SANS

www.sans.org

FORENSICS 610

REVERSE-ENGINEERING MALWARE: MALWARE ANALYSIS TOOLS AND TECHNIQUES

610.1

Malware Analysis Fundamentals

The right security training for your staff, at the right time, in the right location.

IMPORTANT-READ CAREFULLY:

This Courseware License Agreement ("CLA") is a legal agreement between you (either an individual or a single entity; henceforth User) and the SANS Institute for the personal, non-transferable use of this courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA. If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware. BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. IF YOU DO NOT AGREE YOU MAY RETURN IT TO THE SANS INSTITUTE FOR A FULL REFUND, IF APPLICABLE. The SANS Institute hereby grants User a non-exclusive license to use the material contained in this courseware subject to the terms of this agreement. User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of this publication in any medium whether printed, electronic or otherwise, for any purpose without the express written consent of the SANS Institute. Additionally, user may not sell, rent, lease, trade, or otherwise transfer the courseware in any way, shape, or form without the express written consent of the SANS Institute.

The SANS Institute reserves the right to terminate the above lease at any time. Upon termination of the lease, user is obligated to return all materials covered by the lease within a reasonable amount of time.

V2010_1226

Note to Students
About SANS Institute Evaluations

SANS Institute works extremely hard to continuously improve both the quality of courseware and instruction. The evaluation forms that we provide daily are one of the most effective methods to help improve the course and to give feedback to the instructors. The numerical score helps us track trends, and the written comments give us the context to make immediate improvements. We truly appreciate the time you take to provide this feedback.

Some things to remember: 10 = very good, 1 = very bad!

Your scores and comments should be given based on your experience, the quality of the instruction, and the quality of the course material. Instructors should NOT be soliciting scores or in any way trying to influence your evaluations.

poor

1 2 3 4 5 6 7 8 9 10

Value of Course Content:

poor **excellent**

1 2 3 4 5 6 7 8 9 10

If you liked the course and wouldn't mind sharing your comments with others who might consider taking this course in the future, please write a sentence about why you felt this course was valuable.

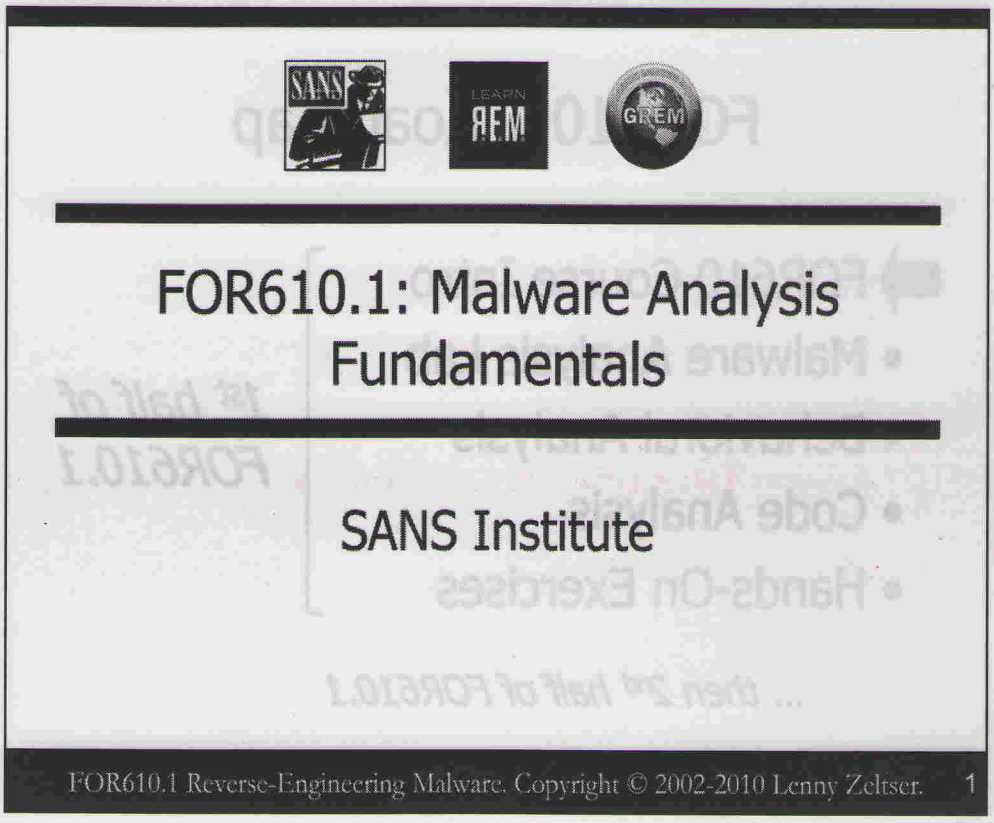
Please make every effort to score each of these areas individually and write comments which you feel are relevant or important in the comments section. If a lab does not work on your computer system, please reflect that in the course content score, and please give us any information you have as to what went wrong so we make appropriate corrections to our courseware. Rest assured that all eval comments are read by the senior management at SANS. Evaluations are used to adapt the course and make it even better, so please do fill them in! If you have specific comments regarding the venue, snacks, etc., we have provided a separate evaluation specifically for those areas.

Some people do not believe in giving 10s, and that is fine. However, please keep in mind that delivering a course is a performance, and performers work harder when they are motivated. If and only if your course is on par with the best instruction you have ever received a 10 is appropriate.

If you give the overall course and/or course content a low score, please explain why in the comments field so we can adapt. You can also be assured if you explain your concerns to an event manager we will be discreet. If you are willing, please sign your name at the bottom of the evaluation. If you are having difficulties, SANS can quickly come to your aid if we know who you are. In addition, we also routinely use quotes from previous students in upcoming brochures. If you do not want to be quoted, please let us know.

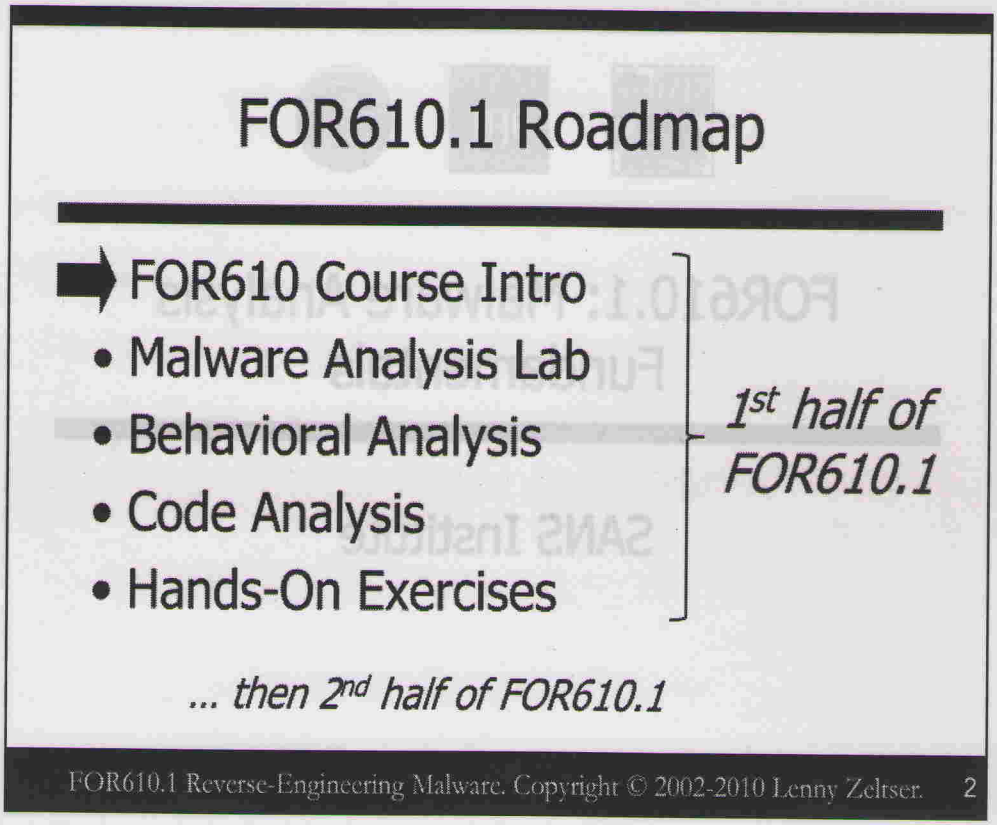
Thank you for your thoughtful consideration,

Eric Bassel, Mason Brown, Stephen Northcutt, and Alan Paller

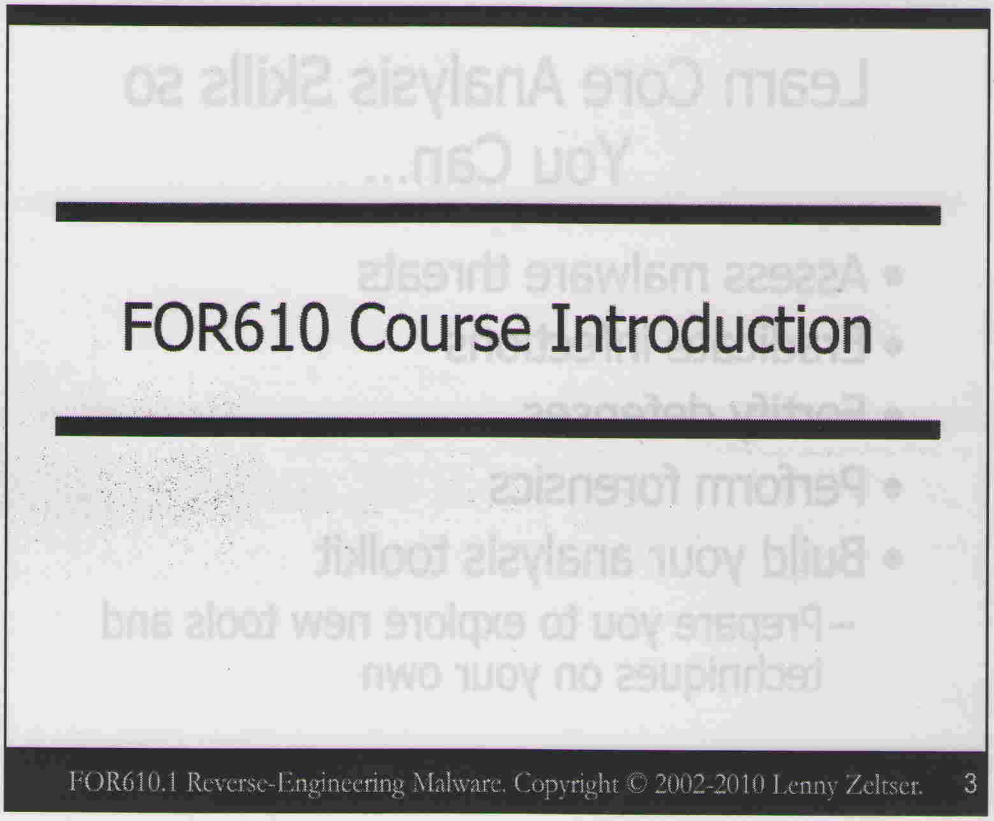


FOR610.1 lays the groundwork for the course by presenting the key tools and techniques malware analysts use to examine malicious programs. You will learn how to save time by exploring malware in two phases. Behavioral analysis focuses on the specimen's interactions with its environment, such as the registry, the network, and the file system; code analysis focuses on the specimen's code and makes use of a disassembler and a debugger. You will learn how to build a flexible laboratory to perform such analysis in a controlled manner, and will set up such a lab on your laptop. Also, we will jointly analyze a malware sample to reinforce the concepts and tools discussed throughout the day.

The materials in this section were created by Lenny Zeltser, and incorporate feedback and recommendations from FOR610 course participants. To learn about Lenny's background and other projects, please visit his website at http://zeltser.com or connect with him on Twitter at http://twitter.com/lennyzeltser.



The FOR610.1 course module is split in two halves. In the fist half, we will begin by looking at a classic incident where malware reverse-engineering skills would come in handy. We will then examine fundamental approaches to malware analysis, and get to know the core tools that we will use in this course. The second half of FOR610.1 will reinforce and expand the skills we learn in the first half, and provide you with several in-depth hands-on exercises.

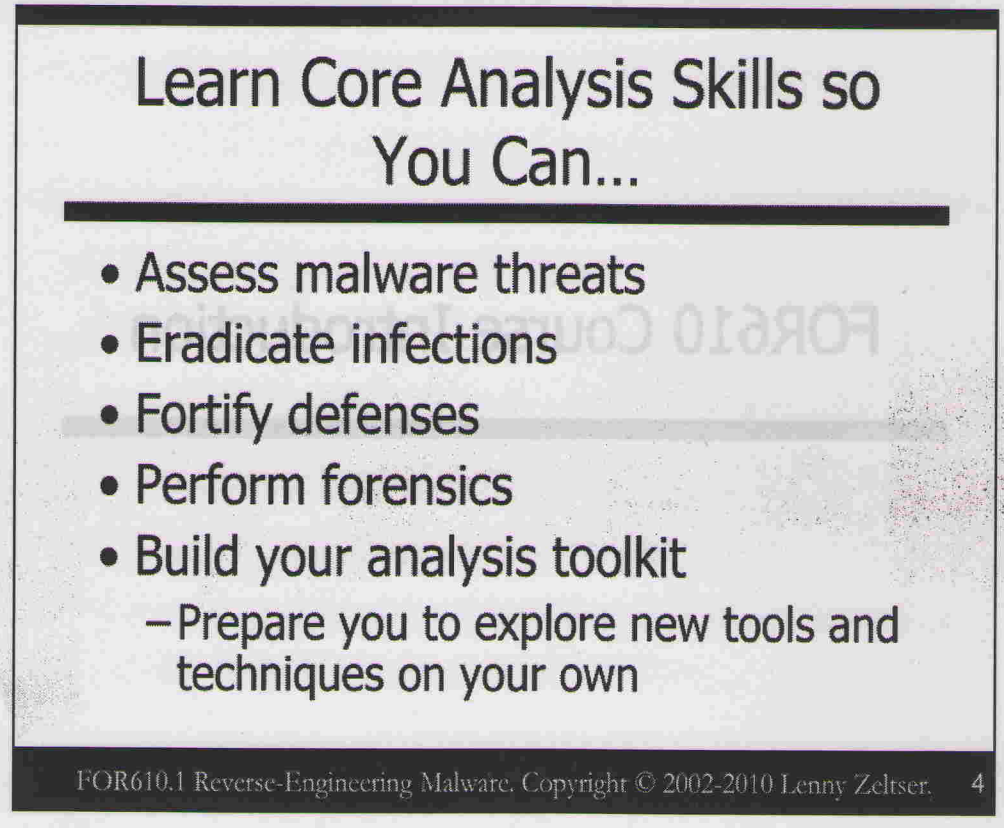


A few words from the course author:

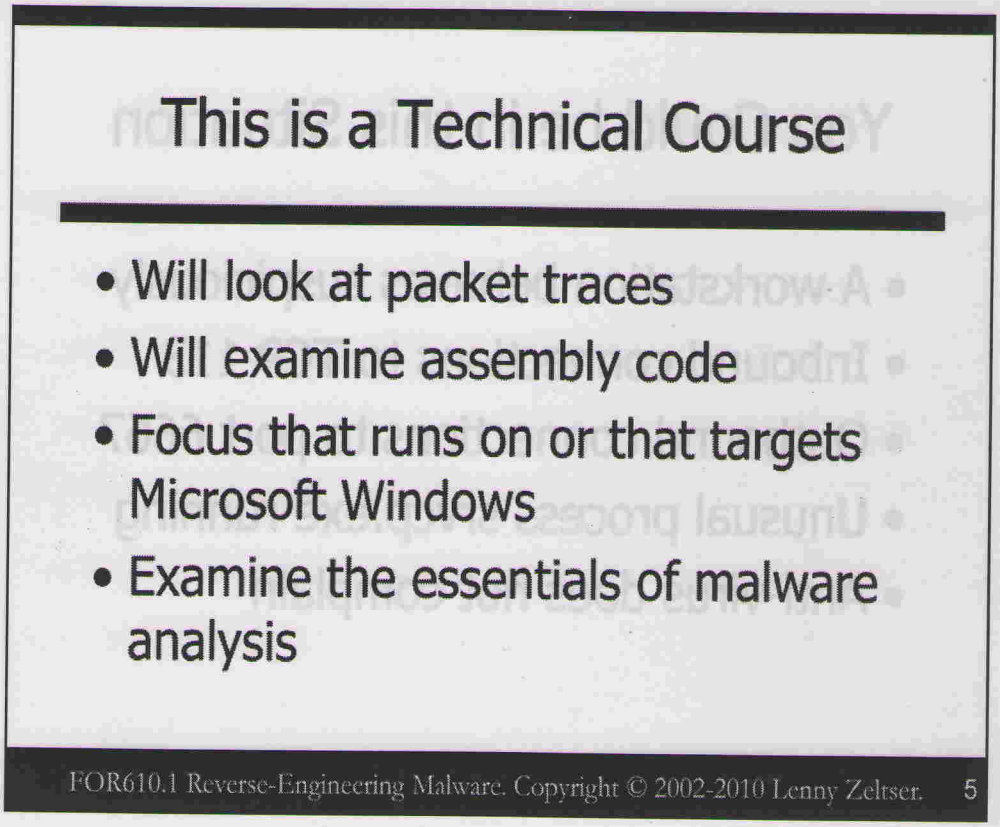
Before we get to the technical content, I'd like to briefly discuss what you can expect from this FOR610 course.

In this course, we will examine the various techniques that will equip us with fighting the threat of malware by examining inner-workings of malicious code. However, the reverse-engineering process is an arms race. As soon as we, the defenders, master a particular technique, the attackers take the appropriate measures to make that technique a little less effective. It's important to keep experimenting and learning about new tools and analysis approaches.

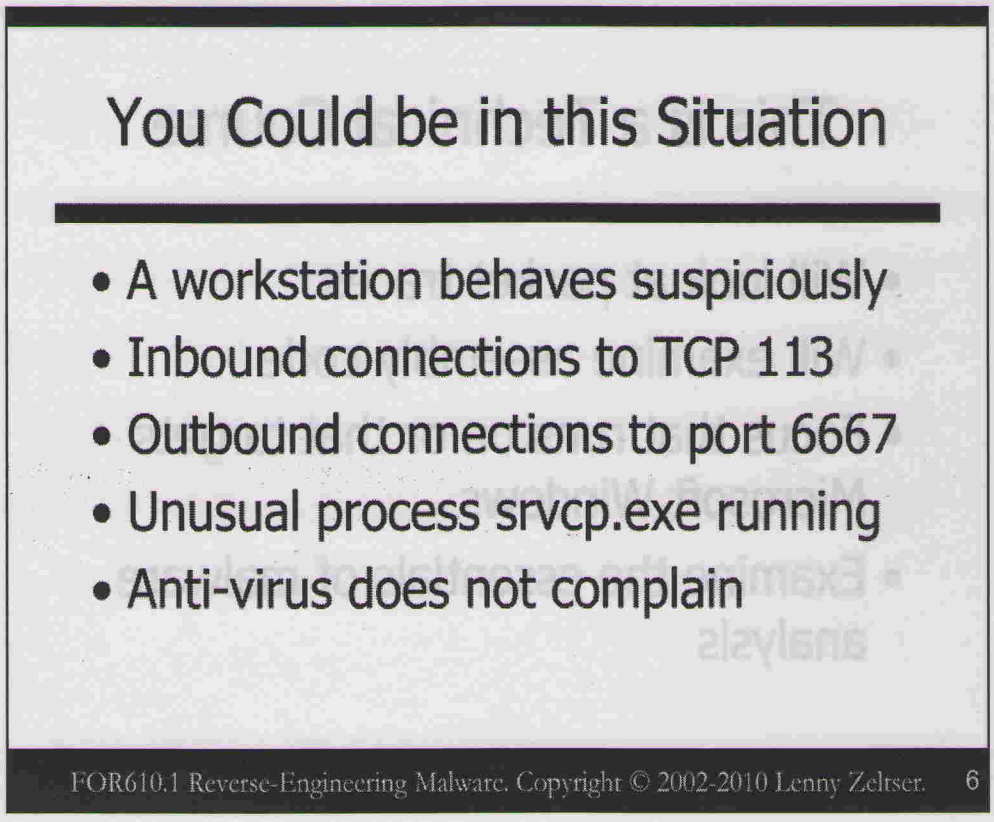
When you examine malware after participating in this course, please let me know when you come across an interesting malicious characteristic, or when you discover an analysis technique or tool that you find useful. This course grows based on feedback like that. You can drop me a note at lenny@zeltser.com or reach me at http://twitter.com/lennyzeltser. I'd love to hear from you!



There are several specific goals I have in mind when teaching this course. I want you to learn malware analysis approaches that help you respond to incidents, improve your ability to perform forensic analysis, and fortify your defenses. In the process, you will build your own reverse-engineering toolkit. Since there is no way a course like this could anticipate every malware analysis challenge you will encounter in the real world, I have structured the material so that it prepares you to explore new analysis tools and techniques on your own according to your interests and specific requirements.

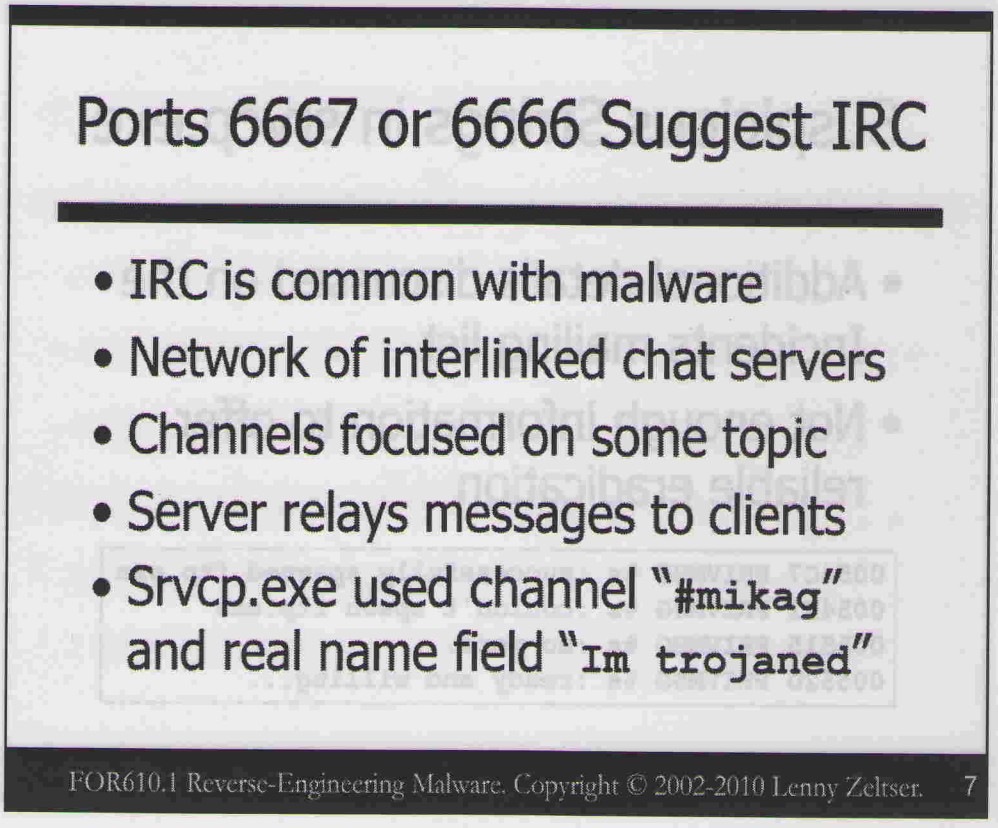


I will be covering a lot of technical details such as packet traces and assembly code. If you are not very familiar with these topics, you should be able to follow the course anyway, but I wanted to warn you that we will definitely be digging pretty deeply into the internals of malicious software. Also, while the core concepts presented here will be applicable to both Unix and Windows platforms, the majority of the examples and tools that I show will focus on analyzing malware that either runs on or that targets Microsoft Windows systems.

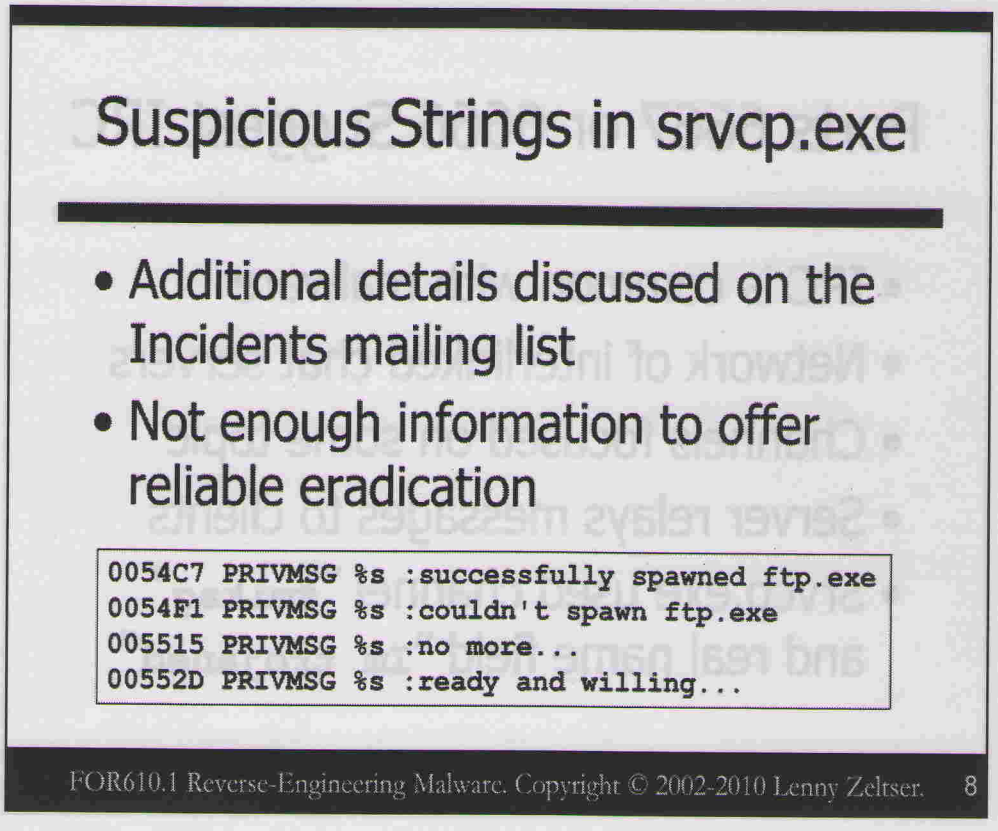


In a posting to the Incidents mailing list, Jeremy L. Gaddis reported noticing inbound connection attempts to TCP port 113 from an unfamiliar host on the Internet, as well as unauthorized outbound connection attempts to a remote server on destination TCP port 6667.

Later examination revealed an unusual process "srvcp.exe" running on the machine, although none of the three anti-virus software packages that Jeremy tried detected anything suspicious. Jeremy knew that TCP port 6667 is often used for connecting to IRC servers, and proceeded to investigate.

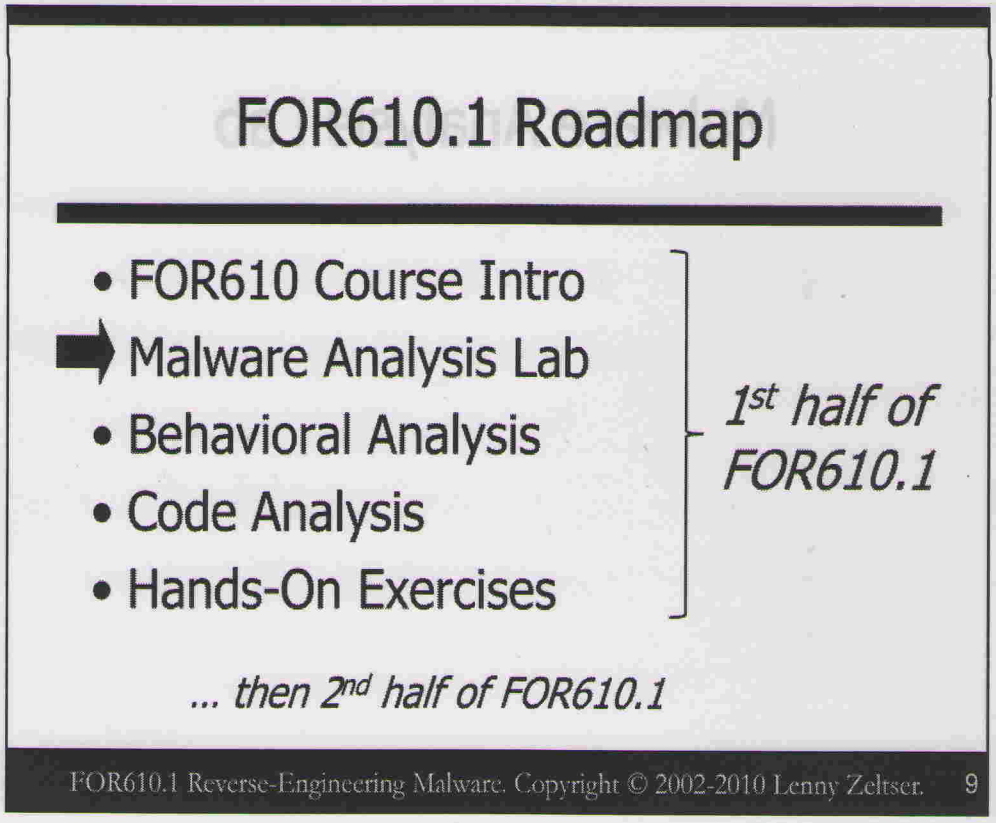


For those not familiar with IRC, let us mention that IRC can be viewed as a network of interlinked servers that allows users to hold real-time online conversations. Participants of a conversation typically join a channel devoted to a particular topic or interest. This is accomplished by having the user's IRC client connect to a server that participates in the desired IRC peering network. When a user types a message meant to be seen by channel participants, it is sent to the IRC server, and the server relays the message to participating clients, as specified in the Request for Comments (RFC) document 1459. IRC clients typically communicate with IRC servers over TCP ports 6666 or 6667. (See http://www.faqs.org/rfcs/rfc1459.html for more details.)

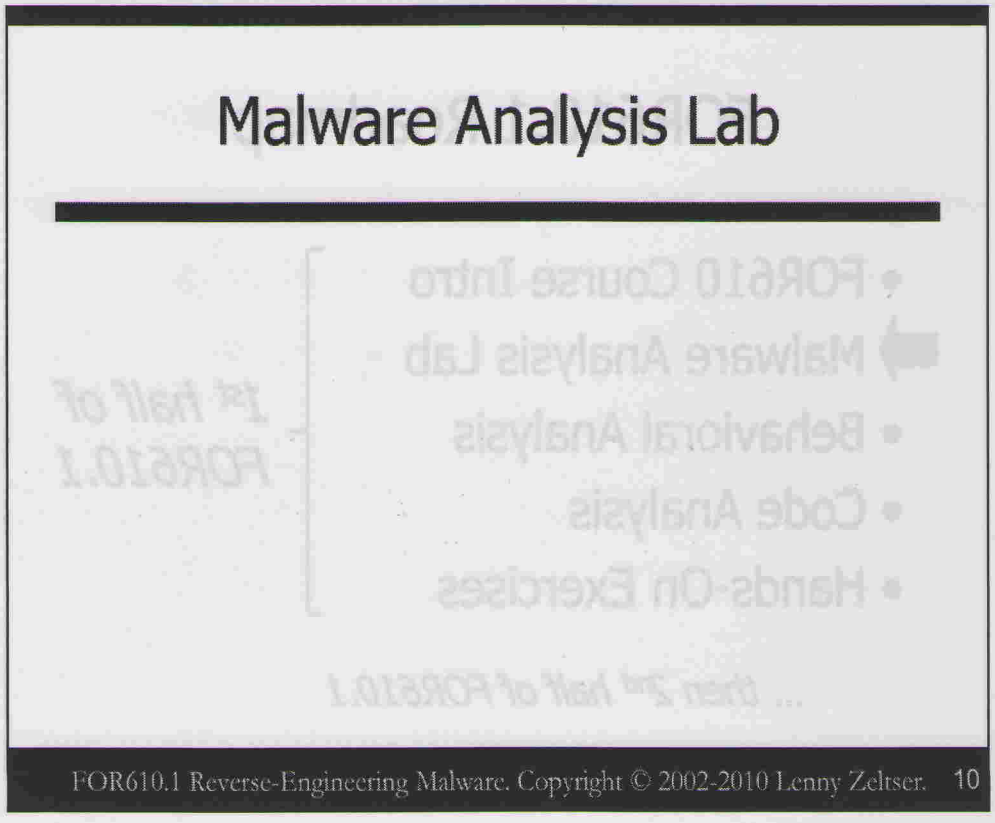


A few days after Jeremy's initial message to the mailing list, Brandon Kittler relayed his experience regarding this trojan in his posting to the list. Brandon supplied details regarding the location of the program's executable and the associated registry entry. Based on his observations and examination of strings present in the executable, he was able to conclude that the trojan was able to receive potentially dangerous commands via IRC.

We're going to re-visit this incident a bit later in this section. Before we proceed to analyze srvcp.exe in greater detail, let's take a look at the general analysis methodology that will help us reverse-engineer malicious software.

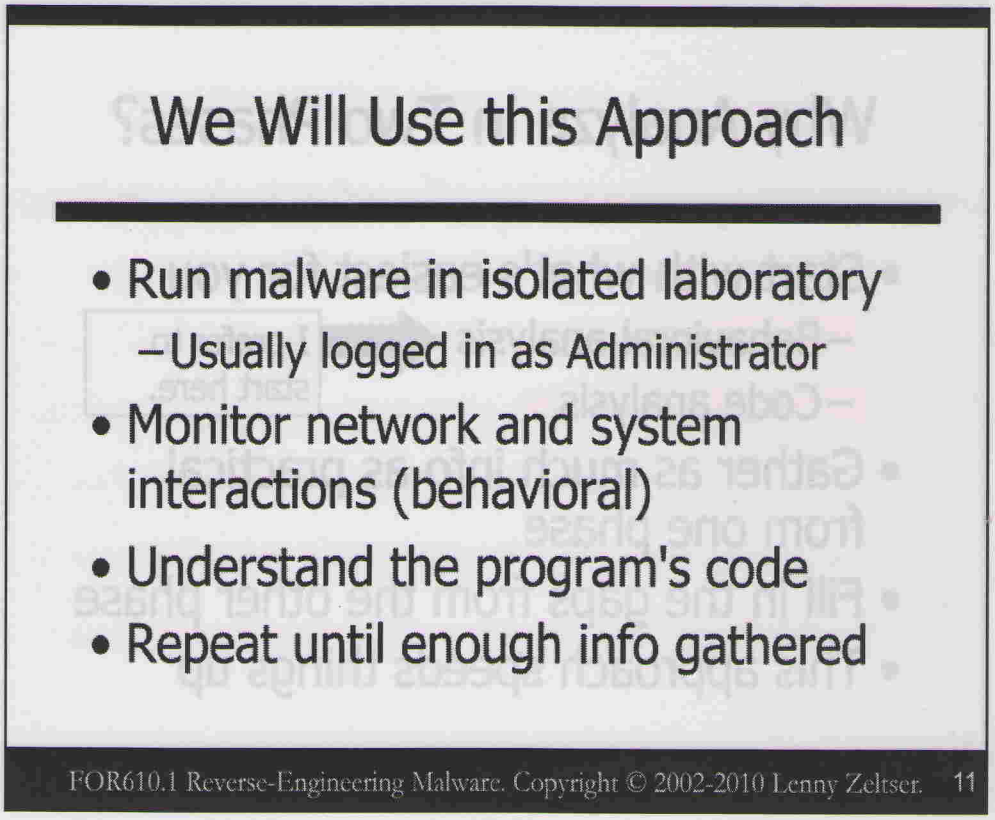


With the FOR610 course introduction behind us, let's see how to build a malware analysis lab to support our reverse-engineering goals.



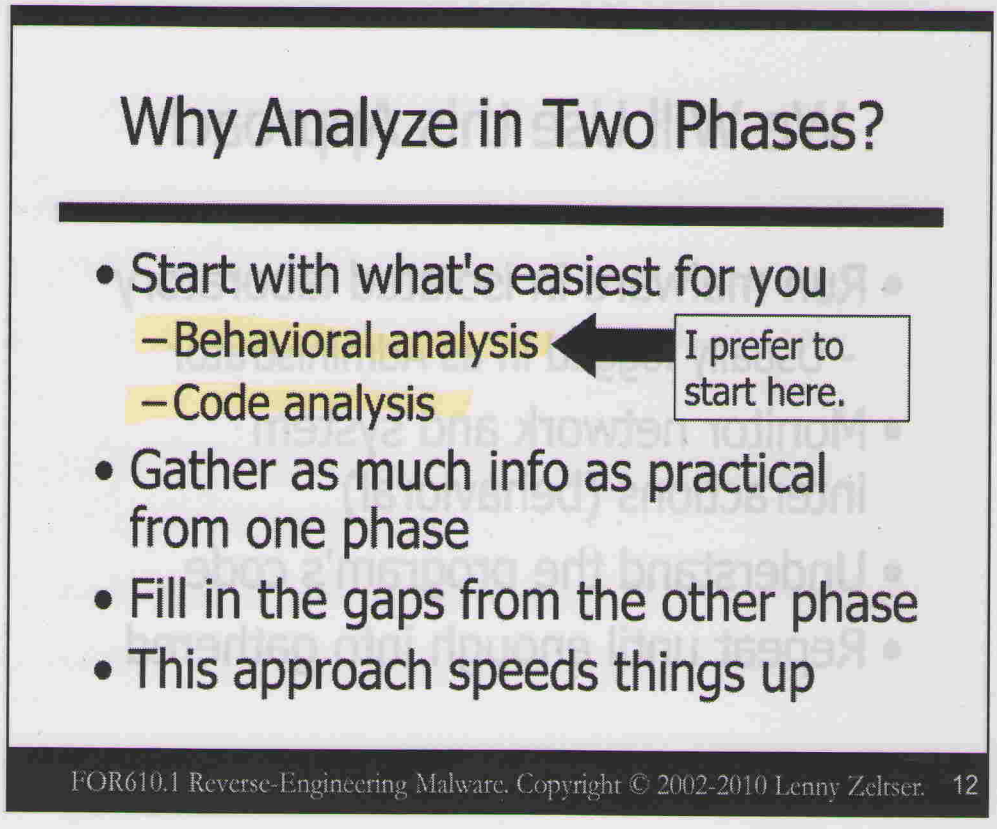
The incident described earlier might resemble a situation where you have obtained some information about a malware specimen that you need to analyze further. You are likely to be limited in financial and hardware resources, and will want to create an inexpensive laboratory in which you can safely observe the trojan's behavior and reverse engineer its functionality. I believe that this approach can be applied to analyzing a wide range of malicious software, and hope that you find it useful for your own needs.

The next series of slides will describe the proposed laboratory configuration and will introduce some of my favorite tools for analyzing malware. Later in the course I will demonstrate how you can use this toolset to analyze viruses, worms, and trojans.

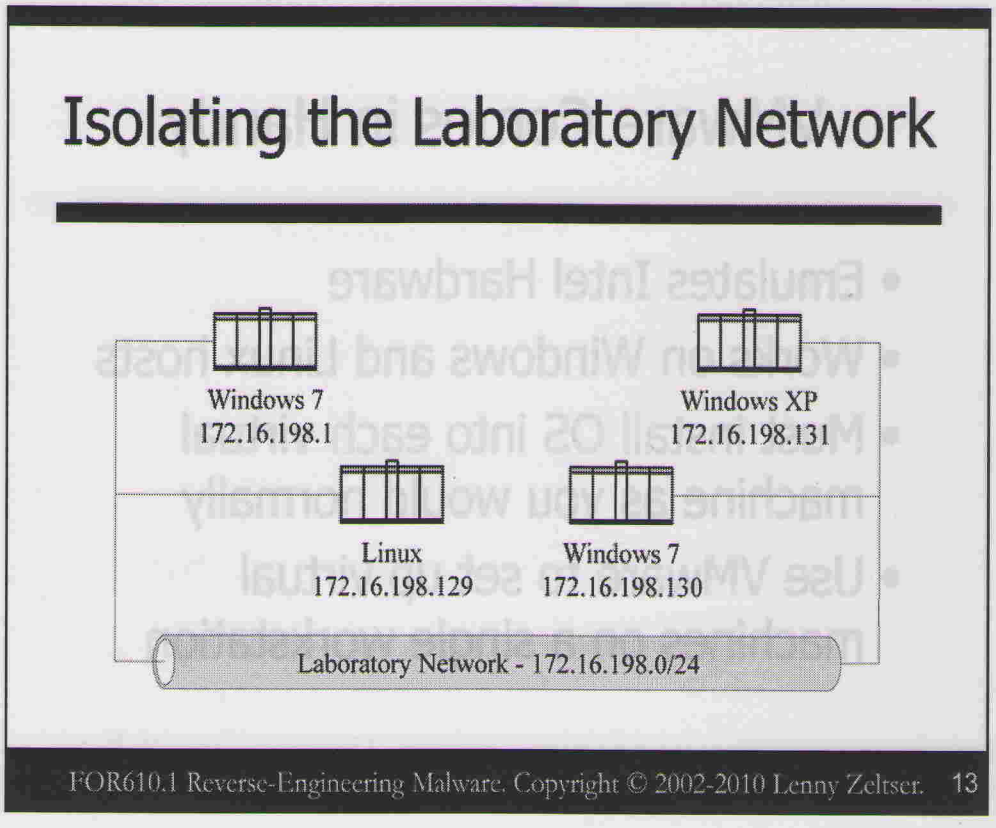


One of the ways to understand the threat associated with a malicious software specimen is to begin by examining its behavior in a controlled environment. This typically involves running the program in a laboratory and studying its actions as it interacts with computer resources and responds to various stimuli. To facilitate an efficient, inexpensive, and reliable research process, we need to have access to a controlled laboratory environment that is flexible and unobtrusive. Because of the trial-and-error nature of most reverse-engineering attempts, the environment needs to facilitate repeating experiments in a deterministic and reasonably easy fashion.

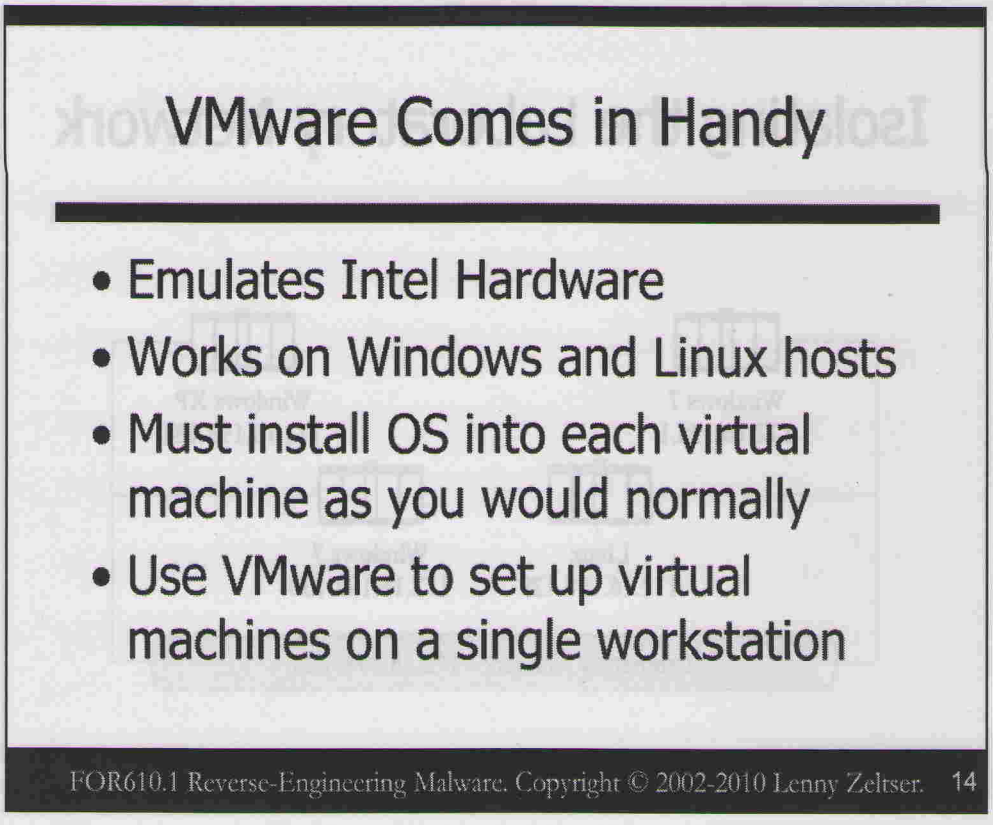
When infecting the laboratory system, you'll usually want to be logged into the targeted Windows system with administrator privileges, to see the full effects malware will have on the system. To mimic other scenarios, log in to the system using the account with the appropriate privileges. In this course, though, we'll be logged in as Administrator to let malware achieve its full potential.



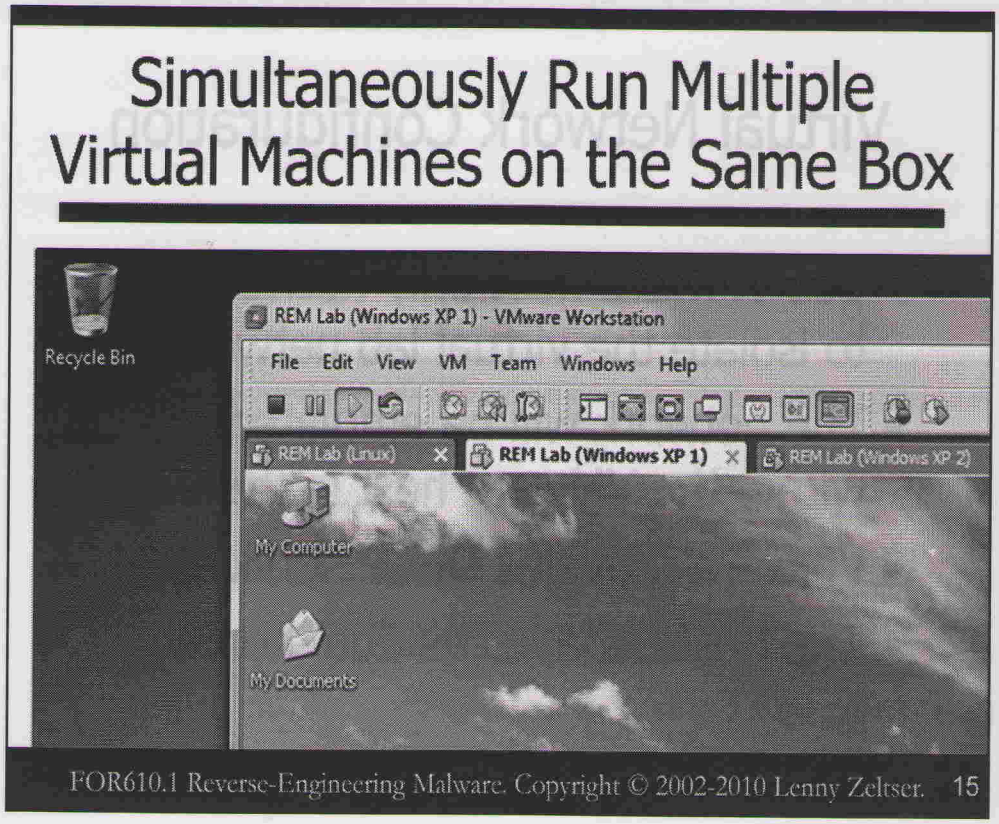
The reverse-engineering approach that works best for me, and that we will use in this class, consists of two phases: behavioral analysis and code analysis.



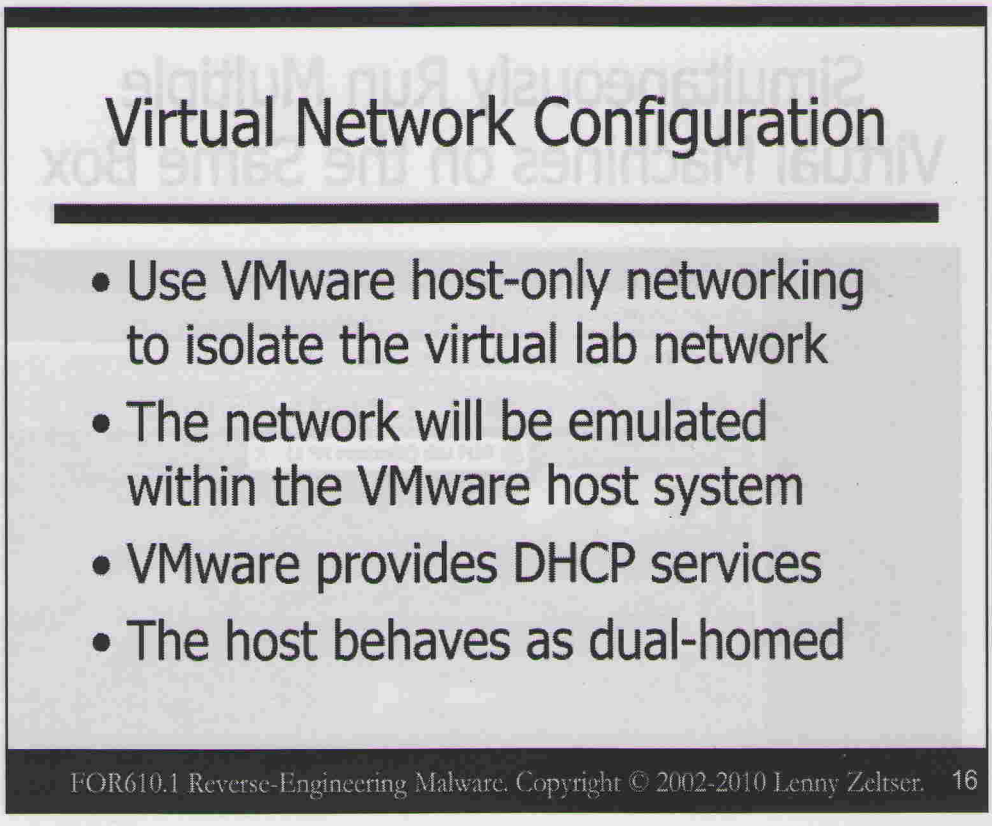
This slide illustrates the network infrastructure that you can use in the malware analysis lab. Note that there are several systems on this isolated network, to accommodate the need for multiple operating systems that may arise during the analysis. In this course you will encounter several services, such as HTTP and IRC, which I like to have available in a laboratory like this.



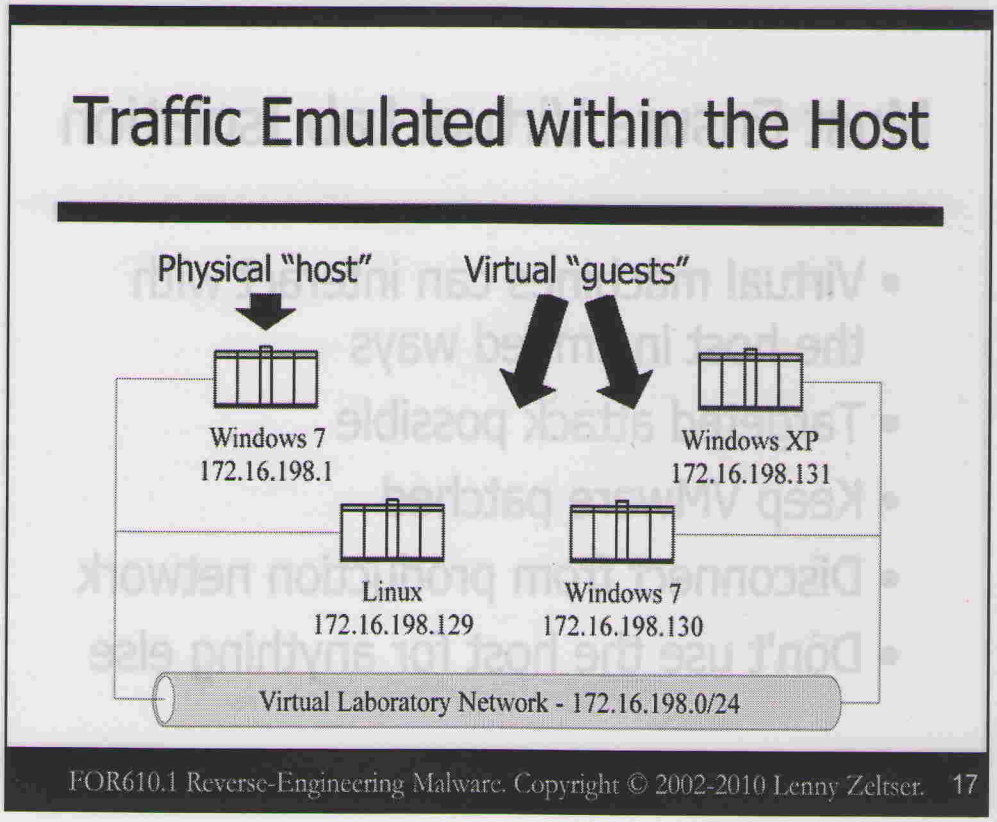
With VMware, multiple operating systems can run simultaneously, and each virtual machine "is equivalent to a PC, since it has a complete, unmodified operating system, a unique network address, and a full complement of hardware devices." VMware can be installed on Windows and Linux systems. (See http://www.vmware.com/products/desktop/ws_faqs.html for more information about capabilities of VMware.)



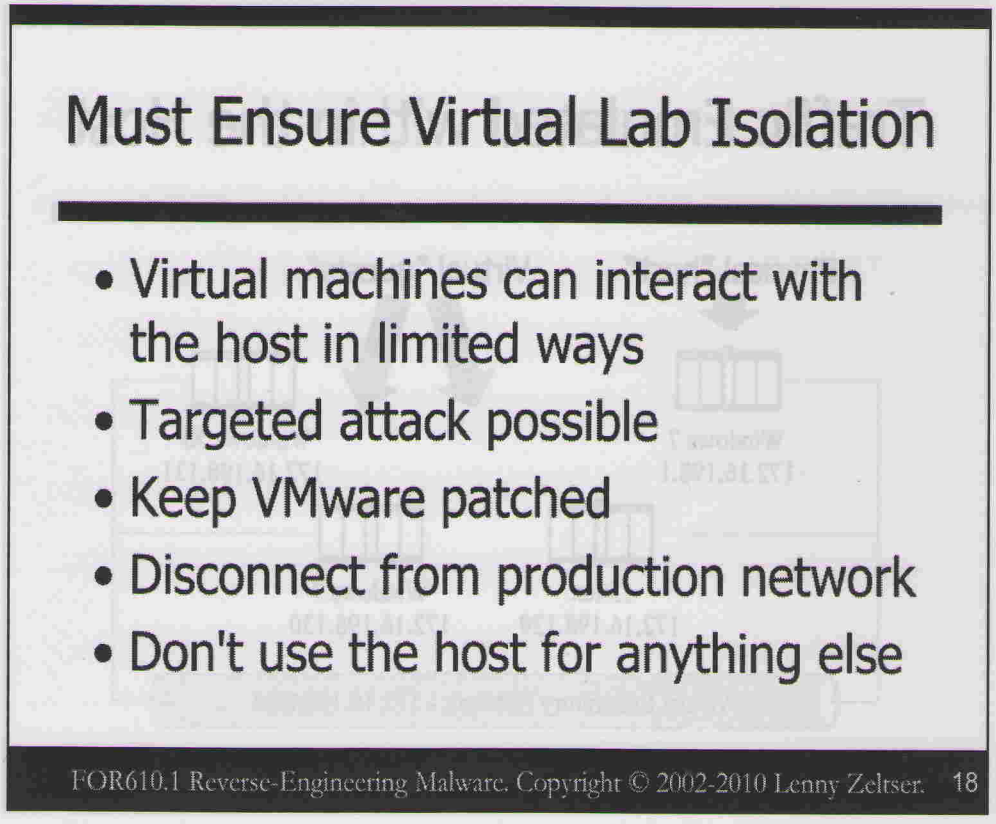
The screenshot on this slide illustrates a typical setup running several virtual machines simultaneously within the host operating system. In this example, VMware is running on the physical Windows 7 host and hosts several guest operating systems, including Linux, Windows XP and Windows 7. VMware encapsulates each virtual machine in a tab; on the screenshot, only the Windows XP tab is visible.



VMware provides a convenient way of isolating the laboratory network through the use of the host-only network, which VMware emulates within the physical host system.



This slide illustrates the laboratory network setup I introduced a few slides ago, but now it is implemented within VMware as a host-only virtual network.



The ability to exchange clipboard data is provided by optional VMware Tools drivers that can be installed on virtual machines. VMware tools also allow the user to move the mouse pointer between guest operating systems and the hosting machine. The problem with such interaction is that it is enabled from within virtual machines, and cannot be disabled from the hosting system. This means that it is possible to craft a targeted attack against a user of a VMware-based laboratory that will achieve some level of access to the hosting system from within a virtual machine.

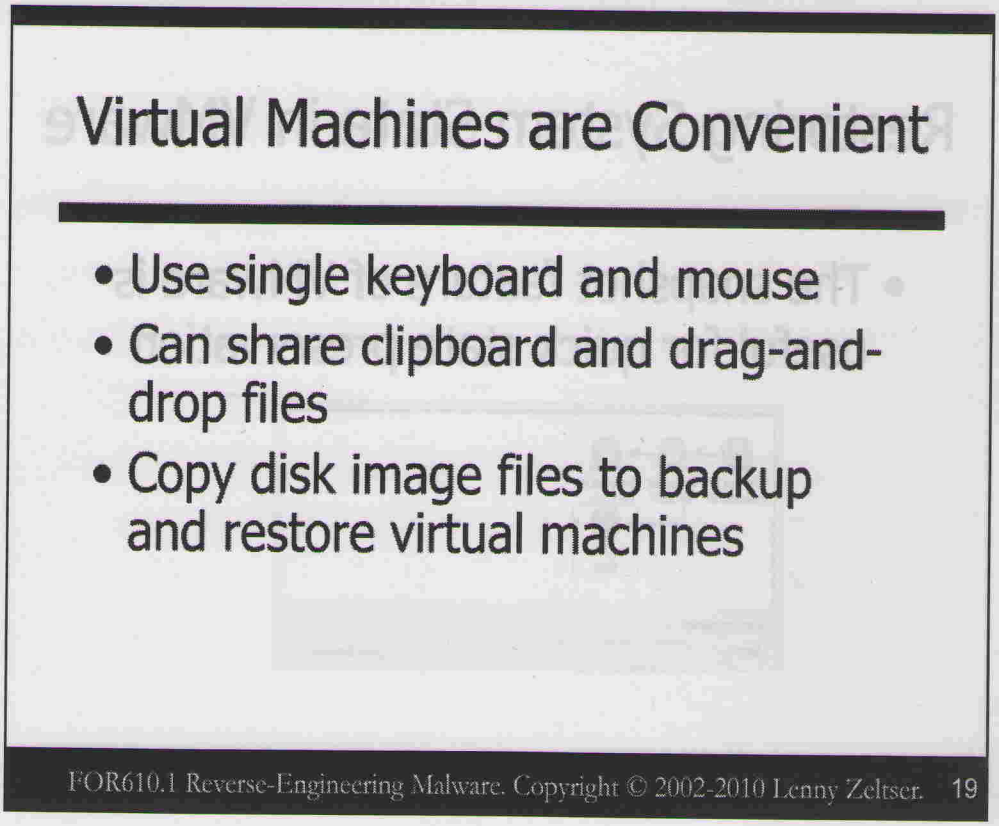
Also, be sure to stay up to date on VMware patches. For instance, a vulnerability announced in December 2005 could allow an attacker to remotely execute arbitrary code on the system hosting VMware. This was possible due to a bug in the NAT processing code in VMware. For more information about this vulnerability, please see:

http://kb.vmware.com/vmtnkb/search.do?cmd=displayKC&docType=kc&externalId=2000&sliceId=SAL_Public

http://www.securityfocus.com/bid/15998

Additionally, the virtual network environment needs to be set up in a way that will prevent traffic from an infected laboratory machine from escaping into the real world through the hosting system. You should ensure that packet routing is disabled on the hosting machine, and install personal firewall software on the system as a second layer of protection. (For instance, ZoneAlarm is free for personal use and is available at http://www.zonelabs.com.) This precaution, however, cannot guarantee complete isolation for the environment.

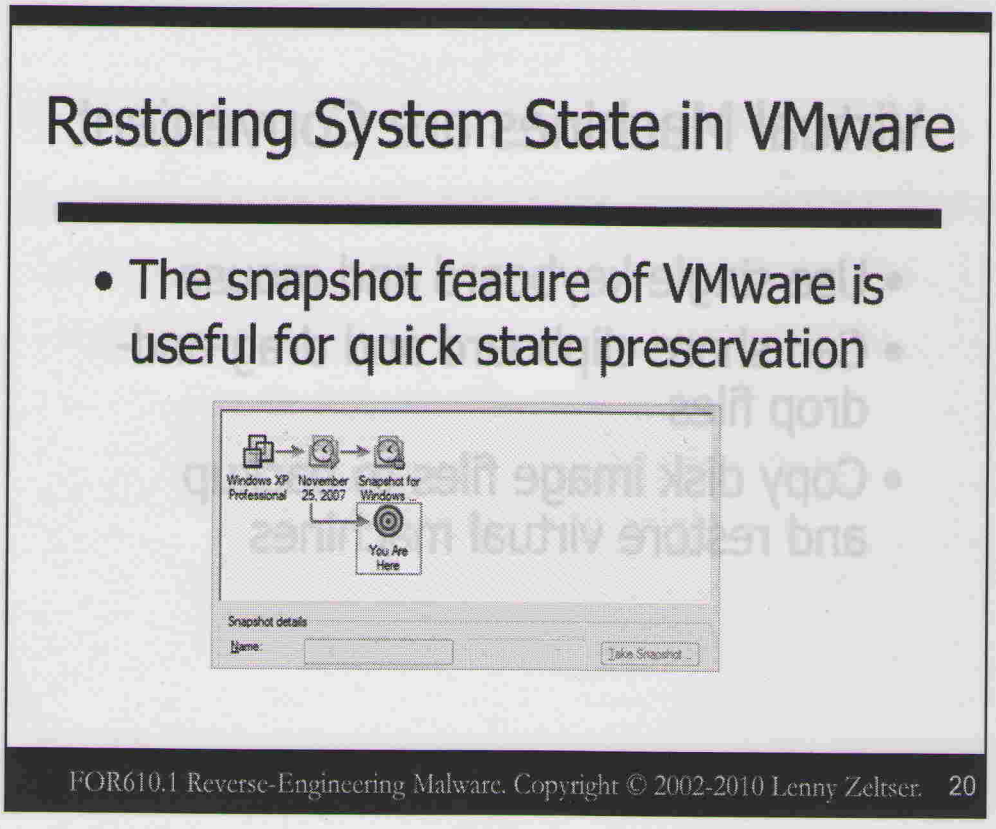
Ideally, the VMware host should not be connected to a production network, and should be considered as dispensable as the virtual machines themselves.



Running multiple virtual machines simultaneously on the same computer allows us to use a single keyboard and mouse to control multiple systems. If the workstation is a laptop, this makes the laboratory environment highly mobile.

VMware also allows virtual machines to share the hosting system's clipboard. I find this to be a convenient way to transfer text and small images between systems, although this capability could potentially threaten the quarantine of the isolated laboratory environment.

