Практический курс «Основы защиты приложений при помощи Guardant API»

Урок 2
Аппаратные алгоритмы ключа (Delphi)

Содержание

1. Постановка задачи и планирование ее решения

Тестовое приложение

В качестве тестового приложения выбрана программа, представляющая собой графический интерфейс работы с БД MS Access, содержащей список клиентов и информации о них.

Постановка задачи

1. Зашифровать конфигурационный файл на алгоритме GSII64.
2. Создать механизм проверки электронной цифровой подписи полезных данных программы в случайные моменты времени.
3. Построить таблицу из 1000 случайных вопросов и ответов на «однонаправленном» алгоритме AES128 и проверять их по срабатыванию определенных триггеров.
4. Построить таблицу вопросов и ответов для данных приложения на «однонаправленном» алгоритме AES128, связывающих телефонный код города с названием города.
5. Разработать и внедрить несколько способов обработки ошибок функций Guardant API.

Действия, необходимые для решения задачи

Написать программу для шифрования случайных данных на алгоритме AES128

1. Создать 1000 случайных строк.
2. Зашифровать строки на однонаправленном алгоритме AES128 *(GrdCrypt).*
3. Подготовить полученные данные для внедрения в код программы.

Написать программу для шифрования полезных данных на алгоритме AES128

1. Создать таблицу соответствий телефонного кода города и его названия.
2. Зашифровать названия городов на однонаправленном алгоритме AES128 *(GrdCrypt).*
3. Подготовить полученные данные для внедрения в код программы.

При установке фильтра

1. Вычислить случайное число, используя Microsoft CryptoAPI *(CryptGenRandom).*
2. Получить имя случайного клиента из БД.
3. Создать цифровую подпись на основе имени клиента из БД *(GrdSign).*

При отключении фильтра

1. Получить случайное число с помощью Guardant API *(GrdTransform).*
2. Проверить, вызывался ли до этого алгоритм цифровой подписи *(GrdCRC).*
3. Проверить цифровую подпись *(GrdVerifySign)*.

При добавлении новой записи в БД

1. Получить по коду города зашифрованное название города.
2. Расшифровать название города с помощью алгоритма AES128 (*GrdCrypt*).

3. Выполнить обработку ошибок выполнения функции *GrdCrypt* нестандартным образом (try..except).

<u>При удалении записи из БД</u>

1. Вычислить случайное число, используя Microsoft CryptoAPI *(CryptGenRandom).*
2. Получить одну из 1000 строк, зашифрованных на алгоритме AES128.
3. Расшифровать строку *(GrdCrypt).*
4. Выполнить сравнение исходной строки и расшифрованной в SQL запросе.

2. Описание решения задачи

2.1 Введение

В прошлом уроке были изучены базовые функции для работы с ключами Guardant. Рассмотрены способы поиска ключа, подключения к ключу, выполнения операции логина, разработан механизм защиты программы с помощью записи информации в защищенную ячейку, с последующим ее чтением и проверкой. Однако основным преимуществом современных электронных ключей Guardant является возможность преобразовывать данные при помощи криптографических алгоритмов внутри ключа. Данные алгоритмы позволяют шифровать и подписывать информацию, а так же хешировать и контролировать результаты.

Аппаратные алгоритмы являются основой противодействия созданию эмуляторов электронного ключа (один из наиболее часто встречаемых способов взлома защиты с помощью ключей, особенно в случаях, когда в защите используются только функции чтения и записи данных). Только грамотное использование аппаратных алгоритмов в программе способно значительно затруднить взломщику анализ защиты и создание программного аналога электронного ключа.

В данном уроке будут продемонстрированы основы работы с различными аппаратными алгоритмами и показаны некоторые методы защиты программного обеспечения от взлома.

2.2 Основы работы с аппаратными алгоритмами

Для начала работы с аппаратными алгоритмами необходимо записать в ключ новую маску, она находится в файле «*lesson2.nsd*». В данной маске содержится описание алгоритмов GSII64, RAND64, ECC160, AES128 Encode, AES128 Decode, которые будут использованы для усиления защиты тестового приложения.

