32位程序对64位进程的远程注入实现

0x00 前言

要对指定进程进行远程注入，通常使用Windows提供的API CreateRemoteThread创建一个远程线程，进而注入dll或是执行shellcode。

在64位系统下，该方法需要特别注意，注入的目标进程要同程序的结构保持一致，即32位程序只能对32进程作注入，64位程序只能对64位进程作注入

> 32位程序对64位程序进行注入时会失败（32位和64位的结构不同）

然而，在某些特殊的环境下，无法提前预知目标进程的结构，准备两个不同版本的程序又不现实

所以只能重新思考这个问题：

> 32位程序真的无法对64位程序进行远程注入吗？

0x01 简介

我在odzhan的博客里找到了解决思路，文章地址如下：

https://modexp.wordpress.com/2015/11/19/dllpic-injection-on-windows-from-wow64-process/

本文将会介绍实现思路，参考odzhan的开源工程"pi"，编写测试代码，生成32位程序，实现对64位进程calc.exe的进程注入，验证32位程序能够对64进程作注入的结论

0x02 实现思路

1、32位程序支持对64位程序的读写

参考资料：

rgb/29a：

http://www.vxheaven.org/lib/vrg02.html

ReWolf：

http://blog.rewolf.pl/blog/

https://github.com/rwfpl/rewolf-wow64ext

2、 利用CreateRemoteThread作进程注入的通用方法

进程注入流程：

- OpenProcess
- VirtualAllocEx
- WriteProcessMemory
- VirtualProtectEx
- CreateRemoteThread
- WaitForSingleObject

在具体的实现过程中，如果指定了进程名称，需要先将进程名称转换为进程ID，参考代码如下：

```
DWORD processNameToId(LPCTSTR lpszProcessName)
{
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPR
    PROCESSENTRY32 pe;
    pe.dwSize = sizeof(PROCESSENTRY32);
    if (!Process32First(hSnapshot, &pe)) {
        MessageBox(NULL,"The frist entry of the process list
        return 0;
    }
    while (Process32Next(hSnapshot, &pe)) {
        if (!strcmp(lpszProcessName, pe.szExeFile)) {
            return pe.th32ProcessID;
        }
    }
    return 0;
}
```

依次实现如下操作：

- 根据进程ID打开进程，获得进程句柄
- 申请内存空间
- 写入数据
- 将内存改为可读可执行(可选)
- 创建线程
- 等待线程退出(可选)

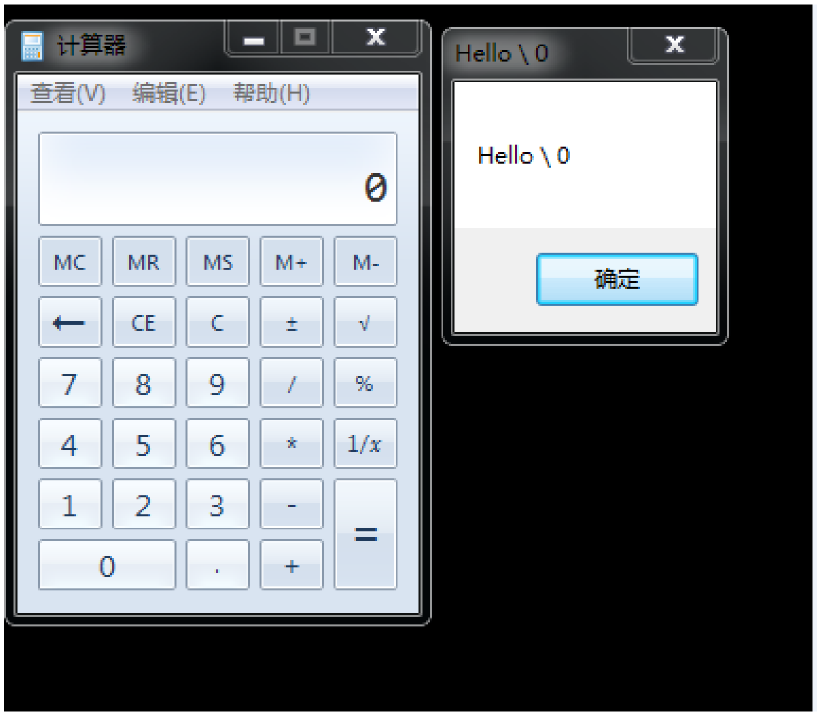
代码可参考：

http://blog.csdn.net/g710710/article/details/7303081

对参考代码作细微修改，将注入进程名称指定为calc.exe，完整代码已上传github，地址如下：

https://github.com/3gstudent/CreateRemoteThread/blob/master/CreateRemoteThreadTest.cpp

