно лишь наличие исходного текста еще не обеспечивает открытости, - требуется, по меньшей мере, грамотно продуманная и качественно составленная документация, а еще лучше - опыт работы с этим текстом. Разобраться с исходными текстами редактора emacs или операционной системы Linux не намного проще, чем написать их "с нуля", - слишком уж скупы их разработчики на комментарии, а документация зачастую и вовсе отсутствует.

В большинстве случаев затраты на анализ чужих исходных текстов сравнимы или даже превышают стоимость заложенных в них алгоритмов (если стоящие алгоритмы там вообще есть), а их модификация просто убийственное занятие: внесенные изменения способны порождать ошибки в непредсказуемых местах и в непредсказуемом количестве. Словом, разумнее было бы говорить о закрытых исходных текстах.

Противодействие анализу двоичного кода

Отсутствие исходных текстов отнюдь не является непреодолимым препятствием для изучения и модификации кода приложения. Методики обратного проектирования позволяют автоматически распознавать библиотечные функции, локальные переменные, стековые аргументы, типы данных, ветвления, циклы и т.д. В недалеком будущем дизассемблеры, вероятно, научатся генерировать листинги, близкие по внешнему виду к языкам высокого уровня.

Сегодня трудоемкость анализа двоичного кода не настолько велика, чтобы надолго остановить злоумышленников. Огромное число постоянно совершаемых взломов - лучшее тому подтверждение. В идеальном случае знание алгоритма защиты не должно влиять на ее стойкость, но это достижимо далеко не всегда. Например, если производитель серверной программы решит установить в демонстрационной версии ограничение на количество одновременно обрабатываемых соединений, злоумышленнику достаточно найти инструкцию процессора, осуществляющую такую проверку и удалить ее. Модификации программы можно воспрепятствовать постоянной проверкой контрольной суммы, но опять-таки код, который вычисляет эту контрольную сумму и сверяет ее с эталоном, можно найти и удалить.

Сколько бы уровней защиты не предусмотреть, один или миллион, программа может быть взломана - это только вопрос времени и затраченных усилий. Но в отсутствии действенных правовых регуляторов защиты интеллектуальной собственности разработчикам приходится больше полагаться на стойкость своей защиты, чем на закон. Бытует мнение, что если затраты на нейтрализацию защитного механизма будут не ниже стоимости легальной копии, ее никто не будет взламывать. Это неверно. Материальный стимул - не единственное, что движет хакером. Гораздо более сильной мотивацией оказывается интеллектуальная борьба с автором защиты, спортивный азарт, любопытство, повышение своего профессионализма, да и просто интересное времяпровождение. Многие молодые люди могут неделями корпеть над отладчиком, снимая защиту с программы стоимостью в несколько долларов, а то и вовсе распространяемой бесплатно (например, диспетчер файлов FAR для жителей России абсолютно бесплатен, но это не спасает его от взлома).

Тем не менее, есть опыт создания защит, сломать которые почти невозможно. (Точнее, их взлом потребовал бы многих тысяч, а то и миллионов лет на типичном бытовом компьютере.)

Гарантированно воспрепятствовать анализу кода позволяет только шифрование программы. Но сам процессор не может непосредственно исполнять зашифрованный код, поэтому перед передачей управления его необходимо расшифровать. Если ключ содержится внутри программы, стойкость такой защиты близка к нулю. Все, чего может добиться разработчик, - затруднить поиск и получение этого ключа, тем или иным способом препятствуя отладке и дизассемблированию программы. Другое дело, если ключ содержится вне программы. Тогда стойкость защиты определяется стойкостью используемого криптоалгоритма (конечно, при условии, что ключ перехватить невозможно). Опубликованы и детально описаны многие криптостойкие шифры, взлом которых заведомо недоступен рядовым злоумышленникам.

В общих чертах идея защиты заключается в описании алгоритма с помощью некой математической модели, одновременно с этим используемой для генерации ключа. Разные ветви программы зашифрованы различными ключами, и чтобы вычислить этот ключ, необходимо знать состояние модели на момент передачи управления соответствующей ветви программы. Код динамически расшифровывается в процессе выполнения, а чтобы расшифровать его целиком, нужно последовательно перебрать все возможные состояния модели. Если их число будет очень велико (этого нетрудно добиться), восстановить весь код станет практически невозможно.

Для реализации этой идеи был создан специальный событийно-ориентированный язык программирования, где события являются единственным средством вызова подпрограммы. Каждое событие имеет свой код, один или несколько аргументов и какое

угодно количество обработчиков, а может не иметь ни одного (в последнем случае вызываемому коду возвращается ошибка).

На основе кода события и значения аргументов диспетчер событий генерирует три ключа - первый только на основе кода события, второй - только на основе аргументов, и третий на основе кода и аргументов. Затем он пытается полученными ключами последовательно расшифровать всех обработчиков событий. Если расшифровка происходит успешно, это означает, что данный обработчик готов обработать данное событие, и тогда ему передается управление.

Алгоритм шифрования должен быть выбран так, чтобы обратная операция была невозможна. При этом установить, какое событие данный обработчик обрабатывает, можно только полным перебором. Для блокирования возможности перебора в язык была введена контекстная зависимость - генерация дополнительной серии ключей, учитывающих некоторое количество предыдущих событий. Это позволило устанавливать обработчики на любые последовательности действий пользователя, например, на открытие файла с именем "Мой файл", запись в него строки "Моя строка" и переименование его в "Не мой файл".

Очевидно, перебор комбинаций всех событий со всеми возможными аргументами займет бесконечное время, а восстановить исходный код программы, защищенной таким образом, удастся не раньше, чем все ее ветви хотя бы однократно получат управление. Однако частота вызова различных ветвей не одинакова, а у некоторых и вовсе очень мала. Например, можно установить на слово "сосна", введенное в текстовом редакторе, свой обработчик, выполняющий некоторые дополнительные проверки на целостность кода программы или на лицензионную чистоту используемого ПО. Взломщик не сможет быстро выяснить, до конца ли взломана программа или нет. Ему придется провести тщательное и кропотливое тестирование, но даже после этого он не будет в этом уверен.

Таким же точно образом осуществляется ограничение срока службы демонстрационных версий. Разумеется, обращаться к часам реального времени бесполезно, их очень легко перевести назад, вводя защиту в заблуждение. Гораздо надежнее опираться на даты открываемых файлов: даже если часы переведены, созданные другими пользователями файлы в большинстве случаев имеют правильное время. Но взломщик не сможет узнать ни алгоритм определения даты, ни саму дату окончания использования продукта. Впрочем, дату в принципе можно найти и полным перебором, но что это дает? Модификации кода воспрепятствовать очень легко: достаточно, чтобы длина зашифрованного текста была чувствительна к любым изменениям исходного. В этом случае взломщик не сможет подправить "нужный" байт в защитном обработчике и вновь зашифровать его. Придется расшифровывать и вносить изменения во все остальные обработчики (при условии, что они контролируют смещение, по которому расположены), а это невозможно, поскольку соответствующие им ключи заранее неизвестны.

Существенными недостатками предлагаемого решения являются низкая производительность и высокая сложность реализации. Если со сложностью реализации можно смириться, то скорость налагает серьезные ограничения на сферу применения. Впрочем, можно значительно оптимизировать алгоритм или оставить все критичные к быстродействию модули незашифрованными (или расшифровывать каждый обработчик только один раз). Интересно другое: действительно ли эта технология позволяет создавать принципиально неизучаемые приложения или в приведенные рассуждения вкралась ошибка?

Заключение

Целесообразность защиты исходных текстов ограничивается конкурентной борьбой - при прочих равных условиях клиент всегда выбирает незащищенный продукт, даже если защита не ущемляет его прав. Сегодня спрос на программистов превышает предложение, но в отдаленном будущем тенденция изменится и разработчикам придется либо сговориться, либо полностью отказаться от защит, а специалисты по защите будут вынуждены искать себе другую работу. Это не значит, что данная статья бесполезна. Напротив, полученные знания целесообразно применять, не откладывая, пока в защите исходных текстов еще не отпала необходимость.

Литература

[1] Николай Безруков, "Разработка программ с открытыми исходниками как особый вид научных исследований"

[2] Манфред Брой, "Информатика. Основополагающее введение. Структуры систем и системное программирование. Часть III" // М.: Диалог-МИФИ, 1996

ТРИ КЛЮЧА

Три ключа необходимы для отказа от явной проверки значения аргументов, которую легко обнаружить анализирующему лицу. Например, пусть событие KEY (key_code) генерируется при каждом нажатии клавиш. Тогда обработчик, считывающий входную информацию, должен привязываться только к коду события (KEY) и получать введенный символ в виде аргумента. Если одна из клавиш или их комбинация зарезервирована для специальной цели (например, задействует некоторые дополнительные функции в программе), то ее обработчик может привязываться одновременно к коду события (KEY) и коду клавиши (key_code), не опасаясь за свое раскрытие, так как правильный ключ дает лишь единственная комбинация KEY и key_code, а явная проверка на соответствие нажатого символа секретному коду отсутствует.

Привязка к аргументам позволяет отлавливать искомые последовательности в потоке данных независимо от того, каким образом они получены. Например, ожидающая пароля MyGoodPassword процедура аутентификации не интересуется, введен ли он с клавиатуры, получен ли с удаленного терминала, загружен ли из файла и т.д. Такой подход значительно упрощает программирование и уменьшает зависимость одних модулей от других. Программа представляет собой совокупность обработчиков, автоматически коммутируемых возникающими событиями. Никакого детерминизма. Это чем-то напоминает взаимодействие биологической клетки с окружающей средой и в скором будущем может стать довольно перспективным направлением.

СТРУКТУРА КОМАНД INTEL 80x86

Победа в борьбе, как правило, достается тому, кто лучше знает противника. Успешно защититься от хакера сможет лишь тот, кто знает не меньше хакера. Каждый уважающий себя хакер должен понимать, как строятся команды, чтобы с легкостью манипулировать им, занимаясь дизассемблированием. Следовательно и тот, кто хочет

защитить свою программу от дизассемблирования, не обойдется без понимания того, как процессор интерпретирует команды.

Формат инструкции архитектуры Intel приведен на рис.1. Это схематичное представление. Пока без размерностей и пояснений.

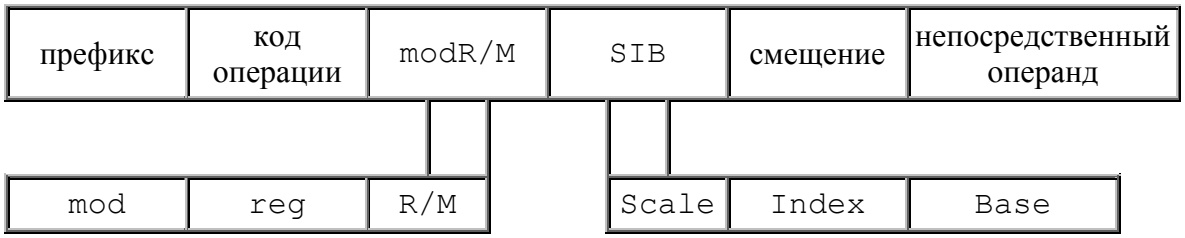


Рис.1. Формат команд процессора x86 фирмы Intel

Кроме поля кода операции все остальные поля являются необязательными, т.е. в одних командах могут присутствовать, а в других нет.

Префикс

Префиксы делятся на четыре группы:

- блокировки и повторения:

```
0F0h   LOCK префикс
0F2h   REPNZ(только для строковых инструкций)
0F3h   REP  (только для строковых инструкций)
```

- переопределения сегмента:

```
02Eh   CS:
036h   SS:
03Eh   DS:
026h   ES:
064h   FS:
065h   GS:
```

- переопределения размеров операндов: `066h`
- переопределение размеров адреса: `067h`

Если используется более одного префикса из той же самой группы, то действие команды не определено и по-разному реализовано на разных типах процессоров.

Префикс переопределения размера операндов используется в 16-разрядном режиме для манипуляции с 32-битными операндами и наоборот. При этом он может стоять перед любой командой, например, `0x66:CLI` будет работать! А почему бы и нет? Интересно, но отладчики этого не учитывают и отказываются работать. То же относится и к дизассемблерам, к примеру IDA Pro:

```
seg000:0100   start   proc near
seg000:0100   db      66h
seg000:0100   cli
seg000:0102   db      67h
seg000:0102   sti
seg000:0104   retn
```

На этом же основан один очень любопытный прием противодействия отладчикам, в том числе и знаменитому отладчику-эмулятору Cup386. Рассмотрим, как работает конструкция `0x66:RETN`. Казалось бы, раз команда `RETN` не имеет операндов, то префикс `0x66` можно просто игнорировать. Но, на самом деле, все не так просто. `RETN` работает с неявным операндом-регистром `ip/eip`. Именно его и изменяет префикс. Разумеется, в реальном и 16-разрядном режиме указатель команд всегда обрезается до 16 бит, и поэтому, на первый взгляд, возврат сработает корректно. Но стек-то окажется несбалансированным! Из него вместе одного слова взяли целых два! Так нетрудно получить и исключение `0Ch` - исчерпание стека.

Любопытно, какой простой, но какой надежный прием. Впрочем, следует признать, что перехват `INT 0Ch` под операционной системой Windows бесполезен, и, не смотря на все ухищрения, приложение, породившие такое исключение, будет безжалостно закрыто. Однако, в реальном режиме это работает превосходно.

Еще интереснее получится, если попытаться исполнить в 16-разрядном сегменте команду `CALL`. Если адрес перехода лежит в пределах сегмента, то ничего необычно ожидать не приходится. Инструкция работает нормально. Все чудеса начинаются, когда адрес выходит за эти границы. В защищенном 16-разрядном режиме при уровне привилегий `CL0` с большой вероятностью регистр `EIP` "обрежется" до шестнадцати бит, и инструкция сработает (но, похоже, что не на всех процессорах). Если уровень не `CL0`, то генерируется исключение защиты `0Dh`. В реальном же режиме эта инструкция может вести себя непредсказуемо. Хотя в общем случае должно генерироваться прерывание `INT 0Dh`. В реальном режиме его нетрудно перехватить и совершить дальний 'far'-переход в требуемый сегмент. Так поступает, например, операционная система Касперского OS\7R, дающая в реальном режиме плоскую (flat) модель памяти. Разумеется, такой поворот событий не может пережить ни один отладчик. Ни трассировщики реального режима, ни v86, ни protect-mode debugger, ни даже эмуляторы с этим справиться не в состоянии.

Одно плохо - все эти приемы не работают под Windows и другими операционными системами. Это вызвано тем, что обработка исключения типа "Общее нарушение защиты" всецело лежит на ядре операционной системы, что не позволяет приложениям распоряжаться им по своему усмотрению. Забавно, но в режиме эмуляции MS-DOS некоторые EMS-драйверы ведут себя в этом случае совершенно непредсказуемо. Часто при этом они не генерируют ни исключения `0Ch`, ни `0Dh`. Это следует учитывать при разработке защит, основанных на приведенных выше приемах.

Обратим внимание так же и на последовательности типа `0x66 0x66 [xxx]`. Хотя фирма Intel не гарантирует корректную работу своих процессоров в такой ситуации, но фактически все они правильно интерпретируют такую ситуацию. Иное дело некоторые отладчики и дизассемблеры, которые спотыкаются и начинают некорректно вести себя.

Есть еще один интересный момент связанный с работой декодера микропроцессора.

Декодер за один раз считывает только 16 байт и, если команда "не уместится", то он просто не сможет считать "продолжение" и сгенерирует исключение "Общее нарушение защиты". Однако, иначе ведут себя эмуляторы, которые корректно обрабатывают "длинные" инструкции.

Впрочем, все это очень процессорно-зависимо. Никак не гарантируется сохранение и поддержание этой особенности в будущих моделях, и поэтому злоупотреблять этим не стоит, иначе ваша защита откажется работать.

Префиксы переопределения сегмента могут встречаться перед любой командой, в том числе и не обращающейся к памяти, например, `CS:NOP` вполне успешно выполнится. А вот некоторые дизассемблеры сбиться могут. К счастью, IDA Pro к ним не относится. Самое интересное, что комбинация `DS: FS: FG: CS: MOV AX,[100]` работает вполне нормально (хотя это и не гарантируется фирмой Intel). При этом последний префикс в цепочке перекрывает все остальные. Некоторые отладчики, наоборот, ориентируются на первый префикс в цепочке, что дает неверный результат. Этот пример хорош тем, что великолепно выполняется под Windows и другими операционными системами. К сожалению, на декодирование каждого префикса тратится один такт, и все это может медленно работать.

Код операции

Само поле кода операции занимает восемь бит и чаще всего имеет следующий формат (рис. 2):

<table>
<tr><td colspan="5" rowspan="3">код операции</td><td colspan="2">направ-
ление</td><td>размер</td></tr>
<tr><td colspan="3">регистр</td></tr>
<tr><td colspan="4">условие</td><td>инверсия</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

Рис. 2. Формат поля "Код операции"

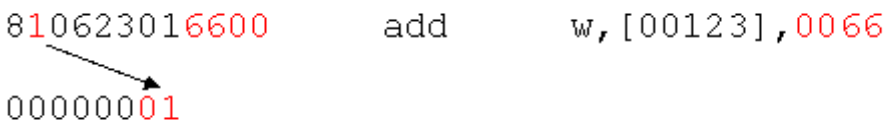
Поле размера указывает на размер операндов. Это поле равно 0, если операнды имеют размер в один байт. Это поле равно 1, если операнды имеют размер в слово (двойное слово в 32-битном режиме или в 16-битном режиме с префиксом `0x66`).

Поле направления обозначает, какой из операндов будет приемником. Если направление равно 0, то приемник - правый операнд, если же направление равно 1, то приемник - левый операнд. На примере инструкции `mov bx,dx` видно, как можно поменять местами операнды, изменив всего один бит:

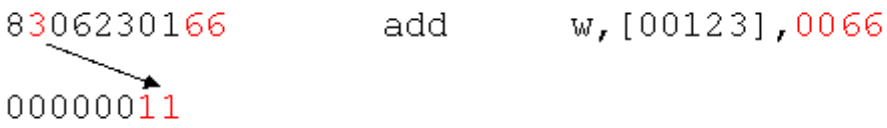
```
8BDA      mov   bx,dx
10001011b
89DA      mov   dx,bx
10001001b
```

Однако, давайте задумаемся, как поле направления будет вести себя, когда один из операндов непосредственное значение? Разумеется, что оно не может быть приемником и

независимо от содержимого этого бита будет только источником. Инженеры Intel учли такую ситуацию и нашли оригинальное применение, часто экономящее в лучшем случае целых три байта. Рассмотрим ситуацию, когда операнду размером слово или двойное слово присваивается непосредственное значение по модулю меньшее `0100h`. Ясно, что значащим является только младший байт, а стоящие слева нули по правилам математики можно отбросить. Но попробуйте объяснить это процессору! Потребуется пожертвовать хотя бы одним битом, чтобы указать ему на такую ситуацию. Вот для этого и используется бит направления. Рассмотрим следующую команду:



Если теперь флаг направления установить в единицу, то произойдет следующие:



Таким образом, мы экономим один байт в 16-разрядном режиме и целых три - в 32-разрядом. Этот факт следует учитывать при написании самомодифицирующегося кода. Большинство ассемблеров генерируют второй (оптимизированный) вариант, и длина команды оказывается меньше ожидаемой. На этом, кстати, основан очень любопытный прием против отладчиков. Посмотрите на следующий пример:

```
00000100: 810600010200   add   w,[00100],00002
00000106: B406           mov   ah,006
00000108: B207           mov   dl,007
0000010A: CD21           int   021
0000010C: C3             retn
```

После выполнения инструкция в строке 0100h приобретет следующий вид:

```
00000100: 8206000102   add   b,[00100],002
00000105: 00B406B2     add   [si],[0B206],dh
          ↑
          ip после выполнения команды процессором
```

То есть, текущая команда станет на байт короче! И "отрезанный" ноль теперь стал частью другой команды! Но при выполнении на "живом" процессоре такое не произойдет, т.к. следующие значение `ip` вычисляется еще **до выполнения** команды на стадии ее декодирования.

Совсем другое дело отладчики, и особенно отладчики-эмуляторы, которые часто вычисляют значение `ip` **после выполнения** команды (это легче запрограммировать). В результате чего наступает крах. Маленькая тонкость - **до** или **после** оказалась роковой, и вот вам в подтверждение дамп экрана:

```
CS:0100  8306000102   add   word ptr [0100],02 | ax 0000 c=0
CS:0105  00B406B2     add   [si-4DFA],dh       | bx 0000 z=0
CS:0109  07           pop   es                 | ex 0000 s=0
CS:010A  CD21         int   21                 | dx 0000 o=0
```

```
CS:010C  C3            ret                          si 0000 p=0
```

Заметим, что этот прием может быть бессилен против трассирующих отладчиков (debug.com, DeGlucker, Cup386), поскольку значение `ip` за них вычисляет процессор и делает это правильно.

Однако, на "обычные" отладчики управа всегда найдется, а с эмуляционными справиться гораздо труднее, и приведенный пример один из немногих, способных возыметь действие на виртуальный процессор.

Формат кода операции разнится от одной команды к другой, однако, можно выделить и некоторые общие правила. Практически для каждой команды, если регистром-приемником фигурирует `AX`(`AL`), существует специальный однобайтовый код, который содержится в трех младших битах регистра-источника. Этот факт следует учитывать при оптимизации. Так, среди двух инструкций:

`XCHG AX,BX` и `XCHG BX,DX`

следует всегда выбирать первую, т.к. она на байт короче. (Кстати, инструкция `XCHG AX,AX` более известна нам как `NOP`. О достоверности этого факта часто спорят в конференциях, но на странице 340 руководства №24319101 "Instruction Set Reference Manual" фирмы Intel это утверждается совершенно недвусмысленно. Любопытно, что, выходит, никто из многочисленных спорщиков не знаком даже с оригинальным руководством производителя).

Для многих команд условного перехода четыре младших бита обозначают ***условие операции***. Точнее говоря, условие задается в битах 1-3, а установка бита 0 приводит к его инверсии (таблица 1).

Таблица 1

Код	Мнемоника	Условие
0000	0	Переполнение
0010	B,NAE	Меньше
0100	Z	Равно
0110	BE,NA	Меньше или равно
1000	S	Знак
1010	P,PE	Четно
1100	L,NGE	Меньше (знаковое)
1110	LE,NG	Меньше или равно (знаковое)

Как видим, условий совсем немного, и проблем с их запоминанием обычно не возникает. Теперь уже не нужно мучительно вспоминать '`jz`' - это `74h` или `75h`. Так как младший бит первого равен нулю, то '`jz`' - это `74h`, а '`jnz`', соответственно, `75h`.

Байт modR/M

Далеко не все коды операций смогли поместиться в первый байт. Инженеры Intel задумались о поиске дополнительного места для размещения еще нескольких бит и обратили внимание на байт `modR/M`. Подробнее он описан ниже, а пока рассмотрим приведенный выше рисунок (рис. 1). Трех-битовое поле `reg`, содержащие регистр-источник, очевидно, не используется, если вслед за ним идет непосредственный операнд. Так почему бы его не использовать для задания кода операции? Однако, процессору требуется указать на такую ситуацию. Это делает префикс `0Fh`, размещенный в первом байте кода. Да, именно префикс, хотя документация Intel этого прямо и не подтверждает. При этом на не-MMX процессорах для его декодирования требуется дополнительный такт. Intel же предпочитает называть первый байт основным, а второй уточняющим кодом операции. Заметим, что это же поле используют многие инструкции, оперирующие одним операндом (`jmp`, `call`). Все это очень сильно затрудняет написание собственного ассемблера/дизассемблера, но зато дает простор для создания самомодифицирующегося кода и, кроме того, вызывает восхищение инженерами Intel, до минимума сокративших размеры команд. Конечно, это досталось весьма непростой ценой. И далеко не все дизассемблеры работают правильно. С другой стороны именно благодаря этому и существуют защиты, успешно противостоящие им.

Избежать проблем можно, лишь четко представляя себе сам принцип кодировки команд, а не просто работая с "мертвой" таблицей кодов операций, которую многие авторы вводят в дизассемблер и на том успокаиваются, так как внешне все работает правильно.

К тонкостям кодирования команд мы еще вернемся, а пока приготовимся к разбору поля `modR/M`. Два трехбитовых поля могут задавать код регистра общего назначения по следующей таблице (таблица 2):

Таблица 2

Код	**8 бит операнд**	**16 бит операнд**	**32 бит операнд**
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Опять можно восхищаться лаконичностью инженеров Intel, которые ухитрились всего в трех битах закодировать столько регистров. Это, кстати, объясняет, почему нельзя выборочно обращаться к старшим и младшим байтам регистров `SP`, `BP`, `SI`, `DI` и, аналогично, к старшему слову всех 32-битных регистров. Во всем "виновата" оптимизация и архитектура команд. Просто нет свободных полей, в которые можно было бы "вместить" дополнительные регистры. Сегодня мы вынуждены расхлебывать результаты архитектурных решений, выглядевшими такими удачными всего лишь десятилетие назад.

Обратите внимание на порядок регистров: AX, CX, DX, BX, SP, BP, SI, DI. Немного не по алфавиту, верно? И особенно странно в этом отношении выглядит регистр BX. Но, если понять причины, то никакой нужны запоминать это исключение не будет, т.к. все станет на свои места: BX - это индексный регистр, и первым стоит среди индексных.

Таким образом, мы уже можем "вручную" без дизассемблера распознавать в шестнадцатеричном дампе регистры-операнды. Очень неплохо для начала! Или писать самомодифицирующийся код. Например:

```
00000000: 800E070024   or    b, [00007] ,024
00000005: FA           cli
00000006: 33C0         xor   ax, ax
00000008: FB           sti
```

Он изменит 6-ю строку на XOR SP,SP. Это "завесит" многие отладчики, и, кроме того, не позволит дизассемблерам отслеживать локальные переменные адресуемые через SP. Хотя IDA Pro и позволяет скорректировать стек вручную, для этого надо сначала понять, что SP обнулился. В приведенном примере это очевидно (но в глаза, кстати, не бросается), а если это произойдет в многопоточной системе? Тогда изменение кода очень трудно будет отследить, особенно в листинге дизассемблера. Однако, нужно помнить, что самомодифицирующийся код все же уходит в историю. Сегодня он встречается все реже и реже.

2-битная кодировка

```
00 ES
01 CS
10 SS
11 DS
```

3-битная кодировка

```
000 ES
001 CS
010 SS
011 DS
100 FS
101 GS
110 Зарезервировано
111 Зарезервировано
```

Первоначально сегментные регистры кодировались всего двумя битами и этого с вполне хватало, т.к. их было всего четыре. Позже, когда количество их увеличилось, перешли на трехбитную кодировку. При этом две кодовые комбинации (110b и 111b) в настоящее время не применяются и вряд ли будут добавлены в ближайшем будущем. Но что же будет, если попытаться их использовать? Генерация INT 06h. А вот отладчики-эмуляторы могут вести себя странно. Одни не генерируют при этом прерывания, чем себя и выдают, а другие - ведут себя непредсказуемо, т.к. при этом требуемый регистр может находится в области памяти, занятой другой переменной (это происходит, когда ячейка памяти определяется по индексу регистра, при этом считываются три бита и суммируются с базой, но никак не проверяются пределы).

Поведение дизассемблеров так же разнообразно. Вот, например,

Hiew:

```
00000000: 8E       ???
00000001: F8       clc
00000002: C3       retn
```

gview:
```
00000000: 8EF8      mov !s,ax
00000002: C3        ret
```
IDA Pro:
```
seg000:0100 start  db  8 Eh
seg000:0101        db  0F8h
seg000:0102        db  0C3h
```

Кстати, IDA Pro вообще отказывается анализировать весь последующий код. Как это можно использовать? Да очень просто - если эмулировать еще два сегментных регистра в обработчике `INT 06h`, то очень трудно это будет как отлаживать, так и дизассемблировать программу. Однако, это опять-таки не работает под Win32!

Управляющие/отладочные регистры кодируются нижеследующим образом:

	Управляющие регистры	**Отладочные регистры**
000	`CR0`	`DR0`
001	Зарезервировано	`DR1`
010	`CR2`	`DR2`
011	`CR3`	`DR3`
100	`CR4`	Зарезервировано
101	Зарезервировано	Зарезервировано
110	Зарезервировано	`DR6`
111	Зарезервировано	`DR7`

Заметим, что коды операций `mov`, манипулирующих этими регистрами, различны, поэтому-то и возникает кажущееся совпадение имен. С управляющими регистрами связана одна любопытная мелочь. Регистр `CR1`, как известно большинству, в настоящее время зарезервирован и не используется. Во всяком случае, так написано в русскоязычной документации. На самом деле регистр `CR1` просто не существует! И любая попытка обращения к нему вызывает генерацию исключение `INT 06h`. Например, cир386 в режиме эмуляции процессора этого не учитывает и неверно исполняет программу. А все дизассемблеры, за исключением IDA Pro, неправильно дизассемблируют этот несуществующий регистр:

IDA Pro:
```
seg000:0100 start    db   0Fh
seg000:0101          db   20h
seg000:0102          db   0C8h
seg000:0103          db   0C3h
```
Sourcer:
```
4305:0100 start:
4305:0100 0F 20 C8   mov  eax,crl
4305:0103 C3         retn
```
или:
```
4305:0100 start:
4305:0100 0F 20 F8   mov  eax, cr7
4305:0103            C3   retn
```

Все эти команды на самом деле не существуют и приводят к вызову прерывания INT 06h. Не так очевидно, правда? И еще менее очевидно обращение к регистрам DR4-DR5. При обращении к ним исключения не генерируется.

Между прочим, IDA Pro 3.84 дезассемблирует не все регистры. Зато великолепно их ассемблирует (кстати, ассемблер этот был добавлен другим разработчиком).

Пользуясь случаем, акцентируем внимание на сложностях, которые подстерегают при написании собственного ассемблера (дизассемблера). Документация Intel местами все же недостаточно ясна (как в приведенном примере), и неаккуратность в обращении с ней приводит к ошибкам, которыми может воспользоваться разработчик защиты против хакеров.

