Hide'n'Seek?
Anatomy of Stealth Malware

Gergely Erdélyi

Gergely.Erdelyi@F-Secure.com
Agenda

• What is stealth malware?
• Simple stealth tricks
• User space stealth code
• Kernel space stealth code
• Real world examples
• Detection of stealth malware
• Conclusion
What is Stealth Malware?

• Stealth is the code that tries to conceal its presence

• Stealth code is nothing new

• The very first PC virus (Brain) was stealth

• DOS viruses took stealth code to extremes
Hiding Behind Complexity

• Windows is excessively complex

• Size of Windows XP:
  • 40 million lines of code
  • around 10,000 files
  • approximately 1 gigabyte (including data)

• It contains a number of files with unclear purpose
Know the Components (?)

Filename: CMS32.DLL
Description: 'Console Messaging Subsystem Library'

Filename: WOW32.DLL
Description: '32-bit WOW Subsystem Library'

Which one does not belong to Windows?
Optical Tricks

Can you see the difference?

kernel32.dll
kerne132.dll
Hiding behind the file manager

• Windows Explorer does not show extensions by default

• Files with HIDDEN and/or SYSTEM attributes are not shown

• The behaviour of Explorer is controlled through registry

[HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced]
Hidden,
SuperHidden,
HideFileExt
Hooking 1/2

Application

Windows API
Hooking 2/2

Application

Hook:
Call original API
Filter data
Return

Windows API
Import Address Table Modification 1/2

[Diagram showing headers and sections with function calls and addresses]
Import Address Table Modification 2/2

headers

<table>
<thead>
<tr>
<th>Code Section</th>
<th>CALL [CreateFileA]</th>
</tr>
</thead>
</table>

import section

| CreateFileA: Addr. Of Hook |

hook:

<table>
<thead>
<tr>
<th>Kernel32.dll</th>
</tr>
</thead>
</table>

| JMP 0x12345678 |

createfilea():

...
Code Patching
(before)

FindNextFileA:

195D6: 55 PUSH EPB
195D7: 8B EC MOV EBP, ESP
195D9: 81 EC 60 02 00 00 SUB ESP, 260

195DF: 53 PUSH EBX
195E0: 8D 85 A0 FD FF FF LEA EAX, [EBP-260]
Code Patching
(after)

FindNextFileA:

195D6: E9 78 56 34 12   JMP   H_FindNextFileA
195DB: 60 02 00 00   XXX

195DF: 53   PUSH   EBX
195E0: 8D 85 A0 FD FF FF LEA   EAX, [EBP-260]
Code Patching
(hook)

H_FindNextFileA(arguments)
{
    Process_Arguments();
    Restore_First_Bytes(Hooked_Function);
    FindNextFileA();
    Alter_Data();
    Patch_First_Bytes(Hooked_Function);
}
Code Patching With Instruction Analysis

FindNextFileA:
195D6: 55  PUSH EPB
195D7: 8B EC  MOV EBP, ESP
195D9: 81 EC 60 02 00 00  SUB ESP, 260

FindNextFileA_Cont:
195DF: 53  PUSH EBX
195E0: 8D 85 A0 FD FF FF LEA EAX, [EBP-260]
Code Patching
With Instruction Analysis

FindNextFileA:
195D6: **E9 78 56 34 12**  JMP  H_FindNextFileA
195DB: 90  NOP
195DC: 90  NOP
195DD: 90  NOP
195DE: 90  NOP

FindNextFileA_Cont:
195DF: 53  PUSH  EBX
195E0: 8D 85 A0 FD FF FF LEA  EAX, [EBP-260]
Code Patching
With Instruction Analysis

H_FindNextFileA(arguments) {
    Process_Arguments();
    Original_FindNextFileA()
    Alter_Data();
}

Original_FindNextFileA:
20000: 55           PUSH EBP
20001: 8BEC         MOV  EBP, ESP
20003: 81EC60020000 SUB  ESP, 260
20009: E9XXXXXXXX   JMP FindNextFileA_Cont
Installing the Hooks (WinNT)

LPVOID VirtualAllocEx(
    HANDLE hProcess,
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD flAllocationType,
    DWORD flProtect);

VirtualProtectEx();
WriteProcessMemory();
CreateRemoteThread();
Installing the Hooks (WinNT)

Attacking Process

- Headers
  - Code Section
    - VirtualAllocEx()
    - VirtualProtectEx()
    - WriteProcessMemory()
    - CreateRemoteThread()

Victim Process

- Headers
  - Code Section
  - Heap
    - Install_Hooks();
**DLL Injection**

- The hooks and install routine is placed in a DLL
- The attacker injects a `LoadLibrary("Nasty.dll")` call
- Using `CreateRemoteThread()` the code is executed
- The system loads the DLL and calls `DllMain()`
- `DllMain()` installs the hooks
- `Nasty.dll` is active in the remote process and monitors it
Direct Memory Writing

