ANTI-INCIENT RESPONSE

Nick Harbour, Principal Consultant
Nick Harbour - Bio

- 14 Years of Intrusion Analysis
- Mandiant (2006-2012)
  Co-developer of OpenIOC format
- Author of dcfldd, red curtain, IOCE, pe-scrambler, tcpxtract, findevil, etc....
- Taught Advanced Malware Analysis at BlackHat for the past 5 years
Outline

• Anti Live Response
• Anti Disk Forensics
• Anti Reverse Engineering
• Anti Incident Response
Anti-Live Response

- Avoiding detection by sysadmins and first responders
- Hiding from running process lists
  - `ps`, `top`, Windows process list
- Hiding network connections from view of common tools
  - `netstat`
Rootkits

• Originally Unix file replacement
• Mostly kernel-level post-1999

• Hides Attacker activity from live view
  – Process
  – Network connections
  – Resources

• Once Detectable, is a Red Herring
Process Injection

• Make good processes do evil things

• Avoids Having a “Malware Process” that needs hiding

• Typically Injects a DLL or block of code as a new thread
Windows Process Injection Mechanisms

- `VirtualAllocEx()`
- `VirtualProtect()`
- `WriteProcessMemory()`
- `CreateRemoteThread()`
- `SetWindowsHookEx()`
- `QueueUserAPC()`
Windows Process Injection

- Inject a DLL
  - Allocate and write the DLL name in the process
  - `CreateRemoteThread()` with `LoadLibrary()` as the thread start address
- `SetWindowsHookEx()` can also force a DLL load

- Inject shellcode
  - Allocate and write the shellcode in the process
  - `CreateRemoteThread()` with the start of the shellcode as the thread start address
  - Or `QueueUserAPC()` to launch code
Windows Thread Hijacking

- `SuspendThread()` on a thread
- Store its context with `GetThreadContext()`
- Make a new stack segment with `VirtualAllocEx()`
- Replace EIP and ESP with `SetThreadContext()`
- Resume the Thread with `ResumeThread()`
- Wait a for a period of time or unique event
- Set thread context back to its original state
- `ResumeThread()`
Unix Process Injection Mechanisms

- `ptrace()`
  - `PTRACE_POKEDATA`
  - `PTRACE_SYSCALL`
    - `sbrk()`
  - `PTRACE_DETACH`
Thread Hijacking Troubles

• Resuming a thread that is in the middle of a System Call

• Problem under Windows and Unix
Getting Around the Syscall Problem

- Windows: Detect if EIP is within NTDLL.DLL range, if so, resume thread and try again later.
- Unix: Detect if EIP is within range of a library object (if dynamically-linked), or disassemble previous instruction and determine if it was a syscall interrupt, and try again later
Hiding Network Activity

- Invoke the Internet Explorer COM object to communicate via HTTP through the IEXPLORE process

- `UrlDownloadToFile()` API function simplifies downloading functionality, calls IE COM object in the back end.
Anti-Forensics

- Avoiding Detection from Forensic Analysts
- Make it difficult to find the malware in the first place
- Obvious stuff I’m not going to talk about:
  - Hit sdelete like it owes you money
  - Timestomp
Evading Forensic Detection of Persistence

• Tools such as Autoruns examine Registry locations for persistence

• Avoid the Registry Like the Plague as much as possible
Service Replacement

• Replace Existing but useless service with a new DLL
  – Wzcsvc on servers

• Many IR shops don’t have the capability to audit at the DLL level
**DLL Search Order Hijacking**

- Causing legitimate programs to accidently load a malicious DLL instead of the real one
- Program expects the DLL to reside in System32
- Program does not run from System32
- DLL is not protected by KnownDlls Registry Key
- KnownDlls shortcuts the DLL search order by going directly to System32
- *https://blog.mandiant.com/archives/1207*
DLL Search Order (Safe Search mode)

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable.
DLL Search Order Hijacking

- Main Culprit: C:\Windows\explorer.exe

- Recursive Problem:
  - Ws2_32.dll is protected by KnownDlls
    - It loads iphlpapi.dll, which is not
Special Case Vulnerable DLLs

- System DLLs which perform `LoadLibrary()` to load an optional DLL during system startup
- No Evidence of loading in registry
- Disassembly of system binaries required.

