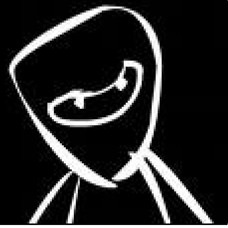
Stealth Secrets of the Malware Ninjas

By Nick Harbour



MANDIANT™

INTELLIGENT INFORMATION SECURITY

Overview

- Intro
- Background Info
  - Malware
  - Forensics and Incident Response
  - Anti-Forensics
  - Executables
- Stealth Techniques
  - Live System Anti-Forensics
    - Process Camouflage
    - Process Injection
    - Executing Code from Memory
  - Offline Anti-Forensics
    - File Hiding
    - Trojanizing
    - Anti-Reverse Engineering

There will be something for the "Good Guys" near the end

- A brand new malware scanning tool

Introduction

- This presentation will cover a variety of stealth techniques currently used by malware in the field.
- Many of the techniques are based on malware studied during MANDIANT’s incident experiences.

Introduction

- The purpose of this talk is to discuss malware stealth techniques other than Rootkits.
- The majority of the material is designed to teach the "Bad Guys" some practical real world techniques to fly beneath the radar.
- For the "Good Guys", learning these malicious techniques will help prepare you to identify and counter malware threats.

Prerequisites

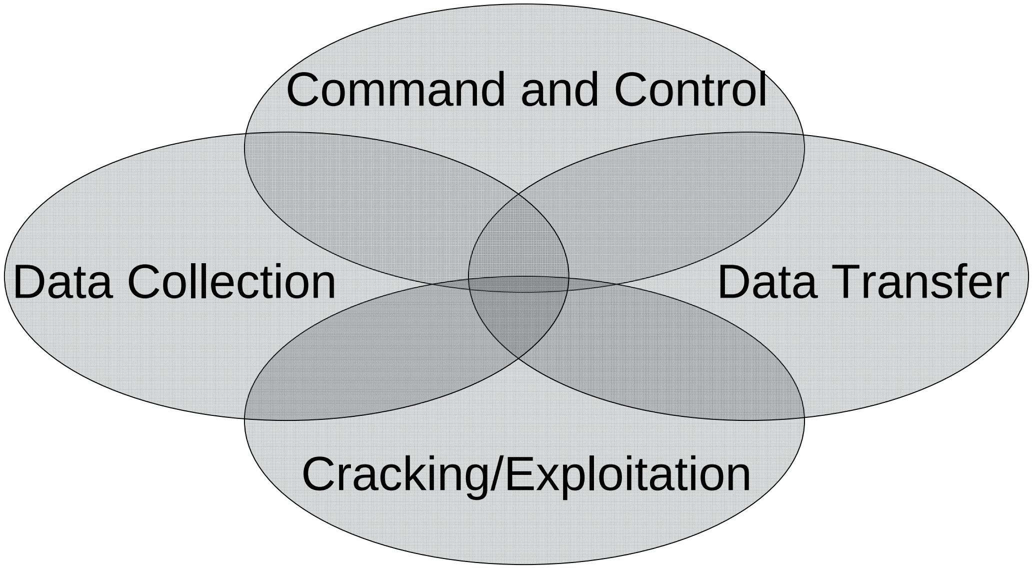
- There's something for everyone!
- The material we will cover the range from basic computing concepts to machine code.
- We will primarily be discussing techniques for Windows, but Linux will also discussed at an advanced level.

Background Information

MANDIANT™
INTELLIGENT INFORMATION SECURITY

Malware

- In intrusion incidents, malware is frequently found in several inter-related families of tools.
- Often found in redundant layers for failover or bootstrapping.



Malware

- In practice, stealth techniques are most often employed to protect an intruder’s command and control mechanism
- These often require persistence which poses a risk of discovery
- Command and Control is the keys to the intruder’s newly acquired kingdom

Forensics and Incident Response

- Traditional Computer Forensics involves examining the contents of computer media for evidence of a crime.
- A suspect system is powered off, the storage media is duplicated then analyzed with in a controlled environment

Forensics and Incident Response

- Incident Response is a specialized discipline which expands upon the role of traditional Computer Forensics.
- Critical data is collected from live systems and network traffic in addition to storage media.
- Incident Response techniques are typically used for Computer Intrusion incidents.

Anti-Forensics

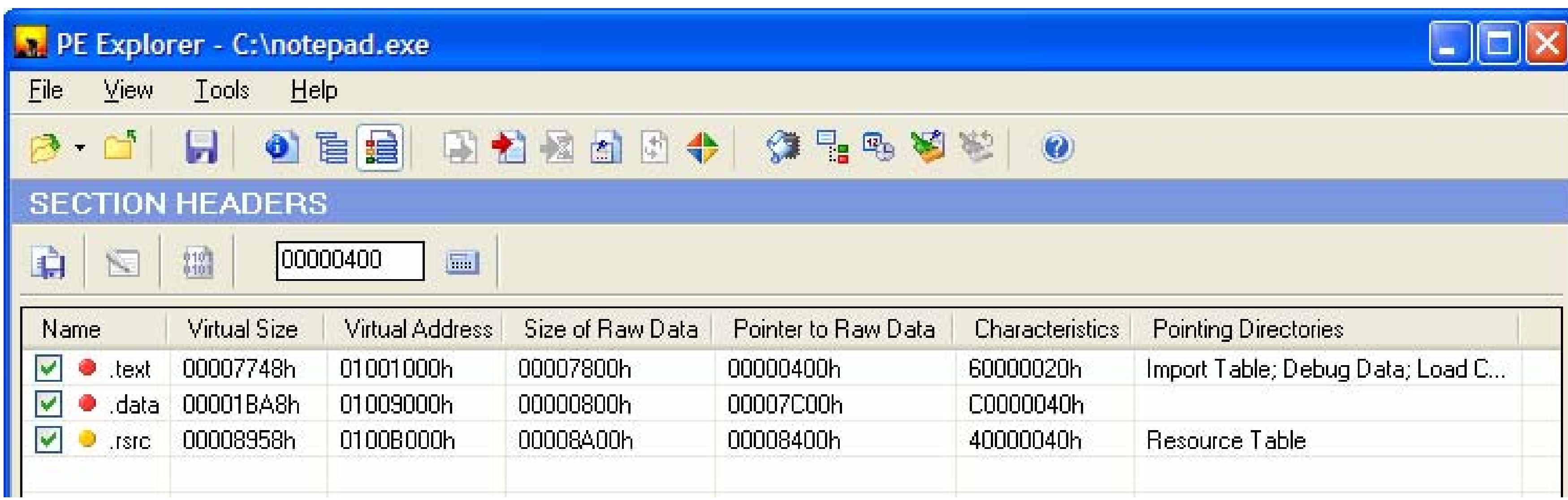
- Anti-Forensics is the practice of avoiding or thwarting detection through forensics, incident response methods or general use.
- Due to increasing levels of sophistication and a growing pool of reverse engineering talent, anti-forensics is growing in importance because it prevents malware from ever being found.

