Guardant®

Система защиты от компьютерного пиратства

Эффективная
защита приложений

Урок 3.4: Запись маски в ключ с использованием Guardant API

Содержание

Программирование ключа при помощи Guardant API

В одном из предыдущих уроков был рассмотрен порядок действий, необходимых для записи маски в электронный ключ при помощи стандартной утилиты программирования ключей GrdUtil.exe.

Иногда этот способ программирования оказывается неудобным или неподходящим для разработчиков. Причины, по сути, две:

- Наличие «лишнего» функционала в стандартной утилите Guardant (в случае, когда утилита используется сотрудником, осуществляющим предпродажную подготовку ключа, наличие большого числа элементов управления «на виду» часто приводит к сложностям в освоении, или даже порождает ошибки)
- Потребность в дополнительной автоматизации процесса формирования и записи масок или его интеграции в другие системы

Тогда и возникает необходимость в написании собственной утилиты программирования ключей.

В данном уроке будет рассмотрен процесс формирования и записи маски в ключ Guardant Sign Net при помощи Guardant API. Подразумевается, что базовые навыки по работе с Guardant API уже получены (см. урок 3.1).

В качестве основы используем модифицированную версию библиотеки **writemask**, поставляемую в составе примеров по работе с Guardant API комплекта разработчика. Ее исходный код на языке C++, а также скомпилированная версия (для вызова из других языков программирования, в частности, Delphi) расположены в директории:

```
%DEV_KIT%\Samples\x86\Win32\Miscellaneous\WriteMask\Lib
```

Примечание

После ознакомления с данным уроком рекомендуется обратить внимание на пример **Dongle Trusted Remote Update**. Модуль **vendor** указанного примера содержит образец использования оригинальной версии библиотеки **writemask** (на языках Delphi и C++).

Используемые термины и обозначения

Маска ключа – образ памяти ключа (совокупность всех полей памяти и их содержимого), сохраненный в базе данных GrdUtil или в виде отдельного файла.

Защищенная ячейка – поле в ключе, защищенное аппаратными запретами на чтение и запись и обладающее определенной структурой. В защищенных ячейках хранятся данные, дескрипторы аппаратных алгоритмов, таблица лицензий.

Дескриптор алгоритма – разновидность защищенной ячейки, содержащая характеристики, режимы использования, а также определитель аппаратного алгоритма.

Определитель алгоритма – ключ шифрования, один из основных элементов аппаратного алгоритма.

Общее описание процесса записи маски

Стандартный порядок действий, необходимых для записи маски в электронный ключ Guardant, выглядит следующим образом:

- Поиск (**GrdFind**) и выбор нужного ключа (если ключей несколько)
- Формирование маски ключа (используем библиотеку **writemask**)
- Начало работы с ключом (**GrdLogin**)
- Инициализация памяти (**GrdTRU_SetKey**)
- Запись маски (**GrdWrite**)
- Установка запретов на операции с памятью ключа (**GrdProtect**)
- Завершение работы с ключом (**GrdLogout**)

Все действия производятся **локально** (то есть значение соответствующего параметра функций **GrdStartup()** и **GrdSetFindMode()** должно быть равно **GrdFMR_Local**).

Инициализация (очистка) памяти ключа выполняется при помощи функций **GrdTRU_SetKey()** и **GrdInit()**. Преимущество первой в том, что она позволяет задать ключ TRU, который необходим для дистанционного программирования ключа.

После инициализации маска записывается в ключ как обыкновенный дамп данных.

Результатом выполнения функции **GrdProtect()** является не только установка аппаратных запретов на запись и считывание памяти ключа (до момента очередной очистки памяти), но и детектирование самим ключом факта записи маски. Иными словами, до вызова **GrdProtect()** попытка обращения к дескрипторам алгоритмов в памяти ключа приведет к ошибке типа «Алгоритм не найден» (34, **GrdE_AlgoNotFound**).

Для формирования маски, как упоминалось выше, будет использоваться модифицированная библиотека **writemask**.

Библиотека writemask

В библиотеке реализован класс **CWritemask**, предоставляющий функционал формирования маски ключа с учетом модели.

