

Guardant®

Система защиты от компьютерного пиратства

Эффективная защита приложений

Урок 5.1: знакомство с технологией загружаемого кода

Содержание

Общее описание технологии	3
Выбор загружаемого кода	4
Загрузка кода в ключ	5
Шаг 1. Выбор объекта и метода защиты.....	5
Шаг 2. Выбор кода для помещения в ключ	5
Шаг 3. Подготовка кода.....	6
Шаг 4. Сборка загружаемого кода	8
Шаг 5. Подготовка маски ключа.....	9
Шаг 6. Запись маски с загружаемым кодом в ключ	10
Шаг 7. Вызов загружаемого кода из приложения	11
Шаг 8. Установка параметров поиска ключа. Сборка примера	11
Контрольные испытания.....	13
Отладка загружаемого кода.....	14
Заключение	15
Дополнительные источники информации	16
WWW: http://www.guardant.ru	16
Служба технической поддержки:.....	16

Общее описание технологии

Технология загружаемого кода состоит в том, что в электронный ключ можно поместить собственный специально подготовленный код и, далее, в процессе работы приложения исполнять его. Стойкость защиты, при условии грамотного подхода к реализации, может быть очень высока.

Основным (но не единственным) **методом** использования технологии загружаемого кода является загрузка в электронный ключ уникального алгоритма, специально созданного разработчиком, на базе которого он может реализовать эффективную защиту приложения (к примеру, некоторого алгоритма принятия решений или преобразования данных), отсутствие которого ограничивает функциональные возможности приложения.

Данный подход, несмотря на свой потенциал, требует значительного внимания при выборе кода для загрузки в ключ и реализации механизмов защиты.

В рамках **данного урока** будут приведены некоторые рекомендации к выбору защищаемого участка кода и требования к загружаемому коду, а также на элементарном примере будут рассмотрены действия, необходимые для загрузки кода в ключ.

Выбор загружаемого кода

При выборе загружаемого кода рекомендуется следовать **требованиям к загружаемому коду**, изложенным во 2-й части Руководства пользователя:

- Требования **к безопасности** касаются сложности анализа загружаемого кода по принципу черного ящика, а также отсутствию его в предыдущих версиях защищаемого приложения
- Требования **к производительности** говорят об ограниченном быстродействии микропроцессора электронного ключа, а также интерфейса ключа с защищенным приложением
- Требования **к реализуемости** определяют наличие или отсутствие в коде отдельных конструкций и вызовов функций, пригодных для переноса в ключ

При разработке загружаемого кода имеют место следующие **ограничения**:

- Производительность микропроцессора ключа Guardant Code составляет порядка 1.25 DMIPS. Этого в подавляющем большинстве случаев достаточно для периодического выполнения криптографических операций, обработки или анализа небольших объемов данных.
- Наибольшая задержка происходит в момент обращения к электронному ключу, вследствие шифрования канала обмена. Т. о., интерфейс между драйвером и ключом является «узким местом» в процедуре обмена данными ключа и приложения.
- Максимальный объем загружаемого кода составляет 128 или 352 Кб (в зависимости от характеристик ключа)
- В оперативной памяти ключа можно сохранять до 20 Кб временных данных, используемых загружаемым кодом и сохраняемых между его вызовами (точный объем зависит от выбранного способа использования оперативной памяти электронного ключа загружаемым кодом)
- Данные, сохранение которых происходит на длительный срок, в т.ч. на время отсоединения ключа от USB порта, рекомендуется располагать в энергонезависимой памяти ключа в защищенных ячейках. Объем таких данных ограничивается только объемом EEPROM ключа, и может достигать 4 Кб.

В электронный ключ рекомендуется помещать код, без которого работа защищенного приложения невозможна, либо бессмысленна, при этом сложность анализа загружаемого кода по принципу черного ящика должна быть не ниже сложности написания аналогичного кода.

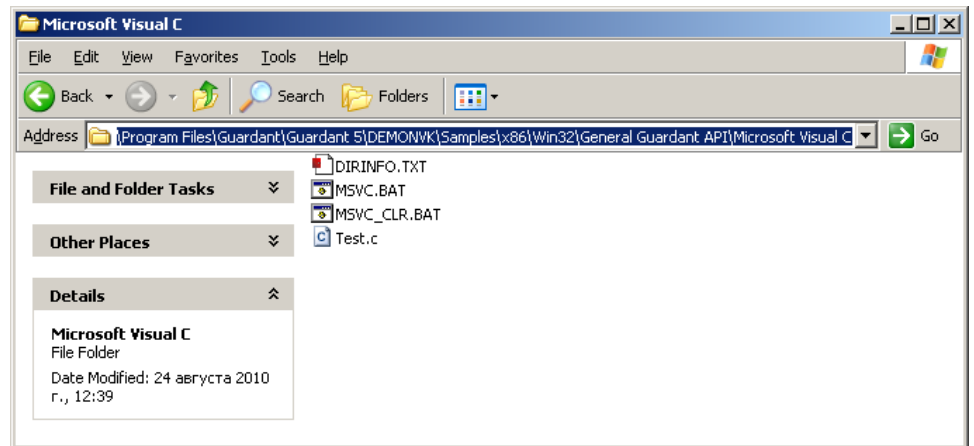
Стойкость загружаемого кода к анализу может быть значительно повышена при использовании криптографических алгоритмов и другой функциональности ключа. Для этого используется библиотека **Guardant Code API**.