Executables

- Microsoft’s PE file format and ELF under Linux are popular examples.
- Most modern formats are quite similar in principle.
- Dynamic Libraries such as .DLL files often use the same file formats as executables.
- In addition to header data, objects called sections are the building blocks of executables

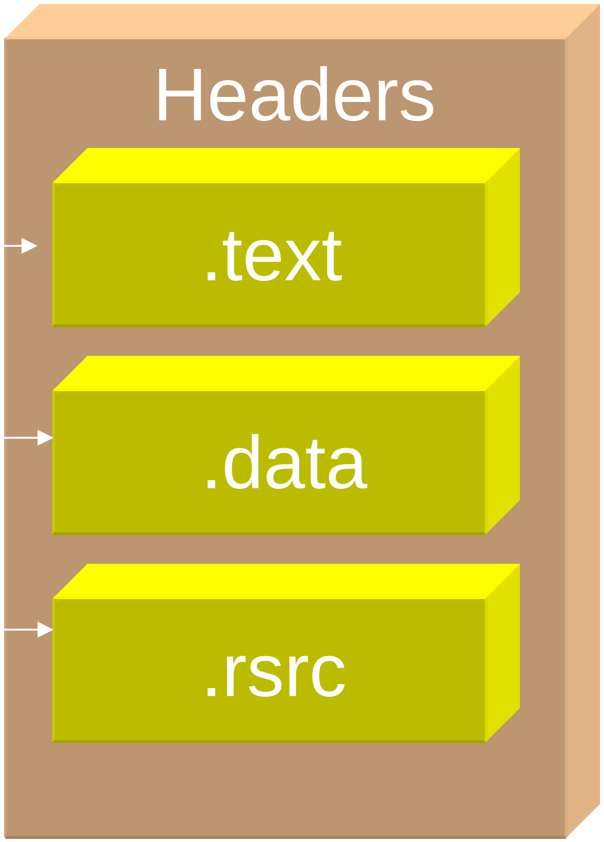
Executables

- Sections contain executable code, data, debugging information, resources and additional metadata used by the program.



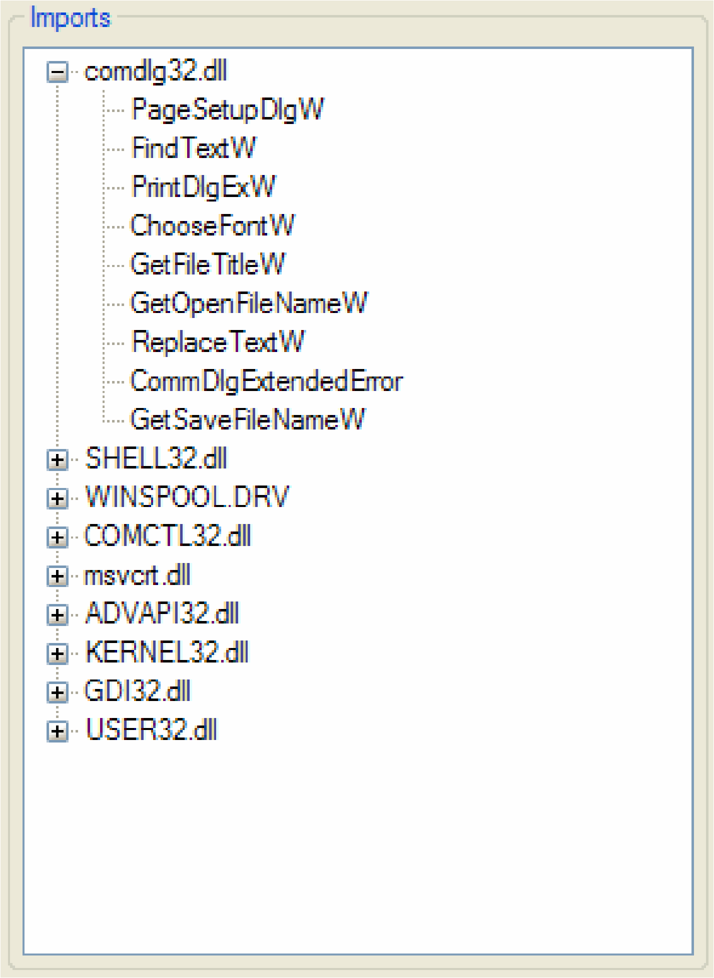
Structure of notepad.exe

- Contains the executable code
- Contains the initialized data
- Contains resources (icons, multi-language strings, etc..)



Imports and Exports

- In order to use code in an external dynamic library, executables contain a list of libraries and associated symbols it needs.
- Similarly, executables and dynamic libraries may list specific functions and variable names in a special Export table so they may be imported into other programs.



MANDIANT™

Executable Loading

- Each section object in the executable file will be loaded into memory by the operating system when the program is run.
- Every Dynamic Library listed in the program's import table is then mapped into memory.
- Imports required by each Dynamic Library are also imported, recursively.