Теперь перейдем к описанию режимов адресации микропроцессоров Intel. Тема очень интересная и познавательная не только для оптимизации кода, но и для борьбы с отладчиками.

Первым ключевым элементом является байт modR/M.

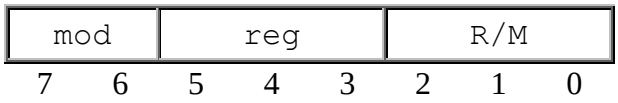


Рис. 3. Формат байта modR/M

Если mod содержит 11b, то два следующих поля будут представлять собой регистры. (Это так называемая регистровая адресация). Например:

```
00000000: 33C3   xor     ax, bx
                         11 000 011
00000000: 32C3   xor     al, bl
                         11 000 011
```

Как отмечалось выше, по байту modR/M нельзя точно установить регистры. В зависимости от кода операции и префиксов размера операндов, результат может коренным образом меняться.

Биты 3-5 могут вместо определения регистра уточнять код операции (в случаи, если один из операндов представлен непосредственным значением). Младшие три бита всегда либо регистр, либо способ адресации, что зависит от значения mod. А вот биты 3-5 никак не зависят от выбранного режима адресации и задают всегда либо регистр, либо непосредственный операнд.

Формат поля R/M, строго говоря, не документирован, однако достаточно очевиден, что позволяет избежать утомительного запоминания совершенно нелогичной на первый взгляд таблицы адресаций (таблица 3).

X	X	X

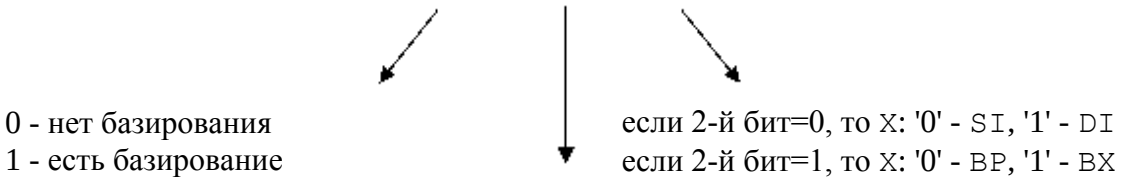


Рис. 4. Формат поля R/M.

Возможно, кому-то эта схема покажется витиеватой и трудной для запоминания, но зубрить все режимы без малейшего понятия механизма их взаимодействия еще труднее, кроме того, нет никакого способа себя проверить и проконтролировать ошибки.

Действительно, в поле R/M все три бита тесно взаимосвязаны, в отличии от поля mod, которое задает длину следующего элемента в байтах.

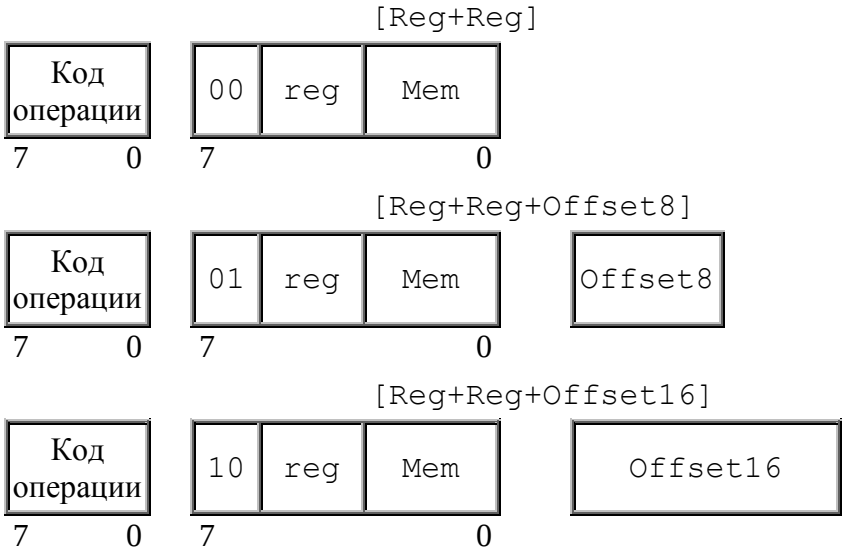


Рис. 5. Формат команды в зависимости от поля mod.

Разумеется, не может быть смещения Offset 12, (т.к. процессор не оперирует с полуторными словами), а комбинация 11 указывает на регистровую адресацию.

Может возникнуть вопрос, как складывать с 16-битным регистром 8-битное смещение? Конечно, непосредственному сложению мешает несовместимость типов, поэтому процессор сначала расширяет 8 бит до слова с учетом знака. Поэтому, диапазон возможных значений составляет от -127 до 127 (или от −0x7F до 0x7FF).

Все вышесказанное проиллюстрировано в приведенной ниже таблице 3. Обратим внимание на любопытный момент - адресация типа [BP] отсутствует. Ее ближайшим эквивалентом является [BP+0]. Отсюда следует, что для экономии следует избегать непосредственного использования BP в качестве индексного регистра. BP может быть

только базой. И `mov ax,[bp]` хотя и воспринимается любым ассемблером, но ассемблируется в `mov ax,[bp+0]`, что на байт длиннее.

Исследовав приведенную ниже таблицу 3, можно прийти к выводу, что адресация в процессоре 8086 была достаточно неудобной. Сильнее всего сказывалось то ограничение, что в качестве индекса могли выступать только три регистра (`BX`, `SI`, `DI`), когда гораздо чаще требовалось использовать для этого `CX` (например, в цикле) или `AX` (как возвращаемое функцией значение).

Поэтому, начиная с процессора 80386 (для 32-разрядного режима), концепция адресаций была пересмотрена. Поле `R/M` стало всегда выражать регистр независимо от способа его использования, чем стало управлять поле `mod`, задающие, кроме регистровой, три вида адресации:

mod	адрес
00	[Reg]
01	[Reg + 08]
10	[Reg + 32]
11	Reg

Видно, что поле `mod` по-прежнему выражает длину следующего поля - смещения, разве что с учетом 32-битного режима, где все слова расширяются до 32 бит.

Напомним, что с помощью префикса `0x67` можно и в 16-битном режиме использовать 32-битный режимы адресации, и наоборот. Однако, при этом мы сталкиваемся с интересным моментом - разрядность индексных регистров остается 32-битной и в 16-битном режиме!

В реальном режиме, где нет понятия границ сегментов, это действительно будет работать так, как выглядит, и мы сможем адресовать первые 4 мегабайта памяти (32 бита), что позволит преодолеть печально известное ограничение размера сегмента 8086 процессоров в 64К. Но такие приложения окажутся нежизнеспособными в защищенном или V86 режиме. Попытка вылезти за границу 64К сегмента вызовет исключение `0Dh`, что приведет к автоматическому закрытию приложения, скажем, под управлением Windows. Аналогично поступают и отладчики (в том числе и многие эмуляторы, включая Cup386).

Сегодня актуальность этого приема, конечно, значительно снизилась, поскольку "голый DOS" практически уже не встречается, а режим его эмуляции Windows крайне неудобен для пользователей.

Таблица 3

16-разрядный режим			32-разрядный режим		
адрес	**mod**	**R/M**	**адрес**	**mod**	**R/M**
[BX+SI]		000	[EAX]		000
[BX+DI]		001	[ECX]		001
[BP+SI]		010	[EDX]		010
[BP+DI]	00	011	[EBX]	00	011
[SI]		100	[-][-]		100
[DI]		101	смещ32		101
смещ16		110	[ESI]		110

Таблица 3

16-разрядный режим			32-разрядный режим		
адрес	**mod**	**R/M**	**адрес**	**mod**	**R/M**
[BX]		111	[EDI]		111
[BX+SI]+смещ8		000	смещ8[EAX]		000
[BX+DI]+смещ8		001	смещ8[ECX]		001
[BP+SI]+смещ8		010	смещ8[EDX]		010
[BP+DI]+смещ8		011	смещ8[EBX]		011
[SI]+смещ8	01	100	смещ8[-][-]	01	100
[DI]+смещ8		101	смещ8[ebp]		101
[BP]+смещ8		110	смещ8[ESI]		110
[BX]+смещ8		111	смещ8[EO1]		111
[BX+SI]+смещ16		000	смещ32[EAX]		000
[BX+DI]+смещ16		001	смещ32[ECX]		001
[BP+SI]+смещ16		010	смещ32[EBX]		010
[BP+DI]+смещ16		011	смещ32[EBX]		011
[SI]+смещ16	10	100	смещ32[-][-]	10	100
[DI]+смещ16		101	смещ8[ebp]		101
[BP]+смещ16		110	смещ8[ESI]		110
[BX]+смещ16		111	смещ8[EDI]		111

Изучив эту таблицу, можно прийти к заключению, что система адресации 32-битного режима крайне скудная и ни на что серьезное ее не хватит. Однако, это не так. В 386+ появился новый байт SIB, который расшифровывается как Scale-Index Base.

Процессор будет ждать его вслед за R/M всякий раз, когда последний равен 100b. Эти поля отмечены в таблице как [-]. SIB хорошо документирован, и назначения его полей показаны на рисунке 6. Нет совершенно никакой необходимости зазубривать таблицу адресаций

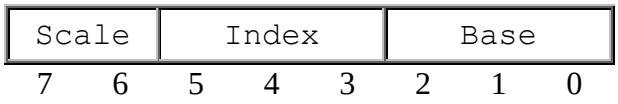


Рис. 6. Формат поля SIB.

Здесь Base - базовый регистр, Index - индексный, а два байта Scale - степень двойки для масштабирования. Поясним введенные термины. Ну, что такое индексный регистр, понятно всем. Например, SI. Теперь же в качестве индексного можно использовать любой регистр. За исключением, правда, SP; впрочем, можно выбирать и его, но об этом позже.

Базовый регистр - это регистр, который суммируется с индексным. Например, [BP+SI]. Аналогично, базовым теперь может быть любой регистр. При этом есть возможность в качестве базового выбрать SP. Заметим, что если мы выберем этот регистр в качестве индексного, то вместо SP получим - "никакой": в этом случае адресацией будет управлять только базовый регистр.

Таблица 4

Base			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP* 101	ESI 110	EDI 111
Index		**S**	**шестнадцатеричное значение SIB**							
[EAX]	000	00	00	08	10	18	20	28	30	38
[ECX]	001		01	09	11	19	21	29	31	39
[EDX]	010		02	0A	12	1A	22	2A	32	3A
[EBX]	011		03	0B	13	1B	23	2B	33	3B
нет	100		04	0C	14	1C	24	2C	34	3C
[EBP]	101		05	0D	15	1D	25	2D	35	3D
[ESI]	110		06	0E	16	1E	26	2E	36	3E
[EDI]	111		07	0F	17	1F	27	2F	37	3F
[EAX*2]	000	01	40	48	50	58	60	68	70	78
[ECX*2]	001		41	49	51	59	61	69	71	79
[EDX*2]	010		42	4A	52	5A	62	6A	72	7A
[EBX*2]	011		43	4B	53	5B	63	6B	73	7B
нет	100		44	4C	54	5C	64	6C	74	7C
[EBP*2]	101		45	4D	55	5D	65	6D	75	7D
[ESI*2]	110		46	4E	56	5E	66	6E	76	7E
[EDI*2]	111		47	4F	57	5F	67	6F	77	7F
[EAX*4]	000	10	80	88	90	98	A0	A8	B0	B8
[ECX*4]	001		81	89	91	99	A1	A9	B1	B9
[EDX*4]	010		82	8A	92	9A	A2	AA	B2	BA
[EBX*4]	011		83	8B	93	9B	A3	AB	B3	BB
нет	100		84	8C	94	9C	A4	AC	B4	BC
[EBP*4]	101		85	8D	95	9D	A5	AD	B5	BD
[ESI*4]	110		86	8E	96	9E	A6	AE	B6	BE
[EDI*4]	111		87	8F	97	9F	A7	AF	B7	BF
[EAX*8]	000	11	C0	C8	D0	D8	E0	E8	F0	F8
[ECX*8]	001		C1	C9	D1	D9	E1	E9	F1	F9
[EDX*8]	010		C2	CA	D2	DA	E2	EA	F2	FA
[EBX*8]	011		C3	CB	D3	DB	E3	EB	F3	FB
нет	100		C4	CC	D4	DC	E4	EC	F4	FC
[EBP*8]	101		C5	CD	D5	DD	E5	ED	F5	FD
[ESI*8]	110		C6	CE	D6	DE	E6	EE	F6	FE
[EDI*8]	111		C7	CF	D7	DF	E7	EF	F7	FF

Масштабирование - это уникальная возможность умножать индексный регистр на 1, 2, 4, 8 (т.е. степень двойки, которая задается в поле `Scale`). Это очень удобно для доступа к различным структурам данных. При этом индексный регистр, являющийся одновременно и счетчиком цикла, будет указывать на следующий элемент структуры даже при единичном шаге цикла (что чаще всего и встречается). В таблице 4 показаны все возможные варианты значений байта `SIB`.

Если при этом в качестве базового регистра будет выбран `EBP`, то полученный режим адресации будет зависеть от поля `mod` предыдущего байта. Возможны следующие варианты:

mod	**действие**
00	смещение32[index] - регистр EBP в адресации не участвует!
01	смещение8[EBP][index]
10	смещение32[EBP][index]

Итак, мы полностью разобрались с кодировкой команд. Осталось лишь выучить непосредственно саму таблицу кодов, и можно отправляться в длинный и тернистый путь написания собственного дизассемблера.

За это время, надеюсь, у вас разовьются достаточные навыки для ассемблирования/дизассемблирования в уме. Впрочем, есть множество эффективных приемов, позволяющих облегчить сей труд. Ниже я покажу некоторые из них. Попробуем без дизассемблера взломать crackme01.com. Для этого даже не обязательно помнить коды всех команд!

```
00000000: B4 09 BA 77 01 CD 21 FE C4 BA 56 01 CD 21 8A 0E |
„.єw.ќ!.”єV.ќ!K.
00000010: 56 01 87 F2 AC 02 E0 E2 FB BE 3B 01 30 24 46 81 |
V.З.м.рт.Ћ;.0$FБ
00000020: FE 56 01 72 F7 4E 02 0C 81 FE 3B 01 73 F7 80 F9 |
.V.rЇN..Б.;.s.A.
00000030: C3 74 08 B4 09 BA BE 01 CD 21 C3 B0 94 29 9A 64 |
”t.„.єЋ.ќ!”ЋФ)Ъd
00000040: 21 ED 01 E3 2D 2A 70 41 53 53 57 4F 52 44 00 6F | !э.y-
*pASSWORD.o
00000050: 6B 01 20 2A 04 B0 20 00 00 00 00 00 00 00 00 00 |
k..*.Ћ.........
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
................
00000070: 00 00 00 00 00 00 00 43 72 61 63 6B 20 4D 65 20 |
.......Crack Me
00000080: 20 30 78 30 20 3A 20 54 72 79 20 74 6F 20 66 6F |  0x0 : Try
to fo
00000090: 75 6E 64 20 76 61 6C 69 64 20 70 61 73 73 77 6F | und valid
passwo
000000A0: 72 64 20 28 63 29 20 4B 50 4E 43 0D 0A 54 79 70 | rd (c)
KPNC..Typ
000000B0: 65 20 70 61 73 73 77 6F 72 64 20 3A 20 24 0D 0A | e password
: $..
000000C0: 50 61 73 73 77 6F 72 64 20 66 61 69 6C 2E 2E 2E | Password
fail...
000000D0: 20 74 72 79 20 61 67 61 69 6E 0D 0A 24          |  try
again..$
```

Итак, для начала поищем, кто выводит текст '`Crack me... Type password:`'. В самом файле начало текста расположено со смещением `77h`. Следовательно, учитывая, что com-файлы загружаются, начиная со смещения `100h`, эффективное смещение равняется `100h+77h=177h`. Учитывая обратное расположение старших и младших байт, ищем в файле последовательность `77h 01h`.

```
00000000: B4 09 BA 77 01 CD 21
```

Вот она! Но что представляет собой код `0BAh`? Попробуем определить это по трем младшим битам. Они принадлежат регистру `DL(DX)`. А `0B4h 09h` - это `* AH,9`. Теперь нетрудно догадаться, что оригинальный код выглядел как:

```
            MOV AH,9
            MOV DX,0177h
```

И это при том, что не требуется помнить код команды MOV! (Хотя это очень распространенная команда и запомнить ее код все же не помешает).

Вызов 21-го прерывания 0CDh 21h легко отыскать, если запомнить его символьное представление '=!' в правом окне дампа. Как нетрудно видеть, следующий вызов INT 21h лежит чуть правее по адресу 0Ch. При этом DX указывает на 0156h. Это соответствует смещению 056h в файле. Наверняка эта функция читает пароль. Что ж, уже теплее. Остается выяснить, кто и как к нему обращается. Ждать придется недолго.

```
                                                  чтение строки          CL(CX)
                                                        ↓                  ↓
00000000: B4 09 BA 77 01 CD 21 FE C4 BA 56 01 CD 21 8A 0E ← 00 001 110
00000010: 56 01 87 F2 AC 02 E0 E2 FB BE 3B 01 30 24 46 81   ↑         ↑
          ↑                                              смещение16
BP
 смещение пароля
```

При разборе байта 0Eh не забудьте, что адресации [BP] не существует в природе. Вместо этого мы получим [offset16]. На размер регистра и приемник результата указывают два младших бита байта 08Ah. Они равны 10b. Следовательно, мы имеем дело с регистром CL, в который записывается содержимое ячейки [0156h].

Все, знакомые с ассемблером, усмотрят в этом действии загрузку длины пароля (первый байт строки) в счетчик. Неплохо для начала? Мы уже дизассемблировали часть файла и при этом нам не потребовалось знание ни одного кода операции, за исключением, быть может, 0CDh, соответствующего команде INT.

Вряд ли мы скажем, о чем говорит код 087h. (Впрочем, обращая внимание на его близость к операции NOP, являющейся псевдонимом XCHG AX,AX, можно догадаться, что 087h - это код операции XCHG). Обратим внимание на связанный с ним байт 0F2h:

```
          SI
          ↓
F2 → 11 110 010
      ↑      ↑
   Reg/Reg  (DX)
```

Эта команда заносит в SI смещение пароля, содержащееся в DX. Такой вывод следует исключительно из смыслового значения регистров (код команды игнорируется). К сожалению, этого нельзя сказать о следующем байте - 0ACh. Это код операции LODSB, и его надо просто запомнить.

0x02 - код операции ADD, а следующий за ним байт - код AH,AL.

0xE2 - код операции LOOP, а следующий за ним байт - знаковое относительное смещение перехода.

```
00000010: 56 01 87 F2 AC 02 E0 E2 FB BE 3B 01 30 24 46 81
                       ↑_____ 5 _____|
```

Чтобы превратить его в знаковое целое, необходимо дополнить его до нуля, (операция NEG, которую большинство калькуляторов не поддерживают). Тот же самый результат мы получим, если отнимем от 0100h указанное значение (в том случае, если разговор идет о байте). В нашем примере это равно пяти. Отсчитаем пять байт влево от начала следующей команды. Если все сделать правильно, то вычисленный переход должен указывать на байт 0ACh (команда LODSB), впрочем, последнее было ясно и без вычислений, ибо других вариантов, по-видимому, не существует.

Почему? Да просто данная процедура подсчета контрольной суммы (или точнее хеш-суммы) очень типична. Впрочем, не стоит всегда полагаться на свою интуицию и "угадывать" код, хотя это все же сильно ускоряет анализ.

С другой стороны, хакер без интуиции - это не хакер. Давайте применим нашу интуицию, чтобы "вычислить", что представляет собой код следующей команды. Вспомним, что 0B4h (10110100b) - это MOV AH,imm8.

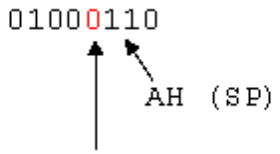
0BEh очень близко к этому значению, следовательно, это операция MOV. Осталось определить регистр-приемник. Рассмотрим обе команды в двоичном виде:

Как уже говорилось выше, младшие три бита - это код регистра. Однако, его невозможно однозначно определить без утончения размера операнда. Обратим внимание на третий (считая от нуля) бит. Он равен нулю для AH и единице в нашем случае. Рискнем предположить, что это и есть бит размера операнда, хотя этого явно и не уточняет Intel, но вытекает из самой архитектуры команд и устройства декодера микропроцессора.

Обратим внимание, что это, строго говоря, частный случай, и все могло оказаться иначе. Так, например, четвертый справа бит по аналогии должен быть флагом направления или знакового расширения, но увы - таковым в данном случае не является. Четыре левые бита - это код операции mov reg, imm. Запомнить его легко - это "13" в восьмеричном представлении.

Итак, 0BEh 03Bh 001h - это MOV SI, 013Bh. Скорее всего, 013Bh - это смещение, и за этой командой последует расшифровщик очередного фрагмента кода. А может быть и нет - это действительно смелое предположение. Однако, байты 030h 024h это подтверждают. Хакеры обычно так часто сталкиваются с функций xor, что чисто механически запоминают значение ее кода.

Не трудно будет установить, что эта последовательность дизассемблируется как XOR [SI],AH. Следующий байт 046h уже нетрудно "угадать" - INC SI. Кстати, посмотрим, что же интересного в этом коде:



Третий бит равен нулю! Выходит команда должна выглядеть как `INC AH`! (Что кстати, выглядит непротиворечиво смысле дешифровщика). Однако, все же это `INC SI`. Почему мы решили, что третий бит - флаг размера? Ведь Intel этого никак не гарантировала! А команда `'INC byte'` вообще выражается через дополнительный код, что на байт длиннее.

Выходит, что как ни полезно знать архитектуру инструкций, все же таблицу кодов команд хотя бы местами надо просто выучить. Иначе можно впасть в глубокое заблуждение и совершить грубые ошибки. Хотя с другой стороны, знание архитектуры порой очень и очень помогает.

Защита от отладчиков

Средства отладки и взлома программ

В настоящее время современные хакеры используют обширный инструментарий. Его можно подразделить на следующие группы:

- отладчики реального режима
- отладчики V86
- эмуляторы
- автоматические распаковщики
- дизассемблеры
- прочие программы

В свою очередь первые две группы опять разделяются на использующие стек отлаживаемой программы и неиспользующие его.

Отладчики реального режима

Наиболее известные:

1. **TurboDebugger by Borland International**
   Созданный в 1988 году двумя братьями Chris'ом и Rich'ем Williams Содержит множество ошибок, активно используемых защитами, таких как:
   a. использование стека отлаживаемой программы
   b. использование int 1, int 3 для трассировки
   c. перехват прерываний int 0, int 1, int 3
   d. некорректная работа с видеобуфером
   e. некорректное выставление начальных значений регистров

   API отсутствует.
   Обладает чрезвычайно развитым оконным интерфейсом, предоставляет возможности по просмотру кода и исходного текста программы, шестнадцатеричного дампа, переменных (при наличии отладочной информации), созданию макросов, отличается завидной медлительностью, связанной с подкачкой оверлея.

2. **CodeView by Microsoft**
   По своим ошибкам ничем не отличается от TurboDebugger'a. Поддерживает собственный формат отладочной информации. В силу устройства самого ядра отладчика не приспособлен в качестве среды для взлома.
3. **AFD**
   Созданный в 1988 году H.Puttkamer'ом отладчик предоставляет следующие возможности: пошаговый режим исполнения инструкций, пошаговое исполнение подпрограмм, сохранение точек останова в пользовательском файле, поиск данных в памяти, создание макросов и запись их в файл. Для использования в качестве среды для взлома не предназначен.

4. **Debug**
   Одним из самых первых отладчиков, существовавших для IBM PC, является отладчик DEBUG, поставляемых с операционной системой MsDos. Содержит все ошибки отладчиков реального режима. В настоящее время нигде не используется.

На сегодняшний день отладчики реального режима не представляют интереса, т.к. весь современный софт в реальном режиме не работает.

Отладчики защищенного режима

1. **TurboDebugger/386 by Borland International**
   Надстройка над TD, представляющая device-driver TDH386.Sys (низкоуровневый интерфейс сопроцессором) и запускающую программу TD386, вводящую процессор в режим V86. Полностью поддерживает ошибки своего предшественника. Предоставляет возможность установки аппаратных точек останова: по обращению на чтение/запись байта в памяти, перекрытие обращения к портам (не всегда корректно обрабатываемое).
2. **Soft-Ice by Nu-Mega Techologies**
   Наиболее мощный отладчик. Поддержка VCPI. Есть разновидности под Win95/WinNT
   Содержит также некоторые ошибки:
   a. Не является полностью stealth-отладчиком, так как оставляет кусок своего кода в conventional memory V86 машины.
   b. Существует API между программой и отладчиком
   c. S-Ice можно обнаружить по устройству SOFTICE1
   d. Загрузчик LDR неправильно выставляет значение SP
   e. некорректное выставление начальных значений регистров

   Предоставляет API через int 3, функции 09-13h. Поддерживает отладочную информацию Microsoft ('NB' в начале отладочной информации), Borland (db 0FB52h)

3. **Soft-Ice/W by Nu-Mega Techologies**
   Отладчик под Windows 3.xx. Обнаруживается по присутствию VxD устройства WINICE. Отслеживает конструкции вида cs:pushf.
4. **Deglucker by S.Gorokhov & A.Ilyushin**
   Ошибки:
   a. Переключение в нестандартный видеорежим
   b. Невозможность перехвата портов ввода/вывода
   c. запирание клавиатуры через i/o портов 60h/64h

   Предоставляет API через int 15h функции 0FFxxh. Трассирует программу через DRx (аппаратные регистры останова).

Эмуляторы

1. **EDB by Serge Pachkovsky**
   Эмулятор 80286 процессора. Крайне убогий интерфейс, на уровне DEBUG. Имеется возможность просмотра/изменения памяти, несколько режимов эмуляции.

2. **Soft Debugger**
   Полноценный эмулятор 80386, без поддержки функций защищенного режима. Поддерживает отладочную информацию компиляторов Borland International. Отслеживает изменение байтов в конвейере, имеется несколько режимов эмуляции: с вызовом собственного int 1/int 3, режим Full Tracing, Auto Tracing и другие.
3. **SD by Dmitry Groshev**
   Удобный и мощный сервис. Гибкие возможности для работы с самыми разнообразными структурами данных. Может подгружать сервисные модули.

Автоматические распаковщики

К автоматическим распаковщикам относятся программы, запускающие в автоматическом или полуавтоматическом режиме защищаемую программу и отслеживающие типовые участки startup-кода и соответственно настраивающие relocations.

Функция автоматических распаковщиков - сдирание защиты с файла и получение работоспособного EXE файла. Физика данного процесса такова: перехватывая первое программное прерывание, вызванное программой после отработки защиты, распаковщик снимает дамп памяти с уже расшифрованным кодом защищенной программы. Первый этап работы по снятию - нахождение этого самого первого прерывания. Это делается при помощи любого отладчика. Не будем вдаваться в подробности отлова первого прерывания, замечу лишь одно - все программы написанные на C/C++ и откомпилированные компилятором любой фирмы одной из первой командой проверяют версию OS:

```
B4 30          mov    ah, 30h
CD 21          int    21h
```

В программах, написанных на Паскале идет перехват векторов 00 и некоторых других:

```
B4 35          mov    ah, 35h
CD 21          int    21h
```

Если посмотреть ссылку на это место, то можно увидеть, что этот фрагмент вызывается после двух far call-ов:

```
call   far    Initturbo
call   far    SwapVectors
```

Таким образом можно найти истинную точку входа в паскалевскую программу.

1. **Autohack by BCP group**
   Предоставляет три варианта запуска:
   1. Распаковка трассировкой. В данном режиме работают почти все распаковщики программ. В данном режиме возможна распаковка программ, не защищенных от трассировки. Режим работает следующим образом: программа загружается в память, перехватывается первое прерывание, возводится флажок пошаговой трассировки (поэтому распаковка в этом режиме относительно медленно работает), управление передается загруженной программе, далее обработчик первого прерывания анализирует сегмент кода трассируемой программы и ждет смены регистра CS (число смен регистра CS запрашивается во время запуска под именем "глубина трассировки"). После этого сбрасываются дампы памяти, и операция повторяется с загрузкой программы с другого начального адреса.

2. Стандартный режим взлома. Режим работы программы основанный на режиме перехвата определенных моментов после отработки механизма защиты и сброса дампов памяти.
3. Режим взлома с поддержкой таблиц компиляторов. Идентичен второму режиму, но нацелен на определенные компиляторы, поэтому взламывает более корректно. о если взламывается программа откомпилированная неизвестным AutoHack-у компилятором, то произойдет запуск взламываемой программы с последующими глюками.

2. **Intruder by Creat0r**
   Отслеживает startup-код.
3. **SnapShot by Dale Co.**
4. **CUP by Cyberware products**
5. **UNP by B.Castricum**
6. **Tron**
7. **TSUP**

Дизасемблеры

Дизасемблер переводит выполняемый код в листинг на асемблере.

Прочие программы

К прочим программам можно отнести программы ориенитированные на специальные языки, например:

- Clipper

  Valkyrie Declipper 5, Version 1.0, Revision K
  "+" Есть возможность анализировать низкоуровневый код, декомпиляция до исходных текстов
  "-" Может работать только с известными ему линковщиками, если линковщик ему не известен, то он отказывается работать.
  Hackers Declipper v1.3 by KrK //UCL
  "+" Позволяет анализировать низкоуровневый код, можно самостоятельно задать начало псевдокода и таблицы имен переменных.
  "-" Полностью ручная работа при декомпиляции, не распознает начала процедур, не создает исходного текста, и т.д и т.п.
  Rescue5 v1.0 CA-Clipper decompiller
  "+" Декомпиляция до исходников.
  "-" Понимает очень мало линкеров, нет возможности анализировать псевдокод

- FoxPro

  ReFox

Программы типа Hiew (Hacker's view by SEN), позволяющие просмотреть код, изменить его, дизасемблировать участки кода.