Единственный **конструктор** класса принимает 4 параметра:

```
CWritemask(
            WORD        wDongleType,
            DWORD       dwDongleModel,
            BYTE        *pAlgos_Buffer,
            BYTE        *pAST_buffer);
```

В первые два (**wDongleType** и **dwDongleModel**) передаются тип и модель ключа. Эти значения могут быть получены из структуры **TGrdFindInfo** в процессе поиска ключей функцией **GrdFind()**. На их основе автоматически будет выбран формат маски, соответствующий указанному типу ключа.

В третий и четвертый параметры передаются указатели на заранее выделенные буферы памяти (в примере **buf_mask** и **buf_mask_header**, под тело маски и структуру ее заголовка, соответственно). Размер буферов – 4096 байт (так как объем энергонезависимой памяти ключа 4 Кб).

После объявления экземпляра класса **CWritemask**, необходимо добавить набор дескрипторов с нужными характеристиками и содержимым. Делается это при помощи основного метода класса – **AddAlgorithm()**, создающего в формируемой маске дескриптор и принимающего на вход информацию, которая должна быть в него помещена:

```
void   AddAlgorithm(
            WORD            wNumericName,
            BYTE            byLoFlags,
            WORD            dwHiFlags,
            BYTE            byAlgorithmCode,
            WORD            dwKeyLength,
            WORD            dwBlockLength,
            DWORD           dwActivatePwd,
            DWORD           dwDeactivatePwd,
            DWORD           dwReadPwd,
            DWORD           dwUpdatePwd,
            TGrdTime        *pBirthTime,
            TGrdTime        *pDeadTime,
            TGrdLifeTime    *pLifeTime,
            TGrdFlipTime    *pFlipTime,
            WORD            dwGP_Counter,
            WORD            dwErrorCounter,
            BYTE            *pDet);
```

Подробное описание полей дескриптора алгоритма находится в главе **Защищенные ячейки** 2-го тома Руководства пользователя Guardant. Объявления всех необходимых структур и констант могут быть найдены в заголовочном файле **structures.h**.

Вспомогательными методами класса являются **LMS_DET_Prepare** и **CODE_DET_Prepare** Они служат для формирования определителей защищенных ячеек типа **Таблица лицензий** и **Загружаемый код**, соответственно. Принцип работы с ними будет рассмотрен на примере.

Запись маски в ключ

Рассмотрим пример записи в ключ **Guardant Time Net** маски, состоящей из одного алгоритма типа ECC160 и таблицы лицензий (**sample.cpp**).

1. Блок инициализации GrdAPI и поиска ключей

Сначала вызывается блок функций инициализации Guardant API, включающий функцию поиска электронного ключа. Предположим, что ключ только один, и опустим обработку кодов возврата функций:

```
//--------------------------------------------------------------------------//

      nRet = GrdStartup(dwRemoteMode);
      hGrd = GrdCreateHandle(NULL, GrdCHM_SingleThread, NULL);
      nRet = GrdSetFindMode(hGrd, dwRemoteMode, dwFlags, wType, dwID,
                            wSN, byVer, wMask, wType, dwModel, dwInterface);
      nRet = GrdSetAccessCodes(hGrd, g_dwPublic, g_dwPrivRD, g_dwPrivWR,
                               g_dwPrivMS);
      nRet = GrdFind(hGrd, GrdF_First, &dwID, &FindInfo);
```

Ключ найден, информация о нем содержится в структуру **FindInfo**. Создаем экземпляр класса **Cwritemask**:

```
      CWritemask wm(FindInfo.wType, FindInfo.dwModel, buf_mask,
                    buf_mask_header);
```

2. Подготовка определителей алгоритмов и таблицы лицензий

Определителем для алгоритма ECC160 является секретный ключ ключевой пары. Он может быть сгенерирован при помощи утилиты программирования ключей **GrdUtil.exe**, либо специального инструмента **ECC160_Pair_Generator.exe**. В нашем примере он был объявлен ранее как байтовый массив.

