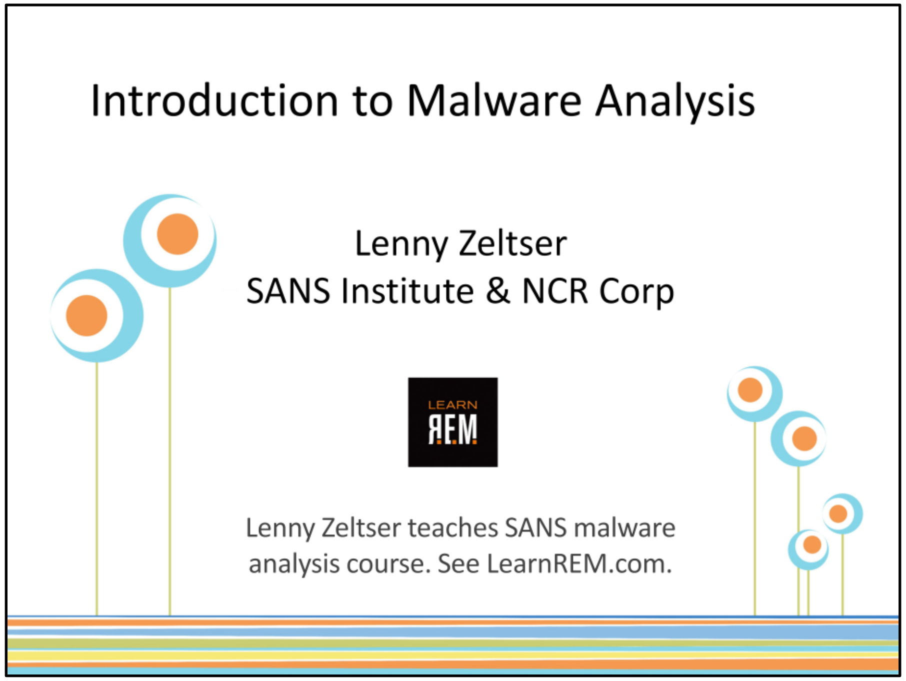


# Introduction to Malware Analysis

Lenny Zeltser
SANS Institute & NCR Corp

Lenny Zeltser teaches SANS malware analysis course. See LearnREM.com.

SANS Institute's popular malware analysis course has helped IT administrators, security professionals, and malware specialists fight malicious code in their organizations. In this briefing, I introduce the process of reverse-engineering malicious software. I cover behavioral and code analysis phases, to make this topic accessible even to individuals with a limited exposure to programming concepts. You'll learn the fundamentals and associated tools to get started with malware analysis.

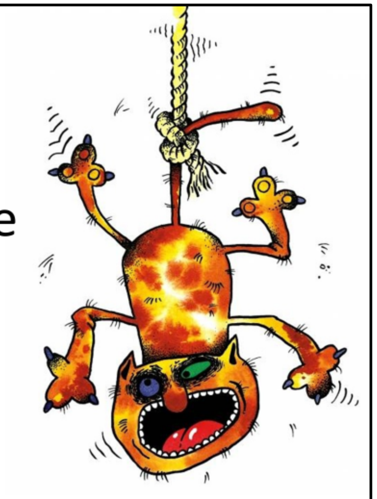
Security incident responders benefit from knowing how to reverse-engineer malware, because this process helps in assessing the event's scope, severity, and repercussions. It also assists in containing the incident and in planning recovery steps. Those who perform forensic investigations also benefit from mastering this topic, because they learn how to understand key characteristic of malware present on compromised systems.

Malicious software is an integral component of many investigations.

How relevant malware has become in the context of computer intrusions! Almost every data breach announced publically, it seems, involves some form of malicious software, such as backdoors, trojans, network worms, exploits, and so on.

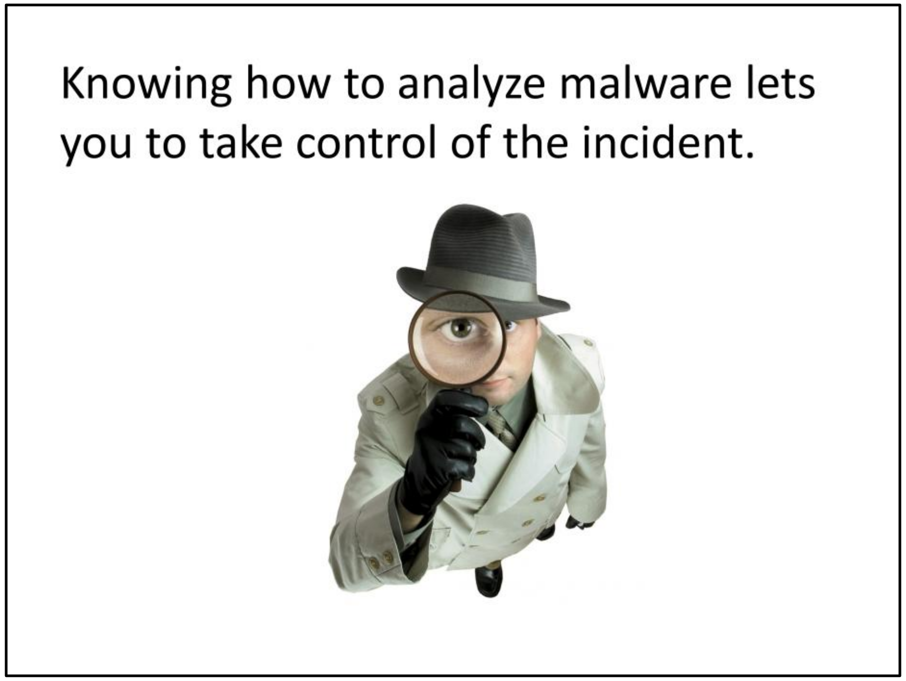
In this session, I will introduce you to the approaches for analyzing malware, so you can turn malicious executable inside out to understand their inner-workings.

Organizations struggle to understand malware they encounter.



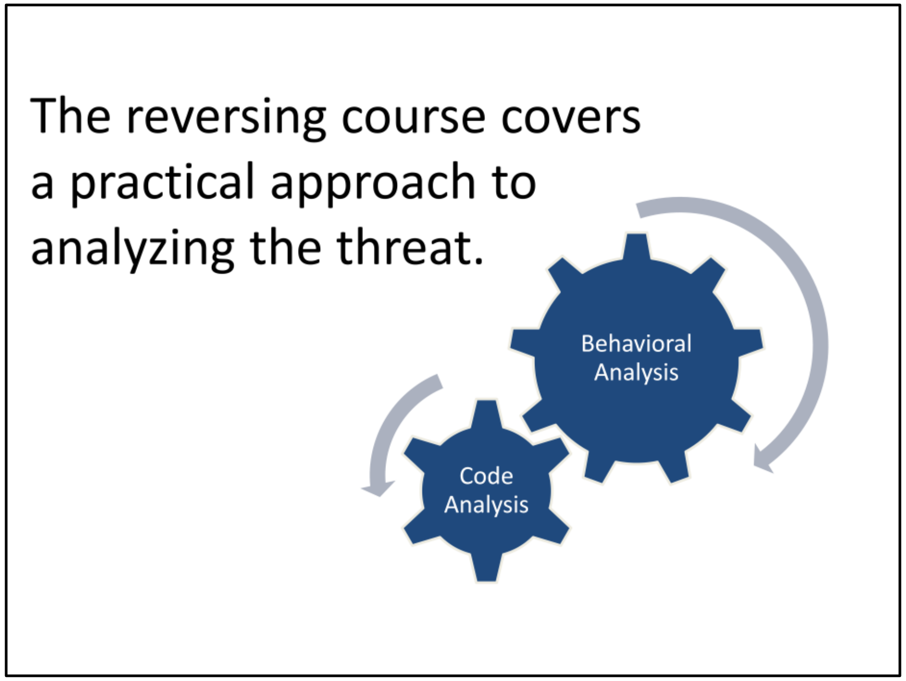
When such an intrusion occurs at your organization, will you be able to quickly assess the threat? Knowing how to analyze malware can help you understand the context of the incident, its severity and repercussions. It can help you plan your response to contain the incident's scope and, in some cases, understand what entities might be behind the intrusion.

Perhaps that is why the individuals who are looking to acquire malware analysis skills are no longer just anti-virus and threat researchers, but also system and network administrators, as well as general security professionals. More and more often, these individuals are being asked to understand the capabilities of malware that their organizations discover.



## Knowing how to analyze malware lets you to take control of the incident.

Knowing how to analyze malware can bring an element of control into an otherwise chaotic environment that exists around a security incident. It’s also a critical aspect of modern forensic analysis actions, because it’s all too frequent for investigators to discover malware on the compromised systems.

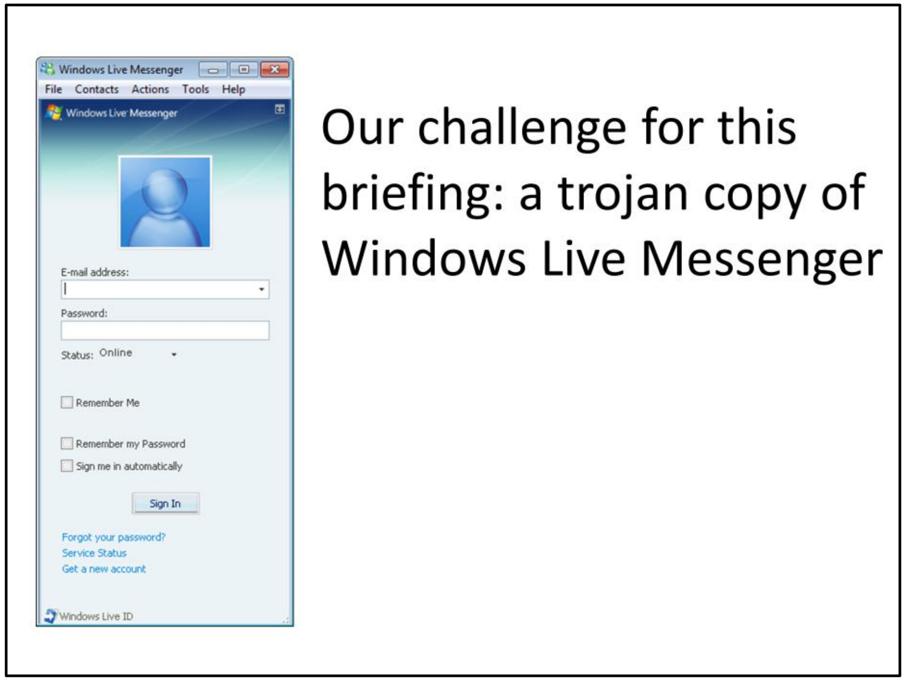


## The reversing course covers a practical approach to analyzing the threat.

Behavioral Analysis
Code Analysis

The approach to reverse-engineering that has worked for many analysts involves two key phases: behavioral analysis and code analysis. During behavioral analysis, we examine how the specimen interacts with its environment. The code analysis phase allows us to learn about the specimen's capabilities by examining the code from which the program is comprised.