Загрузка кода в ключ

Шаг 1. Выбор объекта и метода защиты

Первое на чем необходимо остановиться — это выбрать, какую функциональность приложения будем защищать с помощью загружаемого кода.

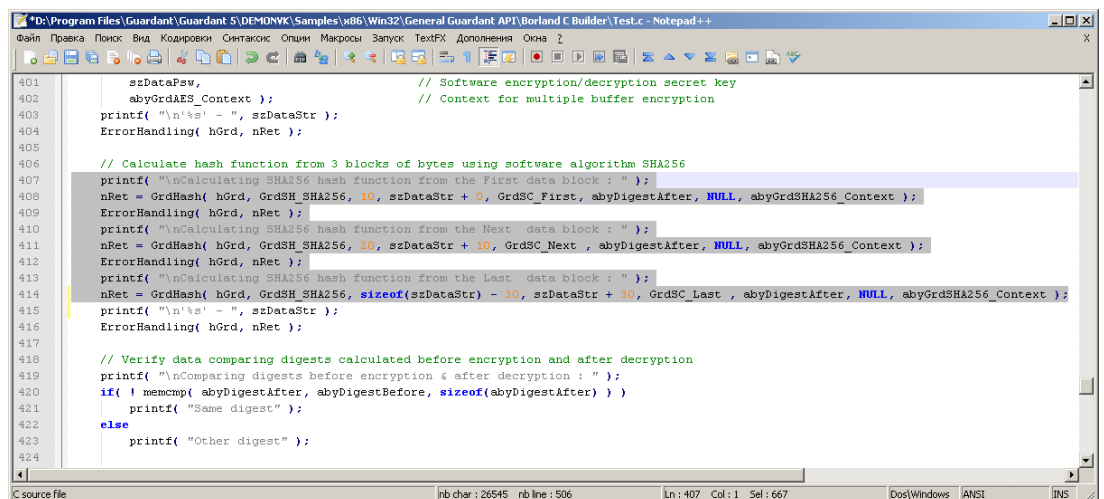
Рассмотрим один из стандартных примеров использования Guardant API - General Guardant API для платформы Microsoft Visual C. Он доступен в составе комплекта разработчика Guardant и находится в директории **...\\Samples\\x86\\Win32\\General Guardant API\\Microsoft Visual C:**



В **демонстрационных целях** реализуем в ключе и в защищаемом приложении один и тот же алгоритм преобразования данных.

Шаг 2. Выбор кода для помещения в ключ

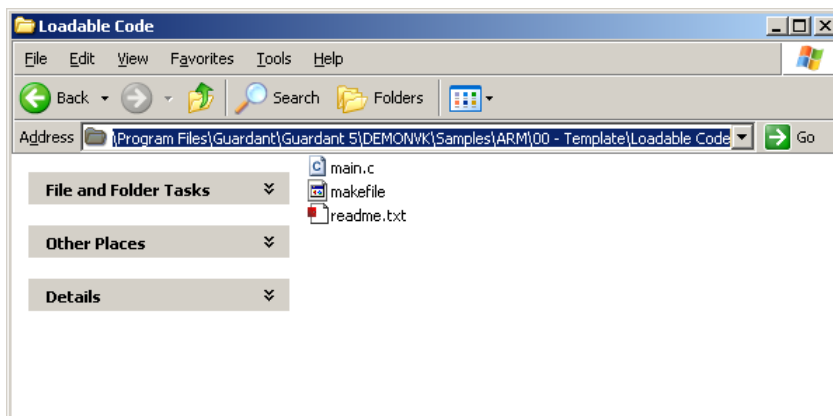
В примере содержится образец подсчета SHA256.



Предположим, что в защищаемое приложение подразумевает обмен зашифрованными сообщениями. Разработаем и загрузим в ключ тривиальный алгоритм получения сессионного ключа шифрования пользователя из его текущего пароля и случайного числа с использованием упомянутого алгоритма хеширования SHA256.

