



Защита программного обеспечения

Лекция 11



Способы распространения ПО

- **FreeWare** – свободное распространение с сохранением прав за автором
- **ShareWare** – дается некоторое время на ознакомление с программой (обычно 2-4 недели), после этого необходимо либо отказаться от использования программы, либо оплатить ее
- **CryptWare** – существует две версии программы: демоверсия с ограниченным функционалом + зашифрованная рабочая версия, за которую нужно платить



Защита от копирования

Защита от копирования — система мер, направленных на противодействие несанкционированному копированию информации, как правило, представленной в электронном виде (данных или кода проприетарного программного обеспечения)

При **защите** могут использоваться организационные, юридические и технические средства



Функции систем защиты

- **Идентификация** (т. е. присвоение индивидуального трудно подделываемого отличительного признака) той среды (диска или ПК), из которой будет запускаться защищаемая программа
- **Аутентификация** (опознавание) той среды, из которой поступает запрос на копирование защищаемой программы
- **Регистрация** санкционированного копирования
- **Реагирование** на попытки несанкционированного копирования
- **Противодействие** изучению алгоритмов работы системы защиты



Основные методы защиты

- Привязка программного обеспечения к уникальным характеристикам компьютера
- Создание ключевых дисков, которые сложно скопировать
- **Электронные ключи**
- Интернет-активация
- Обфускация кода
- **Защита ПО на уровне исходных кодов**

Что такое электронный ключ?

Электронный ключ (также аппаратный ключ, иногда донгл от англ. dongle) — аппаратное средство, предназначенное для защиты программного обеспечения (ПО) и данных от копирования, нелегального использования и несанкционированного распространения.



ОСНОВЫ КЛЮЧА

Основой данной технологии является **специализированная микросхема**, либо защищённый от считывания **микроконтроллер**, имеющие уникальные для каждого ключа алгоритмы работы.

Ключи также имеют:

- **защищённую энергонезависимую память** небольшого объёма
- более сложные устройства могут иметь **встроенный криптопроцессор** (для аппаратной реализации шифрующих алгоритмов)
- **часы реального времени**

Аппаратные ключи могут иметь различные форм-факторы, но чаще всего они подключаются к компьютеру через USB. Также встречаются с LPT- или PCMCIA-интерфейсами.

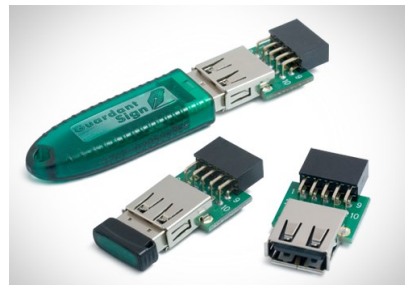


Технологии ключа

- Микропроцессорная технология
- Аппаратные алгоритмы $Y=F(X)$
- Коды доступа к электронному ключу
- Аппаратные запреты
- Кодирование данных в памяти Stealth-ключа
- Аппаратная блокировка отладчиков
- Особенности протокола обмена со Stealth-ключом
- Энергосбережение и полная «прозрачность» Stealth-ключа

Современные ключи

Стоит отметить, что некоторые современные ключи (Guardant Code от Компании "Актив", LOCK от Astroma Ltd., Rockey6 Smart от Feitian, Senselock от Seculab) позволяют разработчику хранить собственные алгоритмы или даже отдельные части кода приложения (например, специфические алгоритмы разработчика, получающие на вход большое число параметров) и исполнять их в самом ключе на его собственном микропроцессоре. Помимо защиты ПО от нелегального использования такой подход позволяет защитить используемый в программе алгоритм от изучения, клонирования и использования в своих приложениях конкурентами.



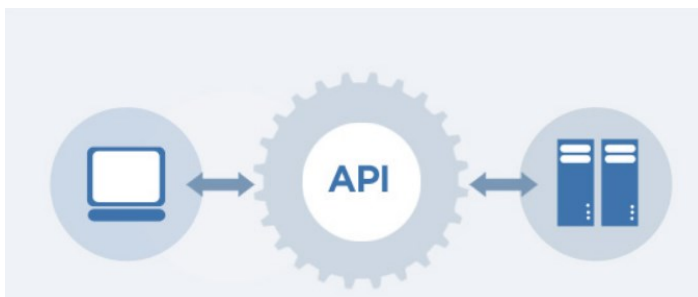
Защита с помощью функций API

Для этого в SDK включены библиотеки для различных языков программирования, содержащие описание функциональности API для данного ключа. API представляет собой набор функций, предназначенных для обмена данными между приложением, системным драйвером (и сервером в случае сетевых ключей) и самим ключом. Умелое применение данного метода обеспечивает высокий уровень защищённости приложений. Нейтрализовать защиту, встроенную в приложение, достаточно трудно вследствие её уникальности и «размытости» в теле программы. Сама по себе необходимость изучения и модификации исполняемого кода защищенного приложения для обхода защиты является серьёзным препятствием к её взлому. Поэтому задачей разработчика защиты, в первую очередь, является защита от возможных автоматизированных методов взлома путём реализации собственной защиты с использованием API работы с ключами.