You'll see this approach in action in the upcoming slides.

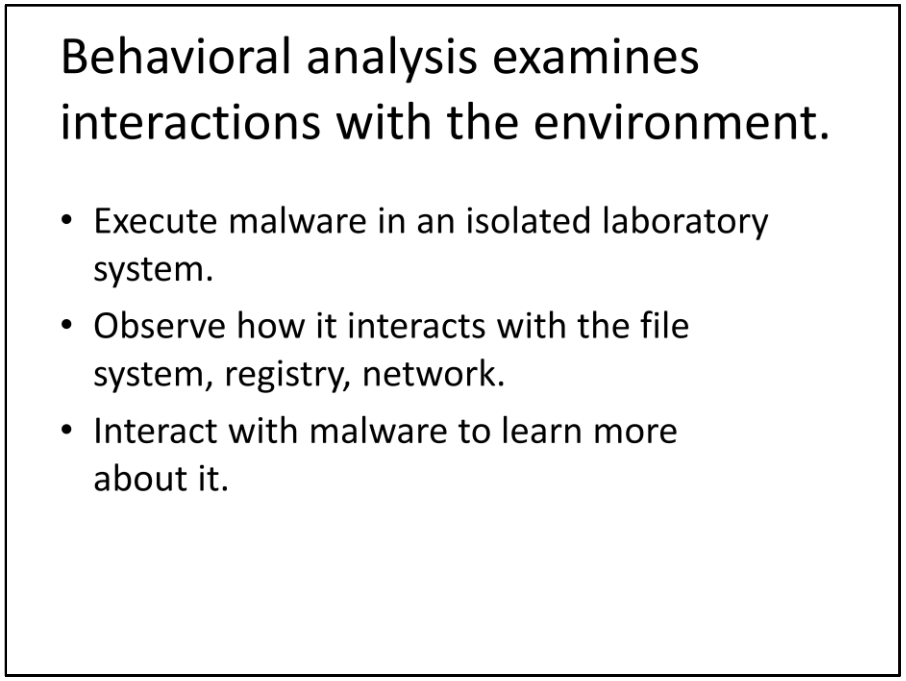


## Our challenge for this briefing: a trojan copy of Windows Live Messenger

I find that the best way to learn malware analysis is by going through examples. The malicious executable from which we'll learn in this session is captured on this slide. It's a trojan copy of Windows Live Messenger—a fake instant messenger client that was being distributed to victims via email. Many such trojans have the capability of capturing the victims' logon credentials, and may have other "undocumented" features.

Let's see what capabilities are built into this malicious executable. As I lead you through the analysis, I'll introduce the tools and techniques that will help with the reverse-engineering process.

Note that in this example, as with the majority of malicious incidents you'll probably encounter, we'll be examining a compiled Windows executable for which we have no source code.



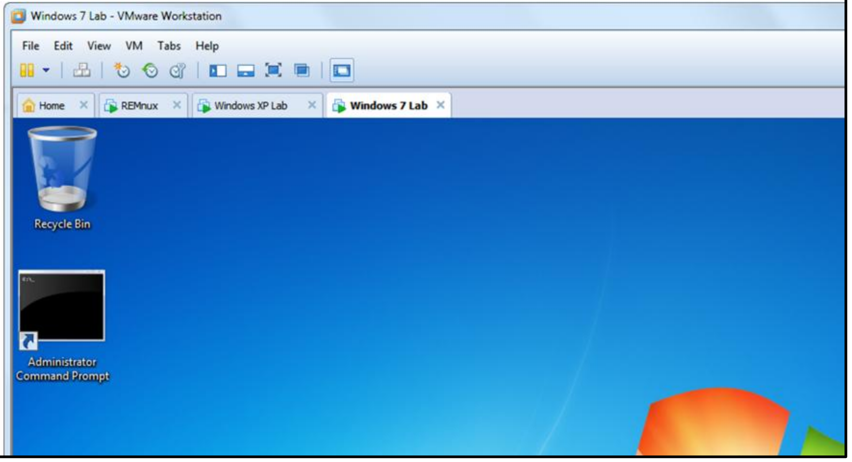
## Behavioral analysis examines interactions with the environment.

- Execute malware in an isolated laboratory system.
- Observe how it interacts with the file system, registry, network.
- Interact with malware to learn more about it.

I typically start examining a malicious executable with behavioral analysis, because it comes more easily to me than code analysis. If your strength is in programming and x86 assembly, then you may prefer to start with the code analysis phase instead.

When performing behavioral analysis, we're going to infect a laboratory system with the specimen. Then we'll observe how the malicious executable accesses the file system, the registry, and the network. As we learn about the program's expectations of its runtime environment, we will slightly adjust the laboratory infrastructure to evoke additional behavior from the program. We will also attempt to interact with the program to discover additional characteristics it may exhibit.

## It’s convenient to virtualize the lab (VMware, VirtualBox, etc.).


Windows 7 Lab - VMware Workstation
File Edit View VM Tabs Help
Home
REMnux
Windows XP Lab
Windows 7 Lab
Recycle Bin
Administrator Command Prompt

When performing malware analysis, it’s convenient to use virtualization software when setting up your lab. Such tools typically simulate the underlying hardware, allowing you two run multiple instances of “virtual” machines simultaneously. For instance, you could use Windows 7 as your base OS, while having a separate instance of Windows 7 running in another window, a Windows XP window in another and a Linux instance in another.

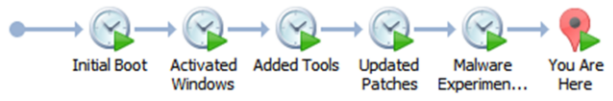
Each virtual machine behaves mostly as “real” physical systems, in that it has its own set of I/O peripherals, RAM, network settings, and so on. All these aspects of the virtual machine are, well, virtualized.

The convenience of a virtualized lab comes, in part, from the flexibility of having multiple instances of various operating systems available to you within a single physical system. Virtualization software can even emulate a network, so that your lab doesn’t need to be connected to a physical network at all. Yet, the virtual machines will be able to communicate with each other over the simulated network, blissfully unaware that the network is not “real.”

I typically use VMware Workstation for virtualization. Other choices include Microsoft Virtual PC, Oracle VirtualBox, etc.

## Being able to switch between snapshots is very helpful.

- A snapshot represents a state of the virtual machine
- VMware Workstation supports multiple snapshots
- For physical systems use dd, Ghost, etc.



One of the most convenient aspects of using virtualization software is its support for snapshots. They allow you to preserve the current state of the virtual machine with a click of a button, and return to it with another click. VMware Workstation support multiple snapshots, which comes in very handy for “bookmarking” different stages of your analysis, so you can move back and forth during your experiments without losing important runtime details.

Snapshot capabilities are also very useful for reverting back to the system’s pristine state after you’ve completed your research and want prepare the lab for your next analysis. Save the state of the virtual machine after you’ve installed the OS, patched it, and set up the necessary tools. Once you’re done with your analysis, click a button to revert to that state. Very convenient!

Malware may have defenses that prevent it from executing properly in a virtualized environment. In these cases, the easiest step might be to use a set of physical systems, instead. To mimic snapshot functionality when you’re unable to use virtualization software, use disk cloning tools such as dd and Norton Ghost.

## Mitigate the risks of malware attempting to escape from the lab.

- Avoid production network connectivity.
- Dedicate a host to the lab.
- Restore the host if anything suspicious occurs.
- Keep up with patches to virtualization software (e.g. VMware).

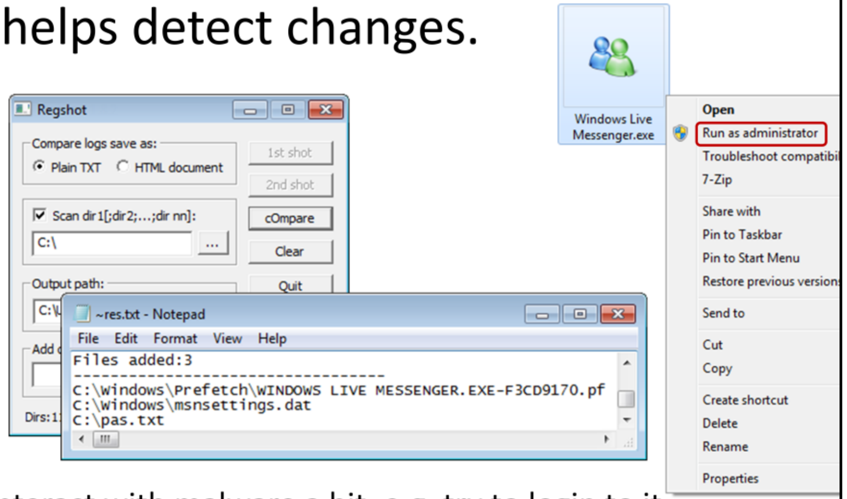
Any malware analysis lab carries the risk of malware finding a way to escape from your sandbox. This risk is greater with a virtualized lab, because the isolation it provides is not as reliable as the literal air gap between physical systems.

Since virtualization software is written by human beings, it will have bugs in it. Some of these bugs are vulnerabilities that malicious software may use in an attempt to escape the sandbox around your laboratory system. To address this risk, I suggest dedicating a single physical system to your virtualized lab: run several virtual machines in it, but don't use that system for another purpose. Also, don't connect the laboratory box to your production network unless required for performing specific tasks.

It's also very important to keep your virtualization software up to date on security patches. Sometimes they're a pain to download and install.

If you notice anything suspicious in the lab environment when performing your analysis, restore the physical system from a backup copy, and keep a close eye on the environment.

## Infect the lab system. Regshot helps detect changes.



```
Files added:3
----------------------------------
C:\Windows\Prefetch\WINDOWS LIVE MESSENGER.EXE-F3CD9170.pf
C:\Windows\msnsettings.dat
C:\pas.txt
```

Interact with malware a bit, e.g. try to login to it.

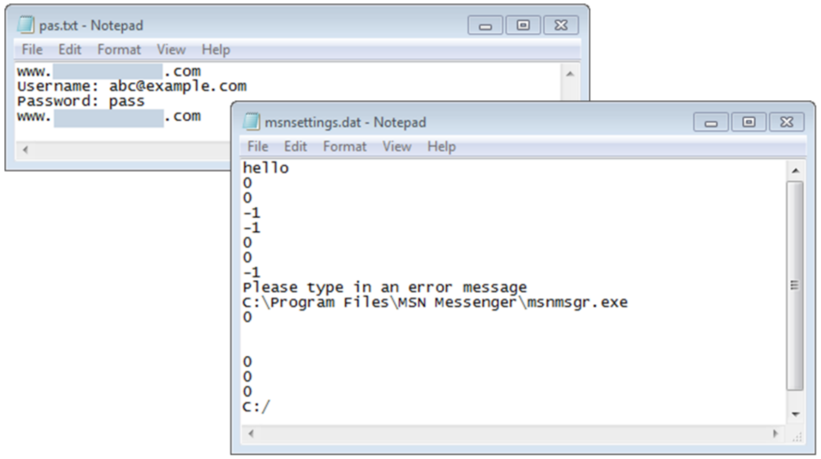
Let’s see this approach in action. Let’s say you have a suspicious executable that you’d like to analyze. You bring it into your lab, possibly via a removable USB disk and place it on the desktop of the virtual machine you’re about to infect. Now what?