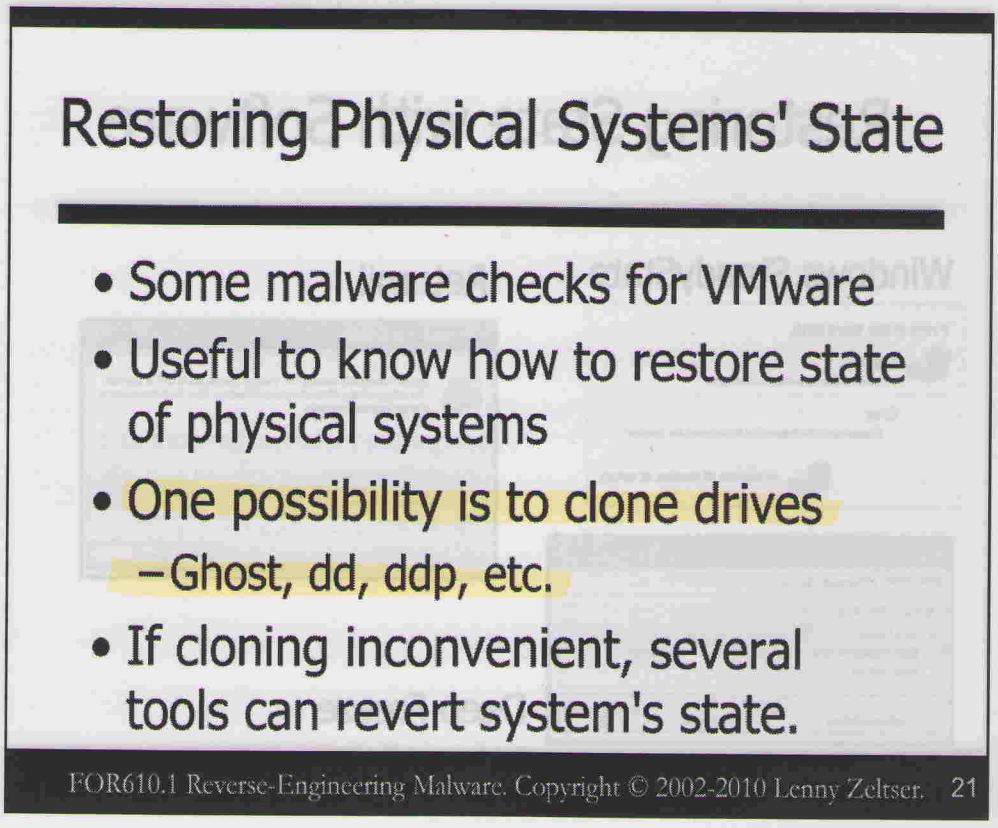
Another advantage of using VMware instead of physically separate systems is the ability to backup and restore full systems in a matter of minutes. Each virtual machine is implemented using several self-contained files by default located in the program's "VMs" subdirectory. Backing up the system can be accomplished by making a copy of the files that are used by VMware to represent the virtual machine. This is particularly useful when analyzing unknown malware, where unless the system is brought to a known state, repeated interactions with the virus or trojan might taint the environment. Additionally, the ease of making copies of virtual machines allows engineers to maintain a number of instances of an operating system with different patch levels.



Another way to preserve the state of a virtual machine is to use the snapshot feature of VMware. Most of the snapshot's state is encapsulated in "redo" log files that VMware can automatically use to bring the virtual machine to the desired state. Using snapshots is usually faster and more space efficient than making full copies of the machine.

VMware Workstation is very flexible in the way it lets you save snapshots of the system's state. It supports multiple snapshots, which is very useful for "bookmarking" different states of your system during the analysis. It even lets you clone snapshots, so that they function as separate virtual machines that are built upon a common system base.

This flexibility in snapshot-taking is what allows many analysts to justify paying for VMware Workstation, even though the VMware Server version of the product is free. VMware Server, as of this writing, only supports a single snapshot.



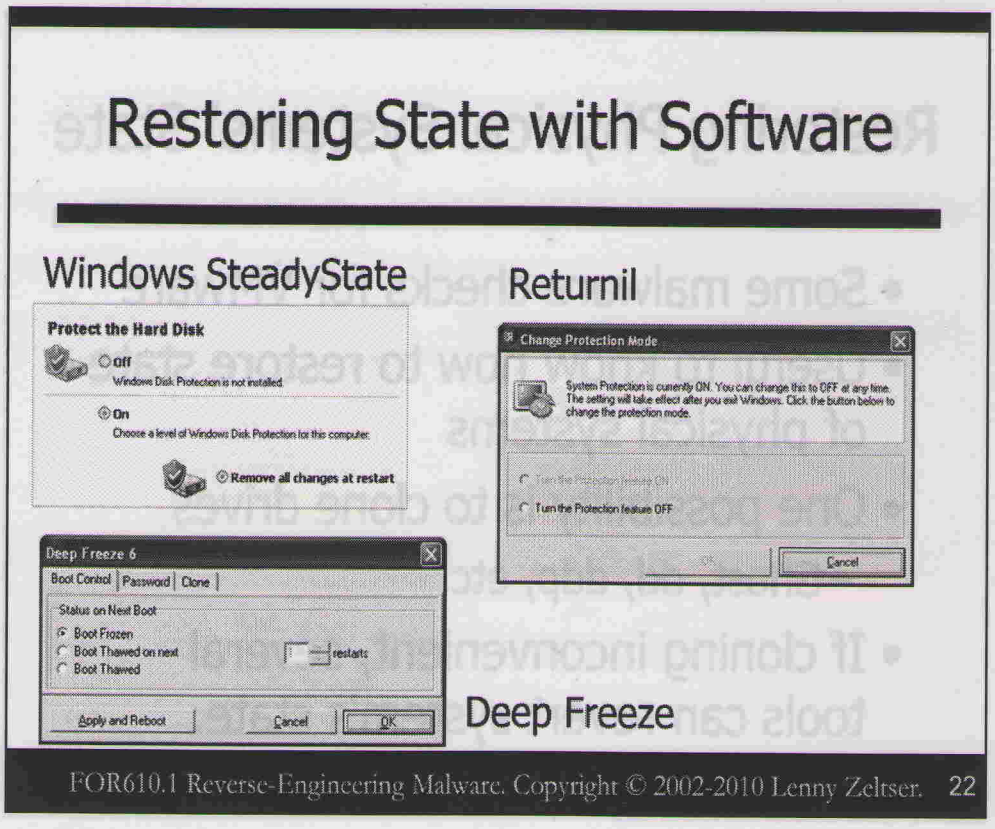
Malware authors may check whether their programs are running within a virtual machine. We will discuss techniques for concealing the use of virtualization later in the course. However, sometimes it is easier to move away from a virtual to a physical system, rather than dealing the virtualization-checking capabilities of malware.

Disk cloning software, such as Ghost or dd allows the analyst to save the laboratory system's hard disk image, and then reapply it after completing the analysis. (dd is available for free for pretty much all Unix-flavored operating systems.) Cloning large disks via this method may be time-consuming. However, while not as convenient as clicking a button to revert the system's state, it is a time-tested and reliable method.

Internet Storm Center published the following tip from its reader Tyler Hudak describing a free tool called ddp (dd-delta-patch), which he used to create a patch from an existing dd image and then re-apply it when he wanted to restore a specific configuration. (See http://isc.sans.org/diary.html?storyid=4147.) The ddp tool is available as a free download from:

http://www.korelogic.com/tools.html

If disk cloning is not convenient or fast enough for you, several tools are available for quickly rolling back the system to a pristine state.



Once installed on the physical system, Deep Freeze lets you "freeze" the system's configuration in its pristine state, automatically reverting to that configuration when necessary after a reboot. You can purchase it from around $14 and up at:

http://www.faronics.com/

Another product in this category is Returnil. It is marketed as a tool for combating malware infections by resetting the system to a trusted state. By enabling its System Protection feature, you can make use of this functionality for rolling back system-level changes in your lab. Free and commercial versions of Returnil are available at:

http://www.returnilvirtualsystem.com/

Windows SteadyState is a free product from Microsoft, and is available for Windows XP. Like Deep Freeze, SteadyState is positioned to help lock-down public systems, such as Internet kiosks and library computers. It has the ability to restore the system to a known state via its Disk Protection feature. Download it for free at:

http://www.microsoft.com/windows/products/winfamily/sharedaccess/default.mspx

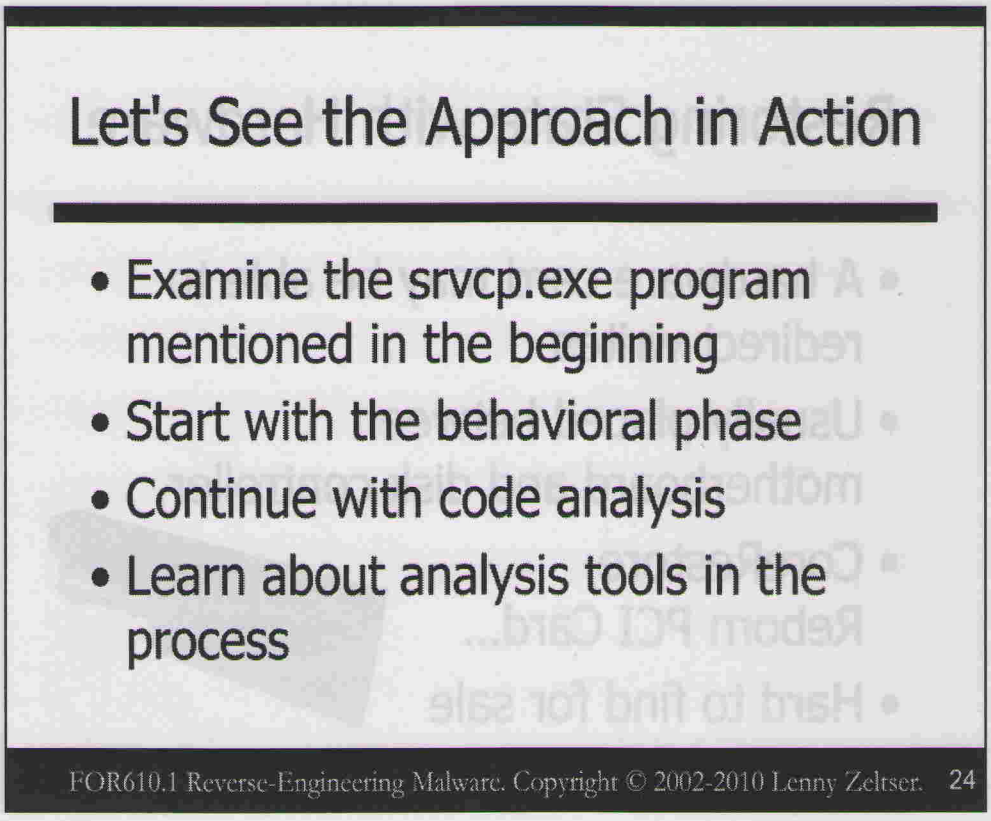
Note that the isolation provided by these products may be bypassed, perhaps by terminating or modifying their processes running on the infecting system, or by targeting the system's Master Boot Record (MBR).

Another way of capturing and restoring system state is to use dedicated hardware modules, which are typically installed between the system's motherboard and the disk drive IDE controller. These products work well; unfortunately, they are becoming hard to find in stores; buying them used on sites such as eBay might be the best option at the moment.

For instance, the CoreRestore card is designed to redirect system changes to a "temporary working area," allowing the administrator to revert to a pristine state via a reboot. Unfortunately, at the time of this writing, CoreRestore has been taken off the market. Its vendor, Infrastructure Development Corp., stated that they are working on a new version of the card, which they plan to begin selling in the future. When it was available, each CoreRestore card cost around US$150, and could be purchased from http://www.corerestore.com.

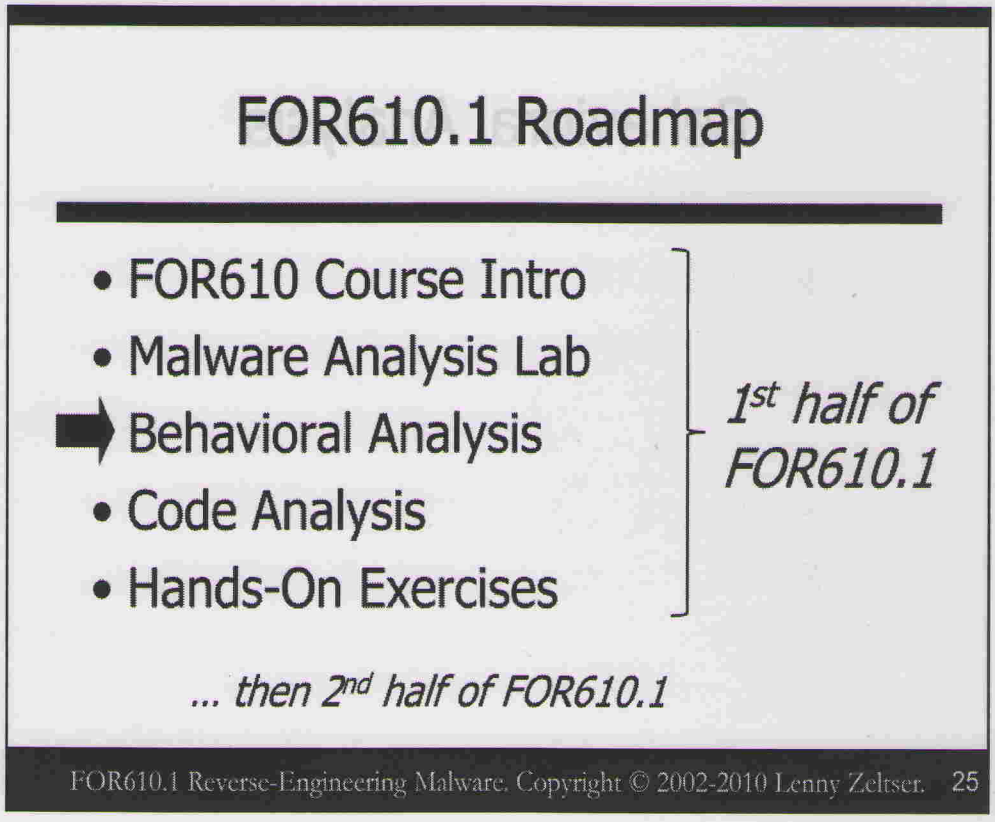
Another option, similar to CoreRestore is the Reborn PCI Card by Lenten Technology Co. (http://www.lenten.com/Products.asp) Unfortunately, just like CoreRestore, the commercial availability of Reborn PCI Card is unclear.

Another similar product was Forensic Phantom by Logicube. Unfortunately, it has been officially discontinued by the company in favor of a forensic disk-duplicating product—not quite what we are looking for.

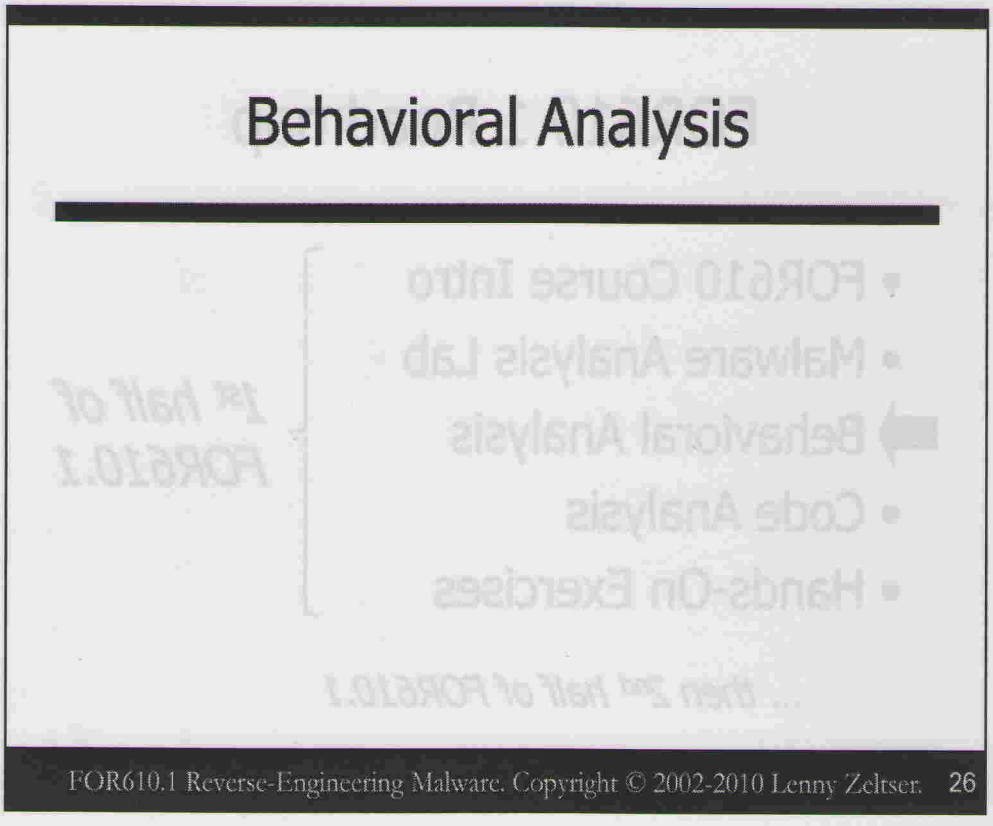


Now that you know how the analysis laboratory network should be set up, let's see this approach in action. As I go over my analysis of the srvcp.exe trojan, I will introduce tools that you will find useful for performing a similar analysis.

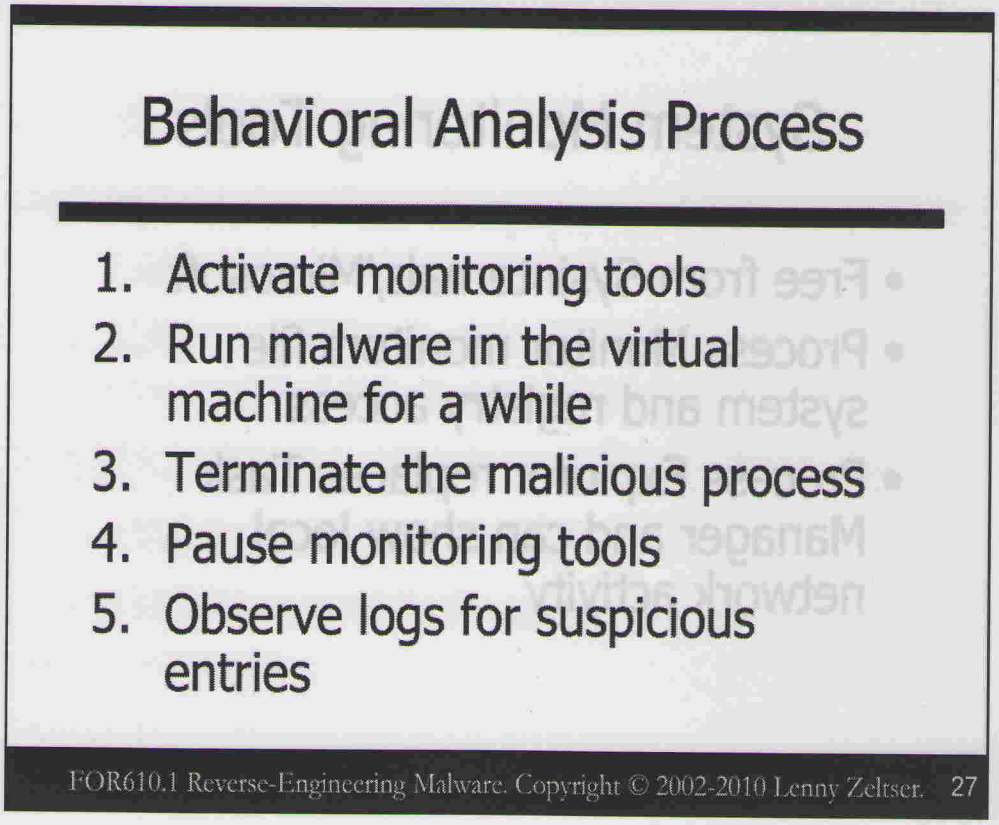
If at some point you want to go through this analysis in your own lab, you can locate a copy of the specimen on your course DVD in \Malware\day1\srvcp.zip.



Now that we discussed how to set up a malware analysis lab, let's look at the behavioral analysis process.

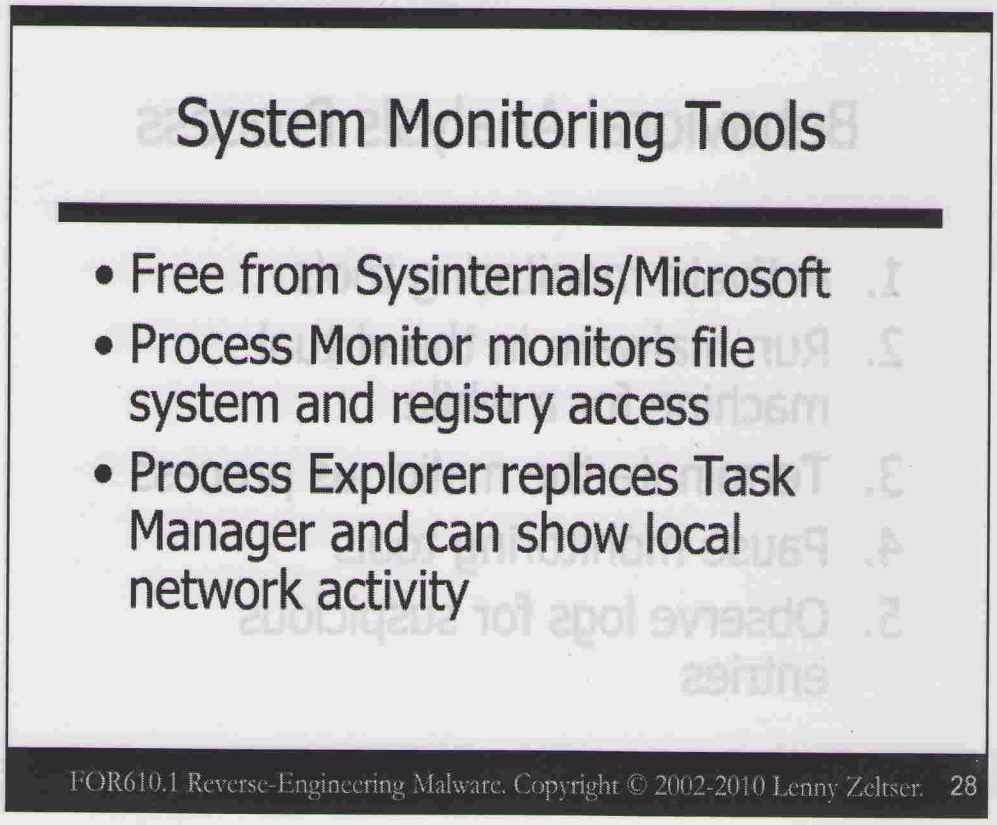


We'll start off with the behavioral analysis phase.



This slide describes core steps in the behavioral analysis steps. As part of this process we will actually infect a laboratory system as we monitor the specimen's interactions with the local system and with the network.

To begin the analysis of srvcp.exe, we will launch the appropriate monitoring tools and observe.

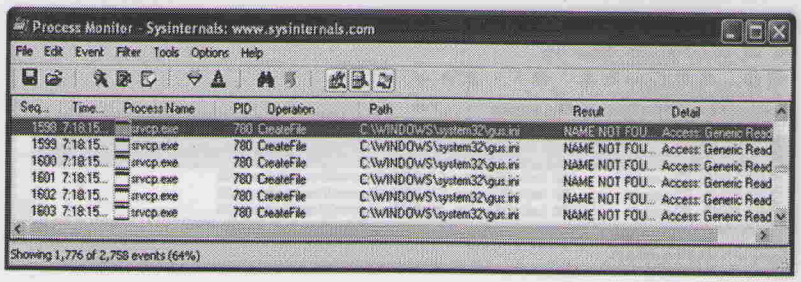


Sysinternals makes a number of great tools for monitoring local interactions between the malicious executable and the system that it is infecting. These, along with most of the tools mentioned in this course, are available on the DVD you received for this course. Sysinternals continually updates its tools, so it's a good idea to check back frequently for updates. In the fall of 2006 Microsoft purchased Sysinternals, and committed to continuing to make its tools available as a free download. As of this writing, these tools are available on the Sysinternals former website (www.sysinternals.com) and the new Microsoft site (www.microsoft.com/technet/sysinternals).

We need to launch the tools mentioned on this slide before infecting the system with the malicious program. For the purposes of the example covered in this section, I launched these tools on a VMware virtual machine and then executed srvcp.exe on that system.

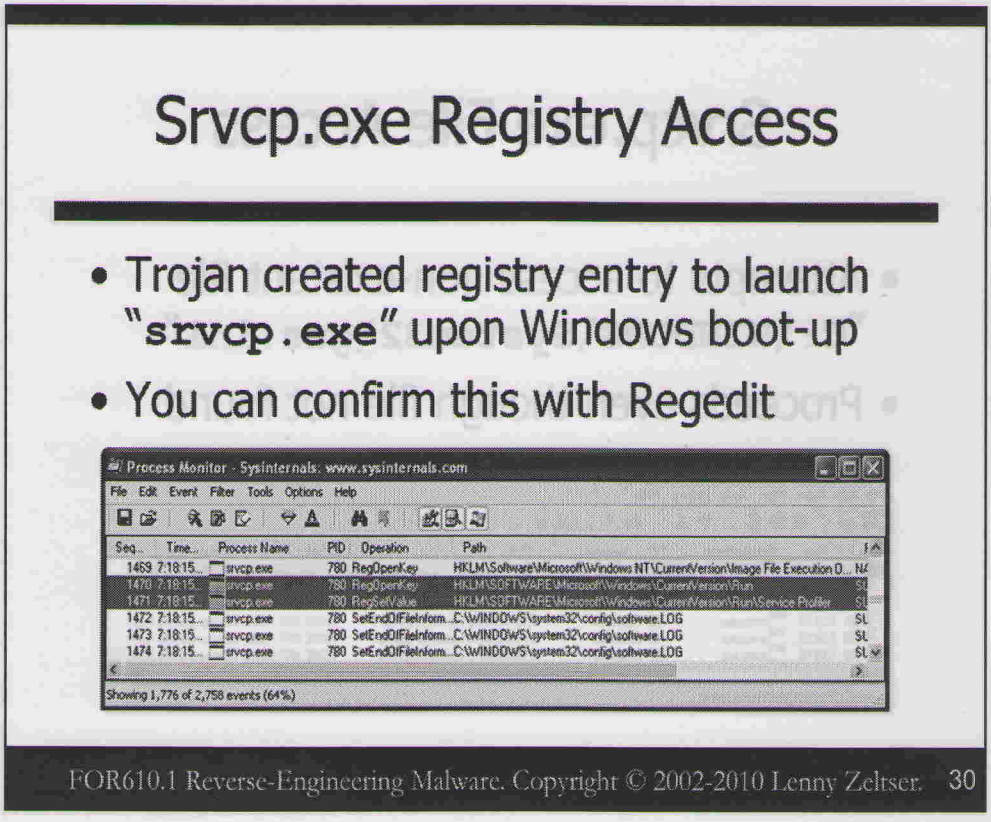
Srvcp.exe File Access

- Attempts to access non-existent file "C:\WINDOWS\System32\gus.ini"
- Proceeds even though file not found



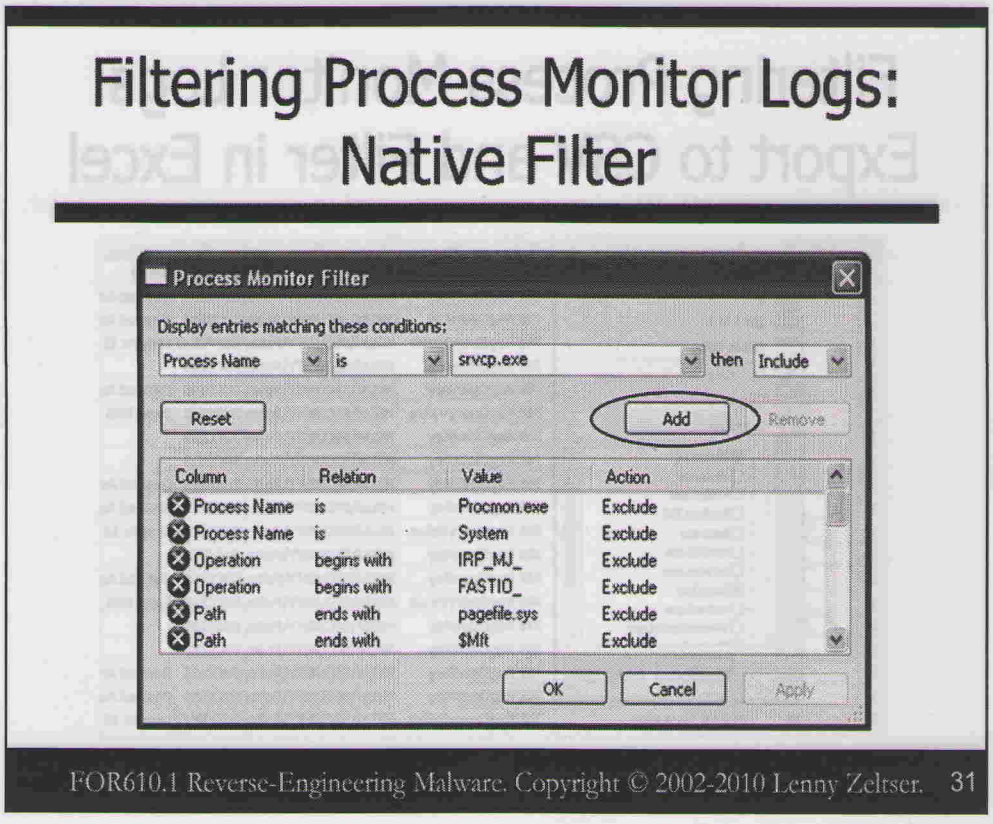
Process Monitor is a free Sysinternals tool that allows us to monitor local file system and registry access. This is a relatively recent tool, and combines functionality that was formerly available in separate tools—Filemon and Regmon. Process Monitor also allows us to monitor local threads and DLLs. The tool also provides convenient data filtering capabilities.

Process Monitor, running while srvcp.exe was infecting the system, registered a number of file access attempts typical for a regular Windows executable. An attempt to access an unfamiliar file gus.ini stood out. The malicious executable attempted to read that file, and, having failed to do so because the file was not found, continued to operate nonetheless.



Process Monitor also recorded a number of registry access attempts typical for a normal Windows executable. A key creation under HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run stood out. This is the mechanism the malicious executable uses to ensure that it runs every time the victim reboots and logs in to the infected system. The subkey was given an innocuous name: "Service Profiler". The value saved into it was "srvcp.exe".

What can we infer regarding the trojan's distribution mechanism based on how it starts up? Note that the full path to the trojan's executable was not specified in the registry entry, which indicates that the author assumed that srvcp.exe would be in the path. This suggests that we do not possess the trojan's distribution mechanism, which would have either modified the system's path, or copied srvcp.exe into a directory that was in the path by default, for instance C:\Windows. In my case, the executable file remained in the temporary directory that I created earlier for the tests; since no new files were created, the trojan would not start upon system boot-up.

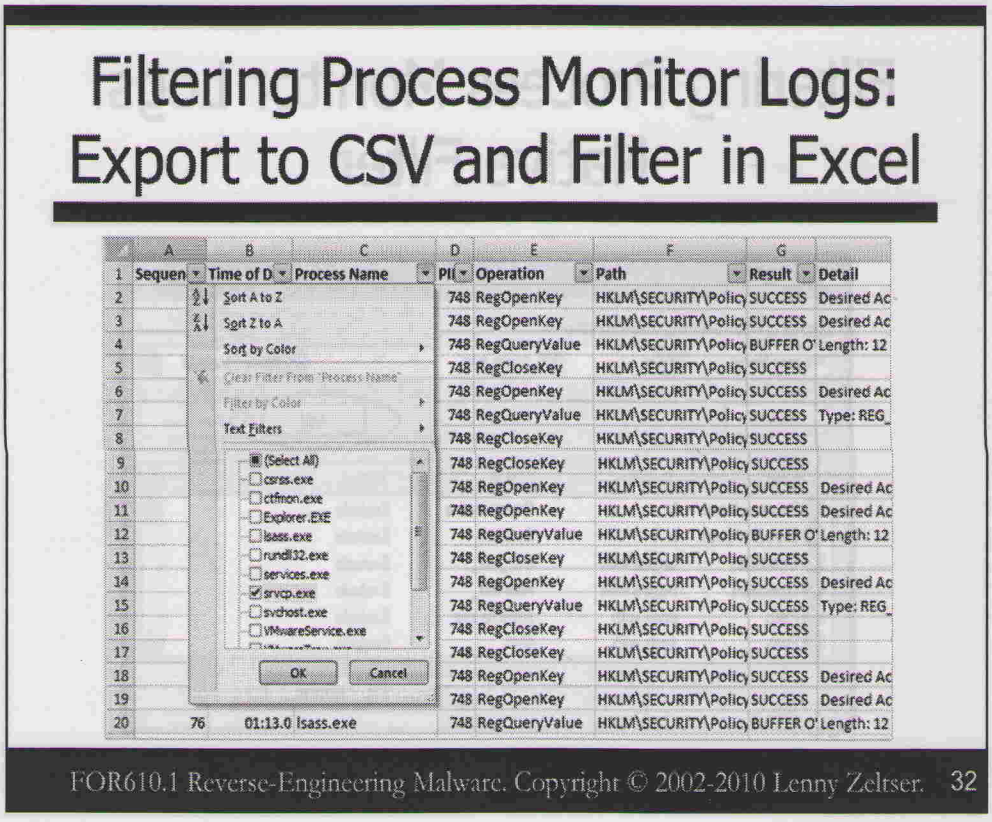


As you can imagine, the logs that Process Explorer generates can be very long. It's easy to miss something important by simply scrolling through them page by page. That's why you may want to filter the results of this tool, to temporarily limit what types of events it displays.

Note that I prefer to capture all information, and filter the displayed results, rather than limiting what the tool captures. This is because you never know what you may want to see until you begin looking through the logs. I'd rather capture more than I need, than not enough.

To filter capture results within Process Monitor, go to its Filter menu and select "Filter…" You will see the window shown on this slide. You can define your filtering parameters on the top, and then click the "Add" button to add them to the filter set. If you mess up and want to return to the default filter set, just click the "Reset" button.

In this example, I defined a filter condition that told Process Monitor to only include events where the Process Name is "srvcp.exe". I then clicked the Add button and the OK button.

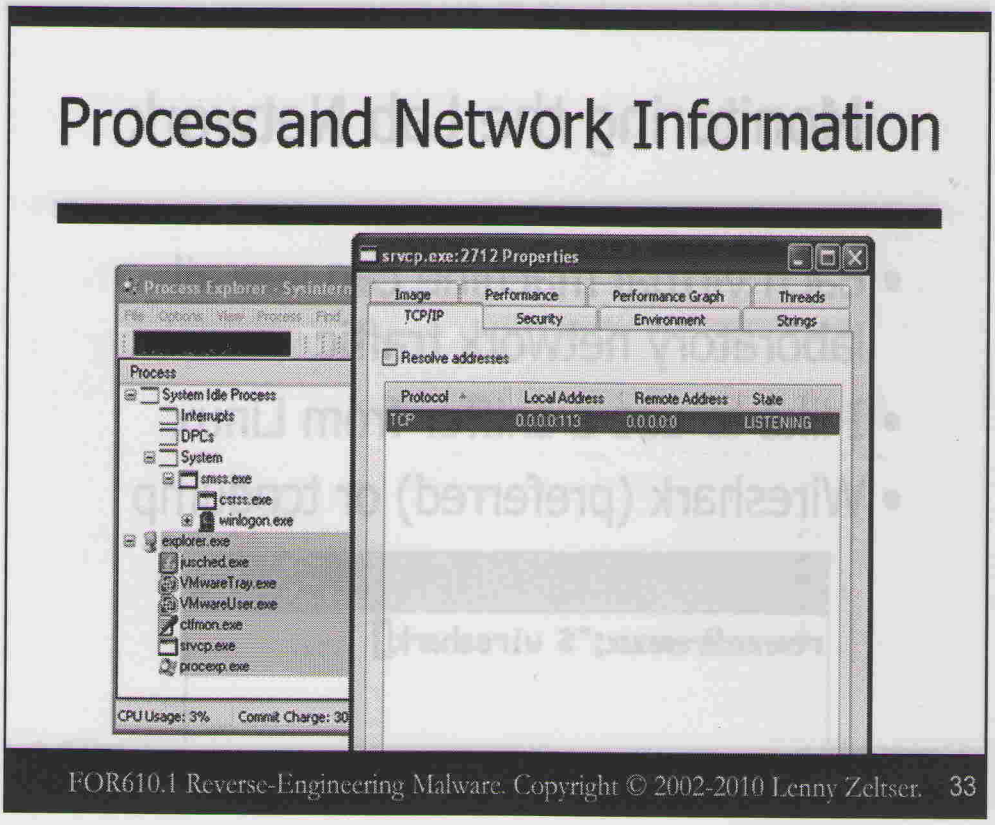


While the filtering capabilities built into Process Monitor are convenient, I prefer the filtering features of Microsoft Excel when analyzing large Process Monitor log files. If you don't have Microsoft Excel, you can use another similar spreadsheet program that can import comma-separated values (CSV) files.

Start by going to the File menu in Process Monitor and clicking "Save…" On the window that pops-up, select the CSV format, specify the file name, and click "OK".

Import the CSV file into Excel, format it as you like, then activate its filter. In Excel 2005 you can do this by going to the Data tab and clicking the "Filter" button. I believe in older versions of Excel this functionality was called auto-filter.

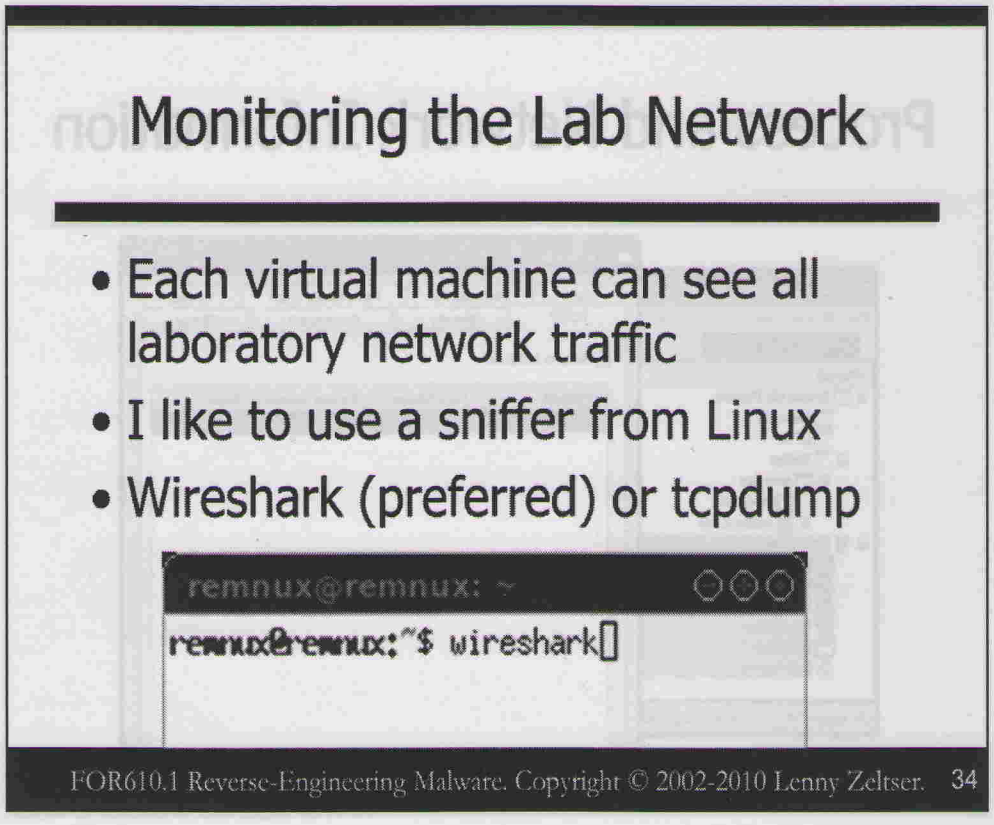
Once Excel's filter is enabled, click on the down arrow by the desired column and define your condition. In this example, I clicked on the down arrow at the "Process Name" column. Now I can select which process names I want to see. (The look of your spreadsheet should resemble the one shown on this slide.)



Process Explorer is a powerful Task Manager replacement. It shows the currently-running processes using a tree-like listing, so you can see process relationships: which process spawned which process. Moreover, you can right-click on any process, select properties, and observe a wealth of information about the process. For example, Process Explore is able to show us that srvcp.exe is listening on TCP port 113.

TCP port 113 is usually associated with the Ident mechanism, which is typically used by Unix systems to obtain the name of the user who initiated a TCP connection. (To learn more about the Ident protocol, see http://www.faqs.org/rfcs/rfc1413.html.)

Many IRC servers issue Ident requests when an IRC client connects; this is done in an attempt to establish the connecting user's identity. Some servers will not even accept the connection unless the Ident phase successfully completes, which is probably the reason that srvcp.exe implemented this mechanism.



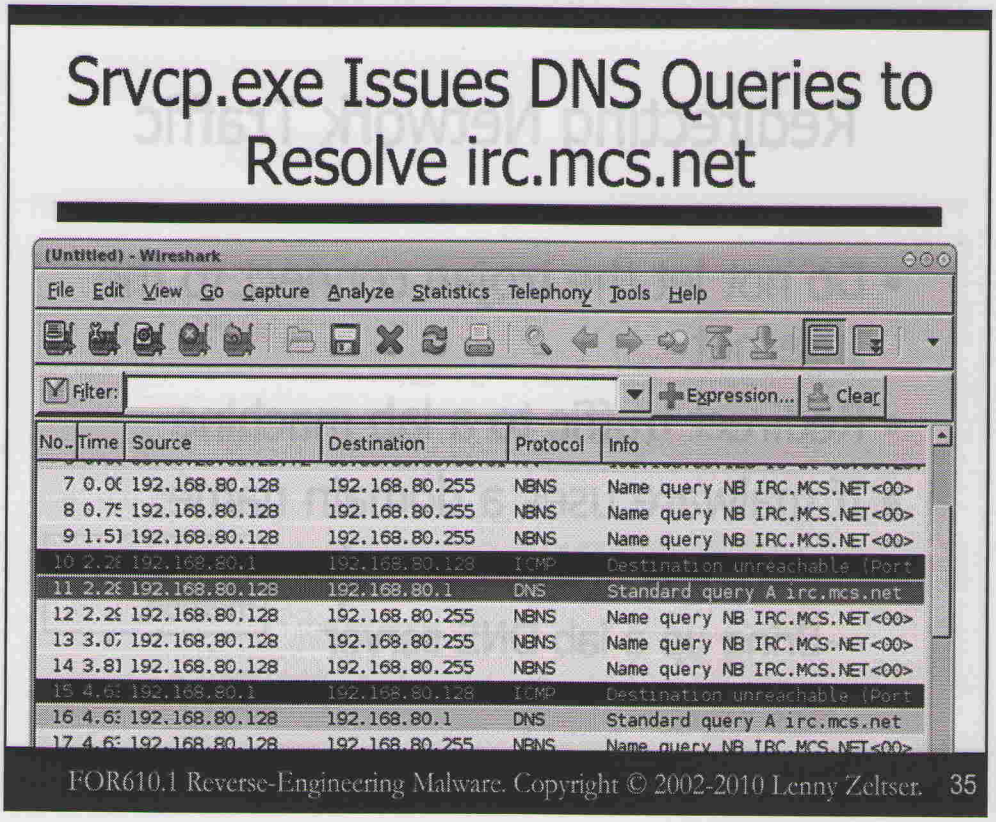
A network sniffer, running within our laboratory environment, is an essential tool for monitoring network activity associated with the malware specimen. I like running the sniffer on the Linux virtual machine in my lab. The sniffer is, by default, configured to monitor the network in a promiscuous mode, so it should be able to see all traffic in the lab.

I like using Wireshark as the sniffer, because it's powerful and free. Another common option is tcpdump, which is nice for situations where you want to run the sniffer from the console without having X Window System running in Linux.

Both Wireshark and tcpdump are available for Windows and Linux; we'll use Wireshark on the Linux virtual machine in this course. The Linux virtual machine also has tcpdump, but we won't explicitly refer to it during labs.

This course uses REMnux for performing analysis steps that are well-suited for Linux platforms. (We'll set it the Linux virtual machine a up a bit later.)

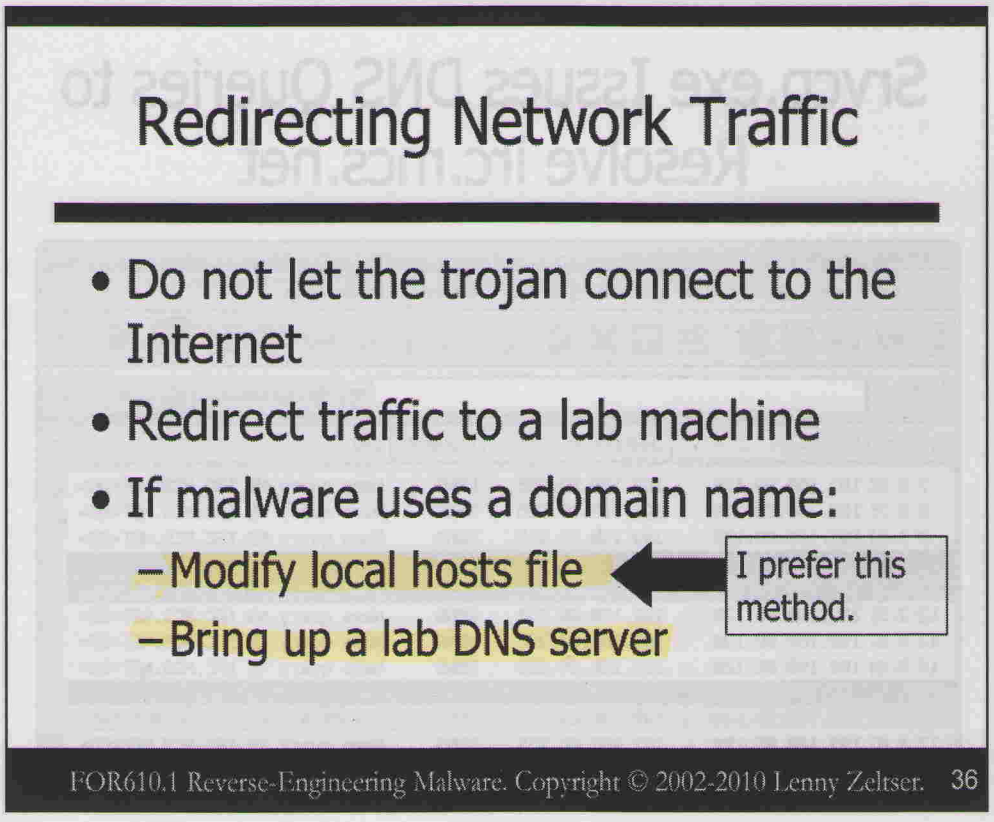
To launch Wireshark, login to REMnux, and type "**`wireshark`**" at the command prompt. In REMnux, this command is defined as an alias that actually executes "**`sudo wireshark`**". This specifies that Wireshark should be executed with root privileges; that's why you may be prompted for the root password when launching Wireshark.



As shown in the network trace on this slide, Wireshark reported that the infected system issued a DNS request in an attempt to obtain the IP address of "irc.mcs.net". This attempt to resolve the hostname is consistent with the behavior reported by Process Monitor, which registered the trojan's requests to read the local hosts file.

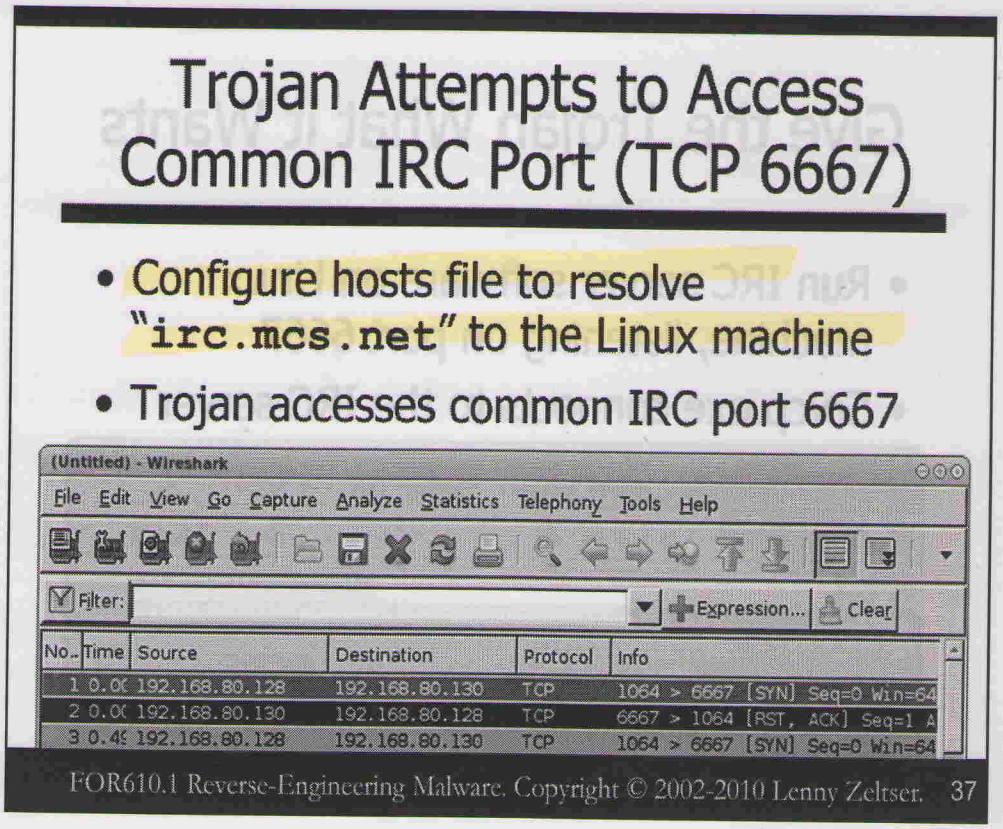
You may notice that when Windows failed to resolve the hostname using DNS, it attempted to query the hostname using the NetBIOS (NBNS) protocol. Neither attempts to resolve the hostname succeeded, because we don't have DNS in the environment.

The ICMP "Destination unreachable" packet you see in the network trace is the result of the system that is receiving DNS queries answering with an error to specify that the DNS service is not available there.



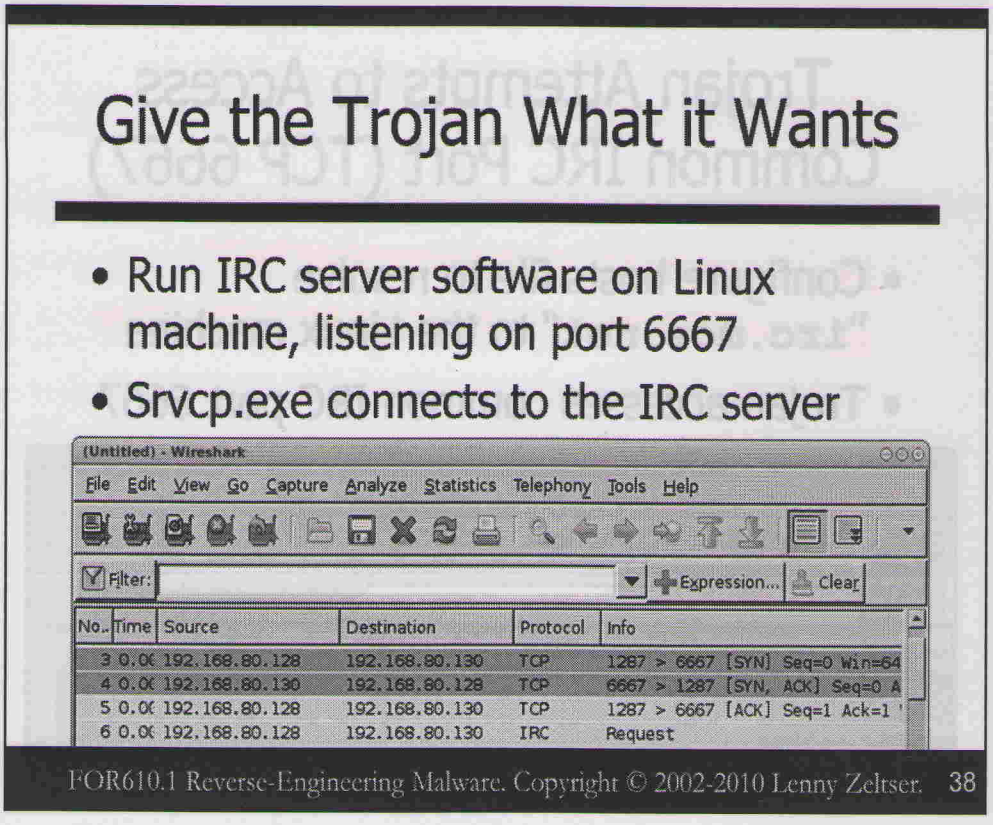
The most likely explanation for the domain name resolution requests is that the trojan wanted to connect to irc.mcs.net. One of the ways to see what the program would do once it reached this system would be to connect the infected machine to the Internet and observe. That, of course, would violate the principle of analyzing the trojan in a controlled and isolated laboratory environment.

Redirecting the trojan to connect to one of our systems, instead of the real irc.mcs.net machine, was relatively simple in this case, since the trojan used a host name, and did not pay attention to the destination's actual IP address. All I had to do was to add an entry to the hosts file on the infected machine that resolved irc.mcs.net to the IP address of our virtual laboratory machine.



Next, configure the infected machine to resolve irc.mcs.net to the IP address of the virtual Linux system. Once the trojan is able to resolve the hostname, it attempts to connect to the system's TCP port 6667, as demonstrated in the network trace on this slide. Note that the server responded with an ACK+RST packet, since it was not listening on the targeted port.

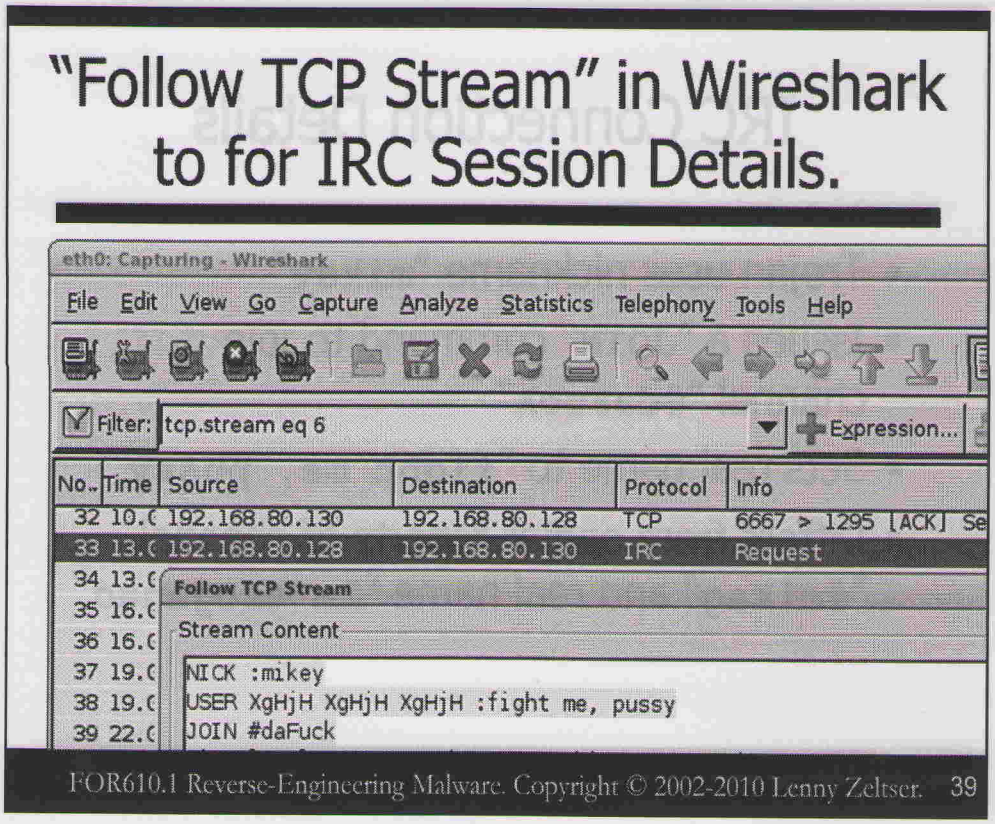
This communication attempt was not surprising, since TCP port 6667 is commonly used for IRC communications, and is consistent with the host name and nature of irc.mcs.net, which is a popular EFnet IRC server in the real world. (Also, we knew from earlier reports to the Incidents mailing list that the trojan connects to an IRC server.)



Yet again, our approach is to give the trojan what it was looking for, and see what would happen next. Therefore, install (if necessary) and activate IRC server software on the Linux machine, and configure it to listen on TCP port 6667.

The REMnux Linux distribution we'll be using in this course includes an IRC server called inspIRCd, which you can activate by typing "**`ircd start`**" at the command prompt. In REMnux, this command is defined as an alias that actually executes "**`sudo invoke-rc.d inspircd`**". This launches the IRC server daemon in the background, running the command root privileges; that's why you may be prompted for the root password when launching the server.

As you can see in the network trace captured by Wireshark, the trojan is now able to connect to TCP port 6667 on our Linux virtual machine (which it thinks is irc.mcs.net). Once the TCP handshake is completed, the trojan issues the IRC request on TCP port 6667 using the IRC protocol.

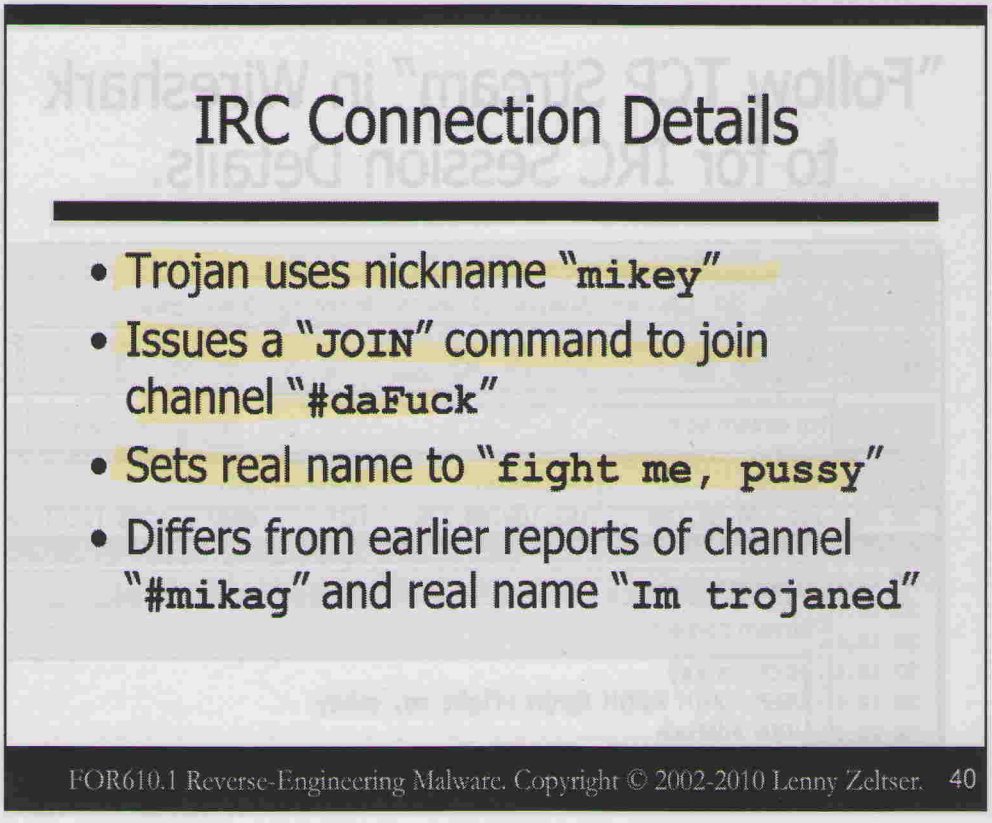


You can examine individual packets in Wireshark to observe how the trojan connects to the IRC server, what channel it joins, what nickname it uses, etc. An easier way to see these details is to right-click on one of the IRC packets in Wireshark and select "Follow TCP Stream". When you do this, Wireshark will bring up a window that shows the ASCII payload of the packets that are part of the session you're following.

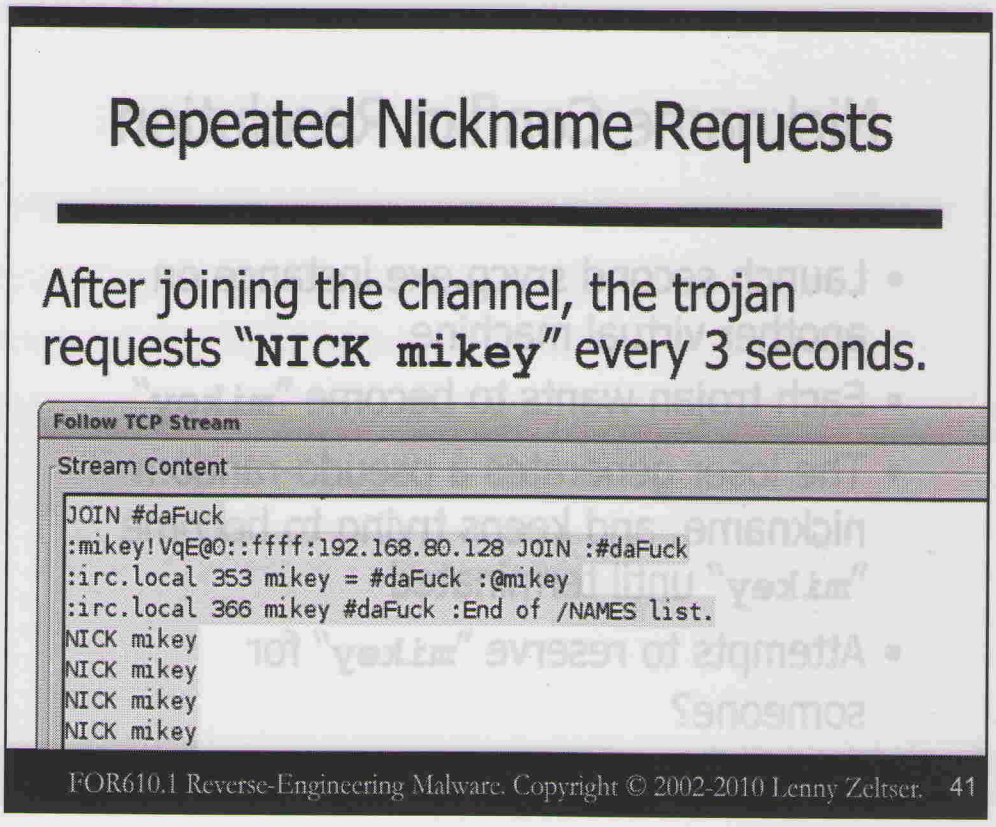
Now that the specimen was able to connect to what it thought was the irc.mcs.net IRC server, it successfully established the connection. As shown in the network trace on this slide, the trojan attempted to join the IRC channel "`#daFuck`" using the nickname "`mikey`".

This specimen uses obscene phrases for its connection attributes. I decided not to obfuscate them in this presentation because you would still recognize them, and because encountering such phrases when examining malicious code is not uncommon.