Работа с аппаратными алгоритмами осуществляется с помощью функций **GrdCrypt** (GrdCryptEx), **GrdTransform** (GrdTransformEx), **GrdHash** (GrdHashEx), **GrdSign** и **GrdVerifySign**.

Все функции, работающие с аппаратными алгоритмами, при обращении к алгоритму используют его номер в маске ключа, так, например, в маске «*lesson2.nsd*» алгоритм ECC160 имеет номер 03.

2.3 Простое шифрование на алгоритме GSII64

Одни из способов усиления защиты – шифрование части данных, необходимых для корректной работы программы. Это не позволит злоумышленнику просто вырезать функции защиты.

В тестовом приложении реализовано шифрование строки соединения с базой данных, которая хранится в файле конфигурации при помощи блочного алгоритма GSII64. Данный алгоритм устойчив к криптоанализу. Он поддерживается ключами Guardant Stealth II, Stealth III и Sign/Time. Алгоритм GSII64 является симметричным, что позволяет использовать его для шифрования/расшифрования данных.

Для шифрования данных на алгоритме GSII64 используется функция **GrdCrypt**. Ее прототип с описанием параметров приведен ниже.

При шифровании данных с использованием аппаратных алгоритмов используется несколько ключевых понятий.

Определитель алгоритма – уникальная последовательность байт, задаваемая в маске ключа (в дескрипторе соответствующего алгоритма), используемая при шифровании, обеспечивающая уникальность преобразования данных.

Вектор инициализации – последовательность байт, используемая при шифровании, для связки последовательно шифруемых блоков данных. Может использоваться в качестве «соли», то есть, чтобы «разнообразить» шифрование одних и тех же данных одним и тем же ключом. Задается непосредственно при вызове алгоритма. Не является секретным. При шифровании и дешифровании одного блока данных вектор инициализации должен иметь одинаковые значения.

Метод преобразования – режим работы алгоритма шифрования (шифрование/расшифрование, ECB/CBC/OFB/CFB). Определяется комбинацией флагов, задаваемых при вызове алгоритма.

При шифровании на алгоритме GSII64 будет использоваться преобразование в режиме OFB. Он позволяет шифровать данные произвольной длины (то есть использовать блочный шифр, как потоковый). Более подробно о методах преобразования можно прочитать в справке по ключам защиты Guardant.

Прототип функции **GrdCrypt**

```
int GRD_API GrdCrypt(
  HANDLE hGrd,              // Хэндл контейнера Guardant
  DWORD dwAlgo,             // Номер аппаратного алгоритма
  DWORD dwDataLng,      // Длина данных для преобразования
  void *pData,              // Указатель на буфер с данными
  DWORD dwMethod,           // Комбинация флагов, определяющая
                            метод преобразования
  void *pIV,                // Вектор инициализации
  void *pKeyBuf,            // Буфер для передачи ключа шифрования,
                            только для программно-реализованных
                            алгоритмов
```

```
        void *pContext                  // Буфер контекста, только для программно-
                                        реализованных алгоритмов
    );
```

Теперь мы готовы перейти к внедрению защиты программы. Для этого в обработку события *Form1_Load* необходимо добавить следующий код.

```
szInitVectorGS2:=IntToHex(InitVector,8);    { Вектор инициализации для алгоритма
                                              GSII64, 8 сиволов }
nRet:=GrdCrypt(hGrd,                        { Хэндл контейнера Guardant }
02,                                         { Номер аппаратного алгоритма }
Length(BDPath),                             { Длина строки для шифрования }
@BDPath[1],                                 { Указатель на строку подключения к БД}
GrdAM_OFB + GrdAM_Encode,                   { Метод преобразования }
@szInitVectorGS2[1],                        { 64-битный вектор инициализации }
nil,                                        { Не используется }
nil );                                      { Не используется }
```

В данном случае в качестве вектора инициализации используется заранее определенное число (его можно увидеть в разделе описания констант). В реальном приложении рекомендуется использовать разный вектор инициализации между циклами шифрования-расшифрования.