First, take a snapshot of the state of the machine’s file system and the registry. This will allow you to quickly see what major changes have occurred on the system after you infect it.

I like the free tool called Regshot for this purpose (http://sourceforge.net/projects/regshot). To use it, enable the “Scan dir1” option, and in the corresponding window type “C:\”. Click “1st shot”.

After Regshot takes the first snapshot, run the malicious executable “as administrator” to allow the malicious program to reach its full potential. Interact with it a bit (e.g., try logging into it). Then kill the malicious process. Next, click the “2nd shot” button in RegShot, and click the “Compare” button. You’ll see a report that describes the major changes to the system’s state. In this case, we see that a few files were added to the system.

## Examine the newly-created suspicious files.


pas.txt - Notepad

```
www.[redacted].com
Username: abc@example.com
Password: pass
www.[redacted].com
```

msnsettings.dat - Notepad

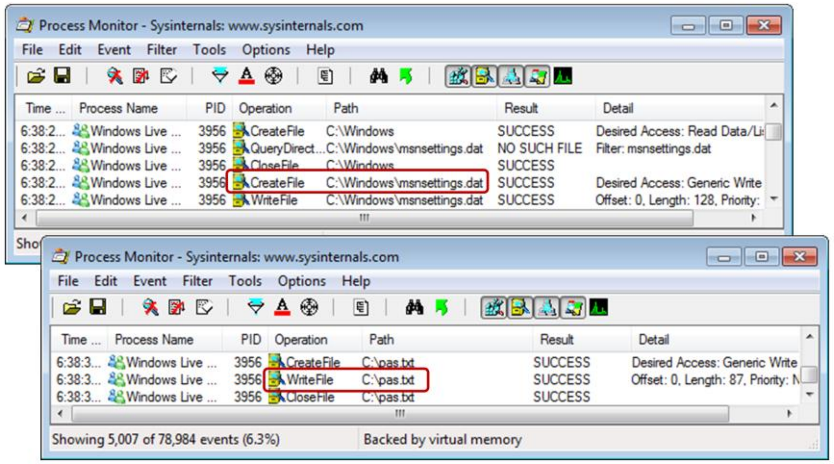
```
hello
0
0
-1
-1
0
0
-1
Please type in an error message
C:\Program Files\MSN Messenger\msnmsgr.exe
0


0
0
0
C:/
```

The two files that appeared on the system after we infected it are pas.txt and msnsettings.dat. Take a look at them using notepad.

It looks like pas.txt has captured the logon credentials we used when logging into the malicious executable. That makes sense, because we received reports that this executable is a trojan copy of Windows Live Messenger.

The msnsettings.dat file looks like a configuration file of some sort.

## Process Monitor observes malware as it infects the system.


Process Monitor - Sysinternals: www.sysinternals.com

| Time ... | Process Name | PID | Operation | Path | Result | Detail |
|---|---|---|---|---|---|---|
| 6:38:2... | Windows Live ... | 3956 | CreateFile | C:\Windows | SUCCESS | Desired Access: Read Data/Li |
| 6:38:2... | Windows Live ... | 3956 | QueryDirect... | C:\Windows\msnsettings.dat | NO SUCH FILE | Filter: msnsettings.dat |
| 6:38:2... | Windows Live ... | 3956 | CloseFile | C:\Windows | SUCCESS | |
| 6:38:2... | Windows Live ... | 3956 | CreateFile | C:\Windows\msnsettings.dat | SUCCESS | Desired Access: Generic Write |
| 6:38:2... | Windows Live ... | 3956 | WriteFile | C:\Windows\msnsettings.dat | SUCCESS | Offset: 0, Length: 128, Priority: |

Process Monitor - Sysinternals: www.sysinternals.com

| Time ... | Process Name | PID | Operation | Path | Result | Detail |
|---|---|---|---|---|---|---|
| 6:38:3... | Windows Live ... | 3956 | CreateFile | C:\pas.txt | SUCCESS | Desired Access: Generic Write |
| 6:38:3... | Windows Live ... | 3956 | WriteFile | C:\pas.txt | SUCCESS | Offset: 0, Length: 87, Priority: N |
| 6:38:3... | Windows Live ... | 3956 | CloseFile | C:\pas.txt | SUCCESS | |

Showing 5,007 of 78,984 events (6.3%) Backed by virtual memory

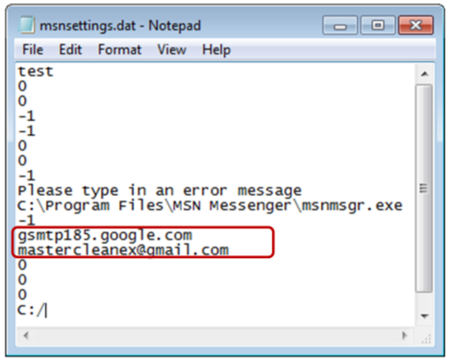
Another free tool that can help us understand how the malicious program interacted with the file system and the registry is Process Monitor (http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx).

To use Process Monitor, run it while infecting the system. I typically launch the tool right after taking the first Regshot snapshot. Remember to pause capture in Process Monitor before taking the second Regshot snapshot.

Process Monitor records API calls that deal with file system, registry and other local activities. In the screen shot on this slide, you see attempts by our malware specimen to create pas.txt file and to locate the msnsettings.dat file.

Process Monitor's log is very comprehensive. However, it is also very noisy. I use Regshot to make sure that I don't miss anything critical, while I rely on Process Monitor to present a comprehensive perspective on the specimen's interactions with the file system and the registry.

## Now you know to look for additional files on the victim's actual system.



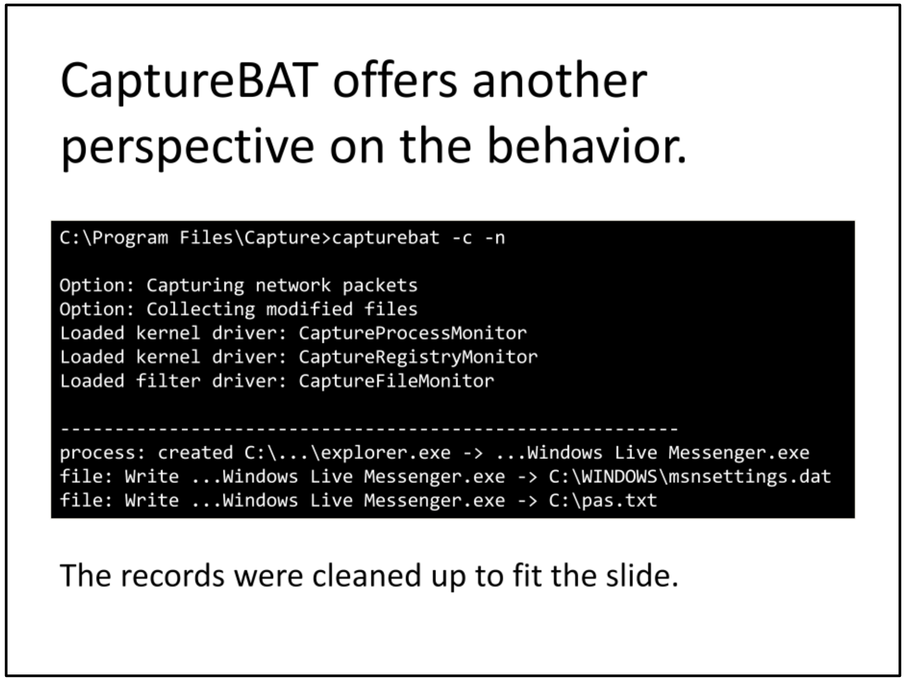
```
test
0
0
-1
-1
0
0
-1
Please type in an error message
C:\Program Files\MSN Messenger\msnmsgr.exe
-1
gsmtp185.google.com
mastercleanex@gmail.com
0
0
0
C:/
```

Several settings on the victim's system differ from the lab's.

Reverse-engineering malware can help you become better at incident response and forensic analysis. In our scenario, we have already discovered that Windows Live Messenger trojan makes use of the msnsettings.dat file. Now you know to look for it on the compromised system, even if you didn't initially realize that this file was important.

Once you have a copy of msnsettings.dat, you can open it to see whether it reveals additional details about the program. On this slide, I've highlighted several lines from that file.

One is a string "test," which we may be able to use later when trying to understand how the trojan processes the msnsettings.dat file. Another line, "gsmtp185.google.com" specifies an SMTP mail server; this suggests that our specimen has the ability to send email. The file also includes an email address, "mastercleanex@gmail.com". This may be the recipient of the information that the trojan might attempt to send out. Of course, these are just theories at this point. We'll need to confirm or deny them during subsequent analysis steps.



## CaptureBAT offers another perspective on the behavior.

```
C:\Program Files\Capture>capturebat -c -n

Option: Capturing network packets
Option: Collecting modified files
Loaded kernel driver: CaptureProcessMonitor
Loaded kernel driver: CaptureRegistryMonitor
Loaded filter driver: CaptureFileMonitor

---------------------------------------------------------
process: created C:\...\explorer.exe -> ...Windows Live Messenger.exe
file: Write ...Windows Live Messenger.exe -> C:\WINDOWS\msnsettings.dat
file: Write ...Windows Live Messenger.exe -> C:\pas.txt
```

The records were cleaned up to fit the slide.

It helps to have several tools to observe the malicious program's interactions with its environment. Another very useful and free tool I'd like to tell you about is CaptureBAT (http://www.honeynet.org/node/315).

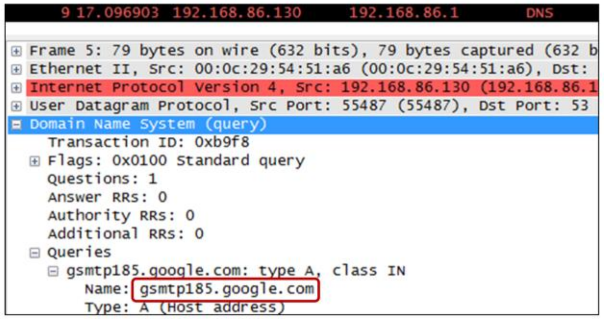
CaptureBAT is similar to Process Monitor in that it records local processes' interactions with their environment. CaptureBAT's logs tend to be less noisy than those created by Process Monitor. This is because CaptureBAT comes with filters that eliminate the majority of standard, non-malicious activities from the logs. You can customize these filters to your liking, as they are text files located in the directory where you install CaptureBAT.

If you launch CaptureBAT with the "-c" parameter, it will capture any files deleted in the background, allowing you to look at and restore even those files that the Windows Recycle Bin cannot capture.

