Computer Forensics and Incident Response: Bringing Sexy Back

Jamie Butler

MANDIANT®
INTELLIGENT INFORMATION SECURITY

Agenda

- Traditional Forensic Analysis
  - Purpose
  - Why the disk is not enough
- More Common Threat Model
  - Infected Processes
  - Causes
  - Example Canvas
- Designing a Memory Analysis Tool
  - Requirements
  - Building blocks
  - Demonstration of the usefulness of Windbg
- Putting it all together
  - Demo – Were we just expoited by Canvas?

Traditional Forensic Analysis

- The focus was on disks where files were always written, so it worked
- This type of analysis usually involved a focused toolset that provided:
  - Parsing file systems (EXT2, XFS, NTFS, FAT32, etc.)
  - Hashing files
  - Analyzing file times
  - Recovering deleted files

Traditional Forensic Analysis

- Evidence of illegal or malicious behavior does not have to reside on disk
- Three examples
  - Canvas
    - Injects shellcode into other processes that provides a callback function
  - Metasploit Meterpreter
    - Injects arbitrary DLLs into a process's address space
  - Sliver rootkit
    - From a kernel driver injects any DLL into any process
    - Injects shellcode to steal any type of process handle

Code Injection Basics

- "Code Injection" refers to techniques used to run code in the context of an existing process
- Motivation:
  - Evasion: Hiding from automated or human detection of malicious code
    - IR personnel hunt for malicious processes
  - Impersonation: Bypassing restrictions enforced on a process level
    - Windows Firewall, etc
    - Pwdump, Sam Juicer

User Mode Injection Techniques

- Windows API
  - SetWindowsHookEx
  - OpenProcess
  - VirtualAlloc/Ex
  - WriteProcessMemory
- AppInit_Dll
- Detours

Kernel Process Injection

Two Halves of the Process

- User land processes are comprised of two parts
  - Kernel Portion
    - EPROCESS and KPROCESS
    - ETHREAD and KTHREAD
    - Token
    - Handle Table
    - Page Tables
    - Etc.

Two Halves of the Process

- User land Portion
  - Process Environment Block (PEB)
  - Thread Environment Block (TEB)
  - Windows subsystem (CSRSS.EXE)
  - Etc.

Kernel Process Injection Steps

- Must find suitable target
  - Has a user land portion
  - Has kernel32.dll and/or ntdll.dll loaded in its address space
  - Has an alterable thread (unless hijacking an existing thread)
- Allocate memory in target process
- Write the equivalent of “shellcode” that calls LoadLibrary
- Cause a thread in the parent to execute newly allocated code
  - Hijack an existing thread
  - Create an APC

Allocate memory in parent process

- Change virtual memory context to that of the target
  - KeAttachProcess/KeStackAttachProcess
  - ZwAllocateVirtualMemory
    - (HANDLE) -1 means current process
    - MEM_COMMIT
    - PAGE_EXECUTE_READWRITE

Creating the Shellcode

- "shellcode" that calls LoadLibrary
  - Copy function parameters into address space
  - Pass the address of function parameters to calls
  - Can use the FS register
    - FS contains the address of the TEB
    - TEB has a pointer to the PEB
    - PEB has a pointer to the PEB_LDR_DATA
    - PEB_LDR_DATA contains all the loaded DLLs

APC

- Cause a thread in the parent to execute newly allocated code - Create an APC
  - Threads can be notified to run an Asynchronous Procedure Call (APC)
  - APC has a pointer to code to execute
  - To be notified, thread should be Alertable

Finding an Alertable Thread

```
PETHREAD FindAlertableThread(PEPROCESS eproc)
{
    PETHREAD start, walk;

    if (eproc == NULL)
            return NULL;
    start = *(PETHREAD *)((DWORD)eproc + THREADOFFSET);
    start = (PETHREAD)((DWORD)start - THREADFLINK);
    walk = start;

    do
    {
            DbgPrint("Looking at thread 0x%x\n",walk);

            if (*(PUCHAR)((DWORD)walk + ALERTOFFSET) == 0x01)
                    return walk;
            walk = *(PETHREAD *)((DWORD)walk + THREADFLINK);
            walk = (PETHREAD)((DWORD)walk - THREADFLINK);
    }while (walk != start);

    return NULL;
}
```

MANDIANT®
INTELLIGENT INFORMATION SECURITY

How do we begin to detect this? Memory Analysis

- Requirements
  - No use of APIs to gather data.
  - Ability to use any analysis solution on both live memory and offline memory image dumps.
    (Implies the ability to do all memory translation independently.)
  - Do not require PDB symbols or any other operating specific information.