Работа API

- Основным способом построения надёжной защиты является использование библиотеки API для работы с электронным ключом. Как правило, API поставляется в виде статической и динамической библиотеки.
- Статические библиотеки, в отличие от динамических, присоединяются (линкуются) к исполняемой программе в процессе сборки. Их использование наиболее предпочтительно, т.к. исключает возможность простой подмены файла. Далее будем рассматривать защиту приложений, использующих именно статическую библиотеку.



Обмен данными

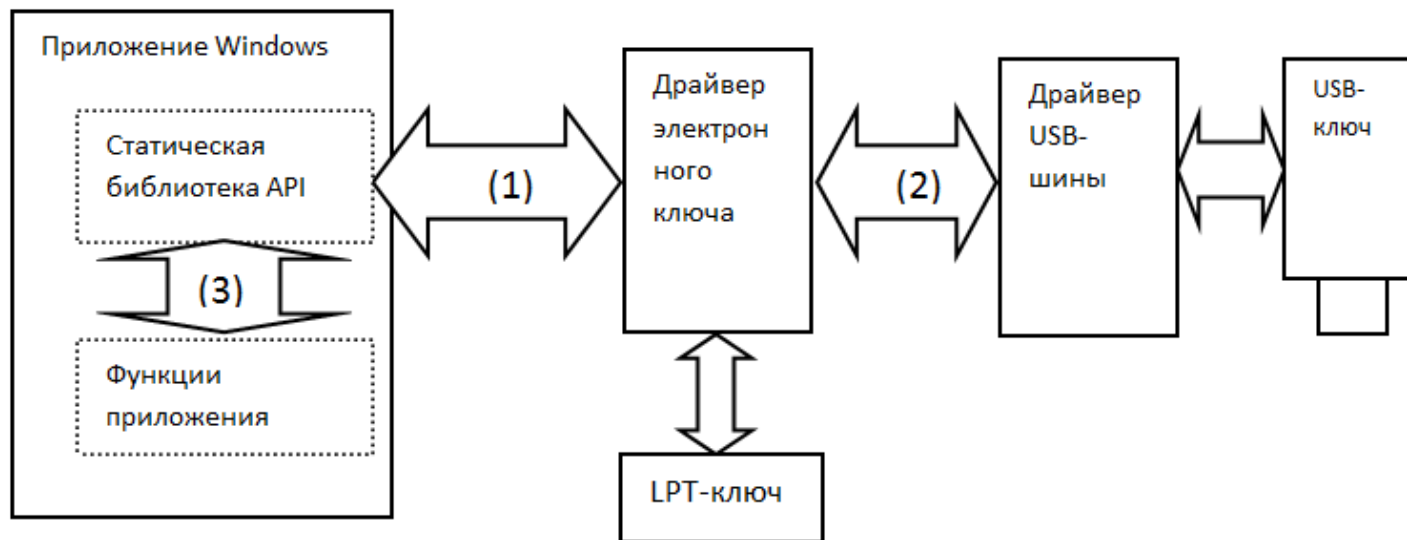


Рис.1

Обход защиты

Задача злоумышленника — заставить защищённую программу работать в условиях отсутствия легального ключа, подсоединённого к компьютеру

У злоумышленника есть следующие возможности:

- перехватывать все обращения к ключу
- протоколировать и анализировать эти обращения
- посылать запросы к ключу и получать на них ответы
- протоколировать и анализировать эти ответы
- посылать ответы от имени ключа и др

Такие широкие возможности противника можно объяснить тем, что он имеет доступ ко всем открытым интерфейсам, документации, драйверам и может их анализировать на практике с привлечением любых средств.