Loaded Executable Memory Space

notepad.exe

01000000	00001000	notepad		PE header	Imag	R	RWE
01001000	00008000	notepad	.text	code,import	Imag	R	RWE
01009000	00002000	notepad	.data	data	Imag	R	RWE
0100B000	00009000	notepad	.rsrc	resources	Imag	R	RWE
5AD70000	00001000	UxTheme		PE header	Imag	R	RWE
5AD71000	00030000	UxTheme	.text	code,import	Imag	R	RWE
5ADA1000	00001000	UxTheme	.data	data	Imag	R	RWE
5ADA2000	00004000	UxTheme	.rsrc	resources	Imag	R	RWE
5ADA6000	00002000	UxTheme	.reloc	relocations	Imag	R	RWE
5CB70000	00001000	ShimEng		PE header	Imag	R	RWE
5CB71000	0000E000	ShimEng	.text	code,import	Imag	R	RWE
5CB7F000	00014000	ShimEng	.data	data	Imag	R	RWE
5CB93000	00001000	ShimEng	.rsrc	resources	Imag	R	RWE
5CB94000	00002000	ShimEng	.reloc	relocations	Imag	R	RWE
629C0000	00001000	LPK		PE header	Imag	R	RWE
629C1000	00005000	LPK	.text	code,import	Imag	R	RWE
629C6000	00001000	LPK	.data	data	Imag	R	RWE
629C7000	00001000	LPK	.rsrc	resources	Imag	R	RWE
629C8000	00001000	LPK	.reloc	relocations	Imag	R	RWE
6F880000	00001000	AcGenral		PE header	Imag	R	RWE
6F881000	00032000	AcGenral	.text	code,import	Imag	R	RWE
6F8B3000	00009000	AcGenral	.data	data	Imag	R	RWE
6F8BC000	00188000	AcGenral	.rsrc	resources	Imag	R	RWE
6FA44000	00006000	AcGenral	.reloc	relocations	Imag	R	RWE
73000000	00001000	WINSPOOL		PE header	Imag	R	RWE
73001000	00020000	WINSPOOL	.text	code,import	Imag	R	RWE
73021000	00002000	WINSPOOL	.data	data	Imag	R	RWE
73023000	00001000	WINSPOOL	.rsrc	resources	Imag	R	RWE
73024000	00002000	WINSPOOL	.reloc	relocations	Imag	R	RWE
74D90000	00001000	USP10		PE header	Imag	R	RWE
74D91000	00044000	USP10	.text	code,import	Imag	R	RWE
74DD5000	00010000	USP10	.data	data	Imag	R	RWE
74DE5000	00002000	USP10	Shared		Imag	R	RWE
74DE7000	00012000	USP10	.rsrc	resources	Imag	R	RWE
74DF9000	00002000	USP10	.reloc	relocations	Imag	R	RWE
76390000	00001000	IMM32		PE header	Imag	R	RWE
76391000	00015000	IMM32	.text	code,import	Imag	R	RWE
763A6000	00001000	IMM32	.data	data	Imag	R	RWE
763A7000	00005000	IMM32	.rsrc	resources	Imag	R	RWE
763AC000	00001000	IMM32	.reloc	relocations	Imag	R	RWE
763B0000	00001000	comdlg32		PE header	Imag	R	RWE
763B1000	00030000	comdlg32	.text	code,import	Imag	R	RWE
763E1000	00004000	comdlg32	.data	data	Imag	R	RWE
763E5000	00011000	comdlg32	.rsrc	resources	Imag	R	RWE
763F6000	00003000	comdlg32	.reloc	relocations	Imag	R	RWE
769C0000	00001000	USERENV		PE header	Imag	R	RWE
769C1000	0009F000	USERENV	.text	code,import	Imag	R	RWE
76A60000	00002000	USERENV	.data	data	Imag	R	RWE
76A62000	0000A000	USERENV	.rsrc	resources	Imag	R	RWE
76A6C000	00007000	USERENV	.reloc	relocations	Imag	R	RWE

Programmatics

- Memory regions (sections) may be added, manipulated or removed after the initial program load using the Win32 API
  - `VirtualAllocEx()`, `VirtualFreeEx()`, `MapViewOfFile()`, `WriteProcessMemory()` to name a few.
- Importing functionality from Dynamic Libraries may also be accomplished easily through the Win32 API
  - `LoadLibrary()`, `GetProcAddress()`

Stealth Techniques

MANDIANT™
INTELLIGENT INFORMATION SECURITY

Live System Anti-Forensics

- Live System Anti-Forensics is specifically concerned with concealing the presence of running malware.
- While Rootkits play decisive role in this field, they are a field unto themselves and receive ample treatment elsewhere.
- We will cover a range of techniques other than Rootkits.

Process Injection

- As the name implies, injects code into another running process.
- Target process obliviously executes your malicious code.
- Conceals the source of the malicious behavior.
- Can be used to bypass host-based firewalls and many other process specific security mechanisms.

Hook Injection

- The easiest method to achieve process injection on a windows host is via the Windows Hooks mechanism.
- Allows you to add specify a piece of code to run when a particular message is received by a Windows application.

Hook Injection

- The `SetWindowsHookEx()` Win32 API call causes the target process to load a DLL of your choosing into its memory space and select a specified function as a hook for a particular event.
- When an appropriate event is received, your malicious code will be executed by the target process.

Windows Message Hooks

Hook Injection Code

```
HANDLE hLib, hProc, hHook;

hLib = LoadLibrary("evil.dll");

hProc = GetProcAddress(hLib,

 "EvilFunction");

hHook =
 SetWindowsHookEx(WH_CALLWNDPROC,
                        hProc, hLib,
 0);
```

Library Injection

- The next easiest method of process injection involves creating a new thread in the remote process which loads your malicious library.
- When the library is loaded by the new thread, the `DllMain()` function is called, executing your malicious code in the target process.

Library Injection

- To create a new thread in a remote process we use the Win32 API call `CreateRemoteThread()`.
- Among its arguments are a Process Handle, starting function and an optional argument to that function.

Library Injection

- We must set our starting function to `LoadLibrary()` and pass our evil library name to it as the optional argument.
- Since the function call will be performed in the remote thread, the argument string (our evil library name) must exist within that process' memory space.
- To solve that problem we can use `VirtualAllocEx()` to create space for the string in the new process.
- We can then use `WriteProcessMemory()` to copy the string to the space in the new process.

Library Injection Code

```
char libPath[] = "evil.dll";
char *remoteLib;
HMODULE hKern32 = GetModuleHandle("Kernel32");
void *loadLib = GetProcAddress(hKern32, “LoadLibraryA”);


remoteLib = VirtualAllocEx(hProc, NULL,
      sizeof (libPath), MEM_COMMIT, PAGE_READWRITE);

WriteProcessMemory(hProc, remoteLib, libPath, sizeof
  libPath, NULL);

CreateRemoteThread(hProc, NULL, 0, loadLib,
                   remoteLib, 0, NULL));
```

Direct Injection

- Direct injection involves allocating and populating the memory space of a remote process with your malicious code.
  - `VirtualAllocEx()`
  - `WriteProcessMemory()`
- This could be a single function of code or and entire DLL (much more complicated).

MANDIANT™

Direct Injection

- `CreateRemoteThread()` is then used to spawn a new thread in the process with a starting point of anything you would like.
- The most powerful, flexible technique.
- Also the most difficult.
- For example, it takes more code than one may fit on a slide.