Теперь создадим определитель для таблицы лицензий:

```
// Ресурс каждого модуля устанавливается равным максимальному ресурсу
найденного сетевого ключа
      wLANRes = FindInfo.byMaxNetRes;

// Общий ресурс и один дополнительный модуль
      pwModules[0] = wLANRes;
      pwModules[1] = wLANRes;

// Подготовка определителся таблицы лицензий
      wm.LMS_DET_Prepare(
            buf_mask_LMS_det,      // Allocated memory
            LMS_FLAG_LSIZE_2BYTES,// LMS flags: 2 bytes/record (otherwise 1)
            1,                // Number of extra records (in fact, 2 modules)
            wLANRes,               // LAN Resource
            pwModules,             // WORD array of module sizes
            &wLMS_DetSize);
```

Модуль таблицы LMS – структура, содержащая число лицензий того или иного компонента многомодульного приложения (номер модуля, к которому должно обращаться приложение, задается при выполнении функции **GrdLogin**).

В нулевом модуле таблицы LMS должен храниться **общий ресурс** ключа, поэтому фактически в таблицу лицензий помещается желаемое число модулей плюс один.

При использовании сетевых ключей высокой емкости (к примеру, с неограниченным ресурсом лицензий) и необходимости создания модулей в таблице LMS, больших 255, необходимо задать флаг **LMS_FLAG_LSIZE_2BYTES**. В результате, размер каждого модуля в таблице будет 2 байта (а не один).

Подготовка определителя загружаемого кода

В качестве дополнения приведем пример вызова метода, подготавливающего определитель защищенной ячейки типа **Загружаемый код** (для ключей Guardant Code/Code Time):

```
      if( FindInfo.wType & GrdDT_LoadableCode )
      {
// Подготовка определителя загружаемого кода
            wm.CODE_DET_Prepare(
                  buf_mask_CODE_det,     // Allocated memory
                  1, "0112",             // Sequence №, letter mask of Flash
                  USE_WHOLE_RAM,         // RAM start address
                  USE_WHOLE_RAM,         // RAM size
                  pPubKey4Verification,
                  pPrivKey4KeyDecryption,
                  &wCode_DetSize);       // Result determinant size
      }
```

Помимо указателей на выделенную память, открытый и секретный ключи двух ключевых пар (для верификации ЭЦП кода и его расшифрования соответственно) и переменную для возврата размера полученного определителя, передаются еще 4 параметра.

Два из них определяют расположение загружаемого кода во **Flash-памяти** ключа. Всего для загрузки кода доступны 4 сектора по 32Кб. Код может размещаться в любом множестве последовательно расположенных секторов (к примеру, при подготовке определителя для 2-го из 3-х планируемых загружаемых алгоритмов, можно задать «1, “0112”», в результате чего код займет второй и третий сектора).

Другие два параметра – начало и размер используемой **RAM памяти** – задаются обычным образом. Эти адреса могут пересекаться для алгоритмов, не выполняемых одновременно (которые не вызывают друг друга).

Необходимо отметить, что собственно **загружаемый код** не записывается в маску. После окончания записи маски он загружается во **Flash-память** посредством вызова функции **GrdCodeLoad()** и размещается по адресам, указанным в соответствующем дескрипторе.

3. Добавление дескрипторов в маску

Определители подготовлены. Осталось добавить набор дескрипторов в маску, указав их характеристики (в т. ч., и указатели на созданные определители):

```
// ECC160  0x0000
wm.AddAlgorithm(0x0000,
      nsafl_ST_III + nsafl_ActivationSrv + nsafl_UpdateSrv,
      nsafh_ReadSrv, rs_algo_ECC160, GrdADS_ECC160, GrdARS_ECC160,
      dwActivatePsw, 0, 0, dwUpdatePsw,
      NULL, NULL, NULL, NULL, 0, 0x000A, pDetECC);
// LMS          0x0002
wm.AddAlgorithm(0x0002,
      nsafl_ST_III,
      nsafh_ReadSrv, rs_algo_PI, wLMS_DetSize, 0, 0, 0, 0, 0,
      NULL, NULL, NULL, NULL, 0, 0x000A, buf_mask_LMS_det);
```

В современных ключах обращение к алгоритму происходит по его т. н. **числовому имени**, без привязки к фактическому расположению алгоритма в маске (в примере 0x0000 и 0x0002). Это позволяет более гибко формировать маску ключа.

