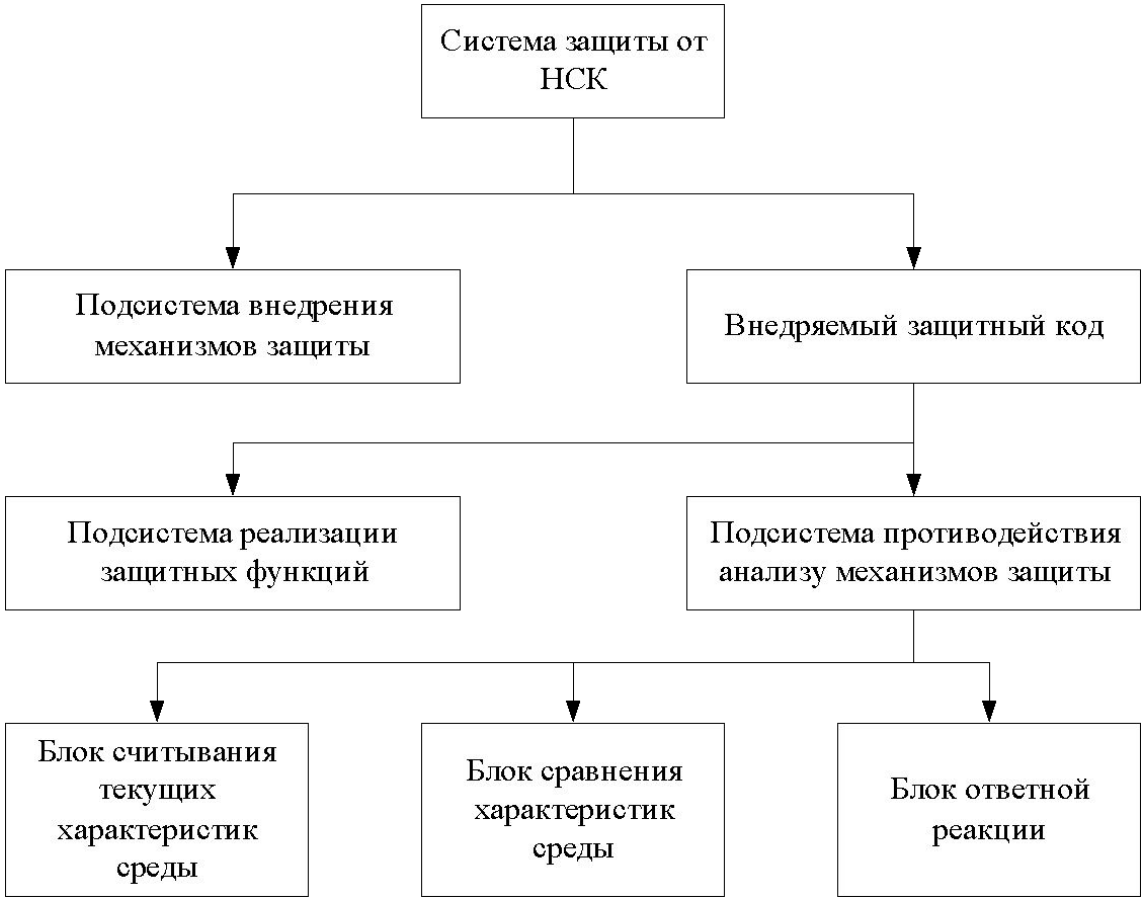
Изучение технических методов защиты от несанкционированного копирования. Изучение аппаратных методов защиты программ с помощью электронных ключей типа “HASP”

Ассистент кафедры БИТ Маро Е.А.

В общем случае **система защиты от НСК** представляет собой комплекс средств, предназначенный для затруднения (в идеале - предотвращения) нелегального копирования (исполнения) защищаемого программного модуля, с которым она ассоциирована.

Система защиты от НСК состоит из двух основных частей: подсистемы внедрения механизмов защиты и внедряемого защитного кода.



Системы защиты от НСК по способу ассоциации (внедрения) механизмов защиты можно подразделить на два типа:

1) встроенные – внедряются при создании программного продукта;
2) пристыковочные – подключаются к уже готовому программному продукту.