Отладчик SoftIce

SoftIce - это универсальный отладчик, которым можно проанализировать и отладить любой код, включая подпрограммы прерывания и драйверы ввода-вывода. SoftIce состоит из отладчика уровня ядра (kernel mode debugger) (собственно это и есть отладчик) и загрузчика отладочной информации (Symbol Loader). Утилита Symbol Loader (SL) загружает отладочную информацию для вашего модуля, позволяет настроить SoftIce, и дает возможность записать историю команд (history buffer) в файл.

Возможности SoftIce:

- Символьная отладка 32-битных приложений, отладка драйверов устройств для WIN NT, драйверов для WIN95, VxD, 16-битных программ для DOS и Windows.
- Отладка фактически любого кода, включая подпрограммы прерывания и внутренние подпрограммы WIN 95 и WIN NT.
- Установка точек останова на операции чтения/записи в память, чтения/записи портов ввода-вывода, прерываний.
- Установка точек останова на сообщения Windows.
- Установка точек останова, срабатывающих при определенных условиях (условных точек останова) и действий которые должны произойти при срабатывании точки останова.

Утилита SL позволяет прочитать отладочную информацию из отлаживаемых программ (EXE, DLL, VxD, 386, OCX) и загрузить ее в отладчик, запустить ваше приложение и автоматически установить точку останова на точку входа в программу, записать в файл протокола отладки.

В поставку SoftIce входит пример GdiDemo.

1. Загрузка отлаживаемой программы

- Запустить SL.
- Выбрать опцию Open module в меню File.
- Открыть Gdidemo.exe.
- Выбрать опцию Load в меню Module.

SL оттранслирует отладочную информацию в .NMS файл, загрузит исходные файлы, запустит отлаживаемую программу (в данном случае Gdidemo) и всплывет в SoftIce, где вы увидите исходный текст программы.

Подсвеченная строка с номером 35 - это точка входа (entry point) в вашу программу. Если SL вывел сообщение типа "An error occured during symbol translation/load", значит в отлаживаемом файле отсутствует отладочная информация, жмите OK и наслаждайтесь [диз]ассемблером.

2. Управление SoftIce'ом

Если все сделано правильно, то вы должны увидеть SoftIce разбитый на несколько окон. Верхнее окно - Register Window (окно регистров) - показывает состояние рабочих

регистров процессора. Под ним находится окно данных Data Window, в нем вы можете посмотреть или отредактировать дамп памяти. Ниже находится Code Window (окно кода) - в нем находится исходный текст программы (если вы загрузили отладочную информацию), или дизассемблированный код программы. В самом низу находится окно команд - Command Window, в нем вы можете вводить команду и видеть результат их выполнения. Самая нижняя строчка - строка помощи, в ней при вводе подсвечиваются возможные варианты команд и их синтаксис. Удобнее всего управлять SoftIce с помощью мышки.

Изменение размера окна - подведите курсор к нижней границе того окна, которому хотите изменить размер или закрыть его, нажмите левую кнопку мыши и ведите ее вниз (увеличение размера) или вниз (уменьшение размера), если хотите закрыть окно, подведите нижнюю границу к верхней, в окне появится фраза Close current window и окно исчезнет.
Примечание: нельзя изменить размер окна регистров и FPU.

Скроллинг на одну строку - подведите курсор к маленьким стрелочкам, расположенным у границ того окна, которое хотите скроллировать и жмите левую кнопку мыши (стрелочки появляются, если размер окна больше или равен двум строчкам).

Скроллинг на экран - подведите курсор к большим стрелочкам расположенным внутри того окна, которое хотите скроллировать и жмите левую кнопку мыши (стрелочки появляются, если размер окна больше или равен четырем строчкам).

Изменение значений регистров - подведите курсор к тому регистру, значение которого хотите изменить, нажмите левую кнопку мыши и введите число, если нужно изменить одну цифру, то подводите курсор к этой цифре и меняйте.

Изменение значений флагов - подведите курсор к тому флагу, который хотите изменить, нажмите левую кнопку мыши, после чего клавишей Ins можно изменить значение флага на противоположное (маленькая буква означает, что флаг не установлен, большая установлен).

Изменение значений ячеек памяти - подведите курсор к тому байту (слову, двойному слову и т.д.), которое хотите изменить, нажмите левую кнопку мыши и вводите ваше значение, если хотите изменить одну или несколько цифр в числе, то подведите курсор с помощью клавиатуры к нужным числам и меняйте.
Примечание: во всех случаях изменения значений они вступают в силу после того, как вы переключитесь в любое другое окно, до этого можно отменить последнее изменение, нажав Esc.

Установка точек останова на исполнение - подведите курсор к той строке в Code Window, в которой хотите остановиться, и двойным щелчком по левой кнопке мыши поставьте точку останова, строка подсветится.

Удаление из Watch Window переменных - установите курсор на переменную, которую хотите удалить, нажмите левую кнопку мыши, переменная подсветиться, нажмите кнопку Del - переменная исчезнет.

Контекстное меню - по правой кнопке мыши вы попадаете в контекстное меню, в котором вам доступны команды:

- Copy - копировать в буфер обмена адрес или данные, находящиеся под курсором.
- Paste - вставить в окно команд, адрес или данные, находящиеся в буфере обмена.
- Copy&Paste - копировать в буфер обмена адрес или данные, находящиеся под курсором, и вставить их в окно команд.
- Display - вывести в окно данных дамп памяти, расположенный по адресу, над которым в данный момент находиться курсор (Аналог команды D).
- Un-Assemble - вывести в Code Window исходный (если есть отладочная информация) или дизассемблированный текст программы, находящийся по адресу, над которым в данный момент находиться курсор (Аналог команды U).
- What - идентифицирует значение, находящееся под курсором с заранее определенными (Аналог команды Wath).
- Previous - отменяет предыдущую команду, введенную из контекстного меню (работает с командами Display и Un-Assemble).

3. Трассировка программы

Воспользуйтесь командой T (trace) для того чтобы оттрассировать одну команду, или клавишей F8, которая закреплена по умолчанию за командой T. Произойдет выполнение команды находящейся в текущей строке и курсор перейдет на следующую строку и подсветит ее. Это строка:

```
LpszLine=LpszLine;
```

Еще раз нажмите F8, курсор передвинется на следующую строку:

```
if(!hPrevInst).
```

В Code Window вы видите исходный текст программы (source mode). Если вы хотите посмотреть дизассемблированный (code mode) текст программы или исходный и дизассемблированный (mixed mode) текст вместе, воспользуйтесь командой SRC или клавишей F3 закрепленной за этой командой. При первом нажатии вы увидите смешанный (исходный текст программы и ассемблерные инструкции, из которых состоит эта строка) текст, при втором нажатии дизассемблированный, третье нажатие вернет вас в режим просмотра исходного текста программы.

Нажмите еще раз F8 и вы перейдете к строке

```
if(!RegisterAppClass(hInst));
```

Для того, что бы отлаживать программу вы пользуетесь командой T, которая исполняет один оператор исходной программы или одну машинную команду.

Еще существует команда P или клавиша F10, которая выполняет один шаг в программе т.е. при трассировке какой-либо функции или прерывания, вы не получите управление до тех пор пока выполнение функции не завершится и вы не вернетесь из функции обратно. Команду P удобно применять в том случае, когда вы отлаживаете основной алгоритм и отвлекаться на трассировку каждой процедуры нерационально.

Примечание: командой T нельзя оттрассировать системные вызовы (WIN32 API calls) находясь в source mode, для их трассировки нужно перейти в mixed или code mode.

4. Просмотр локальных переменных

Окно Locals Window показывает текущий кадр стека. В нашем случае он содержит локальные переменные для функции WinMain.

Командой T войдите в функцию RegisterAppClass, окно Locals Window станет пустым, так как для этой функции еще не определены локальные переменные. Функция RegisterAppClass находится в файле INIT.C. SoftIce показывает текущий файл в левом верхнем углу Code Window

Введите команду T снова, окно Locals Window будет содержать параметр переданный функции RegisterAppClass (hInstance) и локальную структуру wndClass. Перед структурой стоит знак плюс, который означает что внутри находятся переменные, которые можно посмотреть (так же можно смотреть строковые переменные и массивы). Посмотреть структуру можно двойным щелчком мыши. Знак + сменится на - и, вы увидите переменные, из которых состоит структура. Закрыть структуру можно также двойным щелчком мыши.

5. Установка точек останова на выполнение

Точки останова на выполнение делятся на два вида: просто точки останова и однократные точки останова.

Однократные точки останова

Перейдите в Code Window, используя клавишу PgDn переместите курсор на строку с номером 61 (тоже самое можно сделать используя команду U .61), в этой строке находиться первый вызов функции Win32 API RegisterClass. Используя команду HERE (клавиша F7) выполните программу до этой строки.

Команда HERE устанавливает точку останова в программе на адрес или строку на которой находиться курсор и выполняет программу с текущего адреса до адреса на котором находиться курсор, т.е. до тех пор пока не сработает точка останова, после срабатывания SoftIce автоматически отключит эту точку останова, чтобы она больше не срабатывала.

Текущей строкой в отлаживаемой программе стала строка:

```
If(!RegisterClass(&wndClass))
```

Примечание: того же самого результата можно было добиться командой G .61 (исполнять программу до строки 61).

Обычные точки останова

Следующие шаги демонстрируют использование обычных точек останова, т.е. таких, которые будут срабатывать до тех пор пока вы их не отмените. Найдите следующий вызов функции RegisterClass, находящийся в

строке 74. Установите курсор на эту строку и введите команду BPX (BreakPoint eXecutable) или клавишу F9 (по этой команде, в память на место команды расположенной под курсором записывается команда INT3, но вы этого не видите. Строка должна подсветиться. Снять точку останова можно повторным вводом этой же команды. Если вы счастливый обладатель процессора Pentium, то процесс установки и снятия точки останова сводится к двойному щелчку левой кнопкой на той команде, где хотите поставить точку останова. После установки точки останова запустите программу командой G или X (клавиша F5). Когда программа исполнит инструкцию INT3, она передаст управление SoftIce, после чего он появиться перед вами. Посмотреть информацию об установленных точках останова можно командой BL (BreakPoint List):

```
:BL
00) BPX #0137:00402442 (адрес может быть другим).
```

Видим что установлена одна точка останова на исполнение по адресу 00402442, с порядковым номером 0. По этому адресу находиться команда, расположенная в текущем файле INIT.C в строке 74. Вы можете использовать вычисление выражений, для того чтобы получить адрес строки по ее номеру:

```
:? .74
void * = 0x00402442
```

Так как дальнейшая пошаговая трассировка функции RegisterAppClass не имеет для нас смысла, вернемся в то место, откуда эта функция вызывалась. Для этого существует команда P с параметром RET (клавиша F12). Она позволяет выполнять программу до тех пор пока не встретит команду RET (RETF), исполнив эту команду SoftIce выйдет из подпрограммы и остановиться на строке следующей за вызовом этой подпрограммы. Применительно к нашей программе: функция RegisterAppClass вызывается из функции WinMain, SoftIce остановиться в функции WinMain на строке следующей за вызовом функции RegisterAppClass, т.е. подсвечена будет строка:

```
Msg.wParam = 1;
```

Воспользуйтесь командой BC (Breakpoint Clear) с номером точки останова, который вы посмотрели с помощью команды BL, для того, что бы убрать именно эту точку останова, или вместо номера введите *, тогда вы уберете все точки останова (можно вводить номера точек останова через запятую, если хотите убрать несколько точек).

6. Использование информационных команд

SoftIce имеет в своем распоряжении много разных команд, с помощью которых можно узнать состояние и получить иную информацию об операционной системе и запущенных в ней приложениях. Мы рассмотрим только две команды: H (Help) и CLASS. Эти команды выводят достаточно много информации в окне команд (Command Window), поэтому желательно увеличить размер этого окна, для этого закроем окно локальных переменных (Locals Window).

По команда H можно получить помощь по всем командам SoftIce или более подробную информацию о конкретной команде, если введете ее имя в качестве аргумента команды H:

```
:H CLASS
Display window class information
CLASS [-x] [task-name]
Ex: CLASS USER
```

Первая строка дает описание команды, вторая показывает информацию о синтаксисе и аргументах, которые могут использоваться в команде, третья строка содержит пример использования команды.

Целью выполнения функции RegisterAppClass является регистрация шаблона для классов окон, которые будут использованы приложением GdiDemo для создания окон. Используя команду CLASS можно посмотреть зарегистрированные классы для GdiDemo:

```
:CLASS GDIDEMO
```

Результатом выполнения данной команды является информация о каждом зарегистрированном классе окна. Информация включает в себя: имя класса, адрес внутренней структуры данных WNDCLASS, модуль который зарегистрировал данный класс, адрес процедуры, которая обслуживает данный класс и состояние флагов стиля класса. Для получения подробной информации воспользуйтесь ключом -X.

```
  Handle  Class Name  Owner    WndwProc       Styles
  ------------------------Application Private---------------
----------------
  5110    ???         GDIDEMO  2E9F:00000114  07000003
  40AC    ???         GDIDEMO  2E9F:000000FE  07000003
  409C    ???         GDIDEMO  2E9F:000000E8  03000003
  3BC4    ???         GDIDEMO  2E9F:000000D2  03000003
  3BB4    ???         GDIDEMO  2E9F:000000BC  07000003
  3A00    ???         GDIDEMO  2E9F:000000A6  07000003
```

7. Символьные имена

Когда вы загружаете приложение с отладочной информацией, SoftIce автоматически создает таблицу символьных имен которая содержит все имена, определенные в данном приложении. Используя команду TABLE можно посмотреть какие таблицы символьных имен загружены в настоящий момент:

```
:TABLE
GDIDEMO [NM32]
0001044741 Bytes Of Symbol Memory Available
```

Используемая в данный момент таблица символьных имен выделена цветом. Если текущая таблица символьных имен не соответствует той на которую ссылается ваше приложение, то используя команду TABLE с именем вашего приложения в качестве аргумента, вы подключите нужную таблицу (если для вашего приложения создана таблица):

```
:TABLE GDIDEMO
```

Используя команду SYM вы можете посмотреть все символьные имена, определенные в текущей таблице (на экран выводятся по сегментам, внутри них в алфавитном порядке). Если интересует какие-то определенные имена, то используйте шаблоны:

```
:SYM w*
.text (0137:00401000, 000145C1 bytes)
0137:004012E0 WinMain
0137:00405700 WinMainCRTStartup
0137:004013AD WndProc
0137:0040AF50 wcslen
0137:0040C160 wcsncnt
0137:004107A0 wctomb
0137:0040FA50 write_char
0137:0040FAD0 write_multi_char
0137:0040FB20 write_string
```

На экране список всех символьных имен начинающихся с буквы w, все они расположены в сегменте .text (выполняемый сегмент, он начинается с адреса 0137:00401000 и имеет длину 0145C1H байт), т.е. эти имена - имена функций и процедур входящие в приложение GDIDEMO. Данные находятся в сегментах .data, .rdata, .idata.

8. Условные точки останова

Одним из символов, определенных в приложении GDIDEMO, является функция LockWindowInfo. Назначение этой функции - возвращение адреса переменных, которые определяют свойства окна. Для того чтобы ознакомиться с условными точками останова и точками останова на доступ к памяти, мы выполним следующие действий:

- Установим точку останова на функцию LockWindowInfo.
- Отредактируем поставленную точку останова таким образом, чтобы она срабатывала по определенному нами условию.
- Установим точку останова на доступ к ячейки памяти, для того чтобы отследить обращения к этой ячейке.

Установка точки останова на функцию LockWindowInfo.

Командой BPX LockWindowInfo поставим точку останова на выполнение на эту функцию. Каждый раз, когда в одном из окон приложения GDIDEMO нужно будет обновить информацию, программой будет вызываться функция LockWindowInfo, так как на эту функцию поставлена точка останова, то будет вызываться SoftIce. Командой BL проверьте, установилась ли точка останова. Запустите приложение командой X или G. Как только будет вызвана функция LockWindowInfo, SoftIce всплывет. Так как обновление происходит постоянно, то постоянно вызывается и SoftIce, что очень неудобно, если нас интересует обновление какого-либо конкретного окна. Чтобы перехватить вызов на обновление конкретного окна, к примеру, POLYDEMO, воспользуемся условной точкой останова. Из исходного текста программы (файл wininfo.c)

видно, что функция LockWindowInfo получает в качестве входного аргумента один параметр HWND (Handle Window) - дескриптор окна, и возвращает в вызывающую функцию одно значение - указатель на переменные для данного окна. То есть если бы мы заставили срабатывать точку останова только на обработчик окна POLYDEMO, мы бы добились своей цели. Для начала нам необходимо узнать дескриптор нашего окна, для этого воспользуемся командой:

```
     :HWND GDIDEMO
     WindowHandle  hQueue  SZ Qowner  ClassName
WindowProcedure
     0724(1)       10FF    32 GDIDEMO GDIDEMO    365F:000001C4
     0728(2)       10FF    32 GDIDEMO MdiClient  17A7:00001988
     0734(3)       10FF    32 GDIDEMO BOUNCEDEMO 365F:00000232
     0730(3)       10FF    32 GDIDEMO POLYDEMO   365F:000001DA
     072C(3)       10FF    32 GDIDEMO DRAWEMO    365F:0000021c
```

Дескриптор окна POLYDEMO имеет значение 0730. Если в списке вы не увидели нужного окна, то запустите приложение клавишей X или G, опять сработает точка останова, проверьте, создалось ли окно, если нет, то повторите последние действия. Теперь можно останавливать исполнение программы только в том случае, когда в качестве параметра для функции LockWindowInfo используется значение 0730. В Windows параметры для функции обычно передаются через стек. При остановке в функции LockWindowInfo стек будет выглядеть следующим образом (посмотреть содержимое стека можно подведя курсор к регистру ESP, нажав правую кнопку мыши, вызвать контекстное меню и выбрать команду Display, неплохо бы еще сменить командой DD формат вывода данных в DataWindow на показ двойных слов, так как наше приложение 32-разрядное):

```
     ESP = 0055FC00
     013F:0055FC00 00404852 00000730 0055FC3C 00008CAA
```

- Число 00404852 - это адрес, на который программа перейдет после завершения работы нашей функции (адрес возврата).
- Число 00000730 - это дескриптор окна POLYDEMO (собственно то, что нас интересует).

Теперь зная, где и что у нас передается в функцию, мы можем выставить условную точку останова. Для этого вызовем на редактирование точку останова, поставленную на функции LockWindowInfo:

```
:BPE 0
```

В нижней строке командного окна появится строка:

```
:BPX LockWindowInfo
```

и курсор установится в конце строки. Теперь можете редактировать точку останова по своему усмотрению, в конце строки добавьте следующее условие:

```
IF ESP->4 == 00000730
```

и нажмите Enter.

Точка останова теперь будет выглядеть так:

```
:BPX LockWindowInfo IF ESP->4 == 00000730
```

То есть она будет срабатывать тогда, когда двойное слово по адресу ESP+4 будет равным числу 00000730, которое соответствует дескриптору окна POLYDEMO. Проверьте командой BL, соответствует ли точка останова заданной, запустите приложение и убедитесь, что вся эта конструкция замечательно работает.

Установим точку останова на доступ к первому двойному слову данных экземпляра окна POLYWINDOW. Как уже отмечалось выше, входным аргументом для функции LockWindowInfo является дескриптор окна, выходным - адрес данных экземпляра окна. После срабатывания точки останова, поставленной на функции LockWindowInfo с параметром соответствующим дескриптору окна POLYWINDOW, на выходе функции будем иметь адрес данных экземпляра окна POLYWINDOW, по этому адресу и поставим точку останова на доступ к памяти.

Для того чтобы получить адрес данных экземпляра окна, выполним программу до строки с номером 57 (в файле WININFO.C):

```
:G .57
```

Функция возвращает 32-битное значение (в нашем случае адрес) в регистре EAX, поэтому можно, используя команду BPMD (BreakPoint Memory Dword) и значение адреса в регистре EAX, поставить точку останова на доступ к первом слову данных экземляра окна POLYDEMO:

```
:BPMD EAX
```

Эта команда использует регистры аппаратной отладки встроенные в процессор для отслеживания чтения и записи двойного слова по указанному линейному адресу. В данном случае это первое слово данных экземпляра окна POLYDEMO. Используйте команду BL, чтобы проверить правильность установки точки останова.

```
:BL
00)    BPX       LOCKWINDOWINFO IF ((ESP->4)==0x00000730)
01)    BPMD      #015F:0052006C  RW  DR3
```

Точка останова с номером 0 установлена на исполнение функции LockWindowInfo и точка останова с номером 1 стоит на доступ к памяти по адресу #015F:0052006C.

Отключите 0 точку останова командой BD (Breakpoint Disable):

```
:BD 0
```

В отличие от команды BC, которая удаляет точку останова, команда BD временно отключает точку, т.е. она не будет срабатывать, но ее в любой момент можно включить обратно командой BE (Breakpoint Enable).

Отключенные точки останова выделяются звездочками рядом с порядковым номером точки останова.

```
:BL
00) * BPX       LOCKWINDOWINFO IF ((ESP->4)==0x00000730)
01)   BPMD      #015F:0052006C  RW  DR3
```

Запустите SoftIce командами X или G, когда окно POLYDEMO попытается получить доступ к первому двойному слову экземпляра данных окна, сработает точка останова и SoftIce всплывет, это будет происходить в функциях PolyRedraw и PolyDrawBez. Эти функции обращаются к полю nBezTotal, которое находиться по нулевому смещению в данных экземпляра окна POLYDEMO. Значение этого поля задает количество кривых одновременно выводимых в окно POLYDEMO.

Примечание: Из-за особенностей архитектуры процессоров Intel перехват обращения к ячейки памяти произойдет после исполнения команды, обращающейся к памяти, т.е. SoftIce остановиться на следующей команде.

Сбросьте все точки останова командой BC * и выйдете из SoftIce.

Как заставить SoftIce работать?

Config.Sys: Device=c:\...\...\S-Ice.Exe
Затем запускаете Ldr.Exe <программа>. У MS-DOS есть маленький баг: он неверно выставляет значение регистра SP - он уменьшает его значения на 2, и некоторые защиты, активно использующие стек заставляют повеситься задачу. Лечится правкой кода MS-DOS при загрузке INT 21h AX=4B01h или командой "R SP=SP+2"

Как заставить SoftIce/Win/W95 работать?

Отредактировать файл WINICE.DAT, дать возможность грузить отладчику символьную информацию из системных DLL-ей. (В Winice.DAT даются ссылки на USER.EXE, KRNL386.EXE, WIN386.EXE)

```
exp=c:\win\system\user.exe
exp=c:\win\system\gdi.exe
exp=c:\win\system\krnl386.exe
```

Защита от отладчиков

Используя отладчики, искушенные в своем деле системные программисты рано или поздно смогут обнаружить и отключить (или обмануть) средства защиты.

Сейчас мы поговорим о некоторых приемах, позволяющих затруднить обнаружение средства защиты.

Стандартным методом, использующимся для обнаружения средств защиты от копирования, является дизассемблирование программы установки программного пакета или выполнение его под управлением пошагового отладчика. Листинг, получаемый в процессе дизассемблирования, оказывает большую услугу при использовании отладчика, поэтому эти два средства - дизассемблирование и использование отладчика - обычно используют вместе.

Соответственно требуются отдельные средства для борьбы с дизассемблером и для защиты от отладчиков.

Для затруднения дизассемблирования лучше всего подходит шифрование отдельных участков программ или всей программы целиком. Например, часть программы-инсталлятора можно оформить в виде отдельной COM-программы. После трансляции исходного текста этой программы ее можно зашифровать тем или иным способом и в зашифрованном виде подгружать в память как программный оверлей. После загрузки программу следует расшифровать в оперативной памяти и передать ей управление.

Еще лучше выполнять динамическое расшифрование программы по мере ее выполнения, когда участки программы расшифровываются непосредственно перед использованием и после использования сразу же уничтожаются.

При расшифровании можно копировать участки программы в другое место оперативной памяти. Пусть, например, программа состоит из нескольких частей. После загрузки ее в оперативную память управление передается первой части программы. Эта часть предназначена для расшифровки второй части, располагающейся в памяти вслед за первой.

Задача второй части - перемещение третьей части программы на место уже использованной первой части и расшифровка ее там.

Третья часть, получив управление, может проверить свое расположение относительно префикса программного сегмента и, в случае правильного расположения (сразу вслед за PSP), начать загрузку сегментных регистров такими значениями, которые необходимы для выполнения четвертой, инсталляционной части программы.

Если попытаться дизассемблировать программу, составленную подобным образом, то из этого ничего не получится.

Второй способ борьбы с дизассемблером является по сути борьбой с человеком, занимающимся дизассемблированием. Он заключается в увеличении размера загрузочного модуля до сотни-другой килобайт и в усложнении структуры программы.

Объем листинга, получающегося при дизассемблировании программы размером в 30-40 килобайт, достигает 1-1.5 мегабайта. Поэтому большие размеры инсталляционной программы могут сильно увеличить время обнаружения средств защиты.

Что такое усложнение структуры программы, достаточно понятно само по себе. Существует программа, использующая для обращения к одной и той же области памяти, содержащей многочисленные переменные, разные сегментные адреса. Поэтому очень трудно догадаться, что на самом деле программа работает с одной и той же областью памяти.

Перейдем теперь к борьбе с трассировкой программы пошаговыми отладчиками. Стандартные отладчики реального режима используют для работы два вектора:

Int 1 - пошаговый режим выполнения;
Int 3 - точка приостанова.
Ваша задача незаметно, не используя стандартные средства, которые легко отслеживаются как самим отладчиком, так и хакером изменить значения адреса этих векторов. Периодически необходимо проверять занесенные значения. Пошаговый режим выполнения включается при установленном флаге TF процессора. Сброс этого флага в ноль приводит к выключению отладчика. Все отладчики отслеживают команду popf и восстанавливают значение флага TF, но очень мало отладчиков понимают смоделированный возврат из прерывания, когда в стеке находится слово флагов с выключенным флагом трассировки. Аналогично все отладчики отслеживают команду pushf и очищают флаг TF, но большинство отладчиков не понимают команду cs:pushf или es:pushf.

Еще более изящно использовать вектора отладчика в своих целях. Например, вы можете переназначить 21 вектор на 3 и обращаться к MSDOS не через int 21h, а через int 3h это короче на 1 байт и поэтому не позволит взломщику произвести обратную замену.

Слово состояния процессора:

17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
VM	R		NT	I/O PL	OF	DF	IF	TF	SF	ZF		AF		PF		FC

VM - виртуальный режим

R - ошибка отладки
NT - вложенная задача
I/O PL - привилегии: 0 - высший, 3 - низший
OF - переполнение
DF - направление
IF - прерывания
TF - ТРАССИРОВКА
SF - знак
ZF - нуль
AF - вспомогат.перенос

PF - четность

FC - перенос

Еще один способ обнаружения отладчика - замер времени выполнения частей программы. При работе под отладчиком время выполнения значительно больше. Это связанно с тем, что при работе под отладчиком производятся дополнительные действия, такие как нажатие клавиш на клавиатуре для активизации некоторых действий отладчика. Замер времени лучше производить обращаясь непосредственно через порты к таймеру или используя CMOS-часы.

Большинство отладчиков используют стек отлаживаемой программы. Вы можете занести в неиспользуемую часть стека некоторую информацию до участка программы, в котором не используется стек, а после этого участка проверить:

```
        mov    bp,sp
        mov    ax,'a1'
        mov    [bp-2],ax

  ..... // не использовать стек!!!

        cmp    word ptr [bp-2],'a1'
        jne    a61    // под отладчиком
```

При работе в реальном режиме использование инструкций вида mov sp, 1 приводит к генерации int 06(invalid opcode). Вы можете маскировать прерывания на период выполнения критической части программы:

```
   CS:0100 E421              in    al,21h
   CS:0102 0C02              or    al,00000010b  ;IRQ 1 keyboard
irq
   CS:0104 E621              out   21h,al
```

или

```
   CS:0100 E461              in    al,61
   CS:0102 0C80              or    al,10000000b  ; bit 7 -
disable kbd
   CS:0104 E661              out   61h,al
```

или

```
   CS:0100 B4AD              mov   al, 0ADh      ; disable
keyboard
   CS:0102 E664              out   64h,al
```

Маскировка немаскируемого прерывания NMI на AT:

```
 mov    al,0ADh
 out    70h,al
```

Разрешить NMI:

```
 mov    al,2Dh
 out    70h,al
```

Очень хорошие результаты дает распаковка следующих инструкций по сложному самомодифицирующемуся алгоритму обработчиком Int 8 или Int 1.

Необходимо не забывать о возможности закомментировать проверяющие механизмы в вашей программе. Программа должна периодически просчитывать и проверять контрольные суммы участков программы.

Работа в защищенном режиме

Защищенный режим работы процессора открывает перед вами новую возможность. Возьмите любую программу, работающую в защищенном режиме, и попытайтесь запустить ее по управлением какого-либо отладчика (например, Turbo Debugger или Code View). Все будет хорошо до тех пор, пока ваша программа не попытается загрузить регистр IDTR при помощи команды LIDT. После выполнения этой команды отладчик "зависает" и единственное средство вновь оживить компьютер - нажать на кнопку сброса. Причина - изменились расположение и формат дескрипторной таблицы прерываний. Она подготовлена для работы в защищенном режиме, но отладчик работает в реальном режиме. Поэтому обработка всех прерываний, в том числе и от клавиатуры, невозможна. Идея использования защищенного режима работы процессора при создании программ, защищенных от несанкционированного доступа, копирования, очевидна.

Например, перед переключением в защищенный режим вы сможите подготовить в памяти массив контрольной информации. Расшифровка и проверка этого массива, а также запись данных в нестандартные секторы инсталляционной дискеты могут выполнятся в защищенном режиме. При этом пользуясь обычными отладчиками невозможно определить действия, выполняемые в защищенном режиме. Особенно если участок программы, работающей в зашифрованном режиме зашифрован и после выполнения затирается. Далее процессор можно вернуть в реальный режим и продолжить процесс инсталляции.

Находясь в защищенном режиме вы не можете обращаться к функциям DOS и BIOS, - вы можете читать и писать секторы дискеты только используя уровень портов ввода/вывода контроллера флопи-диска.