Шаг 3. Подготовка кода

В качестве шаблона проекта загружаемого кода будем использовать образец тривиального загружаемого кода из комплекта разработчика. Он находится в директории **...\Samples\ARM\00 - Template\Loadable Code**:



В тело функции **main** из буфера обмена необходимо поместить код, предназначенный для выполнения в ключе. Так как нам потребуется подсчет SHA256, возьмем образец вызова из защищаемого примера.

```
/**
 * @file
 * Loadable Code GCC Project template.
 *
 * Aktiv Co. / 2009
 */
#ifdef __ARM__
#include <stdint.h>
#define DBG(...)
#else
#include <stdio.h>
#include <windows.h>
#define DBG printf
#endif

#include "GrdAPI.h"
#include "GcaAPI.h"

// In case of unified In-Out buffer:
extern BYTE iodata[] ALIGNED;

// In case of parted In and Out buffers:
//extern BYTE idata[] ALIGNED;
//extern BYTE odata[] ALIGNED;

DWORD main(DWORD dwInDataLng, DWORD dwOutDataLng, DWORD dwP1)
{
    nRet = GrdHash( hGrd, GrdSH_SHA256, 10, szDataStr + 0, GrdSC_First, abyDigestAfter,
        NULL, abyGrdSHA256_Context );
    ErrorHandling( hGrd, nRet );
    return dwP1;
}
```

Далее, в код необходимо внести ряд изменений (ориентируясь, в числе прочего, на **требования по реализуемости**):

- Удалить вызовы внешних по отношению к загружаемому коду функций
- Заменить вызовы внешнего Guardant API на соответствующие им вызовы внутреннего Guardant Code API
- Изменить фактические параметры функций так, чтобы они обращались к внутренней памяти электронного ключа

Пусть пароль пользователя передается во входном буфере, размер строки определяется параметром dwP1. Получаемый сессионный ключ будем передавать в выходном буфере; его размер фиксирован и равен 32 байтам. **Случайное число**, используемое для генерации сессионного ключа, будем получать внутри ключа, при помощи вызова функции **GccaGetRandom()** внутреннего Guardant Code API.

В данном примере входной и выходной буферы совпадают, однако, при необходимости, они могут использоваться отдельно.

В результате загружаемый код должен приобрести следующий вид:

```
/**
 * @file
 * Loadable Code GCC Project template.
 *
 * Aktiv Co. / 2009
 */

#include <stdint.h>
#include "GrdAPI.h"
#include "GcaAPI.h"

// In case of unified In-Out buffer:
extern BYTE iodata[] ALIGNED;

DWORD main(DWORD dwInDataLng, DWORD dwOutDataLng, DWORD dwP1)
{
    BYTE pbyRand[4];

    GccaGetRandom(0, &pbyRand[0]);
    GccaGetRandom(0, &pbyRand[1]);
    GccaGetRandom(0, &pbyRand[2]);
    GccaGetRandom(0, &pbyRand[3]);

    memcpy(iodata+dwP1, pbyRand, 4);
    GccaHash(NULL, GrdSH_SHA256, dwP1+4, iodata, GrdSC_All, iodata, NULL, NULL);

    return *(DWORD*)pbyRand;
}
```

Полученное случайное число может использоваться в процедурах аутентификации или должно быть передано собеседнику любым другим способом (к примеру, вместе с зашифрованным сообщением).

В случае использования ключей **Guardant Code Time** вместо случайного числа целесообразно было бы использовать значение таймера ключа с точностью, к примеру, до нескольких минут. В этом случае его можно не передавать.

Аналогичным образом может быть реализована любая другая, более сложная схема генерации сессионного ключа, а также функциональность для аутентификации пользователя.

Преимуществом выбора для загрузки в ключ такого кода является отсутствие простых зависимостей между входным и выходным буферами и, как следствие, высокая сложность анализа кода по принципу черного ящика. Из **недостатков** можно выделить отсутствие привязки к определителям аппаратных алгоритмов, размещаемых в EEPROM ключа (таким образом, защита будет основана только на неизвестности использованного алгоритма преобразования данных, о котором, в свою очередь, можно догадаться, анализируя логику работы защищенного приложения).

Шаг 4. Сборка загружаемого кода

Перед компиляцией загружаемого кода необходимо выполнить настройку окружения. Для этого в командной строке из директории, содержащей исправленный образец загружаемого кода, командой **make template** запустим процесс подготовки шаблона:

```
D:\WINDOWS\system32\cmd.exe
D:\Program Files\Guardant\Guardant 5\DEMONU\K\Samples\ARM\00 - Template\Loadable Code>make template

----- begin -----
Clean Template...
rm -f *.out
rm -f *.lst
rm -f *.obj
rm -f *.dep
rm -f Startup.$
rm -f rom.ld
Making Startup.$...
Making rom.ld...
mkdir .dep
mkdir .out
mkdir .obj
mkdir .lst
Errors: none
----- end -----

Template has been successfully generated!

Please add your source files into project. Then customize source
parameters in makefile ($SRC in case C or CPPSRC in case C++ sources).

Your current entry point name is "main".

Entry point C prototype:
    int main(DWORD dwInDataLng, DWORD dwOutDataPl);
Entry point C++ prototype:
    int main(DWORD dwInDataLng, DWORD dwOutDataLng, DWORD dwPl);

D:\Program Files\Guardant\Guardant 5\DEMONU\K\Samples\ARM\00 - Template\Loadable Code>
```