Process Camouflage

- A cleverly named process is often enough to fly beneath the radar and avoid immediate detection.
- Slight variations of legitimate operating system processes or legitimate names whose binaries reside in a non-standard location are the staples of camouflage.
- Take variations on commonly running processes.
- A reasonably well named service will also suffice.

Example Name Variations

- Svchost.exe and spoolsv.exe make the best targets because there are usually several copies running in memory. One more will often go unnoticed.

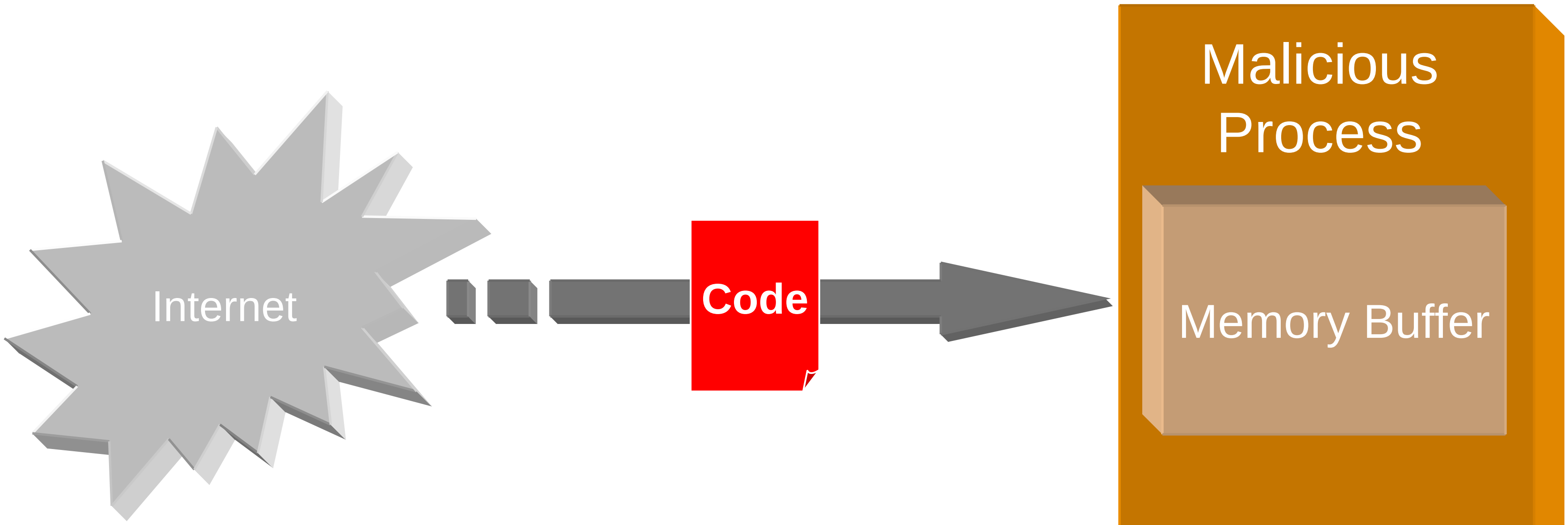
- svhost.exe
- svcshost.exe
- spoolsvc.exe
- spoolsvr.exe
- scardsv.exe
- scardsvc.exe
- lsasss.exe

Executing Code from Memory

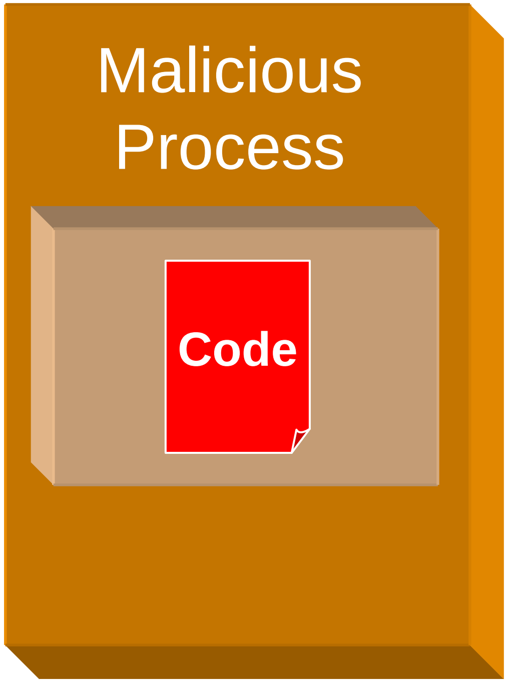
- The ability to execute code directly from memory means that the malicious code never has to reside on the hard drive
- If it is never on the hard drive, it will more than likely be missed during a forensic acquisition.

Executing Code from Memory

- Memory buffer to be executed will most likely be populated directly by a network transfer.



Executing Code from Memory



- The definition of code here extends beyond machine instructions to any program logic
  - Interpreted Code
  - Bytecode Compiled Code
  - Machine Code
  - Executables

MANDIANT™

Embedded Languages

- The easiest approach is to accept code in the form of an interpreted language.
- Interpreted languages are often designed to be easily embedded.
- A large number of interpreted languages contain some equivalent of an `exec()` or `eval()` function, which can execute source code contained in a variable

Embedded Languages

- Malware containing an embedded language forces a potential reverse-engineer into deciphering the structure of the embedded language before they can begin to fully decipher your malicious logic.
- Byte code compiled languages add another layer of obscurity to the process.

Embedded Languages

- A large number of custom languages used by malware captured in the field turn out to be nothing more than cheap x86 knockoffs.
- With little extra effort you can add obscurity
  - Reverse the stack
  - Extensible instruction set
- Really screw ‘em up, embed Lisp!

Malvm

- An example embeddable implementation of a slightly more sophisticated x86 knockoff.
- Soon to be released*!
- Implements a forward stack and extensible instruction set.
- Low level instructions to `LoadLibrary()` and `GetProcAddress()`

*Will be published at `http://www.nickharbour.com`

Executing Code from Memory

- Machine code may also be executed from a buffer. Both position independent shellcode as well as executable files.
- The ability to execute arbitrary executable files from a memory buffer is extremely powerful because it allows existing malware tools to be downloaded and executed in a pure anti-forensic environment.