Программы с потенциально опасными последствиями

Программой с потенциально опасными последствиями назовем программу или часть программы, которая способна выполнить одно из следующих действий:

- скрыть признаки своего присутствия в программной среде ПЭВМ;
- самодублироваться, ассоциировать себя с другими программами и/или переносить свои фрагменты в какие-либо области оперативной или внешней памяти, не принадлежащие программе;
- изменять код программ в оперативной или внешней памяти;
- сохранять фрагменты информации из оперативной памяти в некоторых областях внешней памяти (локальных или удаленных);
- искажать произвольным образом, блокировать и/или подменять выводимый во внешнюю память или канал связи массив информации, образовавшийся в результате работы прикладных программ, или уже находящиеся во внешней памяти массивы данных.

Программы с потенциально опасными последствиями можно условно подразделить на:

- классические программы-"вирусы";
- программы типа "программный червь" или "троянский конь" и фрагменты программ типа "логический люк";
- программы типа "логическая бомба";
- программные закладки - обобщенный класс программ с потенциально опасными последствиями.

Кроме того, такие программы можно классифицировать по методу и месту их внедрения и применения (то есть по "способу доставки" в систему):

- закладки, связанные с программно-аппаратной средой (BIOS);
- закладки, связанные с программами первичной загрузки;
- закладки, связанные с драйвером DOS, командным интерпретатором, сетевыми драйверами, то есть с загрузкой и работой операционной среды;
- закладки, связанные с прикладным программным обеспечением общего назначения (встроенные в клавиатурные и экранные драйверы, программы тестирования ПЭВМ, утилиты, файловые оболочки);
- исполняемые модули, содержащие только код закладки (как правило, внедряемые в пакетные файлы типа BAT);
- модули-имитаторы, совпадающие по внешнему виду с легальными программами, требующими ввода конфеденциальной информации;
- закладки, маскируемые под программные средства оптимизационного назначения (архиваторы, ускорители и т.д.);
- закладки, маскируемые под программные средства игрового и развлекательного назначения (как правило, используются для первичного внедрения других закладок; условное название - "исследователь").

Вирус

Первые исследования саморазмножающихся искусственных конструкций проводились в середине прошлого столетия: в работах фон Неймана, Винера и др. дано определение и проведен математический анализ конечных автоматов, в том числе самовоспроизводящихся. Термин "компьютерный вирус" появился позднее - официально считается, что его впервые употребил сотрудник Лехайского университета (США) Фред Коэн в 1984 году на 7-й конференции по безопасности информации, проходившей в США.

Компьютерным вирусом называется программа, которая может создавать свои копии (не обязательно совпадающие с оригиналом) и внедрять их в файлы, системные области компьютера, сети и так далее. При этом копии сохраняют способность дальнейшего распространения.

Вирусы можно разделить на классы по следующим признакам:

- по среде обитания вируса;
- по способу заражения среды обитания;
- по деструктивным возможностям.

По среде обитания различают вирусы сетевые, файловые, загрузочные и специальные. Сетевые вирусы распространяются по компьютерной сети, файловые внедряются в выполняемые файлы, загрузочные в загрузочный сектор диска(Boot) или сектор, содержащий системный загрузчик винчестера(Master Boot Record). Специальные ориентированы на конкретные особенности ПО, например вирус, заражающий документы редактора Word. Существуют сочетания - например, файлово-загрузочные вирусы, заражающие и файлы и загрузочные сектора дисков. Кроме того, по сети могут распространятся вирусы любых типов.

Способы заражения делятся на резидентный и нерезидентный. Резидентный вирус при инфицировании компьютера оставляет в оперативной памяти свою резидентную часть, которая затем перехватывает обращение операционной системы к объектам заражения и внедряется в них. Резидентные вирусы находятся в памяти и остаются активными вплоть до выключения или перезагрузки компьютера. Нерезидентные вирусы не заражают память компьютера и являются активными ограниченное время. Некоторые вирусы оставляют в оперативной памяти небольшие резидентные программы, которые не распространяют вирус. Такие вирусы считаются нерезидентными.

По деструктивным возможностям вирусы можно разделить на:

- безвредные, никак не влияющие на работу компьютера (кроме уменьшения свободной памяти на диске в результате своего распространения);
- неопасные, влияние которых ограничивается уменьшением свободной памяти на диске и графическими, звуковыми и прочими эффектами;
- опасные вирусы, которые могут привести к серьезным сбоям в работе компьютера;
- очень опасные вирусы, которые могут привести к потере программ, уничтожить данные, способствовать ускоренному износу или повреждению частей механизмов(например, головок винчестеров).

Люк

Люком называется не описанная в документации на программный продукт возможность работы с этим программным продуктом. Сущность использования люков состоит в том, что при выполнении пользователем некоторых не описанных в документации действий он получает доступ к возможностям и данным, которые в обычных условиях для него закрыты (в частности - выход в привилегированный режим).

Люки чаще всего являются результатом забывчивости разработчиков. В процессе разработки программы разработчики часто создают временные механизмы, облегчающие ведение отладки за счет прямого доступа к отлаживаемым частям продукта. По окончанию отладки большинство люков убирается из программы; но люди есть люди - зачастую они забывают о существовании каких-то мелких "лючков".

Одним из наиболее показательных примеров использования "забытых" люков является, пожалуй, широко известный в компьютерном мире инцидент с вирусом Морриса. Одной из причин, обусловивших возможность распространения этого вируса, была ошибка разработчика программы электронной почты, входящей в состав одной из версий операционной системы UNIX, приведшая к появлению малозаметного лючка. Для вас, наверное, будет небесполезно знать, что американские специалисты оценивают ущерб, нанесенный в результате этого инцидента, более чем в 100 миллионов долларов.

Люки могут образовываться также в результате часто практикуемой технологии разработки программных продуктов "сверху вниз". При этом программист приступает сразу к написанию сразу управляющей программы, заменяя предполагаемые в будущем подпрограммы так называемыми "заглушками". В теории моментом завершения разработки конечной программы по такой технологии можно считать момент замены последней заглушки реальной подпрограммой.

В действительности дело обстоит несколько сложнее. Вся беда в том, что авторы часто оставляют заглушки в конечном программном продукте, передаваемом в эксплуатацию. Делают это порой неумышленно: например, на ранних стадиях разработки предполагалось наличие в конечном программном продукте некоторой подпрограммы, однако в процессе разработки выяснилось, что эта подпрограмма в силу каких-либо причин не нужна. Но заглушка-то осталась! Удалить заглушку, не заменяя ее подпрограммой, бывает весьма сложно. Это может спровоцировать программиста оставить заглушку "до лучших времен".

Возможен вариант, когда заглушки оставляются в конечной программе сознательно, в расчете на подключение в дальнейшем к работающей программе новых подпрограмм, реализующих некоторые новые возможности, либо предполагая возможное подключение к программе тестирующих средств для более точной настройки программы. Кто может дать гарантию, что в один прекрасный день такой заглушкой кто-нибудь не воспользуется для подключения к программе совсем иной подпрограммы, работающей в интересах этого "кого-нибудь", а не законного владельца?

Наконец, еще одним распространенным источником люков является так называемый "неопределенный ввод". Не так уж редка ситуация, когда программа создается неопытным программистом, исходящим из предположения, что пользователи будут работать с его программой всегда корректно. В этом случае реакция на неопределенный ввод может быть в лучшем случае непредсказуемой; гораздо хуже, если

программа в случае одинакового неопределенного ввода выполняет некоторые повторяющиеся действия - это дает потенциальному захватчику возможность планировать свои действия по нарушению безопасности.

Таким образом, люк (или люки) может присутствовать в программе ввиду того, что программист:

1. забыл удалить его;
2. умышленно оставил его в программе для обеспечения тестирования или выполнения оставшейся части отладки;
3. умышленно оставил его в программе в интересах облегчения окончательной сборки конечного программного продукта;
4. умышленно оставил его в программе с тем, чтобы иметь скрытое средство доступа к программе уже после того, как она вошла в состав конечного продукта.

Троянский конь

Существуют программы, реализующие, помимо функций, описанных в документации, и некоторые другие функции, в документации не описанные. Такие программы называются "троянскими конями".

Логическая бомба

"Логической бомбой" обычно называют программу или даже участок кода в программе, реализующий некоторую функцию при выполнении определенного условия.

Мировая компьютерная общественность достаточно хорошо знакома с логическими бомбами. Логическая бомба является одним из излюбленных способов мести программистов компаниям, которые их уволили или чем-либо обидели. При этом чаще всего срабатывание бомбы ставится в зависимость от установки в системе даты - так называемые "часовые" бомбы. Это очень удобно: допустим, программист знает, что его уволят 1 марта; в таком случае он может установить "часовую" бомбу на взрыв, допустим, 6 июля или даже на Рождество, когда сам он будет уже вне пределов досягаемости для пострадавшей компании.

В этом отношении интересна высказанная одним из администраторов систем мысль, что при увольнении системного программиста будет лучше, если глава фирмы проводит его до дверей офиса, вежливо попрощается и подаст пальто.

Программные закладки

Для того чтобы закладка смогла выполнить какие-либо функции по отношению к другой прикладной программе, она должна получить управление на себя, то есть процессор должен начать выполнять инструкции, относящиеся к коду закладки.

Это возможно только при одновременном выполнении двух условий:

- закладка должна находиться в оперативной памяти до начала работы программы, на которую направлено ее воздействие;

- закладка должна активизироваться по некоторому общему для закладки и для прикладной программы событию.

Это достигается путем анализа и обработки закладкой общих относительно нее и прикладной программы событий, например, прерываний. Причем данные события должны сопровождать работу прикладной программы или работу всей ПЭВМ.

Исполнение кода закладки может быть сопровождено операциями несанкционированной записи (НСЗ), например, для сохранения некоторых фрагментов информации, и несанкционированного считывания (НСЧ), которое может происходить отдельно от операций чтения прикладной программы или совместно с ними. При этом операции считывания и записи, возможно, не связаны с получением конфиденциальной информации, например, считывание параметров устройства или его инициализация (закладка может использовать для своей работы и такие операции, в частности, для инициирования сбойных ситуаций или переназначения ввода-вывода).

Несанкционированная запись закладкой может происходить:

- в массив данных, не совпадающий с пользовательской информацией - сохранение информации;
- в массив данных, совпадающий с пользовательской информацией или ее подмножеством - искажение, уничтожение или навязывание информации закладкой.

Следовательно, можно рассматривать три основные группы деструктивных функций, которые могут выполняться закладками:

- сохранение фрагментов информации, возникающей при работе пользователей, прикладных программ, вводе-выводе данных, во внешней памяти сети (локальной или удаленной) или выделенной ПЭВМ, в том числе различных паролей, ключей и кодов доступа, собственно конфиденциальных документов в электронном виде;
- изменение алгоритмов функционирования прикладных программ (то есть целенаправленное воздействие во внешней или оперативной памяти), например, программа разграничения доступа станет пропускать пользователей по любому паролю;
- навязывание некоторого режима работы (например, при уничтожении информации - блокирование записи на диск, при этом информация, естественно, не уничтожается), либо замена записываемой информации информацией, навязанной закладкой (например, при выводе на экран слово "неверно" заменяется словом "верно", а "рубль" - "доллар" и т.д.).

Приведем несколько важных примеров. Предположим, что программное средство производит некоторые файловые операции. Для этого открывается файл, часть его считывается в буфер оперативной памяти, обрабатывается и затем, записывается в файл с прежним или новым именем.

Теперь представьте себе, что вводимые вами документы не записываются на диск или записываются в искаженном виде, либо сугубо конфиденциальная информация банка помимо записи в базу данных дополнена к файлу, посланному в сеть. Или вы дали команду зашифровать файл для передачи, а файл отправился "в путь" незашифрованным.

Таковы лишь некоторые из широко известных негативных действий, могущих производиться закладками с файлами-документами.

Рассмотренные действия особенно опасны для программ подтверждения подлинности электронных документов ("электронная цифровая подпись" - ЭЦП). При считывании приготовленного для подписи файла-документа может произойти изменение имени автора, даты, времени, цифровых данных, заголовка документа (например, изменение суммы платежа в платежных поручениях и др.)

Закладки такого типа могут, например, навязывать истинность электронной подписи, даже если файл был изменен.

Широко известна и использовалась во многих банках система ЭЦП Pretty Good Privace (PGP). Многие стали жертвой простейшей программной закладки против этой системы.

Рассмотрим процесс "электронного подписывания", реализованный в программе PGP. Программа считывает файл для вычисления хеш-функции блоками по 512 байт, причем завершением процесса чтения является считывание блока меньшей длины.

Работа закладки была основана на навязывании длины файла. Закладка позволяет программе ЭЦП считать только первый 512-байтный блок и вычислять подпись только на его основе.

Такая же схема действует и при проверке подписи. Следовательно, основная часть файла может быть произвольным образом искажена.

Практика применения ЭЦП в системах автоматизированного финансового документооборота показала, что именно программная реализация ЭЦП наиболее сильно подвержена влиянию со стороны программных закладок, которые позволяют осуществлять проводки заведомо фальшивых финансовых документов и вмешиваться в порядок разрешения споров по факту применения ЭЦП.

Отметим четыре основных метода воздействия программных закладок на ЭЦП:

1. Метод навязывания входной информации - связан с искажением поступающего на подпись файла.
2. Метод навязывания результата проверки - связан с влиянием на признак правильности подписи независимо от результатов работы.
3. Метод навязывания длины сообщения - предъявление программе ЭЦП электронного документа меньшей длины, следовательно, производится подпись только части документа.
4. Метод искажения программы ЭЦП - связан с изменением исполняемого кода самой программы ЭЦП.

Задача борьбы с программными закладками весьма сложна и многопланова. Можно рассматривать следующие условия ее решения:

1. Неизвестно наличие в каком-либо множестве программ фрагментов закладок, ставится задача определения факта их наличия или отсутствия; при этом программы не выполняются (статическая задача).

2. В условиях п.1 программы используются по своему назначению, ставится та же задача выявления закладки, но в данном случае- по результатам работы (динамическая задача).
3. Происходит обмен программным продуктом (в пространстве - передача по каналу связи или на материальном носителе, во времени - хранение), свободным от потенциально опасных действий, программный продукт не исполняется, задача защиты (статическая) ставится в трех вариантах:
    - не допустить внедрения закладки;
    - выявить внедренный код закладки;
    - удалить внедренный код закладки.
4. В условиях п.3 решается динамическая задача - защита от воздействия закладки в ходе работы программ.
5. В условиях потенциальной возможности воздействия закладок решается задача борьбы с их итоговым влиянием, то есть закладки присутствуют в системе, но либо не активны при выполнении критических действий прикладных программ, либо результат их воздействия не конструктивен.

Далее рассмотренные задачи будем упоминать как Задачи 1-5.

С другой стороны, методы борьбы с воздействием закладок можно разделить на классы и увязать с проблемой защиты программного обеспечения вообще:

1. Общие методы защиты программного обеспечения, решающие задачи борьбы со случайными сбоями оборудования и несанкционированным доступом:
    a. Контроль целостности системных областей, запускаемых прикладных программ и используемых данных (решение Задачи 3).
    b. Контроль критических для безопасности системы событий (решение Задачи 2).
    Данные методы действенны лишь тогда, когда контрольные элементы не подвержены воздействию закладок и разрушающее воздействие входит в контролируемый класс. Так, например, система контроля за вызовом прерываний не будет отслеживать обращение на уровне портов. С другой стороны, контроль может быть обойден путем:
        - навязывания конечного результата проверок;
        - влияния на процесс считывания информации;
        - изменения хеш-функций, хранящихся в общедоступных файлах или в оперативной памяти.

    Важно, что включение процесса контроля должно быть выполнено до начала влияния закладки, либо контроль должен осуществляться полностью аппаратно-программными средствами, содержащимися в ПЗУ.

    c. Создание безопасной и изолированной операционной среды (решение Задачи 4).
    d. Предотвращение результирующего воздействия вируса или закладки (например, запись на диск только в зашифрованном на уровне контроллера виде - тем самым сохранение информации закладкой не имеет смысла, либо запрет записи на диск на аппаратном уровне) (решение Задачи 5).

2. Специальные методы выявления программ с потенциально опасными последствиями:
    a. Поиск фрагментов кода по характерным последовательностям (сигнатурам), свойственным закладкам, либо, наоборот, разрешение на выполнение или внедрение в цепочку прерываний только программам с известными сигнатурами (решение Задач 1,2).
    b. Поиск критических участков кода методом семантического анализа, то есть анализа фрагментов кода на выполняемые ими функции, например, выполнение НСЗ, часто сопряжено с дисассемблированием или эмуляцией выполнения (решение Задач 1,2).

Весьма важным является комплекс организационно-технических мер защиты от вирусов и программных закладок.

Меры защиты можно подразделить на две основные группы:

I. Меры защиты на этапе разработки программного обеспечения (ПО) системы.
II. Меры защиты на этапе эксплуатации.

В группу I входят:

1. Меры защиты на этапе разработки прикладного ПО, содержащего внутреннюю защиту от НСД.
   Они направлены на выявление в исходных текстах программ коммуникации и доступа некоторых фрагментов или подпрограмм, облегчающих или не регистрирующих доступ разработчиков программ (вход по фиксированным паролям, беспарольный доступ по нажатию некоторых клавиш, обход регистрации пользователей с фиксированными именами и т.д.). Наличие таких фрагментов фактически сведет на нет весь комплекс информационной безопасности системы, поскольку доступ через них возможен как человеку, так и программе-закладке.
2. Меры защиты при разработке ПО защиты от НСД (должны быть предусмотрены меры по проверке целостности хранимых на внешних носителях программных средств защиты, контроль целостности их в оперативной памяти и т.д.).

В группу II входят:

1. Регулярные меры защиты и контроля, применяемые постоянно с фиксированными временными интервалами.
2. Эпизодические защитные мероприятия (в дополнение к п.1 в период повышения опасности вирусного нападения).
3. Локализационно-восстановительные меры, применяемые в случае проникновения и обнаружения закладок и причинения ими негативных последствий.

К общим способам защиты системы относятся:

a. ограничение физического доступа к программам и оборудованию путем установления соответствующего организационного режима и применения аппаратных или программных средств ограничения доступа к ПЭВМ и ее компонентам;

b. при активизации прикладного ПО контроль его целостности, целостности областей DOS, BIOS и CMOS путем просчета контрольных сумм (вычисления хеш-функций) и сравнения их с эталонными значениями для каждой ПЭВМ;
c. максимальное ограничение и контроль за передачей по сети исполняемых файлов (типа .EXE и .COM), .SYS- и .BIN-файлов в целях предотвращения распространения файловых вирусов, вирусов типа Driver, загрузочно-файловых вирусов, а также размножающихся закладок по сети; использование фильтров и шлюзов при передаче данных;
d. организация выборочного и внезапного контроля работы операторов ПЭВМ с целью выявления фактов использования нерегламентированного ПО;
e. защита от записи на сменных носителях, учет и надежное хранение архивных копий;
f. немедленное уничтожение ценной информации сразу по истечении потребности в ней;
g. периодическая оптимизация внешних носителей (винчестеров) на предмет выявления сбойных или псевдосбойных кластеров и стирания фрагментов конфиденциальной информации при помощи средств типа SPEED DISK.

Средства защиты, учитывающие специфику работы фрагментов системы:

a. для коммуникационных подсистем:
    - средства и методы повышения общей надежности системы (программное или аппаратное дублирование, использование "горячего резерва" и т.д.);
b. для серверов локальных сетей (СЛС):
    - контроль состава и порядка использования ПО, находящегося на СЛС;
    - дублирование стандартных средств защиты от НСД, входящих в состав сетевого ПО;
    - запрет записи на общий диск файл-сервера локальной сети исполняемых файлов, не имеющих отношения к обработке информации в сети.

Что касается мер защиты от вирусов и закладок в процессе разработки самих программ защиты (п.2 группы I), то в данном случае необходимо предусмотреть:

- встроенный самоконтроль ПО системы защиты, установленный на сети, путем просчета контрольных сумм по файлам и коду программ в оперативной памяти;
- переопределение "на себя" существенно важных прерываний (int01h, 03h, 08h, 10h, 13h, 21h) для предотвращения перехвата ввода ключей и паролей и их сохранения закладкой на внешнем носителе, а также блокирование проникновения в логику работы программ защиты при помощи стандартных отладочных средств;
- защиту от переноса установленного ПО защиты коммуникации на другую ПЭВМ, проводимого с целью детального изучения ПО и поиска обходных путей для преодоления защиты; это может быть достигнуто "привязкой" ПО к индивидуальным параметрам ПЭВМ - тем самым работоспособность ПО будет обеспечиваться только на данной ЭВМ сети.

Атака салями

А теперь поговорим о биче банковских компьютерных систем - атаке "салями".

Чтобы понять смысл такой атаки, полезно вспомнить технологию изготовления известного сорта колбасы, которая создается путем соединения в общее целое множества мелких кусочков мяса. Получается достаточно вкусно.

При разработке банковских систем устанавливается правило округления (или усечения), используемое при выполнении всех операций. Вся хитрость состоит в том, как запрограммировать обработку округлений. Можно, конечно, просто удалять несуществующие величины. Но можно и не удалять, а накапливать на некоем специальном счете. Там пол цента, тут пол цента... - а в сумме? Как свидетельствует практика, сумма, составленная буквально из ничего, за пару лет эксплуатации "хитрой" программы в среднем по размеру банке, может исчисляться тысячами долларов.

Можно сказать, что атака салями - компьютерная реализация известной поговорки "С миру по нитке - голому рубаха".

Программные закладки при хаотичной "информатизации" финансовой сферы становятся мощным деструктивным фактором в ее развитии. Трудность обнаружения закладок и борьбы с их воздействием без преувеличения позволяет назвать их информационным оружием.

Легко понять недоумение и огорчение банкиров, видящих заведомо фальшивые платежные поручения, которые системы электронной подписи считают подлинными.

Чтобы избежать подобных разочарований, необходимо постоянно помнить об информационной угрозе и еще ... легенду о троянском коне.

Защита в интернет

Межсетевые экраны

Межсетевой экран или ***брандмауэр*** (по-нем. ***brandmauer***, по-англ. ***firewall***, по-рус. *граница огня*) - это система или комбинация систем, позволяющих разделить сеть на две или более частей и реализовать набор правил, определяющих условия прохождения пакетов из одной части в другую (см. рис.1). Чаще всего эта граница проводится между ***локальной сетью*** предприятия и ***INTERNET***, хотя ее можно провести и внутри локальной сети предприятия. Брандмауэр, таким образом, пропускает через себя весь трафик. Для каждого проходящего пакета брандмауэр принимает решение пропускать его или отбросить. Для того чтобы брандмауэр мог принимать эти решения, ему необходимо определить набор правил. О том, как эти правила описываются и какие параметры используются при их описании, речь пойдет чуть позже.

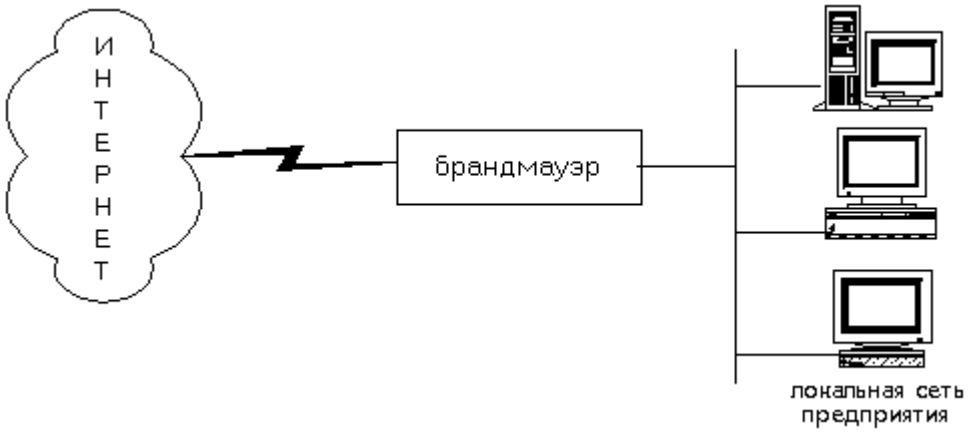


рис.1

Как правило, брандмауэры функционируют на какой-либо UNIX платформе - чаще всего это BSDI, SunOS, AIX, IRIX и т.д., реже - DOS, VMS, WNT, Windows NT. Из аппаратных платформ встречаются INTEL, Sun SPARC, RS6000, Alpha, HP PA-RISC, семейство RISC процессоров R4400-R5000. Помимо Ethernet, многие брандмауэры поддерживают FDDI, Token Ring, 100Base-T, 100VG-AnyLan, различные серийные устройства. Требования к оперативной памяти и объему жесткого диска зависят от количества машин в защищаемом сегменте сети.

Обычно в операционную систему, под управлением которой работает брандмауэр, вносятся изменения, цель которых - повышение защиты самого брандмауэра. Эти изменения затрагивают как ядро ОС, так и соответствующие файлы конфигурации. На самом брандмауэре не разрешается иметь счета пользователей (а значит и потенциальных дыр), только счет администратора. Некоторые брандмауэры работают только в однопользовательском режиме. Многие брандмауэры имеют систему проверки целостности программных кодов. При этом контрольные суммы программных кодов хранятся в защищенном месте и сравниваются при старте программы во избежание подмены программного обеспечения.

Все брандмауэры можно разделить на три типа:

- пакетные фильтры (packet filter);

- сервера прикладного уровня (application gateways);
- сервера уровня соединения (circuit gateways).

Все типы могут одновременно встретиться в одном брандмауэре.

Пакетные фильтры

Брандмауэры с пакетными фильтрами принимают решение о том, пропускать пакет или отбросить, просматривая IP-адреса, флаги или номера TCP портов в заголовке этого пакета. IP-адрес и номер порта - это информация сетевого и транспортного уровней соответственно, но пакетные фильтры используют и информацию прикладного уровня, т.к. все стандартные сервисы в TCP/IP ассоциируются с определенным номером порта.

Для описания правил прохождения пакетов составляются таблицы типа:

Действие	тип пакета	адрес источн.	порт источн.	адрес назнач.	порт назнач.	флаги

Поле "действие" может принимать значения пропустить или отбросить.
Тип пакета - TCP, UDP или ICMP.
Флаги - флаги из заголовка IP-пакета.
Поля "порт источника" и "порт назначения" имеют смысл только для TCP и UDP пакетов.

Сервера прикладного уровня

Брандмауэры с серверами прикладного уровня используют сервера конкретных ***сервисов*** (proxy server) - TELNET, FTP и т.д., запускаемые на брандмауэре и пропускающие через себя весь трафик, относящийся к данному сервису. Таким образом, между клиентом и сервером образуются два соединения: от клиента до брандмауэра и от брандмауэра до места назначения.

Полный набор поддерживаемых серверов различается для каждого конкретного брандмауэра, однако чаще всего встречаются сервера для следующих сервисов:

- терминалы (Telnet, Rlogin);
- передача файлов (Ftp);
- электронная почта (SMTP, POP3);
- WWW (HTTP);
- Gopher;
- Wais;
- X Window System (X11);
- сетевая печать (LP);
- удаленное выполнение задач (Rsh);
- Finger;
- новости Usenet (NNTP);
- Whois;
- RealAudio.

Использование серверов прикладного уровня позволяет решить важную задачу - скрыть от внешних пользователей структуру локальной сети, включая информацию в заголовках почтовых пакетов или службы доменных имен (DNS). Другим положительным качеством является возможность аутентификации на пользовательском уровне

(напоминаю, что аутентификация - процесс подтверждения идентичности чего-либо; в данном случае это процесс подтверждения, действительно ли пользователь является тем, за кого он себя выдает).

При описании правил доступа используются такие параметры, как

- название сервиса,
- имя пользователя,
- допустимый временной диапазон использования сервиса,
- компьютеры, с которых можно пользоваться сервисом,
- схемы аутентификации.

Сервера прикладного уровня позволяют обеспечить наиболее высокий уровень защиты, т.к. взаимодействие с внешним миров реализуется через небольшое число прикладных программ, полностью контролирующих весь входящий и выходящий трафик.

Сервера уровня соединения

Сервер уровня соединения представляет из себя транслятор TCP соединения. Пользователь образует соединение с определенным портом на брандмауэре, после чего последний производит соединение с местом назначения по другую сторону от брандмауэра. Во время сеанса этот транслятор копирует байты в обоих направлениях, действуя как провод.

Как правило, пункт назначения задается заранее, в то время как источников может быть много (соединение типа один - много). Используя различные порты, можно создавать различные конфигурации.

Такой тип сервера позволяет создавать транслятор для любого определенного пользователем сервиса, базирующегося на TCP, осуществлять контроль доступа к этому сервису, сбор статистики по его использованию.

Сравнительные характеристики пакетных фильтров и серверов прикладного уровня

Ниже приведены основные достоинства и недостатки пакетных фильтров и серверов прикладного уровня относительно друг друга.

Достоинства пакетных фильтров:

- относительно невысокая стоимость;
- гибкость в определении правил фильтрации;
- небольшая задержка при прохождении пакетов.

Недостатки пакетных фильтров:

- локальная сеть видна (маршрутизируется) из INTERNET;
- правила фильтрации пакетов трудны в описании, требуются очень хорошие знания технологий TCP и UDP;
- при нарушении работоспособности брандмауэра все компьютеры за ним становятся полностью незащищенными либо недоступными;

- аутентификацию с использованием IP-адреса можно обмануть использованием IP-спуфинга (атакующая система выдает себя за другую, используя ее IP-адрес);
- отсутствует аутентификация на пользовательском уровне.

Достоинства серверов прикладного уровня:

- локальная сеть невидима из INTERNET;
- при нарушении работоспособности брандмауэра пакеты перестают проходить через брандмауэр, тем самым не возникает угрозы для защищаемых им машин;
- защита на уровне приложений позволяет осуществлять большое количество дополнительных проверок, снижая тем самым вероятность взлома с использованием дыр в программном обеспечении;
- аутентификация на пользовательском уровне может быть реализована система немедленного предупреждения о попытке взлома.

Недостатки серверов прикладного уровня:

- более высокая, чем для пакетных фильтров стоимость;
- невозможность использовании протоколов RPC и UDP;
- производительность ниже, чем для пакетных фильтров.