Далее следует непосредственно сборка загружаемого кода командой **make**:

```

C:\WINDOWS\system32\cmd.exe
D:\Program Files\Guardant\Guardant 5\DEMONUJK\Samples\ARM\00 - Template\Loadable Code>make
----- begin -----
Compiling: main.o main.c
arm-elf-gcc -c -mcpu=arm7tdmi -I. -g -D_GCC -D_ARM -O2 -funsigned-char -funsigned-bitfields -fshort-enums -fpack-
-I./../../../../Include -Wpointer-arith -Wswitch -Wredundant-decls -Wreturn-type -Wshadow -Wunused -MD -MP -MF .dep/
ng-declarations -Wmissing-prototypes -Wnested-externs -Wa,-adhlns=.lst/main.lst -std=gnu99 main.c -o .obj/main.o
Assembling: Startup.S
arm-elf-gcc -c -mcpu=arm7tdmi -I. -Wa,-adhlns=.lst/Startup.lst,--gstabs Startup.S -o .obj/Startup.o
Linking: .out/SAMPLE.elf
arm-elf-gcc -mcpu=arm7tdmi -I. -g -D_GCC -D_ARM -O2 -funsigned-char -funsigned-bitfields -fshort-enums -fpack-
-I./../../../../Include -Wpointer-arith -Wswitch -Wredundant-decls -Wreturn-type -Wshadow -Wunused -MD -MP -MF .dep/elf
-fnostartfiles -Wl,-Map=.out/SAMPLE.map,--cref -lc -lm -lgcc -lrom.ld
Creating load file for Flash: .out/SAMPLE.hex
arm-elf-objcopy -O ihex .out/SAMPLE.elf .out/SAMPLE.hex
Creating binary: .out/SAMPLE.bin .out/SAMPLE.hex
./../../../../Bin/hex2bin.exe -b 0x00020000 -s 0x00008000 -o .out/SAMPLE.bin .out/SAMPLE.hex
./../../../../Bin/map_parse.exe .out/SAMPLE.map .out/SAMPLE.bin .out/SAMPLE.bmap
Loading Map File!
Position: -1
[GCC]
Records: 0, sizeof(SADDRTAB) = 8

Size:
.out/SAMPLE.elf :
section      size      addr
.text        420      131072
.rodata      16      131504
.bss         1040     1073754112
.stack       2048     1073755392
.stab        492      0
.stabstr     57      0
.comment     27      0
.debug_aranges 32      0
.debug_pubnames 27      0
.debug_info  271      0
.debug_abbrev 141      0
.debug_line  178      0
.debug_frame 48      0
.debug_str   268      0
.debug_loc   111      0
Total        5176

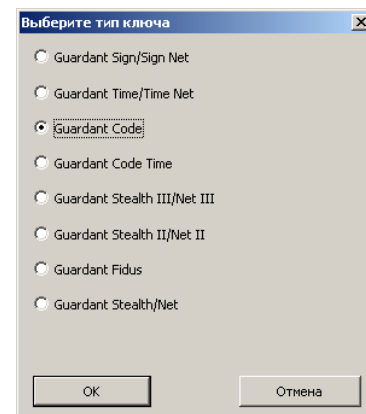
Errors: none
----- end -----
D:\Program Files\Guardant\Guardant 5\DEMONUJK\Samples\ARM\00 - Template\Loadable Code>