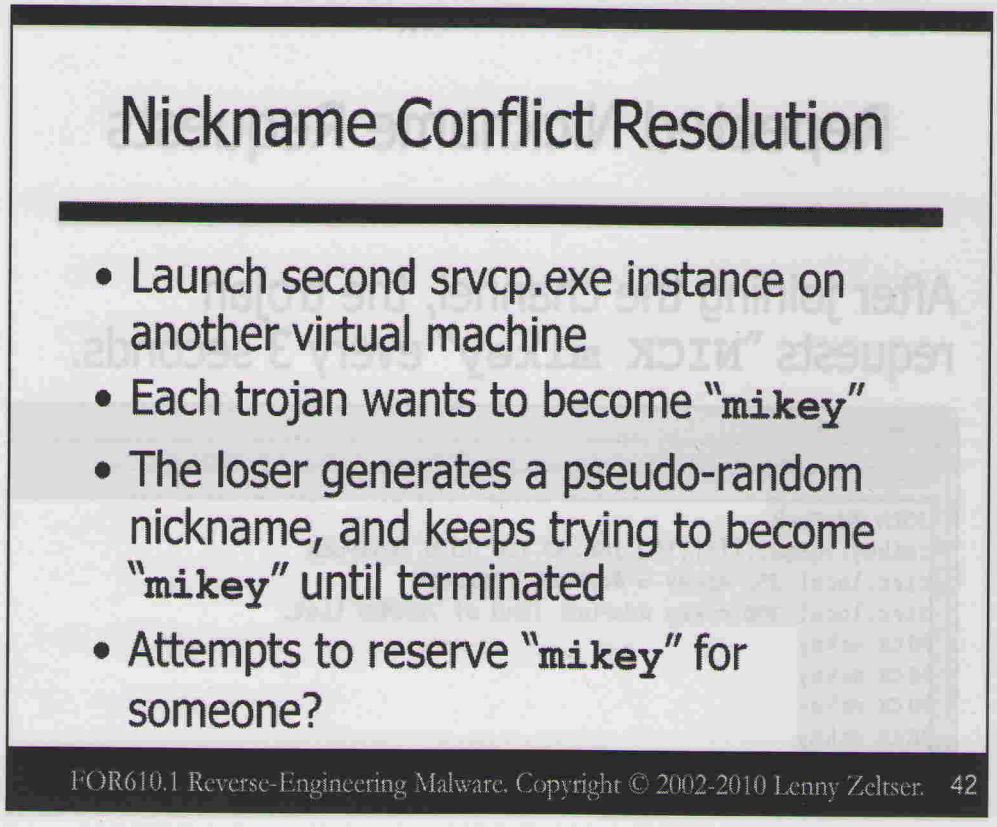
To return to the main view of captured packets in Wireshark, hit the Esc key to close the "Follow TCP Stream" window, and then click the Clear button to remove the filter that Wireshark automatically created to show the packets associated with this TCP session.



The way in which the trojan connected to the IRC server is slightly different from the reports discussed on the Incidents mailing list that I mentioned in the beginning of the class. In particular, it was reported that the trojan's real name on the channel was "`Im trojaned`", while in our case the real name was set to "`fight me, pussy`". In both instances the names were suggestive of abuse, and it is possible that we were looking at a mutated version of the program. Additionally, mailing list correspondents observed the trojan joining the channel "`#mikag`". In our case, we see a possible tie between that channel name "`#mikag`" and our nickname "`mikey`", even though our specimen joined a different channel.



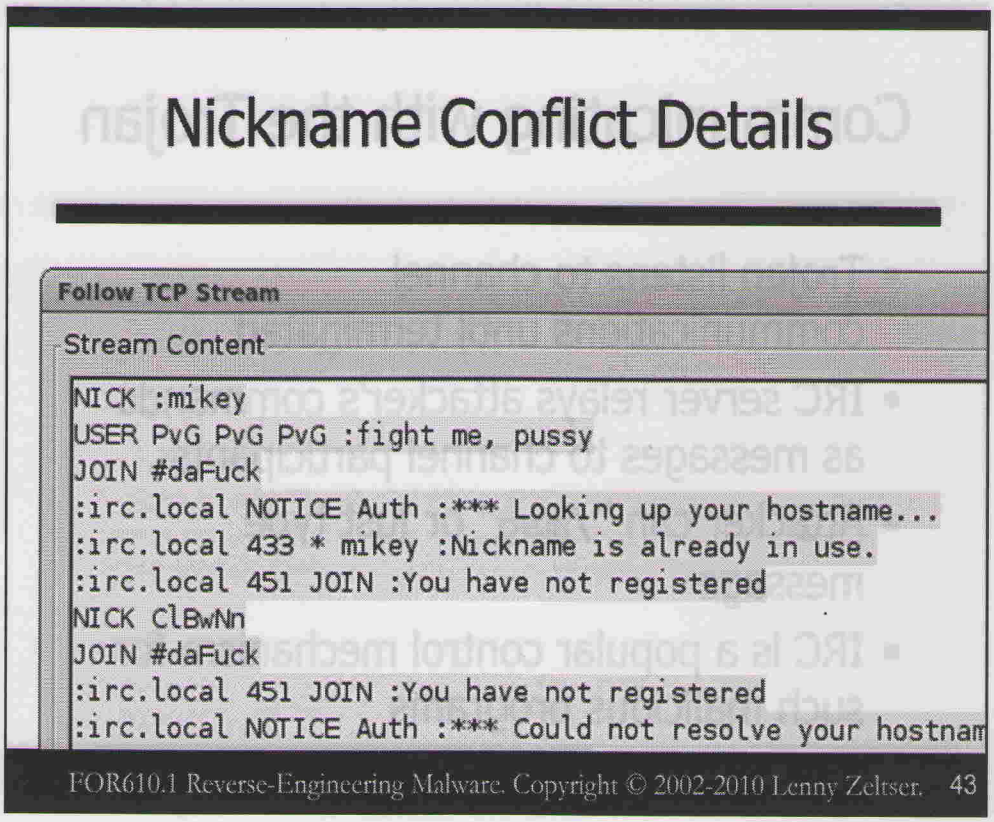
As shown in the TCP stream payload on this slide, after joining the desired channel the trojan continuously issues the "`NICK mikey`" command (this occurred approximately every three seconds.)



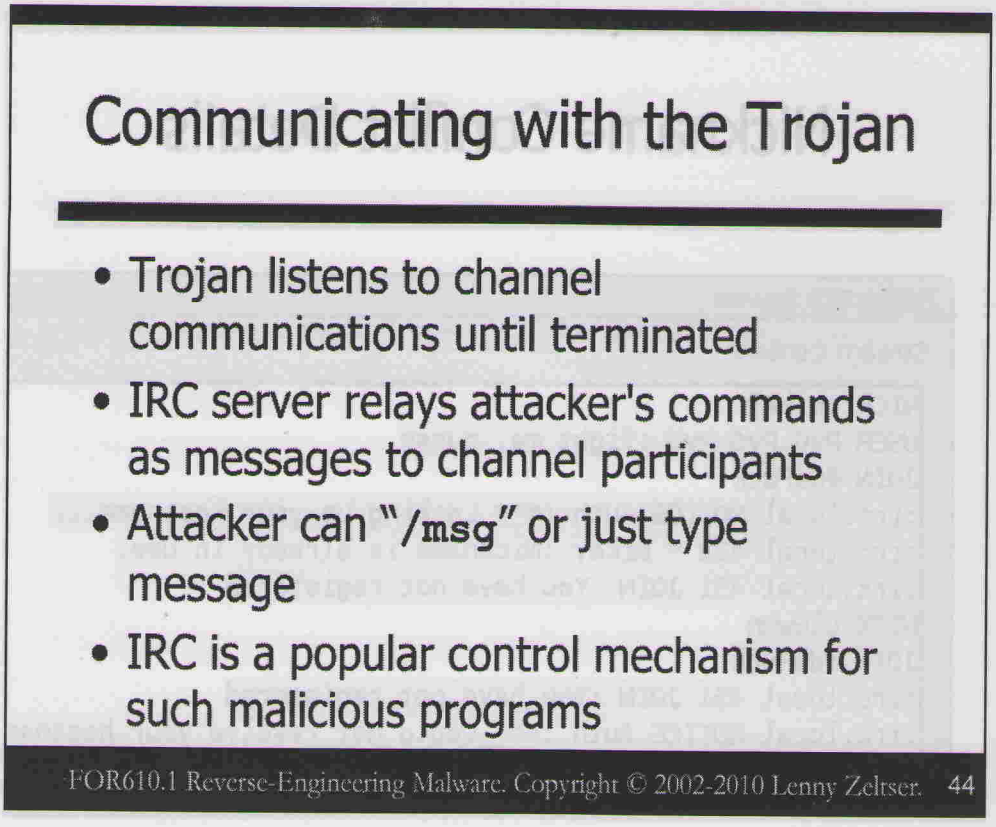
IRC requires that every user has a unique nickname across all servers forming a particular IRC network. To determine how the trojan would react if another instance of it had already taken the nickname "`mikey`", you could launch srvcp.exe on another laboratory machine, having two instances of the trojan running simultaneously.

The first instance to connect to the IRC server would acquire the nickname "`mikey`". When the second instance connected, it would not be allowed to use the same name. This would prompt the second and subsequent trojan instances to generate a different nickname for themselves in a pseudo-random fashion. All instances would then continue trying to become "`mikey`" every three seconds until they were killed.

The trojan seems to have been designed to ensure that at least one of its instances possesses the name "`mikey`". If the trojan instance that held that name were to disconnect from the IRC server, another instance of the program would pick up the name within at most three seconds. The more instances of the trojan were connected, the greater the likelihood that one of them would be called "`mikey`". Perhaps one of the purposes of this trojan is to reserve the nickname for its operator, or to prevent someone from obtaining it. It is possible that the program's author can communicate with it via IRC messages to command the trojan to release the name so that the person may obtain it.



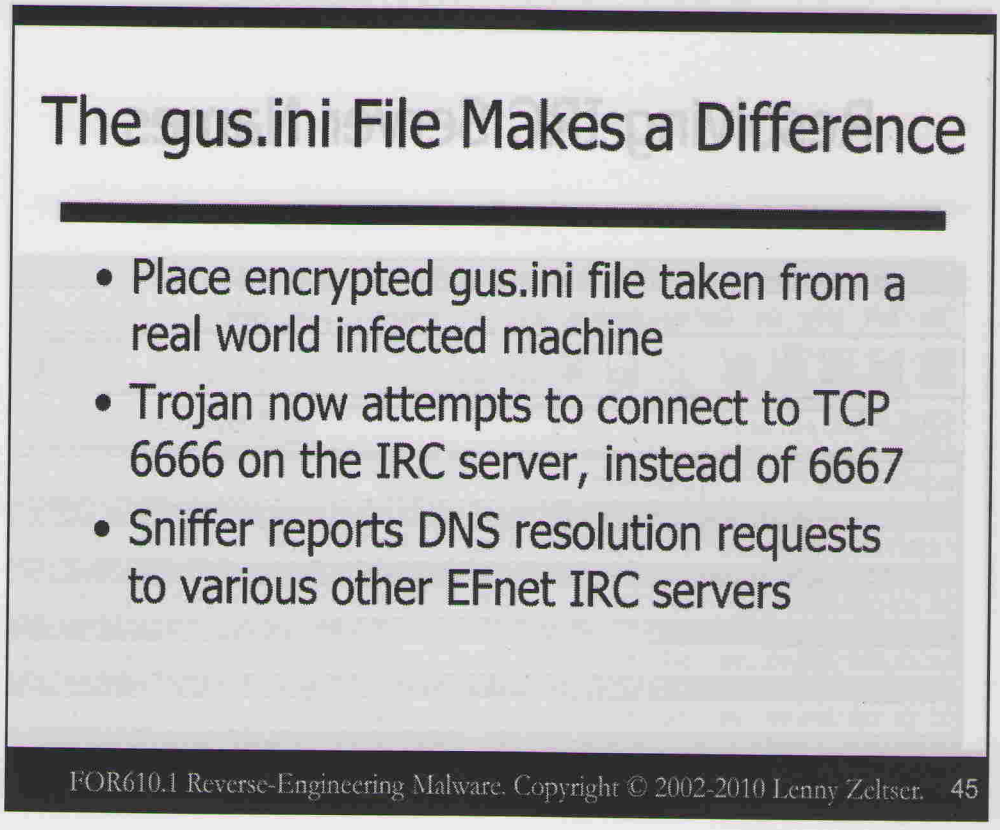
Payload of this TCP stream illustrates IRC commands that we discussed on the previous slide. You can see that the trojan attempted to use the name "`mikey`". The IRC server stated that it is already in use. The trojan then selected a seemingly random nickname "`ClBwNn`" in order to join the "`#daFuck`" channel.



Once the srvcp.exe trojan joins the IRC channel, it remained connected, presumably waiting for commands from its operator via the chat session. According to sniffer logs, the trojan also participates in periodic "PING" - "PONG" message exchanges, as defined in the IRC protocol to ensure that the IRC client is alive.

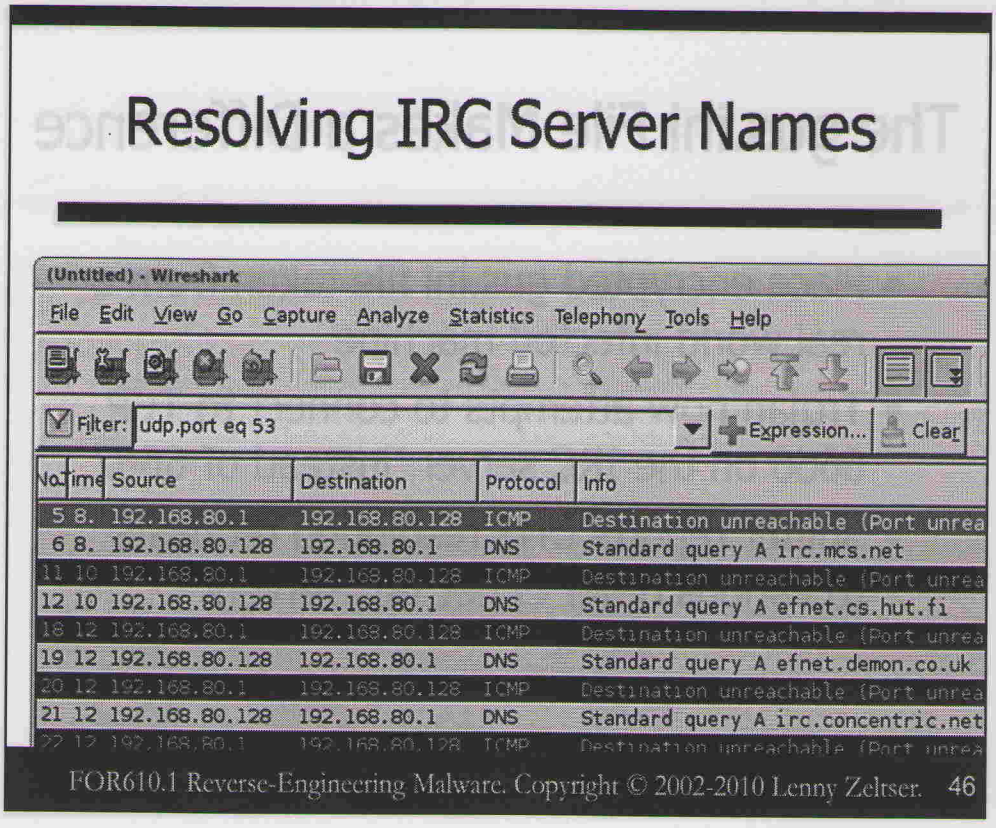
IRC offers a wonderful mechanism for centrally controlling an army of distributed agents, which is one of the reasons that trojans such as srvcp.exe are often assumed to have distributed denial of service capabilities. The attacker has the ability to communicate with multiple trojan instances by issuing a single command on the IRC channel, leaving it up to the server to relay the message to connected trojans. This nature of IRC communications makes it difficult to trace such attacks to their origin. Additionally, IRC offers the ability to communicate with each instance of the trojan individually via private messages.

To send a message to all channel members, the attacker would simply type the desired text at the IRC client's input prompt. To send a message to a particular channel participant, the attacker would use the "/msg" command in the IRC client.

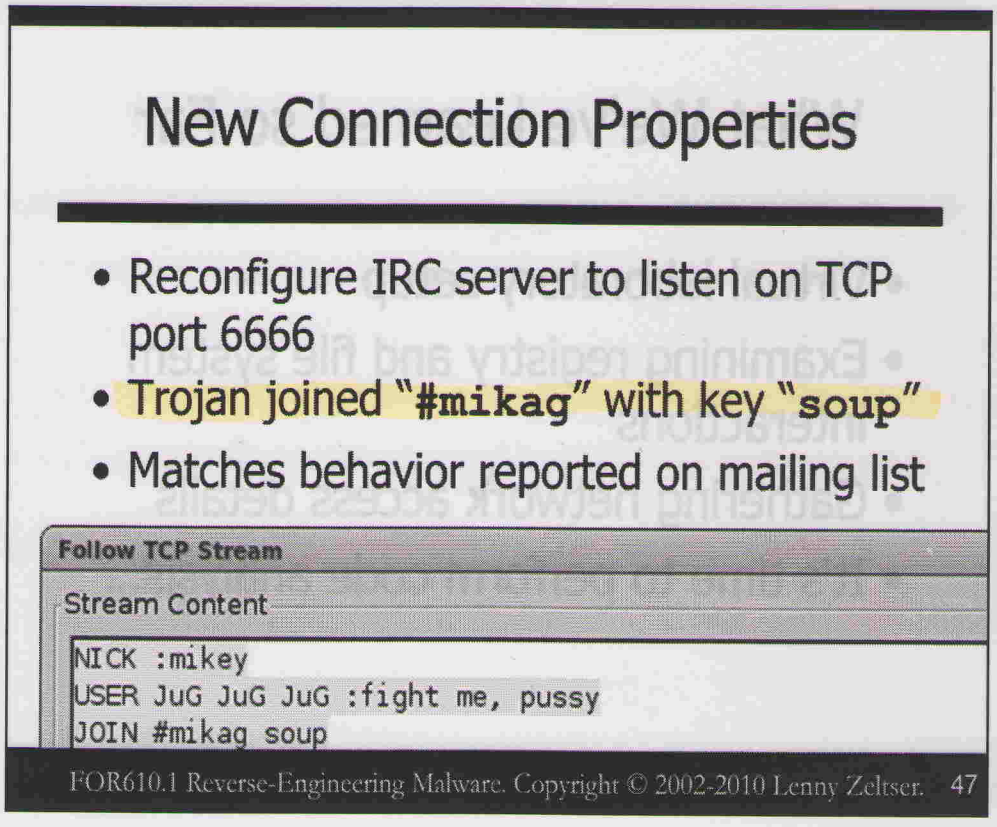


As mentioned earlier, srvcp.exe attempted to read the gus.ini file from the system directory upon start-up. I obtained a copy of this file from an infected machine in the real world, and placed it into the system directory of the laboratory machine. The gus.ini file itself was encoded in a way that prevented me from immediately learning its contents.

With the gus.ini file in place, the trojan attempts to connect to TCP port 6666 on the IRC server, instead of the TCP port 6667 that it used earlier. This connection attempt would fail if our server is not listening on TCP port 6666. As shown on the next slide, the trojan would then attempt resolving host names of various other EFnet IRC servers.

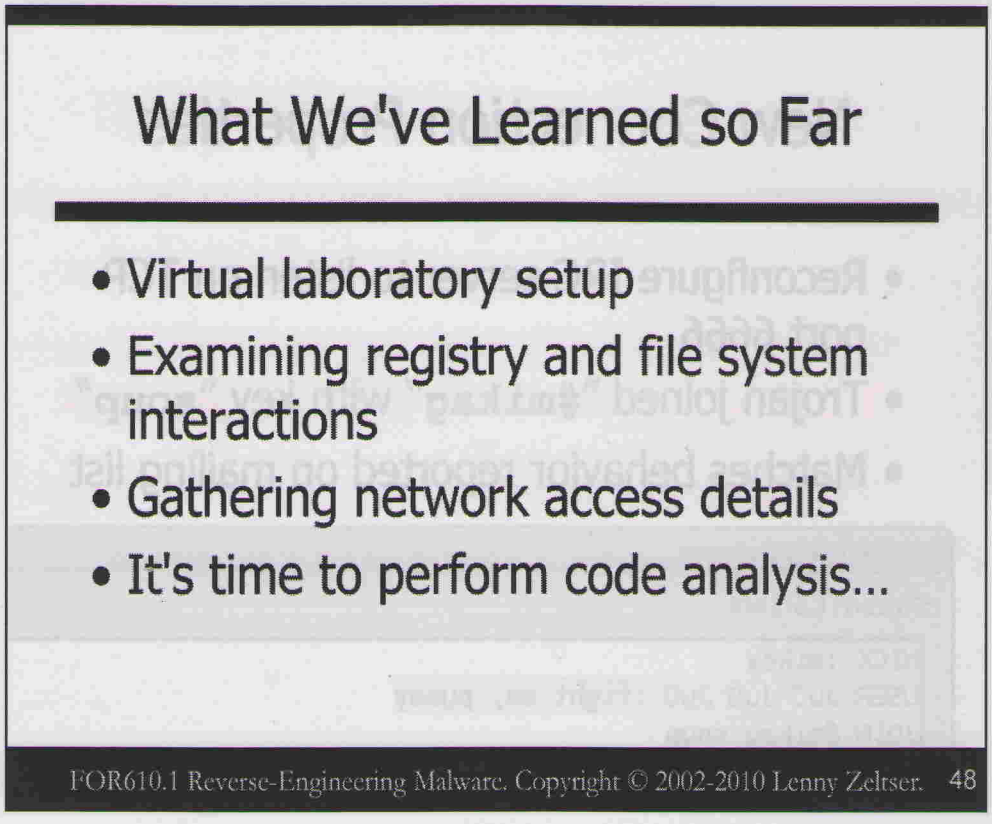


As you can see on this slide, the trojan attempts resolving a number of IRC-related hostnames that we did not observe before, if it is unable to connect to the initial IRC server on the expected port. This change in behavior is most likely related to the gus.ini file that I placed on the infected system. Although the network trace on this slide only mentions three different hostnames, the trojan attempts resolving a number of other names as well.



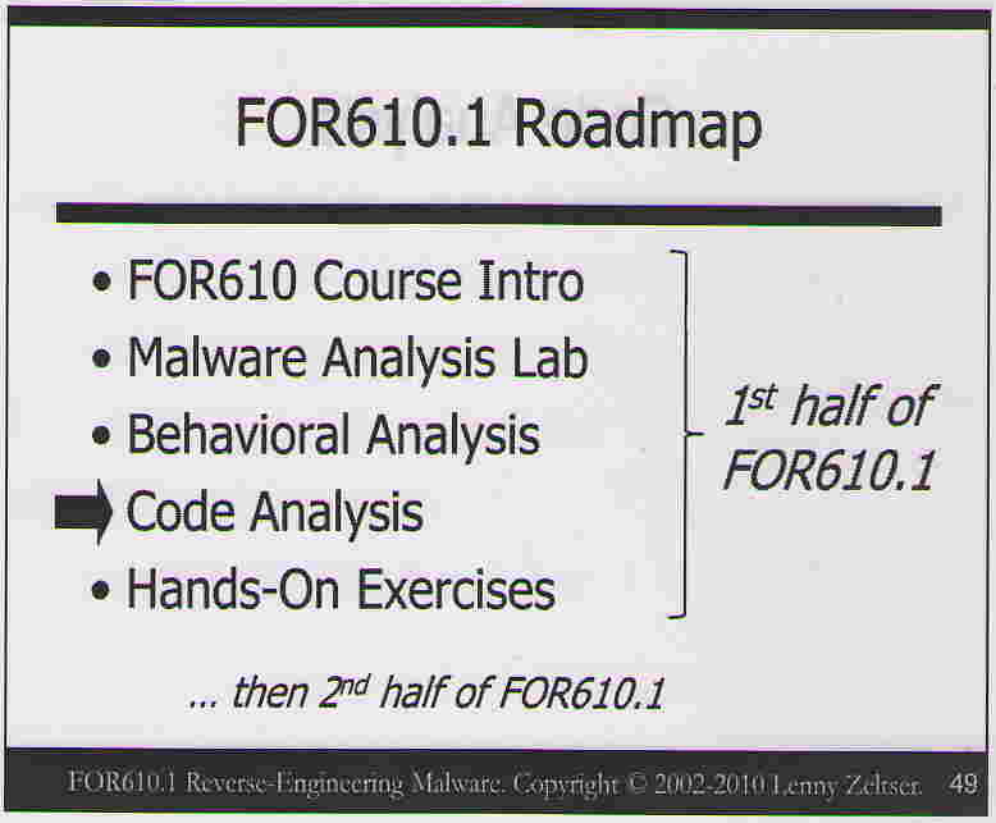
In response to observed behavior, could allow the trojan to resolve one of the hostnames it was looking for, say irc.mcs.net, if that hostname is not yet in the lab system's hosts file. You would then observe the trojan connecting to that host on TCP port 6666. You would configure the IRC daemon on our server to listen to TCP port 6666. Once srvcp.exe is restarted, the sniffer shows that the trojan successfully connected to the IRC server on the new port.

This time, the trojan joins the channel "`#mikag`" with the key "`soup`", as shown in the network trace on the slide. Before we made gus.ini available to the trojan, it used a different channel name, and did not supply a channel key. (A key is sometimes used on IRC to restrict access to a channel.) In fact, the trojan's behavior now matched observations reported to the Incidents mailing lists that we discussed earlier in the course.

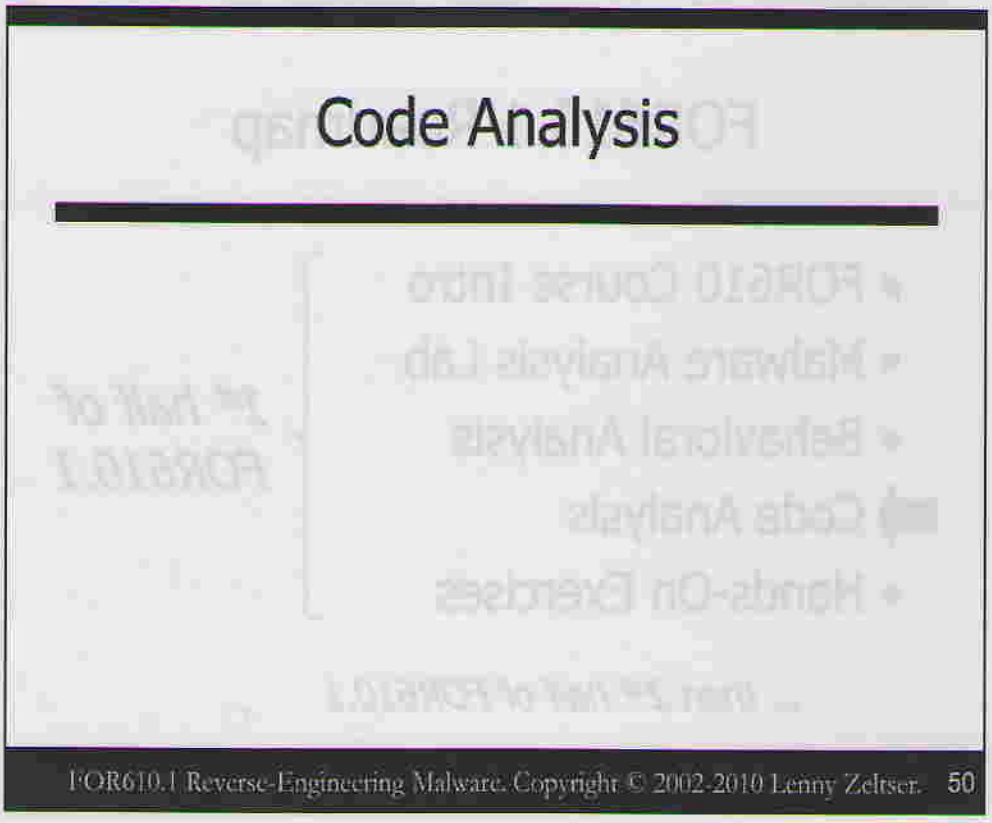


What have we learned so far in this session? We looked at a way of setting up an isolated and flexible laboratory environment for reverse-engineering malware. We also looked at several tools and techniques for examining registry, file system, and network-related activity associated with the malware specimen.

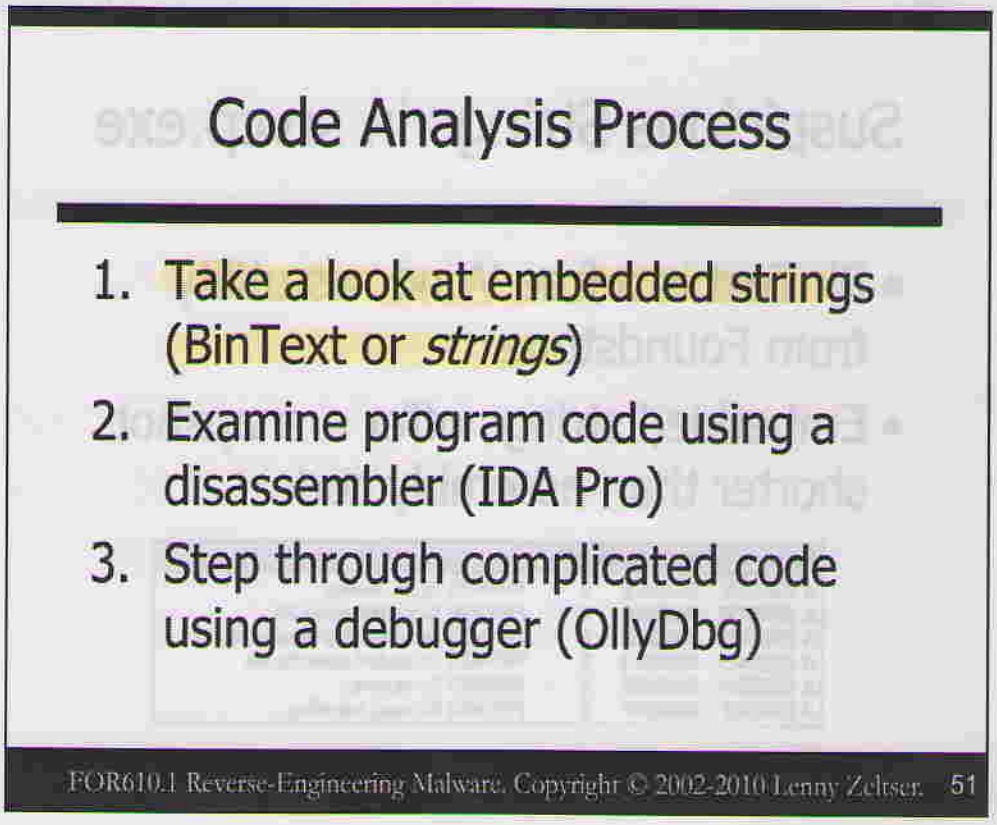
It's time to turn our attention to the code analysis phase…



With the essentials of behavioral analysis behind us, let's look into the code analysis phase of the reverse-engineering process.



While in the previous stage we used behavioral techniques to learn about the malware specimen, in this stage we will focus on the program's code to understand its inner-workings.



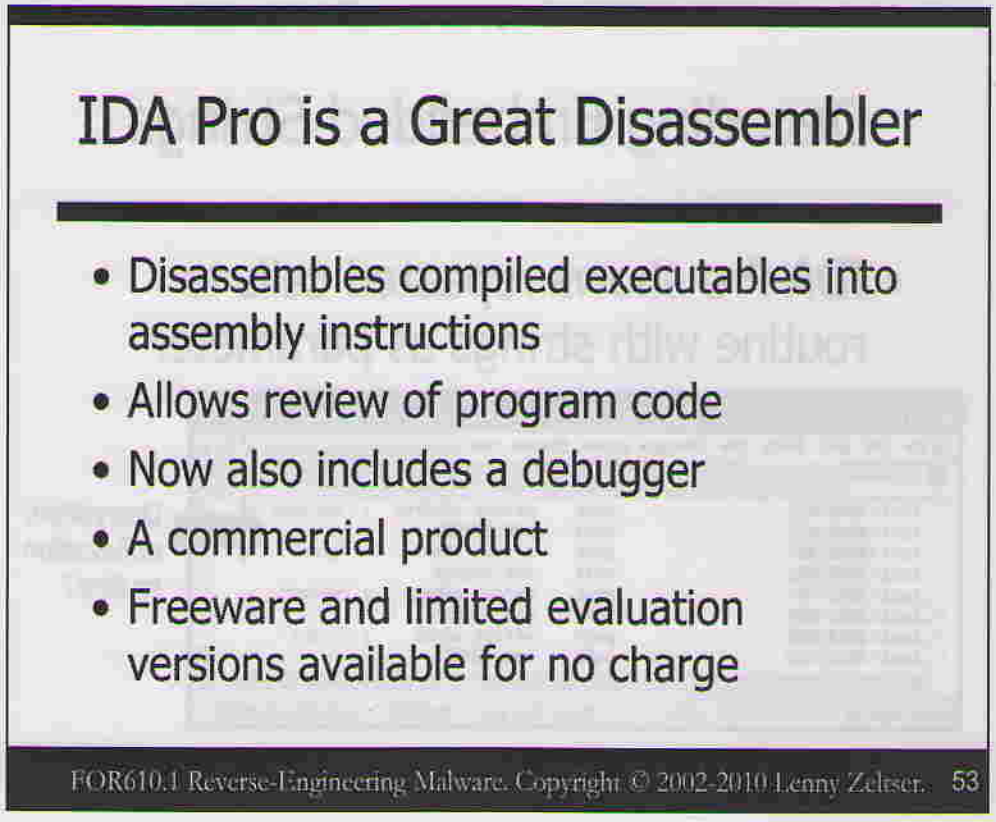
Now that we have a better understanding regarding the trojan's interactions with its environment, we will continue the discovery process by concentrating on the trojan's executable itself. In general, you are unlikely to have source code to a relatively unknown trojan, and will need to resort to using three types of tools to reverse engineer the executable: a string extractor such as *strings* for Unix or BinText for Windows, a disassembler such as IDA Pro, and a debugger such as OllyDbg.

Suspicious Strings in srvcp.exe

- BinText is a free Windows utility from Foundstone
- Embedded strings offer a snapshot shorter than assembly code

File pos	Mem pos	ID	Text
A 0000547E	0040847E	0	PRIVMSG %s :ok.. running
A 00005498	00408498	0	PRIVMSG %s :couldn't spawn file
A 000054B9	004084B9	0	PRIVMSG %s :successfully spawned ftp.exe
A 000054E3	004084E3	0	PRIVMSG %s :couldn't spawn ftp.exe
A 00005507	00408507	0	PRIVMSG %s :no more...
A 0000551F	0040851F	0	PRIVMSG %s :ready and willing...

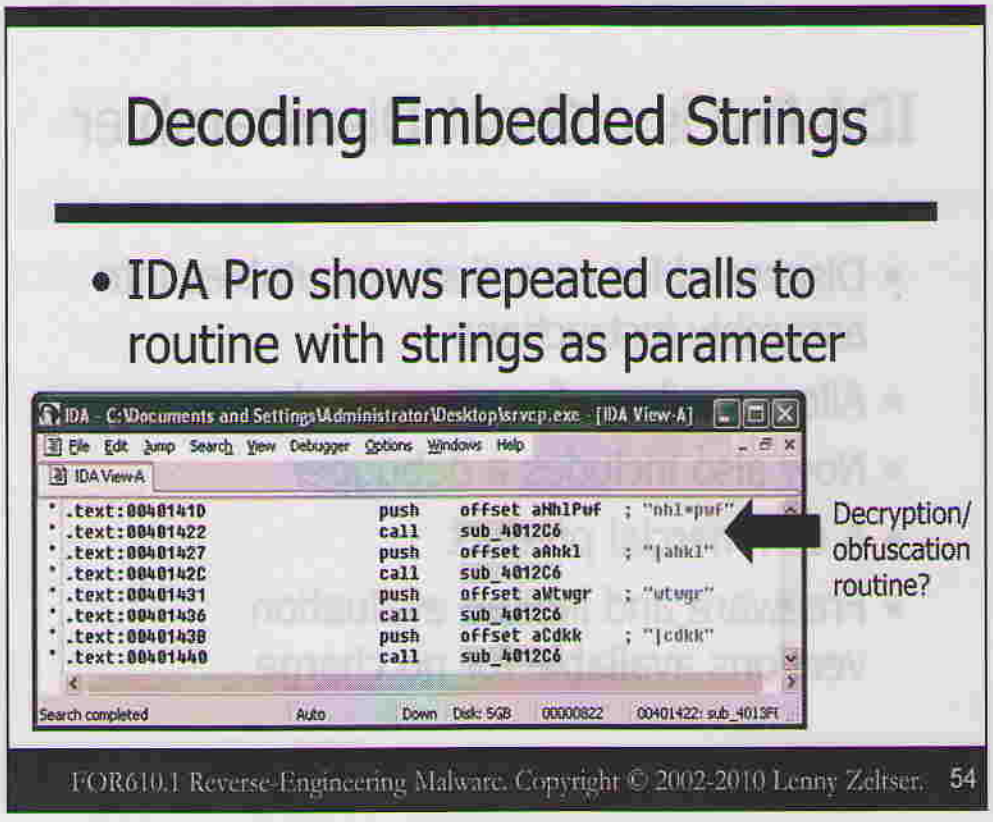
Looking at strings embedded into an executable is an easy, but not very fulfilling, way of getting an initial sense for the specimen's capabilities. We will use the BinText tool for Windows to extract embedded strings contained in the executable. This slide shows a few dangerous-looking strings that I found in srvcp.exe.



A complete listing of a disassembled executable offers much more details than the program's string snapshot. We can use the excellent disassembler IDA Pro to decompose the malicious program into assembly instructions. IDA Pro Standard can be purchased from http://www.hex-rays.com/idapro. (It was formerly offered by DataRescue, but the product has been migrated to a new organization called Hex-Rays.)

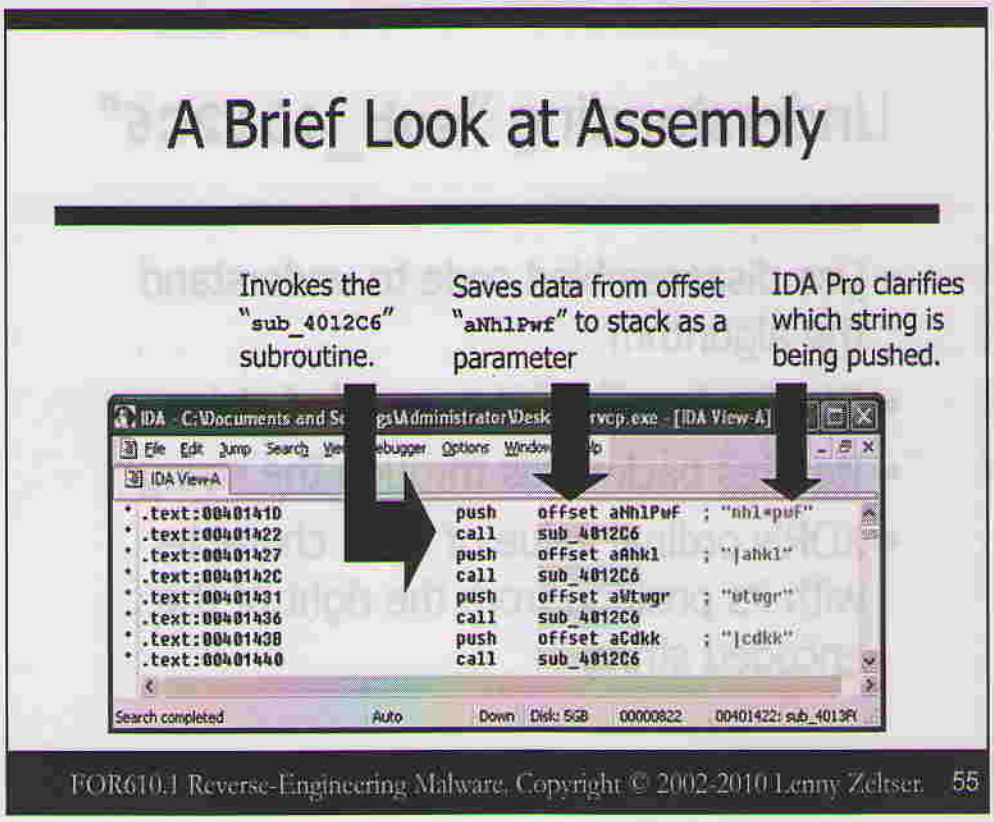
Before purchasing the program, you might take advantage of its limited demo version, which is available for free and might prove to be sufficiently useful during early stages of the analysis.

The company also offers IDA Pro Freeware, which is available for no charge for non-commercial use. It is also slightly behind (in features) from the latest commercial version of the program. You may download this version from Hex-Rays' website. It is also available on the DVD you received for this course.

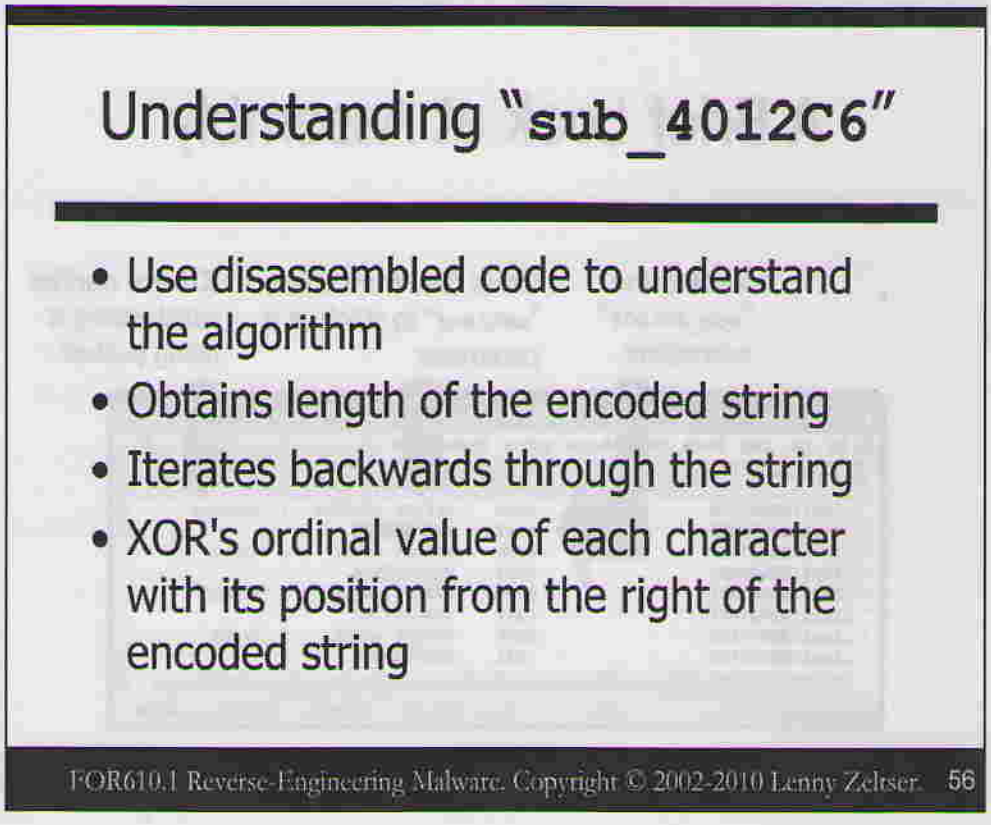


Parsing the specimen's executable with *strings* reveals a number of embedded strings that seem to be encoded, which means they are either encrypted or obfuscated. In an attempt to learn more about these strings, we can load srvcp.exe executable into IDA Pro, which would automatically deconstruct the executable into assembly instructions.

A section of the program's code presented on the slide shows the specimen repeatedly calling the same subroutine after pushing to the memory stack strings that seem to be encoded. This suggests that "`sub_4012C6`" might be the decryption or deobfuscation routine that the program uses upon start-up to decipher strings embedded into its executable.

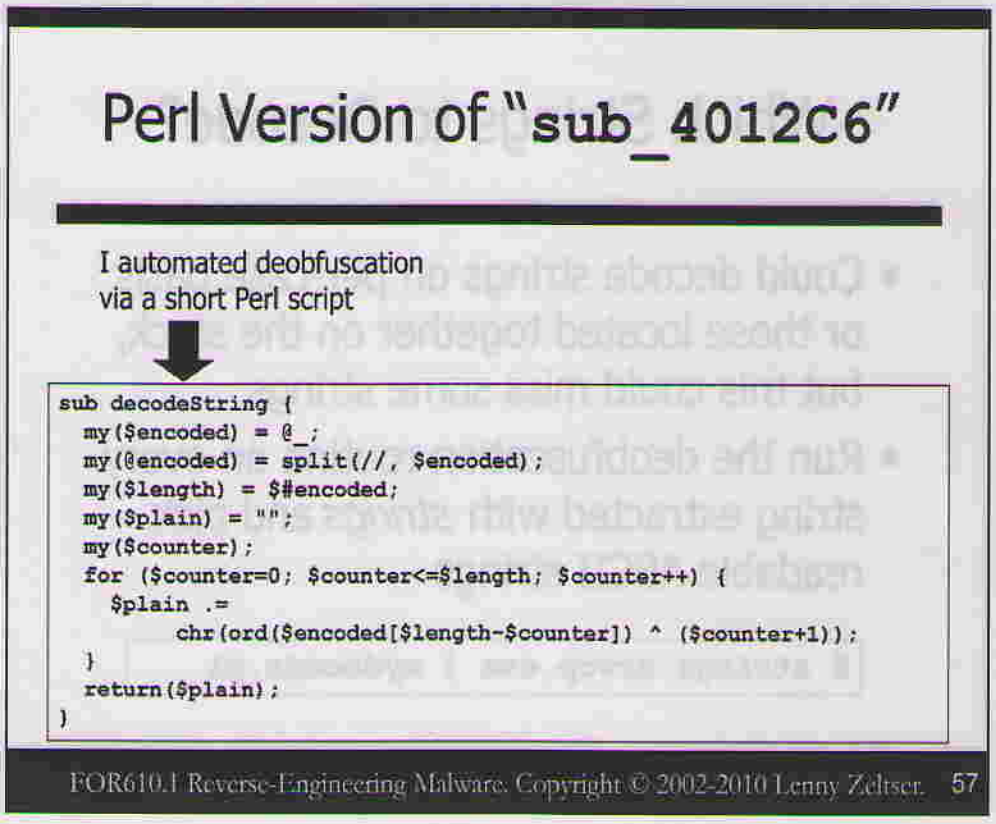


You don't need to be an assembly expert to get a sense for what this piece of code might be doing.



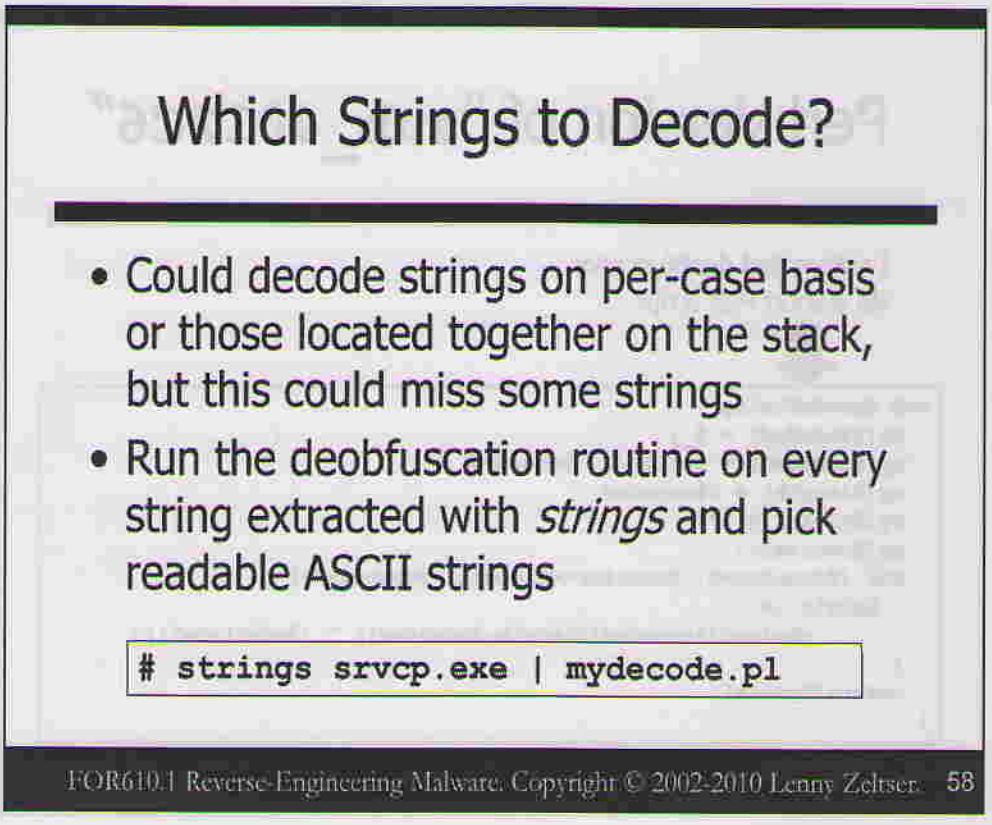
When analyzing this trojan, I was fortunate to have come across a paper written by Joe Abrams, in which he talks about several aspects of the srvcp.exe specimen. In particular, he discusses the algorithm used by the specimen to decrypt embedded strings. His paper served as a very helpful guide to understanding the program's assembly code. I mention it here to give him credit for his excellent document, as well as to emphasize the effectiveness of building upon findings published by other researchers. (You can access Joe's paper at http://www.hackinthebox.org/article.php?sid=1138.)

I examined disassembled code of "`sub_4012C6`" to understand the decryption algorithm while following Joe's interpretation of the process. The routine obtained the length of the encoded string, which was passed to it as a parameter, and iterated through it backwards. While looping through characters of the encoded string, the routine XOR'ed the ordinal value of each character with the character's position from the right of the encoded string. This resulted in the deobfuscated version of the string, which was returned upon the routine's exit.



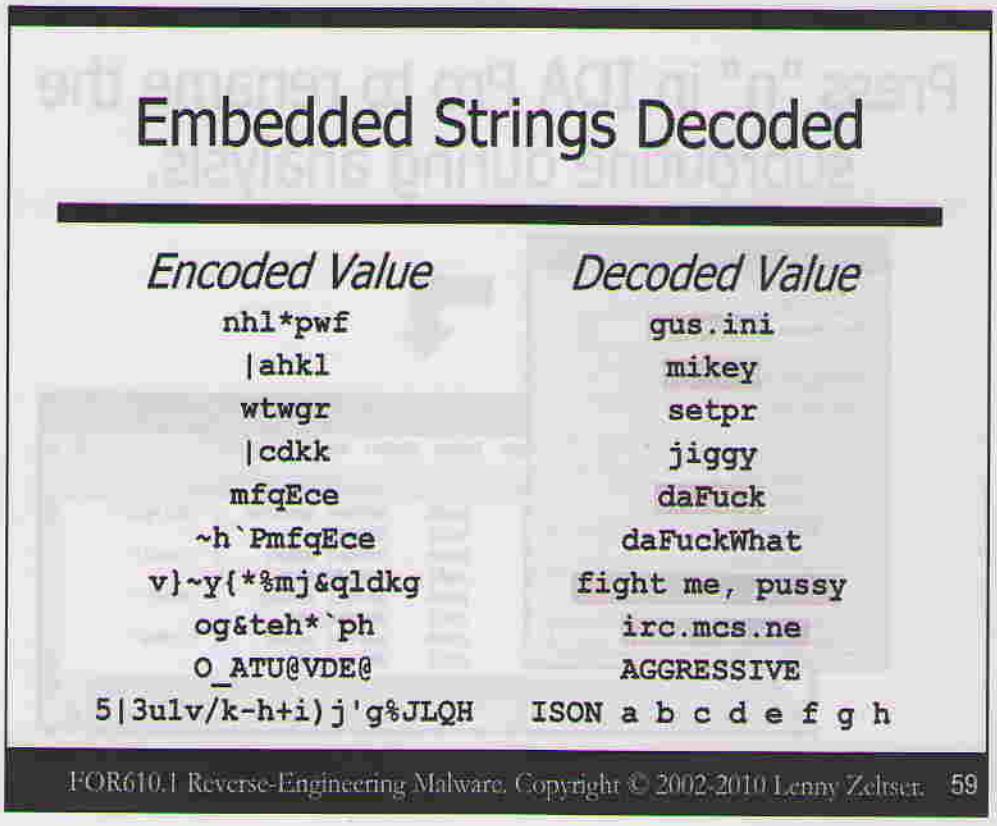
To verify my understanding of the decryption algorithm, and to attempt to deobfuscate strings embedded into srvcp.exe, I implemented the "`sub_4012C6`" routine in Perl. The Perl version of the routine, shown on this slide, is much more readable than its assembly counterpart.

Scripting languages such as Perl and Python are useful for automating miscellaneous tasks related to analysis of malware. They are supported on most Unix and Windows platforms. To run Perl on Windows, see http://www.activestate.com.



Now that I had the Perl routine for deobfuscated embedded strings, I could have used it to decode strings one by one. The challenge of this approach, based on manually picking out "interesting" strings, is to determine which strings should be decoded.

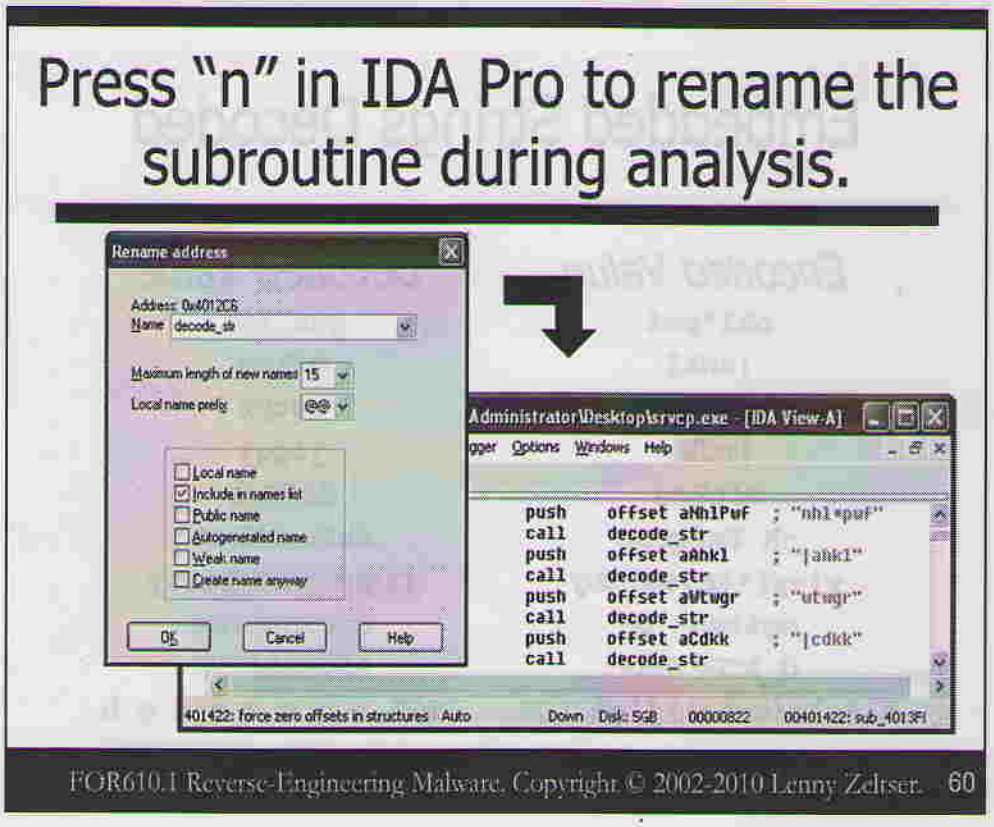
Instead, to ensure that I did not miss any encoded strings, I parsed srvcp.exe using the Unix-based *strings* program, and used a Perl script to automatically attempt deobfuscating each string. I then looked through the resulted list of strings to manually single out readable ASCII strings. My findings are presented on the next slide.



This slide shows strings that were determined to be embedded into srvcp.exe in an obfuscated form, along with the decoded representation of each string. Note that some of these strings were already mentioned in the analysis in earlier slides.

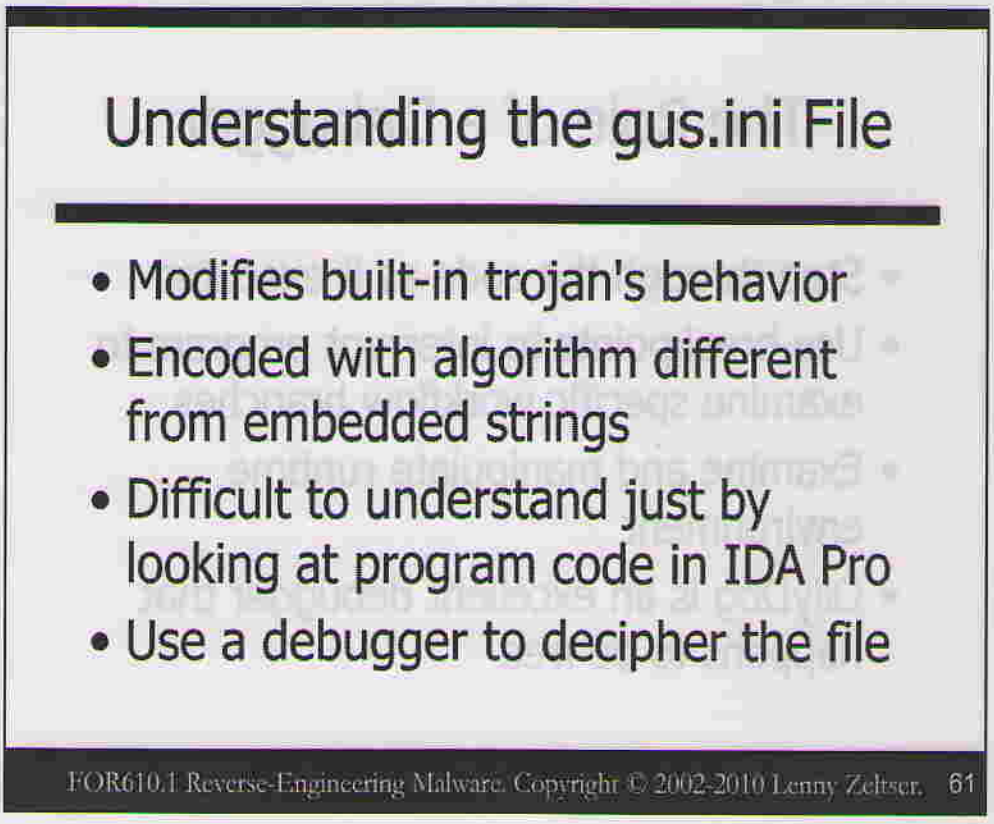
In particular, "`gus.ini`" was the name of the file that the trojan attempted to locate upon start-up to change several aspects of its behavior. The "`mikey`" string matches the nickname that was used when connecting to an IRC server. The "`daFuck`" string, when prefixed with "`#`", is the name of the IRC channel that the trojan joined. The "`fight me, pussy`" string matches the real name property of the trojan's IRC user, as seen by the IRC server. Finally, "`irc.mcs.ne`", with the "`t`" character tagged on, is the host name of the IRC server that the trojan attempted to connect to; the final character is missing from the embedded string probably because the *strings* program was unable to extract it properly.

Now we have a better understanding of what controls the trojan's behavior even when the gus.ini file is missing from the system. The next set of slides concentrates on understanding contents of the gus.ini file, which should provide us with additional insight into the trojan's inner workings, and explain the purpose of other strings listed on this slide.



Authors of malware typically remove symbolic names to internal functions when compiling the executable to make it harder for anti-virus tools and human analysts to identify and understand them. As you examine the malicious program to understand the purpose of a subroutine, rename it in the disassembler to make it easier to read and understand the code.

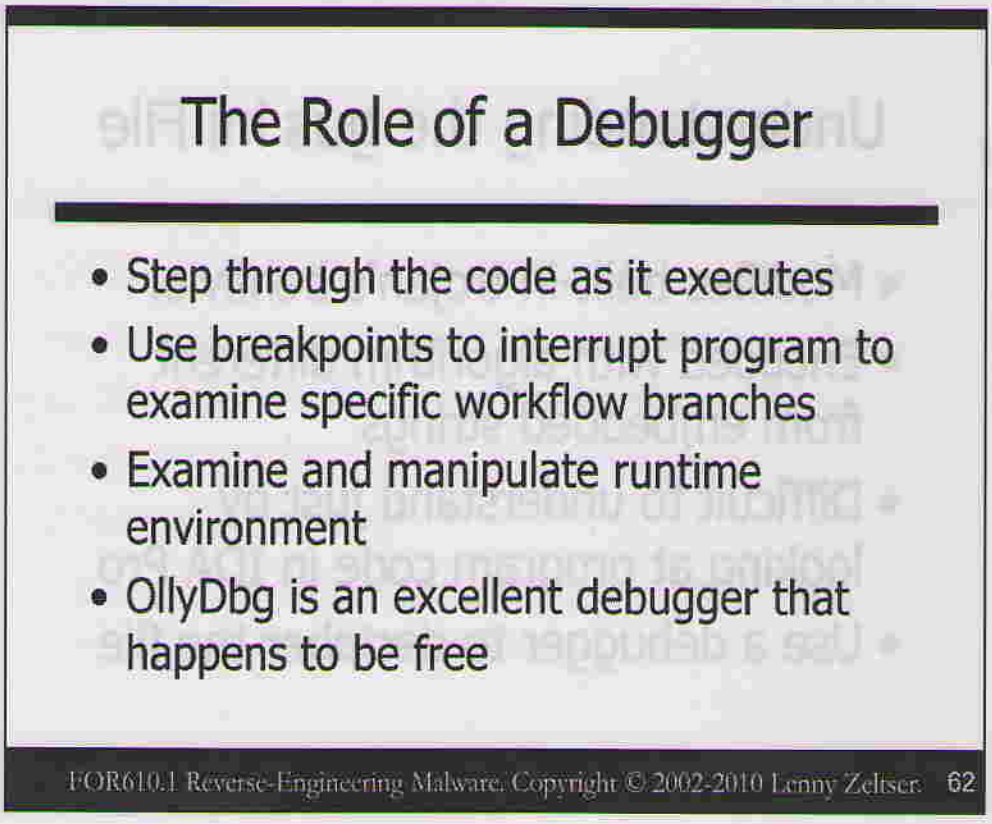
You can do this in IDA Pro by clicking on the name of the subroutine and then pressing the "n" key on your keyboard. IDA Pro will bring up the window shown on the left side of this slide. Enter the new name for the function (I used "decode_str") in this case, and press OK.



What we know so far about gus.ini is that it is read by the trojan when it starts up. We also know that placing gus.ini onto the infected machine modifies the trojan's behavior, apparently overwriting its default values for the IRC server name and port, the channel name, and possibly others.