Виртуальные сети

Ряд брандмауэров позволяет также организовывать виртуальные корпоративные сети (***Virtual Private Network***), т.е. объединить несколько локальных сетей, включенных в INTERNET в одну виртуальную сеть. ***VPN*** позволяют организовать прозрачное для пользователей соединение локальных сетей, сохраняя секретность и целостность передаваемой информации с помощью шифрования. При этом при передаче по INTERNET шифруются не только данные пользователя, но и сетевая информация - сетевые адреса, номера портов и т.д.

Схемы подключения брандмауэров

Для подключения брандмауэров используются различные схемы. Брандмауэр может использоваться в качестве внешнего роутера, используя поддерживаемые типы устройств для подключения к внешней сети (см. рис.1). Иногда используется схема, изображенная на рис.2, однако пользоваться ей следует только в крайнем случае, поскольку требуется очень аккуратная настройка роутеров и небольшие ошибки могут образовать серьезные дыры в защите.

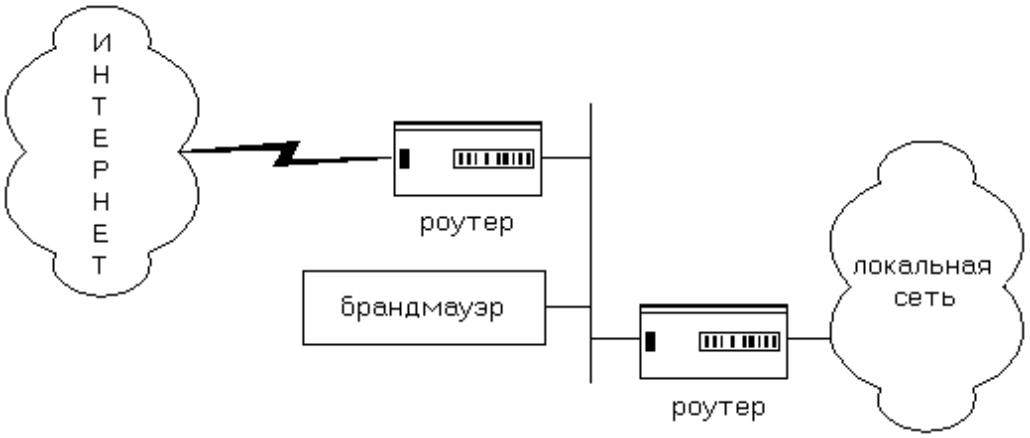


рис.2

Чаще всего подключение осуществляется через внешний маршрутизатор, поддерживающий два Ethernet интерфейса(так называемый dual-homed брандмауэр) (две сетевые карточки в одном компьютре) (см. рис.3).

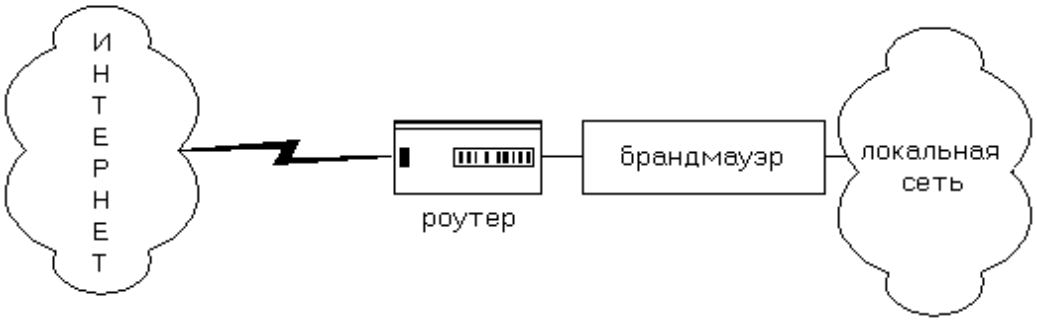


рис.3

При этом между внешним роутером и брандмауэром имеется только один путь, по которому идет весь трафик. Обычно роутер настраивается таким образом, что брандмауэр является единственной видимой снаружи машиной. Эта схема является наиболее предпочтительной с точки зрения безопасности и надежности защиты.

Другая схема представлена на рис.4.

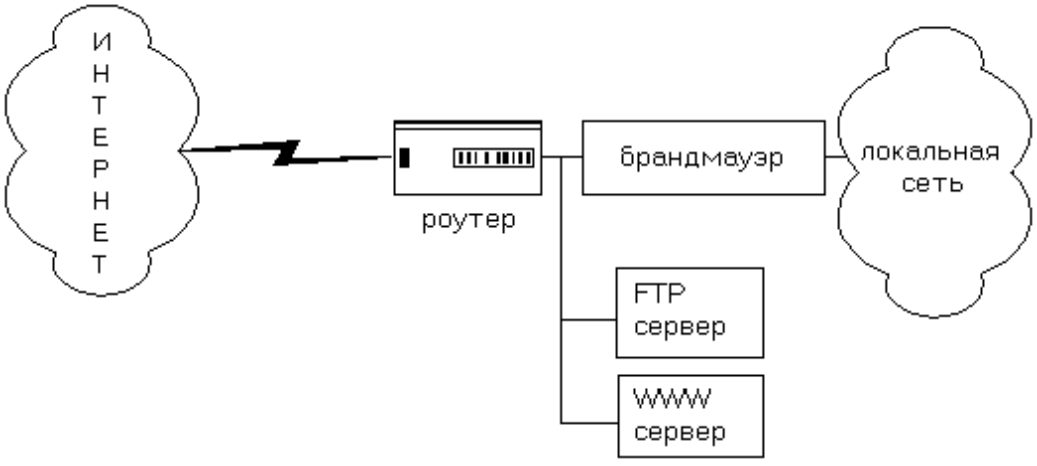


рис.4

При этом брандмауэром защищается только одна подсеть из нескольких выходящих из роутера. В незащищаемой брандмауэром области часто располагают серверы, которые должны быть видимы снаружи (WWW, FTP и т.д.). Большинство брандмауэров позволяет разместить эти сервера на нем самом - решение, далеко не лучшее с точки зрения загрузки машины и безопасности самого брандмауэра.

Существуют решения (см. рис.5), которые позволяют организовать для серверов, которые должны быть видимы снаружи, третью сеть; это позволяет обеспечить контроль за доступом к ним, сохраняя в то же время необходимый уровень защиты машин в основной сети.

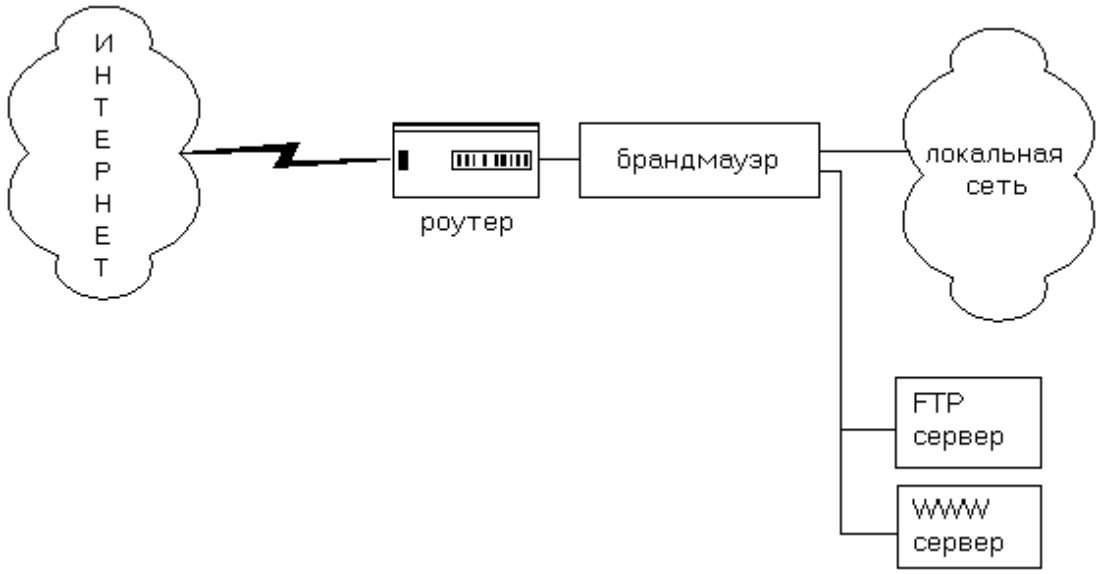


рис.5

При этом достаточно много внимания уделяется тому, чтобы пользователи внутренней сети не могли случайно или умышленно открыть дыру в локальную сеть через эти сервера. Для повышения уровня защищенности возможно использовать в одной сети несколько брандмауэров, стоящих друг за другом.

Администрирование

Легкость администрирования является одним из ключевых аспектов в создании эффективной и надежной системы защиты. Ошибки при определении правил доступа могут образовать дыру, через которую может быть взломана система. Поэтому в большинстве брандмауэров реализованы сервисные утилиты, облегчающие ввод, удаление, просмотр набора правил. Наличие этих утилит позволяет также производить проверки на синтаксические или логические ошибки при вводе или редактирования правил. Как правило, эти утилиты позволяют просматривать информацию, сгруппированную по каким-либо критериям, например, все что относится к конкретному пользователю или сервису.

Системы сбора статистики и предупреждения об атаке

Еще одним важным компонентом брандмауэра является система сбора статистики и предупреждения об атаке. Информация обо всех событиях - отказах, входящих, выходящих соединениях, числе переданных байт, использовавшихся сервисах, времени

соединения и т.д. - накапливается в файлах статистики. Многие брандмауэры позволяют гибко определять подлежащие протоколированию события, описать действия брандмауэра при атаках или попытках несанкционированного доступа - это может быть сообщение на консоль, почтовое послание администратору системы и т.д. Немедленный вывод сообщения о попытке взлома на экран консоли или администратора может помочь, если попытка оказалась успешной и атакующий уже проник в систему. В состав многих брандмауэров входят генераторы отчетов, служащие для обработки статистики. Они позволяют собрать статистику по использованию ресурсов конкретными пользователями, по использованию сервисов, отказам, источникам, с которых проводились попытки несанкционированного доступа и т.д.

Аутентификация

Аутентификация является одним из самых важных компонентов брандмауэров. Прежде чем пользователю будет предоставлено право воспользоваться тем или иным сервисом, необходимо убедиться, что он действительно тот, за кого он себя выдает.

Как правило, используется принцип, получивший название "что он знает" - т.е. пользователь знает некоторое секретное слово, которое он посылает серверу аутентификации в ответ на его запрос.

Одной из схем аутентификации является использование стандартных UNIX паролей. Эта схема является наиболее уязвимой с точки зрения безопасности - пароль может быть перехвачен и использован другим лицом.

Классы защищенности брандмауэров

Применительно к обработке конфиденциальной информации автоматизированные системы (АС) делятся на три группы:

1. Многопользовательские АС, обрабатывающие информацию различных уровней конфиденциальности.
2. Многопользовательские АС, в которых все пользователи имеют равный доступ ко всей обрабатываемой информации, расположенной на носителях разного уровня конфиденциальности.
3. Однопользовательские АС, в которых пользователль имеет подный доступ ко всей обрабатываемой информации, расположенной на носителях разного уровня конфиденциальности.

В первой группе выделяют 5 классов защищенности АС: 1А, 1Б, 1В, 1Г, 1Д, во второй и третьей группах - по 2 класса защищенности: 2А, 2Б и 3А, 3Б сооответственно. Класс А соответствует максимальной, класс Д - минимальной защищенности АС.

Брандмауэры позволяют поддерживать безопасность объектов внутренней области, игнорируя несанкционированные запросы из внешней области, т.е. осуществляют *экранирование*. В результате уменьшается уязвимость внутренних объектов, поскольку первоначально сторонний нарушитель должен преодолеть брандмауэр, где защитные механизмы сконфигурированы особенно тщательно и жестко. Кроме того, экранирующая система в отличие от универсальной устроена более простым, а следовательно, более безопасным образом. На ней присутствуют только те компоненты, которые необходимы для выполнения функций экранирования. Экранирование дает также возможность контролировать информационные потоки, направленные во внешнюю область, что

способствует поддержанию во внутренней области режима конфиденциальности. Помимо функций разграничения доступа, брандмауэры осуществляют регистрацию информационных потоков.

По уровню защищенности брандмауэры делятся на 5 классов. Самый низкий класс защищенности - пятый. Он применяется для безопасного взаимодействия АС класса 1Д с внешней средой, четвертый - для 1Г, третий - для 1В, второй - для 1Б, самый высокий - первый - для 1А.

Для АС класса 2Б, 3Б применяются брандмауэры не ниже пятого класса.

Для АС класса 2А, 3А в зависимости от важности обрабатываемой информации применяются брандмауэры следующих классов:

- при обработки информации с грифом "секретно" - не ниже третьего класса;
- при обработки информации с грифом "совершенно секретно" - не ниже второго класса;
- при обработки информации с грифом "особой важности" - только первого класса.

Показатели защищенности сведены в табл.1.

Обозначения:

"-" - нет требований к данному классу;

"+" - новые или дополнительные требования;

"=" - требования совпадают с требованиями к предыдущему классу.

Таблица 1

Показатели защищенности	**Классы защищенности**				
	5	**4**	**3**	**2**	**1**
Управление доступом (фильтрация данных и трансляция адресов)	+	+	+	+	=
Идентификация и аутентификация	-	-	+	=	+
Регистрация	-	+	+	+	=
Администрирование: идентификация и аутентификация	+	=	+	+	+
Администрирование: регистрация	+	+	+	=	=
Администрирование: простота использования	-	-	+	=	+
Целостность	+	=	+	+	+
Восстановление	+	=	=	+	=
Тестирование	+	+	+	+	+
Руководство администратора защиты	+	=	=	=	=
Тестовая документация	+	+	+	+	+
Конструкторская (проектная) документация	+	=	+	=	+

Руководство для приобретающих брандмауэр

Исследовательским подразделением компании TruSecure - лабораторией ICSA - разработан документ "Firewall Buyers Guide" (Гид покупателей межсетевого экрана). В одном из разделов этого документа дана следующая форма оценки покупателя:

1. Контактная информация - адрес и ответственные лица.
2. Бизнес-среда работы:
    - количество и расположение отдельных учреждений (зданий) предприятия;
    - указание подразделений и информации ограниченного характера и информации, для которой важна доступность данных для взаимодействия подразделений, их размещение;
    - Указание внешних партнеров, с которыми необходимо организовать взаимодействие;
    - описание сервисов, открытых публично;
    - трребования к организации удаленного доступа во внутреннее информационное пространство предприятия;
    - сервисы электронных служб, использующие публичные каналы связи (например, электронная коммерция).
3. Планируемые изменения в бизнес-среде по перечисленным параметрам.
4. Информационная среда:
    - количество пользовательских рабочих станций с указанием аппаратного обеспечения, системного и прикладного программного обеспечения;
    - структура сети с указанием топологии, среды передачи данных, используемых устройств и протоколов;
    - структура удаленного доступа с указанием используемых устройств, а также методов аутентификации;
    - количество серверов с указанием аппаратного обеспечения,системного и прикладного программного обеспечения;
    - существующая система поддержки информационных систем со стороны поставщиков с их указанием и границами сферы деятельности;
    - антивирусные системы и другие системы контроля программного обеспечения;
    - технология упрравления сетью и информационными системами;
    - аутентификационные технологии - список и описание.
5. Планируемые изменения в информационной среде по перечисленням параметрам.
6. Связь с Интернетом:
    - тип интернет-соединения;
    - существующие межсетевые экраны (если они есть);
    - Средства связи с внешней средой, используемые внутренними системами;
    - внутренние системы и сервисы, доступные извне;
    - серверы электронной коммерции и других транзакционных систем;
    - указание на наличие утвержденной политики безопасности доступа и использования Интернета.
7. Планируемые мероприятия (для которых приобретается межсетевой экран):
    - изменение в способах доступа к Интернету и в политике безопасности предприятия;

- появление новых протокоолов, которые необходимо поддерживать отдельно для внутренних пользователей, пользователей с удаленным доступом или специальных пользователей, доступных публично.

8. Требуемая функциональность межсетевого экрана:
   - по контролю доступа;
   - выдаваемым сообщениям;
   - аутентификации;
   - управлению конфигурацией;
   - контролю содержимого проходящего трафика;
   - регистрационным журналам;
   - распознаванию атак;
   - сетевым опциям (количество интерфейсов, способ доступа);
   - удаленному администрированию;
   - системным требованиям (под ключ, интеграция с другими продуктами и т.д.).
9. Прочие условия:
   - предполагаемая стоимость межсетевого экрана (сколько предприятие может потратить);
   - предполагаемая дата начала работы продукта;
   - требования к наличию у продукта сертификатов;
   - требования к предполагаемому администратору продукта и к службе поддержки;
   - специальные условия контракта (если есть);
   - другие замечания, которые не включены в данную форму.

Предполагается, что предприятие, заполнив данную форму и отправив ее производителю, позволит последнему сформировать наиболее качественное предложение для покупателя. Заполнение данной формы, впрочем, и без отправки ее кому-либо, позволит организации лучше понять, какое решение ей необходимо.

Компьютерные атаки и технологии их обнаружения

До сих пор нет точного определения термина "атака" (вторжение, нападение). Каждый специалист в области безопасности трактует его по-своему. Наиболее правильным и полным я считаю следующее определение.

> ***Атакой*** на информационную систему называются преднамеренные действия злоумышленника, использующие уязвимости информационной системы и приводящие к нарушению доступности, целостности и конфиденциальности обрабатываемой информации.

Устраним уязвимости информационной системы - устраним и возможность реализации атак.

На сегодняшний день считается неизвестным, сколько существует методов атак. Говорят о том, что до сих пор отсутствуют какие-либо серьезные математические исследования в этой области. Но еще в 1996 году Фред Коэн описал математические основы вирусной технологии. В этой работе доказано, что число вирусов бесконечно.

Очевидно, что и число атак бесконечно, поскольку вирусы - это подмножество множества атак.

Модели атак

Традиционная модель атаки строится по принципу "***один к одному***" (рис.1) или "***один ко многим***" (рис.2), т.е. атака исходит из одного источника. Разработчики сетевых средств защиты (межсетевых экранов, систем обнаружения атак и т.д.) ориентированы именно на традиционную модель атаки. В различных точках защищаемой сети устанавливаются агенты (сенсоры) системы защиты, которые передают информацию на центральную консоль управления. Это облегчает масштабирование системы, обеспечивает простоту удаленного управления и т.д. Однако такая модель не справляется с относительно недавно (в 1998 году) обнаруженной угрозой - распределенными атаками.

Рисунок 1. Отношение "один к одному"

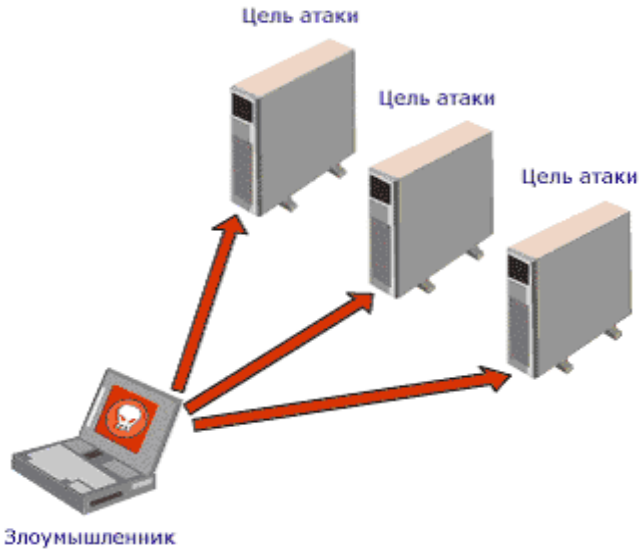


Рисунок 2. Отношение "один ко многим"

В модели распределенной атаки используются иные принципы. В отличие от традиционной модели ***в распределенной модели*** используются отношения "***многие к одному***" (рис.3) и "***многие ко многим***" (рис.4).

Рисунок 3. Отношение "многие к одному"

Рисунок 4. Отношение "многие ко многим"

Распределенные атаки основаны на "классических" атаках типа "отказ в обслуживании", а точнее на их подмножестве, известном как ***Flood-атаки*** или ***Storm-атаки*** (указанные термины можно перевести как "шторм", "наводнение" или "лавина"). Смысл данных атак заключается в посылке большого количества пакетов на атакуемый узел. Атакуемый узел может выйти из строя, поскольку он "захлебнется" в лавине посылаемых пакетов и не сможет обрабатывать запросы авторизованных пользователей. По такому принципу работают атаки SYN-Flood, Smurf, UDP Flood, Targa3 и т.д. Однако в том случае, если пропускная способность канала до атакуемого узла превышает пропускную способность атакующего или атакуемый узел некорректно сконфигурирован, то к "успеху" такая атака не приведет. Например, с помощью этих атак бесполезно пытаться нарушить работоспособность своего провайдера. Но распределенная атака происходит уже не из одной точки Internet, а сразу из нескольких, что приводит к резкому возрастанию трафика и выведению атакуемого узла из строя. Например, по данным России-Онлайн в течение двух суток, начиная с 9 часов утра 28 декабря 2000 г. крупнейший Internet-провайдер Армении "Арминко" подвергался распределенной атаке. В данном случае к атаке подключились более 50 машин из разных стран, которые посылали по адресу "Арминко" бессмысленные сообщения. Кто организовал эту атаку, и в какой

стране находился хакер - установить было невозможно. Хотя атаке подвергся в основном "Арминко", перегруженной оказалась вся магистраль, соединяющая Армению с всемирной паутиной. 30 декабря благодаря сотрудничеству "Арминко" и другого провайдера - "АрменТел" - связь была полностью восстановлена. Несмотря на это компьютерная атака продолжалась, но с меньшей интенсивностью.

Этапы реализации атак

Можно выделить следующие этапы реализации атаки:

1. предварительные действия перед атакой или "сбор информации",
2. собственно "реализация атаки",
3. завершение атаки.

Обычно, когда говорят об атаке, то подразумевают именно второй этап, забывая о первом и последнем. Сбор информации и завершение атаки ("заметание следов") в свою очередь также могут являться атакой и могут быть разделены на три этапа (см. рис.5).

Рисунок 5. Этапы реализации атаки

Сбор информации - это основной этап реализации атаки. Именно на данном этапе эффективность работы злоумышленника является залогом "успешности" атаки. Сначала выбирается цель атаки и собирается информация о ней (тип и версия операционной системы, открытые порты и запущенные сетевые сервисы, установленное системное и прикладное программное обеспечение и его конфигурация и т.д.). Затем идентифицируются наиболее уязвимые места атакуемой системы, воздействие на которые приводит к нужному злоумышленнику результату. Злоумышленник пытается выявить все каналы взаимодействия цели атаки с другими узлами. Это позволит не только выбрать тип реализуемой атаки, но и источник ее реализации. Например, атакуемый узел взаимодействует с двумя серверами под управлением ОС Unix и Windows NT. С одним сервером атакуемый узел имеет доверенные отношения, а с другим - нет. От того, через какой сервер злоумышленник будет реализовывать нападение, зависит, какая атака будет задействована, какое средство реализации будет выбрано и т.д. Затем, в зависимости от полученной информации и желаемого результата, выбирается атака, дающая наибольший эффект. Например:
SYN Flood, Teardrop, UDP Bomb - для нарушения функционирования узла;
CGI-скрипт - для проникновения на узел и кражи информации;
PHF - для кражи файла паролей и удаленного подбора пароля и т.п.

Традиционные средства защиты, такие как межсетевые экраны или механизмы фильтрации в маршрутизаторах, вступают в действие лишь на втором этапе реализации атаки, совершенно "забывая" о первом и третьем. Это приводит к тому, что зачастую совершаемую атаку очень трудно остановить даже при наличии мощных и дорогих средств защиты. Пример тому - распределенные атаки. Логично было бы, чтобы средства защиты начинали работать еще на первом этапе, т.е. предотвращали бы возможность сбора информации об атакуемой системе. Это позволило бы если и не полностью предотвратить атаку, то хотя бы существенно усложнить работу злоумышленника.

Традиционные средства также не позволяют обнаружить уже совершенные атаки и оценить ущерб после их реализации, т.е. не работают на третьем этапе реализации атаки. Следовательно, невозможно определить меры по предотвращению таких атак впредь.

В зависимости от желаемого результата нарушитель концентрируется на том или ином этапе реализации атаки. Например:
для отказа в обслуживании подробно анализируется атакуемая сеть, в ней выискиваются лазейки и слабые места;
для хищения информации основное внимание уделяется незаметному проникновению на атакуемые узлы при помощи обнаруженных ранее уязвимостей.

Рассмотрим основные механизмы реализации атак. Это необходимо для понимания методов обнаружения этих атак. Кроме того, понимание принципов действий злоумышленников - залог успешной обороны сети.

1. Сбор информации

Первый этап реализации атак - это сбор информации об атакуемой системе или узле. Он включает такие действия как определение сетевой топологии, типа и версии операционной системы атакуемого узла, а также доступных сетевых и иных сервисов и т.п. Эти действия реализуются различными методами.

Изучение окружения

На этом этапе нападающий исследует сетевое окружение вокруг предполагаемой цели атаки. К таким областям, например, относятся узлы Internet-провайдера "жертвы" или узлы удаленного офиса атакуемой компании. На этом этапе злоумышленник может пытаться определить адреса "доверенных" систем (например, сеть партнера) и узлов, которые напрямую соединены с целью атаки (например, маршрутизатор ISP) и т.д. Такие действия достаточно трудно обнаружить, поскольку они выполняются в течение достаточно длительного периода времени и снаружи области, контролируемой средствами защиты (межсетевыми экранами, системами обнаружения атак и т.п.).

Идентификация топологии сети

Существует два основных метода определения топологии сети, используемых злоумышленниками:

1. изменение TTL (TTL modulation),
2. запись маршрута (record route).

По первому методу работают программы traceroute для Unix и tracert для Windows. Они используют поле Time to Live ("время жизни") в заголовке IP-пакета, которое изменяется в зависимости от числа пройденных сетевым пакетом маршрутизаторов. Для записи маршрута ICMP-пакета может быть использована утилита ping . Зачастую сетевую топологию можно выяснить при помощи протокола SNMP, установленного на многих сетевых устройствах, защита которых неверно сконфигурирована. При помощи протокола RIP можно попытаться получить информацию о таблице маршрутизации в сети и т.д.

Многие из этих методов используются современными системами управления (например, HP OpenView, Cabletron SPECTRUM, MS Visio и т.д.) для построения карт

сети. И эти же методы могут быть с успехом применены злоумышленниками для построения карты атакуемой сети.

Идентификация узлов

Идентификация узла, как правило, осуществляется путем посылки при помощи утилиты ping команды ECHO_REQUEST протокола ICMP. Ответное сообщение ECHO_REPLY говорит о том, что узел доступен. Существуют свободно распространяемые программы, которые автоматизируют и ускоряют процесс параллельной идентификации большого числа узлов, например, fping или nmap. Опасность данного метода в том, что стандартными средствами узла запросы ECHO_REQUEST не фиксируются. Для этого необходимо применять средства анализа трафика, межсетевые экраны или системы обнаружения атак.

Это самый простой метод идентификации узлов. Однако он имеет два недостатка.

1. Многие сетевые устройства и программы блокируют ICMP-пакеты и не пропускают их во внутреннюю сеть (или наоборот не пропускают их наружу). Например, MS Proxy Server 2.0 не разрешает прохождение пакетов по протоколу ICMP. В результате возникает неполная картина. С другой стороны, блокировка ICMP-пакета говорит злоумышленнику о наличии "первой линии обороны" - маршрутизаторов, межсетевых экранов и т.д.
2. Использование ICMP-запросов позволяет с легкостью обнаружить их источник, что, разумеется, не может входить в задачу злоумышленника.

Существует еще один метод идентификации узлов - использование "смешанного" режима сетевой карты, который позволяет определить различные узлы в сегменте сети. Но он не применим в тех случаях, в которых трафик сегмента сети недоступен нападающему со своего узла, т.е. этот метод применим только в локальных сетях. Другим способом идентификации узлов сети является так называемая разведка DNS, которая позволяет идентифицировать узлы корпоративной сети при помощи обращения к серверу службы имен.

Идентификация сервисов или сканирование портов

Идентификация сервисов, как правило, осуществляется путем обнаружения открытых портов (port scanning). Такие порты очень часто связаны с сервисами, основанными на протоколах TCP или UDP. Например:

- открытый 80-й порт подразумевает наличие Web-сервера,
- 25-й порт - почтового SMTP-сервера,
- 31337-й - серверной части троянского коня BackOrifice,
- 12345-й или 12346-й - серверной части троянского коня NetBus и т.д.

Для идентификации сервисов и сканирования портов могут быть использованы различные программы, в т.ч. и свободно распространяемые. Например, nmap или netcat.

Идентификация операционной системы

Основной механизм удаленного определения ОС - анализ ответов на запросы, учитывающие различные реализации TCP/IP-стека в различных операционных системах. В каждой ОС по-своему реализован стек протоколов TCP/IP, что позволяет при помощи

специальных запросов и ответов на них определить, какая ОС установлена на удаленном узле.

Другой, менее эффективный и крайне ограниченный, способ идентификации ОС узлов - анализ сетевых сервисов, обнаруженных на предыдущем этапе. Например, открытый 139-й порт позволяет сделать вывод, что удаленный узел, вероятнее всего, работает под управлением ОС семейства Windows. Для определения ОС могут быть использованы различные программы. Например, nmap или queso.

Определение роли узла

Предпоследним шагом на этапе сбора информации об атакуемом узле является определение его роли, например, выполнении функций межсетевого экрана или Web-сервера. Выполняется этот шаг на основе уже собранной информации об активных сервисах, именах узлов, топологии сети и т.п. Например, открытый 80-й порт может указывать на наличие Web-сервера, блокировка ICMP-пакета указывает на потенциальное наличие межсетевого экрана, а DNS-имя узла proxy.domain.ru или fw.domain.ru говорит само за себя.

Определение уязвимостей узла

Последний шаг - поиск уязвимостей. На этом шаге злоумышленник при помощи различных автоматизированных средств или вручную определяет уязвимости, которые могут быть использованы для реализации атаки. В качестве таких автоматизированных средств могут быть использованы ShadowSecurityScanner, nmap, Retina и т.д.