Windows Userland Exec

- A technique was introduced by Gary Nebbett to launch executables from a memory buffer under Win32 systems.
- Nebbett’s technique involved launching a process in a suspended state then overwriting its memory space with the new executable.
- Referred to as Nebbett’s Shuttle

Nebbett's Shuttle Abstract Code

- **CreateProcess**(…,"cmd",…,***CREATE_SUSPEND***,…);
- **ZwUnmapViewOfSection**(…);
- **VirtualAllocEx**(…,ImageBase,SizeOfImage,…);
- **WriteProcessMemory**(…,headers,…);

- for (i=0; i < NumberOfSections; i++) {
  - **WriteProcessMemory**(…,section,…);

  }

- **ResumeThread**();

MANDIANT™

Nebbett's Shuttle Step-by-Step

- **`CreateProcess`**`(…,"cmd",…,`***`CREATE_SUSPEND`***`,…);`
  - Creates a specified process ("cmd" in this example) in a way such that it is loaded into memory but it is suspended at the entry point.
- **`ZwUnmapViewOfSection`**`(…);`
  - Releases all the memory currently allocated to the host process ("cmd").
- **`VirtualAllocEx`**`(…,ImageBase,SizeOfImage,…);`
  - Allocate a an area to place the new executable image in the old process space.

Nebbett's Shuttle Step-by-Step

- **WriteProcessMemory**(…,headers,…);
  - Write the PE headers to the beginning of the newly allocated memory region.
- ```
  for (i=0; i < NumberOfSections; i++) {
     WriteProcessMemory(…,section,…);
  }
  ```
  - Copy each section in the new executable image to its new virtual address.
  ```

Nebbett’s Shuttle Step-by-Step

- **`ResumeThread(…);`**

- Once the remote process environment has been completely restored and the entry point pointed to by the EIP, execution is resumed on the process.
- The process still appears as “cmd” in a task list but is now executing our own malicious content.

Additional Benefits

- The code we replace “cmd” with is still running as “cmd”.
- This can be used to present a cover story.
- The malicious code inherits any privileges of the target code, for example exception from the host-based firewall if that is the case.

Finding a UNIX Equivalent to Nebbett's Shuttle

- Unfortunately UNIX does not provide a similar API for remote process similar to Win32.
- Direct portability is not an option.

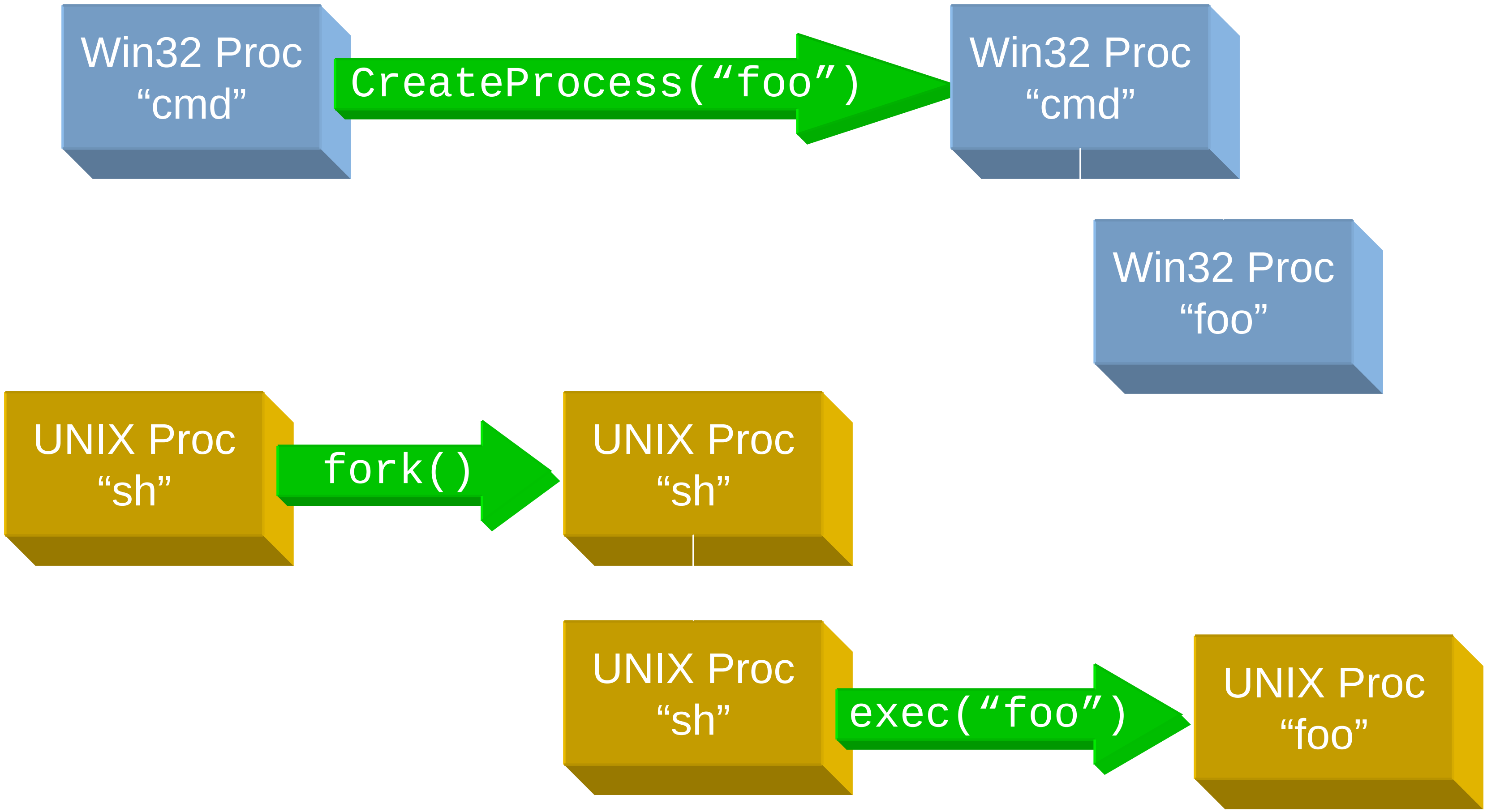
- Two existing techniques from the Grugq.
