Программно-аппаратная защита программного обеспечения от пиратства с использованием электронных ключей

Иванов Е.В., 913 группа

На сегодняшний день это – наиболее надежный и удобный метод защиты тиражируемого ПО средней и высшей ценовой категории. Он обладает высокой стойкостью к взлому и не накладывает ограничений на использование легальной копии программы. Применение этого метода экономически оправдано для программ стоимостью свыше $80, так как использование даже самых дешевых электронных ключей увеличивает стоимость ПО на $10-15. Электронными ключами, в основном, защищают так называемый "деловой" софт: бухгалтерские и складские программы, правовые и корпоративные системы, строительные сметы, САПР, электронные справочники, аналитический софт, экологические и медицинские программы и т. п. Затраты на разработку таких программ велики, и, соответственно, высока стоимость софта, поэтому ущерб от пиратского распространения значителен. В этом случае электронные ключи являются оптимальным средством защиты.Электронные ключи представляют собой небольшие устройства, подсоединяемые к компьютеру через LPT, USB, COM, ADB, PCMCIA. В общих чертах схему работы защиты можно описать следующим образом: приложение «привязывается» к ключу при помощи специального софта, во время работы защищенное приложение обменивается с ключом информацией,которая помогает « опознать » ключ и если ключ отсутствует или имеет неверные параметры,то приложение не работает.

Электронные ключи предоставляют два метода защиты программного обеспечения.

Первый метод, т.н. автоматическая защита, основан на обработке исполняемого файла или dll программы специальной утилитой. В результате программа «одевается» в защитный конверт и «привязывается» к электронному ключу. После этих манипуляций этот самый конверт периодически проверяет наличие ключа, если результат проверки оказывается отрицательным программа либо полностью прекращает работу либо переходит в демонстрационный режим. Установка такой защиты занимает считанные минуты и не требует специальных знаний.

Второй метод основан на внедрении в исходный код приложения функций API. Из-за того что в данном случае требуется написание дополнительного кода(additional programming) для применения этого метода требуется дополнительное время. В этом случае защита полностью интегрируется с программой и составляет с ней единое целое, поскольку встраивание защиты в приложение происходит на уровне исходного кода. Здесь уже разработчик может выбирать число, типы и периодичность вызовов функций проверки наличия ключа, а так же определить свои действия в случае если результат проверки оказался отрицательным. Сегодняшние электронные ключи поддерживают множество различных алгоритмов проверки и благодаря этому разработчик может варьируя эти алгоритмы получать больший уровень

безопасности. Эти алгоритмы прошиваются в микроконтроллер находящийся внутри устройства. Количество таких алгоритмов варьируется в пределах нескольких десятков (например для ключей компании Guardant эта цифра вроде как равна 18, для ключей от WIBU указана цифра 28). Также можно маскировать блокирование работы программы внутри очереди рандомных запросов и случайным образом реагировать на результаты этих запросов, таким образом в конечном итоге увеличивая стоимость взлома программы. Ещё один эффективный прием усложнения логики защиты - это откладывание реакции программы на коды возврата функций API. В этом случае программа принимает решение о дальнейшей работе спустя какое-то время после получения кодов возврата. Что заставляет взломщика прослеживать сложные причинно-следственные связи и исследовать в отладчике слишком большие участки кода.

Далее рассмотрим методы взлома защиты программ защищённых электронными ключами.

Изготовление аппаратной копии электронного ключа. С помощью специальных программных или аппаратных средств , считывается содержимое микросхемы памяти электронного ключа и переносится в аналогичные микросхемы памяти других электронных ключей(болванки). Таким образом,создаются аппаратные копии электронного ключа,пригодные для работы с защищенной программой. Создание аппаратных копий ключа применимо только в тех случаях, когда известен конкретный тип микросхемы памяти, когда имеется специальное аппаратное устройство (программатор) для записи данных в микросхему памяти ключа копии и, самое главное, – при условии, что память электронного ключа никак не защищена от ее нелегального считывания. Именно это третье условие и стало «ахиллесовой пятой» некоторых марок электронных ключей, слабость (а то и полное отсутствие) защиты данных в памяти ряда ключей обусловило распространение этого метода взлома. Методу присущи несколько недостатков. Главный из них в том,что, даже получив нелегальную копию программы, не удастся решить задачу ее незаконного распространения – ведь копия так и осталась защищенной, просто теперь она « привязана » к « левому » электронному ключу. Кроме этого,достаточно велика стоимость такого копирования – чаще всего дешевле просто купить легальную копию продукта. Поэтому такой метод не получил столь же широкого распространения, как следующий.