К преимуществам защит пристыковочного типа относятся:

- простота тиражирования программных систем защиты на объекты заказчика и разработчика;
- простота технологии применения - защита поставляется в виде законченного продукта, которым нужно обработать защищаемую программу;
- обеспечение в большинстве случаев достаточного уровня защищенности данных;
- сложность построения собственной встоенной системы, что приводит к значительному увеличению финансовых и временных затрат на создание конечного программного продукта при часто невысокой степени надежности полученной защиты (в силу непрофессионализма разработчиков).

Нейтрализация защитных механизмов может вестись двумя основными методами – **статическим и динамическим.**
Отдельно следует рассматривать эмуляцию аппаратных ключей.

Статический метод анализа программ

При статическом методе защищаемая программа сначала дизассемблируется. По полученному ассемблерному коду локализуются механизмы защиты, изучается логика их работы.

Динамический метод анализа программ

При динамической нейтрализации изучение и реконструирование логики работы защитных механизмов производятся с помощью отладчиков или эмуляторов, а необходимые изменения вносятся непосредственно в код программы.

Блок установки характеристик среды

В качестве идентифицирующего элемента, позволяющего отличать одну копию программного продукта от другой, могут выступать:

- компьютер (серийный номер винчестера, тип и версия BIOS, привязка программы к расположению на жестком диске);
- машинные носители (ключевые дискеты и компакт-диски).
- специальные аппаратные устройства (электронные ключи, брелки, смарт-карты);
- биометрические характеристики пользователя.

Блок сравнения характеристик среды

Блок сравнения значений характеристик среды должен отвечать ряду требований:

1. Установка значений характеристик среды защищаемой программы и, следовательно, сравнение значения характеристик с эталонными, должны производиться много раз.
2. Сравнение текущих значений характеристик с эталонными должно производиться периодически в течение всего сеанса работы программы.
3. Значения, полученные от блока установки характеристик среды, можно использовать как ключ для расшифрования кода, по которому (перед передачей ему управления) подсчитывается контрольная сумма.
4. Получение результатов сравнения должно быть принудительно распределено в коде.
5. Если код выработки результатов сравнения не может быть распределен, для подтверждения правильности передаваемых результатов должна также передаваться аутентифицирующая информация

Блок ответной реакции

Блок ответной реакции является одновременно самым простым и самым сложным при проектировании систем защиты от НСК. Если значения характеристик среды отличаются от эталонных, то при расшифровывании получается произвольный код, который при выполнении, как правило, "подвешивает" компьютер.

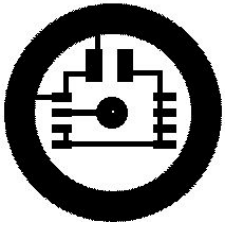
Стратегии защиты

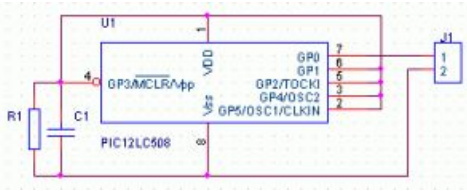
- Использование множественных вызовов
- Шифрование внешних и внутренних данных
- Отсутствие повторяющихся схем защиты
- Разделение шагов вызовов
- Шифрование памяти аппаратного ключа