- `Fxsst.dll`
  - Not the only case
Fxsst.dll

- A fax server DLL, used by Windows Explorer
- Who uses windows to send or receive faxes?
  - Oh, you do?
  - How is life in 1988?
    - Cool story bro
      - Why you disrespecting me bro?
        - I’m not your bro, pal
          - I’m not your pal, friend
            - I’m not your friend, guy
Fxsst.dll

- An optional DLL which is usually* not present on a system
- Even if you replace the legit one, no one will notice
  - Pro-Tip: Nobody uses fax services on Windows
Fxsst.dll

```c
if ( !g_hFaxLib )
{
    v5 = LoadLibraryW(L"fxsst.dll");
    g_hFaxLib = v5;
    g_pIsFaxMessage = 0;
    g_pFaxMonitorShutdown = 0;
    if ( v5 )
    {
        g_pIsFaxMessage = (int)GetProcAddress(v5, "IsFaxMessage");
        g_pFaxMonitorShutdown = GetProcAddress(g_hFaxLib, "FaxMonitorShutdown");
    }
}
```
Fxsst.dll

```c
while ( GetMessageW(&Msg, 0, 0, 0) )
{
    if ( !g_pIsFaxMessage || !g_pIsFaxMessage(&Msg) )
    {
        if ( !IsDialogMessageW(hWnda, &Msg) )
        {
            if ( !CSC_MsgProcess(&Msg) )
            {
                TranslateMessage(&Msg);
                DispatchMessageW(&Msg);
            }
        }
    }
}
```
Anti-Incident Response

• Disrupting, out-maneuvering or confusing the Incident Responders across the enterprise

• Makes Remediation a pain

• Essential to maintaining a long-time foothold on a network, even when detected
Anti-Incident Response Practices

- Maintain a wide variety of malware on the network
- Unique malware instances per host, or low population
Anti-Incident Response Practices

• Pre-deploy multiple stages of inactive backdoors

• Do so as quietly as possible

• Never touch these systems
Anti-Incident Response Practices

• Agile Lateral Movement

• Keep your total number of infected hosts moderate but not large, and keep them fresh

• Create a trail of activity at a faster pace than it takes to investigate
Anti-Incident Response Practices

- Chose busy servers as internal hop-points
  - Event logs cycle within minutes to hours
  - Network activity not out of place

- Chose enormous file servers as a data staging areas
Anti-Incident Response Practices

• Obscure the source of malware transmission

• Example:
  – Login via RDP
  – Paste .eml file text into notepad and save
  – Open .eml on victim host (outlook express)
  – Save attachment

• Example:
  – Lines of an input file for DOS debug inserted into a database
  – Dumped and executed with commandline tools already on the host
Anti-Incident Response Practices

• Replicate a Domain Controller
• Join it to the network
Anti-Incident Response Practices

• Establish a means to split-tunnel VPN clients for C2 communication

• Bypassing most network monitoring infrastructure
Anti-Reverse Engineering

• To prevent or delay discovery of malware or generation of detection mechanisms for the malware

• Can overlap with anti-forensics

• Target is still the responder, not the seasoned malware analyst
Packers

• The more extreme the packer is, the more detectable it is

• Maintain a large pool of custom packers
  – And don’t make unique section names
Packer Detection Woes

- Entropy analysis identifies many packed binaries
  - As well as a lot of non-packed binaries
- Requires a fair amount of expert manpower to review results on a single host
- Infeasible across an enterprise
Packer Detection Woes

• Who says your packed binary needs to be high entropy?

• Simple XOR packer defeats entropy detection
Packer Detection

- FindEvil
  - Not Packed:
  - Packed:
Hiding in Plain Sight

- Use string encoding only
- Delphi/C++
- Delphi Libraries shared with Borland Builder C++
- C++ MFC Default Template App: 232kb
Hiding in Plain Sight

; Attributes: thunk

; class AFX_MODULE_STATE * __stdcall AfxGetModuleState(void)
?AfxGetModuleState@@YGPAAFX_MODULE_STATE@@XZ proc near
jmp    ds:_imp_?AfxGetModuleState@@YGPAAFX_MODULE_STATE@@XZ ; AfxGetModuleState(void)
?AfxGetModuleState@@YGPAAFX_MODULE_STATE@@XZ_endp