2. Реализация атаки

С этого момента начинается попытка доступа к атакуемому узлу. При этом доступ может быть как непосредственный, т.е. проникновение на узел, так и опосредованный, например, при реализации атаки типа "отказ в обслуживании". Реализация атак в случае непосредственного доступа также может быть разделена на два этапа:

- проникновение;
- установление контроля.

Проникновение

Проникновение подразумевает под собой преодоление средств защиты периметра (например, межсетевого экрана). Реализовываться это может быть различными путями. Например, использование уязвимости сервиса компьютера, "смотрящего" наружу или путем передачи враждебного содержания по электронной почте (макровирусы) или через апплеты Java. Такое содержание может использовать так называемые "туннели" в межсетевом экране (не путать с туннелями VPN), через которые затем и проникает злоумышленник. К этому же этапу можно отнести подбор пароля администратора или иного пользователя при помощи специализированной утилиты (например, L0phtCrack или Crack).

Установление контроля

После проникновения злоумышленник устанавливает контроль над атакуемым узлом. Это может быть осуществлено путем внедрения программы типа "троянский конь" (например, NetBus или BackOrifice). После установки контроля над нужным узлом и "заметания" следов, злоумышленник может осуществлять все необходимые несанкционированные действия дистанционно без ведома владельца атакованного компьютера. При этом установление контроля над узлом корпоративной сети должно сохраняться и после перезагрузки операционной системы. Это может быть реализовано путем замены одного из загрузочных файлов или вставка ссылки на враждебный код в файлы автозагрузки или системный реестр. Известен случай, когда злоумышленник смог перепрограммировать EEPROM сетевой карты и даже после переустановки ОС он смог повторно реализовать несанкционированные действия. Более простой модификацией этого примера является внедрение необходимого кода или фрагмента в сценарий сетевой загрузки (например, для ОС Novell Netware).

Цели реализации атак

Необходимо отметить, что злоумышленник на втором этапе может преследовать две цели. Во-первых, получение несанкционированного доступа к самому узлу и содержащейся на нем информации. Во-вторых, получение несанкционированного доступа к узлу для осуществления дальнейших атак на другие узлы. Первая цель, как правило, осуществляется только после реализации второй. То есть, сначала злоумышленник создает себе базу для дальнейших атак и только после этого проникает на другие узлы. Это необходимо для того, чтобы скрыть или существенно затруднить нахождение источника атаки.

3. Завершение атаки

Этапом завершения атаки является "заметание следов" со стороны злоумышленника. Обычно это реализуется путем удаления соответствующих записей из журналов регистрации узла и других действий, возвращающих атакованную систему в исходное, "предатакованное" состояние.

Классификация атак

Существуют различные типа классификации атак. Например, деление на пассивные и активные, внешние и внутренние, умышленные и неумышленные. Однако дабы не запутать вас большим разнообразием классификаций, мало применимыми на практике, предлагаю более "жизненную" классификацию:

1. ***Удаленное проникновение (remote penetration)***. Атаки, которые позволяют реализовать удаленное управление компьютером через сеть. Например, NetBus или BackOrifice.
2. ***Локальное проникновение (local penetration)***. Атака, которая приводит к получению несанкционированного доступа к узлу, на котором она запущена. Например, GetAdmin.
3. ***Удаленный отказ в обслуживании (remote denial of service)***. Атаки, которые позволяют нарушить функционирование или перегрузить компьютер через Internet. Например, Teardrop или trin00.
4. ***Локальный отказ в обслуживании (local denial of service)***. Атаки, которые позволяют нарушить функционирование или перегрузить компьютер, на котором они реализуются. Примером такой атаки является "враждебный" апплет, который загружает центральный

процессор бесконечным циклом, что приводит к невозможности обработки запросов других приложений.

5. ***Сетевые сканеры (network scanners)***. Программы, которые анализируют топологию сети и обнаруживают сервисы, доступные для атаки. Например, система nmap.
6. ***Сканеры уязвимостей (vulnerability scanners)***. Программы, которые ищут уязвимости на узлах сети и которые могут быть использованы для реализации атак. Например, система SATAN или ShadowSecurityScanner.
7. ***Взломщики паролей (password crackers)***. Программы, которые "подбирают" пароли пользователей. Например, L0phtCrack для Windows или Crack для Unix.
8. ***Анализаторы протоколов (sniffers)***. Программы, которые "прослушивают" сетевой трафик. При помощи этих программ можно автоматически искать такую информацию, как идентификаторы и пароли пользователей, информацию о кредитных картах и т.д. Например, Microsoft Network Monitor, NetXRay компании Network Associates или LanExplorer.

Компания Internet Security Systems, Inc. еще больше сократила число возможных категорий, доведя их до 5:

1. Сбор информации (Information gathering).
2. Попытки несанкционированного доступа (Unauthorized access attempts).
3. Отказ в обслуживании (Denial of service).
4. Подозрительная активность (Suspicious activity).
5. Системные атаки (System attack).

Первые 4 категории относятся к удаленным атакам, а последняя - к локальным, реализуемом на атакуемом узле. Можно заметить, что в данную классификацию не попал целый класс так называемых "пассивных" атак ("прослушивание" трафика, "ложный DNS-сервер", "подмена ARP-сервера" и т.п.).

Классификация атак, реализованная во многих системах обнаружения атак, не может быть категоричной. Например, атака, реализация которой для ОС Unix (например, переполнение буфера statd) может иметь самые плачевные последствия (самый высокий приоритет), для ОС Windows NT может быть вообще не применима или иметь очень низкую степень риска. Кроме того, существует неразбериха и в самих названиях атак и уязвимостей. Одна и та же атака, может иметь разные наименования у разных производителей систем обнаружения атак.

Одной из лучших баз уязвимостей и атак является база данных X-Force, находящаяся по адресу: http://xforce.iss.net/. Доступ к ней может осуществляться как путем подписки на свободно распространяемый список рассылки X-Force Alert, так и путем интерактивного поиска в базе данных на Web-сервере компании ISS.

Заключение

Не будь уязвимостей в компонентах информационных систем, нельзя было бы реализовать многие атаки и, следовательно, традиционные системы защиты вполне эффективно справлялись бы с возможными атаками. Однако программы пишутся людьми, которым свойственно делать ошибки. Вследствие чего и появляются уязвимости, которые используются злоумышленниками для реализации атак. Однако это только полбеды. Если бы все атаки строились по модели "один к одному", то с некоторой натяжкой, но межсетевые экраны и другие защитные системы смогли бы противостоять и им. Но появились скоординированные атаки, против которых традиционные средства уже не так

эффективны. И тут на сцене и появляются новые технологии - технологии обнаружения атак. Приведенная систематизация данные об атаках и этапах их реализации дает необходимый базис для понимания технологий обнаружения атак.

Средства обнаружения компьютерных атак

Технология обнаружения атак должна решать следующие задачи:

- Распознавание известных атак и предупреждение о них соответствующего персонала.
- "Понимание" зачастую непонятных источников информации об атаках.
- Освобождение или снижение нагрузки на персонал, отвечающий за безопасность, от текущих рутинных операций по контролю за пользователями, системами и сетями, являющимися компонентами корпоративной сети.
- Возможность управления средствами защиты не-экспертами в области безопасности.
- Контроль всех действий субъектов корпоративной сети (пользователей, программ, процессов и т.д.).

Очень часто ***системы обнаружения атак*** могут выполнять функции, существенно расширяющие спектр их применения. Например,

- Контроль эффективности межсетевых экранов. Например, установка системы обнаружения атак после ***межсетевого экрана*** (внутри корпоративной сети) позволяет обнаружить атаки, пропускаемые МСЭ и, тем самым, определить недостающие правила на межсетевом экране.
- Контроль узлов сети с неустановленными обновлениями или узлов с устаревшим программным обеспечением.
- Блокирование и контроль доступа к определенным узлам Internet. Хотя системам обнаружения атак далеко до межсетевых экранов и систем контроля доступа к различным URL, например, WEBsweeper, они могут выполнять частичный контроль и блокирование доступа некоторых пользователей корпоративной сети к отдельным ресурсам Internet, например, к Web-серверам порнографического содержания. Это бывает необходимо тогда, когда в организации нет денег на приобретение и межсетевого экрана и системы обнаружение атак, и функции МСЭ разносятся между системой обнаружения атак, маршрутизатором и proxy-сервером. Кроме того, системы обнаружения атак могут контролировать доступ сотрудников к серверам на основе ключевых слов. Например, sex, job, crack и т.д.
- Контроль электронной почты. Системы обнаружения атак могут использоваться для контроля неблагонадежных сотрудников, использующих электронную почту для выполнения задач, не входящих в их функциональные обязанности, например, рассылка резюме. Некоторые системы могут обнаруживать вирусы в почтовых сообщениях и, хотя до настоящих антивирусных систем им далеко, они все же выполняют эту задачу достаточно эффективно.

Лучшее использование времени и опыта специалистов в области информационной безопасности заключается в обнаружении и устранении причин реализации атак, скорее чем, в обнаружении самих атак. Устранив причины возникновения атак, т.е. обнаружив и устранив уязвимости, администратор тем самым устраняет и сам факт потенциальной реализации атак. Иначе атака будет повторяться раз за разом, постоянно требуя усилий и внимания администратора.

Классификация систем обнаружения атак

Существует большое число различных классификаций систем обнаружения атак, однако самой распространенной является классификация по принципу реализации:

1. ***host-based***, то есть обнаруживающие атаки, направленные на конкретный узел сети,
2. ***network-based***, то есть обнаруживающие атаки, направленные на всю сеть или сегмент сети.

Системы обнаружения атак, контролирующие отдельный компьютер, как правило, собирают и анализируют информацию из журналов регистрации операционной системы и различных приложений (Web-сервер, СУБД и т.д.). По такому принципу функционирует RealSecure OS Sensor. Однако в последнее время стали получать распространение системы, тесно интегрированные с ядром ОС, тем самым, предоставляя более эффективный способ обнаружения нарушений политики безопасности. Причем такая интеграция может быть реализовано двояко. Во-первых, могут контролироваться все системные вызовы ОС (так работает Entercept) или весь входящий/исходящий сетевой трафик (так работает RealSecure Server Sensor). В последнем случае система обнаружения атак захватывает весь сетевой трафик напрямую с сетевой карты, минуя операционную систему, что позволяет уменьшить зависимость от нее и тем самым повысить защищенность системы обнаружения атак.

Системы обнаружения атак уровня сети собирают информацию из самой сети, то есть из сетевого трафика. Выполняться эти системы могут на обычных компьютерах (например, RealSecure Network Sensor), на специализированных компьютерах (например, RealSecure for Nokia или Cisco Secure IDS 4210 и 4230) или интегрированы в маршрутизаторы или коммутаторы (например, CiscoSecure IOS Integrated Software или Cisco Catalyst 6000 IDS Module). В первых двух случаях анализируемая информация собирается посредством захвата и анализа пакетов, используя сетевые интерфейсы в беспорядочном (promiscuous) режиме. В последнем случае захват трафика осуществляется с шины сетевого оборудования.

Обнаружение атак требует выполнения одного из двух условий - или понимания ожидаемого поведения контролируемого объекта системы или знания всех возможных атак и их модификаций. В первом случае используется технология обнаружения аномального поведения, а во втором случае - технология обнаружения злоумышленного поведения или злоупотреблений. Вторая технология заключается в описании атаки в виде шаблона или сигнатуры и поиска данного шаблона в контролируемом пространстве (например, сетевом трафике или журнале регистрации). Эта технология очень похожа на обнаружение вирусов (антивирусные системы являются ярким примером системы обнаружения атак), т.е. система может обнаружить все известные атаки, но она мало приспособлена для обнаружения новых, еще неизвестных, атак. Подход, реализованный в таких системах, очень прост и именно на нем основаны практически все предлагаемые сегодня на рынке системы обнаружения атак.

Практически все системы обнаружения атак основаны на сигнатурном подходе.

Достоинства систем обнаружения атак

Можно долго перечислять различные достоинства систем обнаружения атак, функционирующих на уровне узла и сети. Однако я остановлюсь только на нескольких из них.

Коммутация позволяет управлять крупномасштабными сетями, как несколькими небольшими сетевыми сегментами. В результате бывает трудно определить наилучшее место для установки системы, обнаруживающей атаки в сетевом трафике. Иногда могут помочь специальные порты (span ports) на коммутаторах, но не всегда. Обнаружение атак на уровне конкретного узла обеспечивает более эффективную работу в коммутируемых сетях, так как позволяет разместить системы обнаружения только на тех узлах, на которых это необходимо.

Системы сетевого уровня не требуют, чтобы на каждом хосте устанавливалось программное обеспечение системы обнаружения атак. Поскольку для контроля всей сети число мест, в которых установлены IDS невелико, то стоимость их эксплуатации в сети предприятия ниже, чем стоимость эксплуатации систем обнаружения атак на системном уровне. Кроме того, для контроля сетевого сегмента, необходим только один сенсор, независимо от числа узлов в данном сегменте.

Сетевой пакет, будучи ушедшим с компьютера злоумышленника, уже не может быть возвращен назад. Системы, функционирующие на сетевом уровне, используют "живой" трафик при обнаружении атак в реальном масштабе времени. Таким образом, злоумышленник не может удалить следы своей несанкционированной деятельности. Анализируемые данные включают не только информацию о методе атаки, но и информацию, которая может помочь при идентификации злоумышленника и доказательстве в суде. Поскольку многие хакеры хорошо знакомы с механизмами системной регистрации, они знают, как манипулировать этими файлами для скрытия следов своей деятельности, снижая эффективность систем системного уровня, которым требуется эта информация для того, чтобы обнаружить атаку.

Системы, функционирующие на уровне сети, обнаруживают подозрительные события и атаки по мере того, как они происходят, и поэтому обеспечивают гораздо более быстрое уведомление и реагирование, чем системы, анализирующие журналы регистрации. Например, хакер, инициирующий сетевую атаку типа "отказ в обслуживании" на основе протокола TCP, может быть остановлен системой обнаружения атак сетевого уровня, посылающей TCP-пакет с установленным флагом Reset в заголовке для завершения соединения с атакующим узлом, прежде чем атака вызовет разрушения или повреждения атакуемого узла. Системы анализа журналов регистрации не распознают атаки до момента соответствующей записи в журнал и предпринимают ответные действия уже после того, как была сделана запись. К этому моменту наиболее важные системы или ресурсы уже могут быть скомпрометированы или нарушена работоспособность системы, запускающей систему обнаружения атак на уровне узла. Уведомление в реальном масштабе времени позволяет быстро среагировать в соответствии с предварительно определенными параметрами. Диапазон этих реакций изменяется от разрешения проникновения в режиме наблюдения для того, чтобы собрать информацию об атаке и атакующем, до немедленного завершения атаки.

И, наконец, системы обнаружения атак, функционирующие на сетевом уровне, не зависят от операционных систем, установленных в корпоративной сети, так как они оперируют сетевым трафиком, которым обмениваются все узлы в корпоративной сети. Системе обнаружения атак все равно, какая ОС сгенерировала тот или иной пакет, если он в соответствие со стандартами, поддерживаемыми системой обнаружения. Например, в сети могут работать ОС Windows 98, Windows NT, Windows 2000 и XP, Netware, Linux, MacOS, Solaris и т.д., но если они общаются между собой по протоколу IP, то любая из систем обнаружения атак, поддерживающая этот протокол, сможет обнаруживать атаки, направленные на эти ОС.

Совместное применение систем обнаружения атак на уровне сети и уровне узла повысит защищенность вашей сети.

Сетевые системы обнаружения атак и межсетевые экраны

Наиболее часто сетевые системы обнаружения атак пытаются заменить межсетевыми экранами, уповая на то, что последние обеспечивают очень высокий уровень защищенности. Однако не стоит забывать, что межсетевые экраны - это просто системы, основанные на правилах, которые разрешают или запрещают прохождение трафика через них. Даже межсетевые экраны, построенные по технологии "", не позволяют с уверенностью сказать, присутствует ли атака в контролируемом ими трафике или нет. Они могут сказать, соответствует ли трафик правилу или нет. Например, МСЭ сконфигурирован так, чтобы блокировать все соединения кроме TCP-соединений на 80 порту (то есть HTTP-трафик). Таким образом, любой трафик через 80-ый порт законен с точки зрения МСЭ. С другой стороны, система обнаружения атак также контролирует трафик, но ищет в нем признаки атаки. Ее мало заботит, для какого порта предназначен трафик. По умолчанию весь трафик для системы обнаружения атак подозрителен. То есть, несмотря на то, что система обнаружения атак работает с тем же источником данных, что и МСЭ, то есть с сетевым трафиком, они выполняют дополняющие друг друга функции. Например, HTTP-запрос "GET /../../../etc/passwd HTTP/1.0". Практически любой МСЭ разрешает прохождение данного запроса через себя. Однако система обнаружения атак легко обнаружит эту атаку и блокирует ее.

Можно провести следующую аналогию. Межсетевой экран - это обычный турникет, устанавливаемый на главном входе в вашу сеть. Но помимо главных дверей существуют и другие двери, а также окна. Маскируясь под реального сотрудника или войдя в доверие к охраннику на турникете, злоумышленник может пронести сквозь турникет взрывное устройство или пистолет. Мало того. Злоумышленник может залезть к вам через окно. Именно поэтому и нужны системы обнаружения атак, которые усиливают защиту, обеспечиваемую межсетевыми экранами, которые являются пусть и необходимым, но явно недостаточным элементом сетевой безопасности.

***Межсетевой экран* - не панацея!**

Варианты реакций на обнаруженную атаку

Мало обнаружить атаку, - необходимо на нее соответствующим образом отреагировать. Именно варианты реагирования во многом определяют эффективность системы обнаружения атак. На сегодняшний день предлагаются следующие варианты реагирования:

- Уведомление на консоль (включая резервную) системы обнаружения атак или на консоль интегрированной системы (например, межсетевого экрана).
- Звуковое оповещение об атаке.
- Генерация управляющих последовательностей SNMP для систем сетевого управления.
- Генерация сообщения об атаке по электронной почте.
- Дополнительные уведомления на пейджер или факс. Очень интересная, хотя и редко применяемая возможность. Оповещение об обнаружении несанкционированной деятельности посылается не администратору, а злоумышленнику. По мнению сторонников

данного варианта реагирования, нарушитель, узнав, что его обнаружили, вынужден прекратить свои действия.

- Обязательная регистрация обнаруживаемых событий. В качестве журнала регистрации могут выступать:
    - текстовый файл,
    - системный журнал (например, в системе Cisco Secure Integrated Software),
    - текстовый файл специального формата (например, в системе Snort),
    - локальная база данных MS Access,
    - SQL-база данных (например, в системе RealSecure).

  Надо только учитывать, что объемы регистрируемой информации требуют, как правило, SQL-базу - MS SQL или Oracle.

- Трассировка событий (event trace), т.е. запись их в той последовательности и с той скоростью, с которыми их реализовывал злоумышленник. Затем администратор в любое заданное время может прокрутить (replay или playback) необходимую последовательность событий с заданной скоростью (в реальном режиме времени, с ускорением или замедлением), чтобы проанализировать деятельность злоумышленника. Это позволит понять его квалификацию, используемые средства атаки и т.д.
- Прерывание действий атакующего, т.е. завершение соединения. Это можно сделать, как:
    - перехват соединения (session hijacking) и посылка пакета с установленным флагом RST обоим участникам сетевого соединения от имени каждого из них (в системе обнаружения атак, функционирующей на уровне сети);
    - блокировка учетной записи пользователя, осуществляющего атаку (в системе обнаружения атак на уровне узла). Такая блокировка может быть осуществлена либо на заданный промежуток времени, либо до тех пор, пока учетная запись не будет разблокирована администратором. В зависимости от привилегий, с которыми запущена система обнаружения атак, блокировка может действовать как в пределах самого компьютера, на который направлена атака, так и в пределах всего домена сети.
- Реконфигурация сетевого оборудования или межсетевых экранов. В случае обнаружения атаки на маршрутизатор или межсетевой экран посылается команда на изменение списка контроля доступа. Впоследствии все попытки соединения с атакующего узла будут отвергаться. Как и блокировка учетной записи злоумышленника, изменение списка контроля доступа может быть осуществлено или на заданный интервал времени или до того момента, как изменение будет отменено администратором реконфигурируемого сетевого оборудования.
- Блокирование сетевого трафика так, как это реализовано в межсетевых экранах. Этот вариант позволяет ограничить трафик, а также адресатов, которые могут получить доступ к ресурсам защищаемого компьютера, позволяя выполнять функции доступные в персональных межсетевых экранах.

Безопасность электронной коммерции

Широкое внедрение Интернета не могло не отразиться на развитии электронного бизнеса.

Одним из видов электронного бизнеса считается электронная коммерция. В соответствии с документами ООН, бизнес признается электронным, если хотя бы две его составляющие из четырех (производство товара или услуги, маркетинг, доставка и расчеты) осуществляются с помощью Интернета. Поэтому в такой интерпретации обычно полагают, что покупка относится к электронной коммерции, если, как минимум,

маркетинг (организация спроса) и расчеты производятся средствами Интернета. Более узкая трактовка понятия "электронная коммерция" характеризует системы безналичных расчетов на основе пластиковых карт.

Ключевым вопросом для внедрения электронной коммерции является безопасность.

Высокий уровень мошенничества в Интернете является сдерживающим фактором развития электронной коммерции. Покупатели, торговля и банки боятся пользоваться этой технологией из-за опасности понести финансовые потери. Люди главным образом используют Интернет в качестве информационного канала для получения интересующей их информации. Лишь немногим более 2% всех поисков по каталогам и БД в Интернете заканчиваются покупками.

Приведем классификацию возможных типов мошенничества в электронной коммерции:

- транзакции (операции безналичных расчетов), выполненные мошенниками с использованием правильных реквизитов карточки (номер карточки, срок ее действия и т.п.);
- получение данных о клиенте через взлом БД торговых предприятий или путем перехвата сообщений покупателя, содержащих его персональные данные;
- магазины-бабочки, возникающие, как правило, на непродолжительное время, для того, чтобы исчезнуть после получения от покупателей средств за несуществующие услуги или товары;
- увеличение стоимости товара по отношению к предлагавшейся покупателю цене или повтор списаний со счета клиента;
- магазины или торговые агенты, презназначенные для сбора информации о реквизитах карт и других персональных данных покупателя.

Протокол SSL

Протокол ***SSL*** (Secure Socket Layer) был разработан американской компанией Netscape Communications. SSL обеспечивает защиту данных между сервисными протоколами (такими как HTTP, NNTP, FTP и т.д.) и транспортными протоколами (TCP/IP) с помощью современной криптографии в соединениях "точка-точка". Ранее можно было без особых технических ухищрений просматривать данные, которыми обмениваются между собой клиенты и серверы. Был даже придуман специальный термин для этого - "sniffer".

Протокол SSL предназначен для решения традиционных задач обеспечения защиты информационного взаимодействия:

- пользователь и сервер должны быть взаимно уверены, что они обмениваются информацией не с подставными абонентами, а именно с теми, которые нужны, не ограничиваясь паролевой защитой;
- после установления соединения между сервером и клиентом весь информационный поток между ними должен быть защищен от несанкционированного доступа;
- и наконец, при обмене информацией стороны должны быть уверены в отсутствии случайных или умышленных искажений при ее передаче.

Протокол SSL позволяет серверу и клиенту перед началом информационного взаимодействия аутентифицировать друг друга, согласовать алгоритм шифрования и сформировать общие криптографические ключи. С этой целью в протоколе используются двухключевые (ассиметричные) криптосистемы, в частности, RSA.

Конфиденциальность информации, передаваемой по установленному защищенному соединению, обеспечивается путем шифрования потока данных на сформированном общем ключе с использованием симметричных криптографических алгоритмов (например, RC4_128, RC4_40, RC2_128, RC2_40, DES40 и др.). Контроль целостности передаваемых блоков данных производится за счет использования так называемых кодов аутентификации сообщений (Message Autentification Code, или MAC), вычисляемых с помощью хэш-функций (например MD5).

Протокол SSL включает два этапа взаимодействия сторон защищаемого соединения:

- установление SSL-сессии;
- защита потока данных.

На этапе установления SSL-сессии осуществляется аутентификация сервера и (опционально) клиента, стороны договариваются об используемых криптографических алгоритмах и формируют общий "секрет", на основе которого создаются общие сеансовые ключи для последующей защиты соединения. Этот этап называют также "процедурой рукопожатия".

На втором этапе (защита потока данных) информационные сообщения прикладного уровня нарезаются на блоки, для каждого блока вычисляется код аутентификации сообщений, затем данные шифруются и отправляются приемной стороне. Приемная сторона производит обратные действия: расшифрование, проверку кода аутентификации сообщения, сборку сообщений, передачу на прикладной уровень.

Наиболее распространенным пакетом программ для поддержки SSL является SSLeay. Он содержит исходный код на C, который может быть встроен в такие приложения, как Telnet и FTP.

В SSL используется криптография с открытым (публичным) ключом, также известная как асимметричная криптография. Она использует два ключа: один - для шифрования, другой - для расшифровывания сообщения. Два ключа математически связаны таким образом, что данные, зашифрованные с использованием одного ключа, могут быть расшифрованы только с использованием другого, парного первому. Каждый пользователь имеет два ключа - открытый и секретный (приватный). Пользователь делает доступным открытый ключ любому корреспонденту сети. Пользователь и любой корреспондент, имеющий открытый ключ, могут быть уверены, что данные, зашифрованные с помощью открытого ключа, могут быть расшифрованы только с использованием секретного ключа.

Если два пользователя хотят быть уверенными, что информацию, которой они обмениваются, не получит третий, то каждый из них, должен передать одну компоненту ключевой пары (а именно открытый ключ), другому и хранит другую компоненту (секретный ключ). Сообщения шифруются с помощью открытого, расшифровываются только с использованием секретного ключа. Именно так сообщения могут быть переданы по открытой сети без опасения, что кто-либо сможет прочитать их.

Целостность и аутентификация сообщения обеспечиваются использованием электронной цифровой подписи.

Теперь встает вопрос о том, каким образом распространять свои публичные ключи. Для этого (и не только) была придумана специальная форма - сертификат. Сертификат состоит из следующих частей:

- имя человека/организации, выпускающей сертификат;
- субъект сертификата (для кого был выпущен данный сертификат);
- публичный ключ субъекта;
- некоторые временные параметры (срок действия сертификата и т.п.).

Сертификат "подписывается" приватным ключом человека (или организации), который выпускает сертификаты. Организации, которые производят подобные операции называются Certificate authority (CA). Если в стандартном Web-браузере, который поддерживает SSL, зайти в раздел security, то там можно увидеть список известных организаций, которые "подписывают" сертификаты. Технически создать свою собственную CA достаточно просто, но также необходимо уладить юридическую сторону дела, и с этим могут возникнуть серьезные проблемы.

SSL на сегодня является наиболее распространенным протоколом, используемым при построении систем электронной коммерции. С его помощью осуществляется 99% всех транзакций. Широкое распространение SSL объясняется в первую очередь тем, что он является составной частью всех браузеров и Web-серверов. Другое достоинство SSL - простота протокола и высокая скорость реализации транзакции.

В то же время, SSL обладает рядом существенных недостатков:

- покупатель не аутентифицируется;
- продавец аутентифицируется только по URL;
- цифровая подпись используется только при аутентификации в начале установления SSL-сессии. Для доказательства проведения транзакции при возникновении конфликтных ситуаций требуется либо хранить весь диалог покупателя и продавца, что дорого с точки зрения ресурсов памяти и на практике не используется, либо хранить бумажные копии, подтверждающие получение товара покупателем;
- не обеспечивается конфиденциальность данных о реквизитах карты для продавца.

Протокол SET

Другой протокол безопасных транзакций в Интернете - ***SET*** (Security Electronics Transaction). SET основан на использовании цифровых сертификатов по стандарту X.509.

Протокол выполнения защищенных транзакций SET является стандартом, разработанным компаниями MasterCard и VISA при значительном участии IBM, GlobeSet и других партнеров. Он позволяет покупателям приобретать товары через Интернет, используя самый защищенный на настоящее время механизм выполнения платежей. SET является открытым стандартным многосторонним протоколом для проведения безопасных платежей с использованием пластиковых карточек в Интернет. SET обеспечивает кросс-аутентификацию счета держателя карточки, продавца и банка

продавца для проверки готовности оплаты товара, целостность и секретность сообщения, шифрование ценных и уязвимых данных. Поэтому SET можно назвать стандартной технологией или системой протоколов выполнения безопасных платежей с использованием пластиковых карточек через Интернет.

SET позволяет потребителям и продавцам подтвердить подлинность всех участников сделки, происходящей в Интернет, с помощью криптографии, применяя, в том числе, и цифровые сертификаты.

Объем потенциальных продаж в области электронной коммерции ограничивается достижением необходимого уровня безопасности информации, который обеспечивают вместе покупатели, продавцы и финансовые институты, обеспокоенные вопросами обеспечения безопасности платежей через Интернет. Как упоминалось ранее, базовыми задачами защиты информации являются обеспечение ее доступности, конфиденциальности, целостности и юридической значимости. SET, в отличии от других протоколов, позволяет решать указанные задачи защиты информации.

В результате того, что многие компании занимаются разработкой собственного программного обеспечения для электронной коммерции, возникает еще одна проблема. В случае использования этого ПО все участники операции должны иметь одни и те же приложения, что практически неосуществимо. Следовательно, необходим способ обеспечения механизма взаимодействия между приложениями различных разработчиков.

В связи с перечисленными выше проблемами компании VISA и MasterCard вместе с другими компаниями, занимающимися техническими вопросами (например IBM, которая является ключевым разработчиком в развитии протокола SET), определили спецификацию и набор протоколов стандарта SET. Эта открытая спецификация очень быстро стала де-факто стандартом для электронной коммерции. В этой спецификации шифрование информации обеспечивает ее конфиденциальность. Цифровая подпись и сертификаты обеспечивают идентификацию и аутентификацию (проверку подлинности) участников транзакций. Цифровая подпись также используется для обеспечения целостности данных. Открытый набор протоколов используется для обеспечения взаимодействия между реализациями разных производителей.

