

# Amateur Malware Analysis

Are you interested in reversing but don't know where to start? Think reversing is beyond your means? This talk will try to make the topic of reversing more accessible and will focus on amateur malware analysis, including setting up your environment, free tools, and some demos.

Craig Leabhart

Politics Junkie

Geek

Viking

Occasional Blogger

Aspiring Cybersecurity Expert

# Craig Leabhart

Political Science, BA – University of Iowa

Computer Science, BS – Drake University

# Why do malware analysis?

1. How did the initial incident occur?
2. Once the incident had occurred, how was it able to continue?
3. How did key events occur and what was the timeline?
4. Exactly what systems or information was compromised?
5. Has the incident been fully remediated?

# It can't be that hard, can it?

- Virtualization - knowledge of type-2 hypervisors such (i.e. Oracle VirtualBox or VMware Workstation)
- Networking – LANs and VLANs
- High Level Programming Language Constructs – Java, C++, Python, etc
- X86 Assembly Language
- Windows OS and Windows API

# Physical Lab vs Virtual Lab

# Physical

## *Pros:*

- Malware will run “correctly” if able to detect virtual environment
- Can observe malware on real (not virtual) network

## *Cons:*

- Takes longer to restore machine images
- Need to have access to physical machines and space to set up network

# Virtual

## *Pros:*

- Quickly restore virtual machine images
- Session recording/playback
- Less space/hardware needed

## *Cons:*

- Some malware knows it is in virtual environment
- Some malware can escape to host OS



To air gap or not to air gap?

No production environments!

Simulate WAN access if possible

Otherwise use protection

Methods

**Static vs Dynamic**

Start with basic static analysis

Look at file details and file header,  
doesn't look at code

Move into basic dynamic analysis

How is it interacting with the victim host?

Registry changes?

Processes starting/stopping

Network communications?

Advanced static analysis

Disassemble the binary and reverse engineer the code

# Advanced dynamic analysis

Run the malware inside a debugger and step through the lines of code being executed

Tools

# ApateDNS

*Dynamic*

Can be used to capture DNS requests, spoof basic DNS responses, and redirect DNS to a specified server.

<https://www.fireeye.com/services/freeware/mandiant-apatedns.html>



# Dependency Walker

*Static*

Can be used to identify the DLLs used by malware, as well as specific functions that the malicious code imports and uses.

[http://www.dependencywalker.com/depends22\\_x86.zip](http://www.dependencywalker.com/depends22_x86.zip)

# IDA Pro

*Static*

IDA Pro is a binary disassembly program with advanced features to assist with reverse engineering malicious code. It can be used to view the assembly language code and make modifications and notations.

[https://www.hex-rays.com/products/ida/support/download\\_freeware.shtml](https://www.hex-rays.com/products/ida/support/download_freeware.shtml)

# INetSim

*Dynamic*

Can be used to simulate most network services, including web servers, email servers, and file servers. Using INetSim you simulate a live Internet connection in an air gapped laboratory environment.

<https://www.inetsim.org/downloads.html>

# md5deep

*Static/Dynamic*

Can be used to calculate the file hash of a suspected malicious file. The file hash can be shared with other researchers or used to search databases of known malicious programs.

Can be used to identify or confirm file modifications by malware. Calculate the file hash before executing the malware, and then again after executing the malware. If the hash has changed so has the file.

<https://github.com/jessek/hashdeep>

# Netcat

*Dynamic*

Can be used for port listening to get information about how malware is communicating.

<https://nmap.org/download.html#windows>

# OllyDbg

*Dynamic*

OllyDbg is a debugging program that can be used to run and analyze malware code line by line. It also supports plugins and scripts to automate or perform complex tasks.

<http://www.ollydbg.de/>

# PEBrowse

*Static*

Can be used to examine the header file of a PE, as well as different sections of the malware and its imports and exports.

<http://www.smidgeonsoft.prohosting.com/pebrowse-pro-file-viewer.html>

# PEiD

*Static*

Can identify the compiler used by the malware author, as well as what packer was used if the malware has been packed.

<http://www.aldeid.com/wiki/PEiD>



# PEview

*Static*

Can be used to examine the header file of a PE, as well as different sections of the malware and its imports and exports.