- New technique

Userland `exec()`

- A technique was developed by the Grugq to function similar to the `execve()` system call but operate entirely in user space.
- The `exec()` family of functions in UNIX replaces the current process with a new process image.
- `fork()` and `exec()` are the key functions for UNIX process instantiation.

MANDIANT™

Windows vs. UNIX Process Invocation



MANDIANT™

Userland `exec()`

- Unlike Nebbett's Shuttle, which simply manipulated a suspended processes memory space, Userland exec() for UNIX must load a new process into its own memory space.

Userland `exec()`

- Uses `mmap()` to allocate the specific memory area used by the program.
- Copies each section into the new memory region.
- Also loads a program interpreter if one is specified in the ELF header (Can be a Dynamic Linker).
- Sets up the heap for the new program using `brk()`.
- Constructs a new stack
- Jumps to the new entry point!

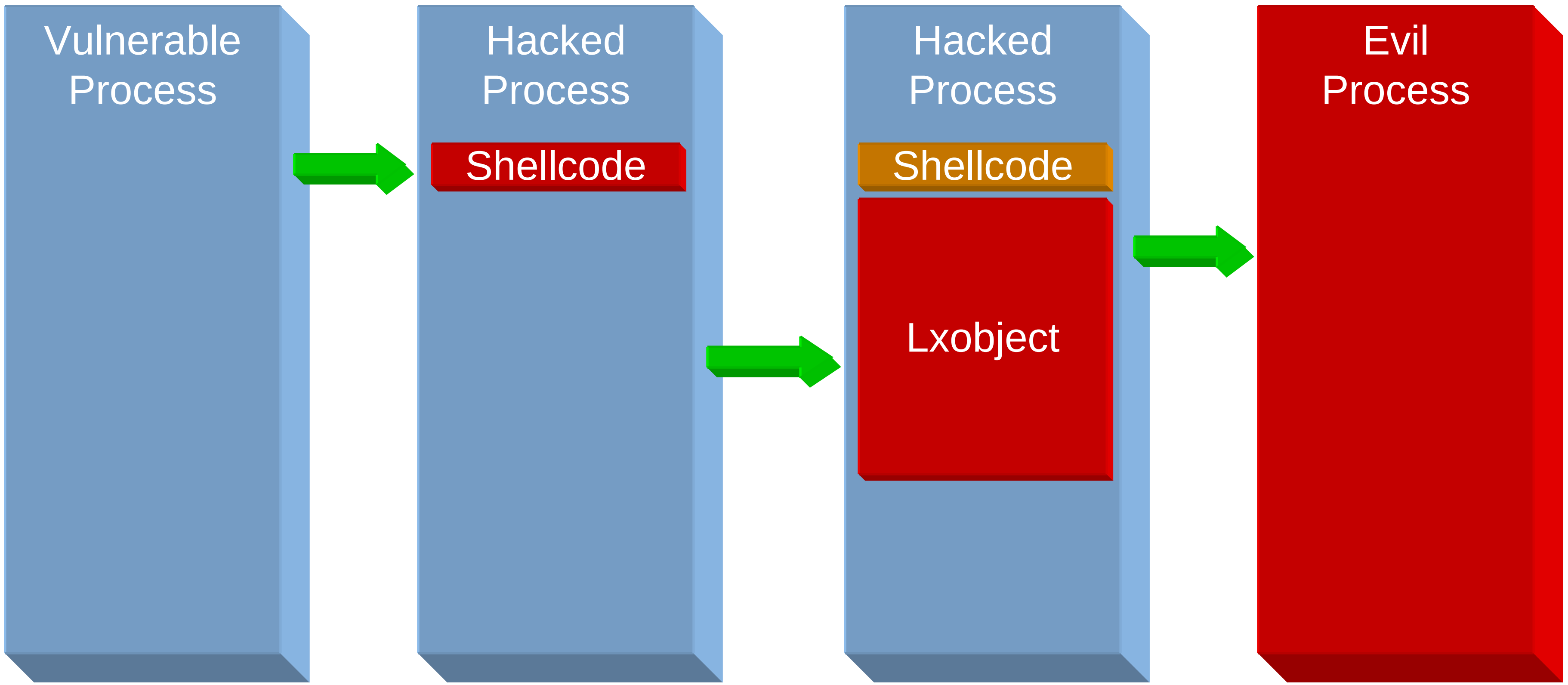
Shellcode ELF Loader

- Building upon his earlier Userland `exec()` code, the grugq later developed a technique to load an ELF binary into a compromised remote process.
- This technique was detailed in Phrack Magazine Volume 0x0b, Issue 0x3f.

Shellcode ELF Loader

- A stub of shellcode is inserted in a vulnerable process.
- The minimalist shellcode simply downloads a package called an lxobject.
  - An lxobject is a self loading executable package. It contains the ELF executable, stack context and shellcode to load and execute the program in the current process.
- The shellcode and jumps to a second phase of shellcode contained within the lxobject.

Shellcode ELF Loader Process



Fresh Ideas

- The current techniques still don’t quite fill the boots of Nebbett’s Shuttle.
- We are still locked into exploiting a vulnerable host process or forking from the process doing the infecting.
- We can expand our anti-forensic possibilities if we had the ability to execute our memory buffer as any other process we want.

UNIX Process Infection

- The only interface on most UNIX systems which allows modification to another processes memory or context is the debugging interface `ptrace()`.
- By creating a program which acts as a debugger we can infect other processes with arbitrary code.

ptrace()