SET обеспечивает следующие специальные требования защиты операций электронной коммерции:

- секретность данных оплаты и конфиденциальность информации заказа, переданной вместе с данными об оплате;
- сохранение целостности данных платежей; целостность обеспечивается при помощи цифровой подписи;
- специальную криптографию с открытым ключом для проведения аутентификации;
- аутентификацию держателя по кредитной карточке, которая обеспечивается применением цифровой подписи и сертификатов держателя карточек;
- аутентификацию продавца и его возможности принимать платежи по пластиковым карточкам с применением цифровой подписи и сертификатов продавца;
- подтверждение того, что банк продавца является действующей организацией, которая может принимать платежи по пластиковым карточкам через связь с процессинговой системой; это подтверждение обеспечивается с помощью цифровой подписи и сертификатов банка продавца;

- готовность оплаты транзакций в результате аутентификации сертификата с открытым ключом для всех сторон;
- безопасность передачи данных посредством преимущественного использования криптографии.

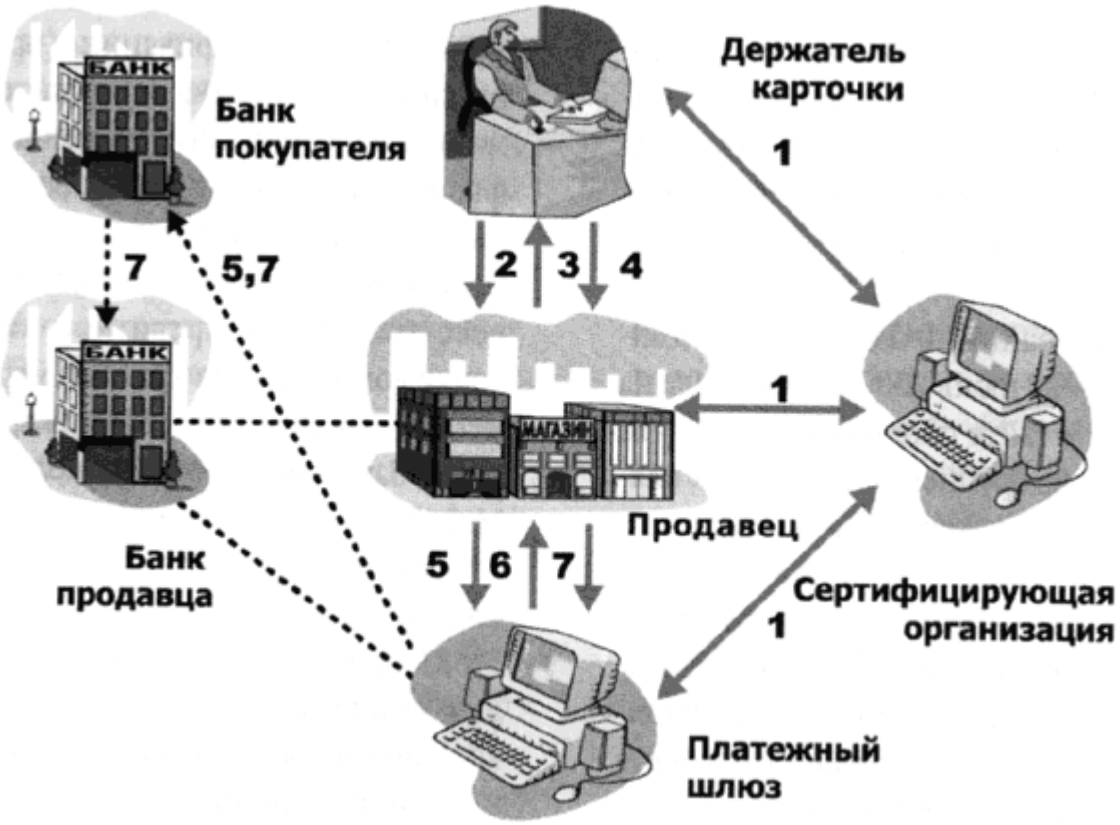
Основное преимущество SET перед многими существующими системами обеспечения информационной безопасности заключается в использовании цифровых сертификатов (стандарт X.509, версия 3), которые ассоциируют держателя карточки, продавца и банк продавца с рядом банковских учреждений платежных систем VISA и MasterCard.

SET позволяет сохранить существующие отношения между банком, держателями карточек и продавцами, и интегрируется с существующими системами, опираясь на следующие качества:

- открытый, полностью документированный стандарт для финансовой индустрии;
- основан на международных стандартах платежных систем;
- опирается на существующие в финансовой отрасли технологии и правовые механизмы.

Кстати, совместный проект, реализованный компаниями IBM, Chase Manhattan Bank USA N.A., First Data Corporation, GlobeSet, MasterCard и Wal-Mart позволяет владельцам карточек Wal-Mart MasterCard, выпущенных банком Chase, приобретать товары на сайте Wal-Mart Online, который является одним из крупнейших узлов электронной коммерции США.

Рассмотрим более детально процесс взаимодействия участников платежной операции в соответствии со спецификацией SET, представленный на рисунке с сайта компании IBM:



На рисунке:

- **Держатель карточки** - покупатель делающий заказ.
- **Банк покупателя** - финансовая структура, которая выпустила кредитную карточку для покупателя.
- **Продавец** - электронный магазин, предлагающий товары и услуги.
- **Банк продавца** - финансовая структура, занимающаяся обслуживанием операций продавца.
- **Платежный шлюз** - система, контролируемая обычно банком продавца, которая обрабатывает запросы от продавца и взаимодействует с банком покупателя.
- **Сертифицирующая организация** - доверительная структура, выдающая и проверяющая сертификаты.

Взаимоотношения участников операции показаны на рисунке непрерывными линиями (взаимодействия описанные стандартом или протоколом SET) и пунктирными линиями (некоторые возможные операции).

Динамика взаимоотношений и информационных потоков в соответствии со спецификацией стандарта SET включает следующие действия :

1. Участники запрашивают и получают сертификаты от сертифицирующей организации.
2. Владелец пластиковой карточки просматривает электронный каталог, выбирает товары и посылает заказ продавцу.
3. Продавец предъявляет свой сертификат владельцу карточки в качестве удостоверения.
4. Владелец карточки предъявляет свой сертификат продавцу.

5. Продавец запрашивает у платежного шлюза выполнение операции проверки. Шлюз сверяет предоставленную информацию с информацией банка, выпустившего электронную карточку.
6. После проверки платежный шлюз возвращает результаты продавцу.
7. Некоторое время спустя, продавец требует у платежного шлюза выполнить одну или более финансовых операций. Шлюз посылает запрос на перевод определенной суммы из банка покупателя в банк продавца.

Представленная схема взаимодействия подкрепляется в части информационной безопасности спецификацией Chip Electronic Commerce, созданной для использования смарт-карточек стандарта EMV в Интернете (www.emvco.com). Ее разработали Europay, MasterCard и VISA. Сочетание стандарта на микропроцессор EMV и протокола SET дает беспрецедентный уровень безопасности на всех этапах транзакции.

Компания "Росбизнесконсалтинг" 20 июня 2000 г. поместила на своем сайте сообщение о том, что одна из крупнейших мировых платежных систем VISA обнародовала 19 июня 2000 г. свои инициативы в области безопасности электронной коммерции. По словам представителей системы, эти шаги призваны сделать покупки в Интернете безопаснее для покупателей и продавцов. VISA полагает, что внедрение новых инициатив позволит сократить количество споров по транзакциям в Интернете на 50%. Инициатива состоит из двух основных частей. Первая часть - это Программа аутентификации платежей (Payment Authentication Program), которая разработана для снижения риска неавторизованного использования счета держателя карточки и улучшения сервиса для покупателей и продавцов в Интернете. Вторая - это Глобальная программа защиты данных (Global Data Security Program), цель которой - создать стандарты безопасности для компаний электронной коммерции по защите данных о карточках и их держателях.

Сравнительные характеристики протоколов SSL и SET

Платежные системы являются наиболее критичной частью электронной коммерции и будущее их присутствия в сети во многом зависит от возможностей обеспечения информационной безопасности и других сервисных функций в Интернете. SSL и SET - это два широко известных протокола передачи данных, каждый из которых используется в платежных системах Интернета. Мы попытаемся сравнить SSL и SET и оценить их некоторые важнейшие характеристики.

Итак, рассмотрим важнейшую функцию аутентификации (проверки подлинности) в виртуальном мире, где отсутствуют привычные физические контакты. SSL обеспечивает только двухточечное взаимодействие. Мы помним, что, в процесс транзакции кредитной карточки вовлечены, по крайней мере, четыре стороны: потребитель, продавец, банк-эмитент и банк-получатель. SET требует аутентификации от всех участвующих в транзакции сторон.

SET предотвращает доступ продавца к информации о пластиковой карточке и доступ банка-эмитента к частной информации заказчика, касающейся его заказов. В SSL разрешается контролируемый доступ к серверам, директориям, файлам и другой информации. Оба протокола используют современную криптографию и системы цифровых сертификатов, удостоверяющих цифровые подписи взаимодействующих сторон. SSL предназначен преимущественно для защиты коммуникаций в Интернете. SET

обеспечивает защиту транзакций электронной коммерции в целом, что обеспечивает юридическую значимость защищаемой ценной информации. При этом через SET транзакция происходит медленней, чем в SSL, и ее стоимость намного выше. Последняя характеристика весьма актуальна для сегодняшнего российского рынка, на котором пока не считают риски и эксплуатационные расходы.

Следует добавить, что, используя SSL, потребители подвергаются риску раскрытия реквизитов своих пластиковых карточек продавцу.

Внедрение и эксплуатация SET осуществляется много лет в нескольких десятках проектов во всем мире. Например, первая транзакция SET была проведена 30-го декабря 1996 в PBS (Датский банк) в совместном проекте IBM и MasterCard. Аналогичная работа проведена в 1997 г. в крупнейшем японском банке Fuji Bank, где пришлось адаптировать протокол к специфическому японскому законодательству. За прошедшее время подобные внедренческие проекты позволили отработать функции протокола и соответствующую документацию.

Кстати, IBM имеет полный набор продуктов, который охватывает все ключевые аспекты комплексного использования SET в целом и обеспечивает развитую инфраструктуру:

- IBM Net.commerce Suite для продавцов, организующих интернет-магазины;
- IBM Consumer Wallet для держателей карточек;
- IBM Payment Gateway - шлюз платежей для банков;
- IBM Net. Payment Registry - продукт для аутентификации и сертификации.

SET функционирует на разных вычислительных платформах таких компаний, как IBM, Hewlett Packard, Sun Microsystems и Microsoft.

В свою очередь SSL используется в основном в Web-приложениях и для защиты коммуникаций в Интернете. Существует также свободно распространяемая версия SSL, называемая SSLeay. Она содержит исходный код на C, который может быть встроен в такие приложения, как Telnet и FTP. Благодаря этим качествам SSL получил широкое распространение в корпоративных интранет-сетях и в системах с небольшим количеством пользователей.

Несмотря на технологическое совершенство протокола SET, его использование в мире весьма ограничено. Тому имеется множество причин, решающей среди которых является высокая стоимость внедрения системы электронной коммерции на базе протокола SET (стоимость SET-решения колеблется от $600 до 1500 тыс.).

Протокол SSL обеспечивает лишь конфинденциальность данных транзакции при их передачи через сеть общего пользования, но при этом является существенно более дешевым для внедрения. В результате подавляющее число современных систем электронной коммерции используют протокол SSL.

Эксперты и разработчики протокола SET ошиблись, предсказывая быстрое и повсеместное внедрение этого стандарта. Более того, ведутся настойчивые разговоры о том, что протокол SET уже является вчерашним днем и его шансы на выживание ничтожны.

Такие разговоры начались еще летом 2000г., когда VISA International сделала заявление, в соответствии с которым протокол 3D SET (разновидность SET) становится стандартом для стран Евросоюза, Латинской Америки и некоторых других европейских стран, включая Россию. В то же время на самом крупном американском рынке в качестве стандарта был провозглашен протокол 3D SSL (другое название протокола - 3D Payer).

Глава российского представительства Visa Int. Лу Наумовский согласен с тем, что SET не нашел спроса:

"Это очень хорошая технология. Но, судя по реакции банков, не только российских, но и зарубежных, - она дороговата. Банку-эмитенту, использующему протокол SET для отслеживания операций по картам, приходится самому держать базу данных банков-эквайреров и торговых точек. Мы пытались найти более дешевую альтернативу этому протоколу".

В мае 2001 г. были опубликованы спецификации на стандарт 3D Secure, претендующий на роль глобального стандарта аутентификации в платежной системе Visa. По решению Европейского союза в июле 2002 г. все интернет-магазины получили идентификацию на уровне этого протокола. Следовательно, банк-эквайрер таких интернет-магазинов должен иметь возможность предоставить им этот протокол. В случае отсутствия 3D Secure всю ответственность при спорных трансакциях несет он сам. Если он использует 3D Secure, а банк-эмитент нет, то ответственность берет на себя последний.

Принцип работы 3D Secure в том, что есть три различных домена - банка-эмитента, интернет-магазина и Visa, через домен которой идет сообщение между покупателем, продавцом и банками. Очень важно, что все сообщения идут через интернет. При этом Visa обеспечивает конфиденциальность информации. После того как покупатель нажимает на интернет-странице на лозунг Verified by Visa и вводит свой пароль, эта информация идет к банку-эмитенту и происходит идентификация. Банк-эмитент через домен Visa отправляет запрос в интернет-магазин, после чего этот магазин идентифицируется своим банком-эквайрером. Таким образом, данные держателя карты известны только банку-эмитенту. В то же время владелец карты уверен в том, что данный магазин имеет Verified by Visa, то есть сертифицирован Visa через банк-эквайрер. В том случае, если банк-эмитент не получит от домена Visa подтверждения, что магазин имеет Verified by Visa, транзакция не произойдет.

Конечно, владелец карты может сделать покупки и в других, не имеющих статуса Verified by Visa, интернет-магазинах. Тогда ответственность по спорным сделкам несет банк-эмитент, и он должен будет предупреждать своих клиентов об этом.

Литература:

Описание протокола HTTPS (SSL)

Безопасность электронных платежных систем

Современную практику банковских операций, торговых сделок и взаимных платежей невозможно представить без расчетов с применением пластиковых карт.

Система безналичных расчетов с помощью пластиковых карт называется ***электронной платежной системой***.

Для обеспечения нормальной работы электронная платежная система должна быть надежно защищена.

С точки зрения информационной безопасности в системах электронных платежей существуют следующие уязвимые места:

- пересылка платежных и других сообщений между банками, между банком и банкоматом, между банком и клиентом;
- обработка информации внутри огранизаций отправителя и получателя сообщений;
- доступ клиентов к средствам, аккумилированным на счетах.

Пересылка платежных и других сообщений связана с такими особенностями:

- внутренние системы организаций отправителя и получателя должны обеспечивать необходимую защиту при обработке электронных документов (защита оконечных систем);
- взаимодействие отправителя и получателя электронного документа осуществляется опосредовано - через канал связи.

Эти особенности порождают следующие проблемы:

- взаимное опознание абонентов (проблема установления взаимной подлинности при установлении соединения);
- защита электронных документов, передаваемых по каналам связи (проблема обеспечения конфиденциальности и целостности документов);
- защита процесса обмена электронными документами (проблема доказательства отправления и доставки документа);
- обеспечение исполнения документа (проблема взаимного недоверия между отправителем и получателем из-за их принадлежности к разным оррганизациям и взаимной независимости).

Для обеспечения функций защиты информации на отдельных узлах системы электронных платежей должны быть реализованы следующие механизмы защиты:

- управление доступом на оконечных системах;
- контроль целостности сообщения;
- обеспечение конфиденциальности сообщения;
- взаимная аутентификация абонентов;
- невозможность отказа от авторства сообщения;
- гарантии доставки сообщения;

- невозможность отказа от принятия мер по сообщения;
- регистрация последовательности сообщений;
- контроль целостности последовательности сообщений.

Итак, в качестве платежного средства в электронной платежной системе используются электронные пластиковые карты.

Электронная пластиковая карта - это носитель информации, который идентифицирует владельца и хранит определенные учетные данные.

Различают кредитные и дебетовые карты.

Кредитные карты являются наиболее распространенным видом пластиковых карт. К ним относятся карты общенациональных систем США Visa и MasterCard, American Express и ряда других. Эти карты предъявляют для оплаты товаров и услуг. При оплате с помощью кредитной карты банк покупателя открывает ему кредит на сумму покупки, а затем через некоторое время (обычно 25 дней) присылает счет по почте. Покупатель должен вернуть оплаченный чек (счет) обратно в банк. Естественно, подобную схему банк может предложить только наиболее состоятельным и проверенным из своих клиентов, которые имеют хорошую кредитную историю перед банком или солидные вложения в банк в виде депозитов, ценностей или недвижимости.

Владелец ***дебетовой карты*** должен заранее внести на свой счет в банке-эмитенте определенную сумму. Размер этой суммы определяет лимит доступных средств. При осуществлении расчетов с использованием этой карты соответственно уменьшается и лимит. Для возобновления или увеличения лимита владелец должен вновь внести деньги на свой счет. Для страхования временного разрыва между моментом осуществления платежа и моментом получения банком соответствующей информации на счете клиента должен поддерживаться неснижаемый остаток.

Как кредитная, так и дебетовая карты могут быть не только персональными, но и корпоративными. *Корпоративные карты* предоставляются компанией своим сотрудникам для оплаты командировочных или других служебных расходов. Корпоративные карты компании связаны с каким-либо одним ее счетом. Эти карты могут иметь разделенный или неразделенный лимит. В первом случае каждому из держателей корпоративных карт устанавливается индивидуальный лимит. Второй вариант больше подходит небольшим компаниям и не предполагает разграничения лимита.

Пластиковая карта представляет собой пластину, изготовленную из специальной пластмассы, устойчивой к механическим и термическим воздействиям. По стандарту ***ISO 9001*** все пластиковые карты имеют размеры 85.6×53.9×0.76 мм.

Для идентификации владельца на пластиковую карту наносятся:

- логотип банка-эмитента;
- логотип платежной системы, обслуживающей эту карту;
- имя владельца карты;
- номер счета владельца карты;
- срок действия карты и т.п.

Кроме того, на карте может присутствовать фотография владельца и его подпись.

Алфавитно-цифровые данные (имя, номер счета и др.) могут быть *эмбоссированы*, т.е. нанесены рельефным шрифтом. Это дает возможность при ручной обработке принимаемых к оплате карт быстро перенести данные на чек с помощью специального устройства - импринтера, осуществляющего "прокатывание" карты.

По принципу действия различают *пассивные и активные пластиковые карты*. Пассивные пластиковые карты всего лишь хранят информацию. К ним относятся пластиковые карты с магнитной полосой.

Карты с магнитной полосой являются пока наиболее распространенными - в обращении находится свыше двух миллиардов карт подобного типа. Магнитная полоса располагается на обратной стороне карты и, в соответствии со стандартом ISO 7811, состоит из трех дорожек. Из них первые две предназначены для хранения идентификационных данных, а на третью дорожку можно записывать информацию (например, текущее значение лимита дебетовой карты). Однако из-за невысокой надежности многократно повторяемого процесса записи/считывания запись на магнитную полосу обычно не практикуется.

Карты с магнитной полосой относительно уязвимы для мошенничества. Для повышения защищенности своих карт системы Visa и MasterCard/Europay используют дополнительные графические средства защиты: голограммы и нестандартные шрифты для эмбоссирования. Эмбоссеры (устройства для тиснения рельефа на карте) выпускает ограниченный круг изготовителей. В ряде стран Запада законодательно запрещена свободная продажа эмбоссеров. Специальные символы, подтверждающие принадлежность карты к той или иной платежной системе, поставляются владельцу эмбоссера только с разрешения руководящего органа платежной системы.

Платежные системы с подобными картами требуют on-line авторизации в торговых точках и, как следствие, наличия разветвленных, высококачественных средств коммуникации (телефонных линий).

Отличительная особенность активной пластиковой карты - наличие встроенной в нее электронной микросхемы. Принцип пластиковой карты с электронной микросхемой запатентовал в 1974 г. француз Ролан Морено. Стандарт ***ISO 7816*** определяет основные требования к картам на интегральных микросхемах или чиповым картам.

Карты с микросхемой можно классифицировать по двум признакам.

Первый признак - принцип взаимодействия со считывающим устройством. Основные типы:

- карты с контактным считыванием;
- карты с бесконтактным (индукционным) считыванием.

Карта с контактным считыванием имеет на своей поверхности от 8 до 10 контактных пластин. Размещение контактных пластин, их количество и назначение выводов различны у разных производителей и естественно, что считыватели для карт данного типа различаются между собой.

Обмен данными между *картой с бесконтактным считыванием* и считывающим устройством производится индукционным способом. Очевидно, что такие карты надежнее и долговечнее.

Второй признак - функциональные возможности карты. Основные типы:

- карты-счетчики;
- карты с памятью;
- карты с микропроцессором.

Карты-счетчики применяются, как правила, в тех случаях, когда та или иная платежная операция требует уменьшения остатка на счете держателя карты на некоторую фиксированную сумму. Подобные карты используются в специализированных приложения с предоплатой (плата за использование телефона-автомата, отлата автостоянки и т.д.). Очевидно, что применение карт со счетчиком ограничено и не имеет большой перспективы.

Карты с памятью являются переходными между картами-счетчиками и картами с микропроцессором. Карта с памятью - это перезаписываемая карта-счетчик, в которой приняты меры, повышающие ее защищенность от атак злоумышленников. Простейшие карты с памятью имеют объем памяти от 32 байт до 16 Кбайт. Эта память может быть организована в виде:

- программируемого постоянного запоминающего устройства ППЗУ (EPROM), которое допускает однократную запись и многократное считывание;
- электрически стираемого программируемого постоянного запоминающего устройства ЭСППЗУ (EEPROM), которое допускает моногократную запись и многократное считывание.

Карты с памятью можно подразделить на два типа:

- с незащищенной (полнодоступной) памятью;
- с защищенной памятью.

В картах первого типа нет никаких ограничений на чтение и запись данных. Эти карты нельзя использовать в качестве платежных, так как их достаточно просто "взломать".

Карты второго типа имеют область идентификационных данных и одну или несколько прикладных областей. Идентификационная область допускает лишь однократную запись при персонализации и дальше доступна только для считывания. Доступ к прикладным областям регламентируется и осуществляется только при выполнении определенных операций, в частности при вводе секретного PIN-кода.

Уровень защиты карт с памятью выше, чем у магнитных карт. В качестве платежного средства карты с памятью используются для оплаты таксофонов общего пользования, проезда в транспорте, в локальных платежных системах (клубные карты). Карты с памятью применяются также в системах допуска в помещения и доступа к рресурсам компьютерных сетей (идентификационные карты).

Карты с микропроцессором называют также интеллектуальными картами или смарт-картами. Это по сути микрокомпьютеры, которые содержат все основные аппаратные компоненты:

- микропроцессор с тактовой частотой 5Мгц;
- оперативное ЗУ емкостью до 256 байт;

- постоянное ЗУ емкостью до 10 Кбайт;
- энергонезависимое ЗУ емкостью до 8 Кбайт.

Смарт-карта обеспечивает широкий набор функций:

- разграничение полномочий доступа к внутренним ресурсам;
- шифрование данных с применением различных алгоритмов;
- формирование электронной цифровой подписи;
- ведение ключевой системы;
- выполнение всех операций взаимодействия владельца карты, банка и торговца.

Некоторые смарт-карты обеспечивают режим "самоблокировки" при попытке несанкционированного доступа.

Все это делает смарт-карту высокозащищенным платежным инструментом, который может быть использован в финансовых приложениях, предъявляющих повышенные требования к защите информации. Именно поэтому смарт-карты являются наиболее перспективным видом пластиковых карт.

Важными этапами подготовки и применения пластиковой карты являются *персонализация и авторизация*.

Персонализация осуществляется при выдаче карты клиенту. При этом на карту заносятся данные, позволяющие идентифицировать карту и ее владельца, а также осуществить проверку платежеспособности карты при приеме ее к оплате или выдаче наличных денег. Первоначальным способом персонализации было эмбоссирование.

К персонализации относятся также кодирование магнитной полосы и программирование микросхемы.

Кодирование магнитной полосы производится, как правило, на том же оборудовании, что и эмбоссирование. При этос часть информации о карте, содержащая номер карты и период ее действия, одинаковая как на магнитной полосен, так и на рельефе. Однако бывают ситуации, когда после первичного кодирования требуется дополнительно занести информацию на магнитную полосу. В этом случае применяются специальные устройства с функцией "чтение-запись". Это возможно, в частности, когда PIN-код для пользования картой не формируется специальной программой, а выбирается клиентом по своему усмотрению.

Программирование микросхемы не требует особых технологических приемов, но зато имеет некоторые организационные особенности. Так операции по программированию отдельных областей микросхемы разнесены территориально и разграничены по правам различных сотрудников. Обычно эта процедура разбивается на три этапа:

- на первом рабочем месте выполняется активация карты (ввод ее в действие);
- на втором рабочем месте выполняются операции, связанные с обеспечением безопасности;
- на третьем рабочем месте производится собственно персонализация.

Такие меры повышают безопасность и исключают возможные злоупотребления.

Авторизация - это процесс утверждения продажи или выдачи наличных по карте. Для проведения авторизации точка обслуживания делает запрос платежной системе о подтверждении полномочий предъявителя карты и его финансовых возможностей. Технология авторизации зависит от типа карты, схемы платежной системы и технической оснащенности точки обслуживания.

Авторизация проводится либо "вручную", либо автоматически. В первом случае осуществляется голосовая авторизация, когда продавец или кассир передает запрос оператору по телефону. Во втором случае карта помещается в автоматизированный торговый ***POS-терминал*** (Point-Of-Sale - оплата в точке продажи), данные считываются с карты, кассир вводит сумму платежа, а владелец карты - PIN-код (Personal Identication Number - персональный идентификационный номер). После этого терминал осуществляет авторизацию, устанавливая связь с базой данных платежной системы (on-line режим), либо реализуя дополнительный обмен данными с самой картой (off-line режим). При выдаче наличных денег процесс имеет аналогичный характер, с той лишь особенностью, что деньги в автоматическом режиме выдаются банкоматом, который и проводит авторизацию.

Испытанным способом идентификации владельца пластиковой карты является использование секретного персонального идентификационного номера ***PIN***. Значение PIN должно быть известно только владельцу карты. С одной стороны, PIN должен быть достаточно длинным, чтобы вероятность угадывания с помощью полного перебора была приемлемо малой. С другой стороны, PIN должен быть достаточно коротким, чтобы владелец мог его запомнить. Обычно длина PIN колеблется от 4 до 8 десятичных цифр, но может достигать 12.

Значение PIN однозначно связано с соответствующими атрибутами пластиковой карты, поэтому PIN можно трактовать как подпись владельца карты.

Защита персонального идентификационного номера PIN для пластиковой карты является критичной для безопасности всей платежной системы. Пластиковые карты могут быть могут быть потеряны, украдены или подделаны. В таких случаях единственной контрмерой против несанкционированного доступа остается секретное значение PIN. Поэтому открытая форма PIN должна быть известна только законному владельцу карты. Она никогда не хранится и не передается в рамках системы электронных платежей.

Метод генерации значения PIN оказывает существенной влияние на безопасность электронной платежной системы. Вообще, персональные идентификационные номера могут формироваться либо банком, либо владельцами карт.

Если PIN назначается банком, то обычно используется один из двух вариантов.

При первом варианте PIN генерируется криптографически из номера счета владельца карточки. Шифрование проводится по алгоритму DES с использованием секретного ключа. Достоинство: значение PIN не нужно хранить внутри электронной платежной системы. Недостаток: при необходимости изменения PIN надо менять либо номер счета клиента, либо криптографический ключ. Но банки предпочитают, чтобы номер счета клиента оставался фиксированным. А с другой стороны, поскольку все PIN вычисляют, используя один ключ, изменение одного PIN при сохранении счета клиента влечет за собой изменение всех персональных идентификационных номеров.

При втором варианте банк выбирает PIN случайным образом, сохраняя это значение в виде криптограммы. Выбранные значения PIN передается владельцам карт по защищенному каналу.

Использование PIN, назначенного банком, неудобно клиентам даже при небольшой его длине. Такой PIN трудно удержать в памяти, и поэтому владелец карты может записать его куда-нибудь. **Главное - это не записывать PIN непосредственно на карту или другое видное место.** Иначе задача злоумышленников будет сильно облегчена.

Для большего удобства клиента используют значение PIN, выбираемое самим клиентом. Такой способ определения PIN позволяет клиенту:

- использовать один и тот же PIN для различных целей;
- задавать в PIN не только цифры, но и буквы (для удобства запоминания).

Выбранный клиентом PIN может быть передан в банк заказной почтой или отправлен через защищенный терминал банковского офиса, который немедленно его шифрует. Если банку необходимо использовать выбранный клиентом PIN, то поступают следующим образом. Каждую цифру выбранного клиентом PIN складывают по модулю 10 (без учета переносов) с соответствующей цифрой PIN, выводимого банком из счета клиента. Получаемое десятичное число называется "смещением". Это смещение запоминается на карте клиента. Поскольку выводимый PIN имеет случайный характер, то выбранный клиентом PIN невозможно определить по его смещению.

Главное требование безопасности состоит в том, что значение PIN должно запоминаться владельцем карты и никогда не должно храниться в любой читабельной форме. Но люди несовершенны и очень часто забывают свои PIN. Поэтому для таких случаев предназначены специальные процедуры: восстановление забытого PIN или генерация нового.

При идентификации клиента по значению PIN и предъявленной карте используются два основных способа проверки PIN: неалгоритмический и алгоритмический.

Неалгоритмический способ осуществляется путем непосредственного сравнения введенного клиентом PIN со значениями, хранимыми в базе данных. Обычно база данных со значениями PIN клиентов шифруется методом прозрачного шифрования, чтобы повысить ее защищенность, не усложняя процесса сравнения.