Обзор типов электронных ключей

Ключи на основе FLASH-памяти

Ключи на основе PIC-котроллеров





Ключи на основе ASIC-чипов

Модели аппаратных ключей HASP HL

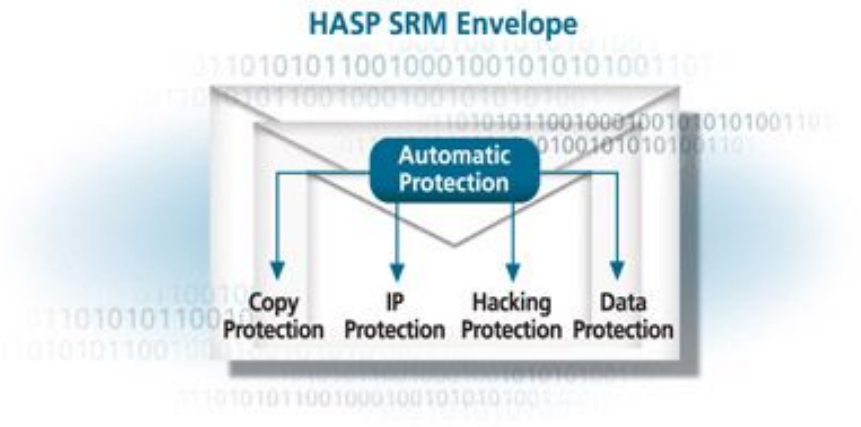
Модель ключа HASP HL	Размер памяти	Количество лицензий	Основные характеристики
HASP HL Basic	Нет	1	Шифрование/дешифрование
HASP HL Pro	112 байт	16	Шифрование/дешифрование HASP HL ID
HASP HL Max	4096 байт	112	Шифрование/дешифрование HASP HL ID
HASP HL Time	4096 байт	112*	Шифрование/дешифрование HASP HL ID Часы реального времени
HASP HL Net 10	4096 байт	112	Шифрование/дешифрование HASP HL ID Сетевой доступ
HASP HL Net 50	4096 байт	112	Шифрование/дешифрование HASP HL ID Сетевой доступ
HASP HL Net 250	4096 байт	112	Шифрование/дешифрование HASP HL ID Сетевой доступ

Защита данных с использованием HASP SRM

HASP SRM – это революционно новое средство защиты программного обеспечения от незаконного использования и распространения. Основное его задачей является ограничение доступа и функций защищенных приложений.

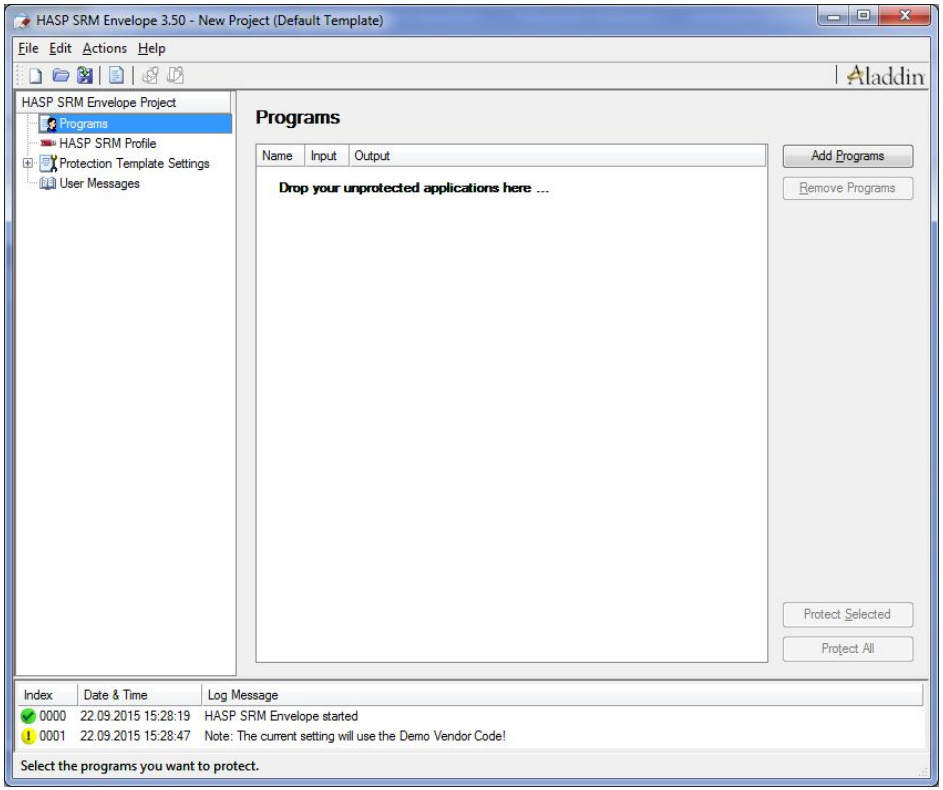
HASP SRM обеспечивает два основных метода защиты:

- HASP SRM Envelope



- HASP SRM Run-time API

HASP SRM Envelope



Средства HASP SRM Envelope позволяют создать защиту от отладки или обратного инжиниринга, а также предотвратить возможность раскрытия алгоритмов, используемых внутри защищенного программного обеспечения.