Launching CaptureBAT with the "-n" parameter tells the tool to capture network traffic, like a sniffer would, saving the result into a local .cap file.

As you can see on this slide, CaptureBAT confirmed our earlier findings about the malware specimen.

## Place the new file into the lab. Now the sniffer (Wireshark) shows DNS.

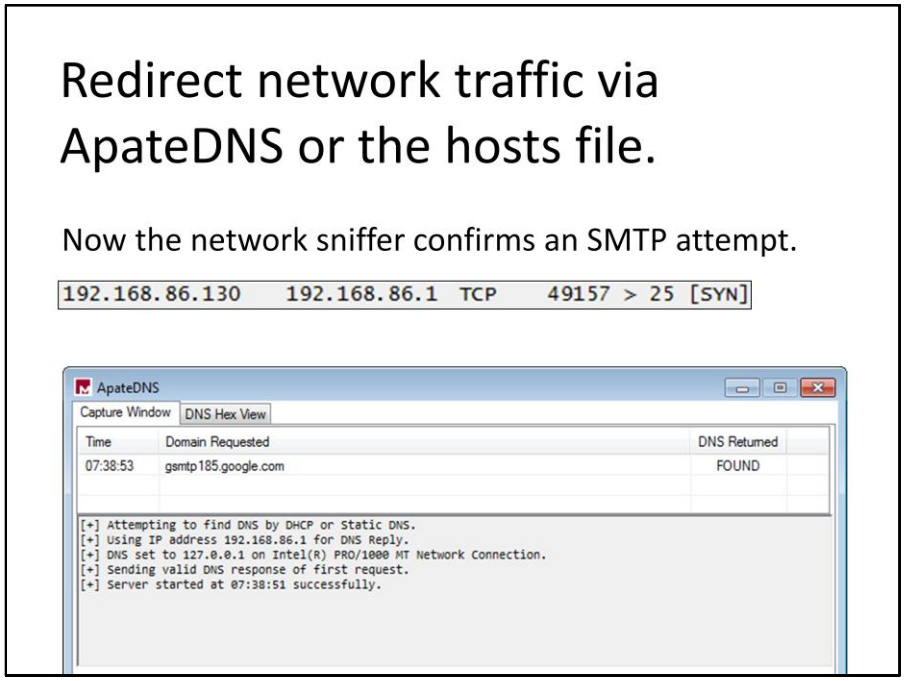


```
9 17.096903  192.168.86.130       192.168.86.1         DNS

⊞ Frame 5: 79 bytes on wire (632 bits), 79 bytes captured (632 b
⊞ Ethernet II, Src: 00:0c:29:54:51:a6 (00:0c:29:54:51:a6), Dst:
⊞ Internet Protocol Version 4, Src: 192.168.86.130 (192.168.86.1
⊞ User Datagram Protocol, Src Port: 55487 (55487), Dst Port: 53
⊟ Domain Name System (query)
    Transaction ID: 0xb9f8
  ⊞ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ gsmtp185.google.com: type A, class IN
        Name: gsmtp185.google.com
        Type: A (Host address)
```

The hostname suggests SMTP, but we'll set up DNS resolution to confirm.

You can load the .cap file created by CaptureBAT into a full-feature network sniffer, such as Wireshark (http://www.wireshark.org). If you don't like using CaptureBAT, you could also use Wireshark to capture traffic direct off the laboratory network.

As you can see on this slide, the sniffer shows that the infected system has issued a DNS query, attempting to resolve the hostname "gsmtp185.google.com". The "smtp" in the hostname suggests that the malware specimen is looking for a mail server to connect to, reinforcing our earlier theory of how the trojan might use this hostname.



## Redirect network traffic via ApateDNS or the hosts file.

Now the network sniffer confirms an SMTP attempt.

```
192.168.86.130    192.168.86.1  TCP    49157 > 25 [SYN]
```

ApateDNS

Capture Window | DNS Hex View

| Time | Domain Requested | DNS Returned |
|---|---|---|
| 07:38:53 | gsmtp185.google.com | FOUND |

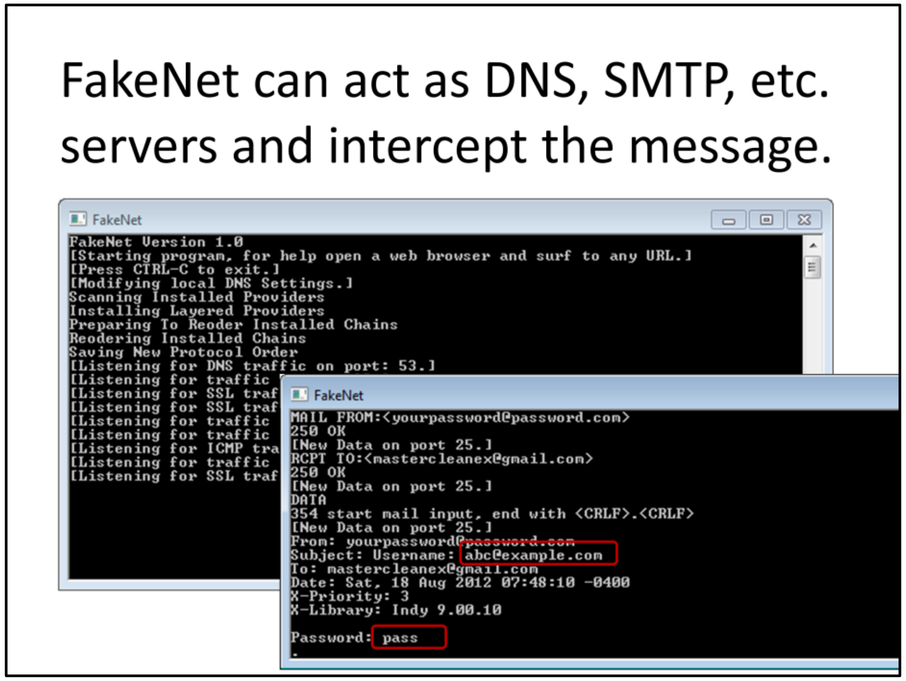
```
[+] Attempting to find DNS by DHCP or Static DNS.
[+] Using IP address 192.168.86.1 for DNS Reply.
[+] DNS set to 127.0.0.1 on Intel(R) PRO/1000 MT Network Connection.
[+] Sending valid DNS response of first request.
[+] Server started at 07:38:51 successfully.
```

To confirm how the specimen wishes to use “gsmtp185.google.com”, allow the trojan to resolve this hostname. Once it can resolve it, it will presumably attempt connecting to it, and you will be able to use a network sniffer to see what service the specimen is trying to access.

To set up name resolution, insert an entry for the hostname into the “hosts” file on the infected system. A faster alternative is to use a tool called ApateDNS, available as a free download at http://www.mandiant.com/resources/download/research-tool-mandiant-apatedns

ApateDNS is a DNS server that you can configure to answer any DNS query with a single IP address of your choice. I usually suggest picking an IP address of some system in your lab on which you can run the service that malware may look for. This will redirect the connection to the host where you’d set up the listener, allowing the connection to be completed so you can learn about its purpose.

In our example, captured on this slide, the network sniffer now confirmed that the infected system is attempting to connect to TCP port 25 on “gsmtp185.google.com”.



## FakeNet can act as DNS, SMTP, etc. servers and intercept the message.

```
FakeNet Version 1.0
[Starting program, for help open a web browser and surf to any URL.]
[Press CTRL-C to exit.]
[Modifying local DNS Settings.]
Scanning Installed Providers
Installing Layered Providers
Preparing To Reoder Installed Chains
Reodering Installed Chains
Saving New Protocol Order
[Listening for DNS traffic on port: 53.]
[Listening for traffic
[Listening for SSL traf
[Listening for SSL traf
[Listening for traffic
[Listening for traffic
[Listening for ICMP tra
[Listening for traffic
[Listening for SSL traf
```

```
MAIL FROM:<yourpassword@password.com>
250 OK
[New Data on port 25.]
RCPT TO:<mastercleanex@gmail.com>
250 OK
[New Data on port 25.]
DATA
354 start mail input, end with <CRLF>.<CRLF>
[New Data on port 25.]
From: yourpassword@password.com
Subject: Username: abc@example.com
To: mastercleanex@gmail.com
Date: Sat, 18 Aug 2012 07:48:10 -0400
X-Priority: 3
X-Library: Indy 9.00.10

Password: pass
```

Now that you know malware is looking for an SMTP server, you can provide that service to it within your lab. An easy way to do this is to use the FakeNet tool, available as a free download from http://practicalmalwareanalysis.com/fakenet

FakeNet automatically redirects network traffic, so there is no need to modify the hosts file or use ApateDNS with this tool. FakeNet emulates various common services, including HTTP and SMTP.

In our example, illustrated on this slide, FakeNet pretends to be a mail server, intercepting the email message that our trojan attempts to send though ""gsmtp185.google.com".

Now you can see the contents of the message that the trojan is mailing to the attacker. As highlighted on this slide, the message includes the victim's Windows Live Messenger username and password. We also see that the exfiltrated data is directed to "mastercleanex@gmail.com".

## Mold the environment based on the observations to evoke new behavior.

- Add services gradually, as you learn what the specimen wants.
- If you give too much at once, you lose cause-effect insights.
- Repeat until no interesting discoveries.

How can we generalize the behavioral analysis process we've been following? As you observe a characteristic of the specimen, you typically notice an element of the environment that the program is looking for, yet does not possess in your lab. For instance, the executable may be attempting to resolve a host name. To evoke new characteristics, you provide to the specimen the service it needs, thus allowing it to perform further actions to fulfill each its true potential.

With every service you add to the environment, you learn more about the specimen. Note that if you change too many environmental characteristics at the same time, you malware may perform too many new actions. This will speed up your analysis at the expense of knowing exactly what change was responsible for which observed characteristic.

When do you stop molding the laboratory environment to match the specimen's expectations and dependencies? When you there are no more changes to introduce into the lab to evoke previously-unseen behavioral characteristics. That's typically the point when you will want to start the next phase of the reverse-engineering process: code analysis.

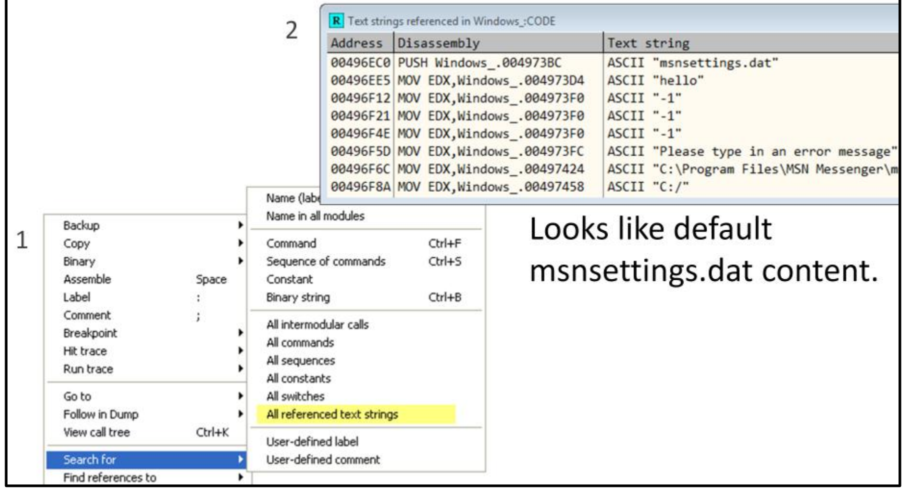
## Code analysis expands and reinforces behavioral findings.

