Guardant®

Система защиты от компьютерного пиратства

Эффективная
защита приложений

Урок 3.1: Guardant API
Настройка среды разработки

Содержание

Основы использования Guardant API

В предыдущих уроках была рассмотрена обобщенная последовательность действий, выполняемых защите приложения и подготовки ключей к передаче конечным пользователям. В процессе их выполнения были получены базовые навыки работы с утилитой программирования ключей GrdUtil и Мастером автозащиты.

Процесс защиты приложения предлагалось разбить на несколько этапов:

- Уточнение политики лицензирования
- Программирование электронного ключа в соответствии с выбранной политикой
- Реализация защиты при помощи Guardant API
- Установка автозащиты

В серии уроков 3.x будут подробно рассмотрены особенности реализации защиты при помощи Guardant API – основного этапа создания защиты приложения.

Данный урок посвящен настройке среды разработки и основам использования Guardant API в приложении.

Используемые термины и обозначения

Демо-коды – комплект демонстрационных (общеизвестных) кодов доступа для работы с электронным ключом Guardant.

Контрольный пример

На примере простого консольного приложения, реализующего выбор напитка, рассмотрим простейший способ использования API – процедуру проверки наличия электронного ключа. Выбор напитка происходит из вектора значений, сориентированного случайным образом. Ниже приведен листинг программы на языке C++:

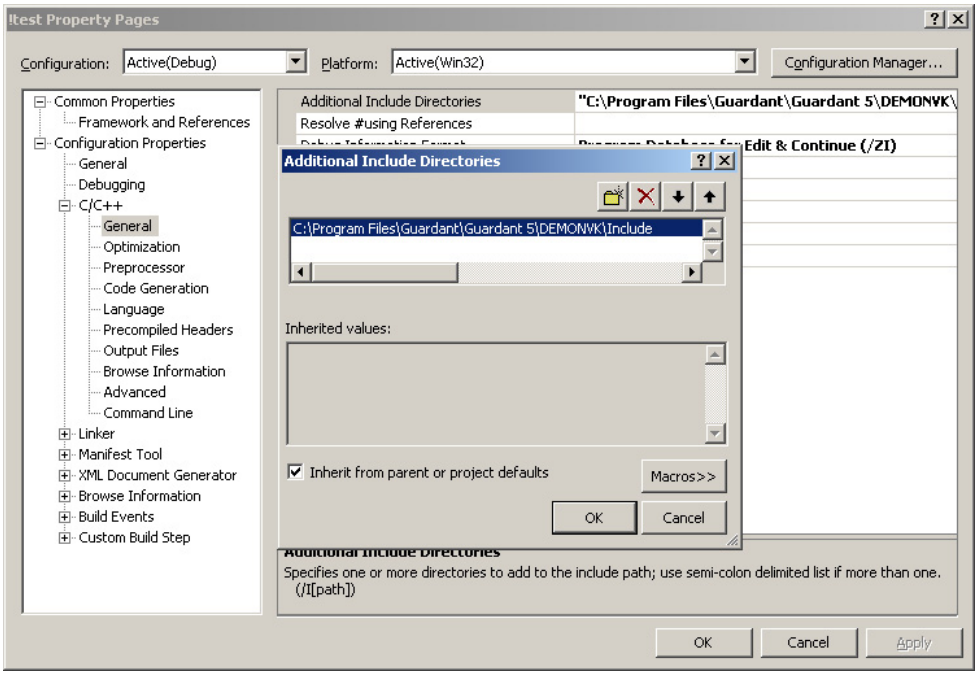
```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <ctime>

using namespace std;
ptrdiff_t random (ptrdiff_t i) { return (rand()% i);}
ptrdiff_t (*p_random)(ptrdiff_t) = random;

int main(int argc, char* argv[])
{
      vector<string> ans;
      string q="";
      srand ( unsigned ( time(NULL) ) );
      ans.push_back("Coffee");
      ans.push_back("Tea");
      random_shuffle(ans.begin(),ans.end(),p_random);
      cout << "============ Tea or Coffee ...============"<< endl;
      cout << "Heads or tails ? Enter you choice ( \"h\" or \"t\" ):";
      while ((q!="t") & (q!="h"))      cin >>q;
      cout << "Today your drink is ";
      if (q=="t")
            cout<< ans[0];
      else
            cout<< ans[1];
      cout << " !!!" << endl;
      cout << "========================================="<< endl;
      return 0;
}
```