```

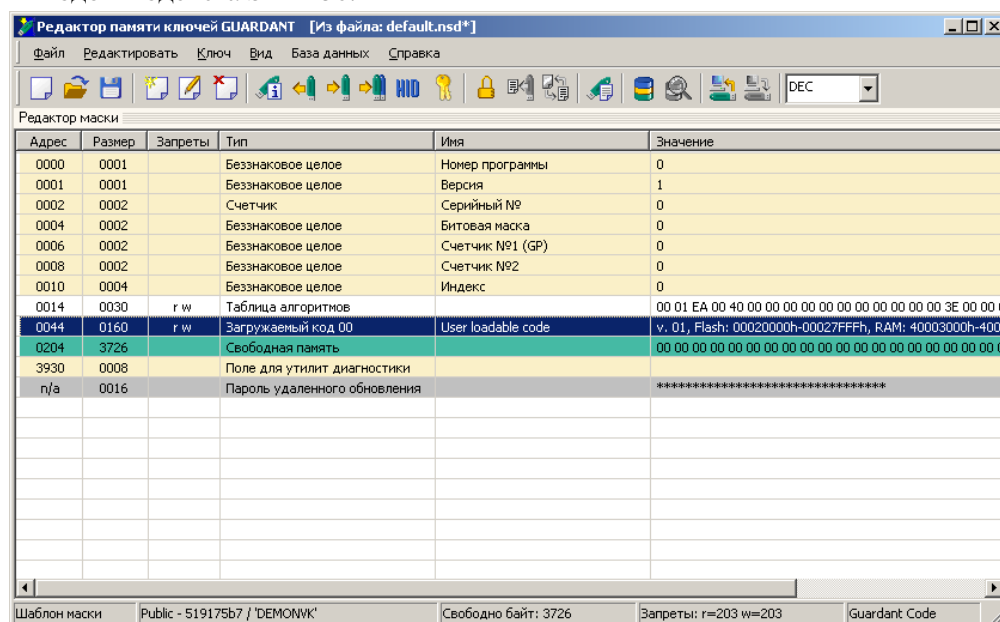

Теперь код скомпилирован и готов к загрузке в электронный ключ.

Шаг 5. Подготовка маски ключа

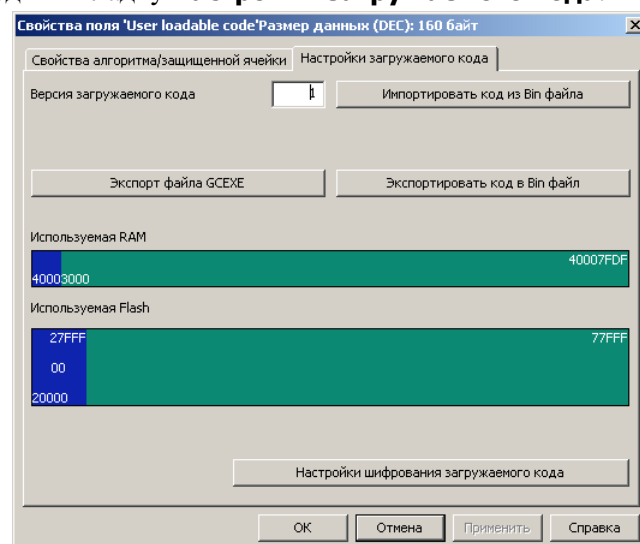
При помощи утилиты программирования ключей **GrdUtil.exe** необходимо создать **стандартную маску для ключа Guardant Code**:



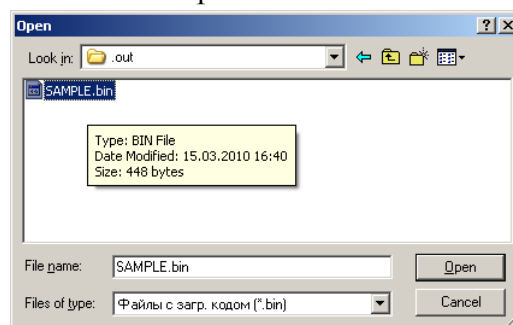
В ней уже содержится умолчательный код, реализующий перемножение двух чисел. Заменяем его нашим кодом подсчета SHA256:



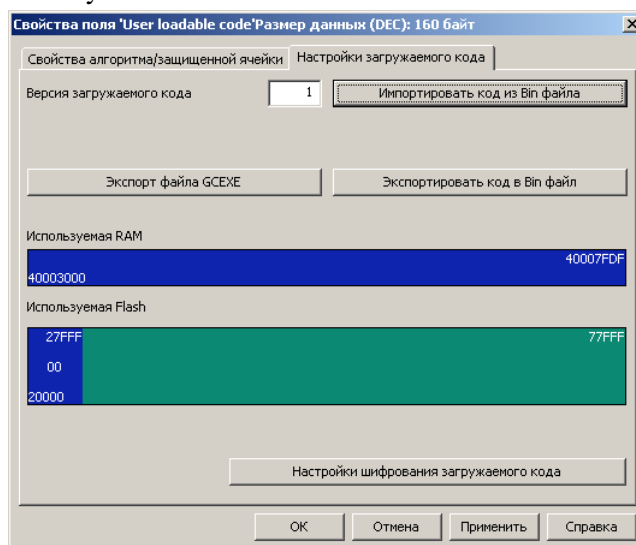
Для этого в параметрах дескриптора находим вкладку **Настройки загружаемого кода**:



..импортируем загружаемый код из созданного на Шаге 4 **bin**-файла:

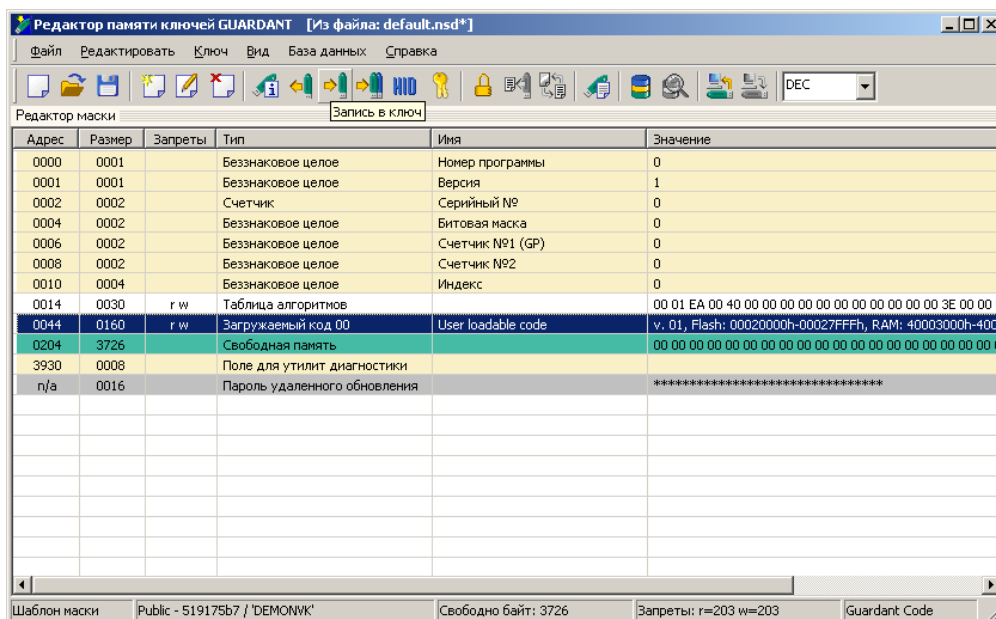


..закрываем окно свойств и нажимаем на кнопку **ОК**:



Шаг 6. Запись маски с загружаемым кодом в ключ

Для **записи маски** в ключ воспользуемся соответствующим элементом панели инструментов. При записи маски произойдет также загрузка кода, добавленного в маску:



Шаг 7. Вызов загружаемого кода из приложения

Вернемся к защищаемому приложению (напомним, что это стандартный пример Guardant General API для Microsoft Visual C). Для наглядности оставим в примере код вызова только необходимых в данном случае функций (т. е., код блока инициализации, логина в ключ и подсчета SHA256).

Добавим в пример вызов загружаемого кода — т.е., функцию **GrdCodeRun()** с нужными параметрами (подробнее см. документацию по Guardant API):

```
// Calculate hash from username and random using Guardant Code dongle
memset(szDataStr, sizeof(szDataStr), 0);
strcpy(szDataStr, "User");

printf( "\nCalculate hash from username and random using Guardant Code dongle: " );
nRet = GrdCodeRun( hGrd, 0x0000, strlen(szDataStr), &dwRand, 32, abyDigestAfter,
                  strlen(szDataStr), szDataStr, NULL);
ErrorHandling( hGrd, nRet );
printf( "\nSession key obtained: %s", abyDigestAfter );

// Calculate hash from username and random without dongle
memset(szDataStr, sizeof(szDataStr), 0);
strcpy(szDataStr, "User");
dwLength = strlen(szDataStr);
memcpy(szDataStr+dwLength, (BYTE*)&dwRand, 4);

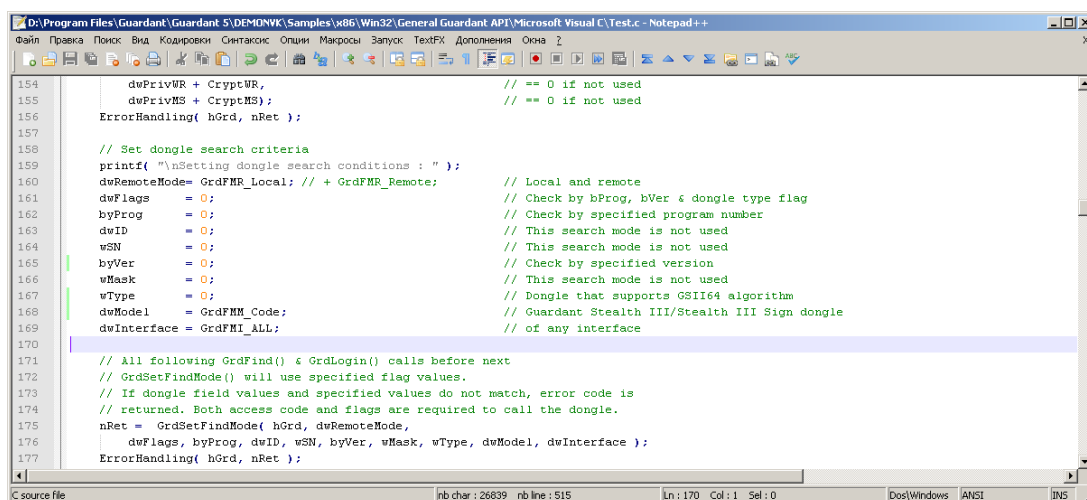
printf( "\nCalculate hash from username and random without dongle: " );
nRet = GrdHash( hGrd, GrdSH_SHA256, dwLength+4, szDataStr, GrdSC_All, abyDigestAfter,
               NULL, abyGrdSHA256_Context );
ErrorHandling( hGrd, nRet );
printf( "\nSession key obtained: %s", abyDigestAfter );

printf( "\nRandom used: 0x%x", dwRand );
```