- Tools of the trade: disassembler and debugger.
- Examine the specimen's assembly code.
- Step through the most interesting parts of the code.

Behavioral analysis can be insightful and relatively fast. However, it will rarely tell you everything you need to know about malware of moderate and advanced complexity. That's where code analysis can be of help. It can help reinforce your behavioral findings, and can shine light on additional properties of the specimen that you may not have discovered behaviorally.

Code analysis can be tricky and time-consuming, because in the world of malware you almost never have the luxury of seeing the source code of the program you're analysis. Instead, you need to reverse-engineer the compiled executable's functionality by examining its code at the assembly level. A debugger and a disassembler can help you in this task. A disassembler converts the specimen's instructions from their binary form into the human-readable assembly form. A debugger lets you step through the most interesting parts of the code, interacting with it and observing the effects of its instructions to understand their purpose.

## Looking at strings is a often a good start. In this example, use OllyDbg.



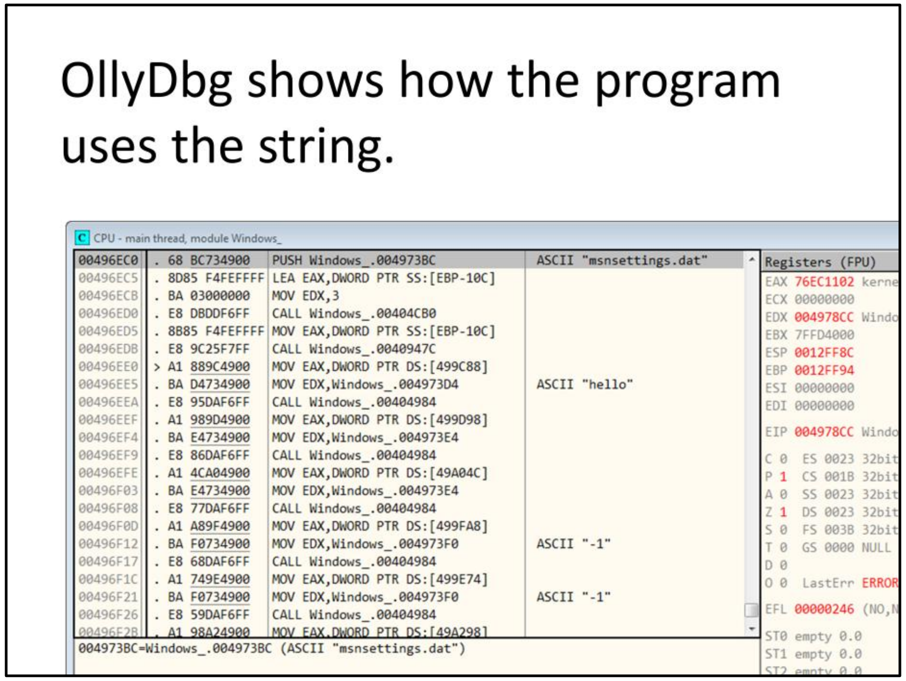
```
Address  Disassembly                  Text string
00496EC0 PUSH Windows_.004973BC       ASCII "msnsettings.dat"
00496EE5 MOV EDX,Windows_.004973D4    ASCII "hello"
00496F12 MOV EDX,Windows_.004973F0    ASCII "-1"
00496F21 MOV EDX,Windows_.004973F0    ASCII "-1"
00496F4E MOV EDX,Windows_.004973F0    ASCII "-1"
00496F5D MOV EDX,Windows_.004973FC    ASCII "Please type in an error message"
00496F6C MOV EDX,Windows_.00497424    ASCII "C:\Program Files\MSN Messenger\m
00496F8A MOV EDX,Windows_.00497458    ASCII "C:/"
```

Looks like default msnsettings.dat content.

OllyDbg is among my favorite tools for performing code analysis. It's free, very powerful, and includes both a disassembler and a debugger. You can download OllyDbg from: http://www.ollydbg.de/

A good way to start analyzing the specimen's code often involves looking at the strings embedded in its executable. To do this with OllyDbg, first load the malicious executable into OllyDbg via File > Open. Then, right-click on the code you will see in the disassembler window, and select Search for > All referenced text strings.

OllyDbg will then bring up a new window that will show the strings it discovered, as you can see on this slide. Notice that we have seen some of these strings during behavioral analysis! Some of them look like contents of the default msnsettings.dat file that our specimen creates when infecting the system.



## OllyDbg shows how the program uses the string.

```
CPU - main thread, module Windows_
00496EC0  . 68 BC734900     PUSH Windows_.004973BC              ASCII "msnsettings.dat"
00496EC5  . 8D85 F4FEFFFF   LEA EAX,DWORD PTR SS:[EBP-10C]
00496ECB  . BA 03000000     MOV EDX,3
00496ED0  . E8 DBDDF6FF     CALL Windows_.00404CB0
00496ED5  . 8B85 F4FEFFFF   MOV EAX,DWORD PTR SS:[EBP-10C]
00496EDB  . E8 9C25F7FF     CALL Windows_.0040947C
00496EE0  > A1 889C4900     MOV EAX,DWORD PTR DS:[499C88]
00496EE5  . BA D4734900     MOV EDX,Windows_.004973D4           ASCII "hello"
00496EEA  . E8 95DAF6FF     CALL Windows_.00404984
00496EEF  . A1 989D4900     MOV EAX,DWORD PTR DS:[499D98]
00496EF4  . BA E4734900     MOV EDX,Windows_.004973E4
00496EF9  . E8 86DAF6FF     CALL Windows_.00404984
00496EFE  . A1 4CA04900     MOV EAX,DWORD PTR DS:[49A04C]
00496F03  . BA E4734900     MOV EDX,Windows_.004973E4
00496F08  . E8 77DAF6FF     CALL Windows_.00404984
00496F0D  . A1 A89F4900     MOV EAX,DWORD PTR DS:[499FA8]
00496F12  . BA F0734900     MOV EDX,Windows_.004973F0           ASCII "-1"
00496F17  . E8 68DAF6FF     CALL Windows_.00404984
00496F1C  . A1 749E4900     MOV EAX,DWORD PTR DS:[499E74]
00496F21  . BA F0734900     MOV EDX,Windows_.004973F0           ASCII "-1"
00496F26  . E8 59DAF6FF     CALL Windows_.00404984
00496F2B  . A1 98A24900     MOV EAX,DWORD PTR DS:[49A298]
004973BC=Windows_.004973BC (ASCII "msnsettings.dat")
```

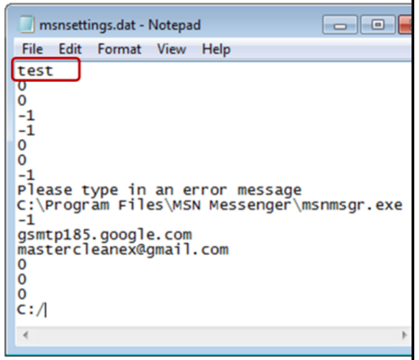
```
Registers (FPU)
EAX 76EC1102 kerne
ECX 00000000
EDX 004978CC Windo
EBX 7FFD4000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 004978CC Windo

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 1  DS 0023 32bit
S 0  FS 003B 32bit
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR

EFL 00000246 (NO,N

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
```

The reason we may be interested in looking at the embedded strings is because the string listing might include a reference to a malicious characteristic or a behavioral trait that we would like to understand. In this case, consider the screenshot on this slide. We got here by highlighting one of the instances of "msnsettings.dat" strings, as shown on the previous slide, and pressing Enter. Now, OllyDbg shows us how the program makes use of this string.

If we wanted to pursue this path of analysis further, we could now set a breakpoint on this command, run the trojan in the debugger, and see what it does. We're not going to investigate this particular aspect of the malicious program, because I want to show you another, more interesting technique.

## What’s the purpose of the “test” string in the victim’s msnsettings.dat file?

- Run the specimen in OllyDbg.
- Look in memory for string “test”.
- Set an access breakpoint.

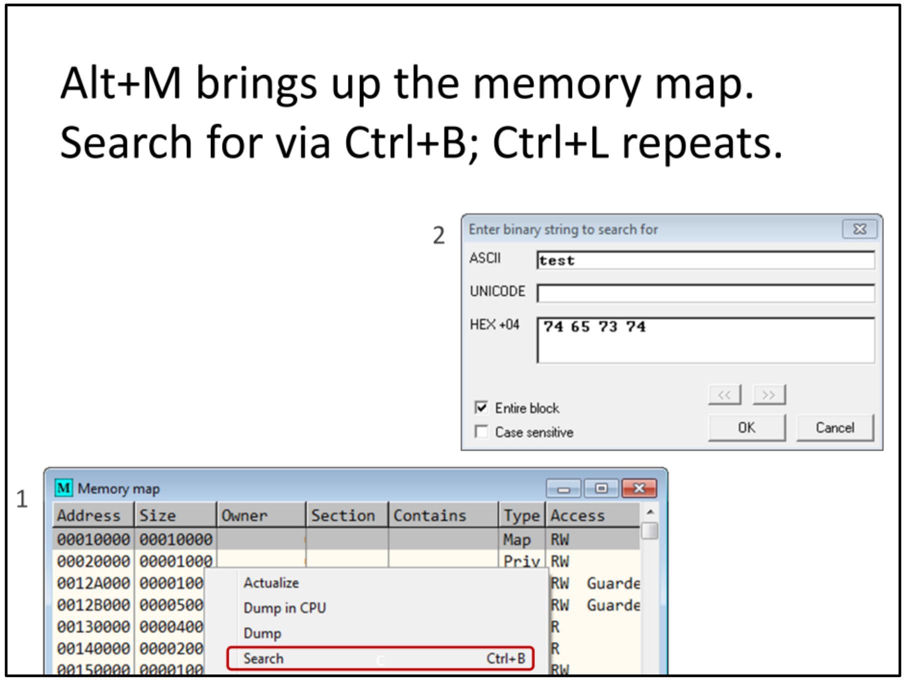


```
test
0
0
-1
-1
0
0
-1
Please type in an error message
C:\Program Files\MSN Messenger\msnmsgr.exe
-1
gsmtp185.google.com
mastercleanex@gmail.com
0
0
0
C:/
```

You may recall that the version of msnsetting.dat on the victim’s system was slightly different from the version that the trojan created on our laboratory system when we first ran it. Specifically, in our case, the file contained the string “hello”, while the victim’s version had the string “test” instead. What’s that about?