Steps to Memory Analysis

- Ability to access physical memory
- Derive the version of the OS – important to know how to interpret raw memory
- Find all Processes and/or Threads
- Enumerate File Handles, DLLs, Ports, etc.

Steps to Memory Analysis

- Virtual to Physical Address Translation
  - Determine if the host uses PAE or non-PAE
  - Find the Page Directory Table – process specific
  - Translate prototype PTEs
  - Use the paging file

Derive the version of the OS

- Find the System Process
  - Allows the derivation of:
    - The major operating system version in question
    - The System Page Directory Table Base
    - HandleTableListHead
    - Virtual address of PsInitialSystemProcess
    - PsActiveProcessHead
    - PsProcessType

Operating System Version

- Find the System image name
- Walk backwards to identify the Process Block
- The spatial difference between major versions of the OS is enough to begin to tell us about the operating system version

Operating System Version

- Drawback: Ghosts
  - There can be more than one System Process
    - Open a memory crash dump in Windbg
    - Run a Windows operating system in VMWare
  - Solution:
    - Non-paged kernel addresses are global
    - We know the virtual address of PsActiveProcessHead
    - PsActiveProcessHead and other kernel addresses should be valid and present (translatable) in both live or dead memory

Memory Translation

- PAE vs non-PAE
  - Different ways to interpret the address tables
  - The sixth bit in the CR4 CPU register determines if PAE is enabled
  - Problem: We do not have access to CPU registers in memory analysis
  - Solution?
    - Kernel Processor Control Region -> KPCRB -> KPROCESSOR_STATE -> KSPECIAL_REGISTERS -> CR4

Memory Translation

- CR4 Heuristic
  - Page Directory Table Base and the Page Directory Table Pointer Base look very different.
- CR3 is updated in the KPCR
  - This can be used to identify a valid Page Directory Table
  - The Page Directory can be used to validate the PsActiveProcessHead

Enumerating Injected DLLs

- Problem:
  - APIs lie.
  - Malware can unlink from the PEB_LDR_DATA lists of DLLs
- Solution:
  - Virtual Address Descriptors (VADs)

VADs

- Self balancing binary tree [1]
- Contains:
  - Virtual address range
  - Parent
  - Left Child and Right Child
  - Flags – is the memory executable
  - Control Area

1. Russinovich, Mark and Solomon, Dave, *Microsoft Windows Internals*, Microsoft Press 2005

A Memory Map to a Name

- VAD contains a CONTROL_AREA
- CONTROL_AREA contains a FILE_OBJECT
- A FILE_OBJECT contains a UNICODE_STRING with the filename

- We now have the DLL name

Leveraging existing tools

- Memory Acquisition
  - dd by George Garner http://gmgsystemsinc.com/fau/
- Memory Translation
  - IA-32 Intel Architecture Software Developer's Manual, Volume 3, Chapter 3
- Finding Things in Memory
  - Windbg
    - Processes
    - Drivers
    - Etc.
  - VADs
    - Russinovich, Mark; David Solomon ***Microsoft Windows Internals***, (Fourth Edition), Microsoft Press December, 2004.
    - *Dolan-Gavitt, Brendan. The VAD Tree: A Process-Eye View of Physical Memory* Digital Investigation 4S (2007) S62-S64

Demo – catching Dave Aitel

Conclusion

MANDIANT®
INTELLIGENT INFORMATION SECURITY

Questions?

- Email: jamie.butler AT mandiant.com
- Job searchers: always looking for talented people to work with.