程序运行后，查找进程calc.exe，接着尝试远程注入，弹出对话框，如图



Hello \ 0

将程序编译成x86，对32位的进程calc.exe进行注入，成功

将程序编译成x64，对64位的进程calc.exe进行注入，成功

将程序编译成x86，对64位的进程calc.exe进行注入，OpenProcess、VirtualAllocEx、WriteProcessMemory、VirtualProtectEx均正常，执行CreateRemoteThread时会报错

解决思路：

参考rgb/29a和ReWolf的思路，将此处的CreateRemoteThread切换为64位后再创建线程，完成后再切换回32位，即可实现32位程序对64位进程的远程注入

3、判断当前系统是32位还是64位

使用API：

```
void WINAPI GetNativeSystemInfo(
  _Out_ LPSYSTEM_INFO lpSystemInfo
);
```

查看结构体中的wProcessorArchitecture可获得CPU架构，进而判断操作系统

代码如下：

```
#include <windows.h>
BOOL Is64BitOS()
{
    typedef VOID (WINAPI *LPFN_GetNativeSystemInfo)( __out LP
    LPFN_GetNativeSystemInfo fnGetNativeSystemInfo = (LPFN_Ge
    if(fnGetNativeSystemInfo)
    {
        SYSTEM_INFO stInfo = {0};
        fnGetNativeSystemInfo( &stInfo);
        if( stInfo.wProcessorArchitecture == PROCESSOR_ARCHIT
            || stInfo.wProcessorArchitecture == PROCESSOR_ARC
        {
            return TRUE;
        }
    }
    return FALSE;
}
int main()
{
    if (Is64BitOS())
        printf("x64\n");
    else
        printf("x86\n");
    return 0;
}
```

4、判断注入的进程是32位还是64位

查找进程ID，打开进程，获得句柄，使用API，传入参数，进行判断

使用API：

```
BOOL WINAPI IsWow64Process(
  __in HANDLE hProcess,
  __out PBOOL Wow64Process
);
```

返回true, 代表进程是32位，否则是64位

完整代码如下：

```
#include <windows.h>
#include <TlHelp32.h>

BOOL IsWow64(HANDLE hProcess)
{
    typedef BOOL (WINAPI *LPFN_ISWOW64PROCESS) (HANDLE, PBOOI
    LPFN_ISWOW64PROCESS fnIsWow64Process;

    BOOL bIsWow64 = FALSE;
    fnIsWow64Process = (LPFN_ISWOW64PROCESS)GetProcAddress(
    GetModuleHandle("kernel32"),"IsWow64Process");

    if (NULL != fnIsWow64Process)
    {
        fnIsWow64Process(hProcess, &bIsWow64);
    }
    return bIsWow64;
}

DWORD processNameToId(LPCTSTR lpszProcessName)
{
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPR
    PROCESSENTRY32 pe;
    pe.dwSize = sizeof(PROCESSENTRY32);
    if (!Process32First(hSnapshot, &pe)) {
        MessageBox(NULL,
            "The frist entry of the process list has not been
        return 0;
    }
    while (Process32Next(hSnapshot, &pe)) {
        if (!strcmp(lpszProcessName, pe.szExeFile)) {
            return pe.th32ProcessID;
        }
    }
    return 0;
}

int main()
{
    BOOL            bWow64;
    char *szExeName="calc.exe";
    DWORD dwProcessId = processNameToId(szExeName);
    if (dwProcessId == 0) {
        MessageBox(NULL, "The target process have not been fo
        return -1;
    }
    HANDLE hTargetProcess = OpenProcess(PROCESS_ALL_ACCESS, F
    if (!hTargetProcess) {
        MessageBox(NULL, "Open target process failed !",
            "Notice", MB_ICONINFORMATION | MB_OK);
        return 0;
    }
    bWow64 = IsWow64(hTargetProcess);

    if(bWow64)
```

```
        printf("32-bit process\n");
    else
        printf("64-bit process\n");
}
```

5、开源工程pi

下载地址：

https://github.com/odzhan/shellcode/tree/master/win/pi

```
usage: pi [options] <proc name | proc id>

       -d          Wait after memory allocation before runnir
       -e <cmd>    Execute command in context of remote proce
       -f <file>   Load a PIC file into remote process
       -l <dll>    Load a DLL file into remote process
       -p          List available processes on system
       -x <cpu>    Exclude process running in cpu mode, 32 or
```

```
examples:

    pi -e "cmd /c echo this is a test > test.txt & notepad te
    pi -l ws2_32.dll notepad.exe
    pi -f reverse_shell.bin chrome.exe
```

