Статья Внедрение своего кода в адресное пространство процессов

Внедрение своего кода(динамически) в чужие процессы — штука достаточно интересная. Это может служить как во благо, так и во зло. Хотя, понятие «зло», местами, весьма абстрактно в информационном мире, я не могу провести точную границу между тем, что «плохо», а что «хорошо», тем более, если это касается внедрения кода...

В данной статье мы займемся созданием своего DLL инжектора. Что это такое, думаю, знают все. Такой способ внедрения стороннего кода достаточно популярен и удобен.

Писать DLL Injector мы будем на C++ в среде Microsoft Visual Studio 2010. Для создания динамически подключаемой библиотеки можно использовать любой инструмент, который вам по душе. Я же для создания библиотеки выбрал CodeGear RAD Studio 2009, язык Delphi(Object Pascal).

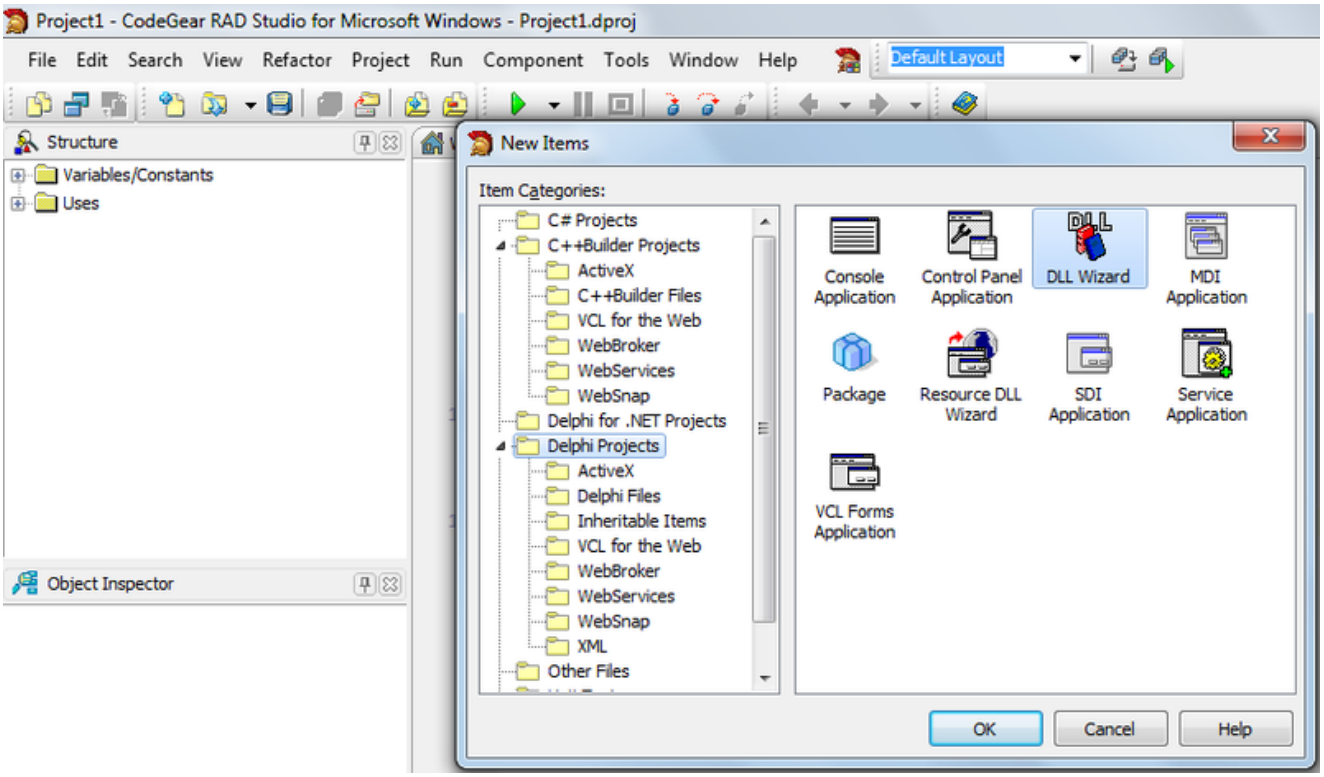
Как же работает DLL Injection ?