```
#include <sys/ptrace.h>
long ptrace(enum __ptrace_request request,
            pid_t pid, void *addr, void *data);
```

- Has the ability attach to remote processes or debug child processes.
- Can manipulate arbitrary memory and registers as well as signal handlers.

How Most Debuggers Work

- `ptrace()` and most debuggers operate by inserting a breakpoint instruction.
- The breakpoint instruction in x86 is “`int 3`” in assembly language which translates to the machine code values of “`CD 03`”.
- Software interrupts transfer control back to the debugging process.
- For most software debuggers on any operating system, the relationship between debugger and debugee is a relationship maintained by the kernel.

A Simple Debugger

```
switch (pid = fork()) {
case -1:       /* Error */
        exit(-1);
case 0:        /* child process */
        ptrace(PTRACE_TRACEME, 0, 0, 0);
        execl("foo", "foo", NULL);
        break;

default:       /* parent process */
        wait(&wait_val);
        while (wait_val == W_STOPCODE(SIGTRAP)) {
           if (ptrace(PTRACE_SINGLESTEP, pid, 0, 0) != 0)
                perror("ptrace");
           wait(&wait_val);
        }
}
```

MANDIANT™

UNIX Infection via Debugging

- By using the `ptrace()` interface we can insert machine code to take control over a process.
- We will use this technique to achieve a UNIX version of Nebbett's Shuttle, but it can also be used for other forms of run-time patching.

The Technique

- Insert a small stub of code which allocates a larger chunk of memory.
- The last instruction in this stub code is the software breakpoint instruction to transfer control back to the debugging process.

- Limitations are that the process you are infecting needs to have enough memory allocated past where the instruction pointer is pointing to support the shellcode. Approximately 40 bytes.

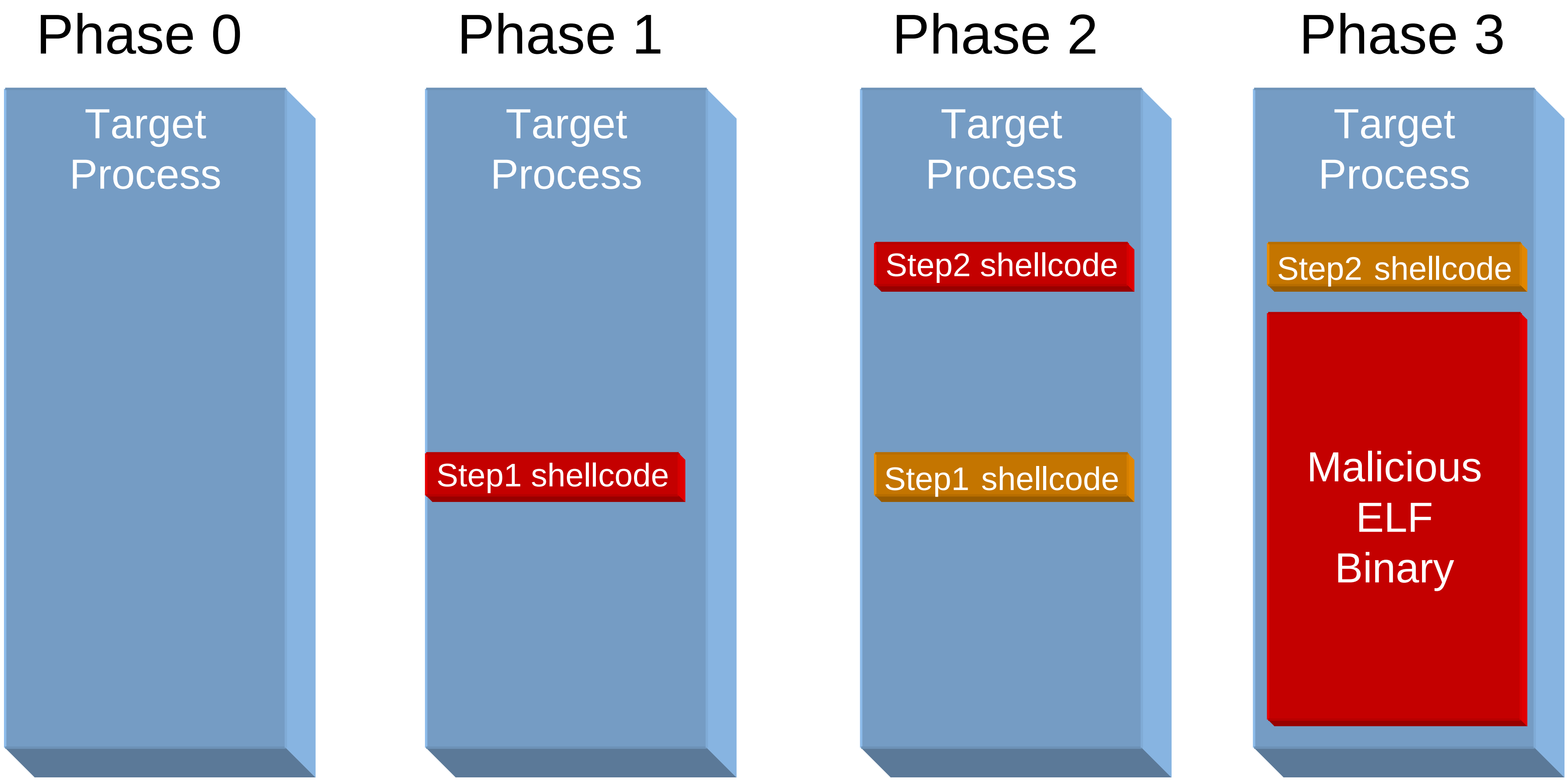
The Technique

- The debugging process then inserts code to clean up the old process memory space and allocate room for the new image in its ideal location.
- The code also sets up the heap for the new process.
- The last instruction in this code is a software breakpoint.
- The debugee is then resumed so that this code may execute and allocate memory.

The Technique

- When control returns to the debugger, it copies the new executable into the process memory in the appropriate manner.
- The debugger process modifies the stack and registers for the process as necessary
- Point at the new entry point.
- Detach.

The Technique



MANDIANT™

Offline Anti-Forensics

- Offline Anti-Forensics are measures taken to eliminate residual disk evidence of an activity.

- Started when ancient hackers discovered that they could delete log or alter log files to cover their tracks.

File Hiding

- Altering of file timestamps to mask its relation to the incident. See Metasploit's Timestomper.
- Alternate data streams under NTFS, though lame, are still being used with surprising effectiveness.
- When a need arises to hide a file, such as a malware binary, there are many places right on the filesystem which are often overlooked.

File Hiding

- C:\Windows\Downloaded Program Files
  - Masks the filenames of all its contents
- System Restore Points
  - Contain Backup copies of files and binaries in certain locations. A good needle in the haystack location.
- C:\Windows\System32
  - The classic haystack for your needle