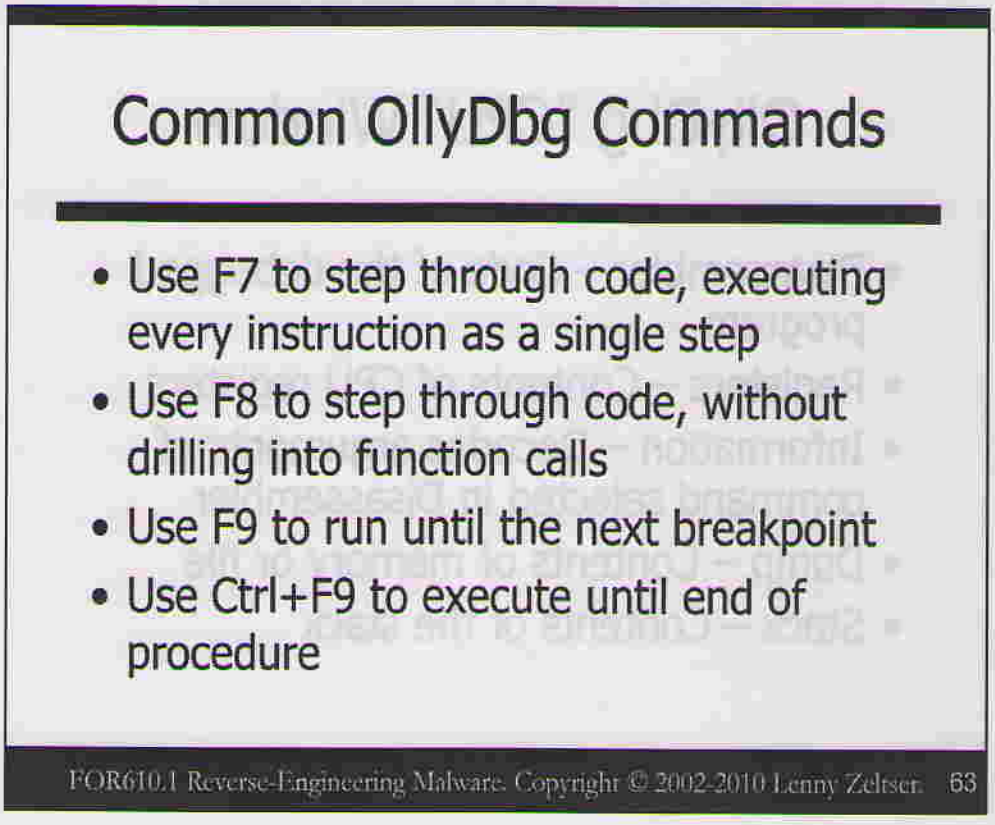
The gus.ini file is stored in an encoded format. I tried deciphering its contents using the decryption routine for embedded strings that were discussed earlier, but failed to produce intelligible text. I then tried tracing the srvcp.exe workflow related to gus.ini decryption in IDA Pro, but that turned out to be too difficult; therefore, I decided to use debugging and environment analysis capabilities of a debugger to decipher contents of gus.ini, as we will discuss in the next set of slides. For your reference, here is an initial portion of the encrypted gus.ini file:

```
JexO215WuK60H7HgI.j11vh1
Or6ZF1EY6FP/Esknw.4bCXN.
Rzply/0QhQ9/Dul3j1ex9J2.
jCggY1dbThf/7FoGR/5IYU/.
HBtJI.zWZtP/e1zcT/nCMAf0Osi.K.vC3lT1
ZC8YD.MBoxJ.wtPW61fAKYi1Vnu6H/yPVda.
YxPgS13wXdq0m4SMh/4NhJj0hN2gw/J/L.W1
fVN.20dmo331uJaSo/CoFfs1RYrQy.lHqPM.bjDB6.d22dU.
YtqCd0dK7Ts1Ej1ZC0Spl R2/pdxlr/i.KQu1L8JvE0iBK82/
aoQLZ/DVMQD0CWVM8.x1CkA0oEMAd.bf8PG13y62h0YUKEV.
pNdkb0wdFFa.mcNo21rXfue/gz6OS/jjCvK.0fNC50tvylg0
Erre71dq9e80r/Z1k.ZxMC4/IbM24/jNtv100fNC50tvylg0
w1UOB1pMDRr.OqcNd.5sPlg/dFB9Z0p4z3J.AyQVA1f1nog.
… cut for brevity …
```



Because assembly is a low-level language, expect to encounter difficulties understanding the flow of some of the more cryptic portions of the code when looking at them with a disassembler. This is where a debugger, such as OllyDbg is of immense help.

Debuggers let you execute malware under highly controlled conditions, with the ability to step through the program as slowly as one instruction at a time. You probably won't have the patience to manually step through every single instruction, so you will want to take advantage of the debugger's ability to set breakpoints that interrupt the execution of the program at specific workflow branches. When stepping through portions of the program, you can peek at its memory and register contents, and even modify this information on the fly.

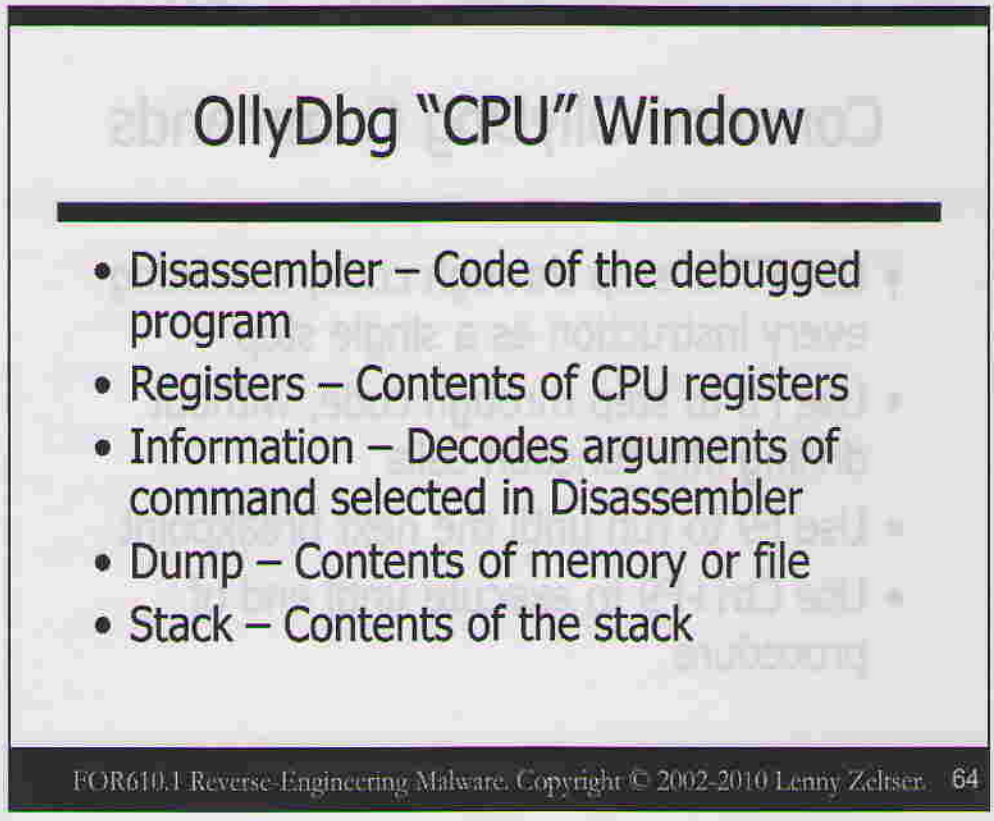


To debug a program you can open it in OllyDbg by selecting Open from the File menu. This will open the executable, letting you examine portions of its code, look at which libraries it will load, and pre-set the appropriate breakpoints. OllyDbg will not start executing the program until you select Run from the Debug menu, or press F9. OllyDbg also lets you attach to an already running executable by selecting Attach from the File menu.

When stepping through the debugged program, we can use the F7 key to step through the code by executing every instruction as a single step. The F8 key works in a similar manner, but does not drill into function calls, executing them in a single step. The F9 key runs the executable until the next breakpoint.

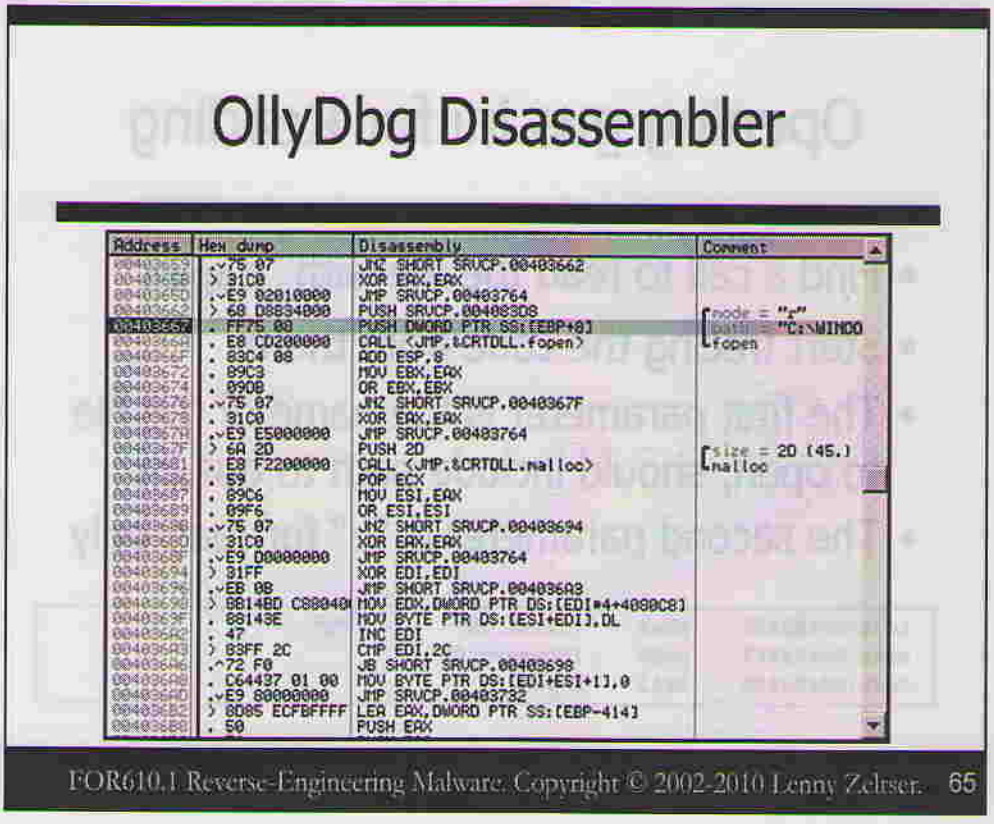
The Ctrl+F9 key combination is very useful when you are stepping through a subroutine one instruction at a time, are no longer interested in stepping through the rest of the subroutine, but want to pick up the debugging process right after the subroutine ends. Pressing Ctrl+F9 will execute all remaining instructions in the subroutine until its end, and then interrupt the execution of the program.

For additional OllyDbg usage tips, shortcuts, and hotkeys, see http://www.ollydbg.de/quickst.htm. Also, take a look at "Reversing for Newbies" video tutorials at http://tuts4you.com/download.php?list.17.



When debugging a program with OllyDbg, you will spend most of your time in the CPU window of OllyDbg. This window is divided into several regions that contain most of the information you would be interested in. These regions are listed below. Please note that sometimes OllyDbg's documentation (the Help file) uses the word "region" when referring to one of these regions:

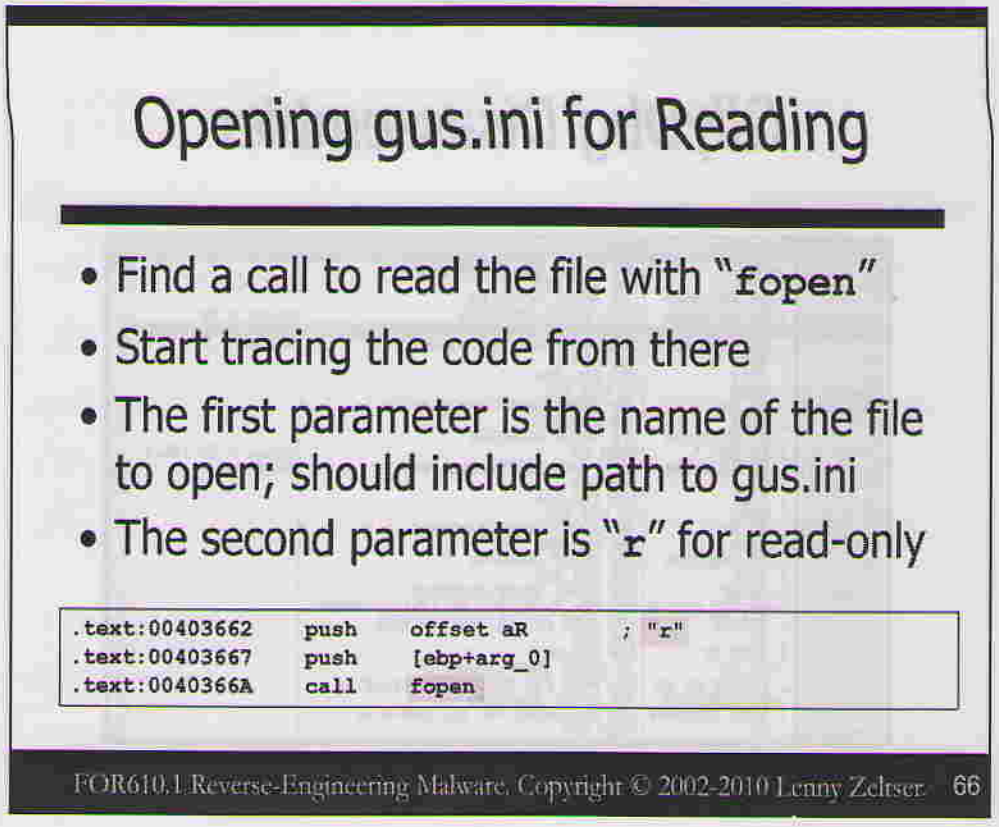
- Disassembler – This region shows assembly code of the debugged program.
- Registers – This region shows contents of CPU registers. You have some control over which registers are displayed in here.
- Information – This small region shows additional information about the command that is currently selected in the Disassembler region.
- Dump – This region shows contents of memory or file.
- Stack – This region shows contents of the stack that is used by the current thread. The *stack* is a special data structure that a program uses to keep track of useful data, memory pointers, saved register contents, function parameters, and other pieces of information that it is currently using or wants to track.



The Disassembler region contains assembly code of the debugged executable, and provides information similar to what would be available via IDA Pro. You may use the Disassembler built into OllyDbg instead of IDA Pro; however, I find that IDA Pro is a little friendlier for analyzing portions of the code without having to execute them. On the other hand, the advantage of OllyDbg's Disassembler is that it attempts to intelligently interpret the code based on what it has learned from having executed portions of the debugged program.

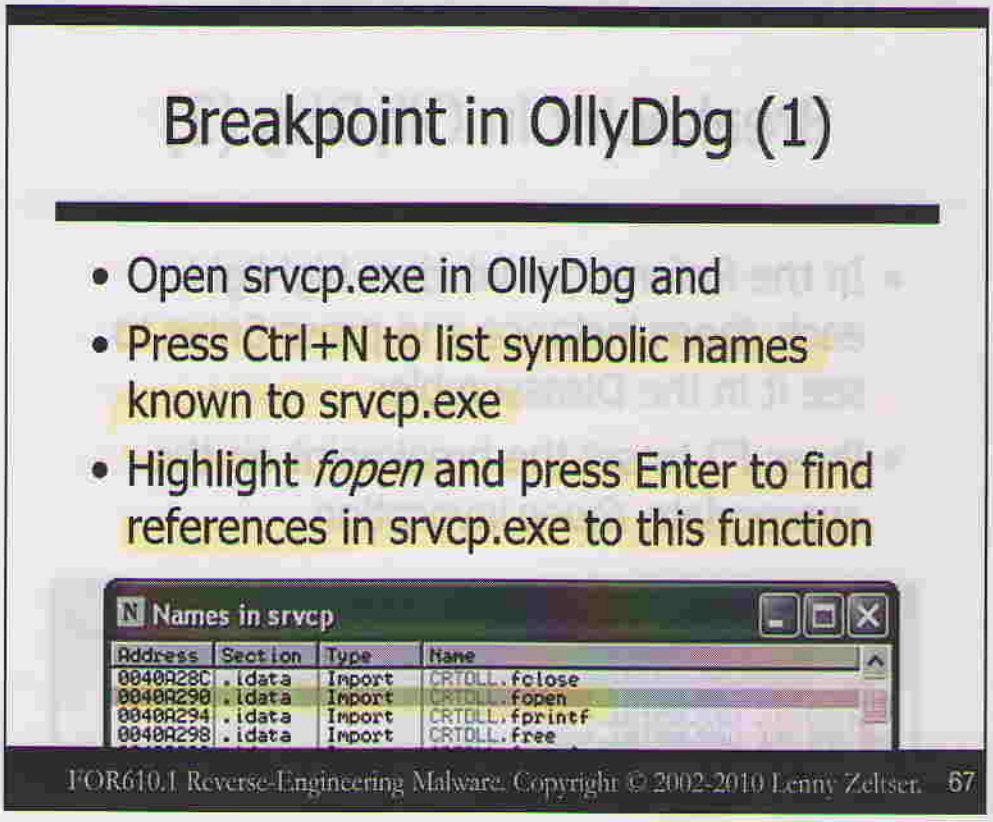
In the screen capture on this slide, the highlighted instruction was just executed on the system. As you can see, the next instruction will be a call to *fopen*. OllyDbg is nice enough to visually group the call to *fopen* with a bracket in the right column. The Disassembler here also displays which parameters are being passed to *fopen*.

When reviewing disassembled code, you can insert your own comments. Commenting the code as you analyze it makes it easier for you to understand it when you review your analysis later. OllyDbg remembers your comments and breakpoints between sessions.



After observing the trojan's behavior, we surmised that it opened the gus.ini file for reading and decrypted its contents during runtime; we are going to use a debugger to step through srvcp.exe as it is decrypting the file in an attempt to understand the decryption process.

Looking through the trojan's code disassembled in IDA Pro, we can see a code segment that makes the *fopen* call shown on this slide. This is the only occurrence of *fopen* being used with the "`r`" parameter, which suggests that this is where the program will attempt to read gus.ini. This matches file system activity reported by Process Monitor that I discussed in earlier slides. The "`arg_0`" parameter pushed onto the stack before calling *fopen* represents the name of the file to open, which in this case should be gus.ini.



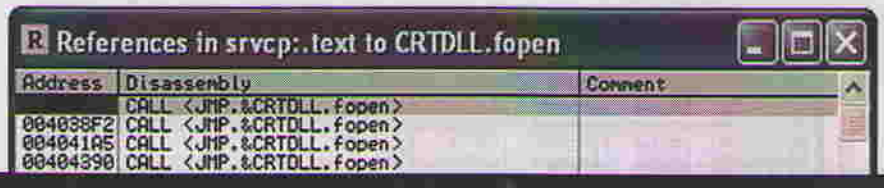
To ensure that we are correct in our understanding of how the gus.ini file was opened, we can use OllyDbg to examine the trojan's runtime when *fopen* is called. We will use similar techniques in the hands-on portion of the course, so be sure to pay attention.

First, you would load srvcp.exe into OllyDbg. Before even executing the trojan, you would press "`Ctrl+N`" to list all symbolic names known to the program. This is a very useful function because it gives you an idea of the kinds of external calls that the executable makes without ever having to load it into IDA Pro.

Here, you can scroll down to locate *fopen*, or simply type "fopen" to let OllyDbg locate it for you. Once *fopen* is selected, pressing Enter will open another window, shown on the next slide, which lists all assembly instructions in srvcp.exe that reference *fopen*.

Breakpoint in OllyDbg (2)

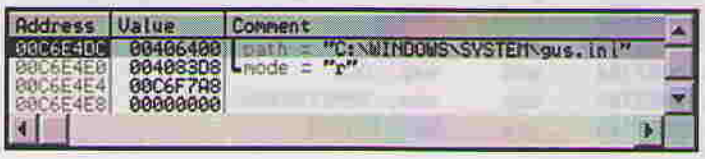
- In the References window highlight each *fopen* instance and press Enter to see it in the Disassembler
- Press F2 to set the breakpoint on the appropriate *fopen* invocation



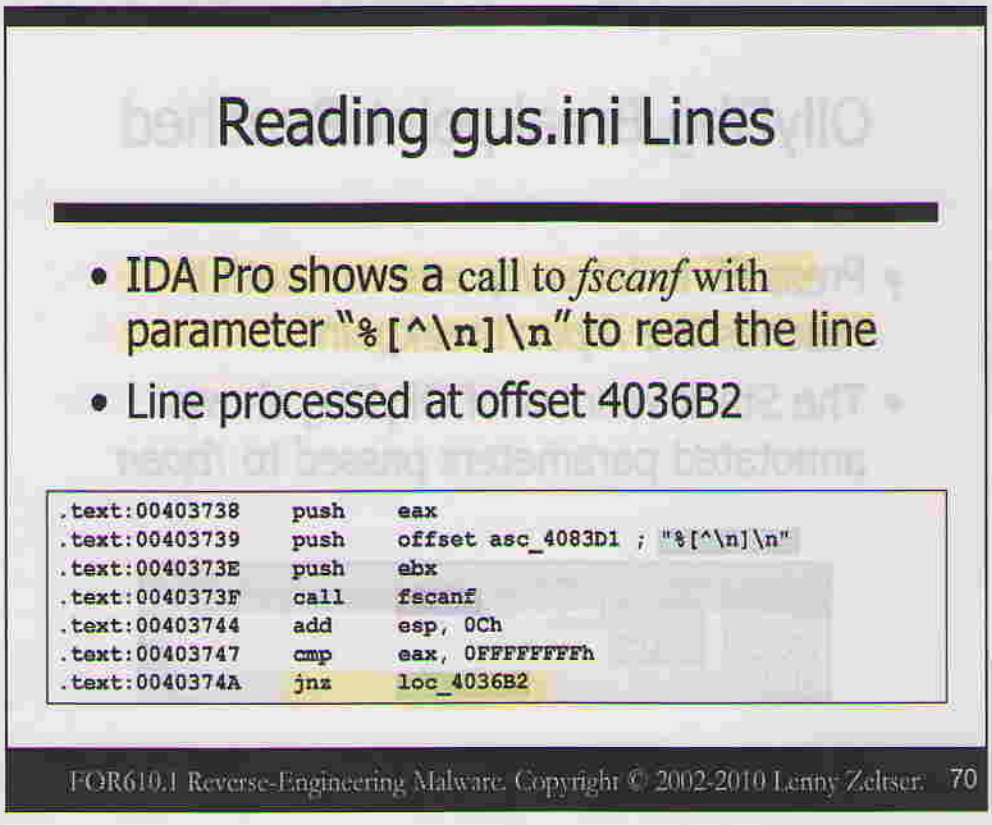
This slide shows a screen capture of the References window in OllyDbg, which lists all commands that make a call to *fopen*. Here you can double-click on each reference to open the Disassembler window to see the portion of the code that includes each reference. Once you located the appropriate invocation of *fopen*, you can press F2 to set the breakpoint on the highlighted line of the trojan's code.

OllyDbg Breakpoint Reached

- Press F9 to let srvcp.exe run until it reaches the *fopen* breakpoint
- The Stack window of OllyDbg shows annotated parameters passed to *fopen*

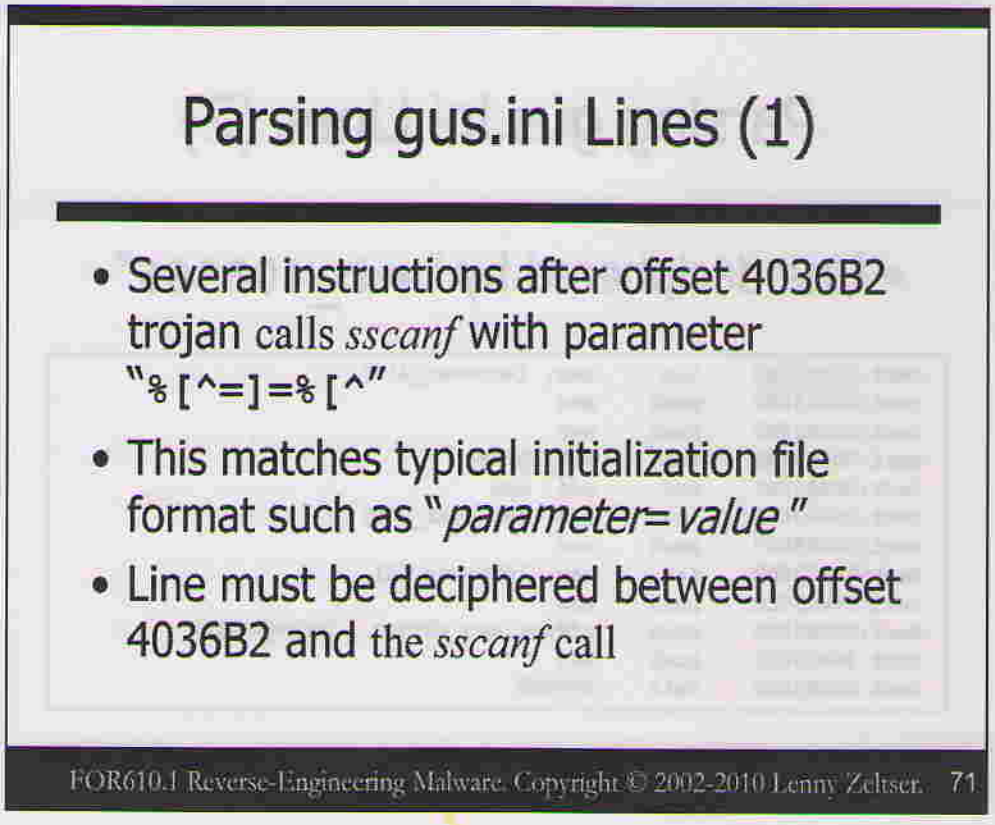


After setting the breakpoint, you can press F9 to let srvcp.exe run until it reaches the *fopen* call. Once OllyDbg interrupts the program's execution, you have a chance to examine its runtime environment. Specifically, if you look at OllyDbg's Stack window, you will see which parameters were passed to *fopen*. As you see on the screen capture on this slide, the mode for opening the file is, indeed "`r`" and the file being opened is gus.ini, as we expected.



Now that we know where in the code the trojan opens the gus.ini file for reading, how do we find out where the file's contents are decrypted? Let's start by finding the place in the code where it first references the decrypted version of the file.

As shown on this slide, at offset 40373F the program calls the *fscanf* function with the parameter "`%[^\n]\n`". This is one of the ways to read in a whole line from the text file, which is how the trojan reads in the gus.ini file line-by-line; at this point the read-in line is probably still encrypted. A few instructions further the executable jumps to another section of the code using the "`jnz loc_4036B2`" instruction; after the trojan returns from the jump, memory is cleaned up and the file handle closed. This suggests that file contents are processed by code located at the offset of 4036B2.



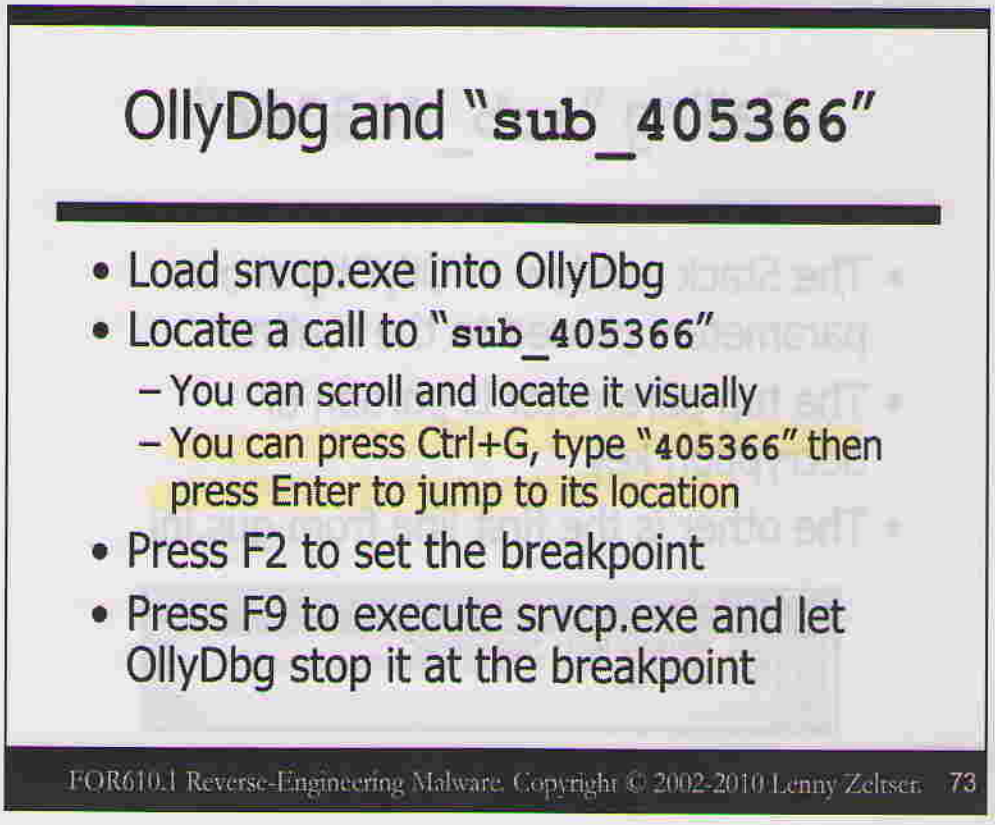
Several lines after the 4036B2 offset the program invokes the *sscanf* function, commonly used to parse strings, with the "%[^=]=%[^" parameter. This string pattern often represents the format of typical initialization files whose lines follow the convention of "*parameter=value*". Since the encrypted gus.ini file did not follow the standard initialization file format for separating parameter names and values with equal signs, the program must be decrypting each line before invoking *sscanf*.

Parsing gus.ini Lines (2)

- Line deciphered by "`sub_405366`"

```
.text:004036B2    lea     eax, [ebp+var_414]
.text:004036B8    push    eax
.text:004036B9    push    esi
.text:004036BA    call    sub_405366
.text:004036BF    mov     edi, eax
.text:004036C1    lea     eax, [ebp+var_9F0]
.text:004036C7    push    eax
.text:004036C8    lea     eax, [ebp+var_14]
.text:004036CB    push    eax
.text:004036CC    push    offset asc_4083C5 ; "%[^=]=%[^"
.text:004036D1    push    edi
.text:004036D2    call    sscanf
```


As you can see on this slide, the only routine called between offset 4036B2, where we obtained the encrypted string, and *sscanf*, where we already have a clear-text version of the string, is "`sub_405366`". This suggests that the gus.ini line is deciphered in the "`sub_405366`" routine.



It's relatively easy to analyze "`sub_405366`" with OllyDbg. First, load the specimen into the debugger and set the breakpoint on the call to the decryption subroutine. OllyDbg lets you load srvcp.exe and set internal breakpoints before executing it.

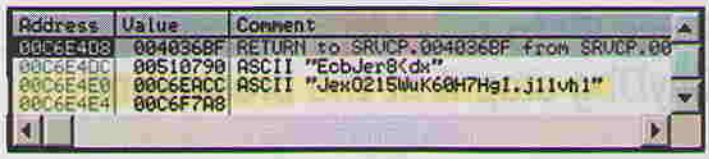
To locate "`sub_405366`" you can scroll visually in the Disassembler window to the desired address. You could also press Ctrl+G to bring up the window that will ask you to "Enter expression to follow". Type "405366" and press Enter. Alternately, you can use the Search function to locate the constant "00405366", highlight the line that calls the subroutine, and press Enter to jump to its location. To find this Search function, right-click on the Disassembler pane, select "Search for", then select "Constant".

Once the breakpoint on "`sub_405366`" is set you can press F9 to run srvcp.exe.

Setting the breakpoint at the actual address of the subroutine would put us directly into the subroutine itself. If you then want to execute it in a single step, press Ctrl+F9 to let OllyDbg run it until its end.

Calling "sub_405366"

- The Stack window of OllyDbg shows parameters passed to the routine
- The top parameter is portion of decryption key
- The other is the first line from gus.ini



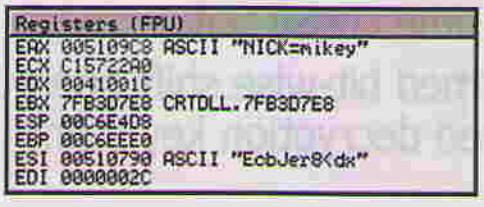
Once OllyDbg interrupted the execution of srvcp.exe at the beginning of "`sub_405366`" you can glance at the Stack window of the "CPU" region to see values of the parameters that were passed to it.

The top parameter on the stack, "`EcbJer8\0dx`", is a portion of what seems to be the decryption key. If you glanced at the Registers window of OllyDbg, you would see these values stored in EAX and ESI registers. The full decryption key, as would be seen in the ESI register is actually "`EcbJer8\0dx.CJVJsAlmIZ`" (note the null character in the middle, presented here as "`\0`"). OllyDbg doesn't seem to interpret the null character properly when displaying the parameters in the Stack window, and only shows the first portion of the string.

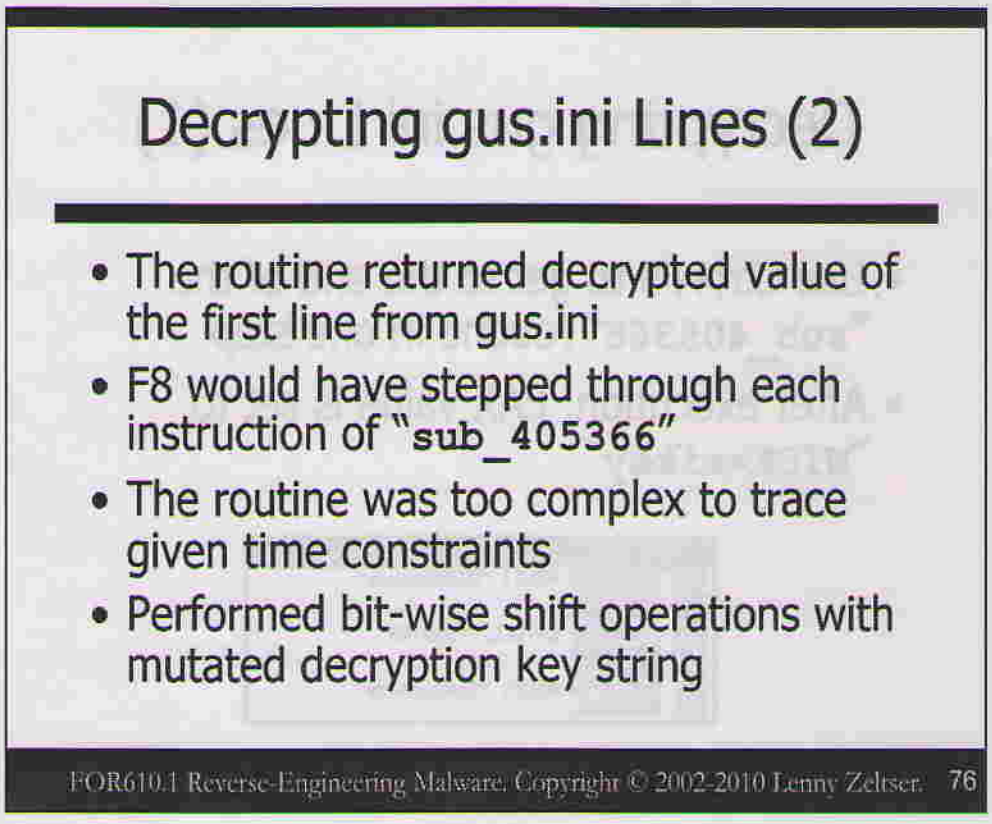
The lower parameter, "`JexO215WuK60H7HgI.j11vh1`", happens to be the first line of the encrypted gus.ini file.

Decrypting gus.ini Lines (1)

- Use Ctrl+F9 to get to the end of the "`sub_405366`" routine in one step
- After execution, EAX value is set to "`NICK=mikey`"



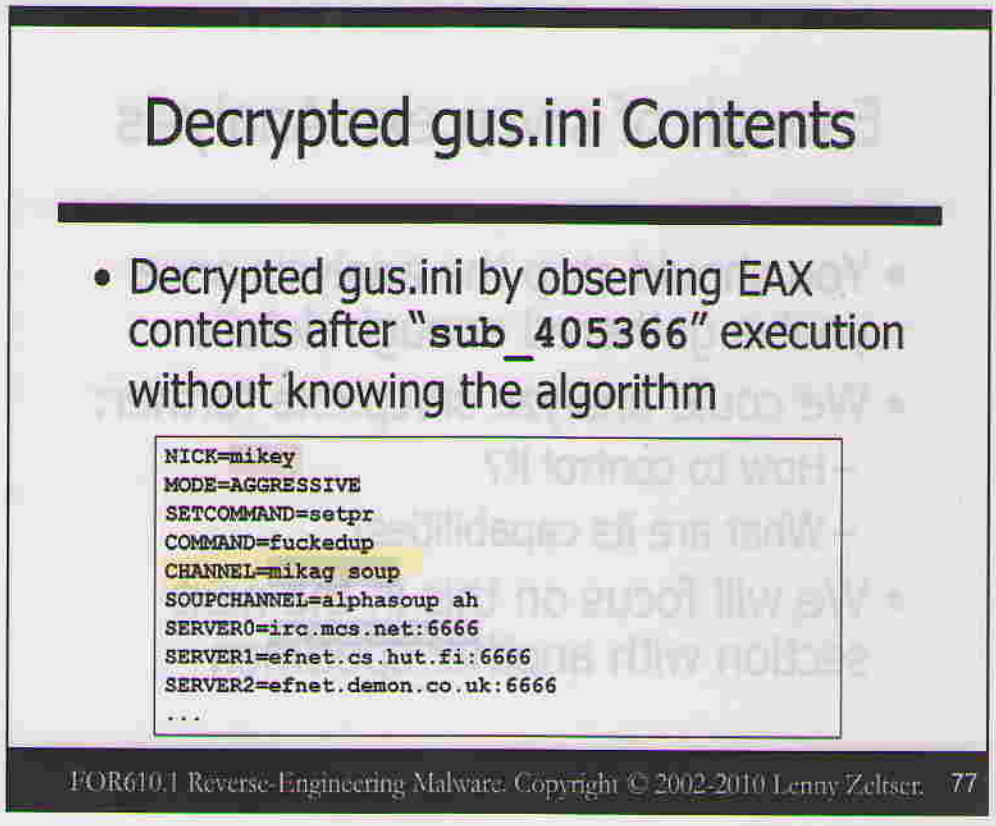
You can then use Ctrl+F9 to let OllyDbg execute the "`sub_405366`" routine in one step, and interrupt its execution at the end of the routine. You can then glance at the Registers window of the CPU region to see the value of "`NICK=mikey`" stored in the EAX register. This is the decrypted first line from gus.ini.



Going through this process several times for other lines of the gus.ini file confirmed that decryption took place in the "`sub_405366`" routine. During the analysis I also stepped into the decryption routine using the F8 key, and spent quite some time trying to understand the decryption algorithm. I could see that the routine performed iterative XOR and bit-wise shift operations using a mutated version of the decryption key string. Overall, the process was too complex to trace thoroughly given time constraints when performing this analysis.

I did gain some insight into how the routine mutated the first portion of the key that it received as one of the parameters. The algorithm was relatively simple: it extracted the portion of the key string until the null character, which turned out to be "`EcbJer8`" and added 7 to the ordinal value of each character in that string. I presume this was done to make recovery of the key a more difficult process for those analyzing the trojan. For your reference, the assembly fragment responsible for these manipulations is presented below.

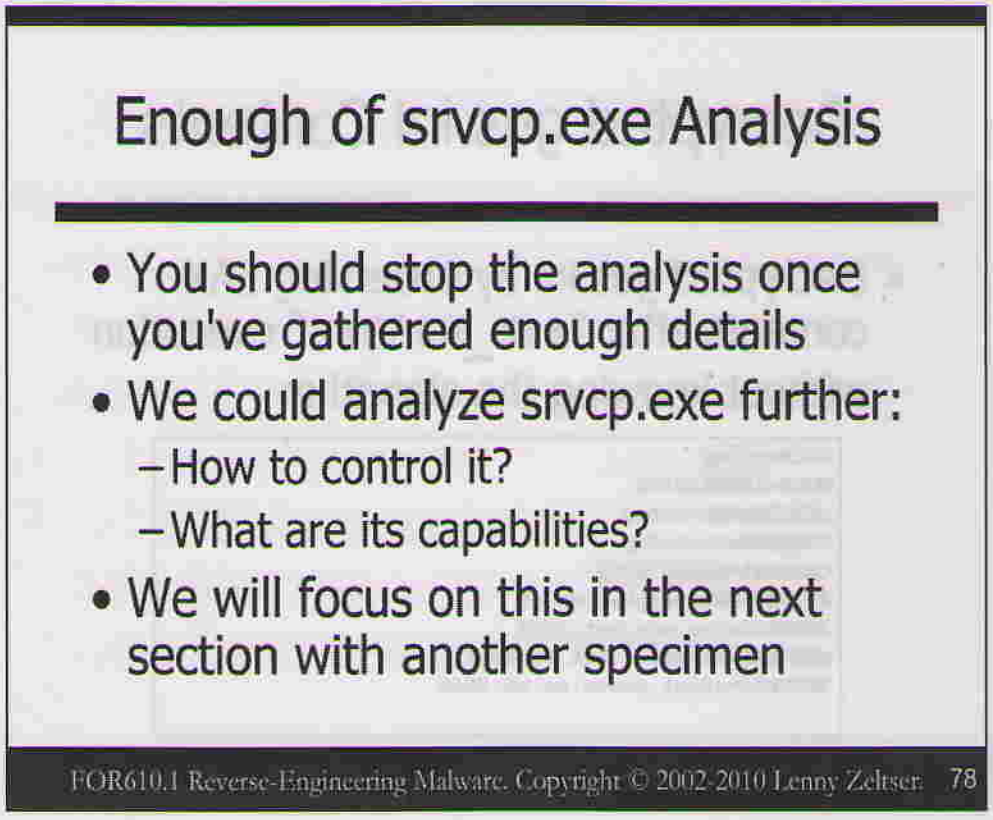
```
.text:00405401    mov     eax, edi
.text:00405403    add     eax, [ebp+var_C]
.text:00405406    add     byte ptr [eax], 7
.text:00405409    inc     edi
.text:0040540A    mov     eax, [ebp+var_C]
.text:0040540D    lea     ecx, [eax]
.text:0040540F    or      eax, 0FFFFFFFFh
.text:00405412    inc     eax
.text:00405413    cmp     byte ptr [ecx+eax], 0
.text:00405417    jnz     short loc_405412
.text:00405419    cmp     edi, eax
.text:0040541B    jb      short loc_405401
```



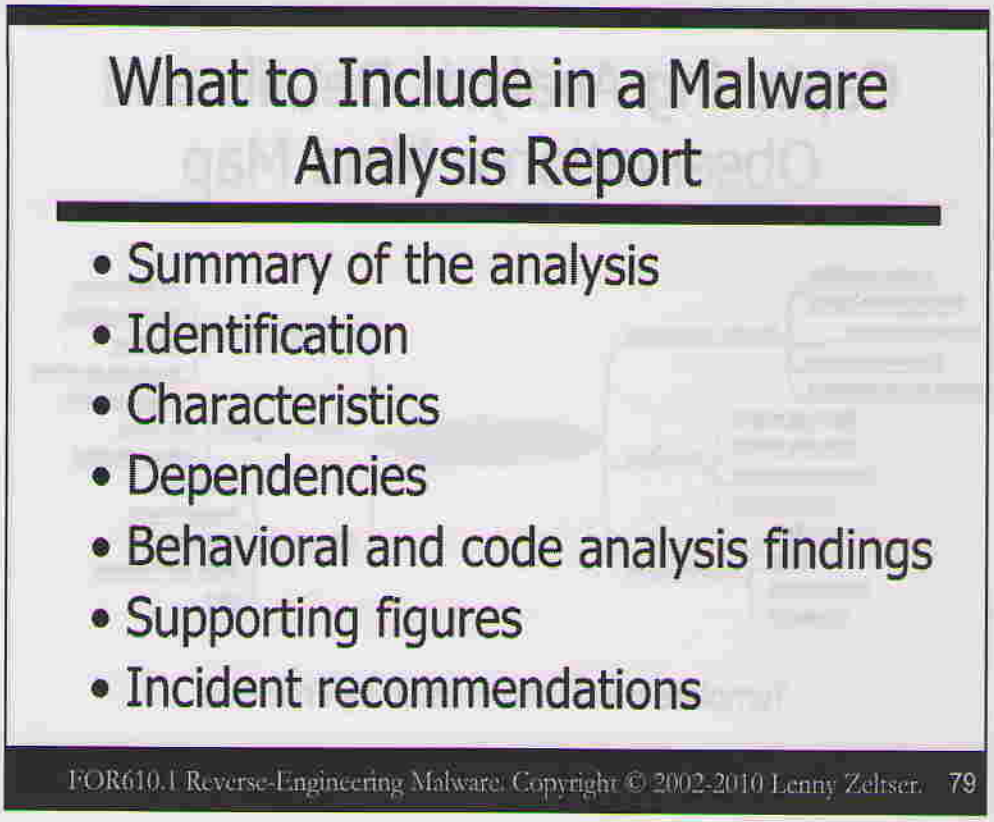
Even without understanding the decryption algorithm, we were able to obtain a deciphered copy of the gus.ini file. As mentioned earlier, we accomplished this by setting a breakpoint at the decryption routine, and looking at the returned value after the routine was executed. This is a very powerful technique, since it lets the trojan do all the hard work for you without forcing you to actively decrypt the file yourself!

This slide shows a portion of the clear-text version of the gus.ini file, which we obtained by peeking at the runtime environment of the srvcp.exe trojan. I abridged the file's contents for brevity; the file continues in a similar fashion and defined a total of thirty-four IRC servers. (You can download a complete copy of the decrypted file from http://zeltser.com/reverse-malware-paper/gus.decrypted.txt.)

I highlighted portions of the file that should look familiar to you from earlier slides. As we suspected, the gus.ini file overwrites default channel name/key values used when connecting to the IRC server, and provides a list of IRC servers/ports that the trojan should use instead. The file also defines the nickname to use in IRC sessions; in this case the overwriting value happened to match the string embedded into the srvcp.exe executable. I will talk briefly about other values defined in the file a little later.



There's more to discover about the capabilities of srvcp.exe, but I don't want to delve too deeply into this particular specimen. We used it to become familiar with the fundamental analysis tools and techniques. In the next section we will look at another specimen, and will become even more comfortable using the tools that I introduced in this section.



A typical malware analysis report covers the following areas:

Summary of the analysis: Key takeaways should the reader get from the report regarding the specimen's nature, origin, capabilities, and other relevant characteristics

Identification: The type of the file, its name, size, hashes (such as MD5, SHA1, and ssdeep), malware names (if known), current anti-virus detection capabilities

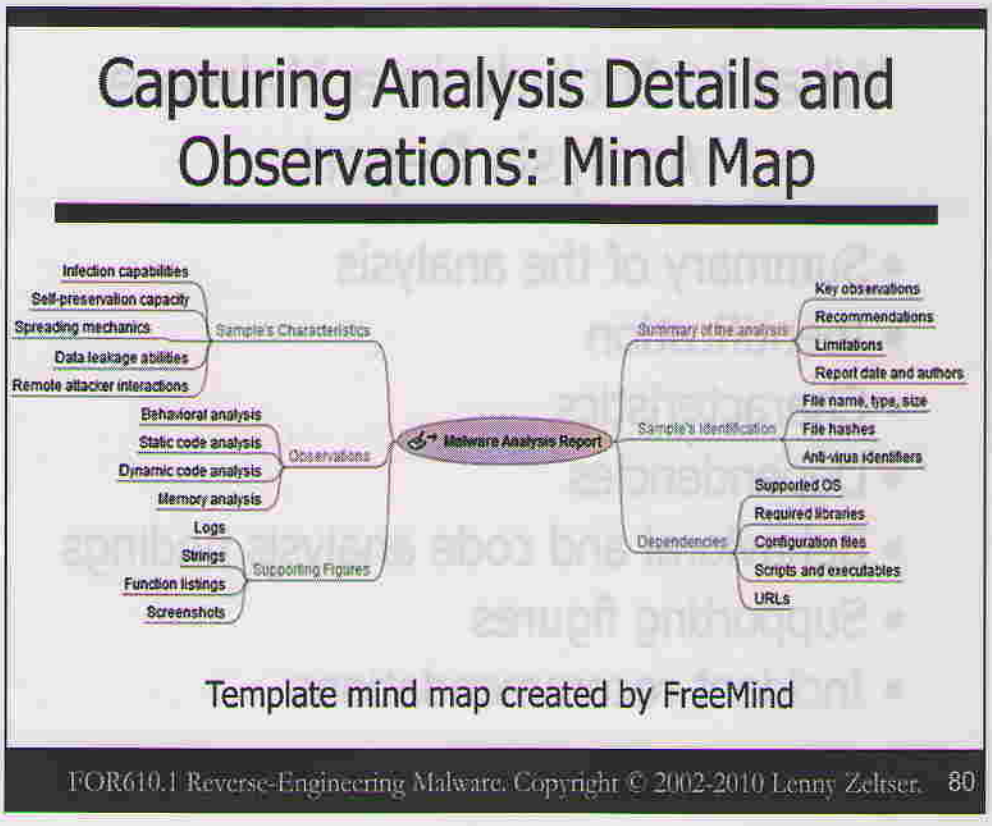
Characteristics: The specimen's capabilities for infecting files, self-preservation, spreading, leaking data, interacting with the attacker, and so on

Dependencies: Files and network resources related to the specimen's functionality, such as supported OS versions and required initialization files, custom DLLs, executables, URLs, and scripts

Behavioral and code analysis findings: Overview of the analyst's behavioral, as well as static and dynamic code analysis observations.

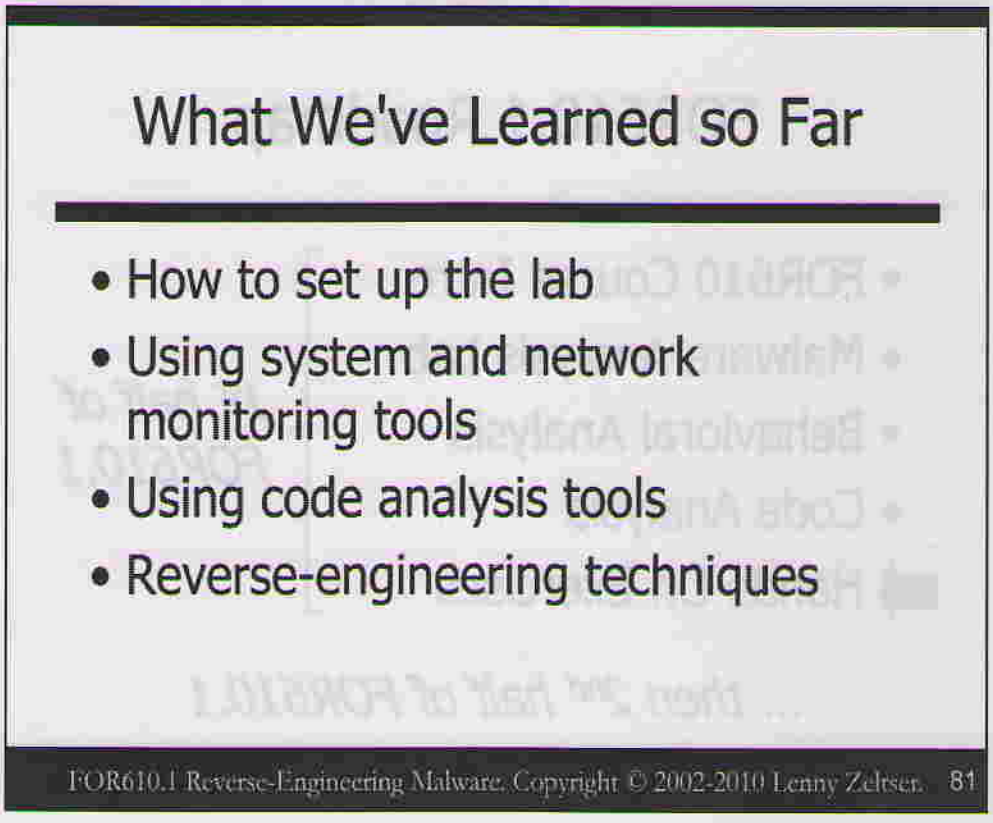
Supporting figures: Logs, screenshots, string excerpts, function listings, and other exhibits that support the investigators analysis

Incident recommendations: Indicators for detecting the specimen on other systems and networks (a.k.a. "indicators of compromise"), and possible for eradication steps



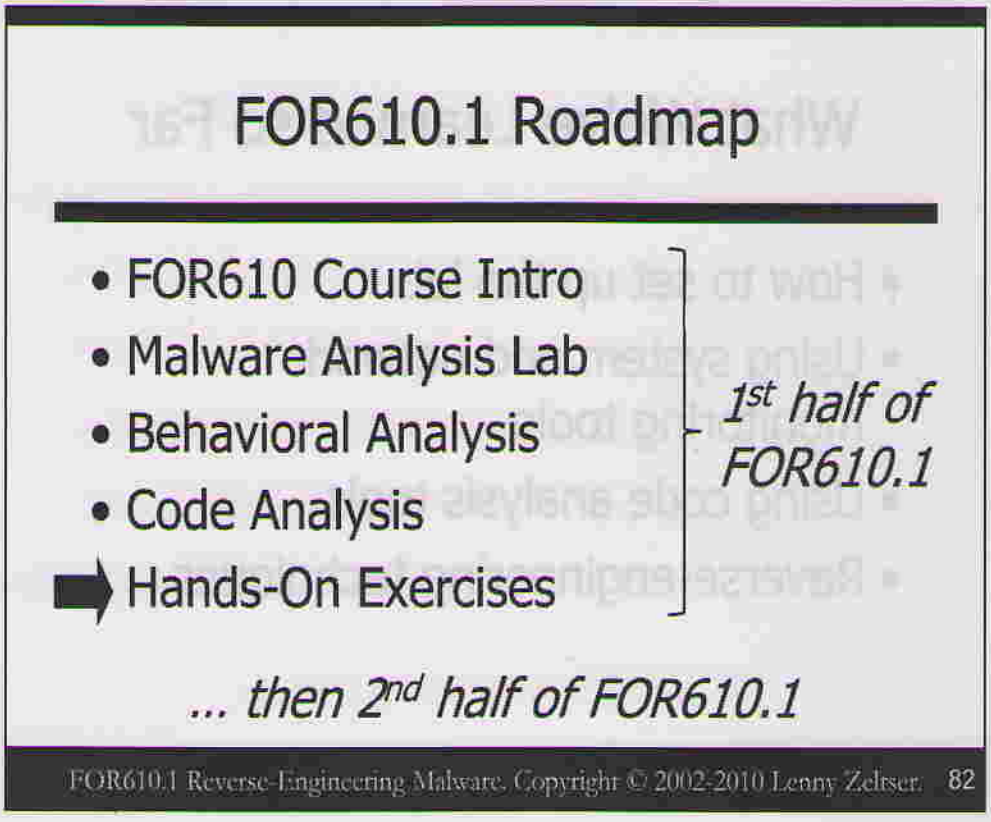
Malware analysis should be performed according to a repeatable process. To accomplish this, the analyst should save logs, take screen shots, and maintain notes during the examination. This data will allow the person to create an analysis report with sufficient detail that will allow a similarly-skilled analyst to arrive at equivalent results.

A convenient way of keeping track of your observations during the reverse-engineering process is to use a mind map, which organizes your notes, links, and screenshots on a single easy-to-see canvas. This slide pictures a mind map template for such a report. You will find it on your course DVD in the \ExtraTools directory. You need to use the free mind-mapping tool called FreeMind to load and edit the template (http://freemind.sourceforge.net/wiki/index.php/Main_Page).

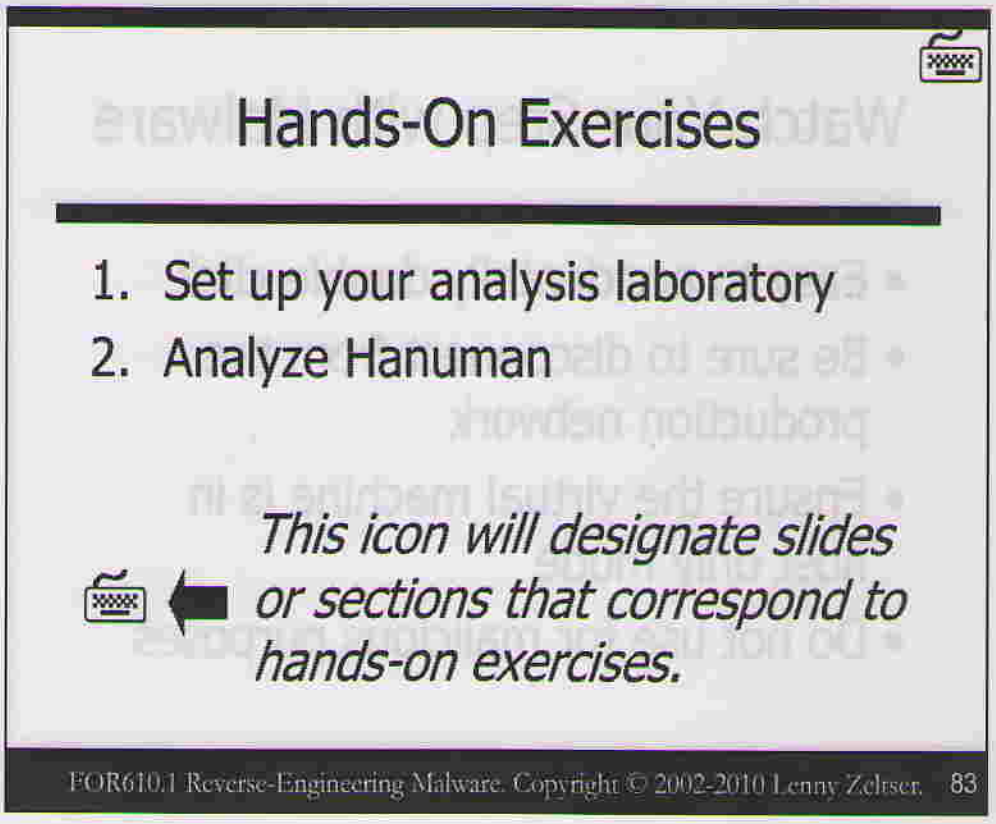


We've reached the end of the first half of FOR610.1. In this section I demonstrated the approach for reverse-engineering malware that, I believe, can be very effective to understanding capabilities of malicious software. We went over the laboratory set up, became familiar with several analysis tools, and went over an example that took advantage of these tools and procedures.

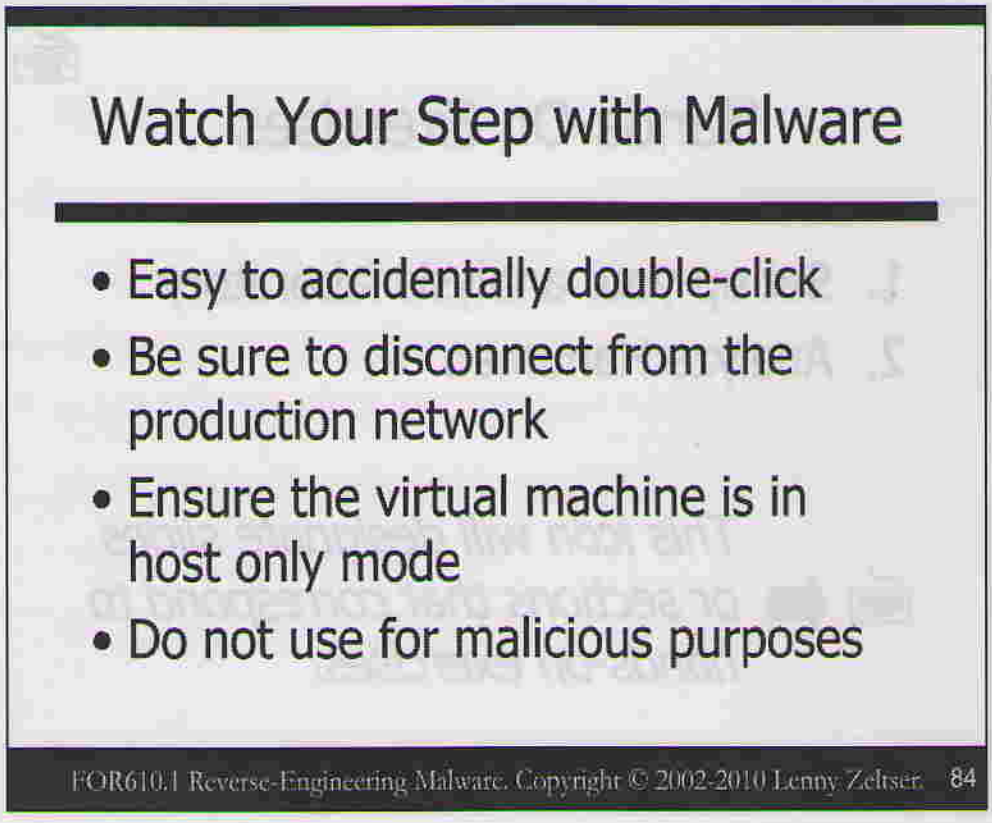
In the following slides you will find hands-on exercises that reinforce the concepts we've just learned.



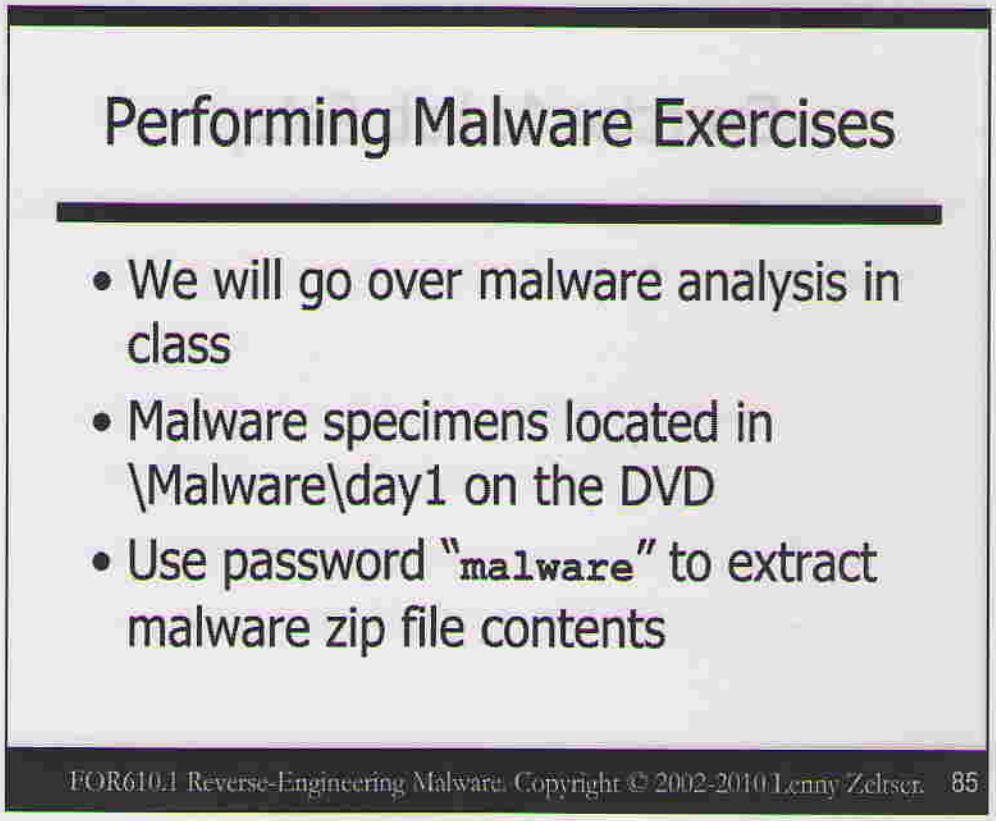
Now that we've covered some of the most essentials materials related to reverse-engineering malware, let's perform some hands-on exercises.



Please complete these exercises before starting the next section of this course. These exercises are described in greater detail in subsequent slides.