Назначение HASP Envelope

- Защита исполняемых файлов, DLL-библиотек и сборок .NET.
- Настройка параметров защиты.
- Ввод Кода Разработчика для проверки на наличие определенного ключа HASP HL.
- Настройка сообщений, которые будут отображаться для пользователей защищенного приложения.
- HASP Envelope обеспечивает не только строгое соответствие между приложением и защитным ключом, но также случайным образом генерирует несколько слоев для защиты

Поиск ключа HASP HL

- На локальном компьютере – защищенное приложение осуществляет поиск ключа только на локальном компьютере.
- В сети – защищенное приложение осуществляет поиск ключа только в сети.
- На локальном компьютере и в сети – защищенное приложение в первую очередь осуществляет поиск ключа на локальном компьютере, а затем в сети (по умолчанию).

Поведение защищенного приложения

- Частота случайных запросов к ключу со стороны защищенного приложения. При обмене данными между приложением и ключом используется шифрование.
- Временной интервал между проверками на присутствие требуемого ключа HASP SRM.
- Отказ или разрешение для программ, требующих оверлеев.
- Время, в течение которого защищенное приложение ожидает загрузки драйверов HASP SRM.

Свойства защиты

- Обнаружение отладочных средств на системном и на пользовательском уровне.
- Определение количества защитных слоев, создаваемых вокруг приложения. Диапазон возможных значений – 1-50 (по умолчанию – 12).
- Можно указать частоту обращений к ключу HASP HL для осуществления шифрования. Количество обращений устанавливается на полосе прокрутки Encryption Level (Уровень шифрования).

Шифрование и расшифрование с использование алгоритма AES

Встроенный крипто-процессор ключей HASP SRM осуществляет криптографические операции на основе алгоритма AES. При шифровании HASP SRM использует набор секретных 128-битных ключей, которые не покидают памяти ключа HASP SRM.

Защита с помощью HASP API

При защите мы должны проанализировать программный код приложения, в том числе на наличие ошибок. Все это сделать несколько раз, встроить в приложение, скомпилировать программный код с учетом вставок и связать его с объектным кодом HASP. Для усиления можно после этого использовать HASP SRM Envelop.

Основные группы сервисов:

HASP Login – устанавливает сеанс доступа к Компоненту и определяет контекст сессии.

HASP Get Info – применяется для запроса информации о системных компонентах в соответствии с заданными параметрами поиска.

HASP Encrypt – шифрует послание на подключенный ключ HASP 5 данные, используется совместно с сервисом HASP Decode Data, проверяет подсоединен ли ключ

HASP Decrypt – дешифруем данные, посылаемые на подключенный ключ.

Защита с использованием HASP SRM Run-time API

Порядок действий защиты с использованием HASP SRM Run-time API:

1. Вставить в код приложения вызов, устанавливающий соединение с ключом HASP SRM. Открытая сессия имеет свой собственный уникальный идентификатор.
2. После начала сессии для обмена данными с ключом могут использоваться и иные функции HASP SRM API.
3. После выполнения предыдущих действий вы получите сообщение о положительных результатах и возможных ошибках.
4. Выполнить действия, описанные в пунктах 2-4 для оставшейся части кода.
5. Скомпилируйте исходный код.

Обзор функций Hasp SRM Run-time API

hasp_datetime_to_hasptime()	Преобразование значений даты и времени в hasptime
hasp_decrypt()	Расшифрование данных из буфера с использованием AES
hasp_encrypt()	Шифрование данных в буфере с использованием AES
hasp_free()	Освобождение ресурсов из памяти
hasp_get_rtc()	Считывание текущего времени из ключа HASP HL Time или HASP HL Nettime
hasp_get_sessioninfo()	Получение данных об условиях сеанса
hasp_get_info()	Получение данных в соответствии с заданными параметрами поиска и представление их в заданном формате
hasp_get_size()	Извлечение данных об объеме файла в памяти
hasp_hasptime_to_datetime()	Преобразование значение времени с часов в формат даты и времени

Обзор функций Hasp SRM Run-time API

hasp_login()	Устанавливает сеанс доступа к Компоненту и определяет контекст сессии.
hasp_login_scope()	Запрос данных о доступе в соответствии с заданными параметрами
hasp_logout()	Завершение текущего сеанс
hasp_read()	Считывание данных на ключе HASP SRM
hasp_update()	Запись обновленной лицензии на ключ HASP SRM
hasp_write()	Запись данных в память ключа HASP SRM

hasp_get_info()

Описание

Данная функция применяется для запроса информации о системных компонентах в соответствии с заданными параметрами поиска, а также для представления этой информации в заданном формате.