Метод преобразования составлен из двух флагов: GrdAM_OFB – определяет выбор шифрования по методу OFB, о нем говорилось выше, и GrdAM_Encode – указывающий, что на данном этапе происходит шифрование данных.

После шифрования строка представляет собой набор байт из печатаемых и непечатаемых символов. Чтобы корректно записать ее в файл конфигурации, преобразуем в кодировку BASE64.

Зашифрованная строка, указывающая путь к базе данных, получена и записана в файл конфигурации. Теперь можно добавить в процедуру обработки события «*Form1.Create»* функцию дешифрования данных. По сути, функция расшифрования аналогична своей предшественнице за исключением одного флага в методе преобразования. Найдите ее в программном коде и определите, что в ней изменилось.

2.4 Использование ЭПЦ (электронной цифровой подписи)

Электронная цифровая подпись – это одно из замечательных нововведений в ключах защиты Guardant. Данный алгоритм позволяет подписывать данные на секретном ключе, который хранится в дескрипторе аппаратного алгоритма, и проверять подпись программными методами. Это позволяет уйти от классической защиты алгоритмами типа вопрос-ответ и повысить ее эффективность.

В тестовом приложении разработана следующая схема защиты:

1. При нажатии кнопки включения фильтра с вероятностью 20% генерируется цифровая подпись, в качестве данных для подписи используется случайно взятое имя клиента из базы данных.

2. При нажатии кнопки отключения фильтра с вероятностью 20% происходит проверка сгенерированной цифровой подписи.

Код с функционалом защиты начинается с вызова функции генерации случайного числа. Большинство экспертов в области защиты сходятся во мнении, что использование стандартного функционала языка программирования для генерации случайных чисел небезопасно. В качестве алгоритма случайных чисел рекомендуется функция *CryptGenRandom* из *Microsoft Crypto API*. Для ее использования достаточно подключить к программе модуль «**wcrypt2**».

Более подробно об инициализации и использовании *Microsoft Crypto API* можно узнать из MSDN.

После получения имени случайного клиента, генерируем цифровую подпись. Это делается с помощью функции **GrdSign**. В ключах Guardant реализован алгоритм цифровой подпись ECC160, который использует математический аппарат эллиптических кривых. При использовании данного алгоритма длина данных для подписи должна составлять 20 байт, при этом длина самой цифровой подписи составляет 40 байт.

```
nRet:= GrdSign( hGrd,          // Хэндл контейнера Guardant
03,                            // Номер аппаратного алгоритма
20,                            // Длина данных для подписи
@DataForSign[1],               // Указатель на блок данных для подписи
40,                            // Длина цифровой подписи
@ECCSign[1],                   // Указатель на блок данных, куда будет помещена
                               цифровая подпись
nil);                          // Не используется
```

Подпись сгенерирована. Можно переходить ко второй части защиты в процедуре обработки нажатия на кнопку отключения фильтра (*Form1.BtnFilterDisabledClick*).

В начале данного блока также необходимо получить случайное число. В этот раз для генерации воспользуемся алгоритмом RND64 из набора Guardant API. Этот алгоритм является безопасным и может быть использован наряду с Microsoft CryptoAPI.

```
nRet:=GrdTransform( hGrd,   // Хэндл контейнера Guardant
```

```
02,                    // Номер аппаратного алгоритма
8,                     // Длина блока данных
@GrdRand[1],           // Указатель на блок данных
0,                     // Должно быть 0
nil);                  // Не используется
```