Создание эмулятора электронного ключа. В данном случае создается некий программный модуль – резидентная программа ,DLL, драйвер или т.п., которые в той или иной степени эмулируют работу электронного ключа.

Качественно написанный эмулятор способен воспроизвести работу ключа во всех тонкостях, включая особенности протокола обмена между защищенной программой и ключом, генерацию им верных ответов для «всех случаев жизни» и т.д. В результате защищенная программа будет спокойно работать, даже не подозревая, что имеет дело отнюдь не с электронным ключом. По степени универсальности эмуляторы можно разделить на универсальные (эмулируют работу любого электронного ключа определенной марки)и неуниверсальные или crack'и (эмулируют работу всех ключей конкретного пользователя защиты, ключей, поставляемых с определенным программным продуктом или с определенной его версией, либо одного конкретного ключа). По принципу организации эмуляторы можно разделить на эмуляторы структуры ключа (как правило, это универсальные эмуляторы ключей без аппаратных алгоритмов, либо ключей с примитивно простыми аппаратными алгоритмами) и эмуляторы

ответов электронных ключей. При этом первые воспроизводят структуру электронного ключа во всех ее тонкостях, что и обусловливает их универсальность, а вторые используют специально созданную таблицу верных ответов ключа.В любом случае,эмуляторы должны верно воспроизводить не только посылаемые ключом данные,но и протокол обмена с ним (в противном случае защищенная программа не сможет «понять» переданных эмулятором данных).

Эмуляторы электронных ключей – это мощный и эффективный метод взлома программно аппаратной защиты, получивший достаточно широкое распространение.

При написании эмулятора электронного ключа перед его создателем встают две задачи:

Отделить модуль автоматической защиты. Так как защита устанавливается на готовую программу, она не может образовать с ней неразрывное целое. Модуль автоматической защиты как бы «приклеивается» к программе, а это значит, что есть теоретическая возможность отделить, «отклеить» его, причем без ущерба для самой программы. Существуют даже специальные утилиты автоматического взлома программ, защищенных таким методом.

Удалить вызовы функций API. На сегодняшний день как правило API существуют для всех популярных языков программирования и содержат самые разнообразные функции. Разработчик защиты должен вписать вызовы нужных функций в исходный текст защищаемой программы, скомпилировать и слинковать его с соответствующим объектным модулем API. В результате защита будет внедрена в тело программы, и при правильной организации логики работы будет составлять с программой неразрывное целое. Взломать защиту такого типа гораздо сложнее, чем автоматическую, хотя бы потому, что невозможно создать универсальное средство автоматического взлома. При помощи дизассемблеров и отладчиков изучается логика работы программы, находятся места, в которых происходят вызовы функций API (либо точки входа в сами эти функции – смотря, что оказывается проще) и нужным образом исправляется программный код.

Далее будет рассмотрено содержимое API, на примере продукции компании «Aladdin» которая владеет самой большой частью рынка электронных ключей.

Далее по тексту будет использоваться аббревиатура: HASP (Hardware Against Software Piracy, а также семейство электронных ключей от этой компании). И русскоязычное понятие Код разработчика(уникальный код, присваиваемый компанией «Aladdin» каждому разработчику программного обеспечения). Код разработчика «зашивается» в микросхему ASIC(что это такое написано здесь) при изготовлении ключа и не поддаётся изменению, обеспечивая, таким образом, ещё одну степень защиты от подделки. Ещё есть пароль разработчика, представляющий из себя два целых числа и однозначно связанный с кодом разработчика.