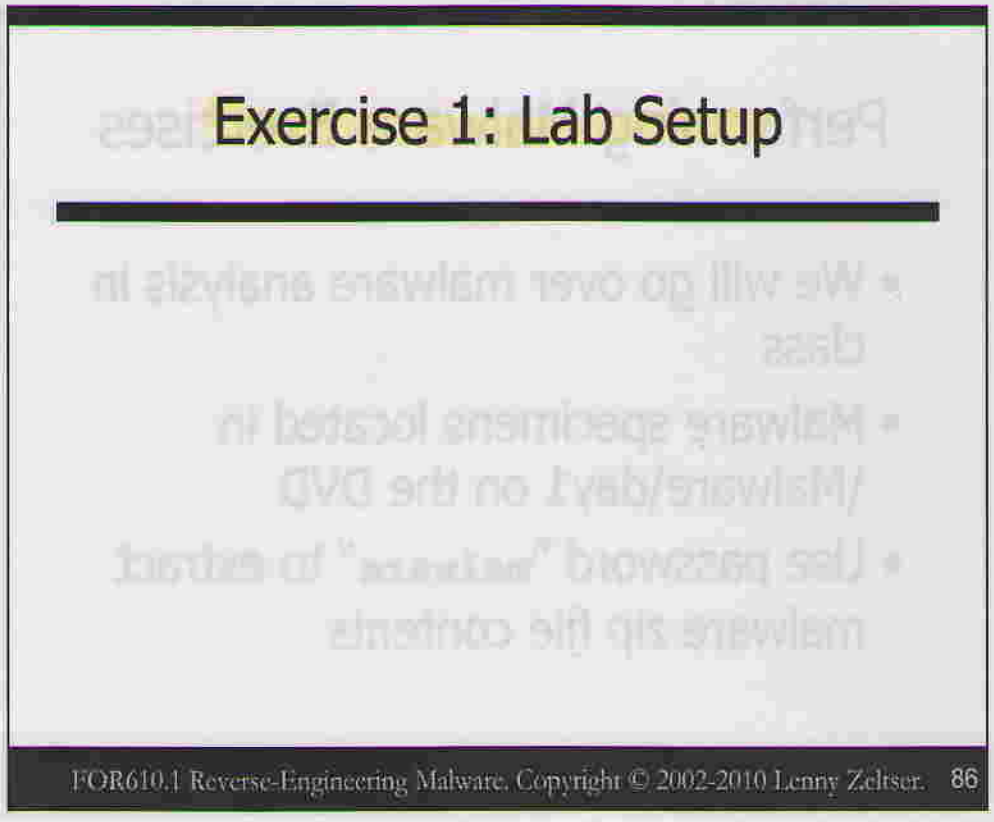


Hands-on exercises for this course involve real-world malware code that is dangerous and needs to be handled with care. Please follow isolation guidelines and common sense precautions to ensure that you do not accidentally impact your production environment.

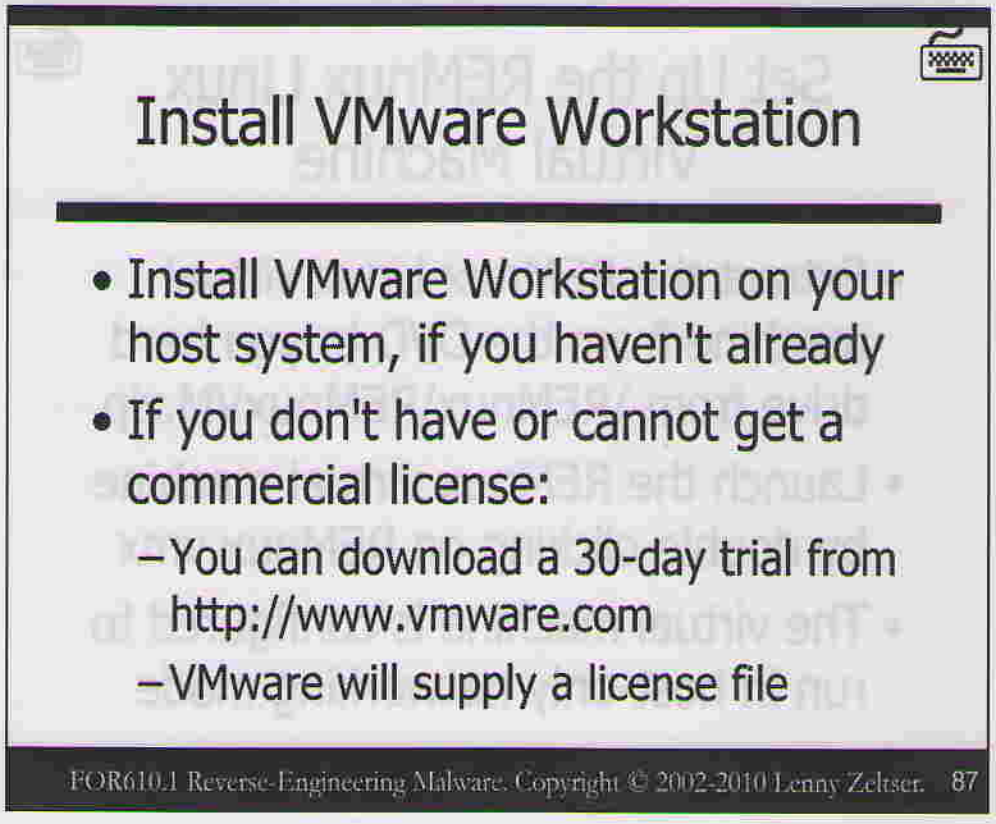


We will go over the solutions to these exercises in class.

The malware specimens used for these exercises are located in the \Malware\day1 directory on the DVD you received for this course. Each specimen is in a dedicated zip file that is protected with the password "`malware`" to help prevent accidental execution and anti-virus detection.

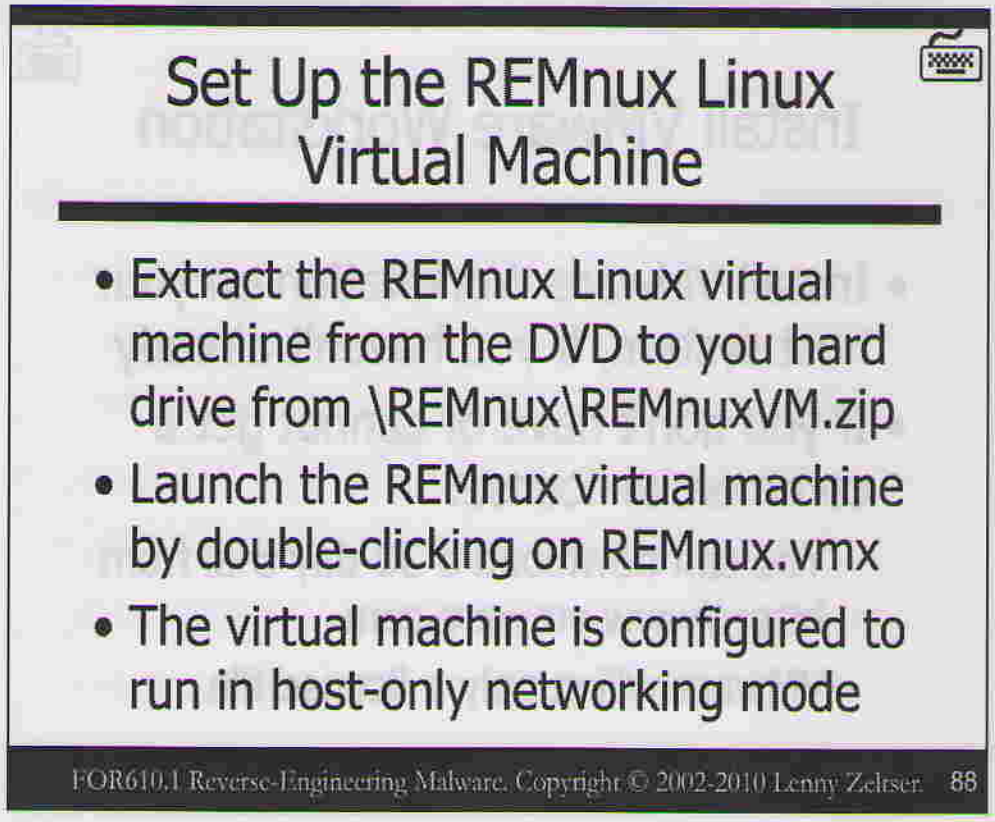


This exercise asks you to set up the isolated laboratory environment that you need to perform the other hands-on exercises.



We will rely heavily on Vmware Workstation in this course. If you are taking this class at a training event, we asked you to install it in advance. If you're taking this class under other circumstances, please install it now. If you don't own VMware, you can download a trial copy of the software for free from http://www.vmware.com. The trial license expires after 30 days; the evaluation period begins from the date when you request the license from VMware.

Please follow VMware Workstation installation instructions, available at the VMware website, to ensure that VMware is properly installed.



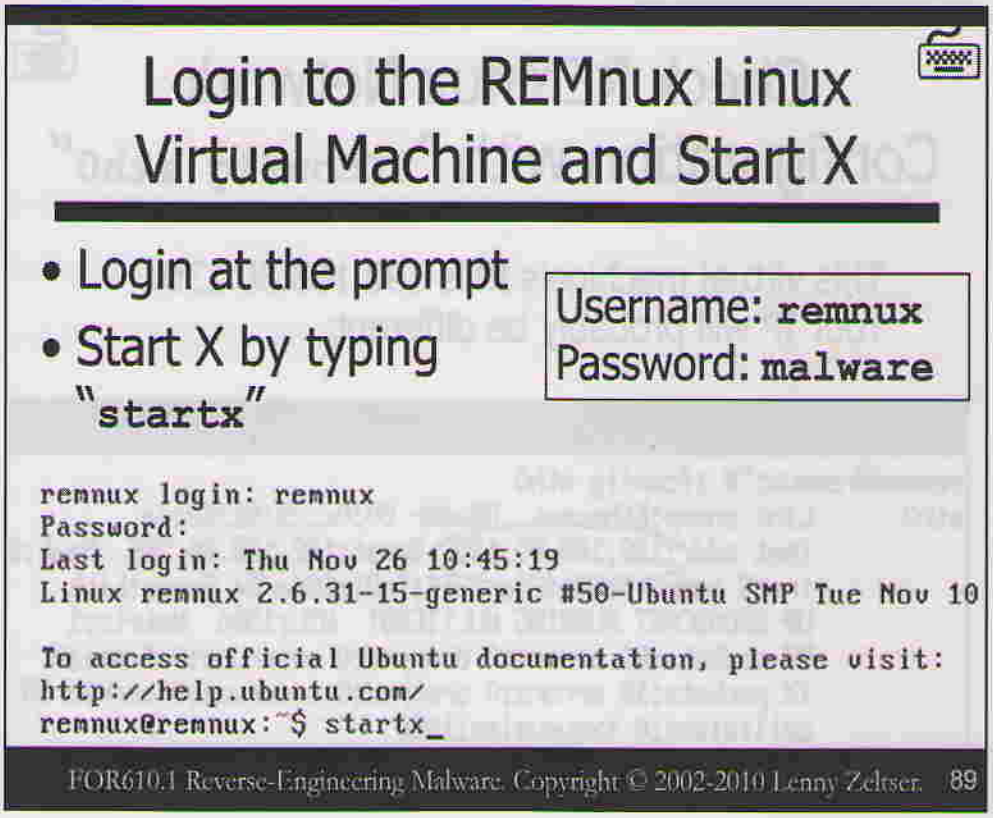
Next, set up the Linux virtual machine that you will use for hands-on exercises. You will use this system to monitor the laboratory network, to provide network services that malicious software may attempt reaching, as well as to examine malware using Linux-based tools.

The DVD that you received for class contains a Linux VMware machine built for reverse-engineering malware. It's called REMnux, and is based on the Ubuntu Linux distribution. All you need to do is extract the \REMnux\REMnuxVM.zip archive from the DVD to a hard drive on your physical host.

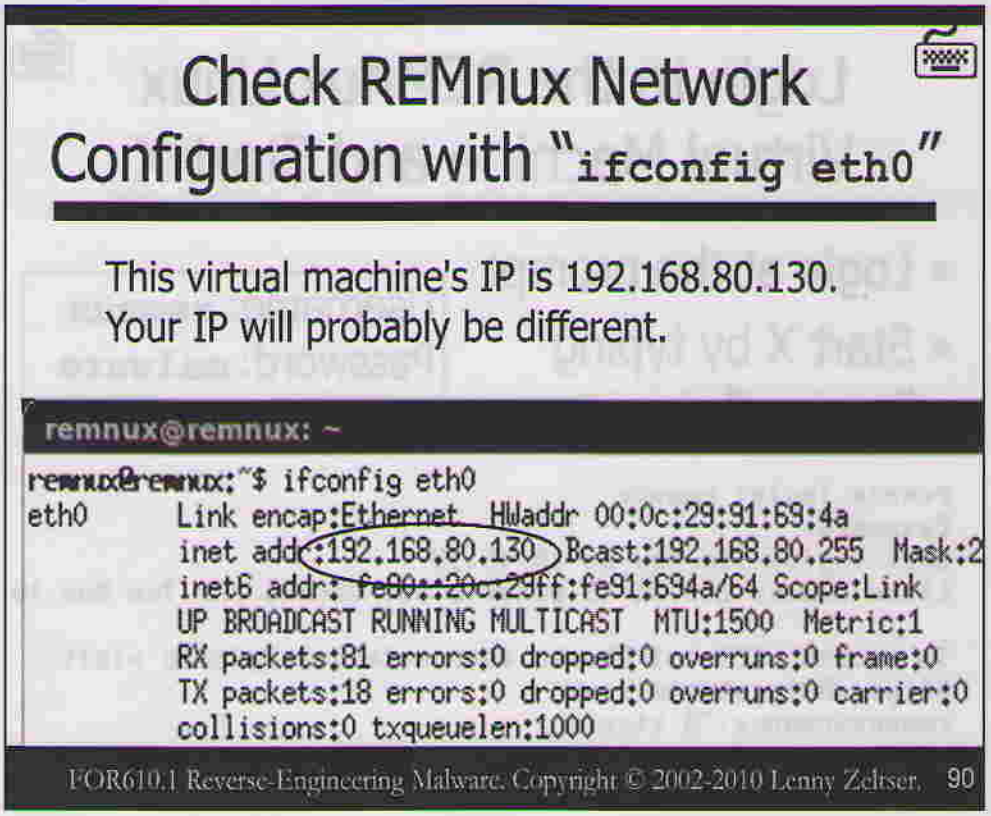
You can extract contents of LinuxVM.zip to the directory on your hard drive where you wish to store VMware virtual machines; extracted LinuxVM files should reside in a dedicated directory.

If your host OS doesn't already have an unzip utility on it, use the one located in the \Extra directory on the DVD.

To launch this virtual machine double-click on the REMnux.vmx that you extracted—among other files—from REMnuxVM.zip.

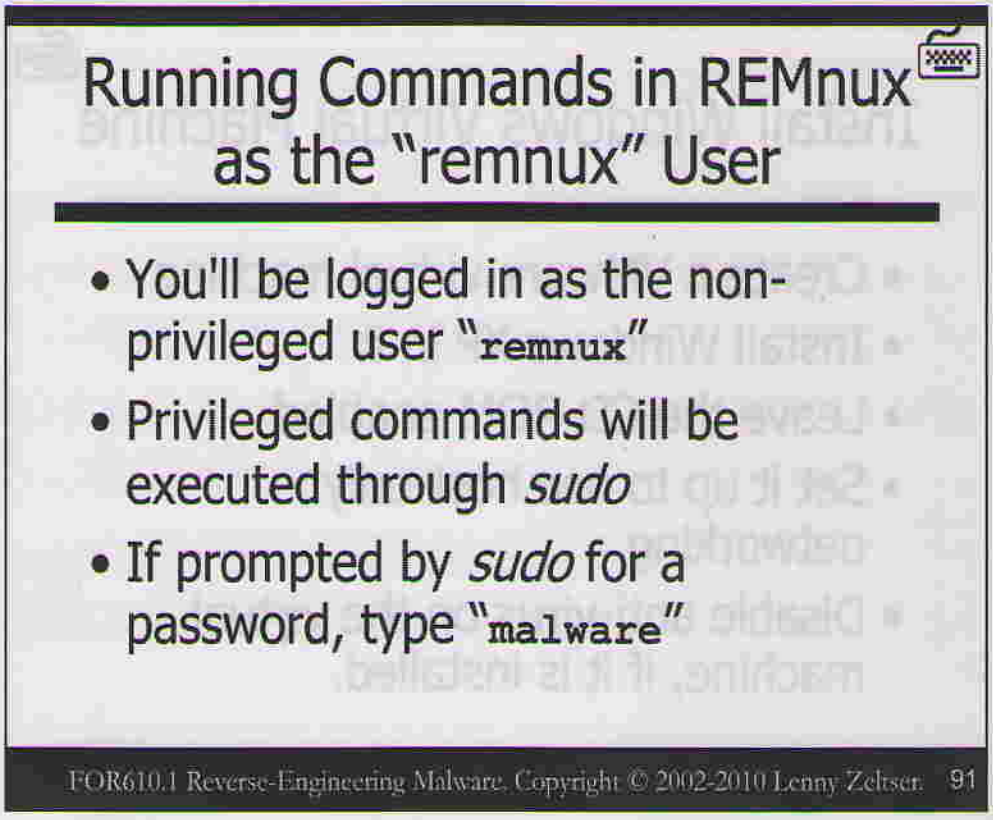


Boot the Linux VMware machine and login as user "`remnux`" with the password "`malware`". If you are going to use this virtual machine after this course, you should remember to later change the password using the *passwd* command. In this course, I recommend keeping the password "`malware`".



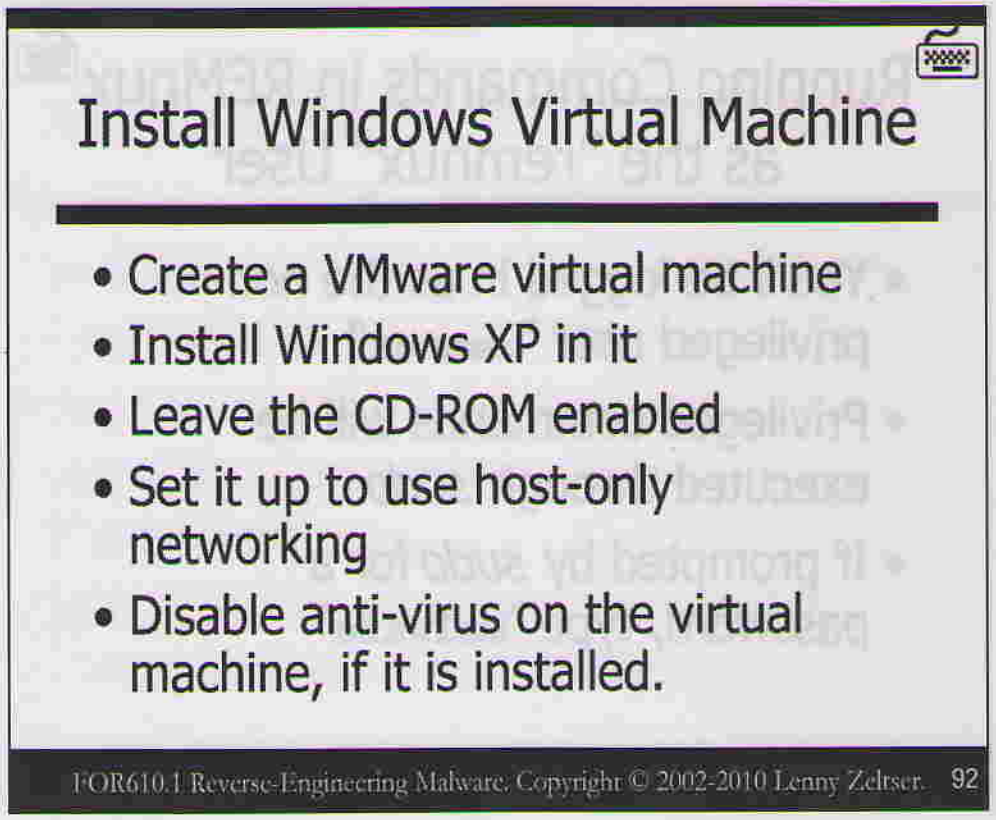
After logging in, run "`ifconfig eth0`" to make sure that the Linux system obtained an IP address from the VMware built-in DHCP service. The IP address should be listed following the "`inet addr`" suffix.

Write down the IP address; you'll need it later.



You'll usually be logged into REMnux using the non-privileged account of the user "`remnux`". To execute commands that require root privileges, you'll use the "**`sudo`**" command.

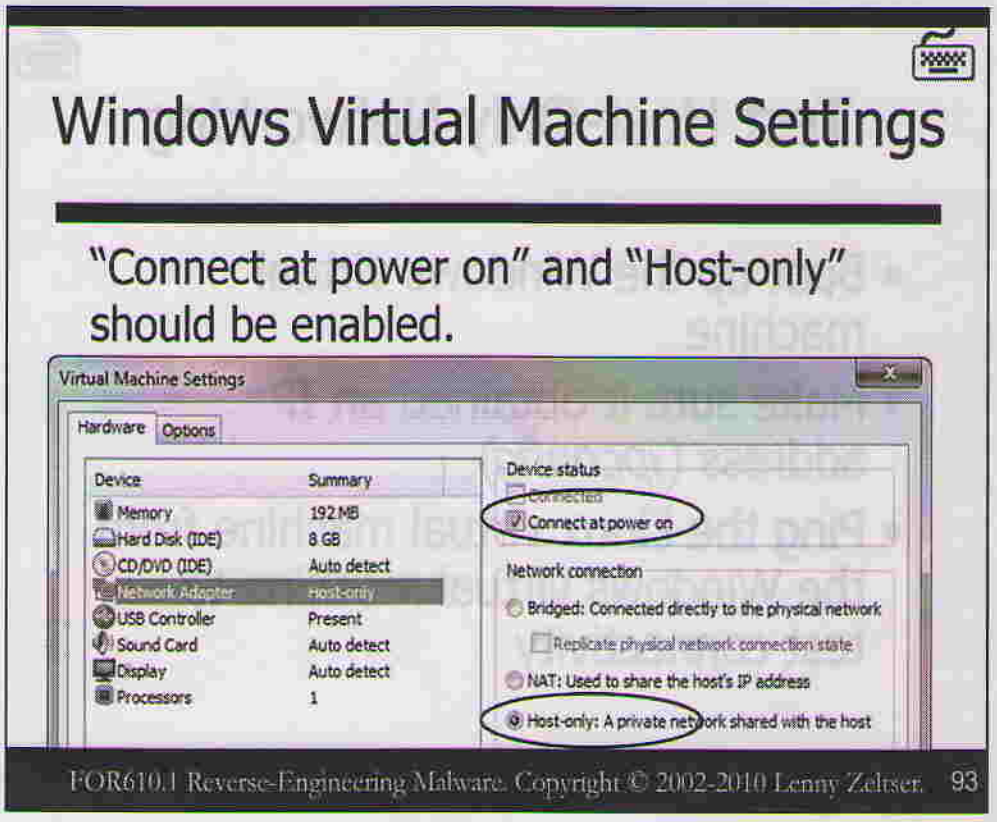
When running a privileged command using *sudo*, you may be prompted for a password. Remember that the password is "`malware`".



To perform the exercises, you will need to have a VMware virtual machine running Windows XP. If you are taking this class at a training event, we asked you to create this virtual machine in advance. If you're taking this class under other circumstances, please create it. You will need original Windows XP installation media to install the operating system on the virtual machine. (Patch Windows XP so that it's running at least Service Pack 2.) Please follow instructions available at the VMware website for installing Windows-based guest operating systems.

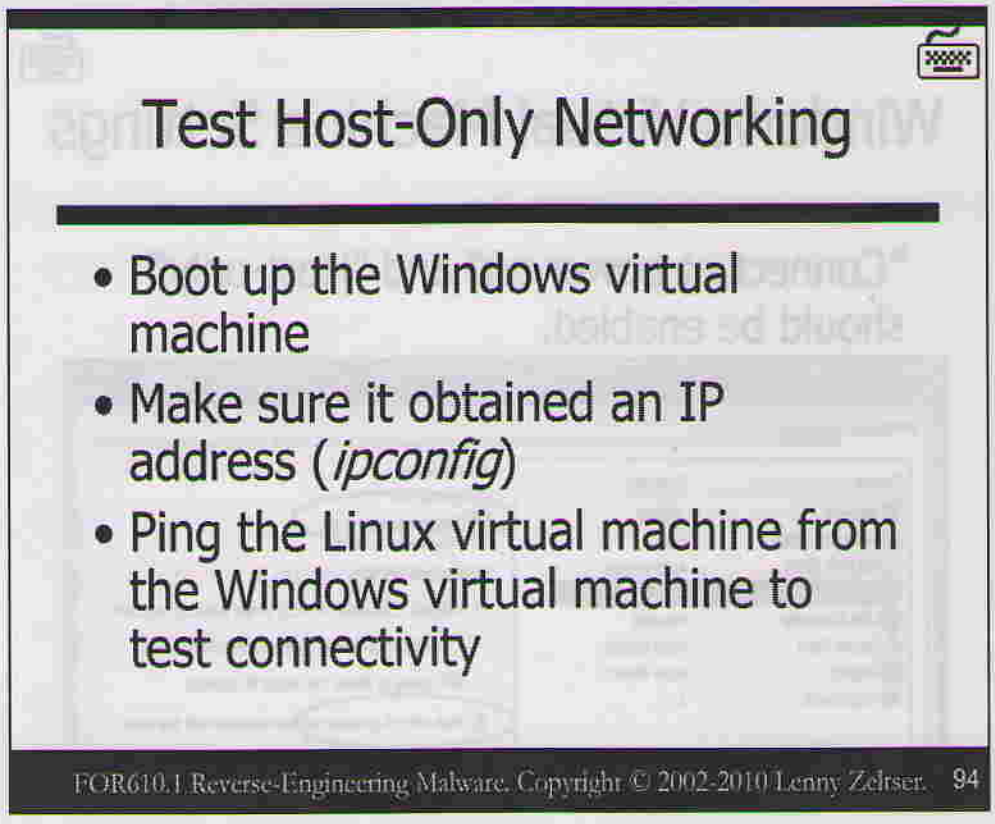
Configure properties of the Windows virtual machine to enable the host-only networking mode, as illustrated in the following slide. Also, please make sure that the CD-ROM is enabled; i.e., it should be set to use the physical drive in auto detect mode, and the "Connect at power on" checkbox should be enabled.

If anti-virus software is installed on your Windows virtual machine, please disable it, so it does not interfere with our exercises.



On your physical host, select your Windows virtual machine in VMware and go to the VM > Settings menu. Then click on "Network Adapter".

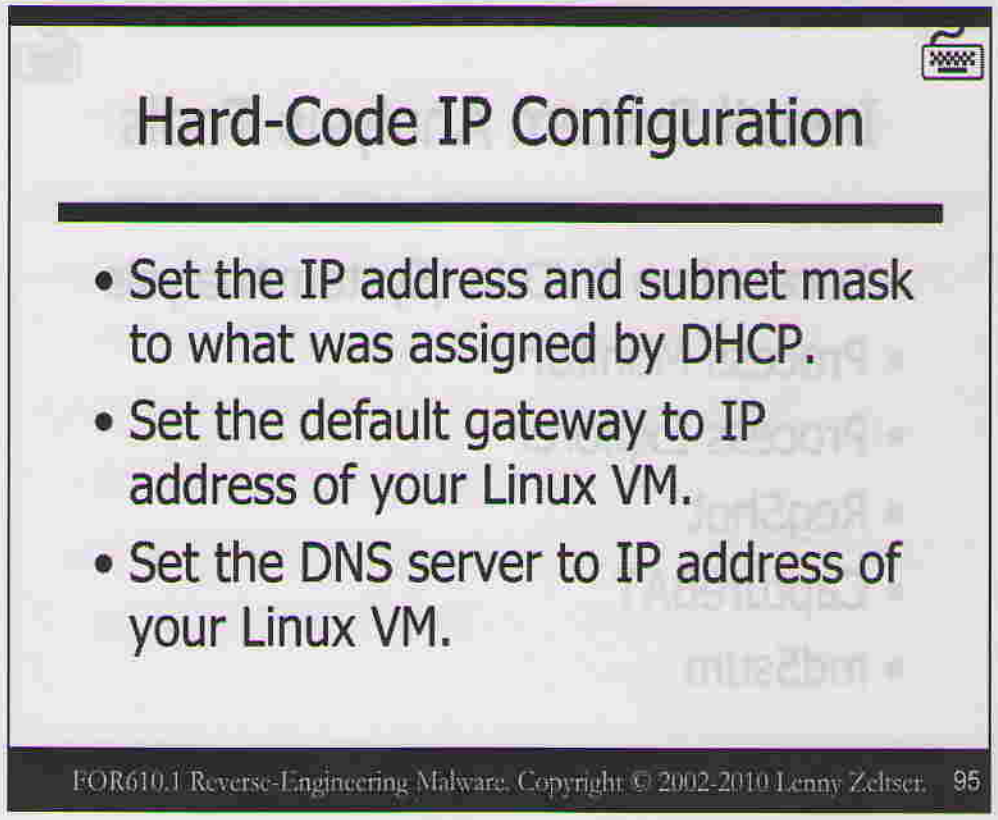
As illustrated on this slide, the Windows virtual machine should be configured to use host-only networking. This will help ensure isolation of your laboratory environment. Also, confirm that the "Connect at power on" option is enabled for the network adapter.



Once the Windows virtual machine has the OS installed, and settings properly set-up, it's time to boot into it and verify that it connected to the laboratory network. Login to the Windows system, and make sure that it has obtained an IP address—under VMware it will obtain its networking configuration from the built-in VMware DHCP service. The virtual system's IP address should be in a private address range that is different than the range associated with your machine's physical card – this is because we use the host-only mode in VMware to isolate the VMware virtual network.

Use the "`ipconfig /all`" command to check the IP address of the virtual machine and make sure it received an IP address.

Ping your Linux virtual machine from the Windows virtual machine to make sure the host-only network is functioning properly. If this works, there is no need to ping the Windows virtual machine from the Linux one; in fact, a personal firewall on your Windows virtual machine might block such connections, and that's OK for the exercises that await us.

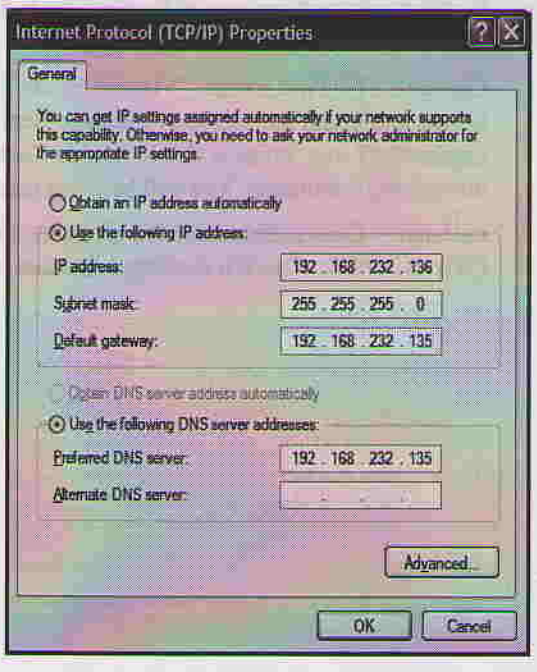


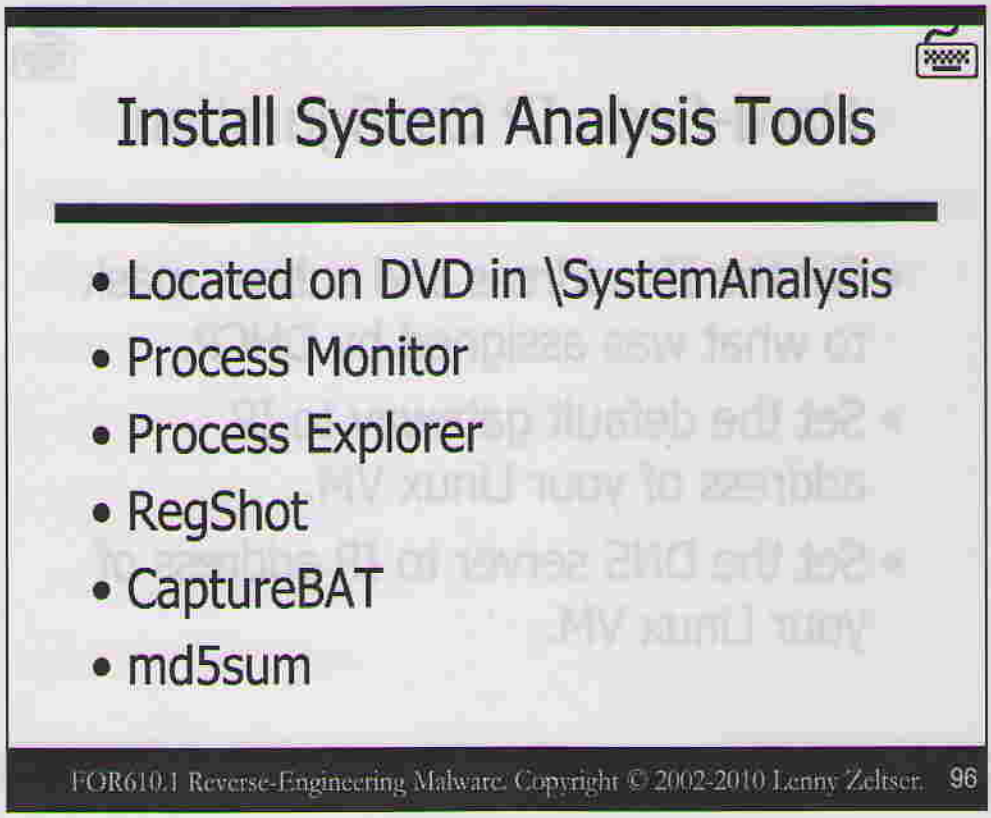
Go to the TCP/IP properties of the network interface of your Windows virtual machine via Control Panel > Network Connections > Local Area Connection > Internet Protocol (TCP/IP) > Properties. You will hard-code this information, rather than preserving the "Obtain IP address automatically" setting.

Set the "IP address" field to the UP address that was automatically assigned to your Windows virtual machine by the VMware DHCP server. Set the "Subnet mask" field to "255.255.255.0".

Set the "Default gateway" and "Preferred DNS server" fields to the IP address of your Linux virtual machines.

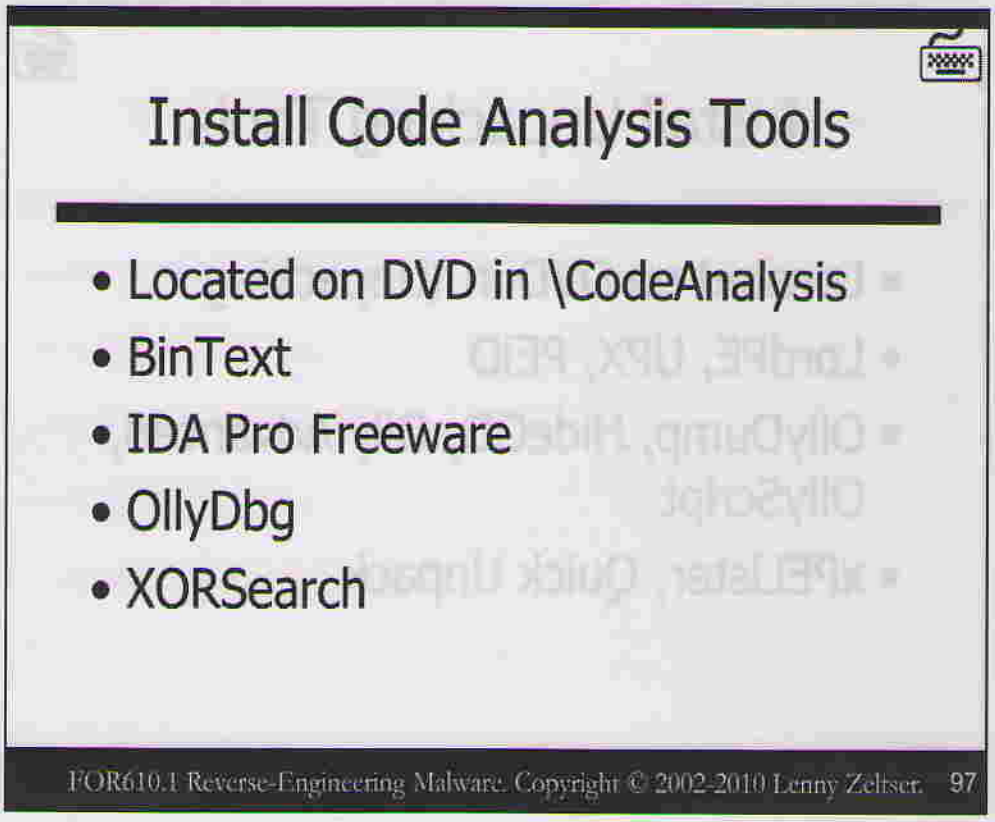
Your configuration should look similar to the screenshot on the right, although **your IP addresses will probably be different**.





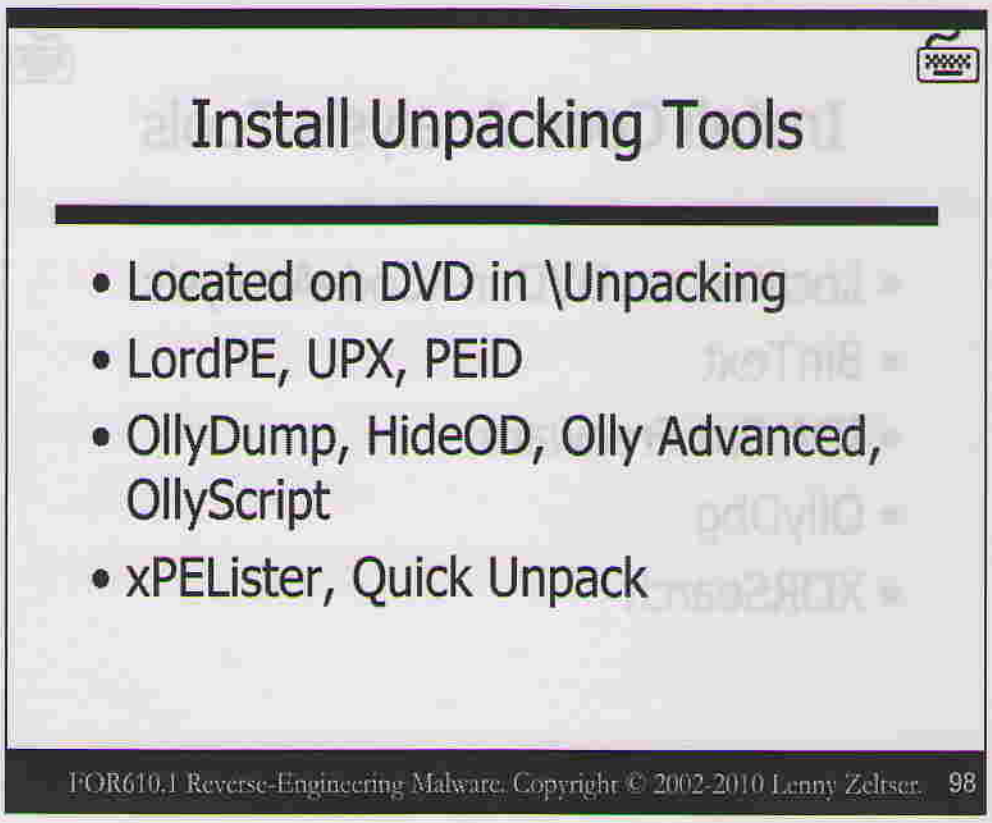
Install system monitoring tools to your Windows virtual machine. They are located in the \SystemAnalysis directory on the course DVD.

- Process Monitor – Extract contents of ProcessMonitor.zip to a dedicated directory (e.g., C:\Program Files\Process Monitor). Place a shortcut to procmon.exe on the Desktop.
- Process Explorer – Extract contents of ProcessExplorer.zip to a dedicated directory (e.g., C:\Program Files\Process Explorer). Place a shortcut to procexp.exe on the Desktop.
- RegShot – Extract contents of RegShot.zip to a dedicated directory (e.g., C:\Program Files\RegShot). Place a shortcut to regshot.exe on the Desktop).
- CaptureBAT – First, extract "Visual C++ 2005 Redistrib Package.exe" from CaptureBAT.zip; launch it to install the necessary libraries. Next, extract and launch the setup file CaptureBAT.exe, which is also located in CaptureBAT.zip; the setup wizard will walk you through the installation. You will be asked (and will need to) reboot after the setup finishes.
- md5sum – Copy md5sum.exe to a directory somewhere in the system's path (e.g., C:\WINDOWS). This is a command-line tool, so no need to create a shortcut to it.



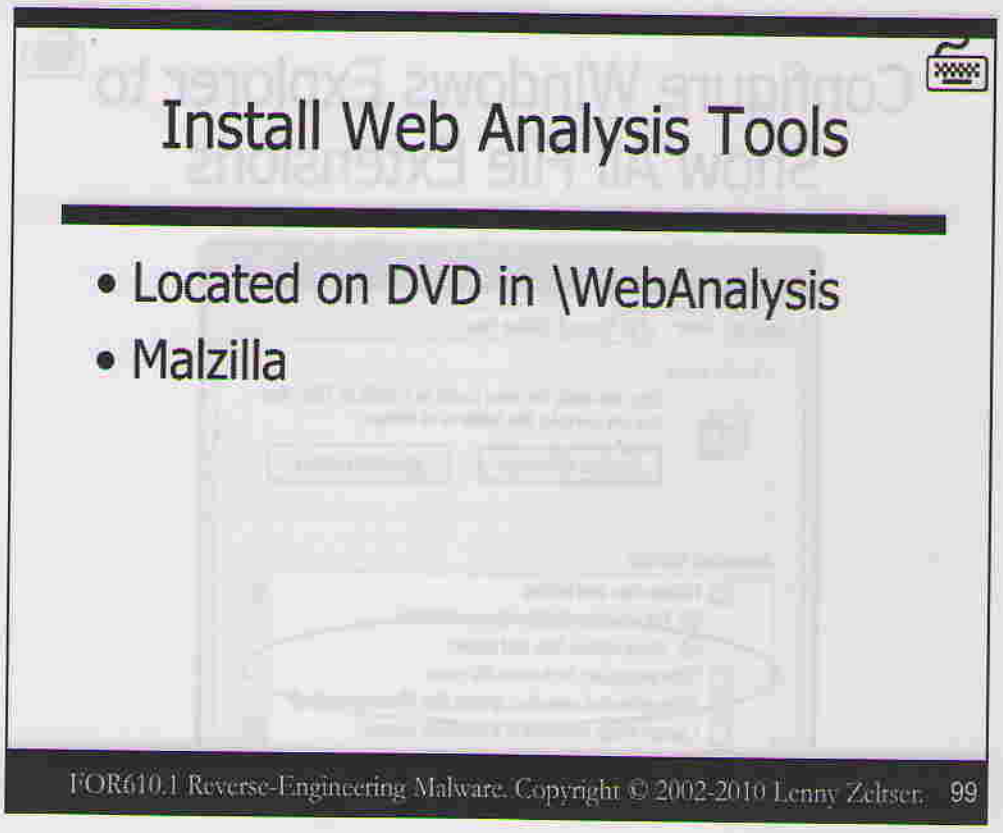
Install code analysis tools to your Windows virtual machine. They are located in the \CodeAnalysis directory on the course DVD.

- BinText – Extract contents of BinText.zip to a dedicated directory (e.g., C:\Program Files\BinText). Create a shortcut to bintext.exe and place it on the Desktop.
- IDA Pro Freeware – Launch the tool's setup program by double-clicking on IDA-Freeware.exe, then step through the installation wizard. Create a shortcut to idag.exe and place it on the Desktop.
- OllyDbg – Extract contents of OllyDbg.zip to a dedicated directory (e.g., C:\Program Files\OllyDbg). Create a shortcut to ollydbg.exe and place it on the Desktop.
- XORsearch – Extract contents of XORSearch.zip to a directory somewhere in the system's path (e.g., C:\WINDOWS). This is a command-line tool, so no need to create a shortcut to it.



Install unpacking tools to your Windows virtual machine. They are located in the \Unpacking directory on the course DVD.

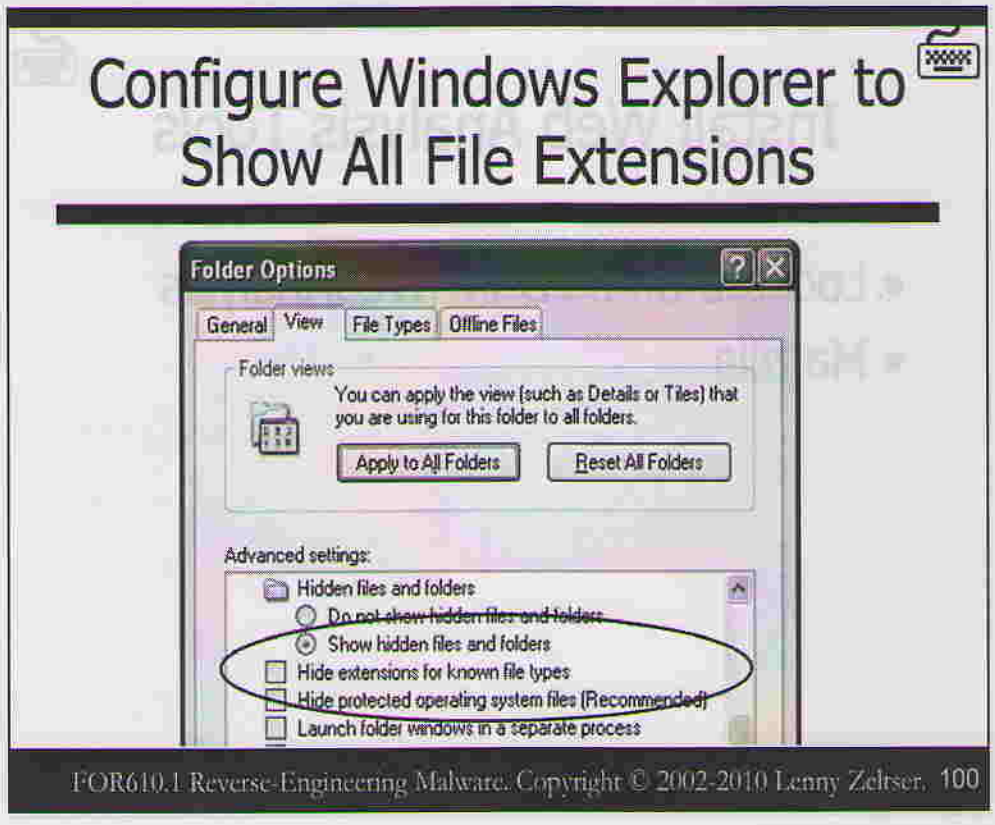
- LordPE – Extract contents of LordPE.zip to a dedicated directory (e.g., C:\Program Files\LordPE). Create a shortcut to LordPE.exe on the Desktop.
- UPX – Extract contents of UPX.zip to a directory somewhere in your system's path (e.g., C:\WINDOWS). This is a command-line tool, so no need to create a shortcut to it.
- PEiD – Extract contents of PEiD.zip to a dedicated directory (e.g., C:\Program Files\PEiD.) Create a shortcut to PEiD.exe on the Desktop.
- OllyDump, HideOD, Olly Advanced, OllyScript – Extract contents of OllyPlugins.zip to the directory where you installed OllyDbg.
- xPELister – Extract contents of xPELister.zip to a dedicated directory (e.g., C:\Program Files\xPELister). Create a shortcut to xPELister and place it on the Desktop. Contents of the zip file are protected with the password "malware" to prevent accidental detection by anti-virus products.
- Quick Unpack – Extract contents of QuickUnpack.zip to a dedicated directory. We suggest using C:\Program Files\QuickUnpack. Create a shortcut to Quick Unpack and place it on the Desktop.



Install web analysis tools to your Windows virtual machine. They are located in the \WebAnalysis directory on the course DVD.

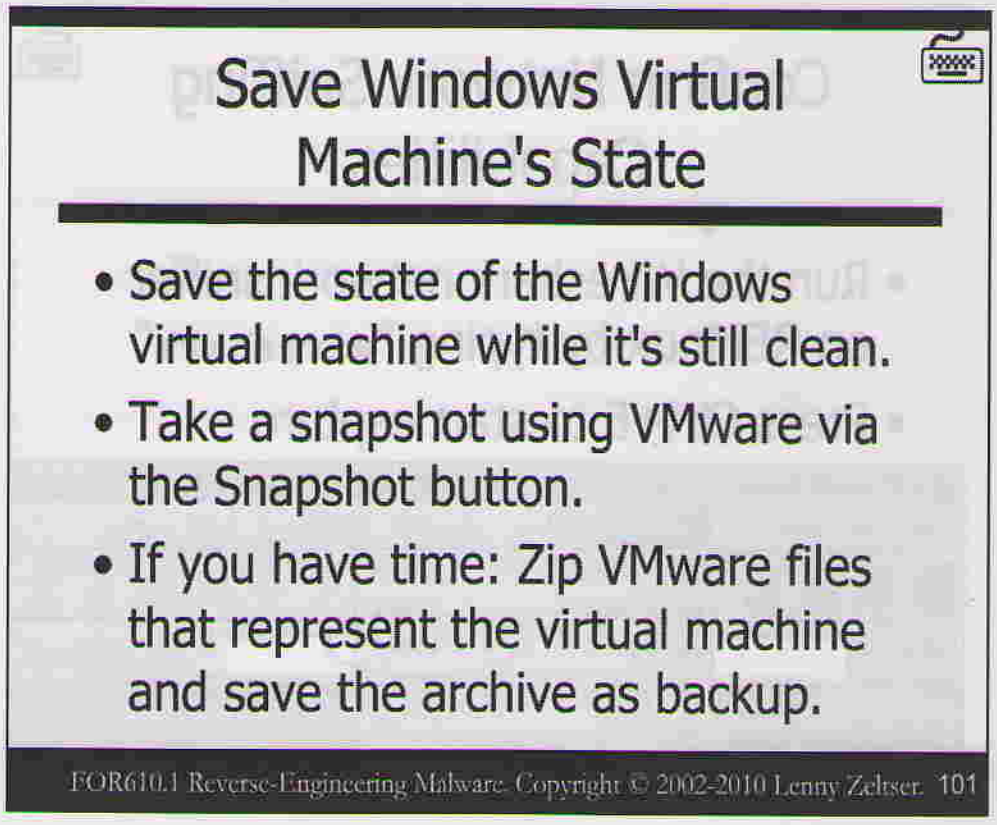
- Malzilla – Extract contents of Malzilla.zip to a dedicated directory (e.g., C:\Program Files\Malzilla). Create a shortcut to malzilla.exe place it on the Desktop.

There's just one tool we need to install on Windows, because the other tools we'll use are already installed on the REMnux Linux distribution.



Sometimes malware will try to fool us by changing its file's extension, but by setting the file's attribute to conceal it. Modify Windows Explorer's default settings to make sure it shows you all hidden and OS files, as well as full file extensions.

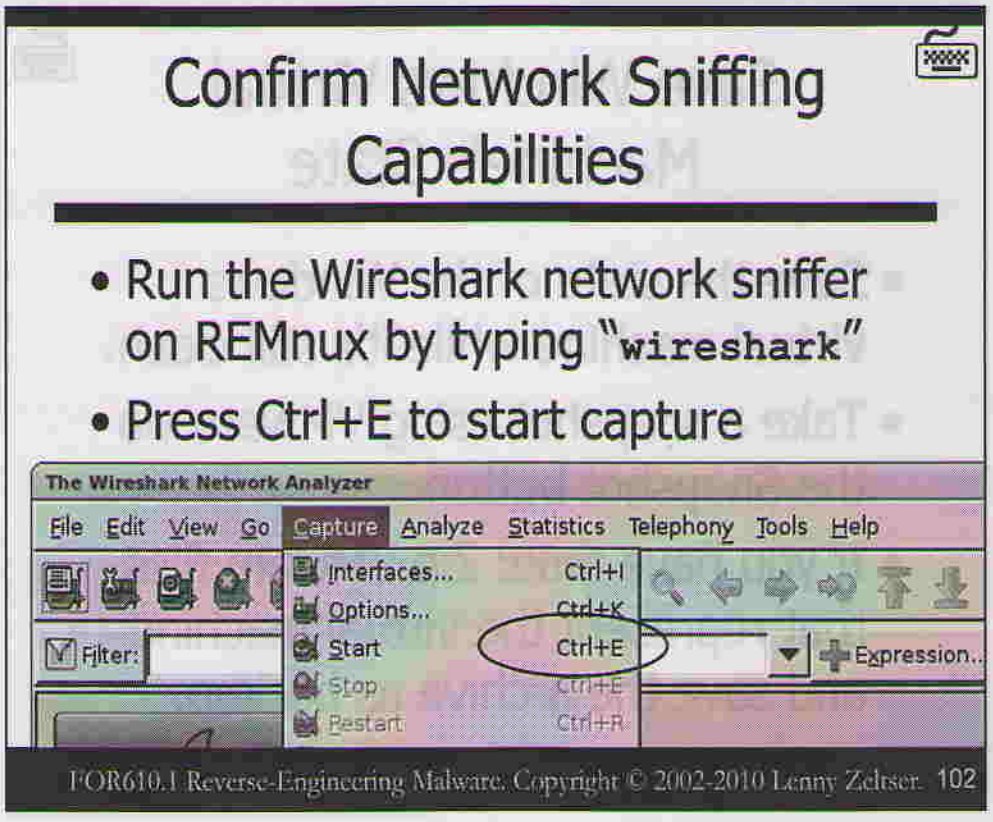
To do this, launch Windows Explorer (for instance, by opening on My Computer). Then go to Tools > Options. Click the View tab, and click on "Show hidden files and folders". Also, uncheck "Hide extensions for known file types" and "Hide protected operating system files".



Once the Windows virtual machine is properly set up, please shut it down. You should now save its state, while it is not yet infected by malware.

A convenient way to maintain the ability to revert a virtual machine to a pristine state involves using the snapshot functionality built into VMware. Simply press the Snapshot button in VMware, and it will remember the virtual machine's current state. Later on, if you ever need to go back to this state, just press the Revert button and VMware will restore the virtual machine it its pristine state.

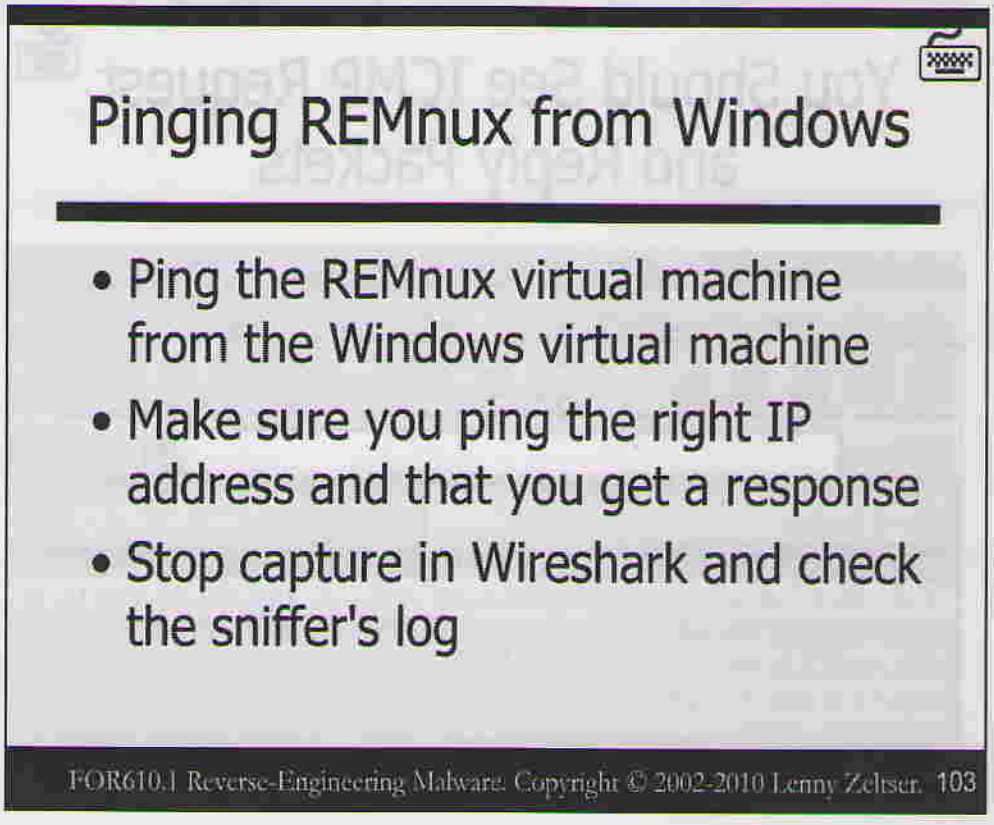
VMware represents each virtual machine using a set of files that are located in a dedicated directory—one directory per virtual machine. You can also save those files to back-up the virtual machine in case you accidentally remove or misconfigure the VMware snapshot. Archiving the virtual machine in this manner take disk space and time, so only do this if you have enough disk space and about 15 minutes to spare. To do this, locate the directory that contains files of your Windows virtual machine and compress its contents into a zip archive. If you have lots of space and don't want to bother with compression, simply make a copy of this directory without zipping its contents. If you ever need to restore the Windows virtual machine to a pristine state, simply delete the directory that represents the infected virtual machine, and put in its place the archive you created.



Now that both the Windows and the REMnux VMware machines are set up, run a sniffer on the Linux system. The network sniffer that we'll be using throughout the course, and which is already installed on REMnux, is Wireshark. To run it, simply type "`wireshark`" at the XTerm prompt in X Window System and press Enter. You may be prompted for the root password. (Remember, the password is "`malware`".)

To begin capturing traffic in Wireshark, press Ctrl+E or go to the Capture > Start menu. This will start the capture and bring up the window that will show you, in real time, any packets Wireshark will see on the laboratory network.

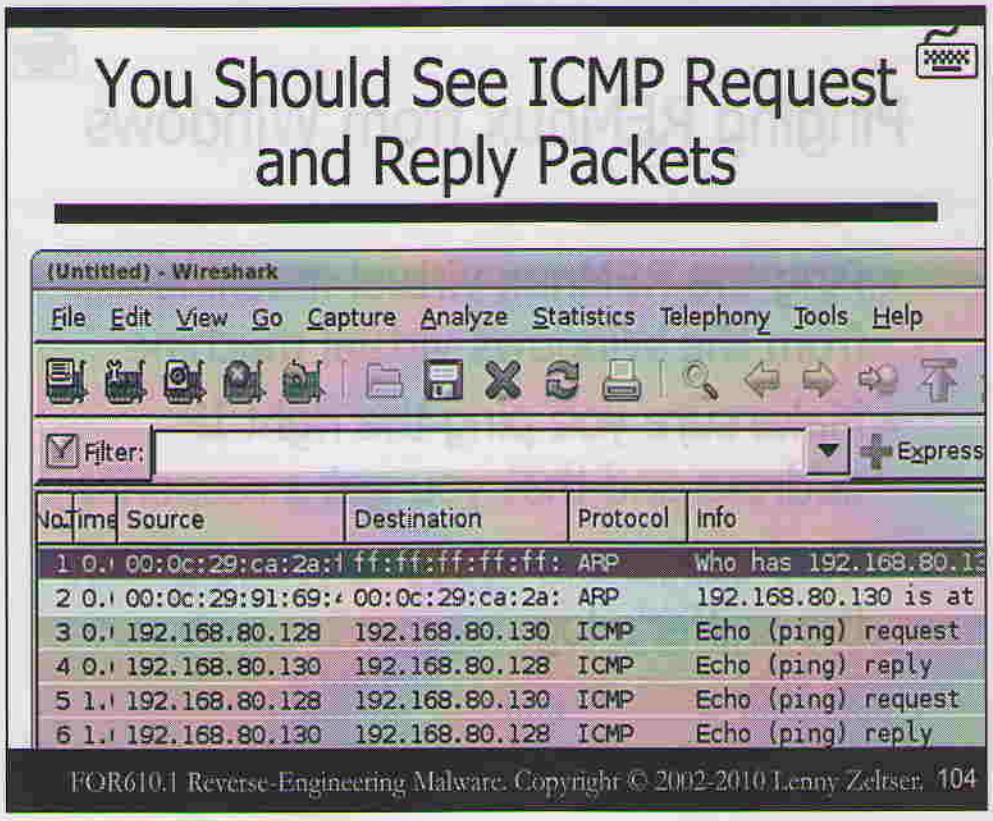
Note that you need be in the graphical X environment before you can launch Wireshark. If you booted into REMnux and are still in the console window, type "`startx`" to launch X.



To make sure you are able to capture traffic on the virtual network, ping the REMnux virtual machine from the Windows virtual machine while the sniffer is running. As shown on the next slide, sniffer logs should display ICMP echo request and echo reply packets.

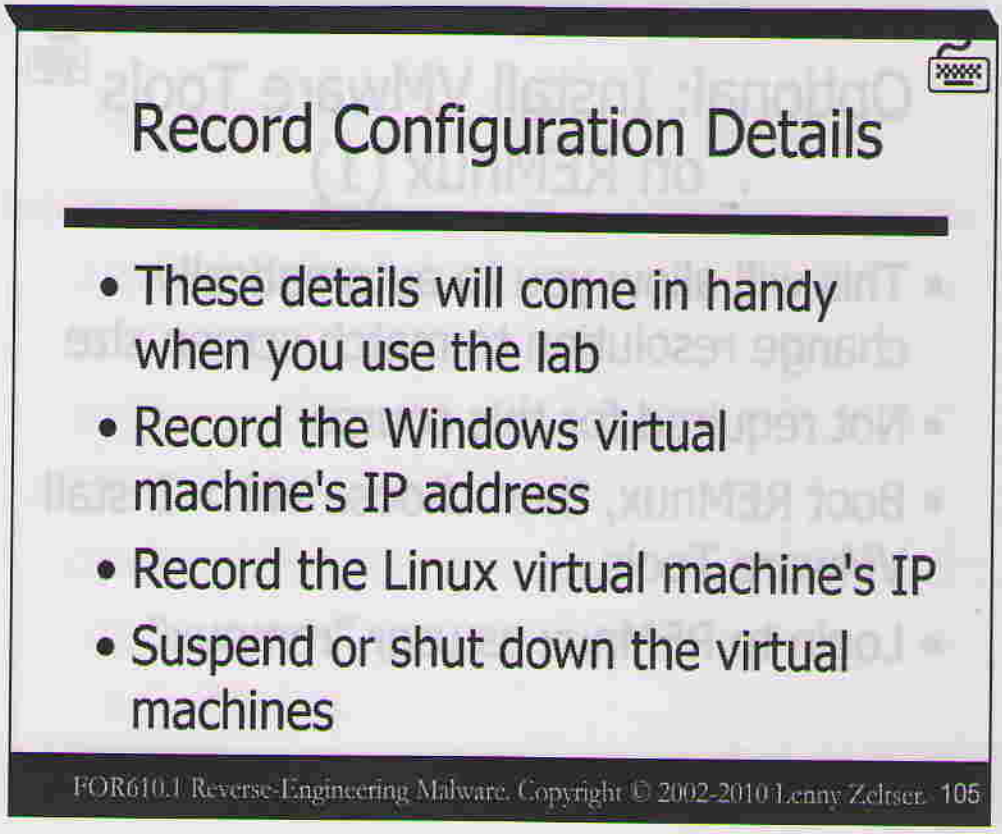
Remember that you should be pinging the IP address that your instance of VMware assigned to the REMnux virtual machine on your system.A common mistake is to ping the IP address of your Windows virtual machine, instead of the IP address of the REMnux virtual machine.

Once ping sent and received a few packet pairs, go to your REMnux virtual machine and press Ctrl+E to stop capture in Wireshark. You can also do this via the Capture > Stop menu option.