Если результат случайного числа попал в 20%, проверяем, сгенерирована ли уже цифровая подпись, т.к. возможен вариант, когда в начале работы программы цифровая подпись не сгенерирована, а проверка должна сработать. Для обеспечения проверки используется функция **GrdCRC** из Guardant API. Она позволяет вычислить CRC участка памяти или кода. В данном случае вычисляется CRC цифровой подписи. Первоначально массив цифровой подписи заполнен байтами с кодом 0, поэтому результат CRC так же равен 0. Для усиления защиты можно первоначально заполнить блок цифровой подписи значениями, отличными от нуля, а затем результат работы функции GrdCRC сравнивать с заранее посчитанным результатом.

```
GrdCRC(@ECCSign[1],    { Указатель на блок данных }
20,                    { Длина блока данных }
0)                     { Начальное значение CRC }
```

Если все условия успешно пройдены, самое время переходить к самому ответственному этапу – проверке электронно-цифровой подписи. Проверка ЭУП осуществляется с помощью функции **GrdVerifySign**.

```
nRet:= GrdVerifySign( hGrd,    {Хэндл контейнера Guardant}
        GrdVSC_ECC160,         { Тип алгоритма }
        40,                    { Длина публичного ключа}
        @ECCPublicKey[1],      { Указатель на публичный ключ }
        20,                    { Длина данных, которые были подписаны }
        @DataForSign[1],       { Данные, которые подписывались }
        40,                    { Длина ЭЦП }
        @ECCSign[1],           { Указатель на ЭЦП }
        nil);                  { Не используется }
```

Как можно заметить, в прототипе отсутствует параметр «*номер алгоритма*», его заменяет «*тип алгоритма*». Это связано с тем, что функция **GrdVerifySign** реализована программно, а не аппаратно внутри ключа. Для усиления защиты можно встроить исходный код данного алгоритма в приложение (он доступен в комплекте разработчика) и смешать его с полезным функционалом программы.

Одним из самых важных параметров функции **GrdVerifySign** является «*указатель на открытый ключ*», по которому и происходит проверка электронно-цифровой подписи. Он представляет собой последовательность из 40 байт. Узнать его можно в редакторе памяти ключей Guardant в режиме редактирования алгоритма на вкладке «Ключ ECC160». Хранить открытый ключ в программе в открытом виде опасно, так как он может быть подменен. В тестовом приложении расшифрование открытого ключа реализовано с помощью аппаратного алгоритма GSII64 в обработке события *Form1.Show*.

```
nRet:=GrdCrypt(hGrd,           // Хэндл контейнера Guardant
01,                            // Номер аппаратного алгоритма
```

```
40,                          // Длина строки для шифрования
@ECCPublicKey[1],            // Указатель на строку для шифрования
GrdAM_OFB + GrdAM_Decode,    // Дешифрование по методу OFB
@szInitVectorGS2[1],         // 64-битный вектор инициализации
nil,                         // Не используется
nil );                       // Не используется
```

Рекомендуется также проверять целостность открытого ключа перед его использованием.

Взлом программы осложнился, однако возможен вариант, при котором хакер подменит код функции проверки ЭЦП, заставив ее всегда выдавать положительный результат. Чтобы исключить данную ситуацию, в код программы можно ввести заведомо неверную проверку и корректно обработать результат выполнения функции GrdVerifySign. Для наглядности проверка неверной ЭЦП добавлена сразу за основной функцией, хотя грамотнее было бы расположить несколько подобных функций в различных местах программы.

2.5 Использование однонаправленных алгоритмов AES128

Алгоритм AES128 (Advanced Encryption Standard) представляет собой блочный симметричный алгоритм шифрования, принятый в качестве стандарта шифрования США. Длина секретного ключа AES128 составляет 16 байт (128 бит). Длина блока данных, преобразуемых алгоритмом за один цикл, также составляет 16 байт. Аналогично GSII64, алгоритм AES может работать в режимах, позволяющих шифровать как кратные 16 байтам блоки данных, так и блоки произвольной длины. В схеме защиты тестового приложения использованы 2 алгоритма AES128: первый – «только шифрование», второй – «только расшифрование». В реальном приложении оставив, к примеру, «только расшифрование», мы можем затруднить злоумышленнику анализ защиты и создание эмулятора. Важное условие для получения работающей пары алгоритмов, один из которых только шифрует, а другой только расшифровывает, определители алгоритмов (в маске ключа), должны быть идентичны.