The string “test” is not visible anywhere within the body of the malicious executable when it’s not running. That’s probably because the trojan loads this string from msnsettings.dat during run time. To understand how the trojan uses the string “test,” we will search for it in the memory of the running trojan.

Once we locate the string in the trojan’s memory, we will set an access breakpoint there. A breakpoint is a condition that tells the debugger when to pause the normal execution of the debugged program. Once the execution is paused, the debugger will give us a chance to review the debugged program’s run time environment to understand what it is doing. This is probably the most useful feature of a debugger in the context of reverse-engineering malware.



Alt+M brings up the memory map.
Search for via Ctrl+B; Ctrl+L repeats.

2

Enter binary string to search for

ASCII: test

UNICODE:

HEX +04: 74 65 73 74

[✓] Entire block
[ ] Case sensitive

1

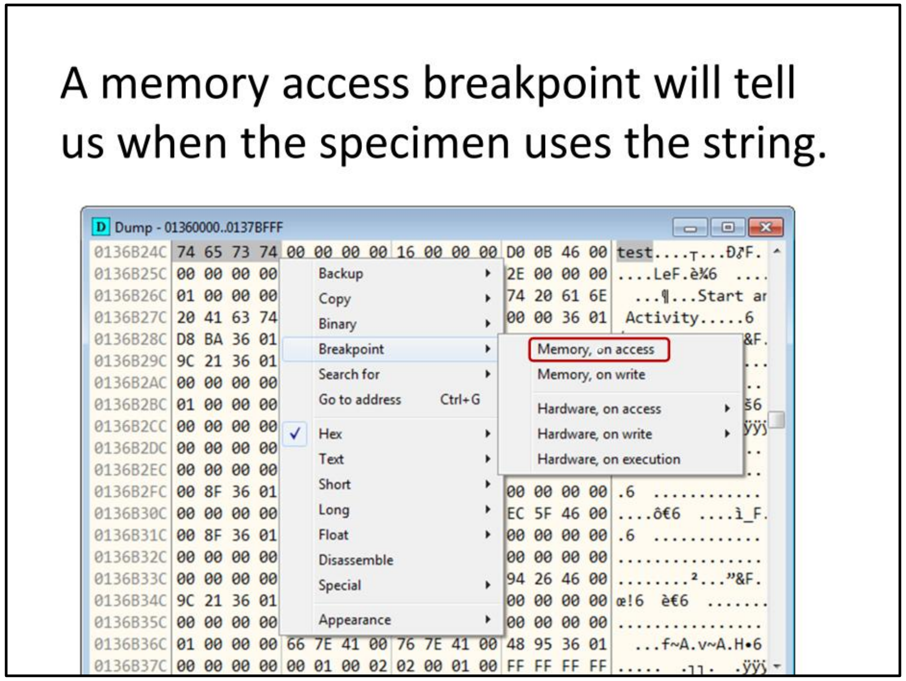
Memory map

| Address | Size | Owner | Section | Contains | Type | Access |
|---|---|---|---|---|---|---|
| 00010000 | 00010000 | | | | Map | RW |
| 00020000 | 00001000 | | | | Priv | RW |
| 0012A000 | 0000100 | | | | | RW Guarde |
| 0012B000 | 0000500 | | | | | RW Guarde |
| 00130000 | 0000400 | | | | | R |
| 00140000 | 0000200 | | | | | R |
| 00150000 | 0000100 | | | | | RW |

To make use of this technique, load the malicious program into OllyDbg, then run it. Once the trojan is running, press Alt+M to bring up the memory map in OllyDbg. This shows the listing of the memory segments mapped and used by the currently-debugged executable. To search the executable's memory for a particular string, press Ctrl+B in OllyDbg; then, enter your string. In this case, we'll enter "test" in the ASCII field of the dialog box. Then press Enter.

It is possible that your string will be located in several memory areas. The one you're interested in won't necessarily be the fist one. To repeat your search, click on the memory map window, then press Ctrl+L. (Don't forget to click on the memory map window!)

In the case of our example, we'll need to perform the initial search via Ctrl+B. This will find us an instance of "test" that is not promising. We will repeat the search by pressing Ctrl+L once.



## A memory access breakpoint will tell us when the specimen uses the string.

Dump - 01360000..0137BFFF

```
0136B24C 74 65 73 74 00 00 00 00 16 00 00 00 D0 0B 46 00 test....ㅜ...Ð♂F.
0136B25C 00 00 00 00                         2E 00 00 00 ....LeF.è¾6  ....
0136B26C 01 00 00 00                         74 20 61 6E   ...¶...Start ar
0136B27C 20 41 63 74                         00 00 36 01  Activity.....6
0136B28C D8 BA 36 01                                                    &F.
0136B29C 9C 21 36 01                                                    ...
0136B2AC 00 00 00 00                                                    ..
0136B2BC 01 00 00 00                                                    š6
0136B2CC 00 00 00 00                                                    ÿÿ
0136B2DC 00 00 00 00                                                    ..
0136B2EC 00 00 00 00                                                    ..
0136B2FC 00 8F 36 01                         00 00 00 00 .6  ...........
0136B30C 00 00 00 00                         EC 5F 46 00 ....ô€6  ....ì_F.
0136B31C 00 8F 36 01                         00 00 00 00 .6  ...........
0136B32C 00 00 00 00                         00 00 00 00 ...............
0136B33C 00 00 00 00                         94 26 46 00 ........²...”&F.
0136B34C 9C 21 36 01                         00 00 00 00 œ!6  è€6  .......
0136B35C 00 00 00 00                         00 00 00 00 ................
0136B36C 01 00 00 00 66 7E 41 00 76 7E 41 00 48 95 36 01   ...f~A.v~A.H•6
0136B37C 00 00 00 00 00 01 00 02 02 00 01 00 FF FF FF FF .....  ·ıı·  .ÿÿ
```

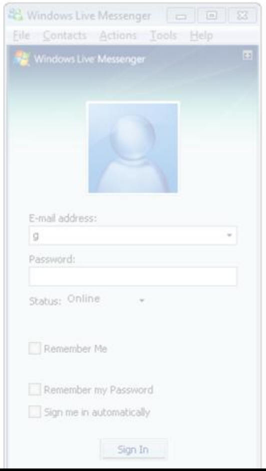
Now that we've located the string "test" in the trojan's memory, we can set a breakpoint there. In this case we'll be setting a memory access breakpoint, so that OllyDbg pauses the program's execution whenever it attempts to access this particular memory area. Effectively, this will allow us to catch the trojan while it is attempting to use the "test" string; we will then be able to see how it makes use of the string.

To set the brakpoint, highlight the exact characters of the string "test", then right-click and click "Breakpoint" > "Memory, on access".

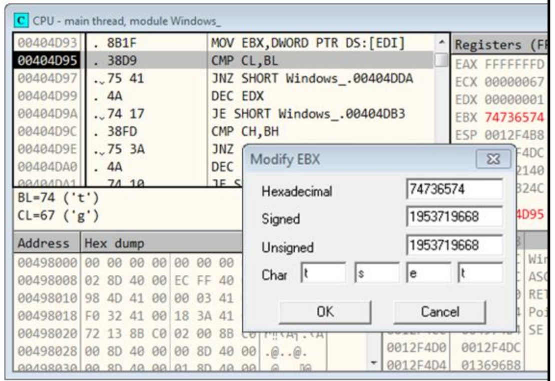
The trojan will continue to run. Now we can either wait for it to try using the sting, or attempt interacting with the program to try to cause it to use the string.

We can try interacting with the trojan by typing some text into its first field, the one labeled "E-mail address". If you type any character there after setting our memory breakpoint, you will immediately trigger the breakpoint, as you can see on the next slide.

## Interact with the program to try triggering the breakpoint.



E-mail address: g
Password:
Status: Online



```
00404D93  . 8B1F      MOV EBX,DWORD PTR DS:[EDI]
00404D95  . 38D9      CMP CL,BL
00404D97  .,75 41     JNZ SHORT Windows_.00404DDA
00404D99  . 4A        DEC EDX
00404D9A  .,74 17     JE SHORT Windows_.00404DB3
00404D9C  . 38FD      CMP CH,BH
00404D9E  .,75 3A     JNZ
00404DA0  . 4A        DEC

BL=74 ('t')
CL=67 ('g')

EAX FFFFFFFD
ECX 00000067
EDX 00000001
EBX 74736574
ESP 0012F4B8

00498000 00 00 00 00 00 00 00
00498008 02 8D 40 00 EC FF 40
00498010 98 4D 41 00 00 03 41
00498018 F0 32 41 00 18 3A 41
00498020 72 13 8B C0 02 00 8B
00498028 00 8D 40 00 00 8D 40 00 .@..@.
```

Modify EBX
Hexadecimal: 74736574
Signed: 1953719668
Unsigned: 1953719668
Char: t s e t

"t" in EBX is compared to "g" in ECX.

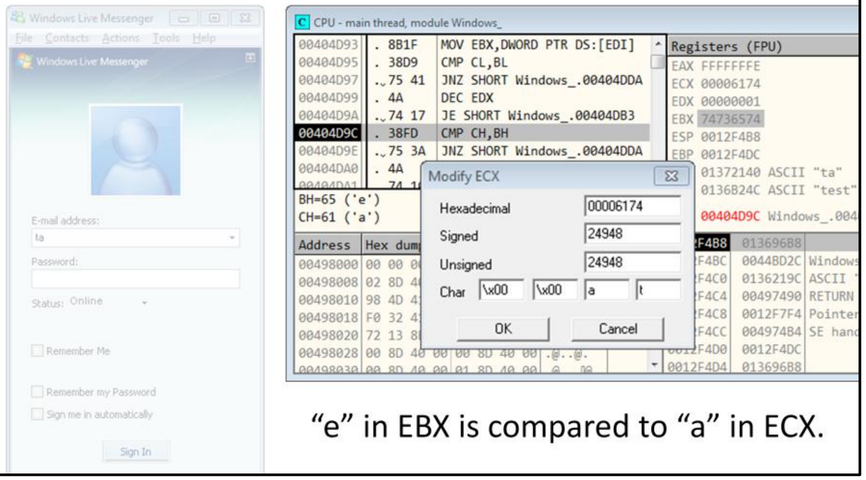
As you can see on the left side of this slide, I entered a character into the field. I picked a letter at random: "g". Right away, OllyDbg comes to the foreground, because we just triggered an attempt by the trojan to somehow use the string "test". You can now interact with the code, looking at its environment, and even running it as slowly as one instruction at a time.

To execute one instruction, press F8. To examine the run-time environment of the program, look at its registers in the top right corner of the OllyDbg window. A register is a specialized location on the CPU that can store data and that is very fast.

What's going on in this part of the code? Don't worry if you don't understand much of the assembly code you see there: this is just an introduction to malware analysis, so I'll walk you through the most important parts. OllyDbg has highlighted the instruction that will be executed next by the program, "CMP CL, BL". This compares contents of two registers, CL and BL. CL points to the lowest byte of ECX; BL points to the lowest byte of EBX, so it's an efficient way of comparing parts of ECX and EBX registers.