Схема работы данного метода проста:

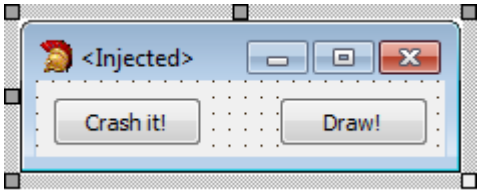
> 1) поиск и получение дескриптора нужного процесса
> 2) выделение памяти в процессе и последующая запись пути в DLL`ке по адресу, где произошло выделение памяти
> 3) создание нового потока в виртуальном пространстве процесса, дескриптор которого был получен.

Начнем с создания DLL.

Как я уже говорил, для этой цели будет использоваться язык Delphi:



Теперь добавим в DLL некоторые ресурсы, а именно — форму и кнопки для управления:



Сама DLL состоит лишь из нескольких строчек кода, которые будут отображать созданную форму. Также можно удалить все комментарии для более удобного визуального восприятия кода:

Теперь программируем интерфейс формы DLL. На форме есть две кнопки. Первая будет «убивать» родительский процесс(т.е процесс, в виртуальном пространстве которого был создан поток, в котором, в свою очередь, выполняется код DLL). Вторая — рисовать 10 квадратов размером 25x25px в контексте всех окон приложения, принадлежащих процессу:
C-like:

```
procedure TForm1.Button1Click(Sender:
TObject);
var _curr_process:DWORD;
    p_handle: THANDLE;
begin
// Убиваем родительский процесс
_curr_process:=GetCurrentProcessId();
p_handle:=OpenProcess(PROCESS_ALL_ACCESS,fal

TerminateProcess(p_handle,4);
end;

var dw:DWORD;  // PID текущего процесса

procedure drawGroup(h:HWND);
var canvas:TBitMap;
rect:TRect;
x,y:integer;
i: Integer;
begin
// рисуем 10 квадратов
for i := 1 to 10 do
begin
canvas:=TBitMap.Create();
canvas.Canvas.Handle:=GetDC(h);  //
устанавливаем дескриптор контекста
randomize;
canvas.Canvas.Brush.Color:=rgb(random(255),r

GetWindowRect(h,rect);
x:=random(rect.Right-rect.Left-25);
y:=random(rect.Bottom-rect.Top-25);
canvas.Canvas.Rectangle(x,y,x+25,y+25);
end;
end;

function func(h:HWND):BOOL;  stdcall;
var pid:DWORD;
begin
GetWindowThreadProcessId(h,pid);
// запускаем функцию рисования в окне, если
оно принадлежит нужному процессу:
if(pid=dw) then drawGroup(h);
result:=true;
end;

procedure TForm1.Button2Click(Sender:
TObject);
begin
// Запускаем функцию рисования квадратов
// ( в перечислителе окон системы )
```

```
library Project1;


uses
  SysUtils,
  Classes,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

var form:TForm1;
begin
form:=TForm1.Create(nil);
form.ShowModal();
end.
```

```
dw:=GetCurrentProcessId();
EnumWindows(@func,0);    // перечисляем
окна
end;
```

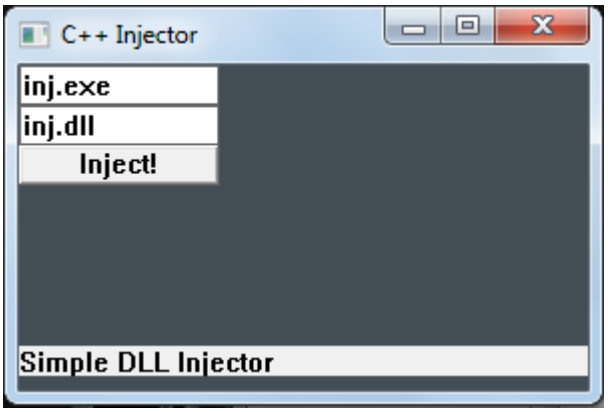
Исходный код интерфейсной части можно посмотреть **здесь** .
Тут все достаточно просто.
Итак, динамическая библиотека написана. Теперь компилируем ее и на выходе получаем скомпилированный файл с расширением ".dll", который можно переименовать для удобства. Я переименую библиотеку в «inj.dll».

Создание DLL завершено.
Осталось лишь скопировать нашу DLL`ку в системную директорию Windows, чтобы любое приложение могло отыскать её лишь по одному имени.

Переходим к разработке инжектора. Идем в Visual Studio и создаем Пустой проект(File->New->Project->Visual C++->General->Empty Project). Вся разработка будет производиться на «чистом» WinApi.