<http://wjradburn.com/software/>

# Process Explorer

*Dynamic*

Can be used to get a real-time overview of processes, as well as in-depth process information such as handles and DLLs used by the process.

<https://technet.microsoft.com/en-us/sysinternals/>

# Process Monitor

*Dynamic*

Can be used to get a real-time overview of what is happening when malware is running. It allows you to observe the filesystem, processes, and the system registry.

<https://technet.microsoft.com/en-us/sysinternals/>

# Regshot

*Dynamic*

Can be used to identify changes to the system registry by taking a snapshot of the registry before and after running the malware. Regshot will report the differences between the two snapshots.

<http://sourceforge.net/projects/regshot/>

# Resource Hacker

*Static*

Can be used to not only examine the resources in a PE, but also modify and export them. This is especially useful if there are other files and pieces of code that might not otherwise be exported unless the malicious program has been run.

<http://www.angusj.com/resourcehacker/>

Note:

# Strings

*Static*

Strings is a simple, command line program that will return a list of all of the string values found in a file. Among other things, string values found might include command and control server IP addresses or names of files that the malware will create or modify.

<https://technet.microsoft.com/en-us/sysinternals/>

# WinDbg

*Dynamic*

WinDbg is a debugging program that can be used to run and analyze malware code line by line. Unlike OllyDbg, WinDbg supports kernel debugging.

<https://msdn.microsoft.com/en-us/windows/hardware/hh852365.aspx>

# Wireshark

*Dynamic*

Wireshark is a packet sniffer used to capture and analyze network traffic. It can be used to analyze the malware's communications.

<https://www.wireshark.org/download.html>



# Hybrid Analysis

*Static/Dynamic*

Free malware analysis service for the community that detects and analyzes unknown threats.

<https://www.hybrid-analysis.com/>

# Where to get samples?

Used to use [malwr.com](http://malwr.com)...

[hybrid-analysis.com](http://hybrid-analysis.com)

[dasmalwerk.eu](http://dasmalwerk.eu)

Could also set up honey pots or just  
check your junk mail...

Basic Static Analysis

Get the file hash.

Search VirusTotal and/or Hybrid  
Analysis

# Basic Static Analysis

Quick check to see if packed  
using PEiD

# Basic Static Analysis

## Run Strings.

Look for interesting string values

# Basic Static Analysis

Learn as much as you can about PE file, libraries, functions, etc.

Look at .text, .rdata, .data, and .rsrc sections with PView

# Basic Static Analysis

Look at DLLs and library functions  
imported/exported with  
Dependency Walker

# Basic Static Analysis

Look at .rsrc section with  
Resource Hacker

Sometimes there is a driver here  
and then you can extract it.



# Basic Dynamic Analysis

Look for network connections

ApateDNS/INetSim

NetCat

WireShark

# Basic Dynamic Analysis

Run Procmon and look at changes.

# Basic Dynamic Analysis

Run Process Explorer and look at changes.