Для проверки корректности работы примера следом расположим аналогичный алгоритм получения сессионного ключа. Он использует то же самое случайное число (сгенерированное внутри электронного ключа).

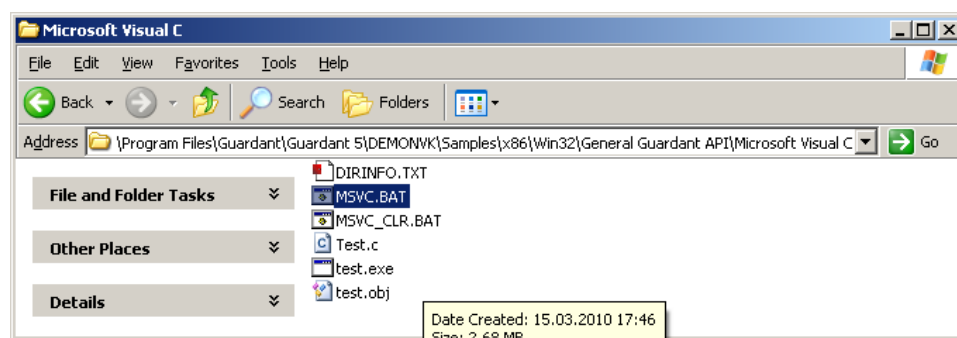
Шаг 8. Установка параметров поиска ключа. Сборка примера

Чтобы при старте приложения ключ Guardant Code с загруженным кодом был найден и корректно инициализирован, в коде примера необходимо указать параметры поиска ключа, а также коды доступа (если они отличны от демонстрационных):



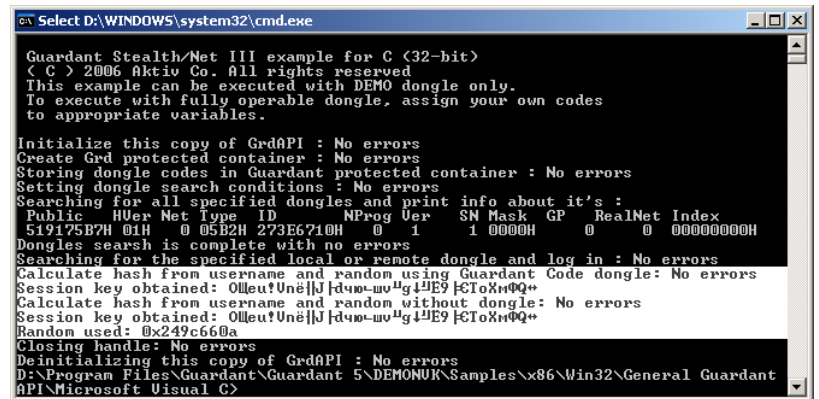
```
154     dwPrivVR + CryptVR, // == 0 if not used
155     dwPrivMS + CryptMS); // == 0 if not used
156     ErrorHandling( hGrd, nRet );
157
158     // Set dongle search criteria
159     printf( "\nSetting dongle search conditions: " );
160     dwRemoteMode= GrdFMR_Local; // + GrdFMR_Remote; // Local and remote
161     dwFlags = 0; // Check by bProg, bVer & dongle type flag
162     byProg = 0; // Check by specified program number
163     dwID = 0; // This search mode is not used
164     wSN = 0; // This search mode is not used
165     byVer = 0; // Check by specified version
166     wMask = 0; // This search mode is not used
167     wType = 0; // Dongle that supports GSII64 algorithm
168     dwModel = GrdFMI_Code; // Guardant Stealth III/Stealth III Sign dongle
169     dwInterface = GrdFMI_ALL; // of any interface
170
171     // All following GrdFind() & GrdLogin() calls before next
172     // GrdSetFindMode() will use specified flag values.
173     // If dongle field values and specified values do not match, error code is
174     // returned. Both access code and flags are required to call the dongle.
175     nRet = GrdSetFindMode( hGrd, dwRemoteMode,
176                          dwFlags, byProg, dwID, wSN, byVer, wMask, wType, dwModel, dwInterface );
177     ErrorHandling( hGrd, nRet );
```