• The attacker uses `VirtualAllocEx()` to allocate memory
• The hooks and installer is copied using `WriteProcessMemory()`
• The installer is started with `CreateRemoteThread()`
• The injected code must be position independent
Kernel Space Hooks

• Code running in kernel space has more control
• Kernel space hooks are more difficult to detect
• Writing kernel code is harder
• Any mistake can cause total system failure
• Kernel code is highly OS version dependent
File System Hooks
(Windows 9x/ME)

IFSMgr_InstallFileSystemApiHook(
    pIFSFFileHookFunc HookFunc
);

FileSystemApiHookFunction(
    pIFSFFunc FSDFnAddr,
    int FunctionNum,
    int Drive,
    int ResourceFlags,
    int CodePage,
    pioreq pir
);
Registry and Other API Hooks (Windows 9x/ME)

- Using Virtual Machine Manager (VMM)
- `Hook_Device_Service()` installs hooks
- Device Driver Kit (DDK) has the headers for most common services (e.g. registry)
Installing Kernel Space Hooks (Windows 9x/ME)

• Loading a device driver (VxD)
  • Hooks are placed to a VxD file
  • VxD can be loaded with `CreateFile("\\\.\hook.vxd")`
  • VxD stays loaded until next restart

• Ring3 to Ring0 jump
  • Works by modifying the Interrupt Descriptor Table
  • Executes code in kernel space
  • Used by the Zerg and Sma viruses for example
Kernel Space Hooks in Windows NT/2k/XP

• Windows NT is based on a microkernel architecture

• On the top of the kernel several user subsystems run
  • Win32
  • OS/2
  • Posix

• The subsystems use the Native NT API to communicate with the kernel

• Hooks in the Native NT API provide global control
A Win32 API Call

APPLICATION LEVEL
KERNEL32.ReadFile()

WIN32 API LEVEL
NTDLL.ReadFile()

NATIVE API LEVEL
NTReadfile()

KERNEL LEVEL
SysCall Interface (KiSystemService)

NT KERNEL
Kernel Service Table

• Entry points to system services are stored in the System Service Descriptor Table (SSDT)

• SSDT stores services in four groups:
  – Core services (exported from NTDLL.DLL)
  – GUI services
  – Reserved
  – Reserved

SSDT is write protected in Windows XP but that can be circumvented by disabling the processor's WP bit
NT System Call

KiSystemService (Int 2Eh)

NTReadFile()
Service ID=0xB7

Service Dispatch Table

NTReadFile()::
...
...
Installable File System

• NT file system drivers can be created using the Installable File System API (IFS)

• IFS kit is available from Microsoft

• Through IFS it is possible to supervise all file operations

• IFS filters can be layered on top of each other

• Antivirus applications use IFS too
Direct Kernel Data Modification

• Requires knowledge on undocumented kernel internals

• Fu backdoor uses this method for different purposes:
  - Hiding processes by removing them from list of active processes
  - Adjust privileges by directly modifying the security token
  - Hide services by unlinking them from the module list

• This approach is OS version dependent and error prone
Installing the Hooks

• Hooks can be added as standard device drivers
  • With CreateService() using SERVICE_KERNEL_DRIVER flag
  • Drivers are loaded/unloaded automatically by the system

• Using NtSetSystemInformation()
  • The function SystemLoadAndCallImage is undocumented
  • It does not need registering to Service Control Manager
  • Loads and starts a driver in a running system
Real World Examples

• Not all programs using these techniques are malicious

• Malware (viruses, backdoors, etc) do misuse them

• Things they try to hide:
  • Files, directories
  • Free disk space change
  • Processes
  • Registry keys, values
  • Services
  • Open network ports
Examples of Stealth Malware 1/2

• HxDef (Hacker Defender) hides:
  • Processes
  • Services, drivers
  • Registry keys, values
  • Files, directories
  • Open network ports
Examples of Stealth Malware 2/2

- Vanquish rootkit uses DLL injection and hides:
  - Processes
  - Files, directories
  - Registry keys and values
  - Logs passwords
Detection of Stealth Malware

- User space stealth can be detected with kernel based scanner code
- Clean booting
- Detection of malware's communication channel
  - Mail slot
  - Other IPC mechanism
  - Challenge/response on the communication channel
- Detection of symptoms not hidden by the malware
- Careful manual inspection
Conclusion

• Stealth code for Windows is reality
• It is becoming more and more common
• Most often it is used in backdoors and rootkits
• Hacking drives the development of stealth techniques
• We can expect more tricky solutions day by day
The Hide'n'Seek has begun...

Questions?