Алгоритмический способ проверки PIN заключается в том, что введенный клиентом PIN преобразуют по определенному алгоритму с использованием секретного ключа и затем сравнивают со значением PIN, хранящимся в определенной форме на карте. Достоинства этого метода проверки:

- отсутствие копии PIN на главном компьютере исключает его раскрытие персоналом банка;
- отсутствие передачи PIN между банкоматом или POS-терминалом и главным компьютером банка исключает его перехват или навязывание результатов сравнения;
- Упрощение работы по созданию программного обеспечения системы, так как уже нет необходимости действий в реальном масштабе времени.

Кражи денег с пластиковых карт с использованием банкоматов

Так выглядела щель банкомата обычно

Так выглядит щель банкомата со скиммером

Принцип здесь таков: скиммер считывает информацию с магнитной ленты Вашей карточки, позволяя легко изготовить ее дубликат. А ***накладка на клавиатуру*** позволяет узнать PIN, который Вы вводите. В результате через 2 минуты после Вашего общения с банкоматом, злоумышленники имеют возможность снять все оставшиеся на вашей карточке деньги (в пределах ограничений на выдаваемые суммы банкоматом, но как их обходить знают все).

Если вы заметите подобное, сразу звоните в саппорт банка по телефону, указаному на карточке! Не отходя от банкомата, т.к. злоумышленники сидят в одной из недалеко припаркованных автомашин. Такой фарш ставится на банкомат на одну ночь или даже на несколько часов. Злоумышленники заранее изучают посещаемость банкомата в ночное время, наличие охраны, камер и прочие факторы, определяющие соотношение риск/заработок в конкретном случае.

Очень часто банкомат выбирается по следующим принципам:

- в спальном районе (большой размер аудитории)
- рядом с отделением банка (наглость конечно, но оправданная - банкомат кажется надежным)
- на улице (вариант установки скиммера на банкомат в помещении менеевероятен)
- рядом с местом где всю ночь тратятся деньги (McDonalds, торговые центры)
- уличных камер нет (я не заметил)
- стихийная автостоянка вокруг (на еще одну припаркованную машину никто не обратит внимания)

Вот так может выглядеть скиммер:

или вот так (вообще не заметишь):

или даже вот так

вот этот скиммер в сравнении с оригинальной щелью

Накладная клавиатура и скиммер:

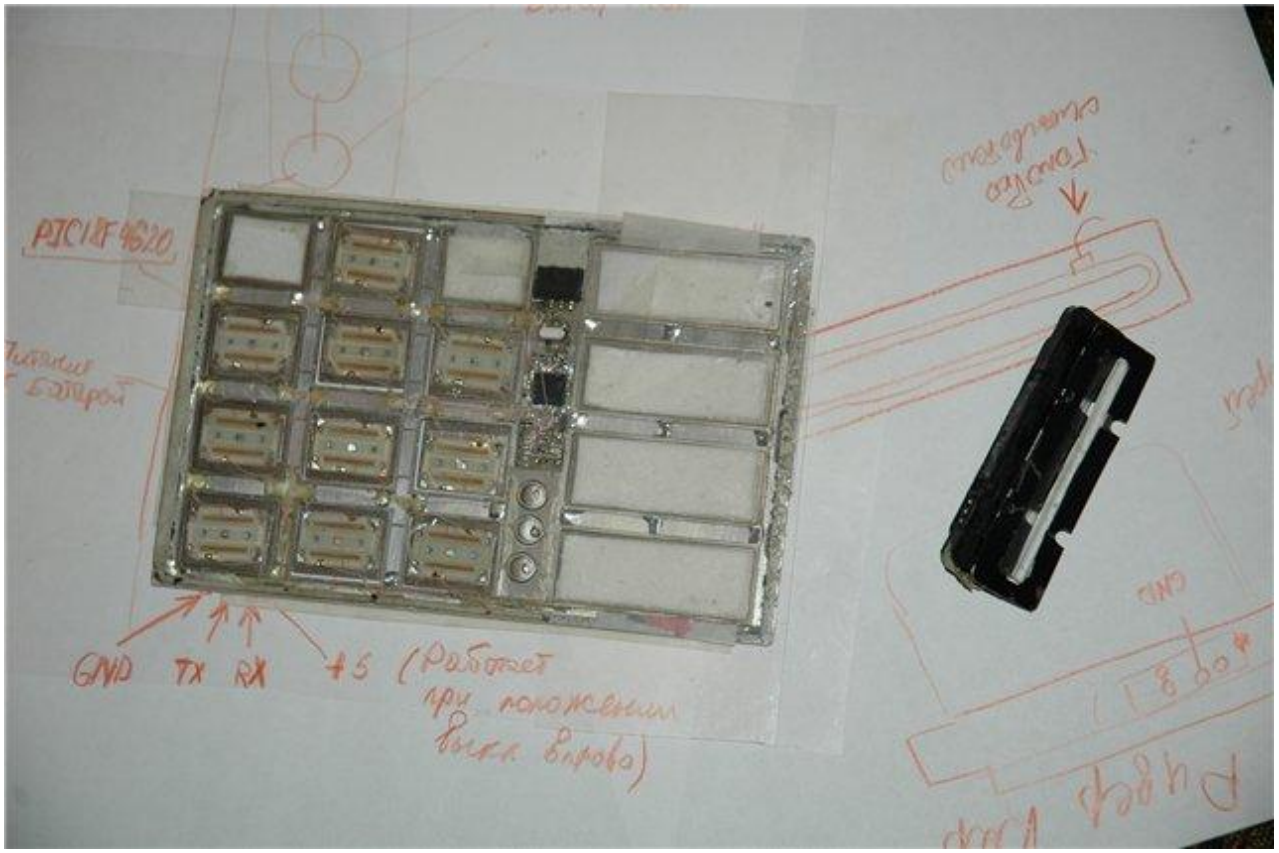
Настоящая щель под скиммером:

Еще один мегадевайс:

FAQ / Вопросы-ответы:

вопрос: Как с помошью скиммера считать ПИН?
ответ: никак!
ПИН считывается одним из двух способов:
1. "накладка" - фальшивая клавиатура поверх настоящей. Вы просто вводите ею свой пин. Причем она работает как прокси, т.е. нажатия передаются на оригинальные клавиши и вы не заметите разницы в эксплуатации банкомата.

К примеру она может выглядеть так:



2. скрытая камера где-то рядом. например в навесном ящичке с буклетами банка

К примеру может выглядеть так:

А потом в сети появляются объявления подобные этому

Идеальная служба информационной безопасности

Что должно быть реализовано в организации при условии, что служба информационной безопасности работает в некоторых идеальных условиях (неограниченный бюджет, достаточное количество квалифицированных специалистов и т.п.):

1. Вся работа службы информационной безопасности (как и всего предприятия) организована в строгом соответствии с законодательством государства.
2. Все кандидаты для приема на работу проходят собеседование и проверку службой информационной безопасности. Вновь принятые работники и сторонние специалисты по контракту проходят обучение основам безопасности, ознакомление с нормативными документами по безопасности, тестирование на уровень квалификации для работы с информационными системами и подтверждают подписью свое обязательчтво следовать нормативам организации (в том числе и по безопасности). Увольняющиеся работники проходят собеседование с представителем службы безопасности, увольнение работника сразу отражается в информационных системах (блокирование и удаление учетных записей, изъятие карт доступа и т.д.). Сотрудники организации проходят периодические семинары и обучение по информационной безопасности.
3. Ведется анализ моральной и психологической обстановки в организации, учет нарушений безопасности конкретными сотрудниками для выявления возможных тенденций. Поощряется тесный информационный контакт пользователей со службой безопасности. Выборочно или постоянно анализируется электронная почта сотрудников с соответствующим нормативным оформлением процедуры. Регулярно производится авторизованный мониторинг активности пользователя на рабочей станции.
4. Все данные, объекты и информационные системы проклассифицированны. Субъекты распределены по ролям. Определена надежная структура и внедрен механизм доступа субъектов к объектам с учетом наименьших привилегий и разделения обязанностей. Каждый новый объект своевременно классифицируется и устанавливается в общую схему информационного пространства.
5. Обеспечено нормативное пространство. Для всех информационных систем существуют политики, для функциональных обязанностей пользователей - правила, процедуры и методики. Изменения в работе организации и сотрудников адекватно отражаются в соответствующих документах.
6. Способы аутентификации пользователей находятся под контролем службы информационной безопасности, т.е. производится периодический анализ отсутствия слабых паролей, фактов передачи паролей другим лицам, оставление токенов без надзора и т.п. Идентификация пользователей стандартизирована, имеется четкая таблица соответствия пользователь - сетевые адреса, разрешенные для работы данного пользователя (username, IP-адрес, MAC-адрес и т.д.).
7. Внешний удаленный доступ в информационную сеть предприятия (выход во внешнее информационное пространство) ограничен единственным центральным шлюзом плюс существует резервный канал связи, неактивный в штатном режиме. Оба канала защищены межсетевыми экранами, которые

надежно функционируют, корректно настроены и регулярно подвергаются проверке службой информационной безопасности. Все модемы и другие устройства удаленного доступа учтены и также сведены к указанной единой точке входа/выхода. Снаружи по отношению к точке входа установлен агент ***сетевой системы обнаружения атак*** (СОА).

8. Агент сетевой СОА присутствует на каждом сегменте в сети организации. Для СОА сигнатурного типа базы данных сигнатур регулярно обновляются, имеется специалист по созданию собственных сигнатур. Для СОА с выявлением аномалий определены все необходимые профили, которые регулярно пересматриваются. Кроме этого производится периодический анализ сетевой активности средствами сетевого мониторинга (перехватчиками сетевых пакетов - снифферами).
9. Агента СОА-хоста установлены на каждом узле сети, база данных атак (или профили) регулярно обновляются. Кроме этого производится периодический выборочный анализ регистрационных журналов (которые настроены на фиксацию всех событий), в том числе лично уполномоченным персоналом. СОА интегрирована с межсетевым экраном и другими устройствами защиты.
10. Настроено подробное ведение регистрационных журналов по всем информационным системам. Ведется регулярная автоматизированная обработка этих журналов, а также периодический выборочный ручной их анализ на предмет выявления подозрительных или злоумышленных событий. Система анализа информационной активности интегрирована с системой физического доступа сотрудников в здание и к рабочим местам. Все регистрационные журналы сохраняются наряду с резервными копиями и архивами бизнес-данных.
11. На каждый компьютер в сети установлен антивирусный пакет, кроме того антивирусы установлены на почтовый сервер, межсетевой экран и другие ключевые узлы. Режим работы антивируса - перехват на лету (autoprotect). Антивирусные базы обновляются ежедневно, а также при поступлении информации о новом вирусе.
12. Ведется строгий учет движения любой единицы аппаратного обеспечения или ее составляющей, а также строгий учет аппаратной или программной конфигурации каждого объекта или системы. При изменении конфигурации немедленно производится сохранение соответствующих настроек. Ведется регулярный мониторинг производительности работы аппаратного обеспечения, операционных систем, информационной системы в целом.
13. Произведена подписка на соответствующие бюллетени по информационной безопасности, изучаются сообщения о новых атаках, вирусах, уязвимостях и т.п., новых решениях и механизмах по безопасности. Налажена процедура регулярного получения и установки программных заплат (patches, hotfixes, updates и т.п.) и их тестирование. Производится аналитическая работа по определению тенденций развития атак, появлению уязвимостей, новых продуктов на рынке безопасности, новых технологий.
14. Все каналы обмена данными в информационной сети криптографически защищены, внутри локальной сети организованы виртуальные сети. Любой обмен данными регистрируется в электронных журналах и снабжен средством контроля целостности. Обеспечен контроль целостности данных, системных файлов и т.п. на серверах и рабочих станциях. Обмен данными между пользователями производится с использованием электронной цифровой подписи. Организовано надежное управление криптографическими ключами.
15. Обеспечен контроль входящей и исходящей информации на внешних носителях. Установлены надежные защищенные (возможно, виртуальные)

шлюзы обмена информацией с внешними информационными системами (обеспечены соответствующие интерфейсы, установлена связь с центром сертификатов, получен корневой сертификат организации и т.д.).

16. Регулярно производится процедура тестирования всей сети или отдельных систем на взлом. В распоряжении службы информационной безопасности имеется команда программистов для создания собственных программ или утилит для анализа защищенности либо, наоборот, для защиты объектов и систем.
17. После изменения любой единицы данных обеспечивается ее резервное копирование либо организовано избыточное сохранение данных (RAID). Обеспечивается географически разнесенное защищенное сохранение архивов и резервных копий. Все сетевые устройства имеют возможность быстрой замены (продублированы), все каналы передачи данных имеют альтернативные линии. Здание организации имеет горячий резерв (hotside). Обеспечено бесперебойное электропитание (и контроль его работы) основного и резервного зданий. Ключевые работники организации могут выполнять функции друг друга либо имеют функциональных дублеров. Имеется регулярно обновляемый аварийный план.
18. Контроль за любой системой или объектом децентрализован. Служба информационной безопасности принимает участие в любом проекте организации (в том числе и в разработке программного обеспечения) с правом внесения серьезных изменений и запретов в рамках своих функций. Служба имеет полномочия к принятию и реализации решений, связанных с информационной безопасностью.
19. Установлена система отвлечения внимания злоумышленника (honeypot), сформирована собственная команда реагирования на инциденты, установлена связь с аналогичными командами других организаций, внешними экспертами, силовыми структурами.
20. Обеспечено единое согласованное неротиворечивое управление всеми автоматизированными средствами информационной безопасности.

Следует обратить внимание на то, что одни работы носят одноразовый характер по типу "сделали и забыли". Вторые - это разовое выполнение с последующим минимальным контролем соответствия текущего состояния некоторым условиям. Третьи - периодические работы по истечении промежутка времени или наступлению какого-то события. И четвертые - рутинные постоянные работы типа мониторинга.

Естественно, что в реальной жизни не удастся реализовать эту идеальную программу полностью. Однако, во-первых, она обозначает некоторую цель, к которой следует стремиться, а во-вторых, многие из обозначенных пунктов можно реализовать в усеченном варианте.

Коммерческая тайна. Защита от изъятия

В каждой организации есть обладающая коммерческой ценностью информация, разглашение которой является нежелательным. Это могут быть сведения о деятельности фирмы, ее контрагентах (наименование, ИНН, адрес), условия заключаемых ею сделок, различные внутренние проекты, расчеты и т.д.

При этом, желая ограничить доступ к такой информации, компании обычно не выходят за рамки устных указаний своим сотрудникам, установления паролей на отдельных офисных компьютерах, приобретения сейфа для хранения документов. Однако всех этих мер оказывается недостаточно, когда речь заходит о контрольных мероприятиях, проводимых различными государственными контролирующими структурами (налоговые органы, органы МВД и прокуратуры и т.п.) При наличии соответствующих оснований компьютеры могут быть изъяты, сейфы - вскрыты, пароли - расшифрованы. В результате организация оказывается незащищенной перед лицом проверок со стороны различных государственных органов.

Существуют ли правовые механизмы для защиты коммерческой информации от изъятия контролирующими органами?

Особой информации - особый режим

В российском законодательстве уже давно появилась возможность защитить легальными способами коммерческую информацию от разглашения. В соответствии с главой 75 ГК РФ каждая организация вправе ограничить доступ к информации, связанной с ее предпринимательской деятельностью, а также установить в отношении нее режим коммерческой тайны, и тогда у этой организации возникнет исключительное право на использование соответствующих сведений, то есть, исключительное право на секрет производства (ноу-хау).

Какая же информация может быть защищена с помощью режима коммерческой тайны? Ответ на этот вопрос содержится в Федеральном законе от 29 июля 2004 г. № 98-ФЗ «О коммерческой тайне» (далее - Закон о коммерческой тайне), согласно которому режим коммерческой тайны может быть распространен на любые сведения, которые имеют действительную или потенциальную коммерческую ценность в силу их неизвестности третьим лицам. К их числу может быть отнесена информация любого характера, в том числе результаты интеллектуальной деятельности в научно-технической сфере, сведения о способах осуществления профессиональной деятельности - например, содержание сделок с контрагентами, результаты исследований рынка, отчеты сотрудников, содержание деловой переписки и т.д. Исключения составляют сведения, содержащиеся в учредительных документах, численность и состав работников, задолженность по выплате заработной платы и некоторые другие сведения.

С момента установления организацией режима коммерческой тайны конфиденциальность подпадающих под действие этого режима сведений охраняется законом, также как и интересы организации, связанные с неразглашением этих сведений (в т.ч. из-за необоснованного изъятия со стороны контролирующих органов, доведения до сведения контрагентов и т.п.). Согласно п. 1 ст. 14 Закона о коммерческой тайне лица, необоснованно получившие доступ к информации, составляющей коммерческую тайну,

могут понести за это дисциплинарную, гражданско-правовую, материальную, административную или уголовную ответственность.

Вопрос: могут ли государственные органы на законных основаниях получить при проведении контрольных мероприятий доступ к информации, составляющей коммерческую тайну.

От чего защитит коммерческая тайна?

Общее правило, регламентирующее порядок предоставления государственным органам информации, составляющей коммерческую тайну, закреплено в ст. 6 Закона о коммерческой тайне. С одной стороны, в ней говорится, что по мотивированному требованию государственного органа предоставить информацию, составляющую коммерческую тайну, обладатель этой информации предоставляет ее на безвозмездной основе. Вместе с тем, законом предусмотрена возможность обладателя конфиденциальной информации отказаться от ее представления. В случае такого отказа соответствующий государственный орган вправе затребовать необходимую ему информацию только в судебном порядке.

Иными словами, если организация сочтет, что истребование у нее «засекреченной» информации может навредить ее интересам, она может отказаться от ее предоставления. В этом случае насильственное изъятие соответствующей информации без санкции суда будет являться незаконным и станет основанием для привлечения виновных лиц к ответственности.

Правоохранительные органы не вправе изымать информацию, составляющую коммерческую тайну. Для ее получения им необходимо обратиться за санкцией суда, который и должен будет принять решение об обоснованности такого изъятия. При этом, даже если силовики осуществят насильственное изъятие информации в обход закона, они не смогут использовать эту информацию в качестве доказательств, поскольку такие доказательства будут считаться полученными с нарушением закона.

Однако, есть исключение. И состоит оно в изъятии у налогоплательщика информации налоговыми органами. Действующее российское законодательство не препятствует налоговикам осуществлять выемку документов и предметов в порядке ст. 94 НК РФ, даже если на них зафиксирована информация, составляющая коммерческую тайну, и для этого им не надо получать специальную санкцию суда. Почему же у налоговых органов особый статус при работе с «засекреченной» информацией?

В налоговом законодательстве предусмотрен специальный институт, направленный на защиту конфиденциальности информации о налогоплательщиках - институт налоговой тайны. Согласно ст. 102 НК РФ налоговую тайну составляют любые полученные налоговым органом сведения о налогоплательщике (за некоторыми исключениями). Все эти сведения имеют специальный режим хранения и доступа и не подлежат разглашению, в противном случае виновных лиц можно привлечь к ответственности. Это в полной мере обеспечивает соблюдение интересов проверяемого лица по сохранению конфиденциальности полученной налоговыми органами информации.

Вместе с тем, выемка документов и предметов налоговыми органами производится только на основании мотивированного постановления, причем исключительно в случае непредставления затребованных документов либо при наличии реальной угрозы

уничтожения, сокрытия, изменения или замены этих документов. Доказать необходимость произведения выемки при отсутствии указанных выше обстоятельств достаточно трудно, поэтому налоговым органам проще привлечь для проведения выемки предметов милиционеров, для которых подобных ограничений нет. Тем более, что при проведении выездных налоговых проверок в состав проверяющих часто включаются представители МВД России, которые активно участвуют в ней, а при необходимости «содействуют» в изъятии. Единственным препятствием для них может стать введение режима коммерческой тайны.

Порядок внедрения режима коммерческой тайны

Внедрение коммерческой тайны является трудоемким процессом и требует профессионального подхода, поскольку данная процедура подвержена тщательной законодательной регламентации, и при несоблюдении каких-либо формальностей режим коммерческой тайны не будет считаться установленным, а соответствующая информация не получит защиты. Чтобы «режим секретности» начал действовать, необходимо выполнение нескольких обязательных условий, связанных с обращением конфиденциальной информации. Их лучше всего закрепить в специально разработанном Положении, которое утверждается приказом генерального директора.

1. Определение перечня сведений, составляющих коммерческую тайну, исходя из существующих потребностей. Сюда целесообразно отнести любые сведения об условиях сделок между организацией и ее контрагентами, содержащиеся в договорах, актах и иных сопутствующих документах, внутренние отчеты о проделанной работе, бухгалтерские справки и т.п. Обратите внимание: при определении перечня информации, составляющей коммерческую тайну не всегда обязательно ссылаться на видовые признаки соответствующих сведений - обычно можно ограничиться их общими характеристиками: например, содержание электронной переписки сотрудников организации с представителями бизнес-партнеров.
2. Ограничение доступа к информации, составляющей коммерческую тайну. Выполнение данного условия достигается за счет введения специальных правил хранения материальных носителей информации, составляющей коммерческую тайну (например, документы - в сейфе, на жестких дисках - пароли), а также за счет наделения работников организации полномочиями по работе с такой информацией. При этом вопрос о полномочиях работников можно решить двумя путями: либо наделять каждого конкретного сотрудника определенными правами и обязанностями по работе с конфиденциальной информацией, либо в общем виде оговорить ограничения, вводимые для отдельных категорий сотрудников в связи с введением в организации режима коммерческой тайны. Не следует также забывать, что в трудовые договоры с ответственными сотрудниками следует внести дополнения об обязанности по соблюдению действующего в организации режима коммерческой тайны (в этом случае проще будет добиться выполнения установленных правил, а при необходимости - привлечь работника к дисциплинарной или материальной ответственности).
3. Учет лиц, которым предоставляется или передается конфиденциальная информация. Обычно это достигается путем утверждения соответствующих ведомостей (табелей), обязанности по ведению которых возлагаются на конкретных сотрудников организации.

4. Установление ограничений на использование конфиденциальной информации работниками на основании трудовых договоров и контрагентами на основании гражданско-правовых договоров. Иными словами, работники и контрагенты, в случае получения доступа к такой информации должны быть уведомлены о статусе данных сведений, о порядке и условиях их использования.
5. Нанесение на материальные носители (документы, компьютеры и т.д.), содержащие информацию, составляющую коммерческую тайну, грифа «Коммерческая тайна» с указанием на полное наименование организации и место ее нахождения (для нанесения грифов лучше заказать ненаборный штамп в специализированной организации). Несмотря на кажущуюся формальность данного требования, его соблюдение также обязательно, поскольку сам факт отсутствия грифа на материальных носителях является достаточным основанием для признания режима коммерческой тайны неустановленным. В то же время предназначение грифа вполне ясно: он сигнализирует посторонним лицам об особом статусе информации, зафиксированной на конкретном материальном носителе.

Только при реализации всех вышеперечисленных мер режим коммерческой тайны будет считаться установленным, и у организации появится возможность защищать информацию от разглашения. Еще раз отметим: соблюсти все тонкости в данном случае непросто, поэтому если существует реальная потребность в защите конфиденциальности информации, то проведение соответствующих процедур стоит доверить специалистам соответствующего профиля.

Оглавление

Оглавление

Защита информации в компьютерных системах 1

Вступление. Информация как объект защиты 1

Информационная безопасность 4

Основные угрозы информационной безопасности 4

Обеспечение информационной безопасности 6

Аппаратно-программные средства защиты информации 7

1. Системы идентификации и аутентификации пользователей 8

2. Системы шифрования дисковых данных 8

3. Системы шифрования данных, передаваемых по сетям 9

4. Системы аутентификации электронных данных 9

5. Средства управления криптографическими ключами 10

Перечень документов 10

Критерии оценки безопасности компьютерных систем."Оранжевая книга" США 13

Основные элементы политики безопасности 14

Произвольное управление доступом 14

Безопасность повторного использования объектов 15

Метки безопасности 15
Принудительное управление доступом 16
Классы безопасности 17
Требования к политике безопасности 17
Требования к подотчетности 19
Требования к гарантированности 20
Требования к документации 23
Криптографические средства защиты (шифрование) информации 25
Простые криптосистемы 25
Основные требования к криптографическому закрытию информации в АС 25
Классификация основных методов криптографического закрытия информации 25
Шифрование методом замены (подстановки) 26
Одноалфавитная подстановка 26
Многоалфавитная одноконтурная обыкновенная подстановка 27
Многоалфавитная одноконтурная монофоническая подстановка 29
Многоалфавитная многоконтурная подстановка 29
Шифрование методом перестановки 30
Простая перестановка 30
Перестановка, усложненная по таблице 31
Перестановка, усложненная по маршрутам 31
Шифрование методом гаммирования 32
Шифрование с помощью аналитических преобразований 33
Комбинированные методы шифрования 33
Организационные проблемы криптозащиты 34
Литература 35
Стандарт шифрования данных Data Encryption Standard 36
Режимы работы алгоритма DES 45
DES-CFB 48
Алгоритм шифрования данных IDEA 50
Отечественный стандарт шифрования данных 54
Режим простой замены 54
Режим гаммирования 56
Режим гаммирования с обратной связью 56
Выработки имитовставки 57
Концепция криптосистемы с открытым ключом 58

Однонаправленные функции 58
Система распределения ключей Диффи-Хеллмана 59
Система RSA 61
Pretty Good Privacy (PGP) (довольно хорошая секретность) 64
Электронная подпись в системах с открытым ключом 64
О "двуличии" в алгоритмах цифровой подписи 65
Электронная цифровая подпись 67
1. Проблема аутентификации данных и электронная цифровая подпись 67
2. Однонаправленные хэш-функции 68
Основы построения хэш-функций 69
Однонаправленные хэш-функции на основе симметричных блочных алгоритмов 70
Алгоритм MD5 72
Алгоритм безопасного хэширования SHA 74
Отечественный стандарт хэш-функции 76
3. Алгоритмы электронной цифровой подписи 77
Алгоритм цифровой подписи RSA 78
Алгоритм цифровой подписи Эль Гамаля (EGSA) 81
Алгоритм цифровой подписи DSA 83
Отечественный стандарт цифровой подписи 85
Литература 86
Генерация личных ключей 88
Защита от копирования 90
Привязка к дискете 91
Перестановка в нумерации секторов 91
Введение одинаковых номеров секторов на дорожке 91
Введение межсекторных связей 92
Изменение длины секторов 92
Изменение межсекторных промежутков 93
Использование дополнительной дорожки 93
Ведение логических дефектов в заданный сектор 93
Изменение параметров дисковода 95
Технология "ослабленных" битов 95
Физическая маркировка дискеты 95
Применение физического защитного устройства 96
"Привязка" к компьютеру. 96

Физические дефекты винчестера 97
Дата создания BIOS 97
Версия используемой OS 97
Серийный номер диска 97
Тип компьютера 97
Конфигурация системы и типы составляющих ее устройств 98
Получение инженерной информации жесткого диска 101
Опрос справочников 101
Введение ограничений на использование программного обеспечения 102
Защита CD от копирования 104
Защиты от несанкционированного доступа 115
Идентификация и аутентификация пользователя 116
Взаимная проверка подлинности пользователей 120
Литература 122
Протоколы идентификации с нулевой передачей знаний 123
Упрощенная схема идентификации с нулевой передачей знаний 124
Параллельная схема идентификации с нулевой передачей знаний 125
Схема идентификации Гиллоу - Куискуотера 128
Компьютерная графология 130
Защита исходных текстов и двоичного кода 132
Противодействие изучению исходных текстов 132
Динамическое ветвление 132
Контекстная зависимость 133
Хуки 133
Противодействие анализу двоичного кода 134
Заключение 137
ТРИ КЛЮЧА 137
СТРУКТУРА КОМАНД INTEL 80x86 137
Префикс 138
Код операции 140
Байт modR/M 142
Защита от отладчиков 155
Средства отладки и взлома программ 155
Отладчики реального режима 155
Отладчики защищенного режима 156

Эмуляторы....156
Автоматические распаковщики....157
Дизасемблеры....158
Прочие программы....158
Отладчик SoftIce....159
1. Загрузка отлаживаемой программы....159
2. Управление SoftIce'ом....159
3. Трассировка программы....161
4. Просмотр локальных переменных....162
5. Установка точек останова на выполнение....162
6. Использование информационных команд....163
7. Символьные имена....164
8. Условные точки останова....165
Как заставить SoftIce работать?....168
Как заставить SoftIce/Win/W95 работать?....168
Защита от отладчиков....169
Работа в защищенном режиме....171
Программы с потенциально опасными последствиями....173
Вирус....174
Люк....175
Троянский конь....176
Логическая бомба....176
Программные закладки....176
Атака салями....181
Защита в интернет....183
Межсетевые экраны....183
Пакетные фильтры....184
Сервера прикладного уровня....184
Сервера уровня соединения....185
Сравнительные характеристики пакетных фильтров и серверов прикладного уровня185
Виртуальные сети....186
Схемы подключения брандмауэров....186
Администрирование....188
Системы сбора статистики и предупреждения об атаке....188
Аутентификация....189

Классы защищенности брандмауэров 189
Руководство для приобретающих брандмауэр 191
Компьютерные атаки и технологии их обнаружения 192
Модели атак 193
Этапы реализации атак 195
1. Сбор информации 196
2. Реализация атаки 198
Цели реализации атак 199
3. Завершение атаки 199
Классификация атак 199
Заключение 200
Средства обнаружения компьютерных атак 201
Классификация систем обнаружения атак 201
Достоинства систем обнаружения атак 202
Сетевые системы обнаружения атак и межсетевые экраны 204
Варианты реакций на обнаруженную атаку 204
Безопасность электронной коммерции 205
Протокол SSL 206
Протокол SET 208
Сравнительные характеристики протоколов SSL и SET 212
Безопасность электронных платежных систем 215
Кражи денег с пластиковых карт с использованием банкоматов 222
FAQ / Вопросы-ответы: 226
Идеальная служба информационной безопасности 229
Коммерческая тайна. Защита от изъятия 232
Особой информации - особый режим 232
От чего защитит коммерческая тайна? 233
Порядок внедрения режима коммерческой тайны 234
Оглавление 235