测试系统：

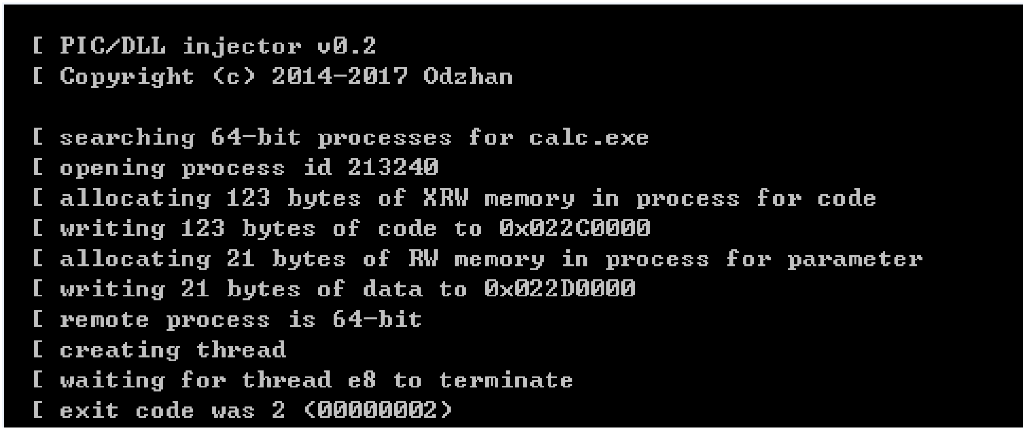
Win7x64

cmd执行：

```
pi32.exe -e "cmd /c start calc.exe" -x32 calc.exe
```

上述命令将对64位的calc.exe进行注入

回显如图



```
[ PIC/DLL injector v0.2
[ Copyright (c) 2014-2017 Odzhan

[ searching 64-bit processes for calc.exe
[ opening process id 213240
[ allocating 123 bytes of XRW memory in process for code
[ writing 123 bytes of code to 0x022C0000
[ allocating 21 bytes of RW memory in process for parameter
[ writing 21 bytes of data to 0x022D0000
[ remote process is 64-bit
[ creating thread
[ waiting for thread e8 to terminate
[ exit code was 2 (00000002)
```

payload没有成功执行

0x03 最终代码

虽然pi测试失败，但是代码值得参考，提取关键代码，开发测试程序

测试程序结构如下:

判断当前系统

- 如果为32位系统， 调用系统api CreateRemoteThread，对目标进程尝试远程注入，弹出对话框
- 如果为64位系统，进入下一个分支，对进程判断

判断进程calc.exe

- 如果为32位，调用系统api CreateRemoteThread，对目标进程尝试远程注入，弹出对话框
- 如果为64位，调用自定义api CreateRemoteThread64，对目标进程尝试远程注入，执行payload：”cmd /c start calc.exe”

完整代码已上传github，下载地址如下:

https://github.com/3gstudent/CreateRemoteThread/blob/master/CreateRemoteThread32to64.c

0x04 实际测试

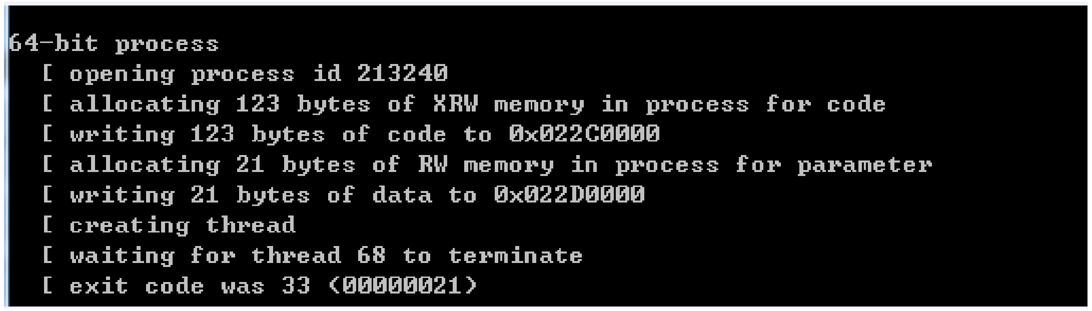
测试系统:

Win7 x64

1、将程序编译成32位，打开64位calc.exe

2、运行测试程序

命令行输出如图



```
64-bit process
  [ opening process id 213240
  [ allocating 123 bytes of XRW memory in process for code
  [ writing 123 bytes of code to 0x022C0000
  [ allocating 21 bytes of RW memory in process for parameter
  [ writing 21 bytes of data to 0x022D0000
  [ creating thread
  [ waiting for thread 68 to terminate
  [ exit code was 33 (00000021)
```

成功执行payload：”cmd /c start calc.exe”，弹出计算器

0x05 小结

本文介绍了32位程序对64位进程远程注入的实现方法，参照以上代码可实现Windows 32位/64位系统下进程注入的通用模板。

LEAVE A REPLY

Written on February 10, 2017