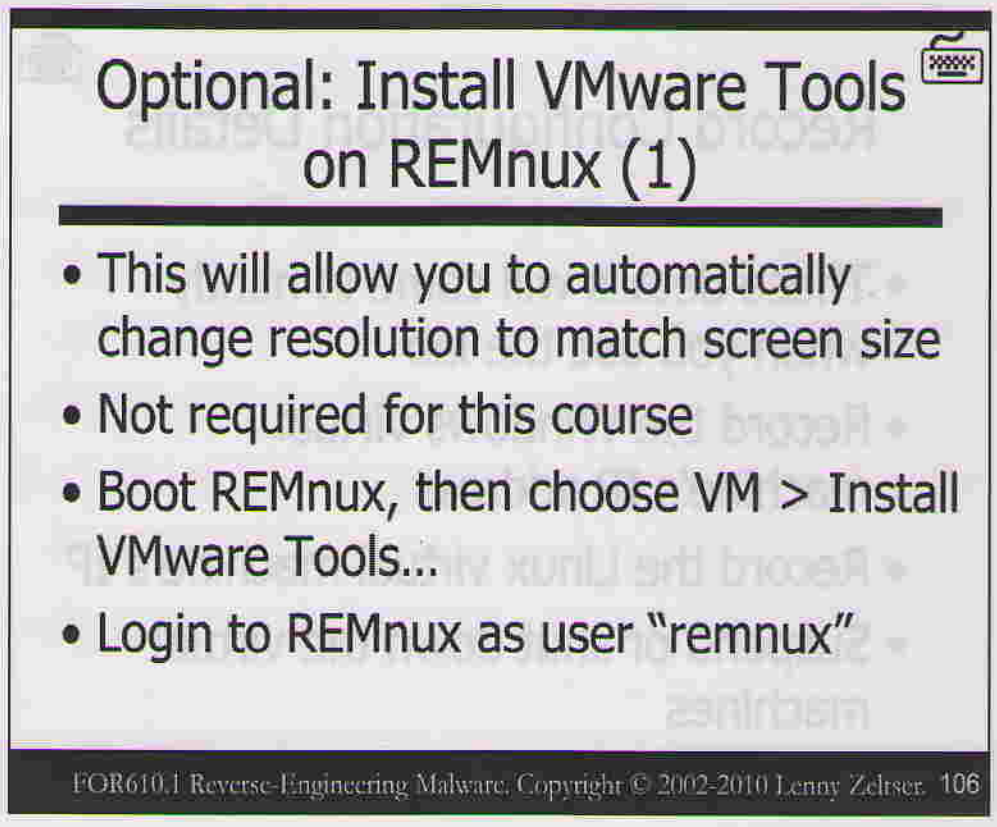
The sniffer's network capture should show several ICMP echo request and reply packets. If you're able to see this traffic, you've confirmed that the network sniffer is able to monitor the network to the extent that'll allow us to perform hands-on exercises in this course.

You can now exit Wireshark by pressing Ctrl+Q or by using the File > Quit menu. If Wireshark asks you whether to save capture file, select "Quit without Saving".



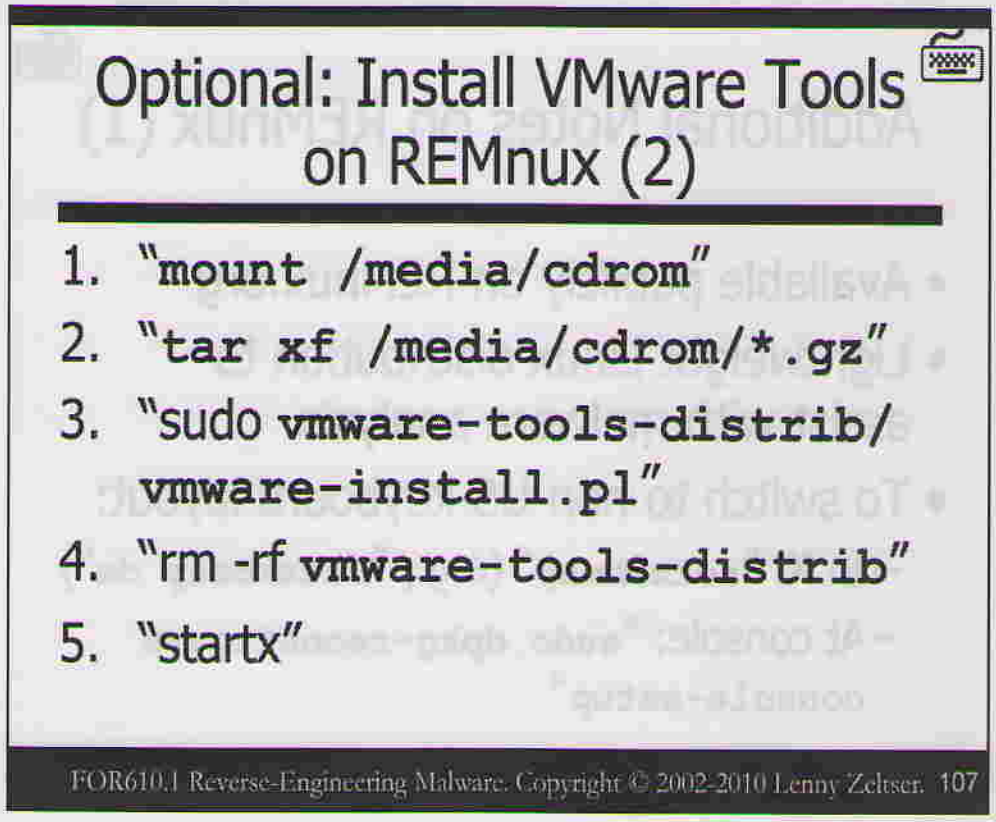
Now that your laboratory is set up, take a minute to write down configuration details so that you can easily recall them during future exercises.

You can now shut down the virtual machines. To do so in REMnux, type "`shutdown`". However, you may find it more convenient to suspend both virtual machines by pressing the VMware pause button or via the VM > Power > Suspend menu option in VMware; this way, it'll be faster for your to spin up your lab when you have the need for it.



VMware Tools is not required for the proper operation of REMnux. Those using REMnux as a virtual appliance with VMware will benefit from installing VMware Tools; namely installing VMware Tools will allow the user to change the resolution of REMnux to automatically match the user's screen size.

If you decide to install VMware Tools, boot the REMnux virtual machine, if it's not already running. Then from the VMware menu select VM > Install VMware Tools... If you're not already logged into REMnux, login as the user "remnux", then continue by following instructions on the next page of this book.

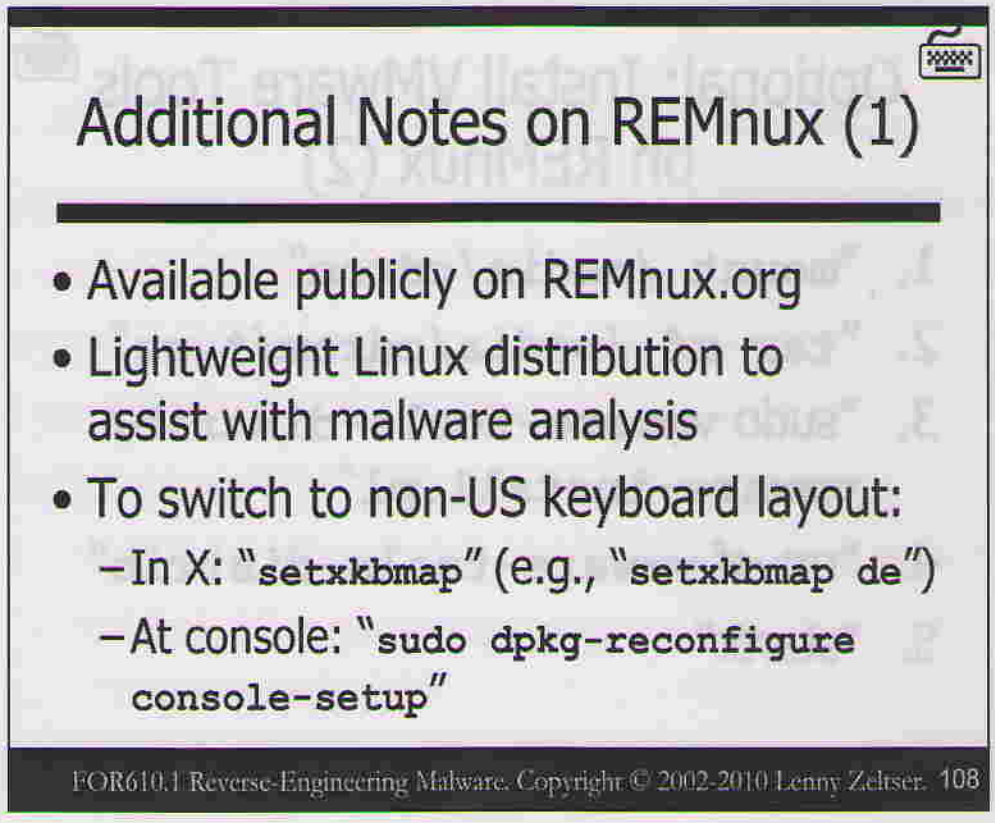


After following instructions on the previous page, continue by executing the commands on this slide (without quotation marks) to install VMware Tools on REMnux.

When running the "`vmware-install.pl`" you'll be asked questions about how to install the tools. Please accept all default values by pressing Enter when presented with a question.

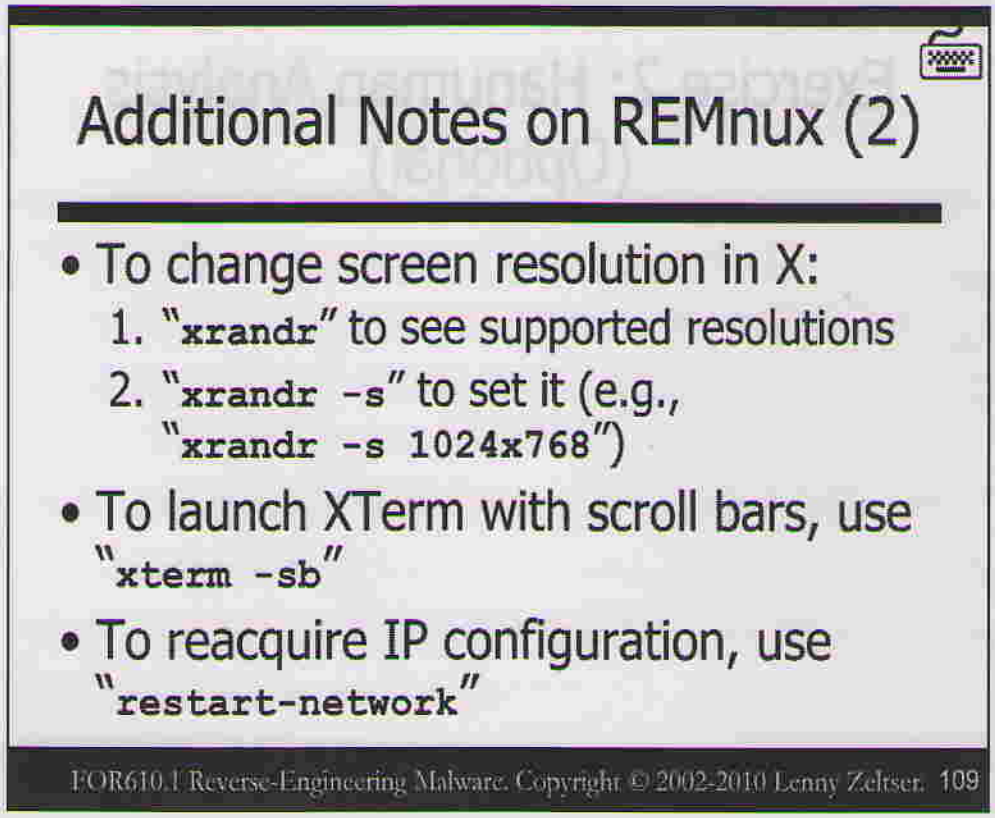
Installing VMware Tools takes a few minutes, so be patient.

Optionally, if you wish to activate the VMware Toolbox utility, run "`vmware-toolbox &`" after starting X.



A few additional notes on REMnux, which we will use throughout the FOR610 course. It's a is a lightweight Linux distribution for assisting malware analysts in reverse-engineering malicious software. The distribution is based on Ubuntu, and was originally designed to be used in this course. It's also available to the general public from http://REMnux.org.

If you are using a system that has a non-US keyboard layout, you may need to change the layout of your keyboard by using *setxkbmap* and *dpkg-reconfigure* as shown on this slide.



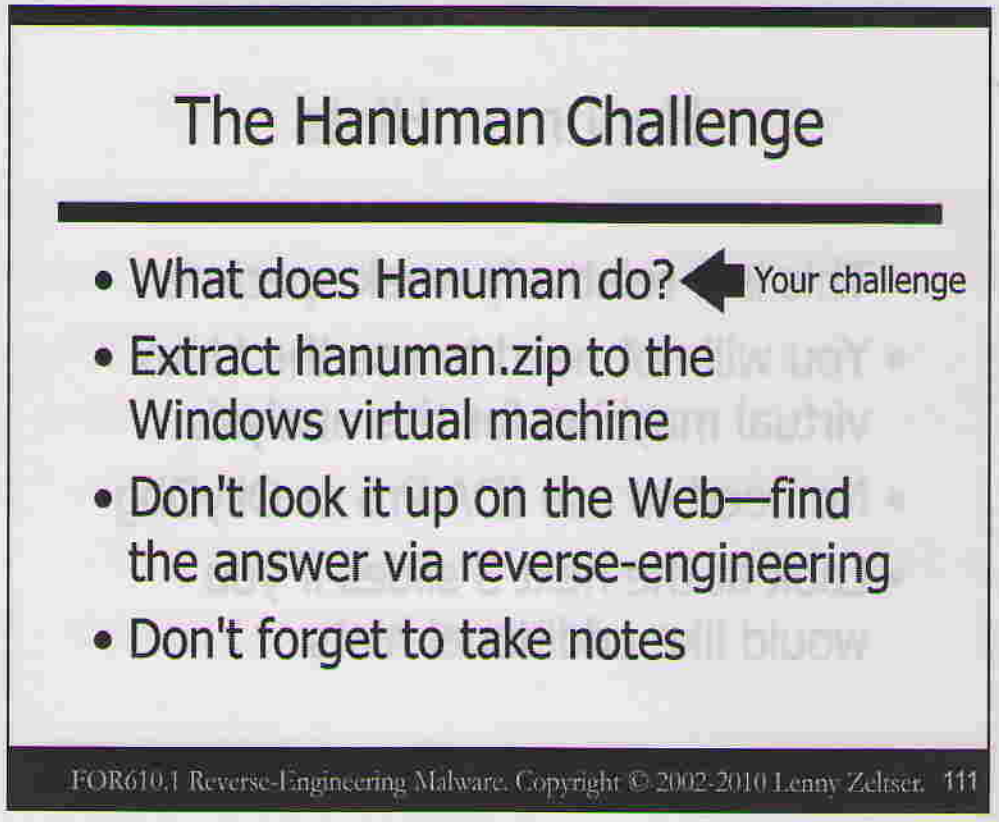
If you wish to change the screen resolution of REMnux when running X, you can do so by first running "`xrandr`" to see a listing of the resolutions supported by the system. Then run "`xrandr -s`" to set the desired resolution (e.g., "`xrandr -s 1024x768`"). Note that using "`xrandr`" is not necessary if you installed VMware Tools into the REMnux virtual machine; installing VMware Tools allows VMware to automatically resize the resolution to match your screen size.

If you wish to launch XTerm that has scroll bars, use the "`xterm -sb`" command.

REMnux is configured to automatically start a DHCP client. To reacquire your network configuration, type "`restart-network`".

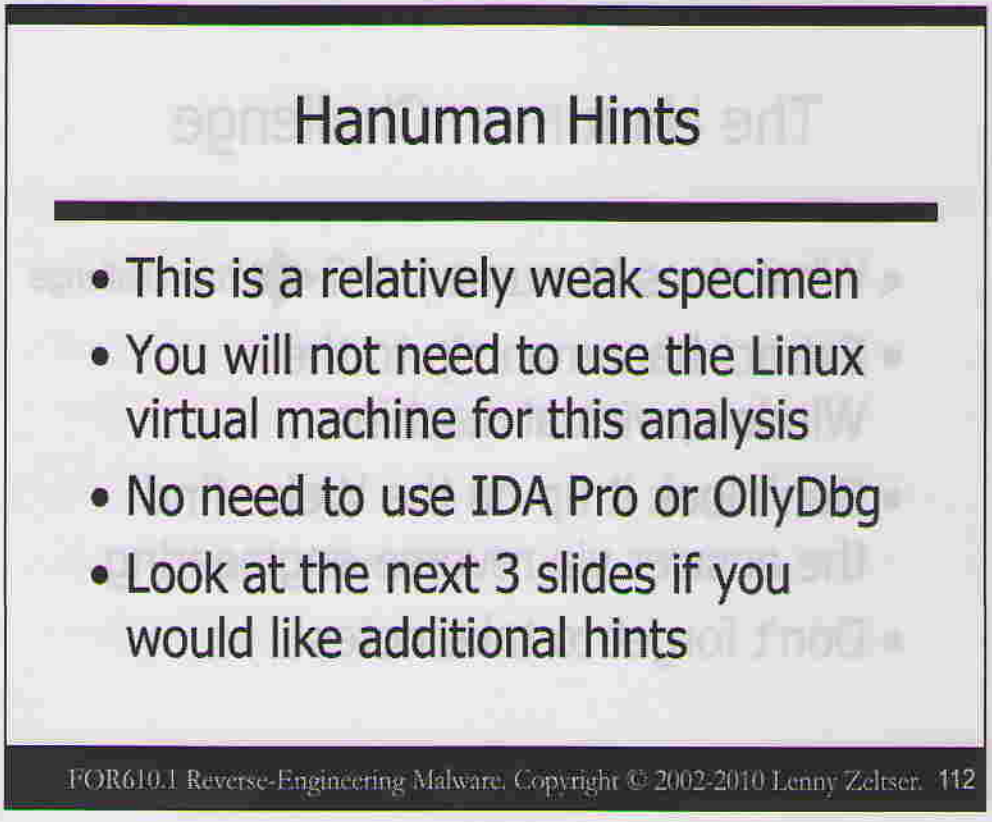
Exercise 2: Hanuman Analysis (Optional)

In this optional exercise we will examine a malware specimen called Hanuman. Please work on it only if you've already completed Exercise 1 and have some time before continuing with the course.



Your challenge is to get a sense for the capabilities of this malware specimen. Use the tools and techniques you learned in this session to learn about Hanuman. Please fight the temptation to learn about Hanuman by browsing the web.

An essential part of every analysis is taking notes as you uncover new information or perform new experiments. It's easy to get caught up in the moment and not write down your findings, but it's equally easy to forget what occurred during which exact circumstances. I encourage you to be patient and take notes as you proceed with your analysis in this, as well as in future exercises.

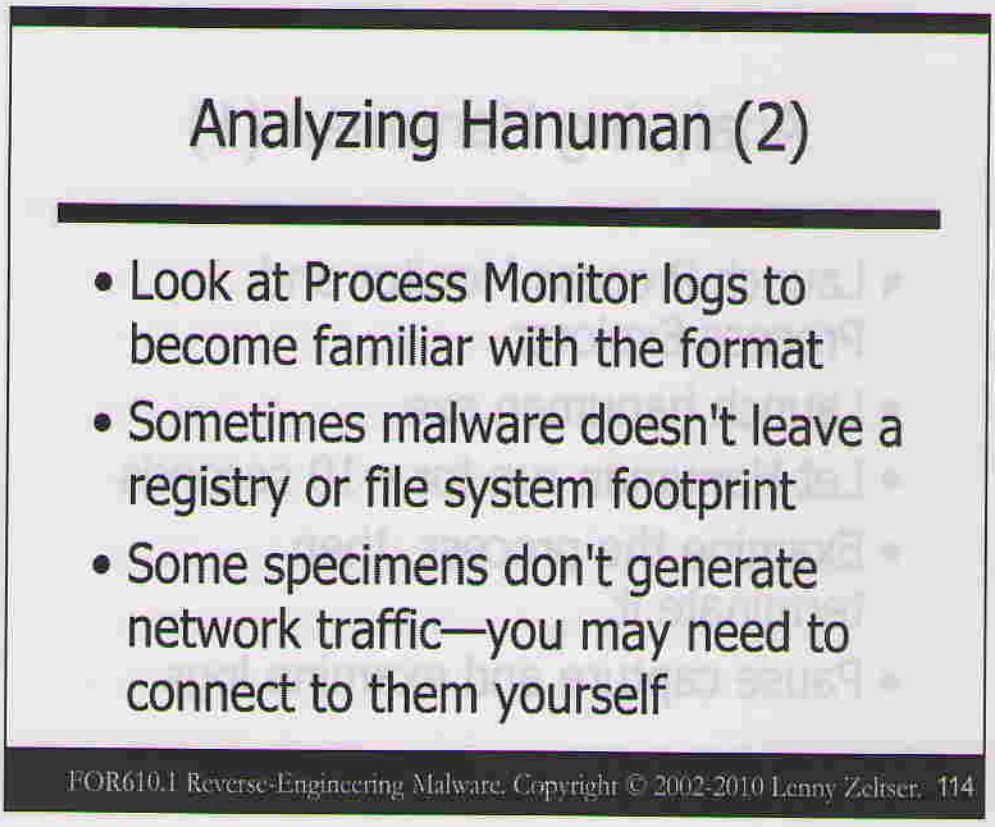


Hanuman is a pretty weak malware specimen—it is certainly not as powerful as srvcp.exe, which we examined in this section. To get a sense for its capabilities, you probably won't need to use a disassembler or a debugger, but you can certainly use this as an opportunity to become more familiar with these tools.

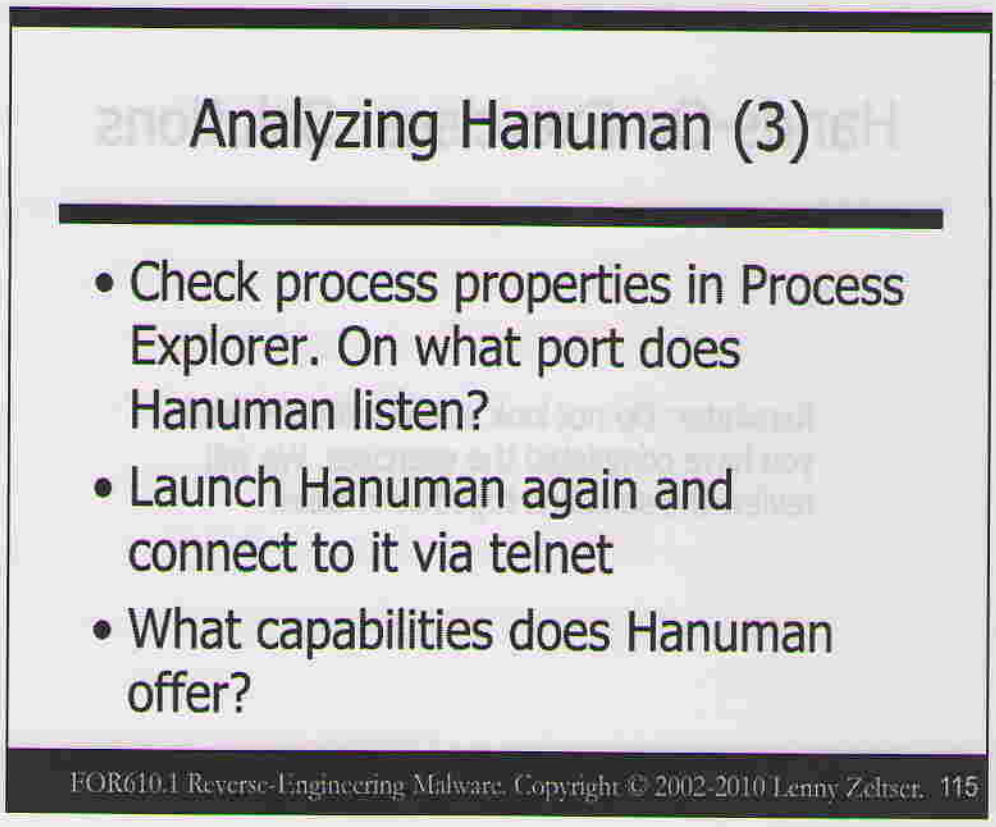
Analyzing Hanuman (1)

- Launch Process Monitor and Process Explorer
- Launch hanuman.exe
- Let Hanuman run for ~10 seconds
- Examine the process, then terminate it
- Pause capture and examine logs

You can use the behavioral process outlined on this slide to begin analyzing Hanuman.



Continue with your analysis by looking over the logs captured by Process Monitor. Don't be surprised if you don't see anything suspicious—some specimens don't write to the file system or the registry, and may not generate any traffic.

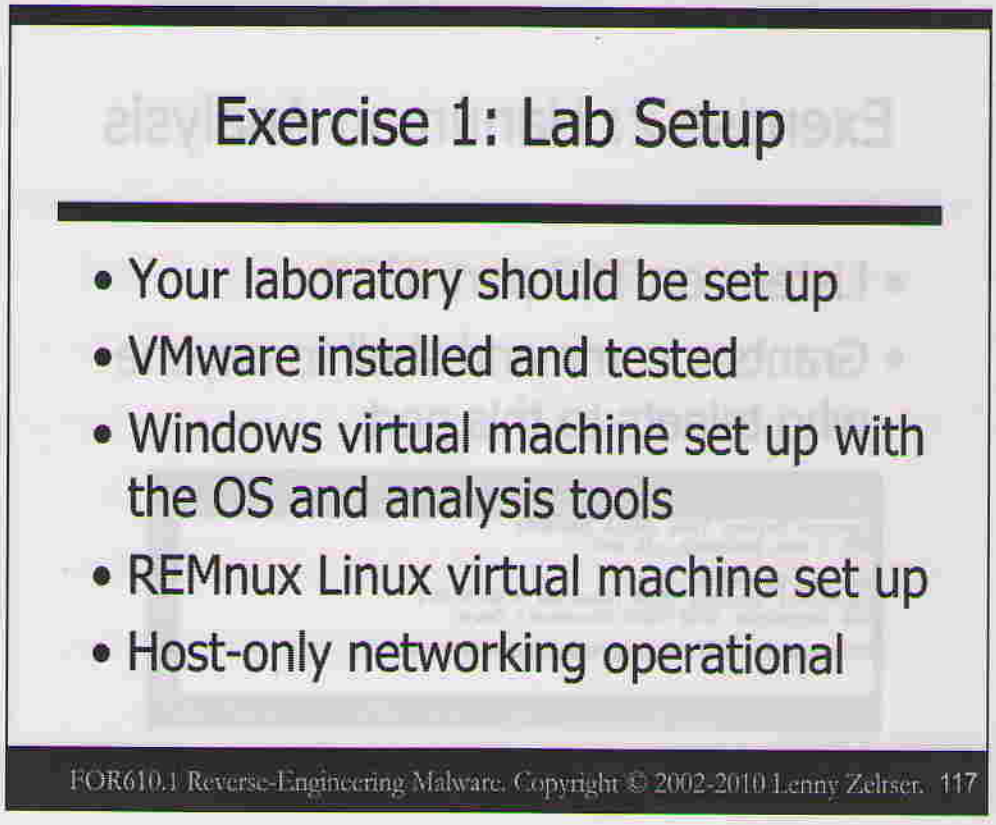


Process Explorer will show that Hanuman listens on a local port. Use telnet to determine the capabilities of this backdoor.

Hands-On Exercises: Solutions

Reminder: Do not look at the solutions until you have completed the exercises. We will review the solutions together in class.

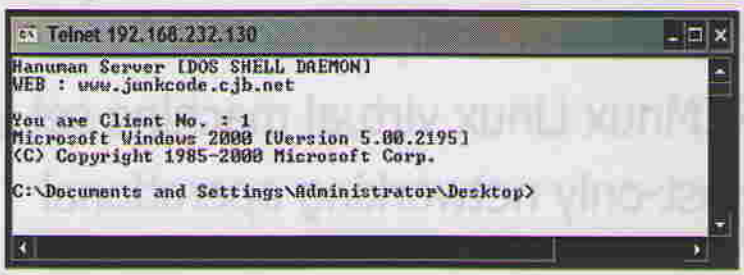
Let's go over the solutions to the previously-presented exercises. Reminder: Please do not look at these solutions until you have completed all exercises for this section. We will review the solutions together in class.



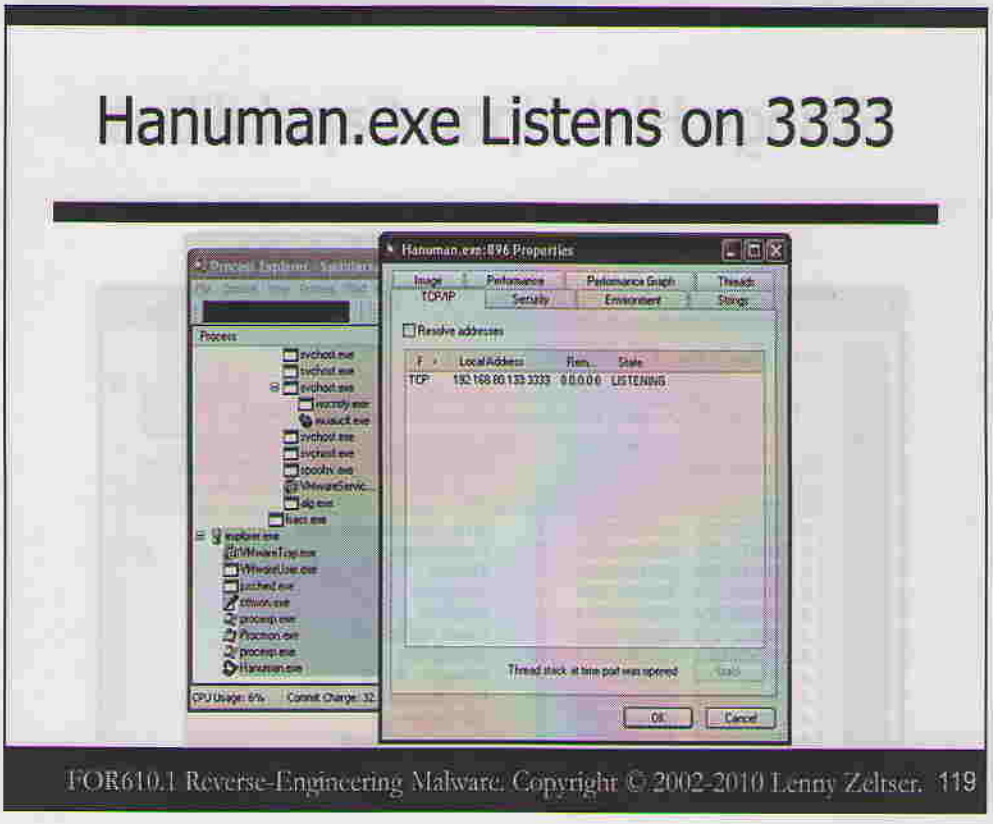
Exercise 1 presented you with detailed instructions for setting up your laboratory environment. You will use this environment for performing hands-on exercises through this course and, I hope, to reverse-engineer malicious software after this course is over.

Exercise 2: Hanuman Analysis

- Listens on TCP port 3333
- Grants a command shell to anyone who telnets to this port

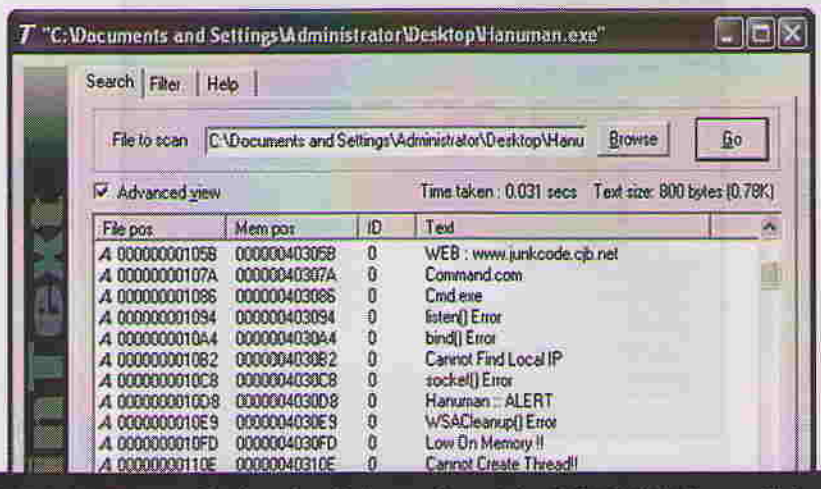


Exercise 2 asked you to take a brief look a Hanuman, which is a simple backdoor program that listens on TCP port 3333 and grants a command shell to anyone who connects to that port using *telnet*.

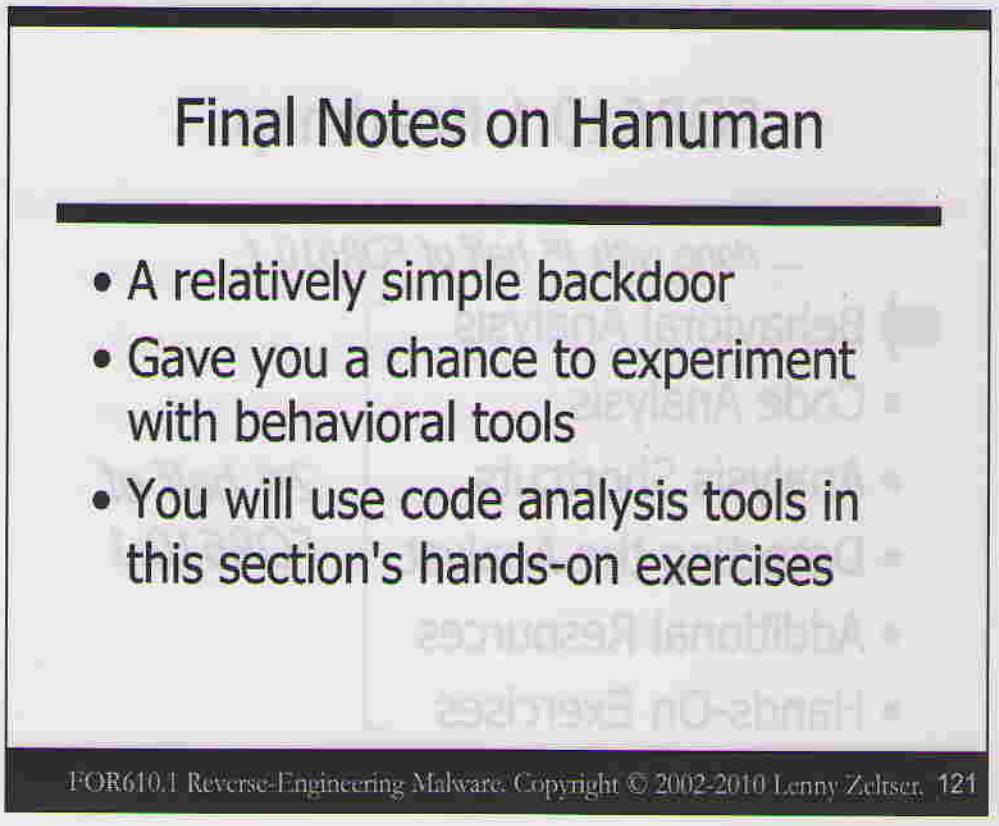


Process Explorer shows the Hanuman.exe process. To review its network activity, right-click on the process, select properties, and go to the TCP/IP tab. This is a very convenient way to see that the process listens on TCP port 3333.

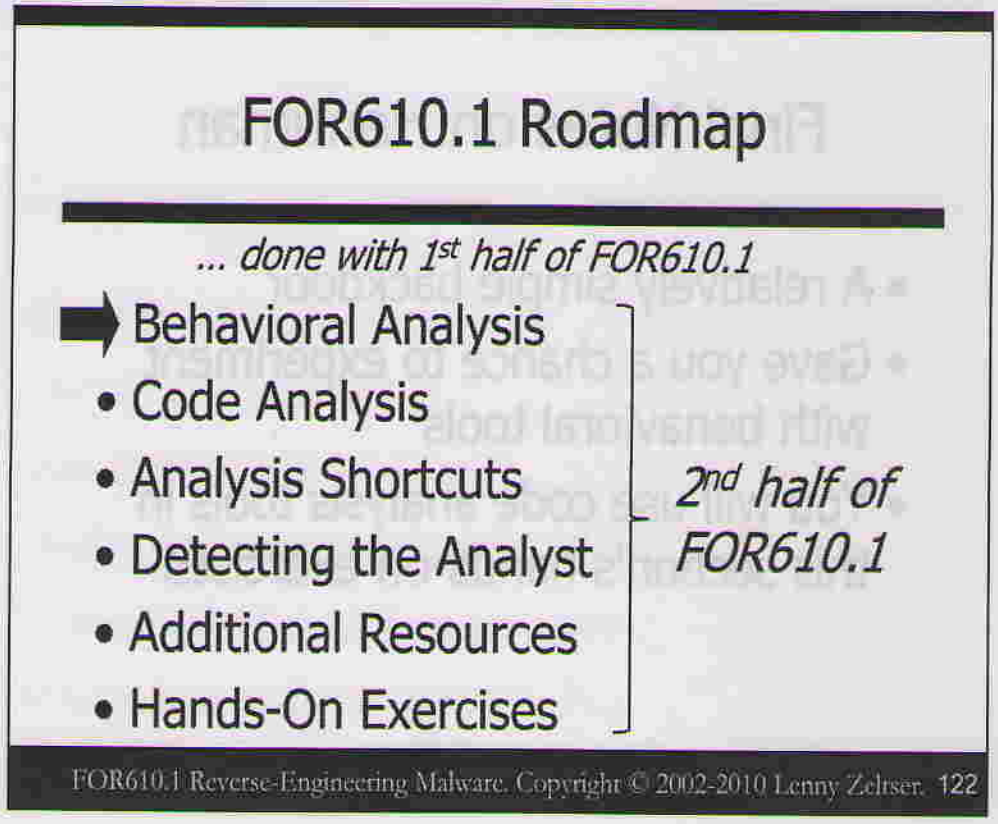
Strings Hint Upon Capabilities



You could use BinText to look at strings embedded into Hanuman. This offers several indications regarding the program's capabilities, but they are not definitive.

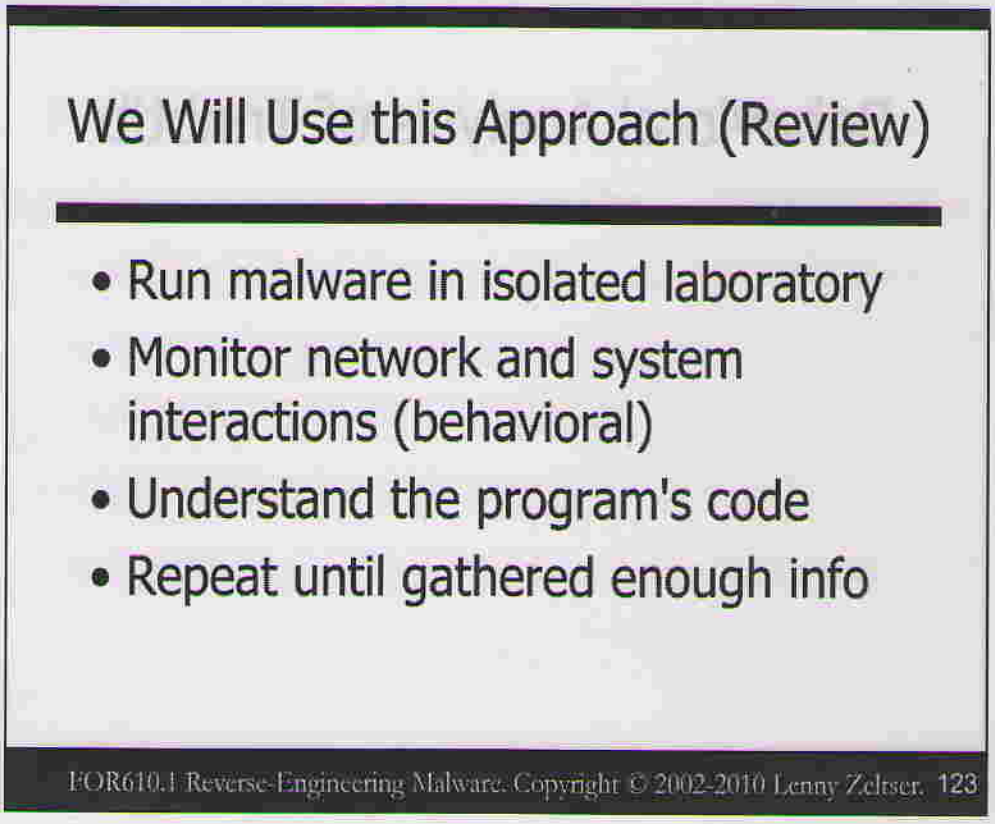


I asked you to look at Hanuman because it's a relatively straightforward backdoor that, at the same time, gives you a chance to become familiar with some of the tools mentioned in the last section.

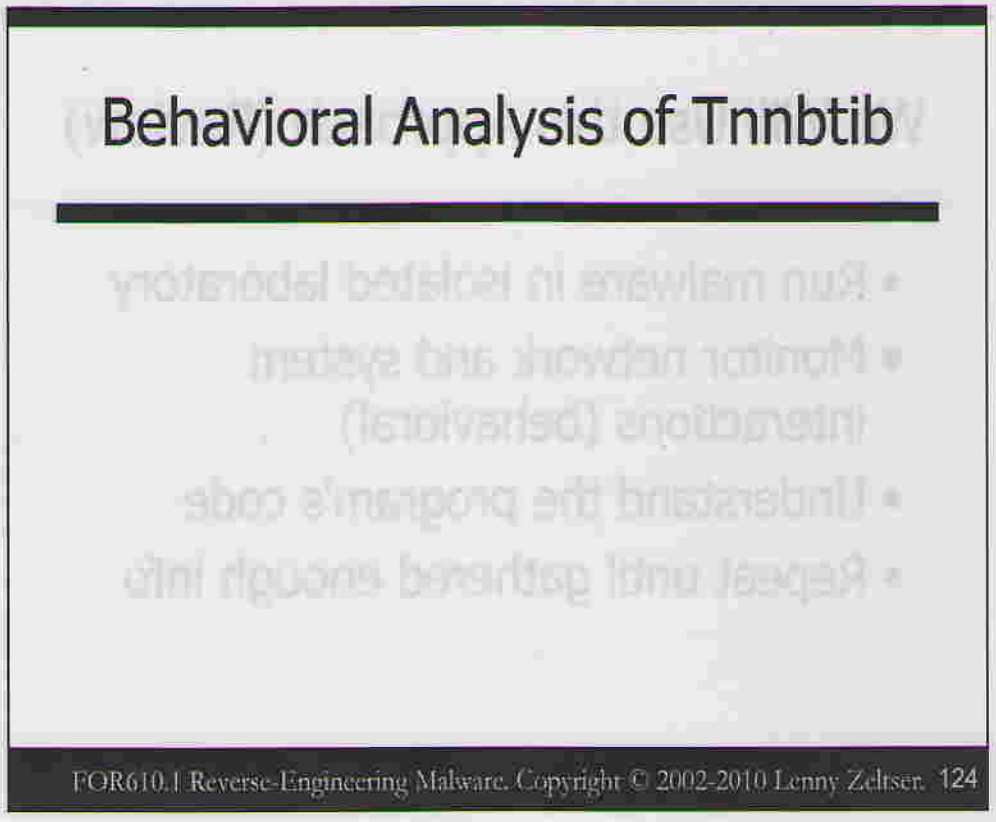


The bulk of this section will be devoted to a case study of a trojan named Tnnbtib, in which we will attempt to gain control over this malware specimen. We will also explore several miscellaneous topics, such as shortcuts for behavioral analysis, detecting the analysis sandbox, and additional learning resources.

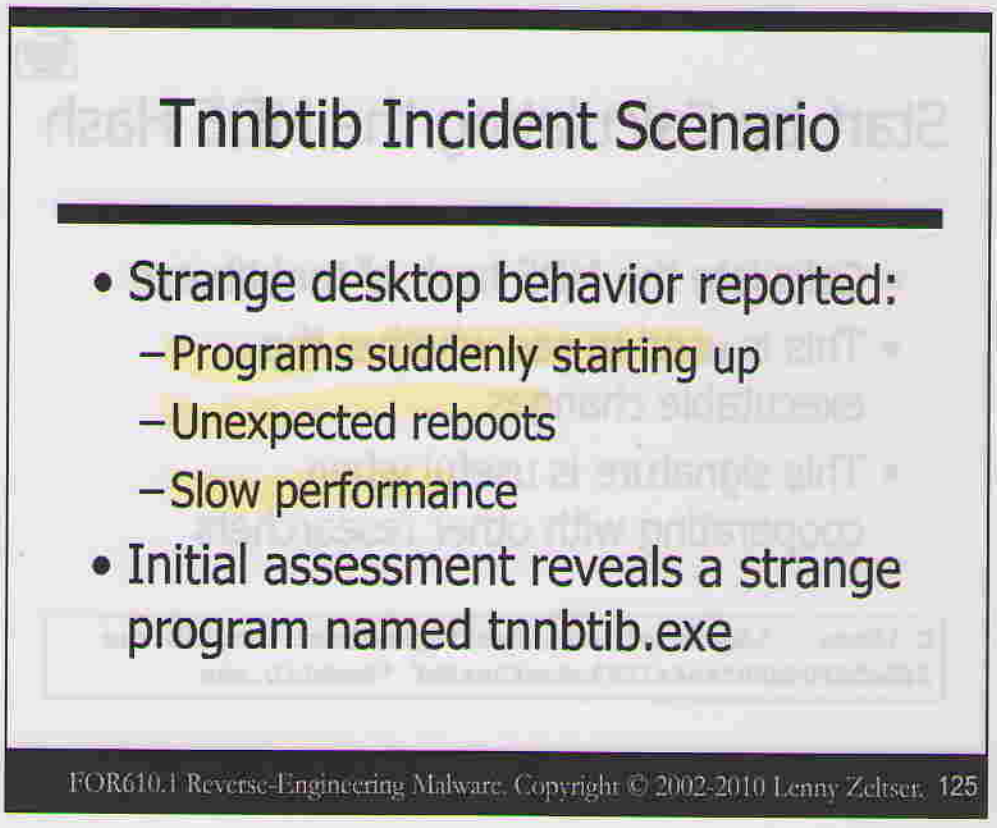
In this section we will reinforce the analysis methodology introduced earlier, and will focus on techniques for gaining control over a trojan in the isolated laboratory environment.



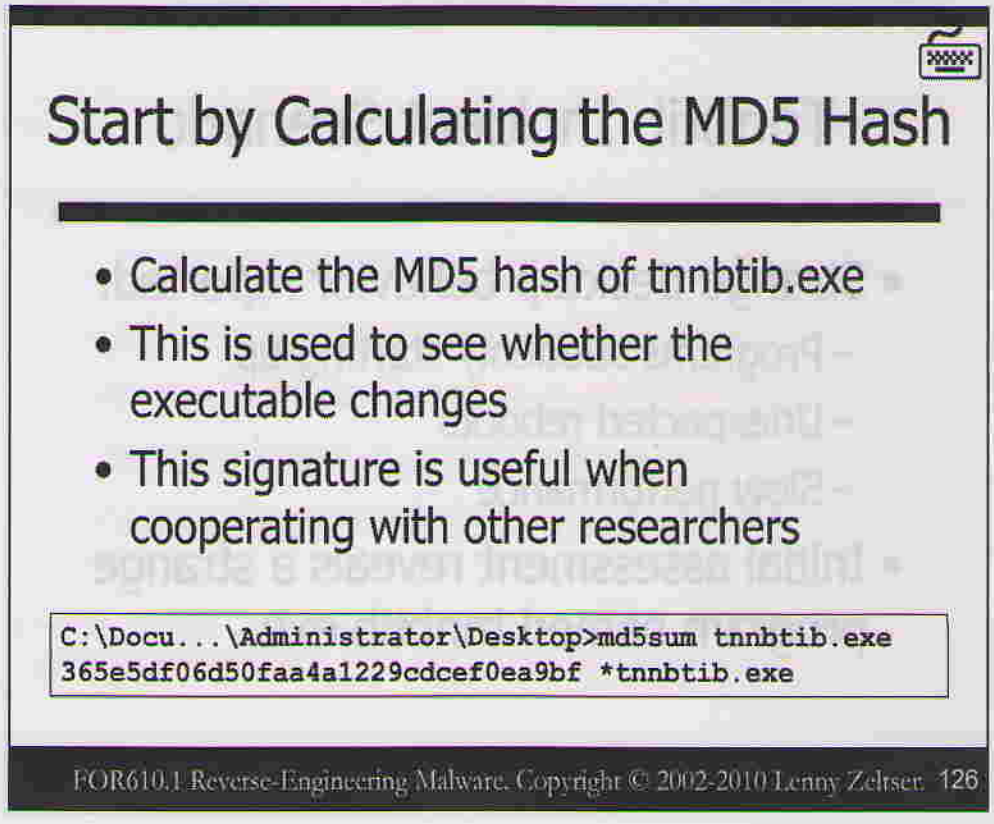
When performing the analysis we will use the approach that I introduced in the previous section.



To achieve this section's goals, we will analyze a malware specimen called Tnnbtib. A copy of this specimen is on your course DVD in \Malware\day1\tnnbtib.zip.



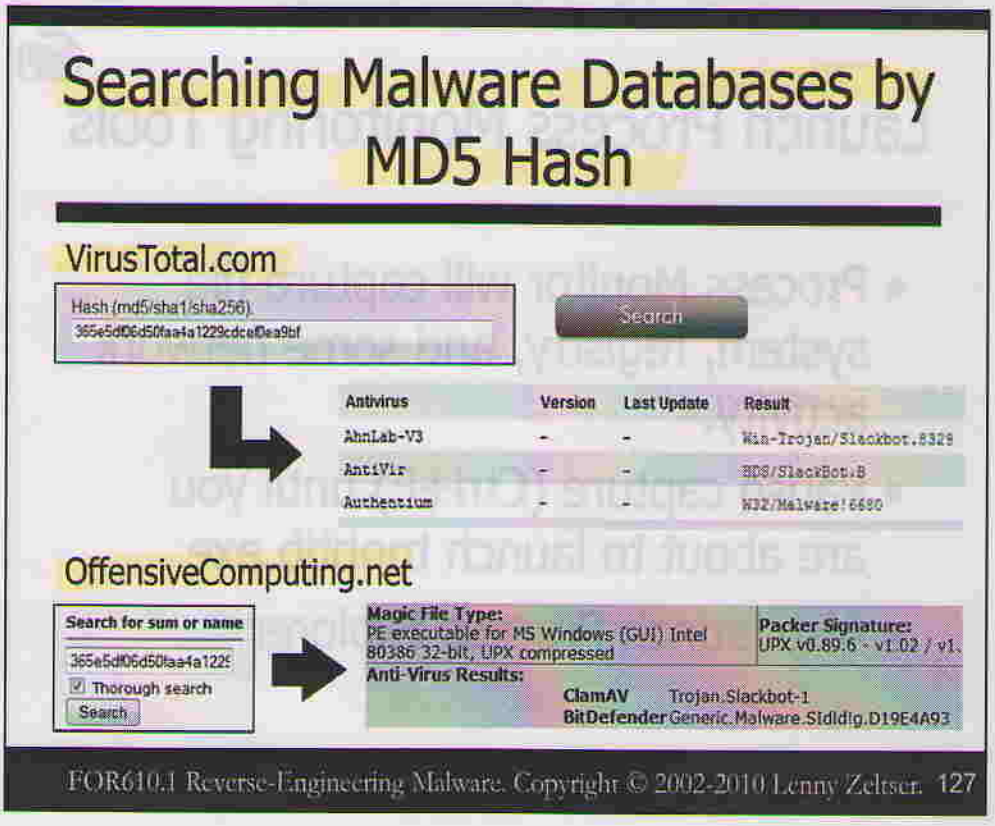
Consider this scenario: A user in your organization reports that her desktop is misbehaving in the manner outlined in this slide. Your initial assessment reveals an unfamiliar program named tnnbtib.exe running on the user's workstation. Anti-virus tools don't recognize this program as malware, and you cannot locate any relevant information about it on the Web. You grab a copy of tnnbtib.exe, roll up your sleeves, and begin the reverse-engineering process.



As soon as you receive an executable that you are about to analyze, it is a good idea to calculate its MD5 hash. When you execute "`md5sum tnnbtib.exe`", the output should be the same as the hash presented on this slide. Please run this command and make sure this is, indeed, so.

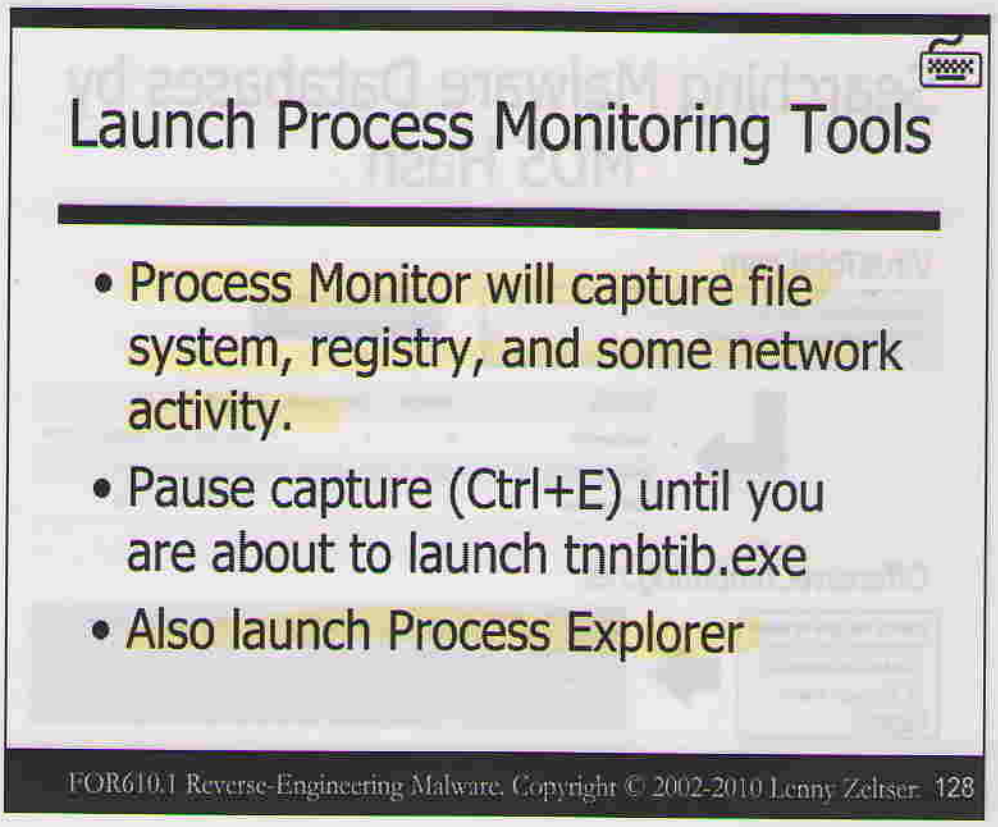
There are several advantages to knowing the MD5 hash of the malicious executable. First, it is not uncommon for the executable to remove itself from the location from which you ran it, and move itself to another location. Alternatively, the executable may automatically extract other files from the original file. Whichever the case, having an MD5 hash will allow you to check whether a file that is added to the system after running the executable is just a copy of the executable itself, or whether it is a newly created file that needs to be analyzed independently.

Additionally, knowing an MD5 hash of a malware specimen helps when communicating with fellow researchers. The hash allows you to check whether the malware file that your colleague has is the same as yours or not. Too often researchers try to share findings without establishing whether they're analyzing the same executable.



One practical mechanism for using the specimen's MD5 hash to benefit from other researcher's findings involves querying malware databases. Two good sites for this, which allow a search by MD5 hash of the file, are VirusTotal.com and OffensiveComputing.net. As you can see on this slide, both slides were able to present at least some information about our specimen. In many cases, this can be very helpful for getting a general idea about the malware sample we are about to analyze in the lab.

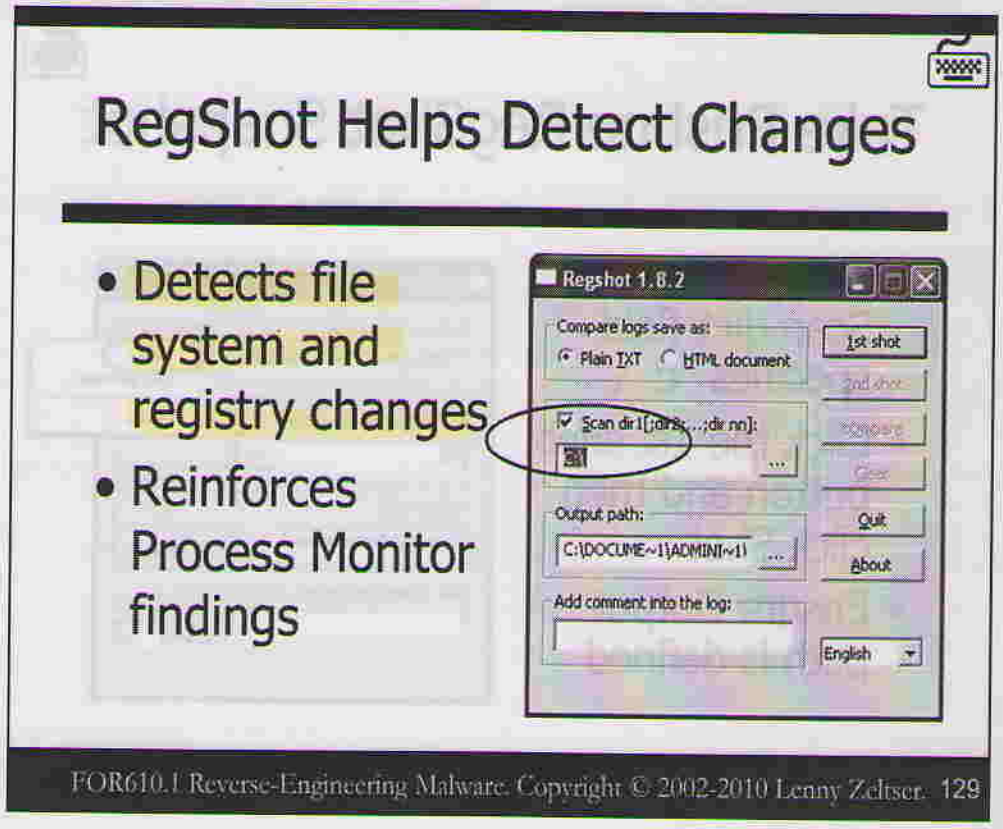
We will touch upon the usefulness of sites such as VirusTotal a bit later in the course. VirusTotal lets you also upload a suspicious executable, in which case it will scan it with multiple anti-virus engines in an attempt to determine whether the file is malicious. Even if your organization's policy does not allow you to share suspicious files with a third party such as VirusTotal, it may allow you to query VirusTotal's database of past scans by the file's MD5 hash.



Next, launch Process Monitor, which will give you the ability to record an activity log of file system, registry, and network-related operations performed by tnnbtib.exe

As soon as you launch these tools, you will probably begin to see activity that is part of normal operation of Windows: files being accessed, registry keys being queried and sometimes set… To keep the amount of noise in your logs to a minimum, I suggest pressing Ctrl+E in Process Monitor until you are about to launch tnnbtib.exe. After pausing capture, press Ctrl+X to clear the logs and purge unnecessary records.

Also launch Process Explorer. You will use it to examine the malicious process while it runs, and to terminate the malicious process.

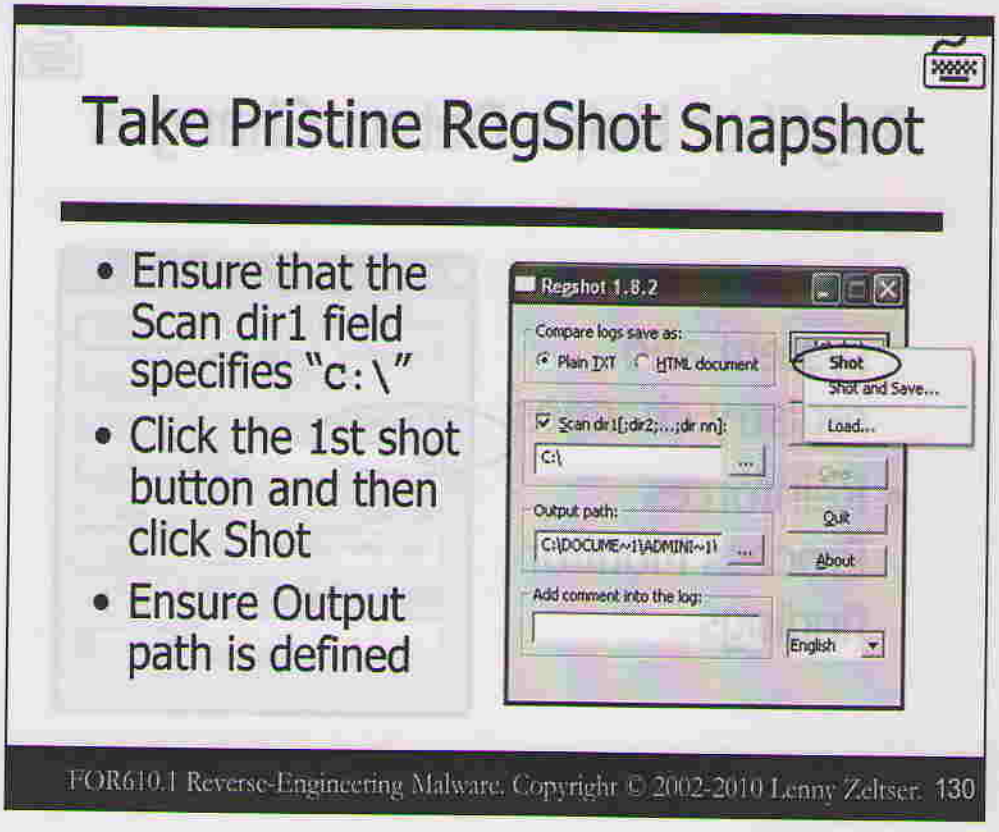


You can use RegShot, a freely available utility, to detect changes made to the system by malicious software. RegShot allows us to create a baseline of the pristine system, and compare it to the system's state after the trojan ran. Using RegShot in conjunction with Process Monitor allows us to be relatively certain that we can detect the trojan's effect on the system.

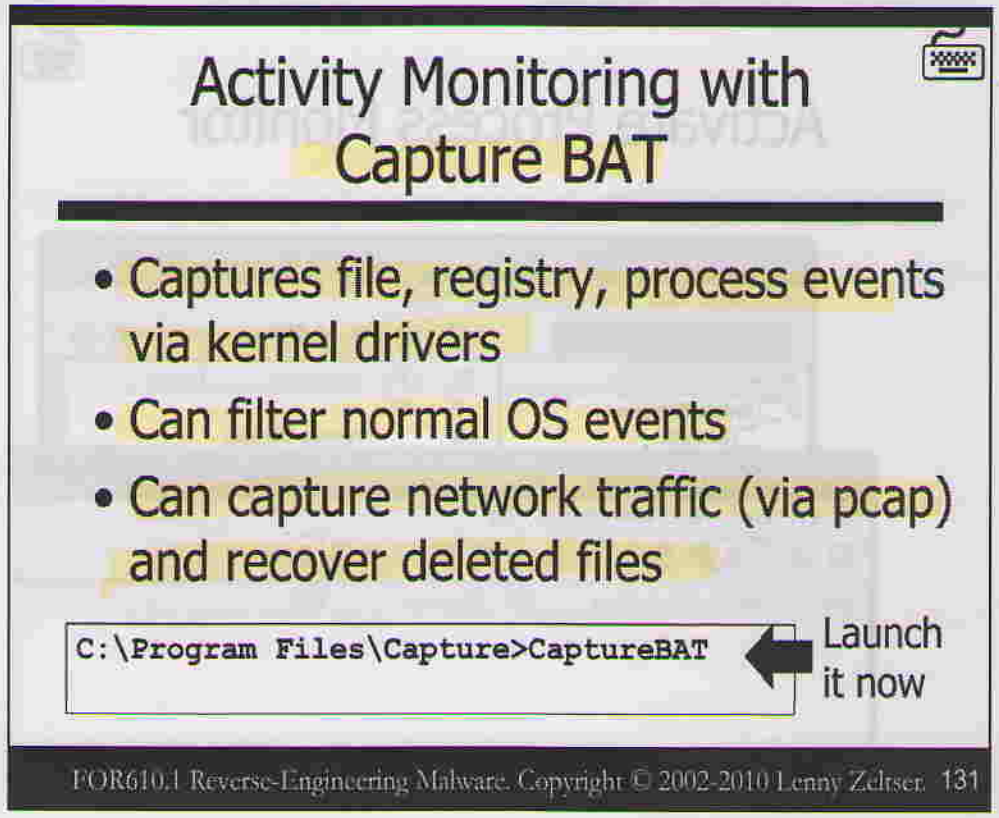
To ensure that RegShot captures both changes to the file system and to the registry, activate the "Scan dir" checkbox, and type there "`C:\`"; this way, RegShot will scan the full C: drive from its root.