Созданные алгоритмы позволяют получить таблицу «вопросов-ответов», с помощью которой осуществляется проверка наличия аппаратного ключа. В рамках тестового приложения представлено два варианта подобной защиты. Они находятся в функции обработке события *DBNavigator1.BeforeAction*.

Первый вариант защиты построен на таблице из 1000 вопросов-ответов алгоритма AES128. Проверка располагается в блоке удаления строки из базы данных.

Для построения таблицы в качестве примера было создано отдельное консольное приложение. Его можно найти в папке «GenerateAES» данного урока, в подпапке «1000». В результате выполнения данной программы в файле result1000.txt будет создан исходный код на языке C#, готовый к вставке в проект. Аналогичного результата можно добиться при использовании функции «Создание отчета алгоритма» утилиты программирования ключей.

Рассмотрим подробно программу генерации вопросов-ответов.

В начале программы происходит подключение к ключу и активация алгоритма. Алгоритм шифрования находится в неактивном состоянии, т.к. для работы основной программы он не нужен, и давать потенциальному взломщику возможность шифровать другие данные нельзя. Методы активации и деактивации алгоритма аналогичны подобным методам, работающим с защищенными ячейками. Они подробно рассматривались в первом уроке, поэтому заострим внимание на методе шифрования.

Для работы с алгоритмом шифрования AES128 используется функция GrdCrypt. Даная функция уже рассматривалась ранее при работе с алгоритмом GSII64. Использование ее с алгоритмом AES128 происходит так же.

Единственное, что следует уточнить, это режим шифрования. Используем CBC. Этот режим преобразует блок открытого текста в блок шифротекста такой же длины, но в отличие от алгоритма OFB длина блока шифрования должна быть кратна блоку преобразуемых данных (то есть 16 байтам). Шифрование каждого следующего блока в этом режиме зависит от результата шифрования предыдущего.

После завершения генерации 1000 вопросов-ответов в файле «result1000_arr.txt» находится исходный код, который необходимо вставить в основную программу. Чтобы

не засорять главный модуль программы большим количеством текста, лучше всего создать отдельный (в тестовой программе это «UnitTelCode.pas») и поместить код туда.

Защита программы на основе таблицы вопросов-ответов реализована в блоке удаления записи из БД *(процедура TForm1.DBNavigator1BeforeAction)*. С помощью Microsoft CryptoAPI получается случайное число, которое используется для выбора одной записи из 1000. Далее выбранная запись расшифровывается с помощью GrdCrypt.

Обратите внимание, после дешифрования записи она не сравнивается напрямую «*if (StrDeCrypt = StrNoCrypt) then…*» так как в этом случае взломщик сможет легко заметить и вырезать проверку, а используется в качестве параметра в SQL запросе, что также затрудняет анализ и модификацию защиты.

Второй вариант защиты, основанный на использовании алгоритма AES128 по вызовам фнкций GrdCrypt, аналогичен первому, но в нем реализована более интересная схема защиты.

Защита строится на создании ассоциативного массива, индексом в котором является телефонный код города, а значением зашифрованное название города. При добавлении нового клиента из ассоциативного массива по телефонному коду зашифрованная строка расшифровывается на алгоритме AES128, и результат заносится в базу данных. В этом случае беспрепятственно вырезать или отключить алгоритм расшифровки не получится, т.к. он задействован в логике программы.

Более подробно ознакомиться с реализацией второго варианта можно в коде добавления новой записи в БД тестового приложения (процедура *«TForm1.DBNavigator1BeforeAction»)* и в дополнительной консольной программе *«GenerateTownAES.dpr»*.