Double-click the registers to see their contents. ECX contains the character we entered, "g". EBX contains the string that our input is being compared to, "test" (it's stored backwards).

## Repeat the experiment. Enter "t" to pass the first test, then "a".



```
00404D93  . 8B1F     MOV EBX,DWORD PTR DS:[EDI]
00404D95  . 38D9     CMP CL,BL
00404D97  .,75 41    JNZ SHORT Windows_.00404DDA
00404D99  . 4A       DEC EDX
00404D9A  .,74 17    JE SHORT Windows_.00404DB3
00404D9C  . 38FD     CMP CH,BH
00404D9E  .,75 3A    JNZ SHORT Windows_.00404DDA
00404DA0  . 4A
BH=65 ('e')
CH=61 ('a')
```

```
Registers (FPU)
EAX FFFFFFFE
ECX 00006174
EDX 00000001
EBX 74736574
ESP 0012F4B8
EBP 0012F4DC
    01372140 ASCII "ta"
    0136B24C ASCII "test"

    00404D9C Windows_.004
```

Modify ECX

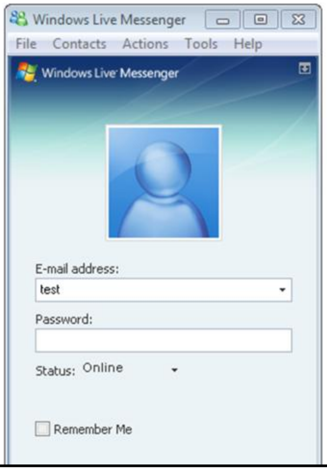
| | |
|---|---|
| Hexadecimal | 00006174 |
| Signed | 24948 |
| Unsigned | 24948 |
| Char | \x00 \x00 a t |

"e" in EBX is compared to "a" in ECX.

Press F9 to continue executing the trojan. Delete the "g" character you've entered previously. This time, let the program match the first character of the "test" string, and see how it compares the second character. To do this, enter "ta" in the "E-mail address" box. If you keep triggering the breakpoint, press F9 to continue. You want to pause right after you've had a chance to type "ta".

Press F8 to execute one instruction after you've triggered the breakpoint, just like you did previously. This time, if you look at contents of ECX and EBX registers, you'll notice that the trojan is comparing the character "a" that we entered to the character "e" that it seems to expect. That's because the CH register points to the second lowest byte of ECX; the BH register points to the second lowest byte of EBX.

## Looks like the specimen is comparing contents of the email field to “test”.



E-mail address: test

Password:

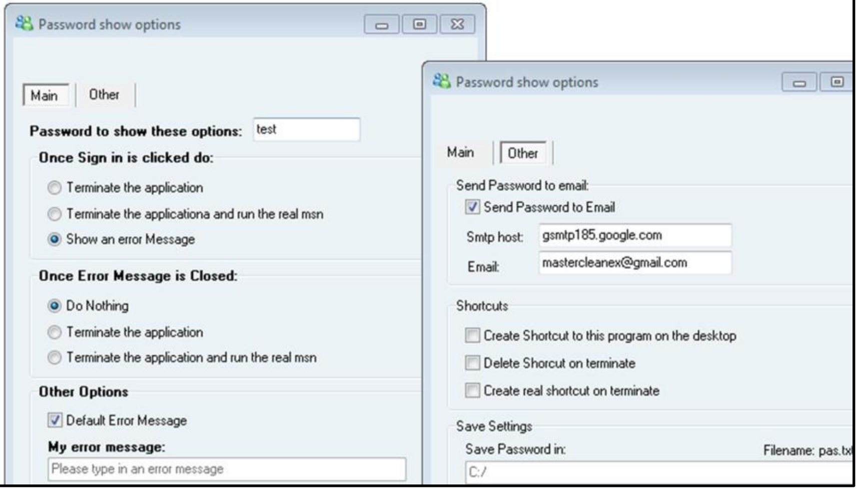
Status: Online

☐ Remember Me

Enter “test” in the field to see what happens (outside the debugger).

So, the trojan seems to be looking for the string “test” in the “E-mail address” field. Exit the debugger, launch the trojan by itself, and enter “test” to see what happens.

## It seems that entering "test" activates configuration screens.



Password to show these options: test

Once Sign in is clicked do:

- ○ Terminate the application
- ○ Terminate the applicationa and run the real msn
- ● Show an error Message

Once Error Message is Closed:

- ● Do Nothing
- ○ Terminate the application
- ○ Terminate the application and run the real msn

Other Options

- ☑ Default Error Message

My error message: Please type in an error message

Send Password to email:

- ☑ Send Password to Email

Smtp host: gsmtp185.google.com

Email: mastercleanex@gmail.com

Shortcuts

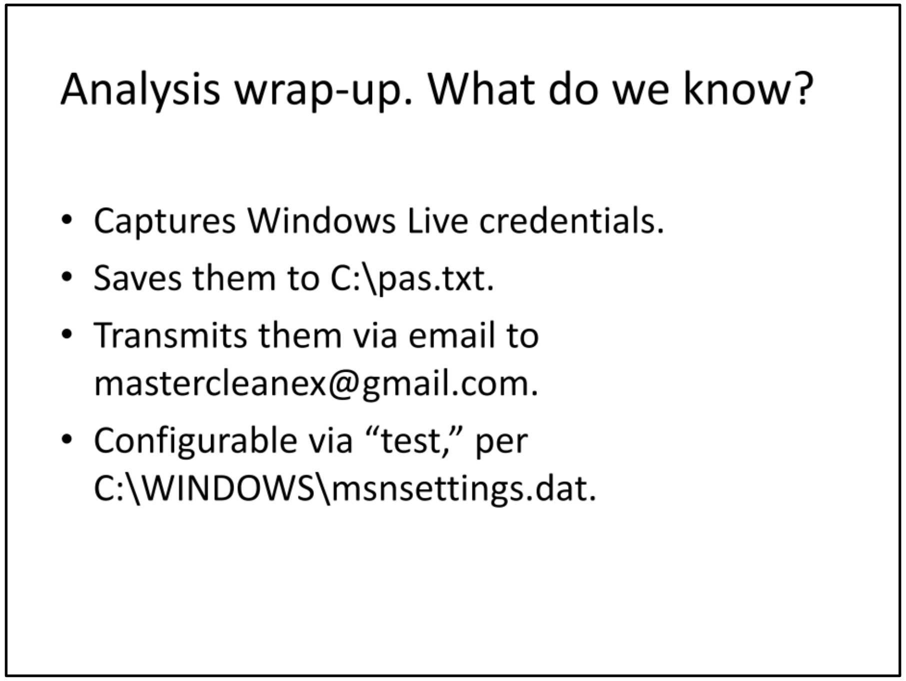
- ☐ Create Shortcut to this program on the desktop
- ☐ Delete Shorcut on terminate
- ☐ Create real shortcut on terminate

Save Settings

Save Password in: C:/

Filename: pas.tx

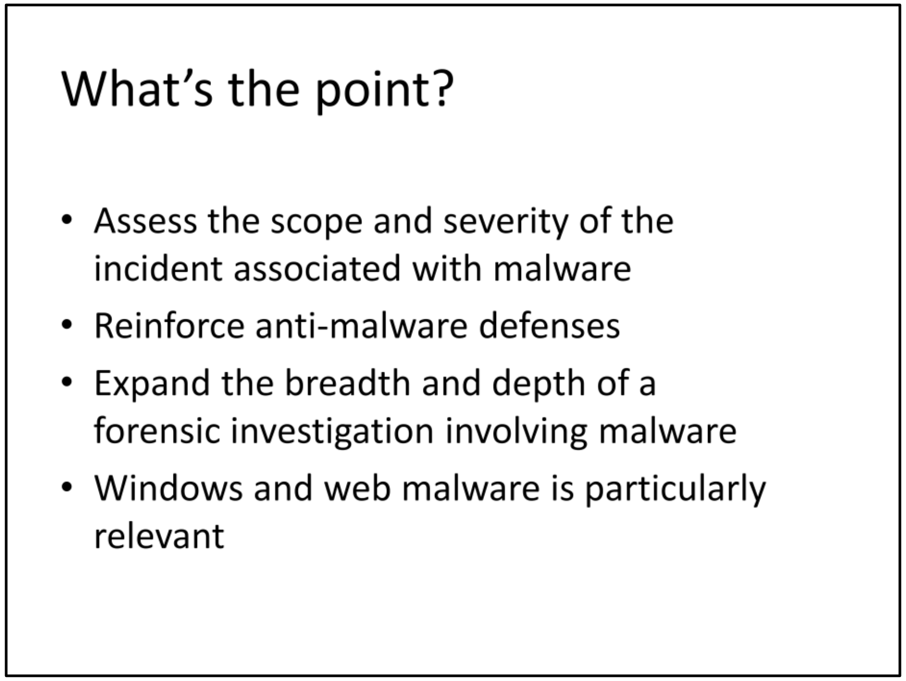
Voila! When you enter "test", the trojan brings you to a brand new screen that seems to allow you to configure the trojan's operation. As you can see on this slide, the configuration options let you define the passphrase to activate this string, the address where the trojan will send captured logon credentials, etc.



## Analysis wrap-up. What do we know?

- Captures Windows Live credentials.
- Saves them to C:\pas.txt.
- Transmits them via email to mastercleanex@gmail.com.
- Configurable via “test,” per C:\WINDOWS\msnsettings.dat.

It’s time to wrap up our analysis. What have we learned about the trojan through the steps I demonstrated? We established that the malware specimen captures the victim’s Windows Live credentials entered into the trojan version of Windows Live Messenger. It saves the username and password to a local file, and then sends it to the attacker via Gmail. We also identified a file, msnsettings.dat, which the trojan uses to store its configuration. The attacker can customize the configuration by typing “test” into the “E-mail address” field of the trojan; this keyword is based on the previously-saved contents of msnsettings.dat.