Остальные параметры задают (по порядку):

Нижнее и верхнее поля флагов
Код (тип) создаваемого дескриптора
Размер определителя
Размер блока данных, преобразуемых с помощью алгоритма
4 пароля для сервисов (активации, деактивации, чтения и обновления)
Указатели на структуры временных зависимостей (Birth-, Dead-, Life- и Flip-time)
Счетчик GP (счетчик вызовов)
Счетчик ошибок ввода пароля при обращении к сервисам
Указатель на созданный ранее определитель алгоритма

4. Запись маски в ключ

Следующий этап – **запись маски** в ключ. При создании собственной маски этот блок можно и оставить без изменений, за исключением, может быть, заполнения структуры таймера ключа.

```
      printf("\nStart writing mask with number of algorithm's into
dongle");

      nRet = GrdLogin(hGrd, 0xFFFFFFFF, GrdLM_PerStation);
      nRet = GrdSetWorkMode(hGrd, GrdWM_UAM, GrdWMFM_DriverAuto);

      nRet = GrdInit(hGrd);

      nRet = GrdMakeSystemTime(hGrd, 2009, 5, 5, 8, 0, 0, 0, 0, &SysTime);
```

```
      nRet = GrdSetTime(hGrd, &SysTime, NULL);     // Set dongle time
      nRet = GrdWrite(hGrd, 0, GrdUAM_abAlgoAddr, buf_dongle_header, NULL);
      nRet = GrdWrite(hGrd, GrdUAM_abAlgoAddr, wm.wAST_Size,
buf_mask_header, NULL);
      nRet = GrdWrite(hGrd, GrdUAM_abAlgoAddr + wm.wAST_Size, wm.wMaskSize,
buf_mask, NULL);
```

При вызове функции **GrdProtect** (см. ниже) часть параметров рассчитывается автоматически, а другая часть должна задаваться вручную.

К автоматически рассчитываемым параметрам относятся аппаратные запреты на чтение/запись памяти ключа, а также количество дескрипторов, созданных в маске.

К параметрам, задаваемым вручную, – числовое имя дескриптора защищенной ячейки, в которую помещена таблица лицензий, а также **глобальные флаги** из набора:

GrdGF_ProtectTime	1	Блокировка вызова функции GrdSetTime
GrdGF_HID	2	Включение HID-режима работы ключа
GrdGF_OnlyOneSessKey	4	Единственный сессионный ключ для Guardant API. При установленном флаге будет работоспособна только одна копия приложения, защищенного Guardant API
GrdGF_2ndSessKey	8	Единственный сессионный ключ для автозащиты. При установленном флаге будет работоспособна только одна копия приложения, накрытого автозащитой

При задании флага **GrdGF_HID**, ключ перейдет в HID-режим при следующем подсоединении его к порту USB.

```
      nRet = GrdProtect(hGrd,
            GrdSAM_abyAlgoAddr + wm.wAST_Size + wm.wMaskSize,
            GrdSAM_abyAlgoAddr + wm.wAST_Size + wm.wMaskSize,
            wm.wNumberOfItems, 0x0002,  // LMS numeric name is 0x0002
            0,                          // Set global flags here
            NULL);

      if(nRet)
            printf("\nFailed writing algo mask into dongle");
      else
            printf("   ...   Done.");

      nRet = GrdLogout(hGrd, 0);
      nRet = GrdCloseHandle(hGrd);
      nRet = GrdCleanup();

//--------------------------------------------------------------------------//
```

Запись маски в электронный ключ завершена. Теперь обращения к алгоритмам и защищенным ячейкам будут проходить успешно.

Заключение

В рамках данного урока была раскрыта тема перехода от стандартной утилиты программирования ключа к собственной «прошивалке», основанной на Guardant API.

Процесс записи маски в ключ состоит из ряда последовательных этапов (формирование определителей для различных структур, добавление дескрипторов в маску и собственно запись маски в ключ), каждый из которых подробно описывается в уроке.

Дополнительные источники информации

При возникновении вопросов, на которые вам не удалось найти ответа в этом пособии, рекомендуем обратиться к следующим дополнительным источникам информации:

WWW: http://www.guardant.ru

Web-сайт разработчика содержит большой объем справочной информации об электронных ключах Guardant.

Служба технической поддержки:

e-mail: hotline@guardant.ru

тел. +7(495)925-77-90