2.6 Способы обработки кодов ошибок Guardant API

В первом уроке и в части методов этого урока для облегчения восприятия использовалась одна стандартная функция проверки результатов выполнения Guardant API. Однако в реальном приложении использование одной функции проверки не рекомендуется, так как она может подсказать взломщику адреса вызовов всех функций Guardant API, а также позволит сразу отключить все проверки. Решить эту проблему можно, осуществляя проверку различными способами. В этой части будут представлены некоторые из них.

Все виды проверок для наглядности размещены в одном месте (в процедуре *TForm1.DBNavigator1BeforeAction*). В реальном приложении проверки кодов ошибок обязательно должны использоваться для контроля результатов выполнения вызовов Guardant API, особенно в функциях, работающих с защищенными ячейками и аппаратными алгоритмами.

Виды проверок:

Обычная проверка (условный переход)

В отличие от использования единой функции проверки кода возврата, данный способ затрудняет взломщику поиск и модификацию всех проверок, хотя по коду ничем не отличается от отдельно выделенной функции.

```
if (nRet<>0) then
Begin
GrdLogout(hGrd, 0);              // Завершаем сеанс работы с ключом
GrdCloseHandle( hGrd );          // Закрываем хэндл, освобождаем память
GrdCleanup();              // Деинициализация копии Guardant API
ExitProcess(0);exit;
End;
```

Проверка с помощью создания исключительной ситуации

В блоке try..except создается исключительная ситуация, например, деление на ноль, а при обработке исключения происходит выполнение нужных действий. Созданный подобным образом обработчик сложнее анализируется и может серьезно затруднить взломщику поиск кода проверки.

```
try
TempReal := Random(200)/nRet;   // При успешном результате nRet = 0
except
on EZeroDivide do
begin
                                        //нужный код
end;
end;
```

Проверка с помощью использования результата в вызове функции или в вычислениях

В расчет принимается то, что при успешном завершении функция Guardant API возвращает 0, поэтому данное число может быть использовано в различных видах вычислений. Например,

```
Form1.Top := Form1.Top + nRet *1589;
```

При этом в случае корректного завершения работы функции результат вычисления будет верным, и ничего не изменится, в противном случае главная форма программы пропадет с экрана. Этот же принцип будет работать для других математических вычислений. Однако будьте внимательны: в случае критичности результатов вычислений для данных пользователя старайтесь не использовать этот прием. Так же можно использовать результат в различных действиях. Например, при конструировании SQL запросов.

```
ADOQuery1.SQL.Add('DELETE FROM ClientsList WHERE (ClientID='+ClientID+'
AND '+IntToStr(nRet)+'=0 )');
```

В этом случае, если функция Guardant API вернула ошибку, то SQL запрос не будет выполнен.

Использование проверки результата в вычислениях и действиях позволяет сильно осложнить взлом программы злоумышленником.

Отложенная проверка

Суть метода состоит в проверке возможной ошибки не сразу после выполнения метода Guardant API, а через какое-то время в другом блоке. Это затрудняет взломщику анализ логики защиты. Данный метод удобно сочетать с одним из способов, описанных выше.

3. Несколько важных моментов

Набор аппаратных алгоритмов для защиты программ, предоставляемых ключами Guardant, разнообразен. Но многообразие возможностей не дает гарантии надежной защиты, если их не использовать. Только активное использование различных аппаратных алгоритмов во всех важных частях программы в совокупности с замысловатой логикой самой защиты может сделать ее по-настоящему надежной и устойчивой к взлому.

Важный совет. При работе с аппаратными алгоритмами старайтесь уйти от обычных проверок, которые позволяют взломщику легко менять логику программы. Используйте результаты работы алгоритма в различных вычислениях или действиях. Этим же принципом нужно руководствоваться при обработке ошибок.

Стоит отметить то, что, несмотря на многообразие реализованных проверок в логике защиты, многие из них по своей сути просты и могут быть взломаны так называемым «битхаком» (например, изменением условного перехода на безусловный). Также не исключены атаки на вызовы функций генераторов случайных чисел, также способные привести к неработоспособности ряда механизмов защиты.