  - Be warned, Your malware might get backed up to a restore point!

Trojanizing

- To leave your malware on a system without leaving an executable on the filesystem it may be a viable option to simply trojanize an existing executable on the system.
- This approach will bypass a large number of computer forensics examiners.
- Persistence may be established by trojanizing a binary which is loaded on system boot.

The Executable Toolkit

- A toolkit for performing a variety of tasks against executable files
  - Wrapping an executable with a fixed command line or standard input
  - Wrapping an executable with fixed DLLs
  - Manipulating sections
  - Trojanizing through entry point redirection
  - Trojanizing through TLS
  - Detours Support

*Available at http://nickharbour.com or SourceForge.

Anti-Reverse Engineering

- If you are unlucky enough to be caught by a computer forensic examiner who isn't afraid to peek inside a binary it will be important for you to conceal your true identity.
- Packers are the primary method used today.

Packers

- Most low-level reverse engineers know only how to use automated tools to unpack.
- A custom packer, even a simplistic one, will likely defeat the low-level reversers.
- Custom packed binaries are less likely to be identified at all.

- An example custom packer with source code is included with the Executable Toolkit (exetk) package.

Something for the Good Guys

- Packer detection tools today such as PEiD are easily fooled.
- We have developed something better.
- Mandiant Red Curtain.
  - A tool for detecting packed and anomalous binaries.
  - Uses section based entropy, imports and anomalies to compute a score.
  - Available at http://www.Mandiant.com

MANDIANT™

Mandiant Red Curtain

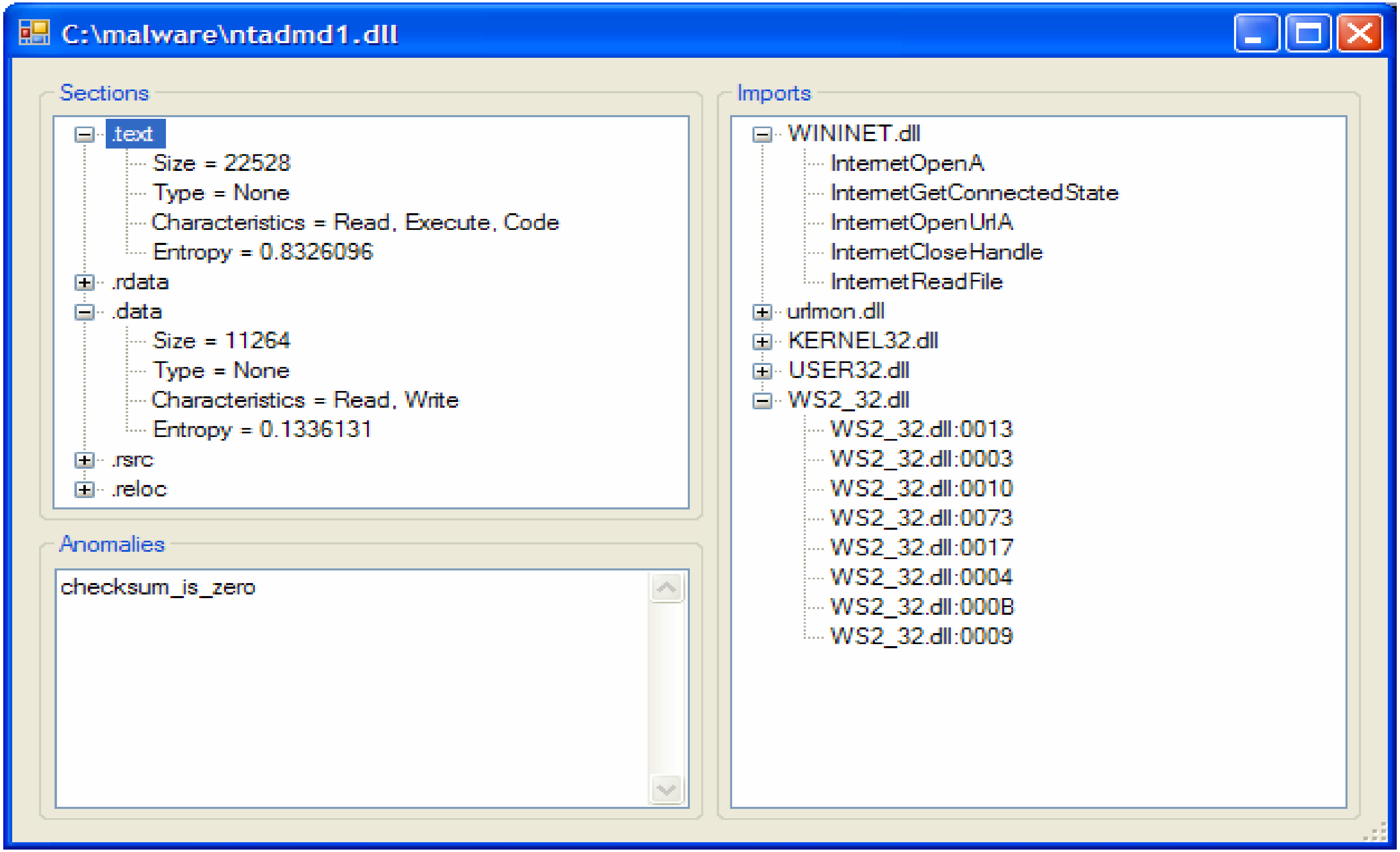
Mandiant Red Curtain v1.0 - [Unsaved]

File Edit Options Help

Score	File	Entry Point Signature	Size	Entropy	Code Entropy	Anomaly Count	Details
4.925	C:\malware\scarbrauv-restorepoint\A0085299.exe		25600	1.0228...	0	2	Details
4.925	C:\malware\restorepoint-malware\fp.exe		25600	1.0228...	0	2	Details
3.825	C:\malware\restorepoint-malware\fp-dump.bin		57856	0.8275...	0	4	Details
3.825	C:\malware\Malware Collection\fp-dump.bin		57856	0.8275...	0	4	Details
3.706	C:\malware\hlp.hlp.memdump.exe	LCC Win32 v1.x	75264	0.9109...	0	2	Details
1.310	C:\malware\2007_05_18\Malware\A_new_mosaicin...		7767	0.7248...	0	2	Details
1.310	C:\malware\Tag4-NDHMC4SINF03\extracted-winsys...		7767	0.7248...	0	2	Details
1.310	C:\malware\Tag4-NDHMC4SINF03\extracted-A0047...		7767	0.7248...	0	2	Details
1.310	C:\malware\2007_05_18\Malware\A Criticism on FA...		7767	0.7248...	0	2	Details
1.310	C:\malware\Malware Collection\winsys.exe		7767	0.7248...	0	2	Details
1.025	C:\malware\restorepoint-malware\pul.exe	UPX-Scrambler RC v1.x	27648	1.1362...	0	1	Details
1.025	C:\malware\scarbrauv-restorepoint\A0085291.exe	UPX-Scrambler RC v1.x	27648	1.1362...	0	1	Details
1.025	C:\malware\Malware Collection\pul.exe	UPX-Scrambler RC v1.x	27648	1.1362...	0	1	Details
0.956	C:\malware\restorepoint-malware\ok\wrhome.exe	UPX-Scrambler RC v1.x	14336	1.0745...	0	1	Details
0.956	C:\malware\restorepoint-malware\wrhome-WINRAR_...	UPX-Scrambler RC v1.x	14336	1.0745...	0	1	Details

1 of 161 selected.

Mandiant Red Curtain



Thank You!

Nick Harbour
MANDIANT
Senior Consultant
675 North Washington St, Suite 210
Alexandria, VA 22314
703-683-3141
NickHarbour@gmail.com

MANDIANT™
INTELLIGENT INFORMATION SECURITY