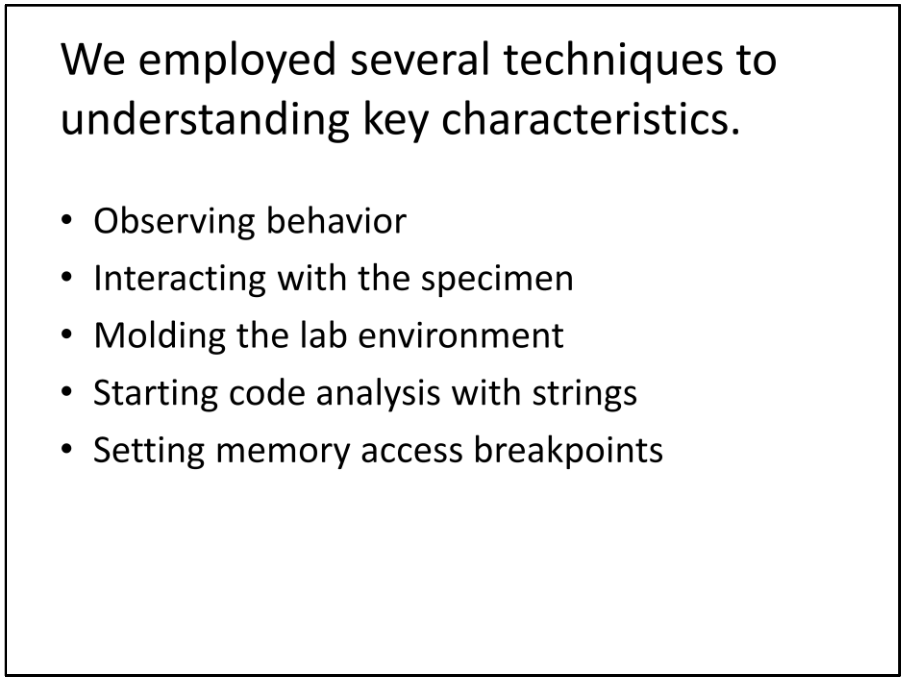


## What's the point?

- Assess the scope and severity of the incident associated with malware
- Reinforce anti-malware defenses
- Expand the breadth and depth of a forensic investigation involving malware
- Windows and web malware is particularly relevant

Great, we learned a bunch of details about a malware sample. What's the point? My goal was not to teach you about this particular trojan's capabilities. Instead, I wanted to use it as the context for introducing you to the key concepts behind reverse-engineering malicious software.

The results of malware analysis are very useful for security, systems, and network professionals. The findings can help during incident response and forensic investigation. They can also help you fine-tune your defensive mechanisms, and help you create intrusion detection signatures for locating the specimen across your enterprise.



## We employed several techniques to understanding key characteristics.

- Observing behavior
- Interacting with the specimen
- Molding the lab environment
- Starting code analysis with strings
- Setting memory access breakpoints

The general malware analysis approach, which I described in this presentation, included behavioral and code analysis phases.

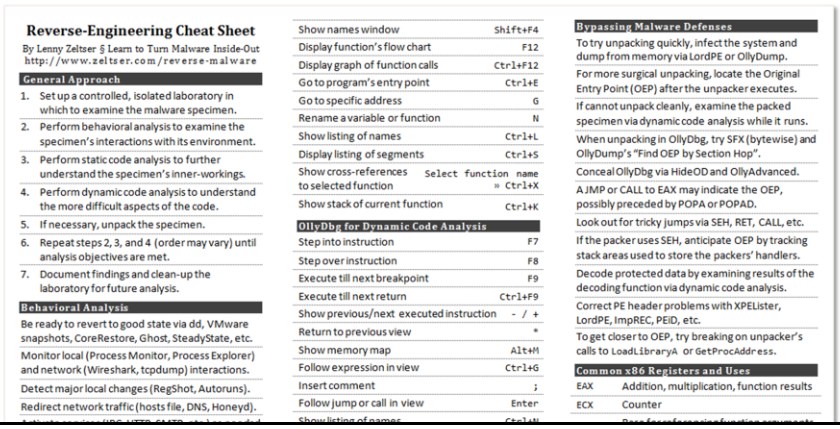
We began by observing the specimen's behavior in an isolated lab using several monitoring tools. We used our observations to determine how to interact with the trojan, which produced additional results. We were able to evoke additional malicious characteristics by gradually molding the laboratory environment to match the world within which the specimen expected to operate.

Armed with an initial understanding of the program's capabilities, we employed code analysis to further understand the program's characteristics. We began this phase by looking how the program uses interesting strings, and employed memory access breakpoints to identify areas of the code worth examining further.

The best way to reinforce the techniques I discussed here is to try the analysis on your own. This document includes links to the tools I used. You can also download a copy of the trojan on the website that hosts this presentation and the corresponding webcast: http://tinyurl.com/malcast. The full version of the URL is: http://zeltser.com/reverse-malware/malware-analysis-webcast.html.

## The reverse-engineering cheat sheet summarizes the approach.

http://tinyurl.com/reverse-malware-sheet



**Reverse-Engineering Cheat Sheet**

By Lenny Zeltser § Learn to Turn Malware Inside-Out
http://www.zeltser.com/reverse-malware

**General Approach**

1. Set up a controlled, isolated laboratory in which to examine the malware specimen.
2. Perform behavioral analysis to examine the specimen's interactions with its environment.
3. Perform static code analysis to further understand the specimen's inner-workings.
4. Perform dynamic code analysis to understand the more difficult aspects of the code.
5. If necessary, unpack the specimen.
6. Repeat steps 2, 3, and 4 (order may vary) until analysis objectives are met.
7. Document findings and clean-up the laboratory for future analysis.

**Behavioral Analysis**

Be ready to revert to good state via dd, VMware snapshots, CoreRestore, Ghost, SteadyState, etc.

Monitor local (Process Monitor, Process Explorer) and network (Wireshark, tcpdump) interactions.

Detect major local changes (RegShot, Autoruns).

Redirect network traffic (hosts file, DNS, Honeyd).

| | |
|---|---|
| Show names window | `Shift+F4` |
| Display function's flow chart | `F12` |
| Display graph of function calls | `Ctrl+F12` |
| Go to program's entry point | `Ctrl+E` |
| Go to specific address | `G` |
| Rename a variable or function | `N` |
| Show listing of names | `Ctrl+L` |
| Display listing of segments | `Ctrl+S` |
| Show cross-references to selected function | `Select function name » Ctrl+X` |
| Show stack of current function | `Ctrl+K` |

**OllyDbg for Dynamic Code Analysis**

| | |
|---|---|
| Step into instruction | `F7` |
| Step over instruction | `F8` |
| Execute till next breakpoint | `F9` |
| Execute till next return | `Ctrl+F9` |
| Show previous/next executed instruction | `- / +` |
| Return to previous view | `*` |
| Show memory map | `Alt+M` |
| Follow expression in view | `Ctrl+G` |
| Insert comment | `;` |
| Follow jump or call in view | `Enter` |

**Bypassing Malware Defenses**

To try unpacking quickly, infect the system and dump from memory via LordPE or OllyDump.

For more surgical unpacking, locate the Original Entry Point (OEP) after the unpacker executes.

If cannot unpack cleanly, examine the packed specimen via dynamic code analysis while it runs.

When unpacking in OllyDbg, try SFX (bytewise) and OllyDump's "Find OEP by Section Hop".

Conceal OllyDbg via HideOD and OllyAdvanced.

A JMP or CALL to EAX may indicate the OEP, possibly preceded by POPA or POPAD.

Look out for tricky jumps via SEH, RET, CALL, etc.

If the packer uses SEH, anticipate OEP by tracking stack areas used to store the packers' handlers.

Decode protected data by examining results of the decoding function via dynamic code analysis.

Correct PE header problems with XPELister, LordPE, ImpREC, PEiD, etc.

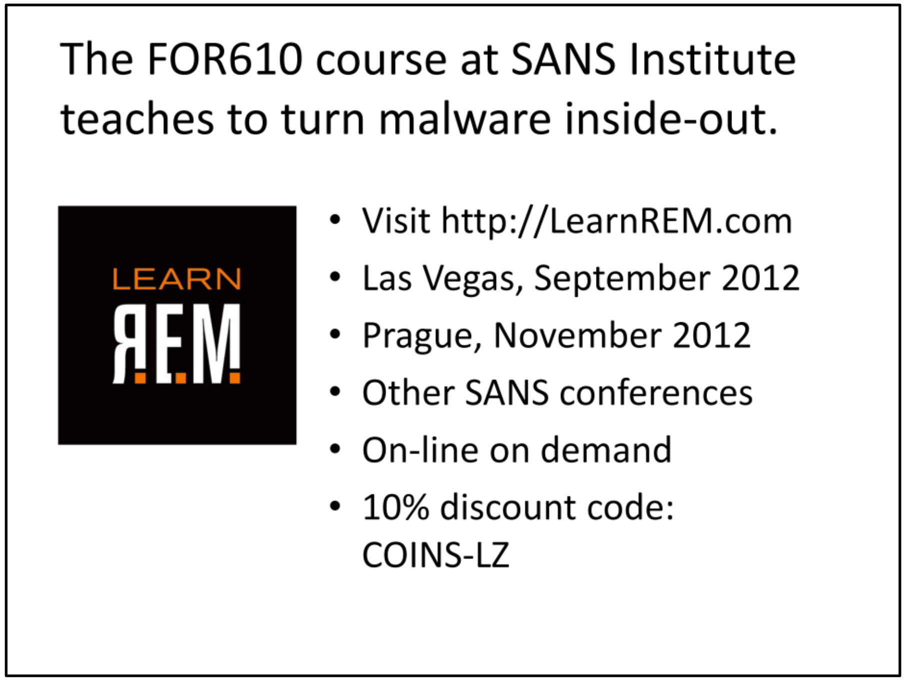
To get closer to OEP, try breaking on unpacker's calls to `LoadLibraryA` or `GetProcAddress`.

**Common x86 Registers and Uses**

| | |
|---|---|
| EAX | Addition, multiplication, function results |
| ECX | Counter |

To help you master malware reverse-engineering skills, I created a one-page cheat sheet, which you can download and customize freely. It's available at http://tinyurl.com/reverse-malware-sheet. The full version of the URL is: http://zeltser.com/reverse-malware/reverse-malware-cheat-sheet.html.

You may also find my other security cheat sheets useful. You'll find them at: http://zeltser.com/cheat-sheets.



## The FOR610 course at SANS Institute teaches to turn malware inside-out.

LEARN REM

- Visit http://LearnREM.com
- Las Vegas, September 2012
- Prague, November 2012
- Other SANS conferences
- On-line on demand
- 10% discount code: COINS-LZ

My hope is that you'll find this topic as fascinating as I do. If you'd like to learn more about how to reverse-engineer malware, consider taking my SANS Institute course called Reverse-Engineering Malware: Malware Analysis Tools and Techniques (FOR610), and you can read all about it at: http://LearnREM.com.

The course teaches how to understand key characteristics of malware that runs on or targets Microsoft Windows systems. This includes both executable files compiled to run natively on Windows, as well as browser-based malware, such as malicious JavaScript or Flash files. The course makes use of the tools installed on REMnux, as well as those that run on Microsoft Windows.

**Lenny Zeltser**

blog.zeltser.com

twitter.com/lennyzeltser

If you want to keep an eye on my research and activities, take a look at blog.zeltser.com. I'm also on Twitter at twitter.com/lennyzeltser.

A bit about me:

I'm a seasoned IT professional with a strong background in information security and business management. As a product management director at NCR Corporation, I focus on safeguarding IT operations of small and midsize businesses. Before NCR, I led an enterprise security consulting team at a major IT hosting provider.

My most recent work has focused on malware defenses and cloud-based services. I teach how to analyze and combat malware at SANS Institute, where I am a senior faculty member. I also participate as a Board of Directors member at SANS Technology Institute and volunteer as an incident handler at the Internet Storm Center.

I often speak on security and related business topics at conferences and industry events, write articles, and have co-authored books on forensics, network security and malware.