Сборка примера может производиться при помощи файла **msvc.bat** (требуется установленная MS Visual Studio 2Kx), или в любой другой среде разработки:



Контрольные испытания

Производим **контрольный вызов**. Легко видеть, что подсчитанное внутри ключа значение SHA256 корректно:



```
Select D:\WINDOWS\system32\cmd.exe

Guardant Stealth/Net III example for C (32-bit)
< C > 2006 Aktiv Co. All rights reserved
This example can be executed with DEMO dongle only.
To execute with fully operable dongle, assign your own codes
to appropriate variables.

Initialize this copy of GrdAPI : No errors
Create Grd protected container : No errors
Storing dongle codes in Guardant protected container : No errors
Setting dongle search conditions : No errors
Searching for all specified dongles and print info about it's :
Public  HUser Net Type  ID      NProg Ver  SN Mask GP  RealNet Index
519175B7H 01H      0 05B2H 273E6710H  0  1    1 0000H  0  0 00000000H
Dongles search is complete with no errors
Searching for the specified local or remote dongle and log in : No errors
Calculate hash from username and random using Guardant Code dongle: No errors
Session key obtained: 0lleu!Unë|J |dчю-wvµg4JE9 |çToXнФQ»
Calculate hash from username and random without dongle: No errors
Session key obtained: 0lleu!Unë|J |dчю-wvµg4JE9 |çToXнФQ»
Random used: 0x249c660a
Closing handle: No errors
Deinitializing this copy of GrdAPI : No errors
D:\Program Files\Guardant\Guardant 5\DEMONU\K\Samples\x86\Win32\General Guardant
API\Microsoft Visual C>
```

Отладка загружаемого кода

Отладка защищаемого приложения может производиться в среде разработки. Для этого в электронный ключ рекомендуется записать рабочую маску (для корректного выполнения обращений Guardant API из загружаемого кода). Помещение в ключ разрабатываемого загружаемого кода не требуется. Вместо него загружается **отладочный модуль**. В процессе отладки используется библиотека **GcaAPI.dll**.

Отладка происходит следующим образом:

- В ключ записывается **рабочая маска**, на место рабочего загружаемого кода в дескриптор типа Loadable Code помещается **отладочный модуль** (он загружается по-умолчанию, при создании очередного дескриптора типа Loadable Code)
- В настройках проекта загружаемого кода устанавливается использование библиотеки **GcaAPI.dll**: при компиляции использовать файл **GcaAPI.h** (вместо GcaAPI.dll.h), в процессе линковки - **GcaAPI.lib**
- На входе в функцию **main** загружаемого кода добавить стандартный **блок инициализации** ключа (инициализацию Guardant API и логин в ключ), содержащий вызовы следующих функций: GrdStartup(), GrdCreateHandle(), GrdSetAccessCodes(), GrdLogin()
- Добавить после блока инициализации Guardant API вызов макроса инициализации отладочной библиотеки **DEBUGDLL_INIT(x,y)**: в качестве первого параметра передается созданный в результате вызова функции GrdCreateHandle() хендл, второй должен быть равен числовому имени дескриптора Loadable Code в маске ключа с **отладочным модулем**

В результате указанных действий при обращении к функциям внутреннего Guardant Code API вызовы будут автоматически перенаправлены в ключ при помощи отладочной библиотеки и отладочного модуля.

После завершения отладки загружаемого кода отладочный модуль в маске ключа должен быть заменен загружаемым кодом. Для этого необходимо открыть рабочую маску при помощи утилиты программирования ключей GrdUtil (к примеру, найти ее в БД прошивок) и импортировать скомпилированный загружаемый код, указав путь к соответствующему файлу .BIN в параметрах дескриптора типа Loadable Code.

Заключение

В данном уроке был рассмотрен набор требований и ограничений, имеющих место при работе с загружаемым кодом. Также был приведен порядок действий, необходимых для создания, отладки и загрузки кода в электронный ключ Guardant Code.

Возможность исполнения собственного кода на доверенном отчуждаемом устройстве поднимает уровень защиты ПО от анализа, несанкционированного использования, анализа и модификации на качественно новый уровень — конечно, при условии соблюдения требований, изложенных в данном уроке и документации Guardant.

Стоит также понимать, что описанная технология не подразумевает извлечение части кода приложения и перенос его в ключ «как есть». Такой подход, во-первых, в «чистом» виде невозможен, так как исполняемый в ключе **Guardant Code** код не может иметь внешних зависимостей. Во-вторых, в подавляющем большинстве случаев, он просто не имеет смысла из-за того, что извлеченный участок кода приложения часто не удовлетворяет требованиям по безопасности (в т.ч. по сложности его анализа) и не может обеспечить стойкость защиты приложения в целом.

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:

е-mail: hotline@guardant.ru

тел. +7(495)925-77-90