RegShot works with all Windows operating systems, and is able to detect changes to the system's registry and file system. You can download RegShot for free from http://sourceforge.net/projects/regshot.

Free alternatives to RegShot include SpyMe Tools (see http://lcibrossolutions.com) and InstallWatch (http://epsilonsquared.com).



Use RegShot to take the "before" snapshot of the Windows box while it's still clean.



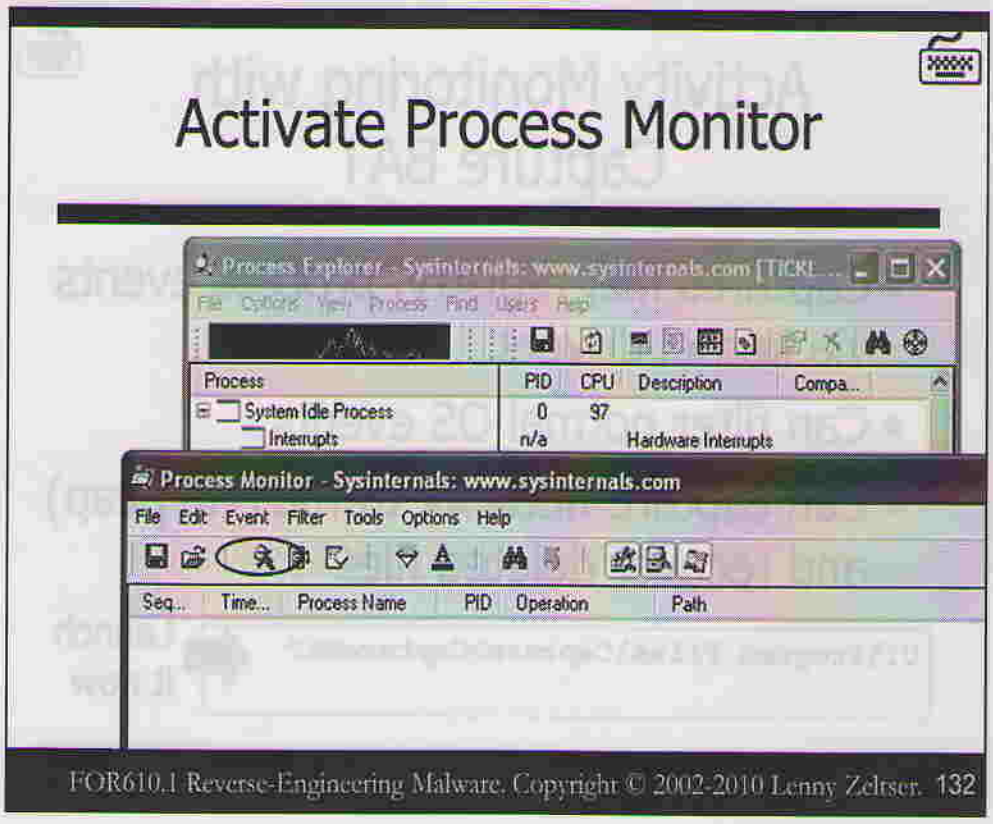
Capture BAT is a free utility developed at Victoria University of Wellington by Ramon Steenson and Christian Seifert. It is designed to capture behavioral events associated with file system, registry and process activities on a Windows system.

The tool can be configured to ignore certain normal Windows events, to make it easier to spot anomalous activities. These filters are defined in text files FileSystemMonitor.exl, RegistryMonitor.exl, and ProcessMonitor.exl, located in Capture BAT's installation directory. The tool ships with the configuration that ignores particularly noisy normal Windows events.

Capture BAT can also save copies of files deleted or modified while the tool was running. To activate this feature, launch it with the "`-c`" parameter. This way, the tool will copy deleted files to its "log\deleted_files" directory. Consider redirecting Capture BAT's output to a file, in case it prints many records.

If you have WinPcap (www.winpcap.org) libraries installed on the system Capture BAT will be able to capture packets entering and leaving the system. The packets will be stored in a .pcap file in the program's "log" directory. To analyze the file's contents, you will need to use a network sniffer program, such as Wireshark (www.wireshark.org). Conveniently, Wireshark is installed on REMnux

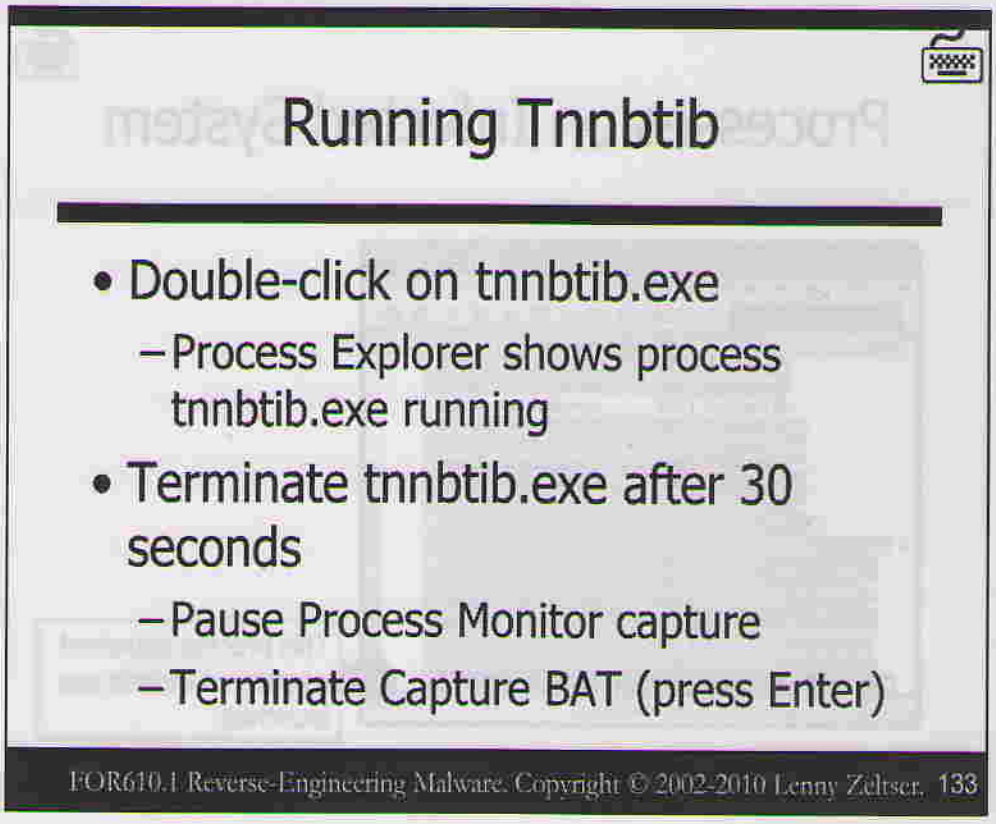
For more details about Capture BAT see: https://honeynet.org/node/315



With Process Monitor launched but paused, make sure you have tnnbtib.exe in sight. Also, make sure Process Explorer is running, so that you can see which process the executable will run as, and so that you can terminate it.

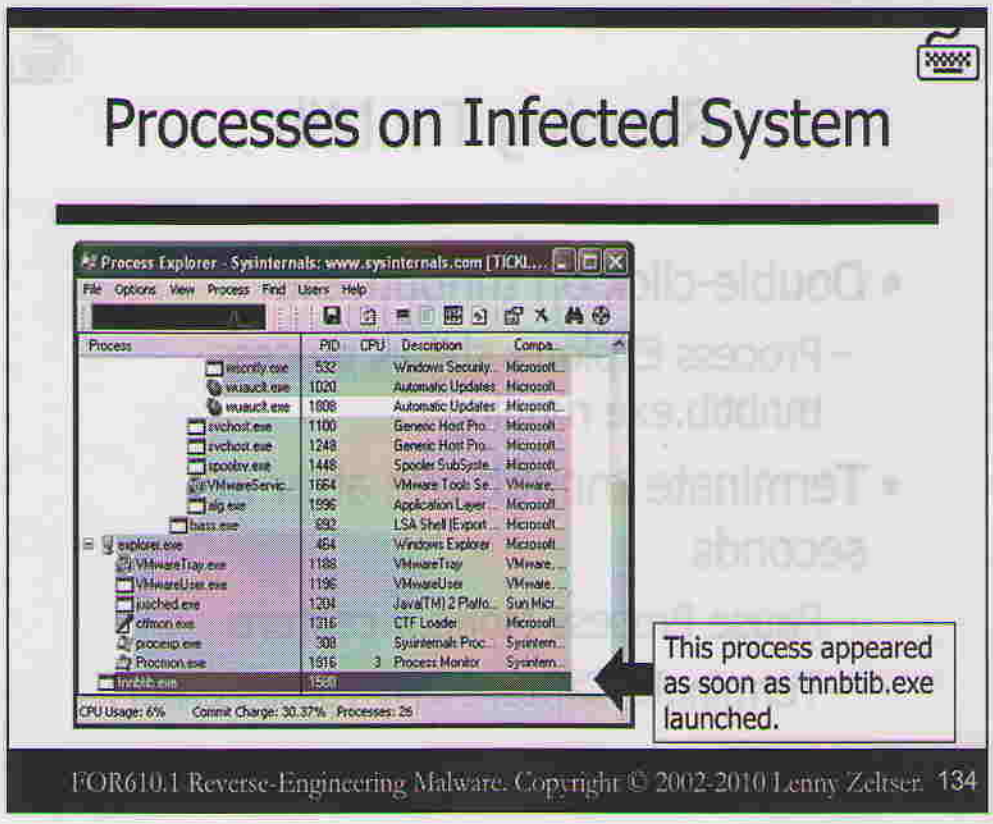
Enable capture in Process Monitor.

Now, double-click tnnbtib.exe to launch it!

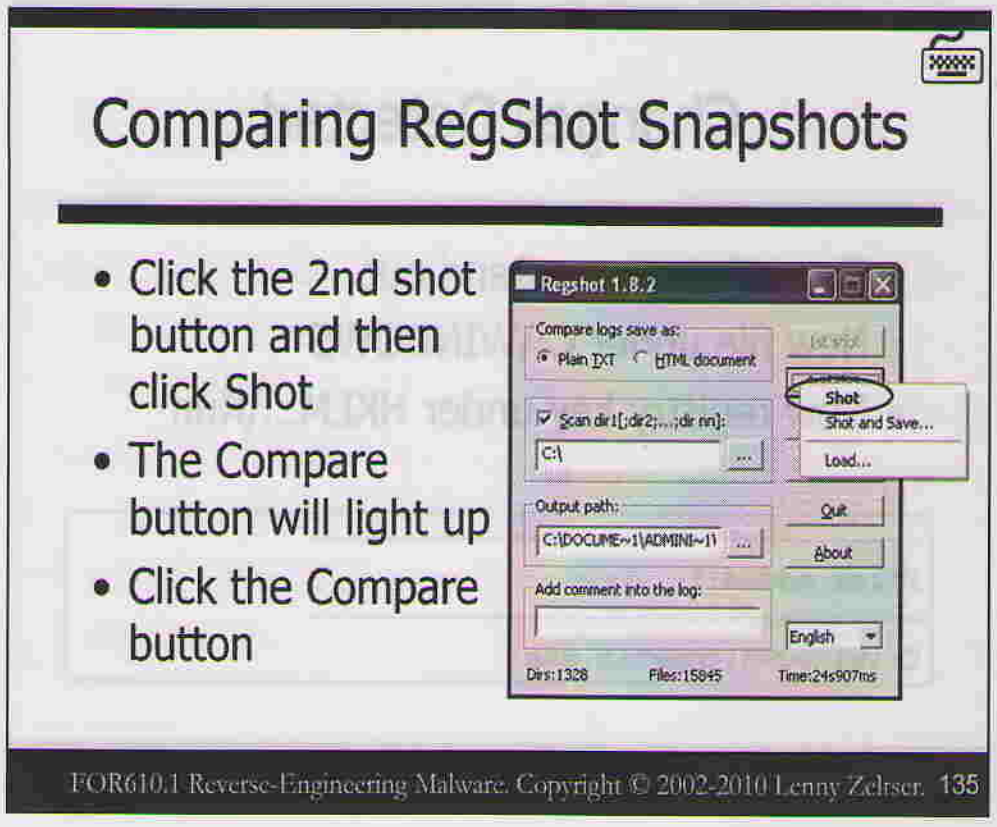


If you look at Process Explorer, you should notice an unfamiliar process running on the now-infected system called tnnbtib.exe. After letting the process run for about half a minute, terminate it using Process Explorer. Then go to Process Monitor and pause capture. Then go to the Capture BAT window and kill Capture BAT by pressing Enter. (You can also kill Capture BAT by pressing Ctrl+C, but in this case it won't archive the files it captured in a zip file.)

We will examine what these tools recorded very shortly.

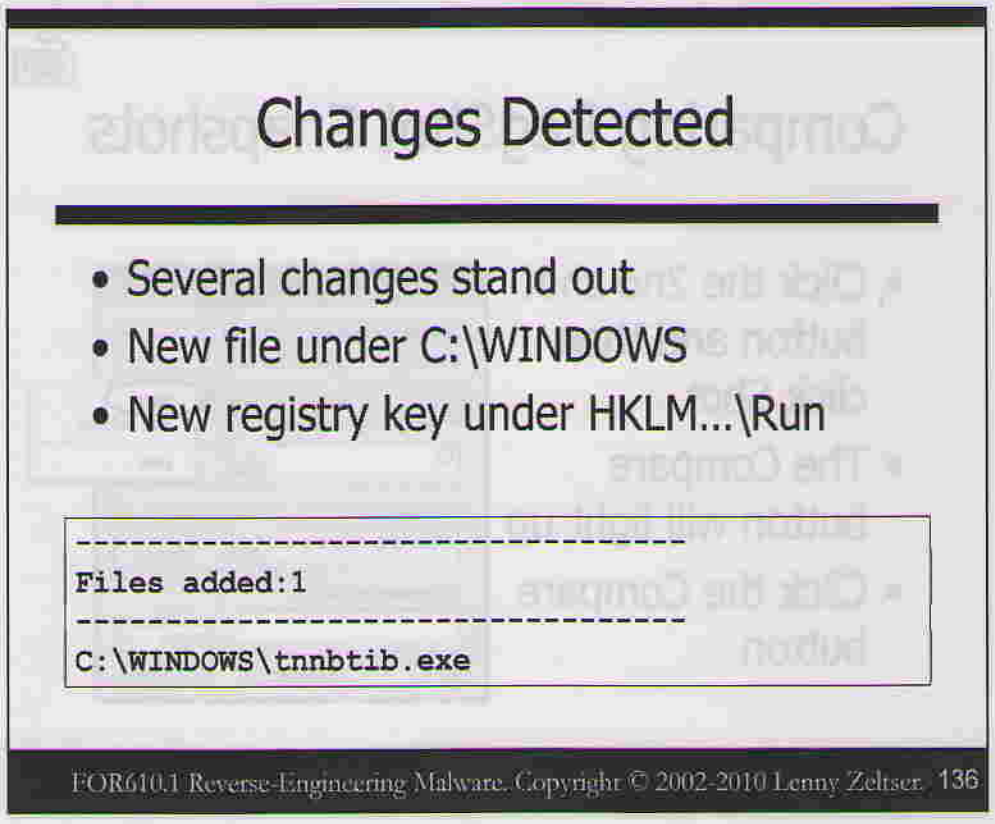


This slide shows a screenshot of Process Explorer displaying processes running on the infected system in our lab.



Before looking at logs of Sysinternals tools, which can be a little overwhelming, let's see what changes can be detected with RegShot. Go to the RegShot window where you took the initial snapshot, click the "2nd shot" button, and select "Shot".

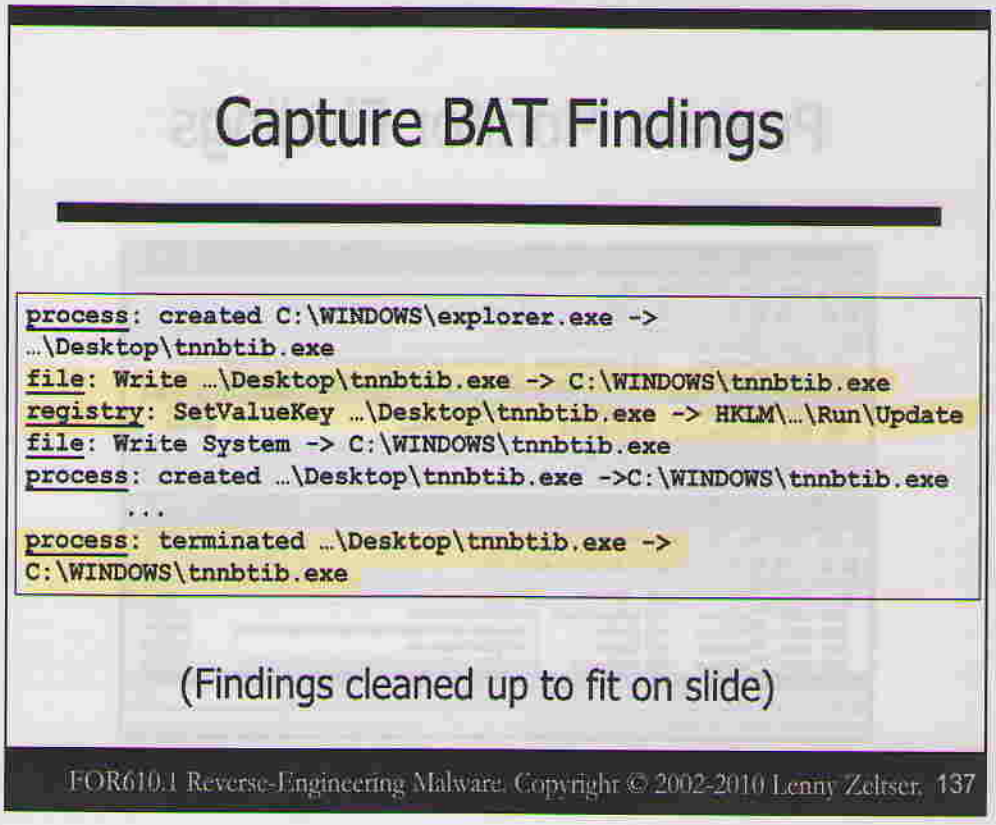
Once RegShot finishes scanning the system, select "Plain TXT" as the log format, and click the "cOmpare" button. The program will launch Notepad, showing you results of the comparison. This comparison log will also be saved in the directory specified in RegShot's "Output path" window.



Some of the changes detected by RegShot are expected to occur on most Windows systems. However, a few changes stand out. Specifically, you should be able to notice that a new file was created under the C:\WINDOWS directory (or under C:\WINNT if you're running Windows 2000). You may notice that the name of the new file matches the name of the new process that we discovered with Process Explorer.

You should also notice that a new registry key was added, with the purpose of running the newly created executable when the system reboots. This behavior is not uncommon for malicious software.

RegShot offers an easy way to see major changes to the file system and the registry. However, it cannot provide information about the specific sequence of events, or detect temporary changes to the system's state that we restored when the executable finished running. These are some of the reasons why we use tools such as Process Monitor to gather detailed logs regarding the executable's actions.



Capture BAT's output helps us understand what took place on the system when we infected it with tnnbtib.exe:

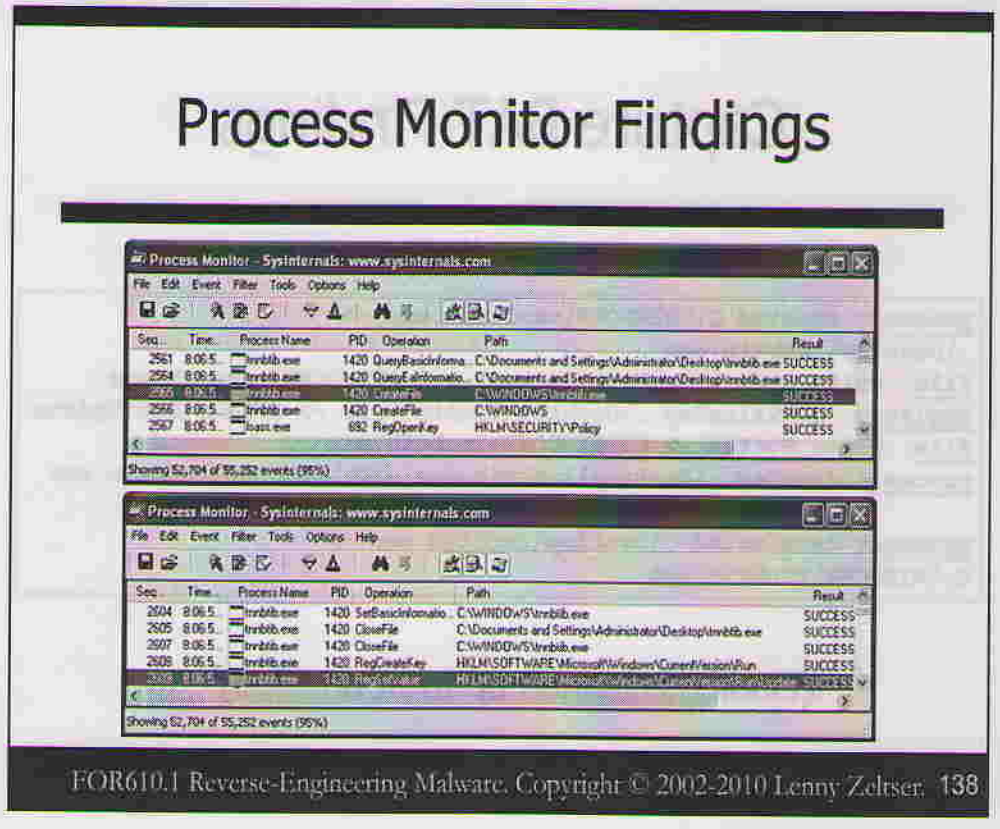
- explorer.exe (the Windows GUI shell) launched tnnbtib.exe from the desktop when we double-clicked on its icon.
- tnnbtib.exe process created the file tnnbtib.exe in the Windows directory.
- tnnbtib.exe process created the registry key …\Run\Update
- eventually tnnbtib.exe is terminated

Note that by default Capture BAT sends its output to the command window. We could have saved the output to a text file by redirecting it using the greater-than sign, such as:

```
CaptureBAT > c:\output.txt
```

Alternatively, we could have saved it to a file in a format that would be easier to process with a script or load it into Excel by specifying the "`-l`" parameter, such as:

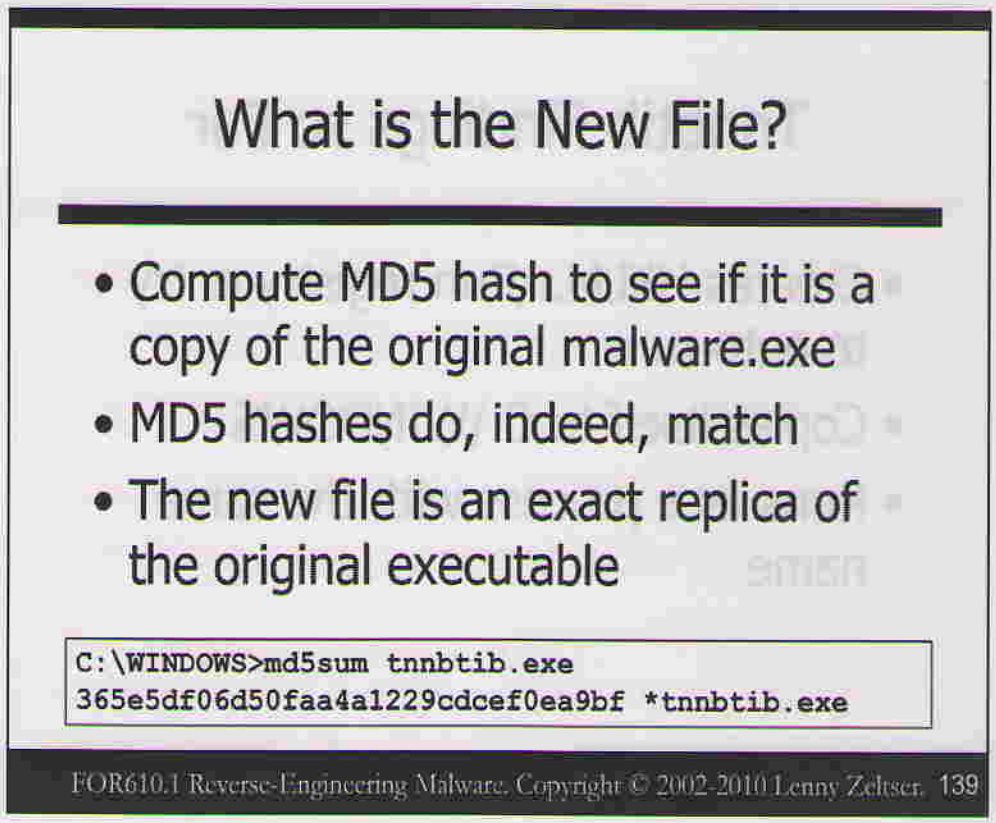
```
CaptureBAT -l c:\output2.txt
```



In the beginning of the Process Monitor log you would see tnnbtib.exe reading itself from the file system, and eventually writing to the file under C:\WINDOWS. If you experimented to see how the program would behave if the file already existed, you would see that tnnbtib.exe would make sure the registry key is properly set, and then launch the file under C:\WINDOWS.

Hint: To jump to the first entry in the log that mentions tnnbtib.exe, press Ctrl+F in Process Monitor, then enter "tnnbtib" and click OK. You could also use the filter functionality to only view events whose process name is "tnnbtib.exe".

Process Monitor logs also show how and when tnnbtib.exe created the HKLM…\Run key that we saw in RegShot and Capture BAT.

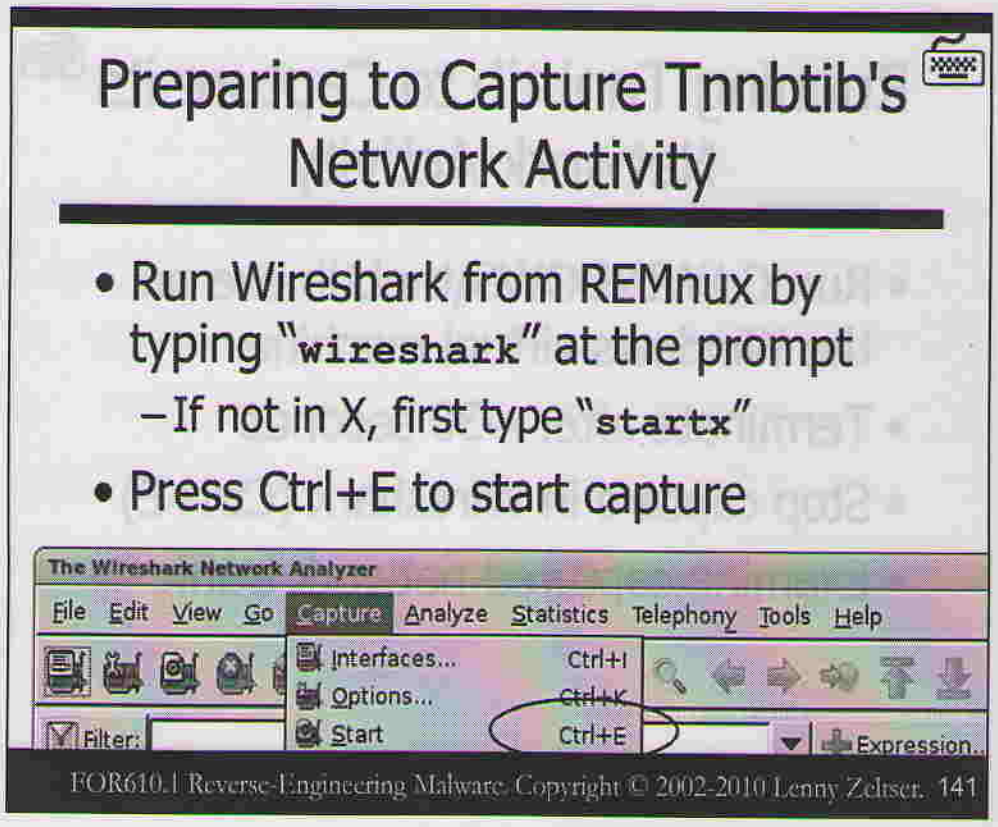


What is the newly created file? One of the easiest things to check is whether it is just a copy of the original. You can compute the new file's MD5 hash, as shown on this slide, and compare it to the hash of malware.exe. The two hashes are identical, which means that the newly created file is an exact replica of the original.

Tnnbtib Findings so Far

- Creates HKLM..\Run registry entry to auto-run
- Copies itself to C:\WINDOWS
- Runs as a process with the same name

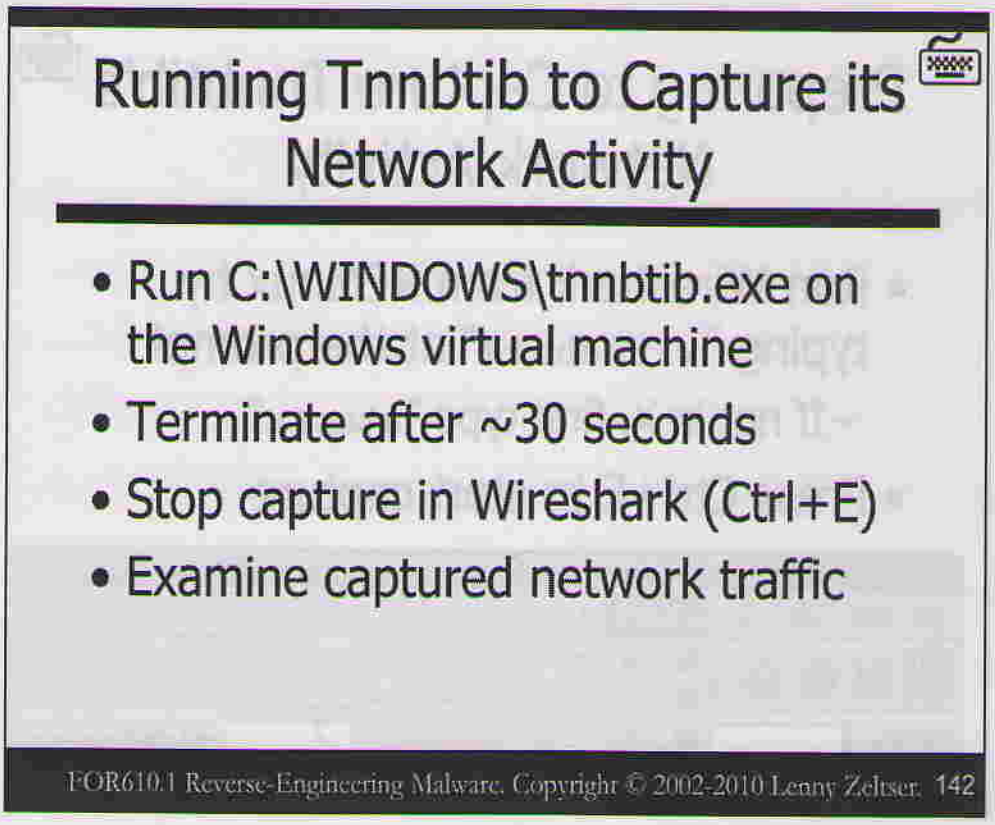
Once launched, tnnbtib.exe creates a copy of itself under C:\WINDOWS. The executable also creates a registry entry that ensures that its C:\WINDOWS copy will run whenever the system is rebooted. Then, tnnbtib.exe launches the copy of itself from C:\WINDOWS. The name of the process matches the name of the file that the program is running from.



Now that we know a little about how the program interacts with the local system, let's take a look at its network-related activity.

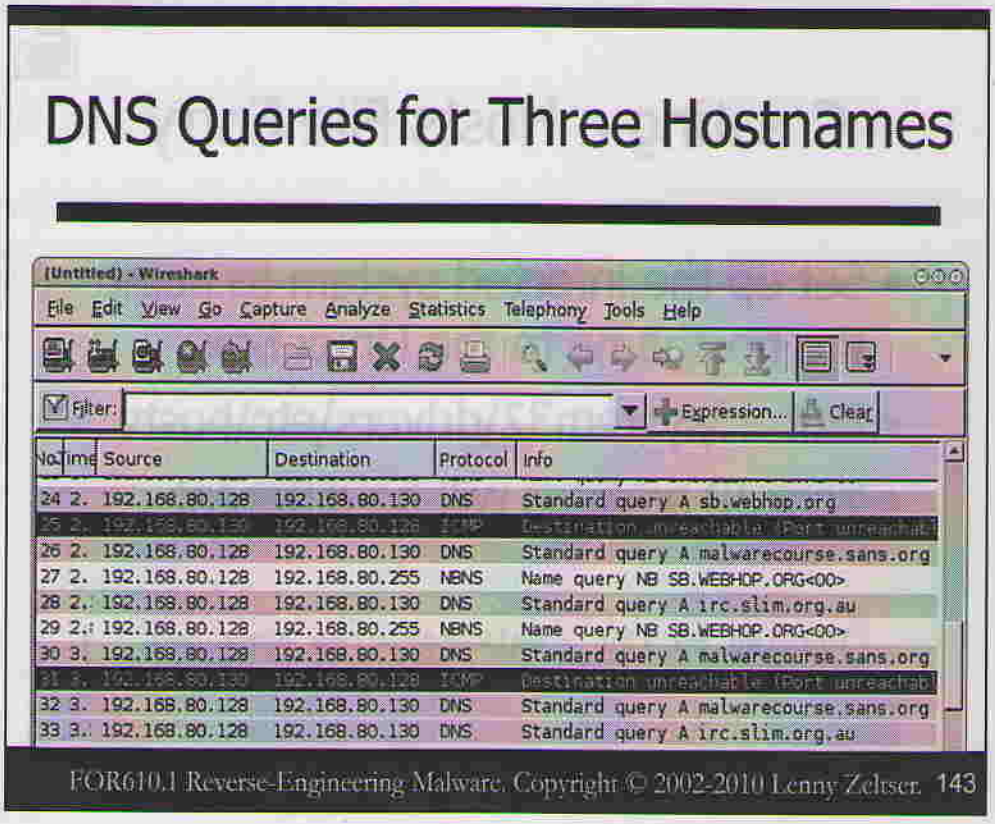
To monitor the malware specimen's network access, use the Wireshark sniffer, which is installed on the REMnux virtual machine. If REMnux isn't already running, start it up and, after logging in (user: "`remnux`"; password: "`malware`"), type "`startx`". Then, simply type "`wireshark`" at the Xterm prompt in X Window System and press Enter. You may be prompted for the root password. (Remember, the password is "`malware`".)

To begin capturing traffic in Wireshark, press Ctrl+E or go to the Capture > Start menu. This will start the capture and bring up the window that will show you, in real time, any packets Wireshark will see on the laboratory network.



Once Wireshark is monitoring the laboratory network, reinfect the Windows virtual machine by double-clicking on tnnbtib.exe in the C:\Windows folder. Let the malicious process run for about half a minute—to give it some time to perform actions worth observing—then terminate it using Process Explorer.

Then, stop capture in Wireshark by either pressing Ctrl+E or by using the Capture > Stop menu option. Then, examine the packets captured by the network sniffer.

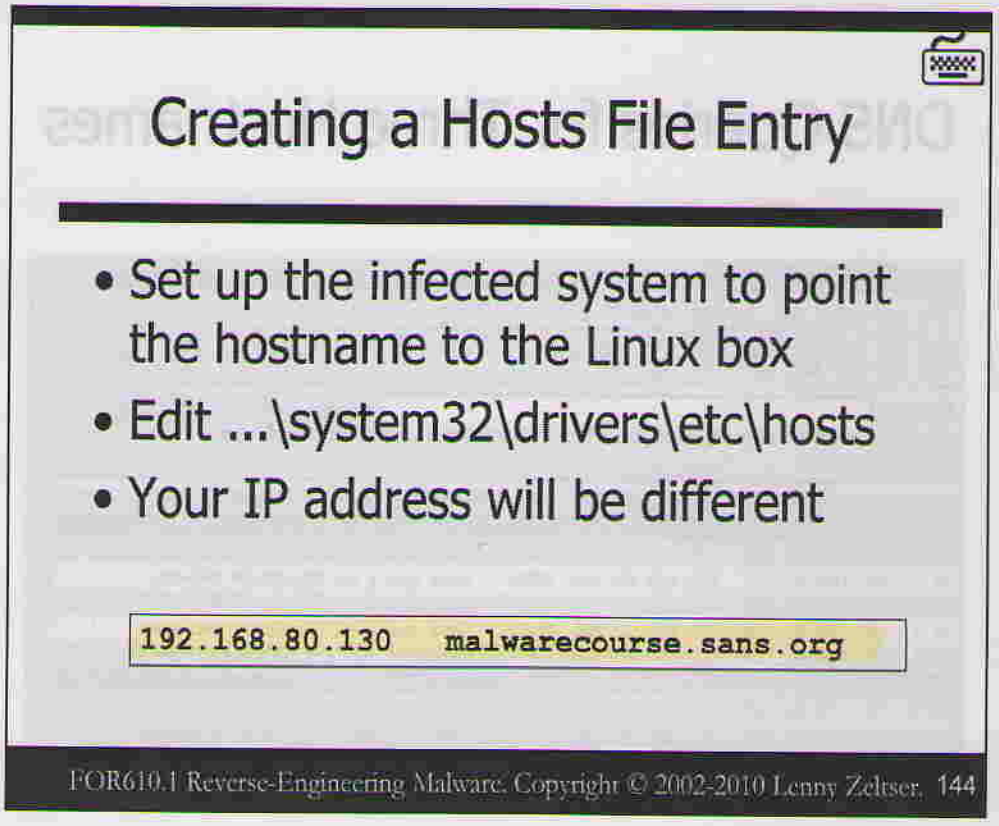


This slide shows an excerpt from the network sniffer log that includes DNS query packets issued by the malicious executable. Traces that you collected should include DNS packets that attempt to resolve the following three hostnames:

- irc.slim.org.au
- malwarecourse.sans.org
- sb.webhop.org

The trojan is using "malwarecourse.sans.org" because I customized the executable to use that hostname. If you encountered this program outside of this workshop, it would have had a name of a real system instead.

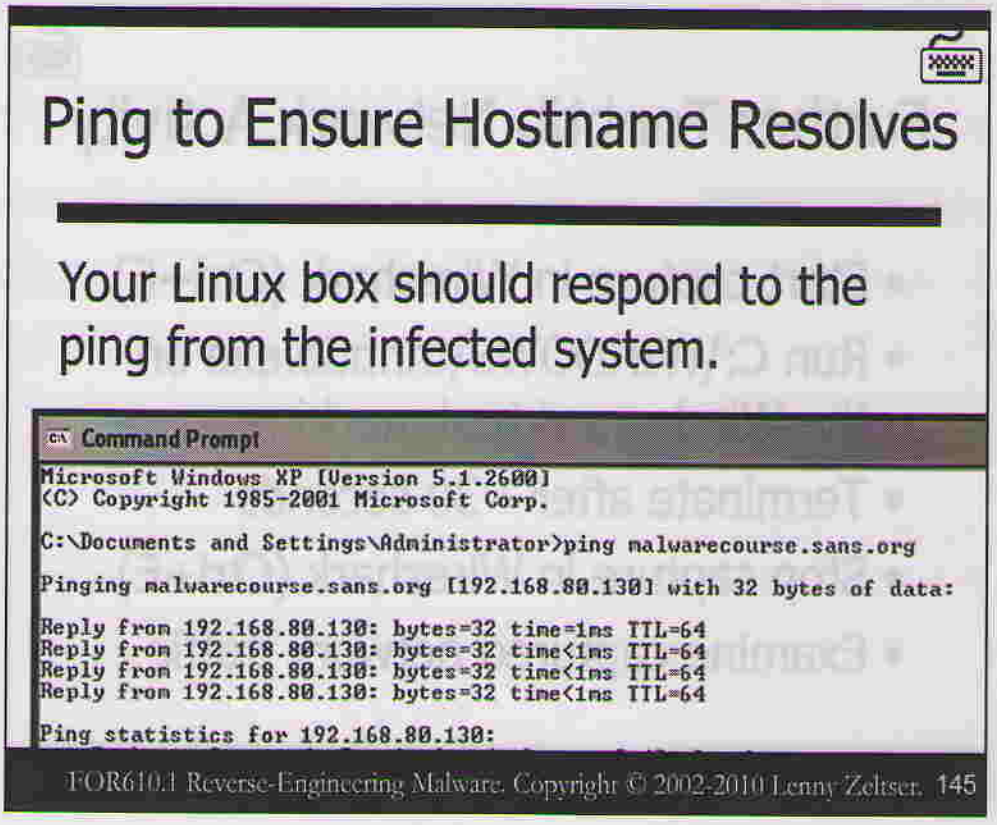
Let's continue the analysis determining why the program attempts resolving "malwarecourse.sans.org". You are probably curious about the other hostnames as well, but let's proceed one name at a time.



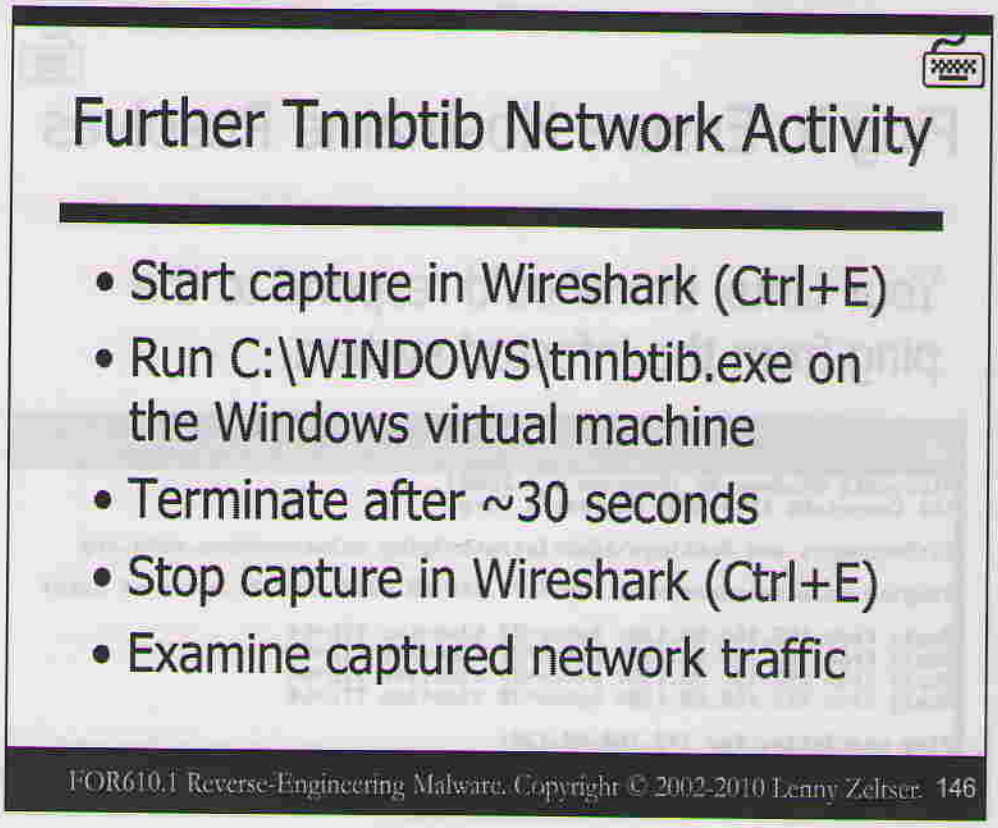
The technique for redirecting the infected system to the desired laboratory host is relatively simple, as you may recall from the first half of the course. You need to create an entry in the hosts file on the infected machine, pointing the hostname (malwarecourse.sans.org, in this case) to the IP address of the system that will pretend to be a server later in the experiment (your Linux box). The actual IP address of your Linux box will most likely be different than the one shown on this slide – the network address range for the virtual network is randomly chosen by VMware for each installation.

The hosts file does not exist by default on all Windows systems, so you may need to create one. On Windows XP and higher it should be "C:\WINDOWS\system32\drivers\etc\hosts".

Please note: When saving the file using Notepad, double-check to make sure Notepad did not add a .txt extension to the file when saving it. Rename the file if necessary. To avoid the problems you are likely to have with the file's extension, place the name of the file in quotes when saving it in Notepad.

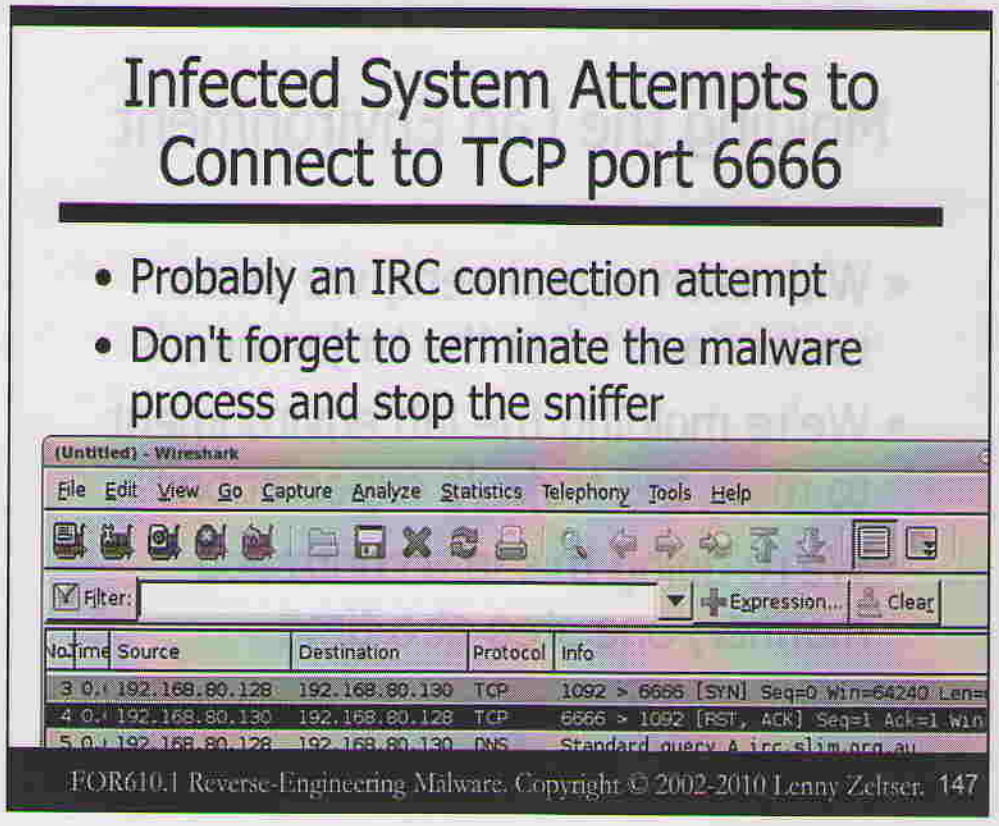


This slide shows a screenshot of the Linux box responding to a ping from the Windows virtual machine after the modifications to the hosts file.



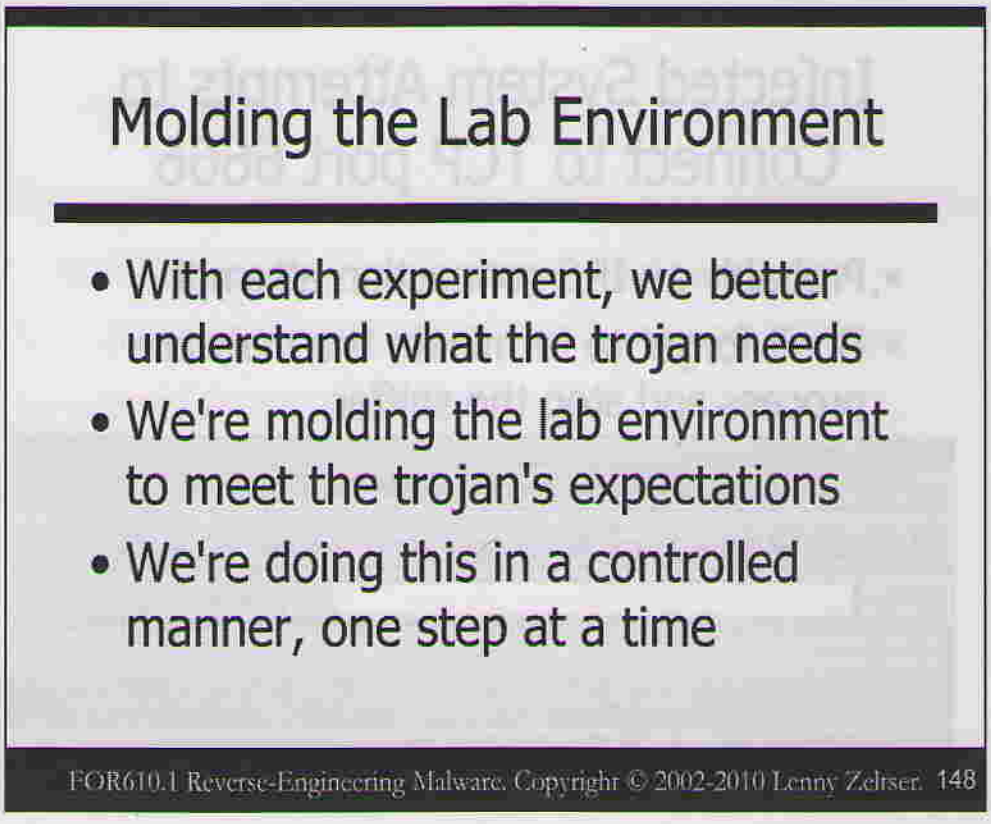
After modifying the hosts file on the infected system, start Wireshark and launch the tnnbtib.exe process again. Again, let the malicious process run for about half a minute—to give it some time to perform actions worth observing—then terminate it using Process Explorer.

Stop capture in Wireshark by either pressing Ctrl+E or by using the Capture > Stop menu option. Then, examine the packets captured by the network sniffer.

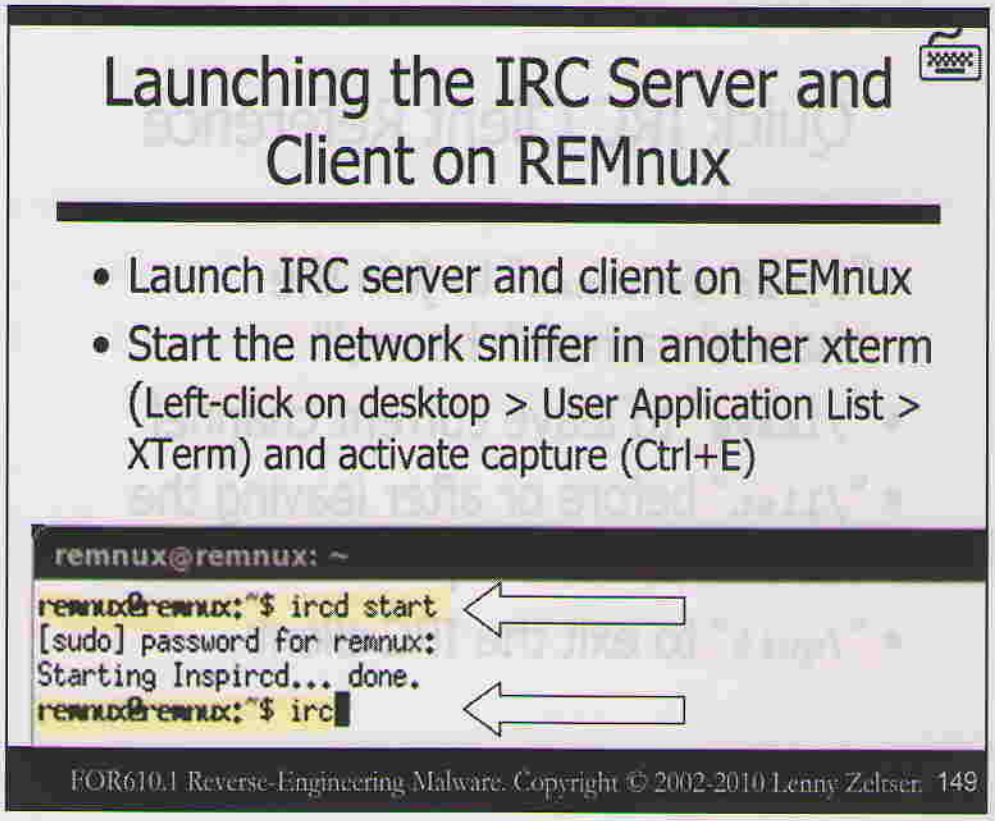


In sniffer logs you should see connection attempts to the Linux box's TCP port 6666, which is commonly used for IRC traffic. The malware specimen is probably trying to connect to the IRC service on malwarecourse.sans.org.

Now that we know this, we can adjust the laboratory configuration in a way that more closely resembles the malware specimen's expectations of its environment. Specifically, we will start an IRC server on the Linux box.



The analysis process that we're following allows us to gradually mold the laboratory environment as we discover new details about the malware specimen.

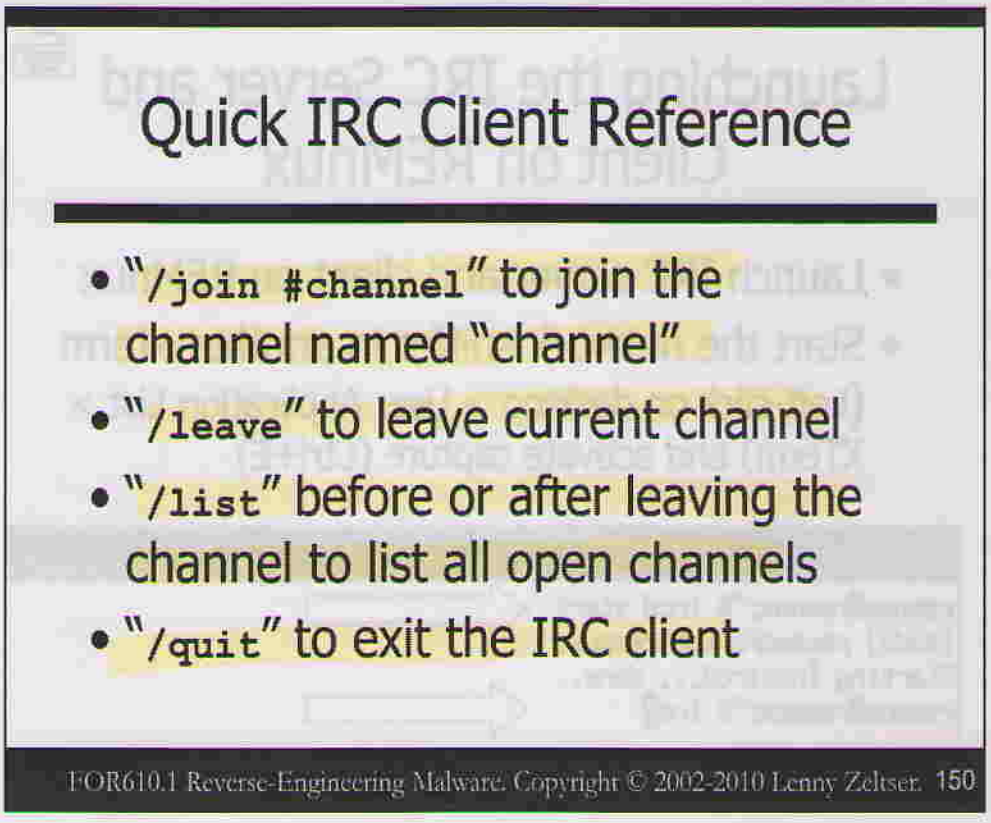


Conveniently, REMnux comes with an IRC server already installed. To start it, type "`ircd start`" at the command shell. If prompted for password, type "**`malware`**".

Once the IRC server is running, run type "**`irc`**" at the command shell to launch the IRC client installed on REMnux, which will automatically connect to the local IRC server. We will use this client later for interacting with the malicious program through IRC.

Once the IRC software is running, start the sniffer (so that you can observe how the program connects to the IRC server). The original xterm window will be busy with the IRC client running in it. You'll need to open another xterm in which you'll type "`wireshark`" to launch the sniffer. To open another xterm window, left-click with your mouse on the desktop of REMnux, then select User Application List > XTerm. Then, type "`wireshark`"; you may be prompted for the "`malware`" password.

Don't forget to begin capturing traffic in Wireshark (Ctrl+E).



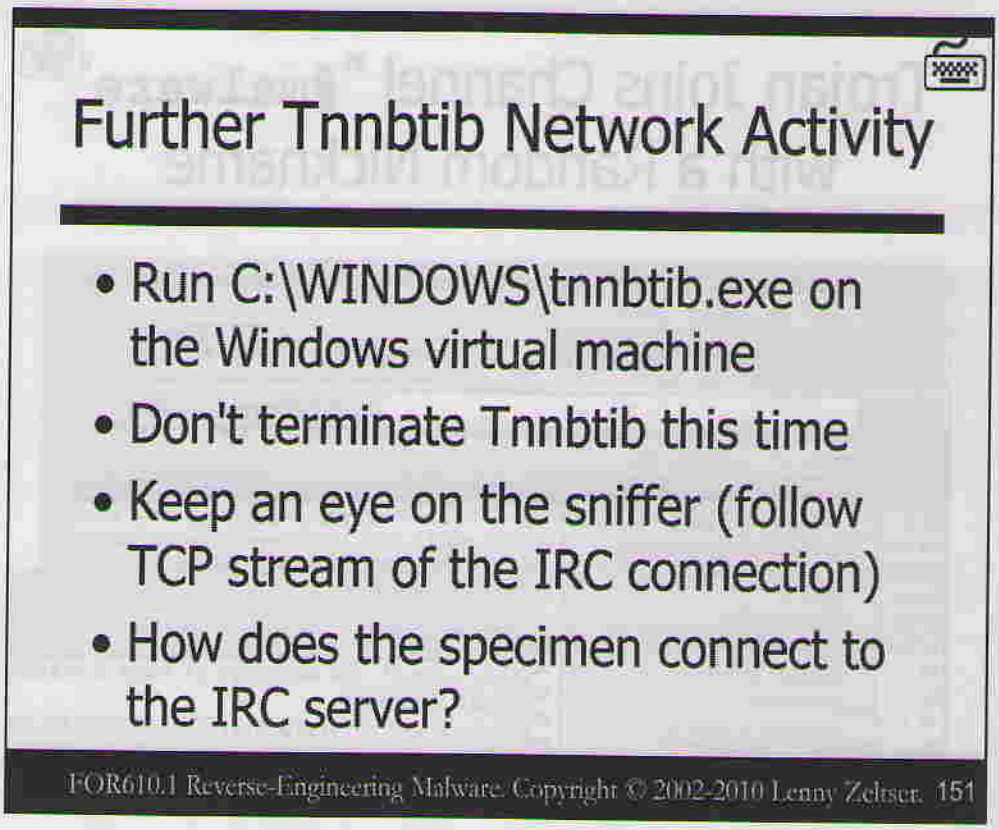
This slide shows some of the most useful commands you can enter when you're inside the IRC client installed on REMnux. Each command is preceded with "/". If you type something without a preceding "/", the IRC client will assume that you are typing text that you want to share with members of the current channel.

Once the IRC client is running, you can join the desired channel by using the "`/join`" command; for instance, to join the channel named "channel", you would type "`/join #channel`".

To leave the channel, type "`/leave`".

To see what channels are currently open, leave your current channel (if you are already in a channel), and type "`/list`".

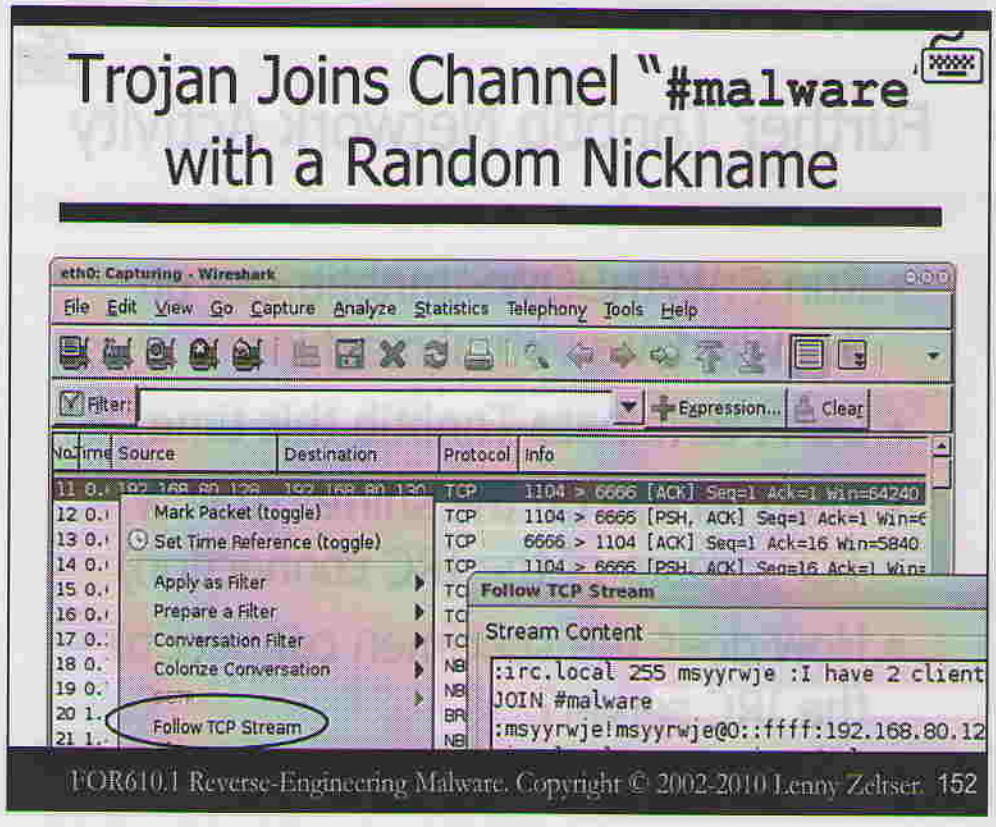
To quit exit the IRC client and return to the Unix shell, type "`/quit`".



This procedure should be already familiar to you.

Once Wireshark is running, run tnnbtib.exe, and keep an eye on the sniffer's logs.