Настройка среды разработки

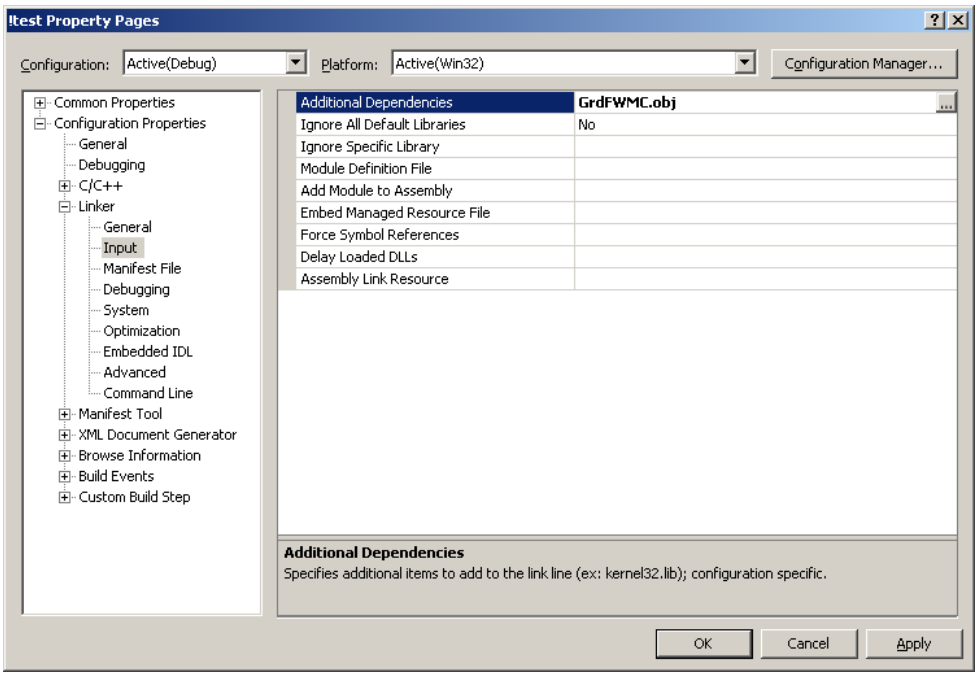
В качестве среды разработки возьмем Microsoft Visual C++ 2008 Express Edition. Ее можно бесплатно скачать с сайта Microsoft.

Настройка проекта

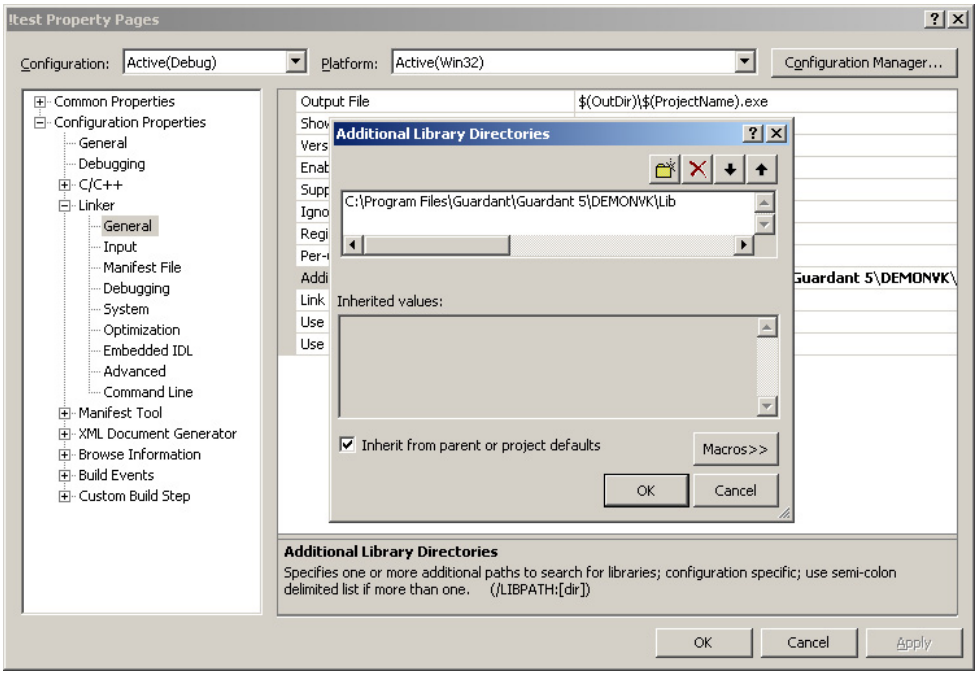
Для использования функций Guardant API необходимо указать компилятору путь к заголовочным файлам Guardant API (будем использовать заголовочный файл **grddongle.h**, в котором определен C++ класс **CGrdDongle** для работы с ключом):