Атака на драйвер электронного ключа

Данный вид атаки является наиболее простым. Оригинальный драйвер электронного ключа заменяется на драйвер-эмулятор. Данные, передаваемые в ключ и возвращаемые обратно, перехватываются и сохраняются в файле на диске. Затем оригинальный ключ извлекается из компьютера и программа начинает взаимодействовать с эмулятором, продолжая искренне верить в то, что общается с ключом



Рис.2



Атака на драйвер USB-шины

К сожалению, полностью избавиться от программных эмуляторов на уровне USB-шины, используя ключи с симметричной криптографией, невозможно. Тем не менее, хорошая защита, построенная на постоянном обмене с электронным ключом, может потребовать не один день для записи всех возможных посылок и ответов к ключу и сработать у нелегального пользователя в самый неподходящий момент. Взломанные программы, перестающие работать по непонятным причинам, лишь тому подтверждение

Отдельно хочется упомянуть о защитах, когда разработчики ограничиваются простой проверкой наличия электронного ключа. Это грубая ошибка. Используя дизассемблер, “независимость” такой программе можно подарить за 15 минут

В электронных ключах с асимметричной криптографией обмен данными между защищённой программой и электронным ключом шифруется на сеансовых ключах. По этой причине единственно возможным является третий вариант атаки



Эмуляция ключа

- При эмуляции никакого воздействия на код программы не происходит, и эмулятор, если его удастся построить, просто повторяет все поведение реального ключа. Эмуляторы строятся на основе анализа перехваченных запросов приложения и ответов ключа на них. Они могут быть как табличными (содержать в себе все необходимые для работы программы ответы на запросы к электронному ключу), так и полными (полностью эмулируют работу ключа, так как взломщикам стал известен внутренний алгоритм работы)
- Построить полный эмулятор современного электронного ключа — это достаточно трудоёмкий процесс, требующий большого количества времени и существенных инвестиций. Информации о полной эмуляции современных ключей Guardant не встречалось



Исследование программ

Обратное проектирование
(англ. *reverse engineering*) —

процесс исследования и анализа машинного кода, нацеленный на понимание общих механизмов функционирования программы, а также на его перевод на более высокий уровень абстракции вплоть до восстановления текста программы на исходном языке программирования



Цели обратного проектирования

- Получение закрытых сведений, заложенных в программу
 - алгоритм работы программы
 - протоколы обмена данными
 - форматы данных
 - скрытые данные
- Обнаружение уязвимостей и недокументированных возможностей
- Модификация программы
 - отключение защитных механизмов
 - внедрение закладок
- Создание устройства или программы с аналогичными функциями



Задачи исследования

- **Восстановление кода программы** (или отдельных фрагментов) на языке программирования высокого уровня
 - Распаковка кода
 - Локализация нужного модуля
 - Идентификация кода и данных
- **Анализ алгоритма**
 - Определение структуры программы, назначение отдельных блоков
- **Изучение структур данных**



Декомпиляция

- Трансляция исполняемого модуля в эквивалентный исходный код на языке программирования высокого уровня

Частный случай: **дизассемблирование** - перевод исполняемого модуля программы на язык ассемблера

- Удачность декомпиляции зависит от:
 - правильности интерпретации кода
 - объема и структурированности декомпилированного кода

Дизассемблер

Дизассемблер – это транслятор, преобразующий машинный код в программу на языке ассемблера

- **Автоматические** -
 - генерируют готовый листинг, который можно затем править в текстовом редакторе
- **Интерактивные** -
 - позволяют изменять правила дизассемблирования в процессе работы



Выполнение в контролируемой среде

Отладка - динамическое исследование

- трассировка программы
 - пошаговое выполнение программы с остановками на каждой команде или строке
- отслеживание значений переменных в процессе выполнения программы
- контрольные точки и условия останова
- вмешательство в процесс выполнения



Анализ потоков данных

- Изучение входных и выходных данных
- Прослушивание каналов передачи данных, накопление статистики, анализ
 - мониторы событий
 - файловых операций
 - обращений к реестру
 - операций ввода-вывода
 - сетевые пакетные снифферы



Возможность защиты от исследования


- Команды однозначно интерпретируются процессором
- Данные могут быть перехвачены
- Среда исполнения может быть эмулирована

Выводы:

Защититься от исследования невозможно

Но можно усложнить задачу

- Защита от декомпиляции и отладки
- Обфускация алгоритма и данных



Применение различных методов исследования

- Метод «черного ящика»
 - анализ реакции программы на различные входные данные
- Метод «прозрачного ящика»
 - изучение восстановленного кода
- Метод «серого ящика»
 - частичное восстановление кода