Первым делом создадим простейший визуальный интерфейс: форма, кнопка и текстовое поле. Выглядеть это будет примерно так:

Как видно, это приложение, обладающее минимальным дизайном и простейшим интерфейсом. В нем присутствуют два текстовых поля, предназначенных для ввода имени процесса(или его определенной части), в который требуется произвести инжектирование DLL и для ввода имени самой DLL. Кнопка, соответственно, запускает процесс инжекта.



Для инжекта DLL в адресное пространство процессов понадобятся следующие WinApi функции:

Для поиска нужного процесса:
CreateToolHelp32SnapShot
Process32First
Process32Next

Для инжекта:
OpenProcess
GetProcAddress

VirtualAllocEx
WriteProcessMemory
CreateRemoteThread

Итак, обдумаем, как же, собственно организовать «умную» архитектуру приложения, которая была бы понятной и простой.
Я предлагаю начать с обработки щелка по кнопке
C-like:

```
MSG msg;
    while (GetMessage(&msg,NULL,0,0))
    {

        // Button Click
        if(msg.hwnd==button && msg.message==WM_LBUTTONUP)
        {
            // Getting Process Name
            GetWindowText(edit_proc,buff,sizeof(buff));
            strncpy(_p_name,buff,BUFF);
            // Getting DLL Name
            ZeroMemory(buff,sizeof(buff));
            GetWindowText(edit_dll,buff,BUFF);
            strncpy(_dll_name,buff,BUFF);
            // Start Injection
            StartInjection();
        }
```

Ок, далее, переходим к реализации функции
C-like:

```
StartInjection();
```

которая служит для поиска процесса по его имени:
C-like:

```
int StartInjection()
{
    // Searching of Process
    PROCESSENTRY32 pe;
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
    if(snapshot==INVALID_HANDLE_VALUE)
    {
        ShowMessage("SnapShot Failed.");
        return 0;
    }
    pe.dwSize = sizeof(PROCESSENTRY32);
    int curr = Process32First(snapshot,&pe);
    while(curr)
    {
        CharLowerBuff(pe.szExeFile,lstrlen(pe.szExeFile));
        if(strstr(pe.szExeFile,_p_name)) {pid = pe.th32ProcessID; break;}
        curr = Process32Next(snapshot,&pe);
    }
    if(pid==NULL)
    {
        ShowMessage("Searching of process failed.");
        return 0;
    }

    bool result = Inject();
    if(result==false)
    {
        ShowMessage("Injection failed.");
        return 0;
    }

}
```

И, наконец, завершающая процесс инжектирования функция
Code:

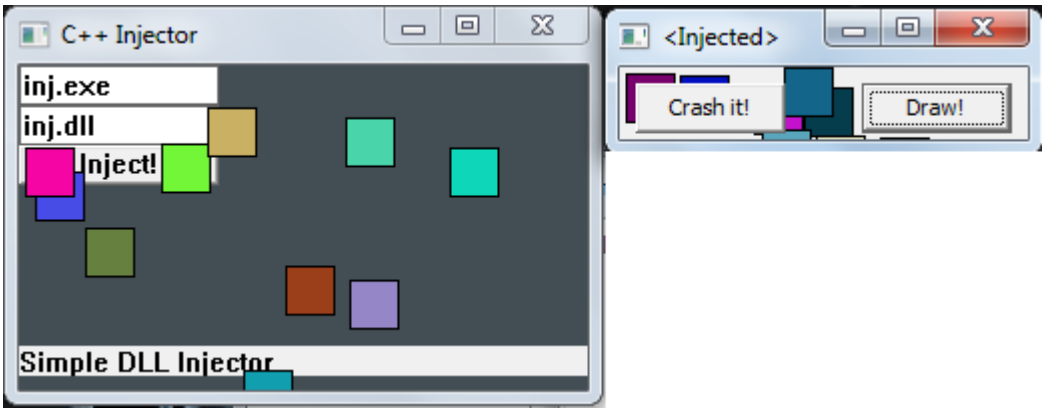
```
Inject()
```

C-like:

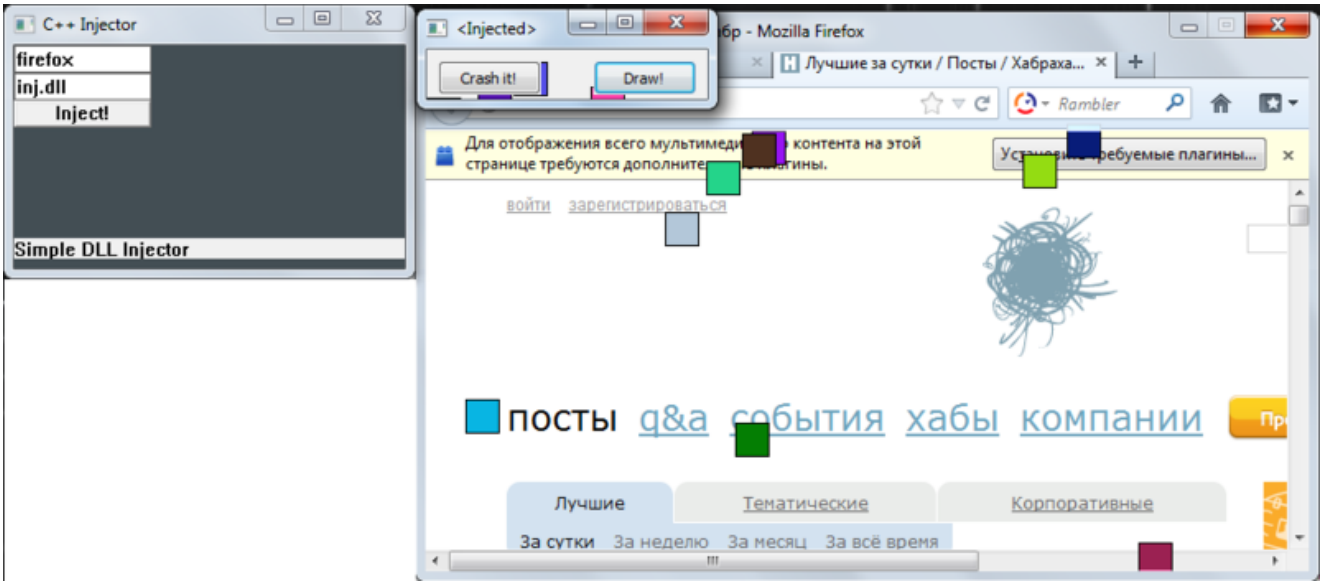
```
// Injection
bool Inject()
{
    if(pid==NULL) return false;

    // Получение дескриптора процесса
    HANDLE process = OpenProcess(PROCESS_ALL_ACCESS,false,pid);
    if(process==NULL) return false;

    // "Вытягивание" функции из системной библиотеки для динамической
    // подгрузки DLL в адресное пространство открытого процесса
    LPVOID fp = (LPVOID)GetProcAddress(GetModuleHandle("kernel32.dll"),"LoadLibraryA");
    if(fp==NULL) return false;

    // Выделение участка памяти размером strlen(_dll_name) для последующей
    // записи имени библеотеки в память процесса.
    LPVOID alloc = (LPVOID)VirtualAllocEx(process,0,strlen(_dll_name), MEM_RESERVE |
MEM_COMMIT, PAGE_READWRITE);
    if(alloc==NULL) return false;

    // Запись имени инжектируемой DLL в память
    BOOL w = WriteProcessMemory(process,(LPVOID)alloc,_dll_name,strlen(_dll_name),0);
    if(w==NULL) return false;

    // Создание "удаленного" потока в адресном пространстве
    // открытого процесса и последующая подгрузка нашей DLL.
    HANDLE thread = CreateRemoteThread(process,0,0,(LPTHREAD_START_ROUTINE)fp,
(LPVOID)alloc,0,0);
    if(thread==NULL) return false;

    CloseHandle(process);
    return true;

}
```

Полный код инжектора можно посмотреть **здесь**.

Тестируем работоспособность инжектора:



Сначала инжектимся «сами в себя». При клике на кнопку «Draw» происходит рисование 10 квадратов. При клике на «Crash it!» происходит немедленное завершение «родительского» процесса. Теперь попробуем инжектиться во что-нибудь посерьезнее, например, в браузер Mozilla Firefox. Для этого необходимо поменять лишь имя процесса в первом текстовом поле и нажать на кнопку:



Как видно, инжект успешно удался. Квадраты рисуются во всех окнах, принадлежащих родительскому процессу браузера. При нажатии на кнопку «Crash it!» Mozilla FireFox немедленно закрывается.

Outro

Для чего нам это вообще нужно? Ведь наша основная цель — это взлом(игр, софта). Так вот в дальнейшем мы, наверняка, будем использовать этот инжектор для внедрения своего кода в чужое адресное пространство. Благодаря этому можно сэкономить немало времени, сил и нервов

Удачи!

(c) Asen
Last edited by a moderator: Jul 8, 2019