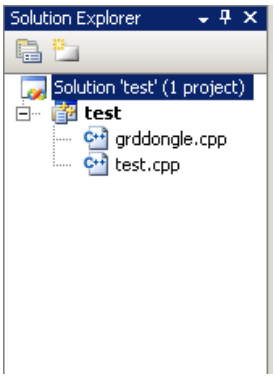
Также необходимо сообщить линкеру об использовании соответствующего среде разработки объектного файла (указывается в примере из комплекта разработчика для соответствующего языка программирования). Для Microsoft Visual C++ это **GrdFWMC.obj**:



Путь к нему также необходимо указать во вкладке **General** настроек линкера:



Добавим в проект дополнительно файл с реализацией методов класса **CGrdDongle** – **grddongle.cpp**:



Теперь проект готов к использованию.

Порядок вызовов Guardant API

Общая структура вызовов описана в справочной системе по Guardant API (см. **GrdAPI.chm**). Ее упрощенно можно представить следующим образом:

- Блок инициализации
  - GrdStartup()
  - GrdCreateHandle()
  - GrdSetAccessCodes()
  - GrdSetFindMode()
  - GrdFind()
  - GrdLogin()
- Основной блок
- Блок деинициализации
  - GrdLogout()
  - GrdCloseHandle()
  - GrdCleanup()

Блок инициализации необходим для установки критериев поиска электронного ключа и параметров работы с ним. Блок завершается вызовом функции **GrdLogin()**, успешное выполнение которой говорит о начале сессии взаимодействия с ключом.

Вызов функций **GrdSetFindMode()** и **GrdFind()** необязателен. Они служат для сужения критериев поиска ключей и получения общей информации о присутствующих в системе ключах, до непосредственного соединения с одним из них.

Блок деинициализации выполняется при завершении работы с защищенным функционалом приложения, а также при обработке исключений, в случае аварийного завершения защищенного приложения.

Добавим в представленный выше код тестового приложения вызовы Guardant API, осуществляющие проверку наличия ключа Guardant.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <ctime>
#include "grddongle.h"

using namespace std;
ptrdiff_t random (ptrdiff_t i) { return (rand()% i);}
ptrdiff_t (*p_random)(ptrdiff_t) = random;

int main(int argc, char* argv[])
{
      vector<string> ans;
      string q="";
      srand ( unsigned ( time(NULL) ) );

      int nRet = 0;

      CGrdDongle GrdDongle( GrdFMR_Local );
      nRet = GrdDongle.Create(GrdDC_DEMONVK, GrdDC_DEMORDO, 0, 0);
      nRet = GrdDongle.Login( -1l );
      if(nRet)
            cout << "Dongle login failed!" << endl;
      else
      {
            ans.push_back("Coffee");
            ans.push_back("Tea");
            random_shuffle(ans.begin(),ans.end(),p_random);
            cout << "============ Tea or Coffee ...============="<< endl;
            cout << "Heads or tails ? Enter you choice ( \"h\" or \"t\" ):";
            while ((q!="t") & (q!="h"))      cin >>q;

            nRet = GrdDongle.Check ();
            if(nRet)
                  cout << "Dongle not found!" << endl;
            else
            {
                  cout << "Today your drink is ";
                  if (q=="t")
                        cout<< ans[0];
                  else
                        cout<< ans[1];
                  cout << " !!!" << endl;
                  cout << "=========================================="<< endl;
            }
      }
      return 0;
}
```

Блоки инициализации и деинициализации реализованы в конструкторе и деструкторе класса **CGrdDongle** соответственно. Так как созданный экземпляр класса расположен в статической памяти, деструктор будет вызван автоматически при выходе из функции **main**.

Функция **GrdCheck()**, вызываемая перед принятием решения и его выводом осуществляет проверку наличия в порту USB-ключа Guardant, с которым был выполнен **GrdLogin()**.

Заключение

В процессе выполнения данного урока была рассмотрена процедура настройки среды разработки и реализована при помощи Guardant API простейшая процедура проверки наличия электронного ключа.

В приложении к уроку содержится рассмотренный пример в составе полностью настроенного и готового к использованию проекта среды Microsoft Visual C++ 2008.

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