В процессе работы программа контролирует наличие ключа с помощью шифрования и дешифрования данных самим ключом. Распознавание наличия ключа HASP основано на использовании функций шифрования и дешифрования, что влечет за собой необходимость в некоторых действиях. Сначала необходимо иметь уже некоторые данные, которые ранее были

зашифрованы. После этого данные посылаются на ключ, используя функцию DecodeData. Происходит их дешифрование, в результате чего можно проверить, верны ли дешифрованные данные. Если так, можно сделать заключение о наличии ключа. Дешифрованные данные могут быть верифицированы как путем простого сравнения, так и более безопасным образом – использованием этих данных в защищенном приложении. Зашифрованные данные представляют собой функцию посланных на ключ данных и уникального, присвоенного разработчику, «кода разработчика». Вследствие этого, при шифровании одной и той же строки для двух разных разработчиков будет получен различный результат.

HASP API реализуется в виде объектного файла, присоединяемого к приложению, либо DLL, вызываемой из приложения. Поскольку модуль API сам по себе защищён и зашифрован, он обеспечивает весьма высокий уровень защиты приложений. С помощью API можно для усиления защиты разнести обращения к ключу HASP по всему приложению. В любой точке приложения можно проверить наличие ключа и принять решение о дальнейших действиях в соответствии с результатами проверки. Можно также считывать из памяти ключа HASP хранящиеся там данные.

Защита API встраивается в приложение посредством использования процедуры hasp( ). Процедура hasp( ) проверяет наличие ключа HASP, шифрует данные во время работы приложения и выполняет операции чтения и записи данных с памятью ключа. Связь с ключом HASP реализуется через процедуру hasp( ), которая вызывается следующим образом:
hasp (Service, SeedCode, LptNum, Password1, Password2,Par1, Par2, Par3, Par4)
Процедура hasp( ) имеет девять параметров. Первый, Service (функция), определяет операцию, выполняемую процедурой
Par1, Par2, Par3 и Par4 – указатели.
Параметр LptNum используется для указания номера параллельного порта, в котором следует искать локальный ключ HASP.

Далее указаны четыре основные функции, доступные для всех локальных ключей HASP:

IsHasp Проверяет, подключён ли к компьютеру какой-либо ключ HASP.
Синтаксис:

hasp (Service, SeedCode, LptNum, Password1, Password2, Par1, Par2, Par3, Par4)

Аргументы:

Service: 1

LptNum: Номер параллельного порта, в котором следует искать HASP.

Возвращаемые значения:

Par1: HASP found (наличие HASP)

0 – Ключей HASP не обнаружено

1 – Присутствует ключ HASP любого типа

HaspStatus Определяет тип HASP, подключенного к компьютеру, к какому параллельному порту подключен ключ HASP размер памяти у ключей HASP с памятью.
Синтаксис:

hasp (Service, SeedCode, LptNum, Password1, Password2, Par1, Par2, Par3, Par4)

Аргументы:

Service: 5

LptNum: Номер параллельного порта, в котором следует искать HASP.

Рекомендуемое значение: 0.

Password1: Первый пароль HASP.
Password2: Второй пароль HASP.

Возвращаемые значения:

Par1: Размер памяти
1 – HASP4 M1
4 – HASP4 M4
0 – остальные
Par2: Тип HASP
0 – HASP4 без памяти
1 – HASP4 M1 или HASP4 M4
5 – HASP4 Time
Par3: Действительный номер LPT – параллельный порт, в котором обнаружен ключ HASP (200 и более для USB Hasp)
Par4: Объектная версия HASP – текущая версия API

HaspEncodeData Шифрует данные, посылаемые на присоединенный ключ HASP4. Использовать совместно с функцией HaspDecodeData для верификации присоединения требуемого ключа.

Синтаксис:

hasp (Service, SeedCode, LptNum, Password1, Password2, Par1, Par2, Par3, Par4)

Аргументы:

Service: 60
LptNum: Номер параллельного порта, в котором следует искать HASP.
Password1: Первый пароль HASP.
Password2: Второй пароль HASP.
Par1 0 (резервный)
Par2: Размер буфера. Размер в байтах шифруемого буфера. Размер буфера должен быть не менее 8 байт.
Par3: Сегментная часть адреса буфера. Вам не требуется ее указывать при использовании 32-битного интерфейса HASP API.
Par4: Смещение в буфере.

Возвращаемые значения:

Par3: Статус. Код индикации статуса операции.

HaspDecodeData Дешифрует данные, посланные на присоединенный ключ HASP4. Использовать совместно с функцией HaspEncodeData для верификации присоединения требуемого ключа.