# Basic Dynamic Analysis

Registry snapshot

Run

Registry snapshot

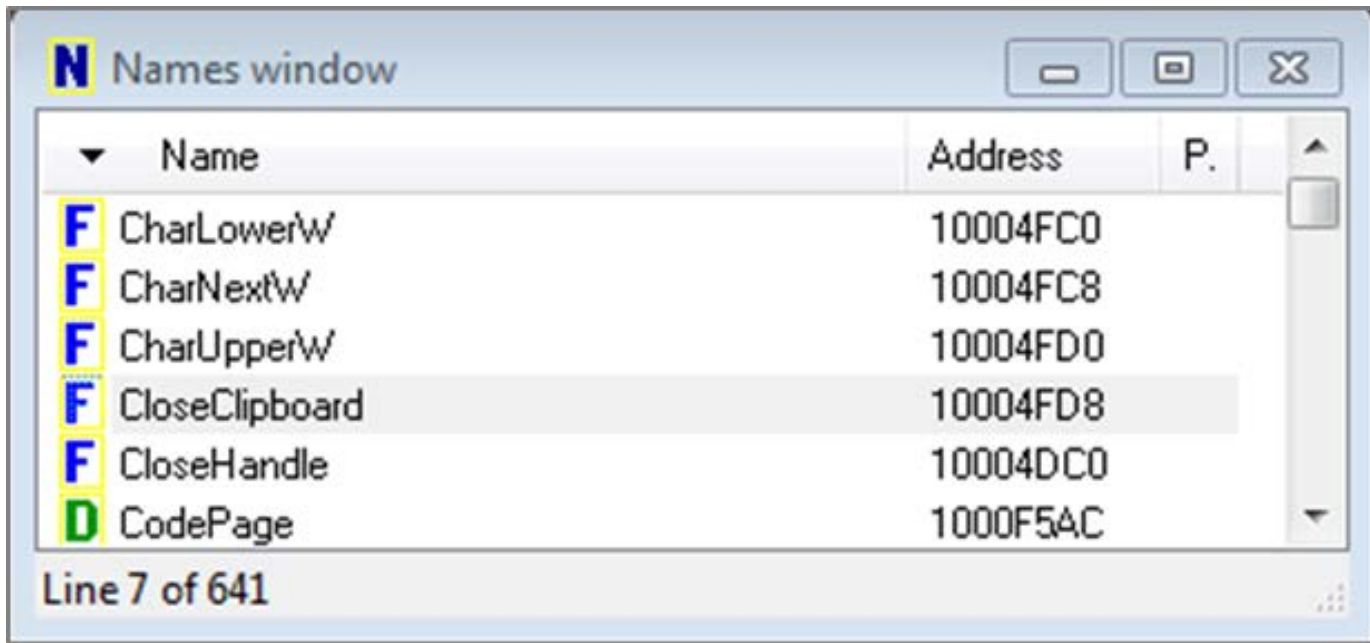
Compare

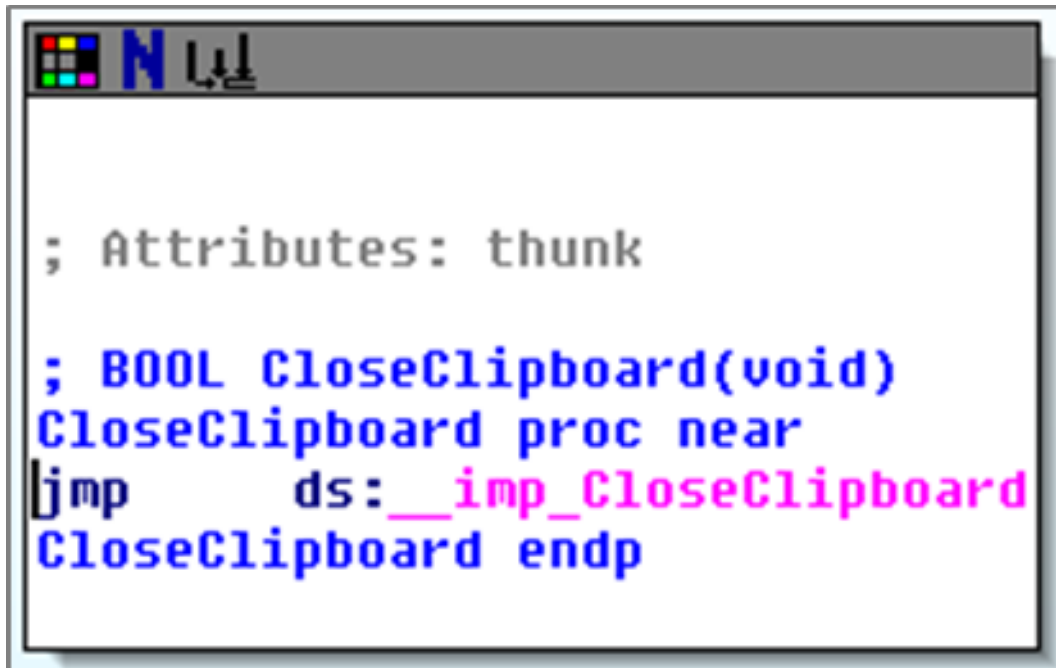
# Advanced Static Analysis

**IDA Pro!**

# Advanced Static Analysis

## IDA Pro!





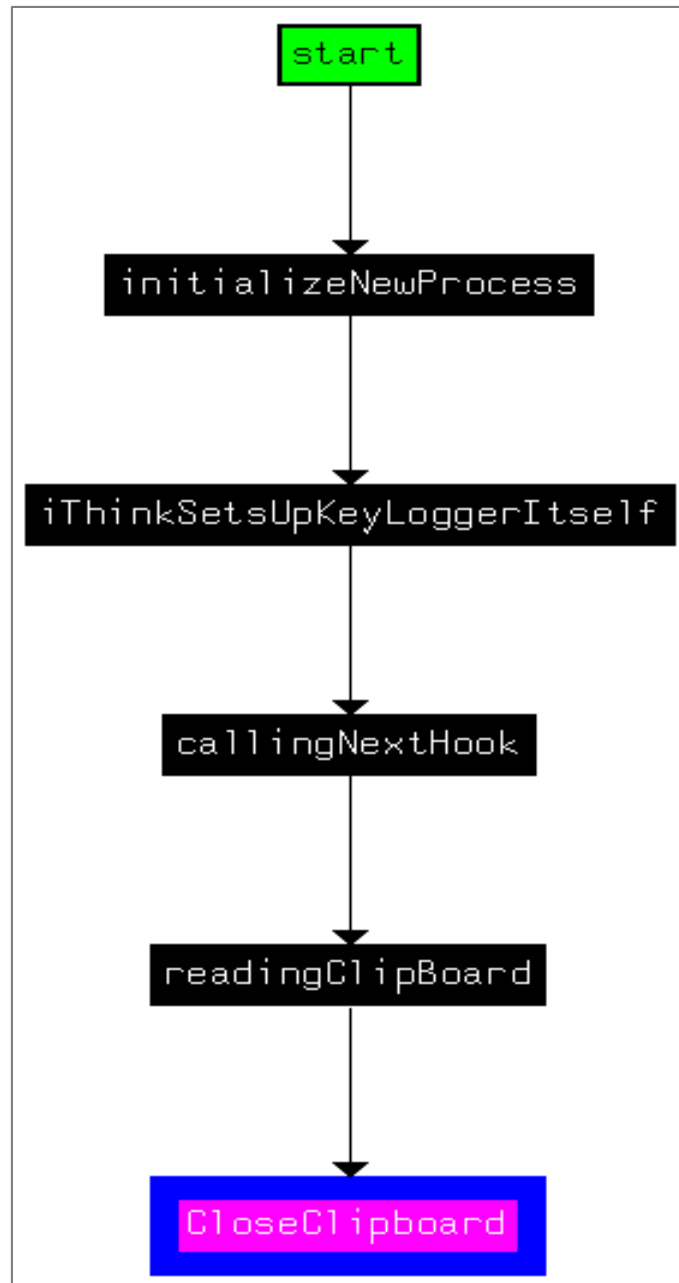
```
; Attributes: thunk  
  
; BOOL CloseClipboard(void)  
CloseClipboard proc near  
| jmp     ds: __imp_CloseClipboard  
CloseClipboard endp
```

```

CODE:100069DC
CODE:100069DC ; :::::::::::::::::::: S U B R O U T I N E ::
CODE:100069DC
CODE:100069DC ; Attributes: bp-based frame
CODE:100069DC readingClipboard proc near
CODE:100069DC
CODE:100069DC var_18          = dword ptr -18h
CODE:100069DC var_14          = dword ptr -14h
CODE:100069DC hMem          = dword ptr -0Ch
CODE:100069DC var_5          = byte ptr -5
CODE:100069DC var_4          = dword ptr -4
CODE:100069DC
* CODE:100069DC          push    ebp
* CODE:100069DD          mov     ebp, esp
* CODE:100069DF          add     esp, 0FFFFFFE8h
* CODE:100069E2          push   ebx
* CODE:100069E3          push   esi
* CODE:100069E4          xor    ecx, ecx

```





```
startBackdoor proc near
push    ebx
mov     ebx, eax
```

```
loc_1000B21B:
mov     ecx, offset dword_1000F814
mov     edx, [ebx+13A8h]
lea    eax, [ebx+18h]
call   connectToServerAndDownload
mov     byte ptr ds:aYsl+4, al
push   3E8h           ; dwMilliseconds
call   Sleep
jmp     short loc_1000B21B
startBackdoor endp
```

# Advanced Dynamic Analysis

## Debugging!