Синтаксис

```
hasp_get_info(
char * scope,
char * format,
hasp_vendor_code_t vendor code,
char ** info
)
```

Hasp_login()

Описание

Устанавливает сеанс доступа к Компоненту и определяет контекст сессии.

Синтаксис

```
hasp_login(
hasp_feature_t feature_id,
hasp_vendor_code_t vendor_code,
hasp_handle_t * handle
)
```

hasp_logout()

Описание

Закрывает сессию или контекст.

Синтаксис

hasp_logout(
Hasp_handle_t handle
)

hasp_read()

Описание

Данная функция считывает данные из памяти ключа HASP SRM.

Синтаксис

hasp_read (
hasp_handle_t handle,
hasp_fileid_t fileid,
hasp_size_t offset,
hasp_size_t length,
void * buffer
)

hasp_write()

Описание

Функция hasp_write() используется для записи данных в память ключа HASP SRM.

Синтаксис

```
hasp_write(
hasp_handle_t handle,
hasp_fileid_t fileid,
hasp_size_t offset,
hasp_size_t length,
void * buffer
)
```

hasp_decrypt()

Описание

Расшифровывает данные в буфере с использованием алгоритма AES.

Синтаксис

```
hasp_decrypt(
hasp_handle_t handle,
void * buffer,
hasp_size_t length
)
```

hasp_encrypt()

Описание: Шифрует буфер с помощью алгоритма AES.

Синтаксис:

```
hasp_encrypt(
hasp_handle_t handle,
void * buffer,
hasp_size_t length
)
```

Сравнение Hasp SRM Envelop и Hasp SRM Run-time API

HASP SRM Envelope	HASP SRM Run-time API
Оперативная защита приложения в автоматическом режиме	Все обращения к API-функциям необходимо прописывать в исходном коде вручную
Определение особых параметров защиты ПО	Контролируемый и кропотливый процесс, обеспечивающий максимальную защиту. Степень защищенности зависит от объема функций API, реализованных при создании защиты.
Отсутствует необходимость редактирования исходного кода	Исходный код необходимо редактировать вручную
Антиотладочные механизмы и защита от обратного инжиниринга	Максимальная гибкость

Пример использования Hasp SRM Run-time API

```
#include "stdafx.h"
#include "hasp_api.h"
#include "hasp_vcode.h"

int _tmain(int argc, _TCHAR* argv[]) {
    hasp_handle_t handle = HASP_INVALID_HANDLE_VALUE;
    hasp_status_t status;
    const hasp_feature_t feature = HASP_DEFAULT_FID;
    TCHAR szText;

    ...

// Начало критичного участка кода
    status = hasp_login(feature, vendor_code, &handle);
    if(status != HASP_STATUS_OK) {
        sprintf(&szText, "Error #%d", status);
        MessageBox(NULL, &szText, "HASP SRM Login", MB_OK |
MB_ICONERROR);
        return status;
    }
```

Пример использования Hasp SRM Run-time API

```
// Критичные данные
    ...

    hasp_encrypt(handle, /*буфер, который необходимо зашифровать*/ &buff,
sizeof(buff));
    hasp_write(handle, HASP_FILEID_RW, 0, sizeof(buff), &buff);
    status = hasp_logout(handle);

    ...

// Часть кода, где необходимо использовать защищенный буфер
    status = hasp_login(feature, vendor_code, &handle);
    if(status != HASP_STATUS_OK) {
        sprintf(&szText, "Error #%d", status);
        MessageBox(NULL, &szText, "HASP SRM Login", MB_OK |
MB_ICONERROR);
        return status;
    }
    hasp_read(handle, HASP_FILEID_RO, 0, sizeof(buff), &buff);
    hasp_decrypt(handle, /*буфер, который необходимо расшифровать*/
&buff, sizeof(buff));
    status = hasp_logout(handle);

    ...
    return 0;
}
```