Синтаксис:

hasp (Service, SeedCode, LptNum, Password1, Password2, Par1, Par2, Par3, Par4)

Аргументы:

Service: 61
LptNum: Номер параллельного порта, в котором следует искать HASP.
Password1: Первый пароль HASP.
Password2: Второй пароль HASP.
Par1 0 (резервный)
Par2: Размер буфера. Размер в байтах дешифруемого буфера. Размер буфера должен быть не менее 8 байт.
Par3: Сегментная часть адреса буфера. Вам не требуется ее указывать при использовании 32-битного интерфейса HASP API.
Par4: Смещение в буфере.

Возвращаемые значения:

Par3: Статус. Код индикации статуса операции.

Далее указаны значения, возвращаемые функцией hasp()

0 Операция завершена успешно.

-1 Тайм-аут: неуспешная операция записи.
-2 Недопустимый адрес.
-3 HASP с указанными паролями не найден.
-4 Ключ HASP обнаружен, однако это не MemoHASP.
-5 Ошибка записи.
-6 Параллельный порт в данный момент недоступен. Другое устройство, связанное с этим портом, например, принтер, активно. Повторить вызов API через несколько секунд.
-7 Размер буфера слишком мал. Эта ошибка может иметь место для функций, имеющих требование минимального размера буфера.

Здесь ещё раз повторю что эти функции доступны для ВСЕХ типов ключей. Однако есть ещё множество прочих функций характерных для отдельных разновидностей ключей от «Aladdin»(например для ключей у которых присутствует память или свои внутренние источники питания, есть соответствующие hasp() использующие эти преимущества). Напоследок приведу рекомендации от специалистов компании «Aladdin» относительно того что надо для того чтобы защита стала надежнее:

- Использовать больше вызовов hasp()(раскидать их побольше по исходному коду)
- Шифровать внутренние и внешние данные приложения
- Избегать повторяющихся схем
- Разделять шаги обращения к процедуре hasp()(вызов самой hasp(), оценка значений возвращённых hasp(), реакция на эти значения)
- Шифровать память ключа(если она там есть)
- Использовать контрольную сумму(CRC etc…)
- Использовать функционирование программы в качестве ответа на «HASP not found»(например заблокировать клавиатуру, если ключ не присоединен, а затем закончить операцию нормально, после того как ключ будет позже установлен)
- Прятать пароли
- Генерировать шум(вызвать hasp() с рандомными параметрами и не реагировать на возвращаемый результат)
- Использовать HASP-зависимые данные

Что это такое поясню подробнее:
Например, вместо того, чтобы проверять считывание данных из памяти ключа, можно использовать их непосредственно для перехода к некоторой метке, где выполняется операция, как это описано в следующем псевдокоде:

```
Begin
•••••••••• ••••••••• FLAG •••••••••••• •••••••••
••••••••.
Call ••••••••• hasp() • •••••••• ReadWord.
•••••••••• FLAG •••••••••••• •••••••• ••••••,
••••••••• •• •••••• ••••• HASP (• •••• •••••••
100).
Goto FLAG
. . .
. . .
. . .
Label 100
••••••••••• ••••••••, •••••••••••• ••• ••••••••••••
```

если ключ установлен, и правильное значение 100 считано из памяти ключа, программа продолжает работу и выполняет предписанную операцию после перехода к метке 100. Если же ключ не установлен, программа не достигнет назначенной метки, и предписанная операция выполняться не будет.

- Использовать HASP Envelope(что это такое см. выше)
- Менять свои стратегии(ну тут всё понятно)

В настоящее время лидерами в области электронных ключей являются следующие компании: RAINBOW Technologies, ALADDIN Knoweledge System, WIBU System, Marx Datentechnik, FAST Software Security, Software Security, EliaShim microcomputers, ProTech Marceting, MicroMacro, Glenco Engineering, АКТИВ (она известна как НОВЭКС-Софт/NOVEX Software Ltd.), Softlok International, Transcend Information.

Ссылки на использованные ресурсы:

- http://www.guardant.com/protection/methods.htm
- http://www.guardant.ru/data/booklett2002.zip
- http://www.rainbow.com/library/results.asp#
  (статья Preventing Software Piracy)
- http://www.wibu.com/us/wibukey.php
- http://www.ealaddin.com