Локализация модулей

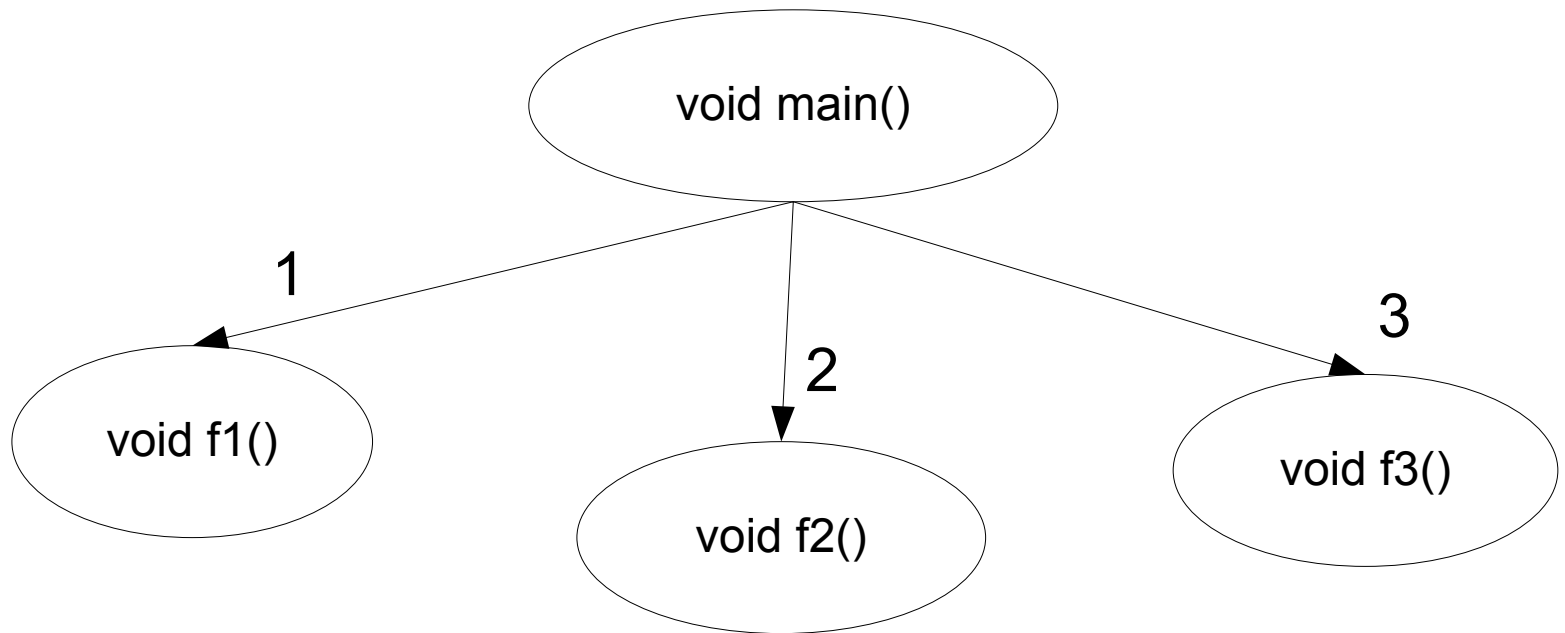
- Анализ изменений после ввода данных
 - Поиск в памяти введенных значений и контроль обращений программы к этим данным
 - Перехват вызовов функций ввода данных
- Ожидаемый вывод на экран
 - Поиск в памяти осмысленных последовательностей символов и контроль обращений программы к адресам, по которым хранятся эти последовательности
 - Перехват вызовов функций вывода данных

Обычный вызов функций

```
void f1(void)    {...};    // f1 ()
void f2(void)    {...};    // f2 ()
void f3(void)    {...};    // f3 ()

void main()
{
    f1 ();        // ВЫЗОВ f1
    f2 ();        // ВЫЗОВ f2
    f3 ();        // ВЫЗОВ f3
};
```

Дерево вызовов функций



Динамическое ветвление

```
typedef void (*fun) (void); // указатель на  
функцию
```

```
void f1(void)    {...};      // f1()
```

```
void f2(void)    {...};      // f2()
```

```
void f3(void)    {...};      // f3()
```

```
fun table[3] = {f1, f2, f3};
```

```
void main()
```

```
{
```

```
    for (int i=0; i<3; i++)
```

```
        (**table+i) ();
```

```
};
```

Использование переменной-флага

```
int flag;    // Глобальная переменная

int f1(void)  {... return 3 };    // f1()

int f2(void)  {... return 2 };    // f2()

int f3(void)  {... return 4 };    // f3()

void main() {

    flag = 1;

    while (flag!=4)

switch (flag) {

    case 1: flag = f1(); break;

    case 2:  flag = f2(); break;

    case 3:  flag = f3(); break;

}};
```

Потеря трассировочного прерывания

...

`pop SS;` Берем из стека SS. В режиме трассировки после этой команды `Int1` не вызывается

`push f;` помещаем в стек регистр PSW

`pop ax;` переписываем значение в `ax`

`test ax,0000000100000000b;` Проверка TF

`jnz tracing`

...

`tracing:`

Вычисления портящие данные

Использование стека

...

```
mov bp, sp
```

```
mov ax, 'a1'
```

```
mov [bp-2], ax
```

...

Участок, где явно не используется стек

...

```
cmp word ptr [bp-2], 'a1'
```

```
jne tracing
```

...

```
tracing:
```

Вычисления портящие данные