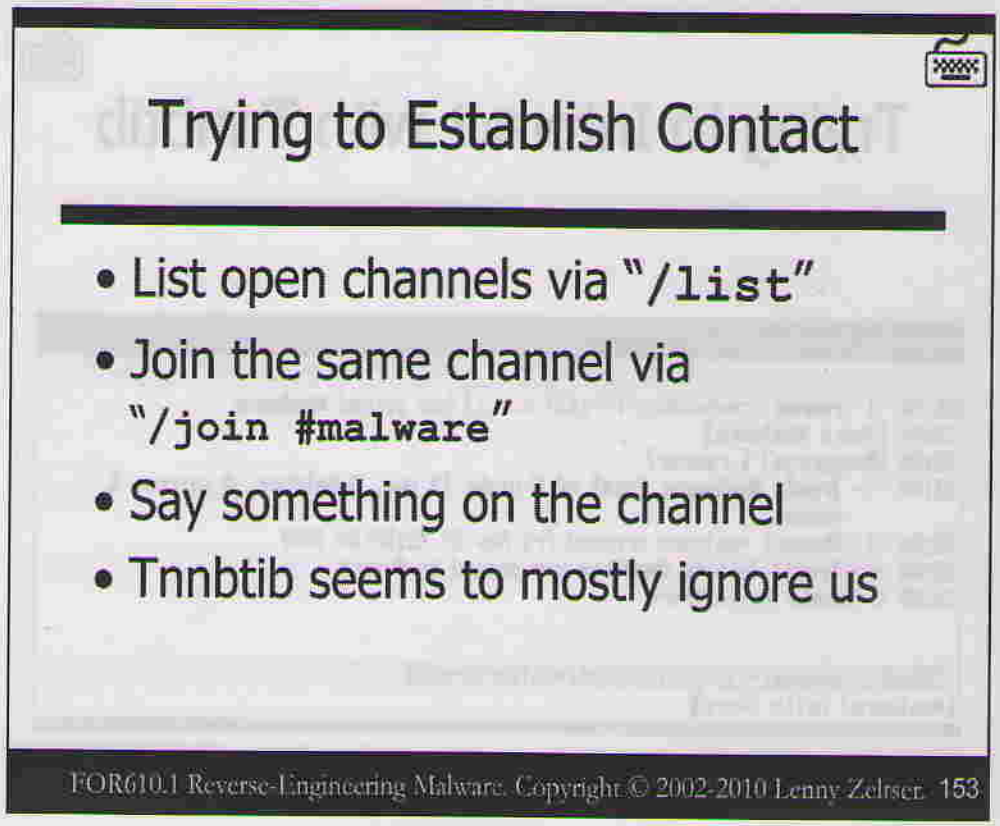
In this experiment, don't terminate the tnnbtib.exe process, because later you'll attempt to interact with it using your IRC client.



Your sniffer logs should reveal that the program connected to a channel "**`malware`**" with a nickname that seems to be randomly generated.

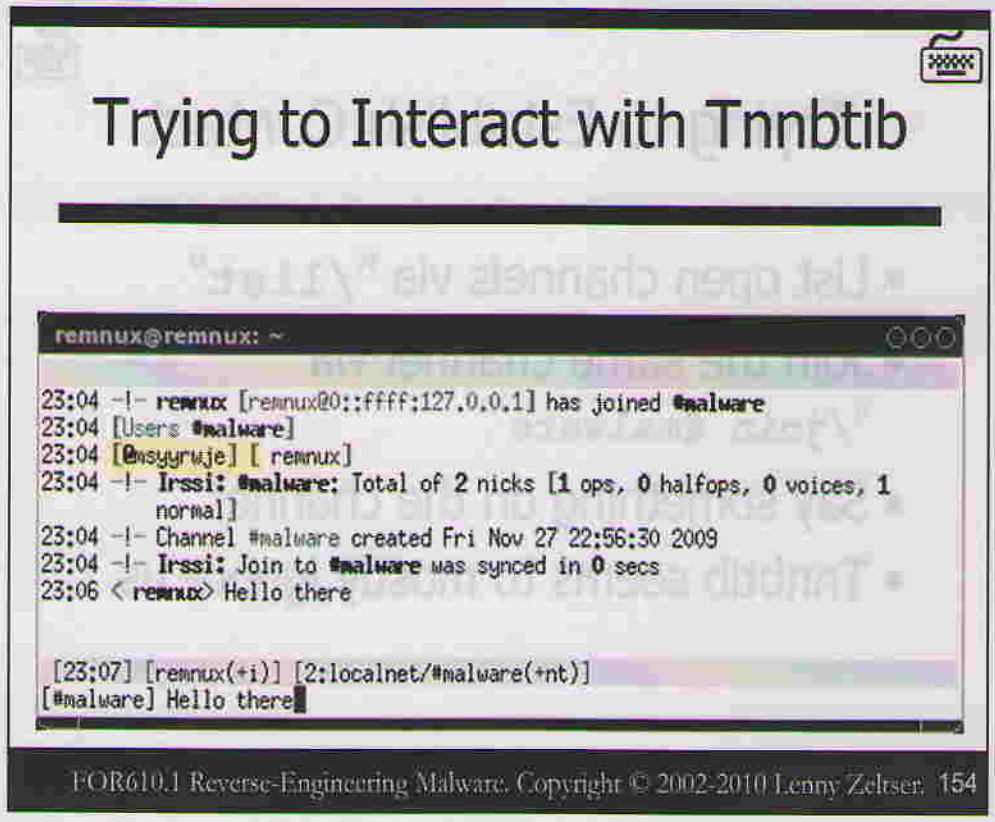
To see that, locate one of the IRC session's packets in Wireshark, then right-click on it and select "Follow TCP Stream". The payload of the packets should include the string "`JOIN #malware`", as well as other properties of the IRC session.

Now that you know the channel name, you can join it yourself…

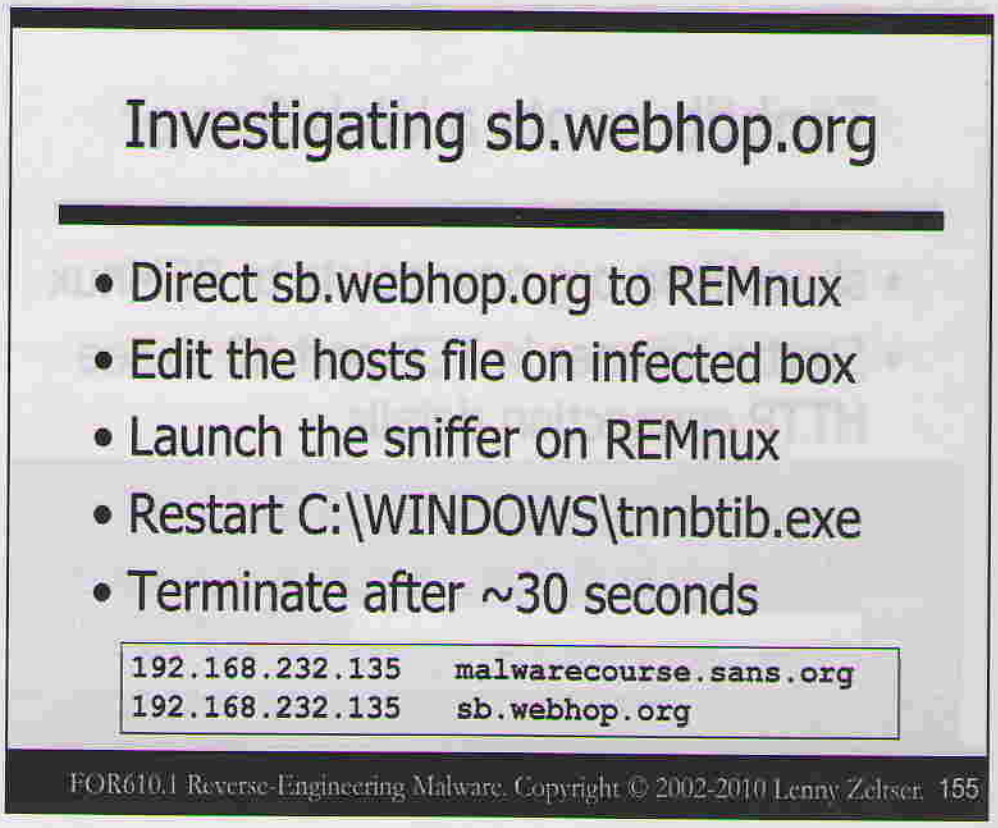


You can determine which channel name the specimen wants to use without employing a network sniffer. To do this, type the command "`/list`" in the IRC client. If the trojan connected to the IRC server and joined the channel, the channel's name will show up on the list, along with the number of users on that channel.

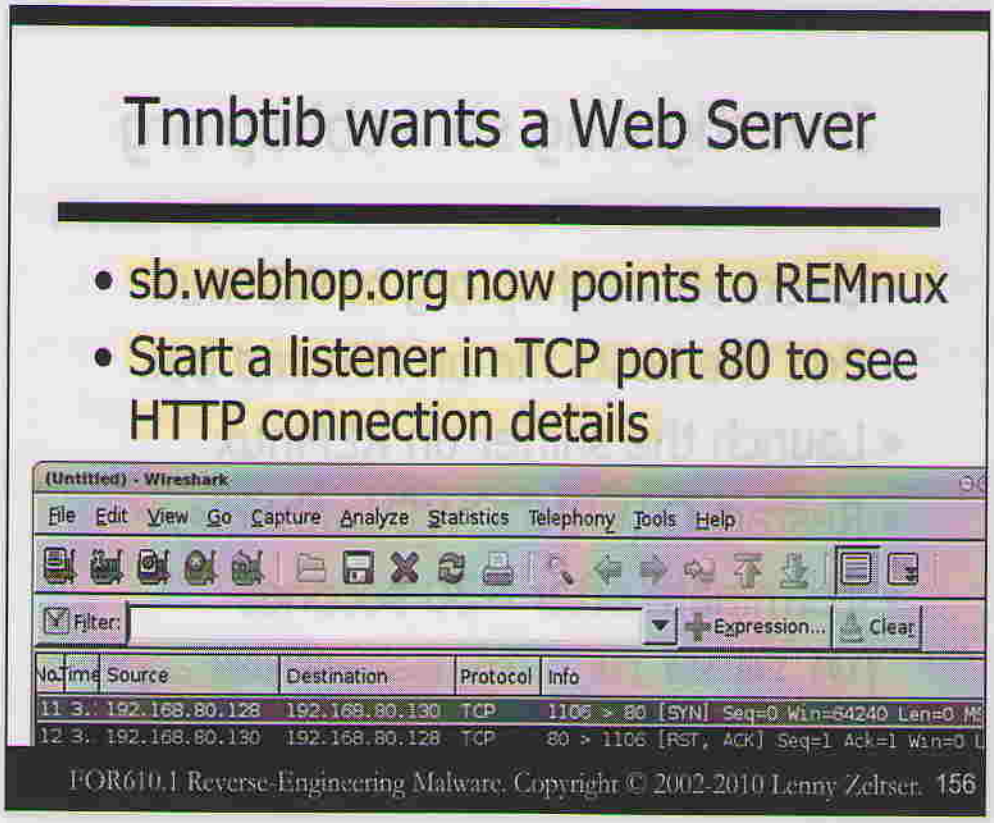
To join the channel, type "`/join #malware`" in the IRC client. Try saying something, just to see if the malicious executable will respond.



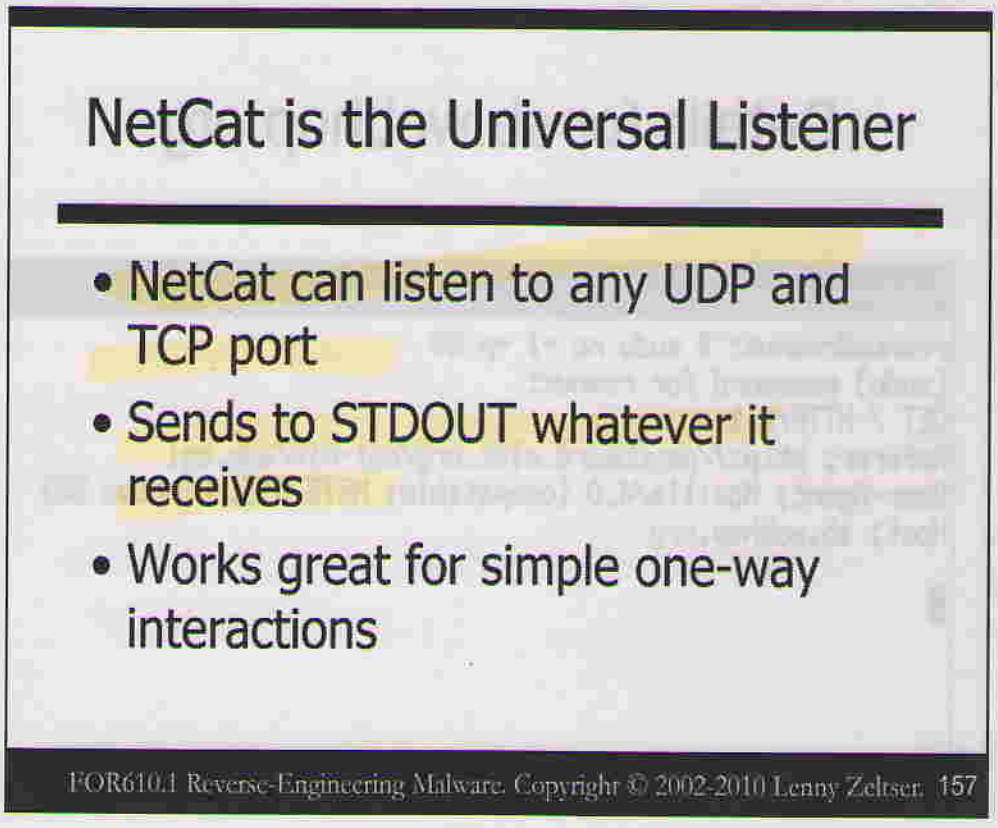
This slide shows the screenshot of me trying to interact with the malicious process via the IRC session. As soon as I joined the channel via "`/join #malware`", the IRC client displayed to me the users on the channel. There were two: I was the user "`remnux`". The other user was "`msyyrwje`"—that was the IRC user representing the infected system.



Now that we know how to get the malicious executable to join a IRC channel on malwarecourse.sans.org, let's direct our attention to sb.webhop.org, whose hostname Tnnbtib tried to resolve.

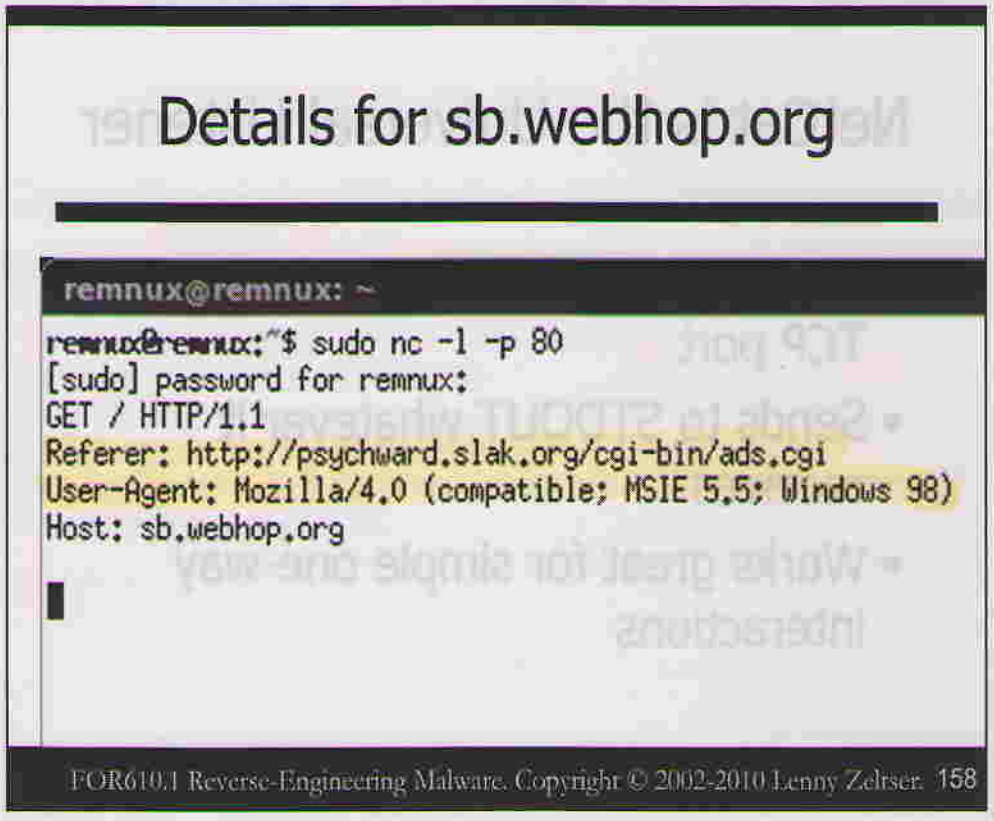


The network sniffer reveals that Tnnbtib is attempting to reach a Web server. If it wants a Web server, we'll give it a Web server.



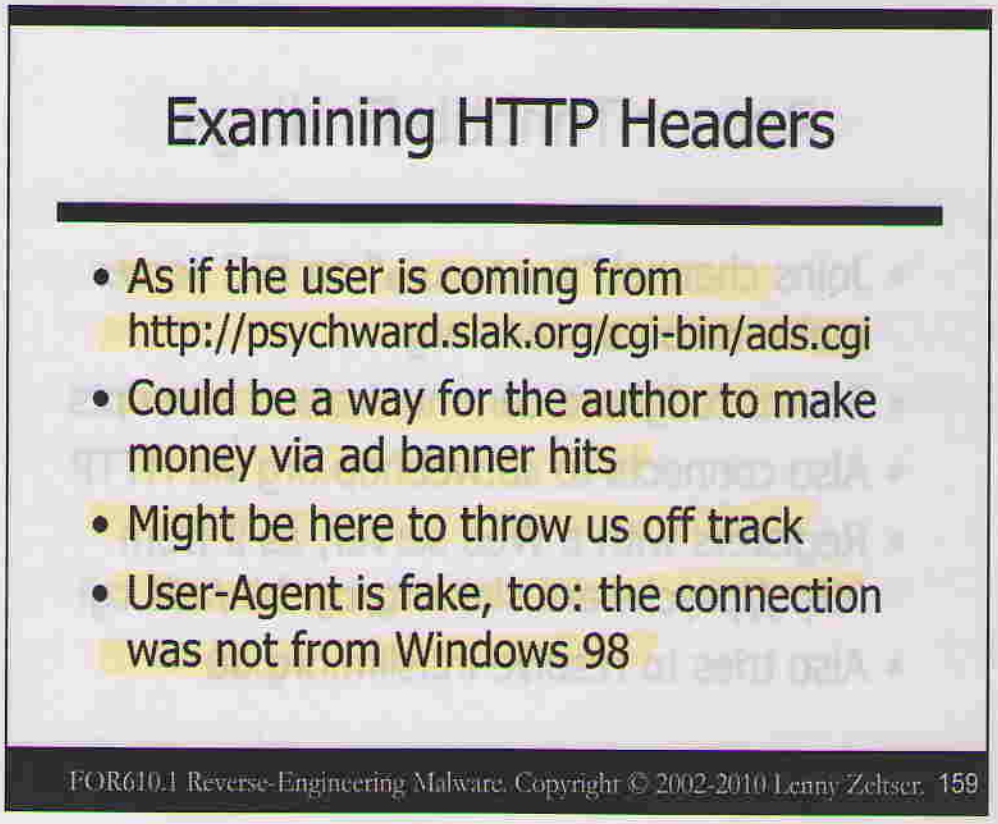
Instead of using a full-blown Web server to intercept connections to TCP port 80, we can use NetCat. NetCat is a simple but flexible tool for initiating and accepting network connections. When running in listening mode, NetCat accepts connections on the specified port, accepts the remote host's data, and prints it to standard output. So, to accept connections on TCP port 80, we would run NetCat like this: "`sudo nc -p 80 -l -n`". The "`-p`" parameter specifies the port, "`-l`" puts NetCat into listening mode, and "`-n`" tells NetCan not to attempt resolving IP addresses to their hostnames.

NetCat can come in very handy when you need to listen on an arbitrary port, and don't have an application handy that is supposed to listen on that port. NetCat is great for simple text-based protocols such as HTTP or SMTP. However, if the malicious executable uses an interactive protocol that requires the server to respond in some way (for instance IRC), you are usually better off looking for a native application that would listen on that port. Even with HTTP, if the malicious executable attempts to retrieve or execute a particular file on a remote Web server, it is much easier to present an expected response with a real web server, since you can prepare the server to have appropriately named files or scripts.



Listening on TCP port 80 requires root privileges. Therefore, we need to launch NetCat using sudo: "`sudo -l -p 80`". Sudo may prompt you for the root password ("`malware`").

The screenshot on this slide shows HTTP headers captured by the NetCat listener after Tnnbtib connected to it, thinking it's connecting to sb.webhop.org.



Tnnbtib attempted to retrieve contents of "/" on the Web server. Sniffer logs also show that the HTTP request was crafted to look as if the user is coming from http://psychward.slak.org/cgi-bin/ads.cgi.

There are several possible explanations for the crafted Referrer string. It is possible that the author of Tnnbtib wants to increment an ad hit counter on the server to which sb.webhop.org redirects its visitors. The more machines are infected with Tnnbtib, the more ad hits are registered with the ad hit counter, and (presumably) the more money the author is paid for "effective" advertisements.

Alternatively, this may be a way for Tnnbtib to register the infected machine's IP address with the author. The Referrer field could be simply a red herring, designed to confuse the analyst. The Referrer field could also be a way to uniquely identify a particular version of Tnnbtib, if the author uses this Web server to keep track of different versions of his or her creations.

Yet another explanation could be that this packet captures Tnnbtib's attempt to update itself, by downloading the latest version of the executable, or a supplementary file, from sb.webhop.org.

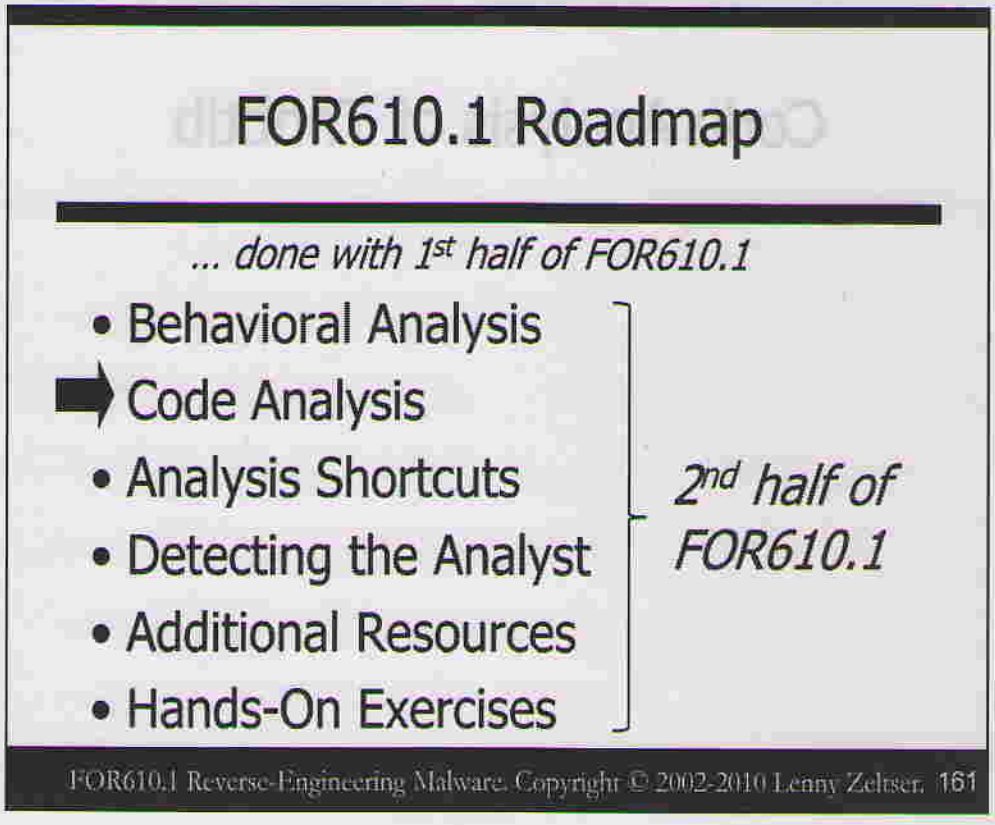
Recent Tnnbtib Findings

- Joins channel "`#malware`" on IRC server malwarecourse.sans.org
- Seems to ignore communication attempts
- Also connects to sb.webhop.org via HTTP
- Registers with a Web server, as if from http://psychward.slak.org/cgi-bin/ads.cgi
- Also tries to resolve irc.slim.org.au

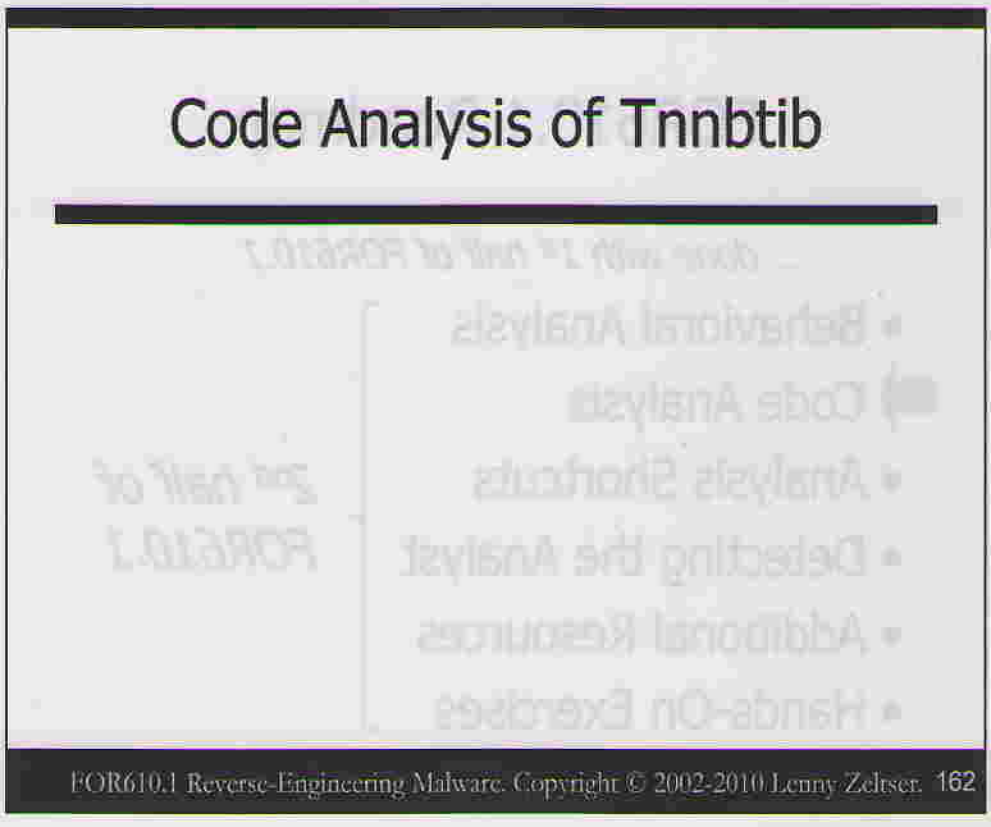
Let's summarize what we've learned about the malware specimen by examining its interactions with network resources. We know that Tnnbtib connects to an IRC server "malwarecourse.sans.org" with a randomly selected nickname, and joins a channel "#malware". Tnnbtib presumably can accept commands typed by the operator on the IRC channel; however, we were unable to evoke a response by simply typing text in the channel.

The executable also connects to the sb.webhop.org Web server and uses a crafted HTTP request to access "/" on that server.

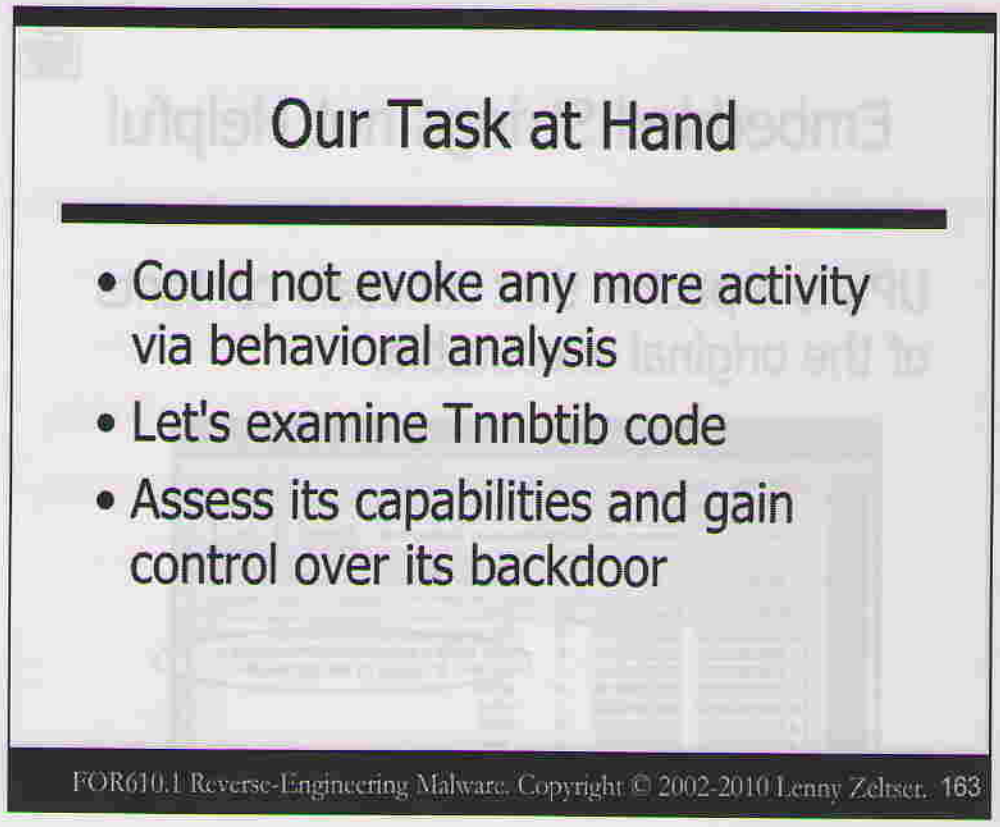
We also observed that Tnnbtib attempted to DNS-resolve "irc.slim.org.au." We did not examine this aspect of the program's behavior, because I wanted to leave this for later.



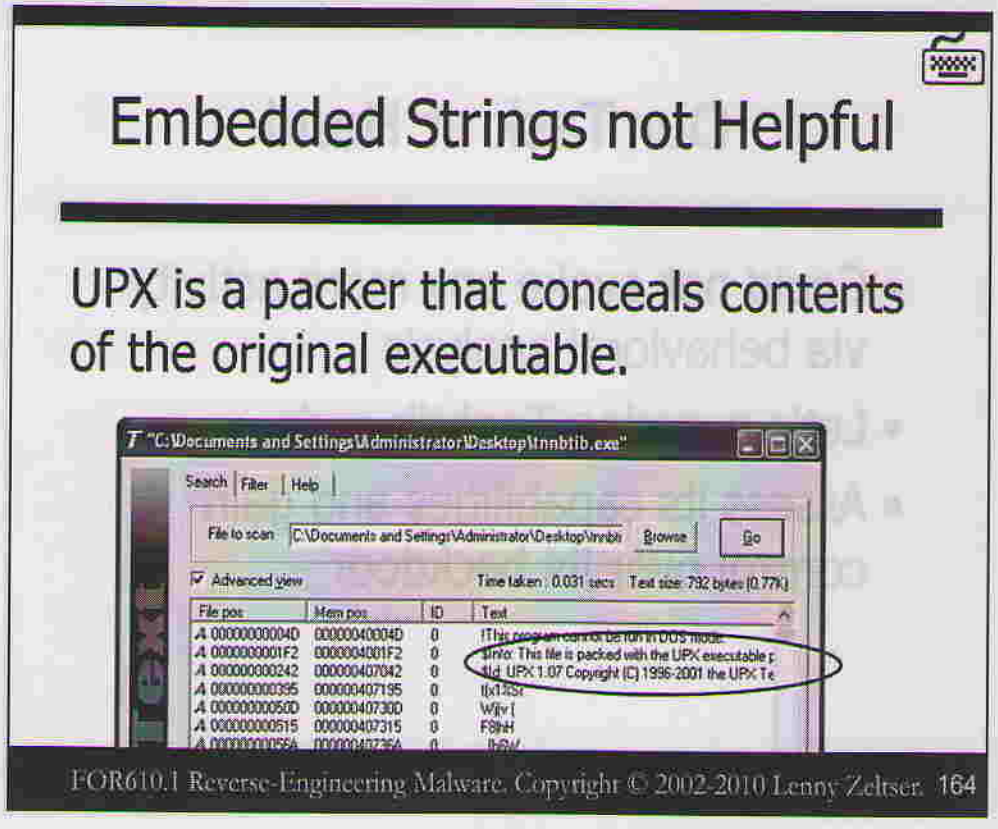
Code analysis of Tnnbtib is next!



We're now ready to examine the code of Tnnbtib, in an attempt to understand what commands it accepts, how to communicate with it, and what is its threat capacity.

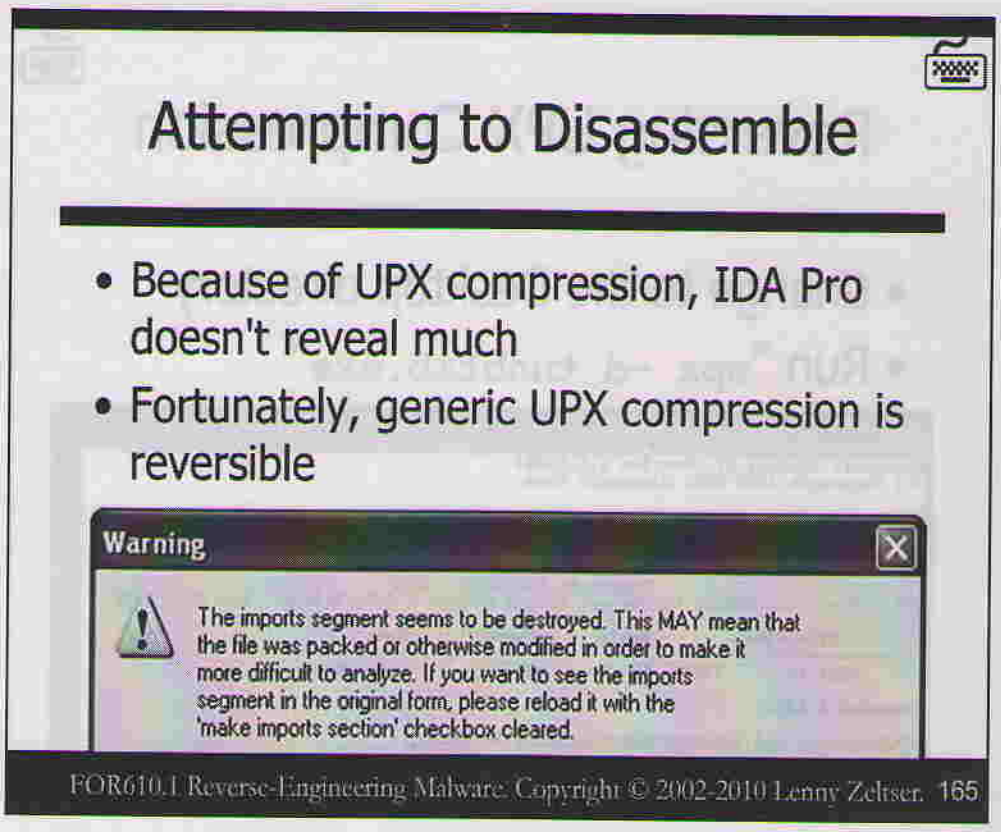


Since we couldn't evoke any more activity from Tnnbtib using behavioral techniques, we know this is a good time to start the code analysis phase.

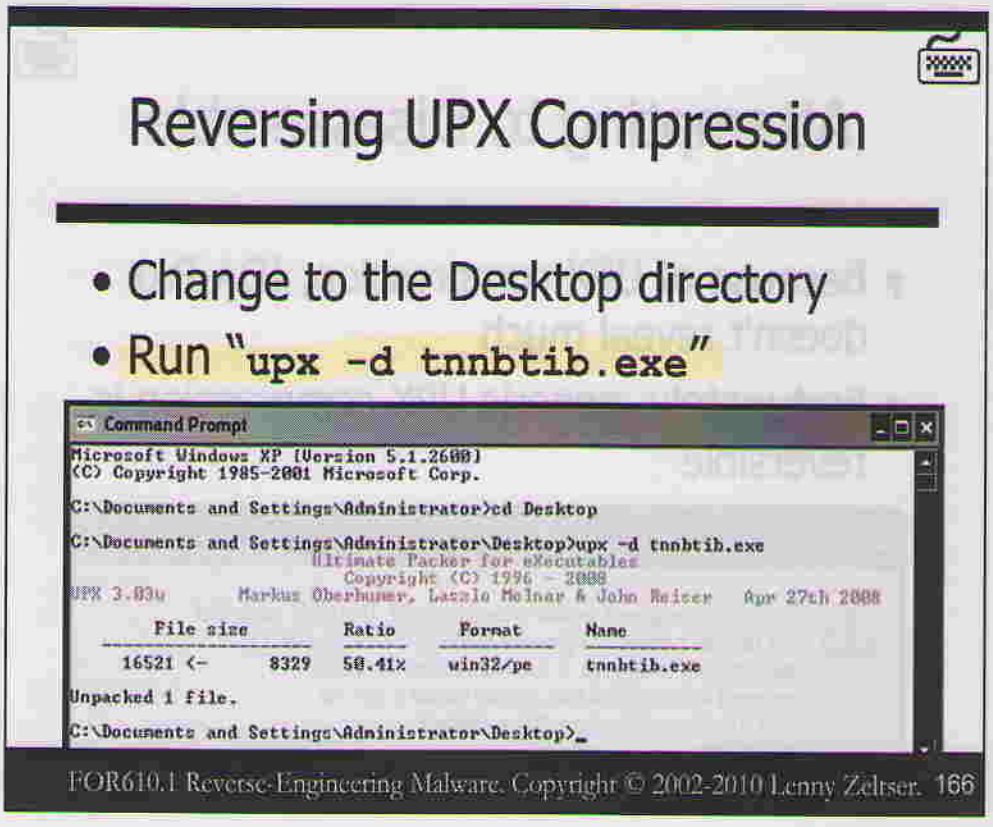


One of the most common ways of starting an analysis is to take a quick look at the strings embedded into the executable. This may reveal file names, hosts names, or registry keys that the program may attempt to access, and can help focus subsequent steps of the investigation. Of course, we cannot trust all the strings embedded into the program because they might have been embedded there to throw us off.

Looking at Tnnbtib using BinText, we don't see many legible strings embedded into the executable. The possible explanation for that is the presence of the UPX identifier, presented on this slide, which suggests that the executable has been compressed with UPX, a tool frequently used for this purpose.



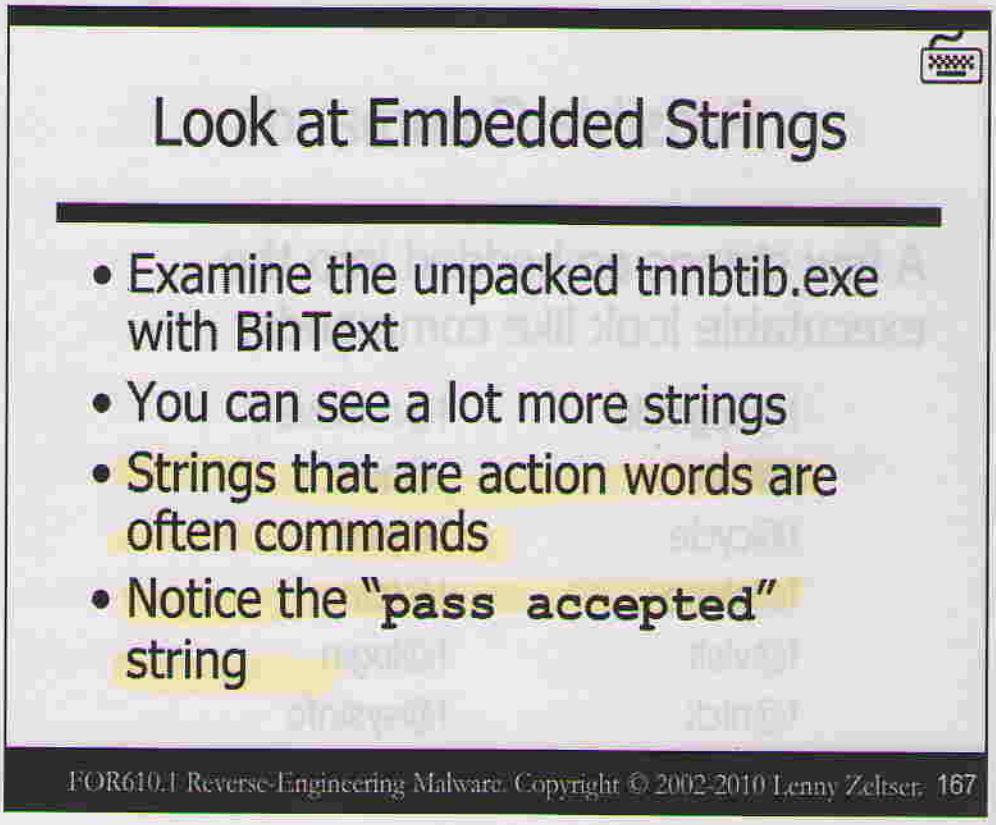
BinText output included strings that seemed to be contents of a UPX header, which suggests that the executable was packed with UPX—a tool frequently used to protect malicious executables. If you try loading tnnbtib.exe into IDA Pro, you will see an error like the one shown on this slide. IDA Pro will be able to show you some disassembled instructions, but most of the file will be shown as a data segment, without visible assembly instructions.



Fortunately, generic UPX compression is reversible using the very same tool that compressed the original executable. The DVD you received contains a copy of upx.exe. It is also installed on REMnux. You can also download it for yourself from http://www.upx.org.

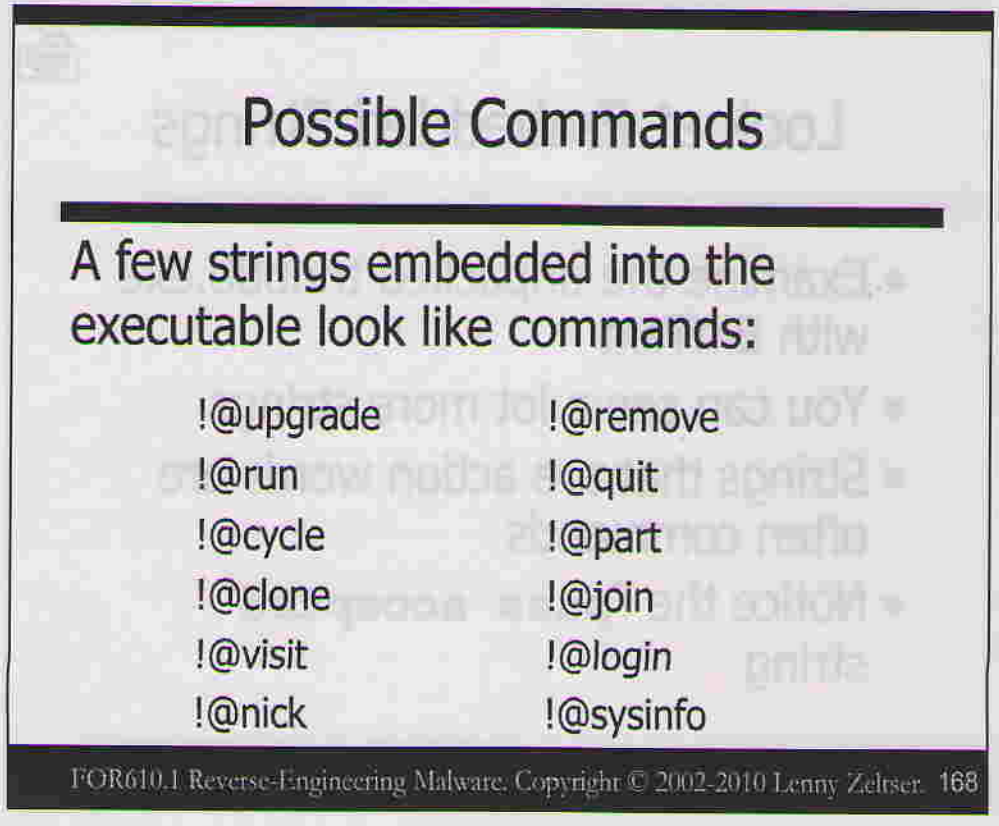
To unpack (decompress) tnnbtib.exe, open the command prompt. Then change to the directory where you saved tnnbtib.exe (probably your Desktop). Then run the command "`upx -d tnnbtib.exe`". If upx.exe responds with "Unpacked 1 file", as shown on the screenshot on this slide, it succeeded at reversing the compression.

Note that the upx.exe utility decompresses executables on the spot, overwriting the original file. Therefore, it is important to retain the copy of the original file, should you need it later during your analysis in an unadulterated form. (In this case, the original copy of tnnbtib.exe is in the tnnbtib.zip file on your DVD in \Malware\day1.)

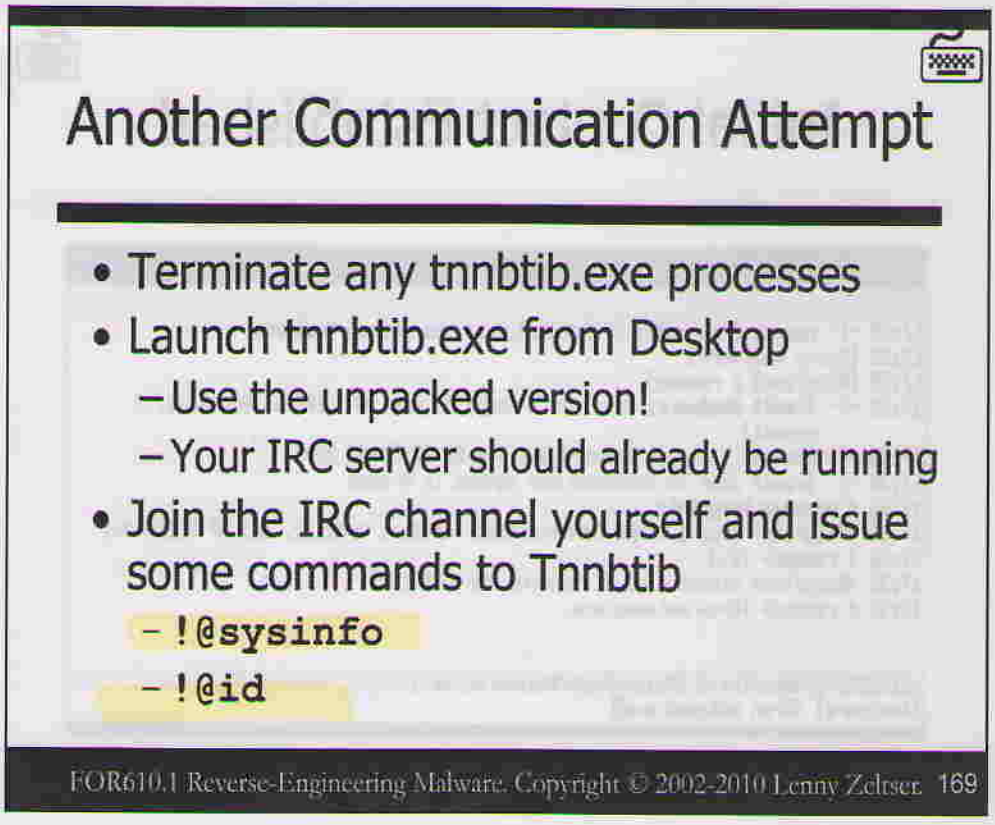


Use BinText to look at strings embedded into the unpacked version of tnnbtib.exe. You should be able to see a lot of meaningful strings. Pay particular attention to those that look like they may be commands.

Also, notice that one of the strings is "`pass accepted`". This suggests that the bot possesses a password-based authentication system, and may require valid login credentials before accepting commands.



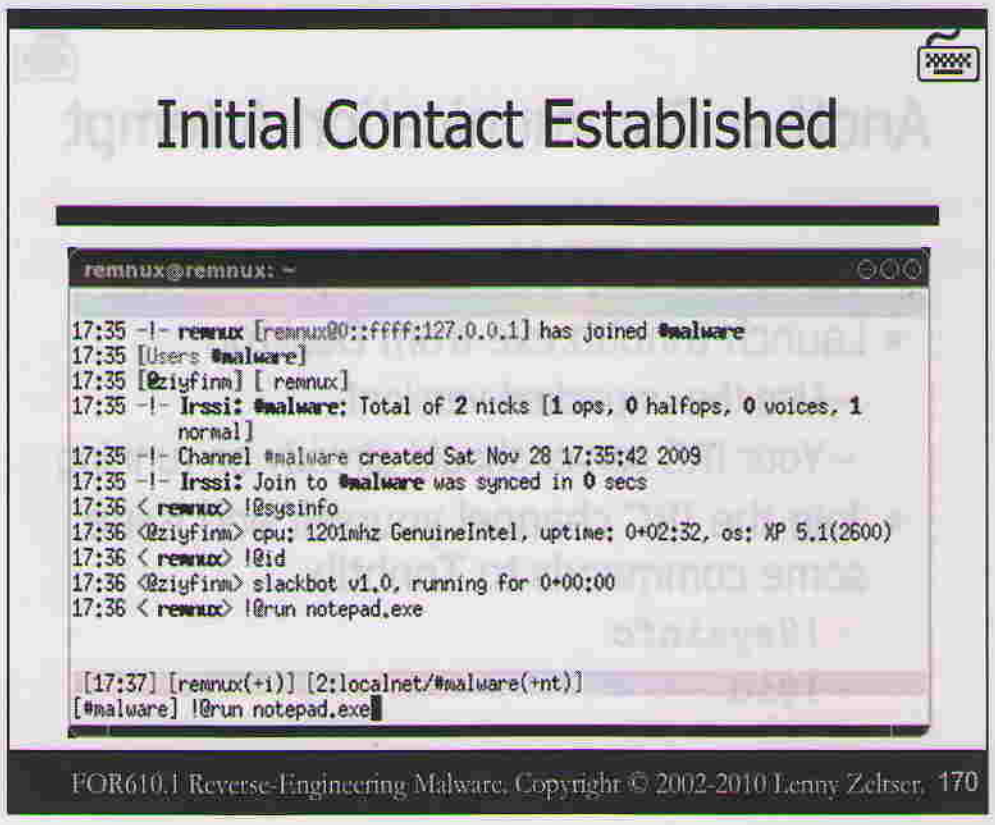
This slide lists a few strings embedded into the executable that look like commands; there are a number of other strings similar to these that are not listed on the slide. What capabilities do these commands offer the bot's operator?



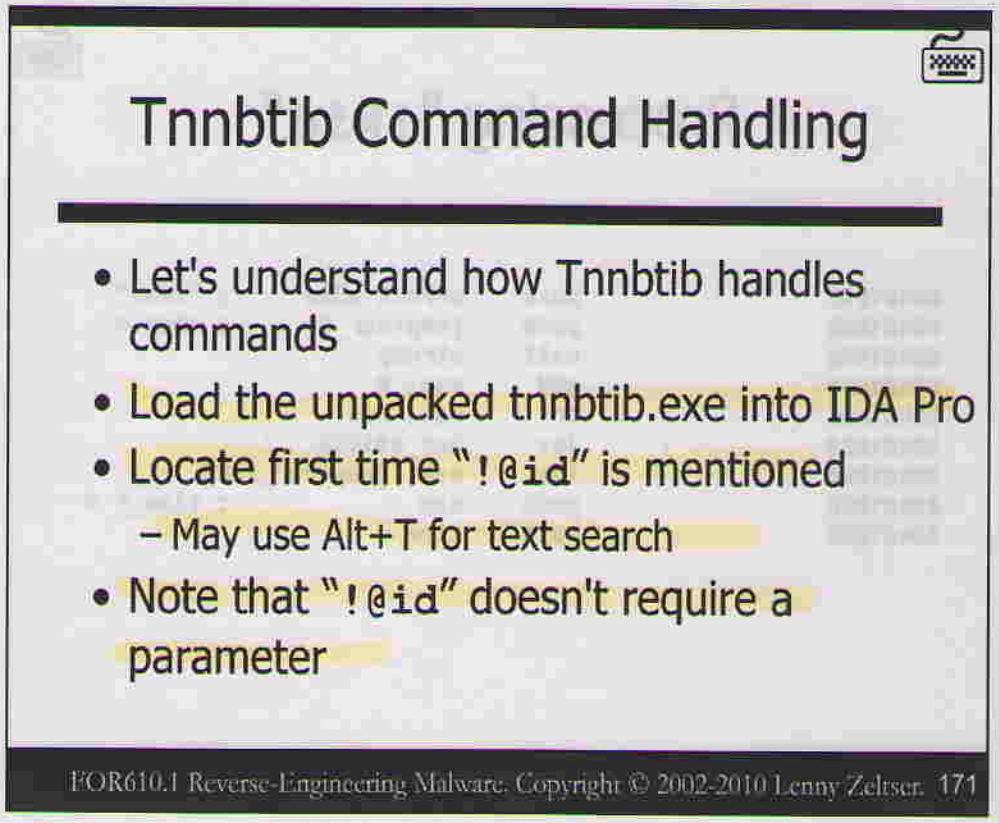
Now that you know a few commands that the trojan may accept, try communicating with it. To do that, first ensure that no existing instance of tnnbtib.exe is already running on your system. Terminate any such instance using Process Explorer or Task Manager, if necessary. Then, infect your system with the version of tnnbtib.exe that you unpacked (decompressed) using upx.exe. If that version of the file is located on your Desktop, simply double-click on tnnbtib.exe. It will copy itself to the C:\WINDOWS directory, overwriting the earlier (packed) version of tnnbtib.exe there. (Note that the file won't be overwritten if you forget to terminate existing instances of the tnnbtib.exe process, which would be locking that file.)

After launching the malicious executable, let it join the IRC channel, and join the channel yourself. As one of the commands, type "`!@sysinfo`" – you should receive a response from the bot. Try submitting a few other commands and notice which of them are accepted, and which are ignored.

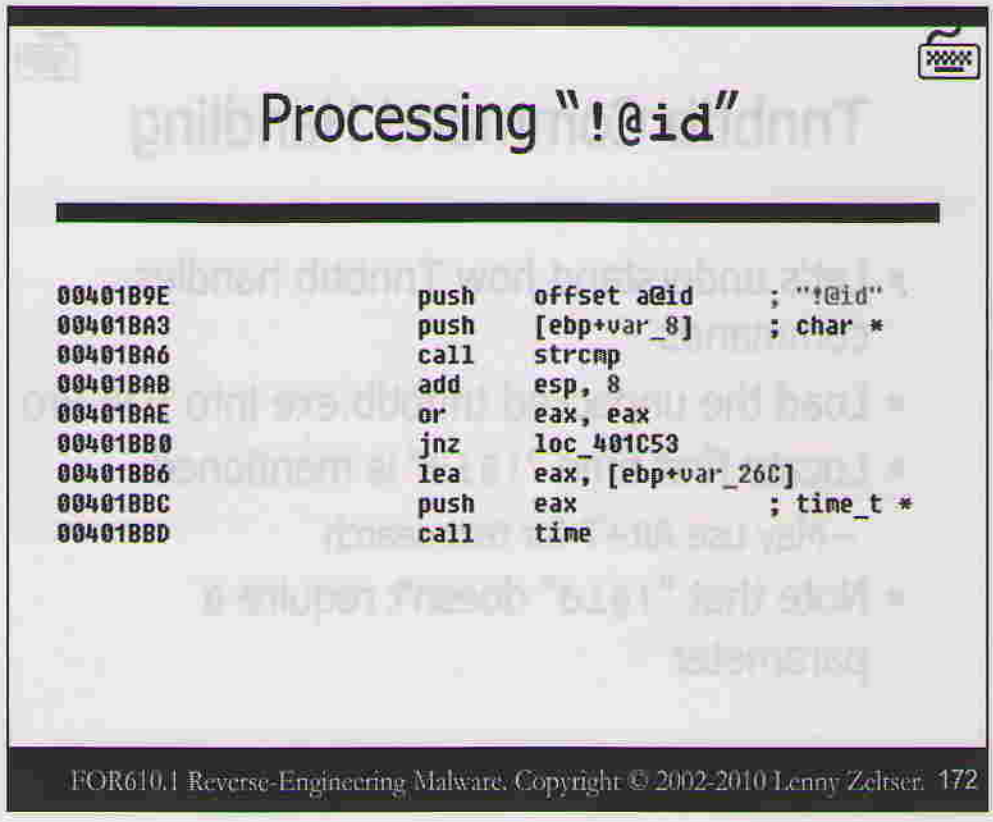
Reminder: If you don't already have the IRC server running, you can start it (in the background) by typing "`ircd start`" at the command shell in REMnux. If you don't already have the IRC client running, you can start it by typing "`irc`" at the command shell in REMnux.



This slide shows a screenshot of an IRC session that documents a few attempts to communicate with Tnnbtib. Notice that the program responded to "`!@sysinfo`" and "`!@id`" commands. However, when I attempted to launch Notepad on the infected system using "`!@run notepad.exe`", Tnnbtib quietly ignored me. Either my syntax for the "`!@run`" command is incorrect, or this command requires authentication.



Let's take a close look at the bot's assembly code to better understand how it handles commands, with the goal of eventually understanding the authentication mechanism. Please begin by loading the unpacked version of tnnbtib.exe into IDA Pro, and locating the first time that the "`!@id`" string is mentioned in the code. (Use Alt+T to perform the search.) Note that, as we discuss on the next slide, this command doesn't seem to require a parameter.



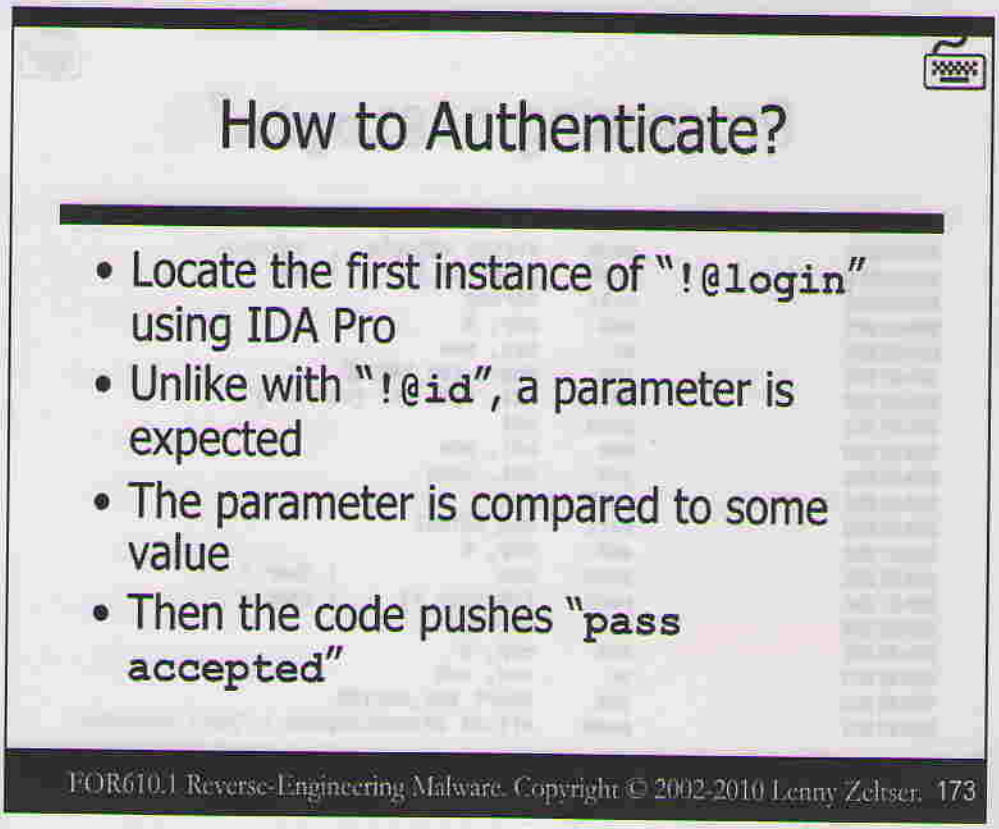
The screenshot on this slide shows IDA Pro in action, displaying the portion of the malware-unupx.exe code that processes the "!@id" command. Let's go over this code segment, to get familiar with assembly and the way IDA Pro presents it.

The first two lines set parameters for the "strcmp" call. First, the string "!@id" is pushed to the stack, then another value is pushed to the stack, then the "strcmp" call is made. Most likely, these lines are used to compare the supplied command to "!@id". The "strcmp" function is external to the malicious executable, and returns 0 if the two strings are identical.

The fourth line ads 8 to the ESP register to point the stack pointer to the next unoccupied memory space. The next instruction executes "EAX or EAX"–this is meant to provide a value to the following "jnz" instruction, and will result in 1 if EAX is 1, and will result in 0 if EAX is 0.

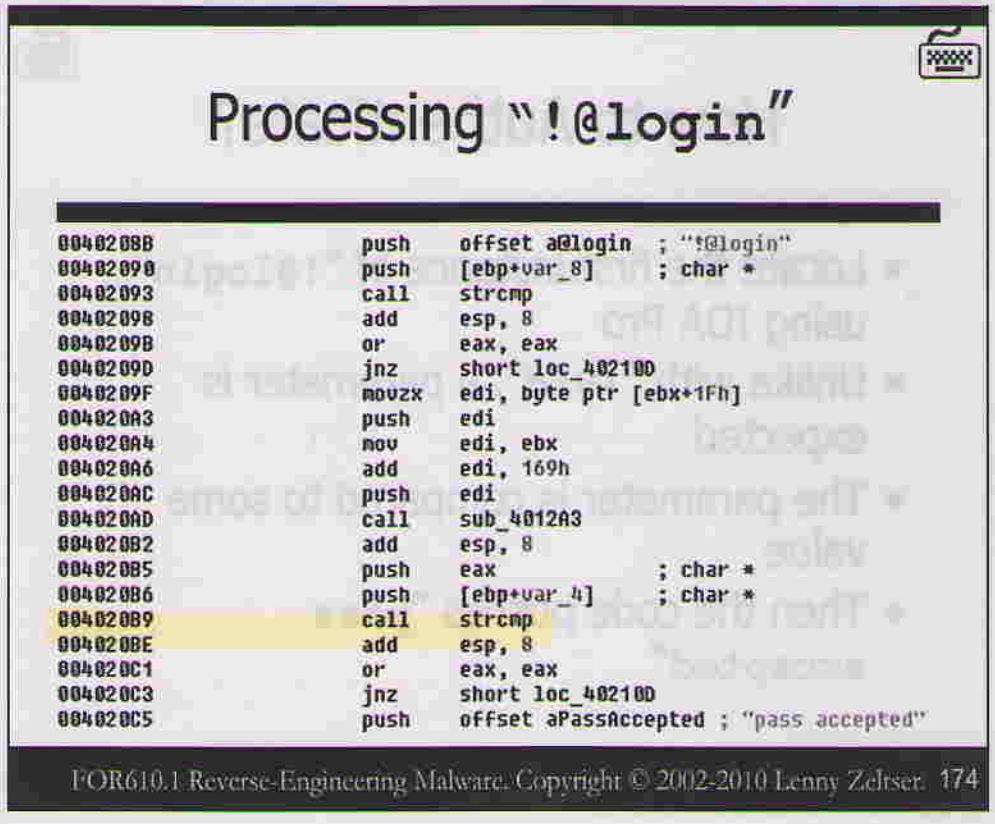
The sixth line executes a jump to another portion of the code, depending whether the value of the EAX register is 0 or not. (The jump will be performed if the previous instruction resulted in a non-zero value.) Ultimately, this depends on the results of the earlier comparison operation. If the comparison resulted in 0, the jump will not be performed, and the executable will proceed by executing the "time" call. (If you remember, the bot responds to the "!@id" command by calculating how long it has been running for; that's why it needs to determine the current time.)

If the comparison resulted in 1, then the supplied command was not "!@id", and the program jumps to comparing the supplied command to another possible value.

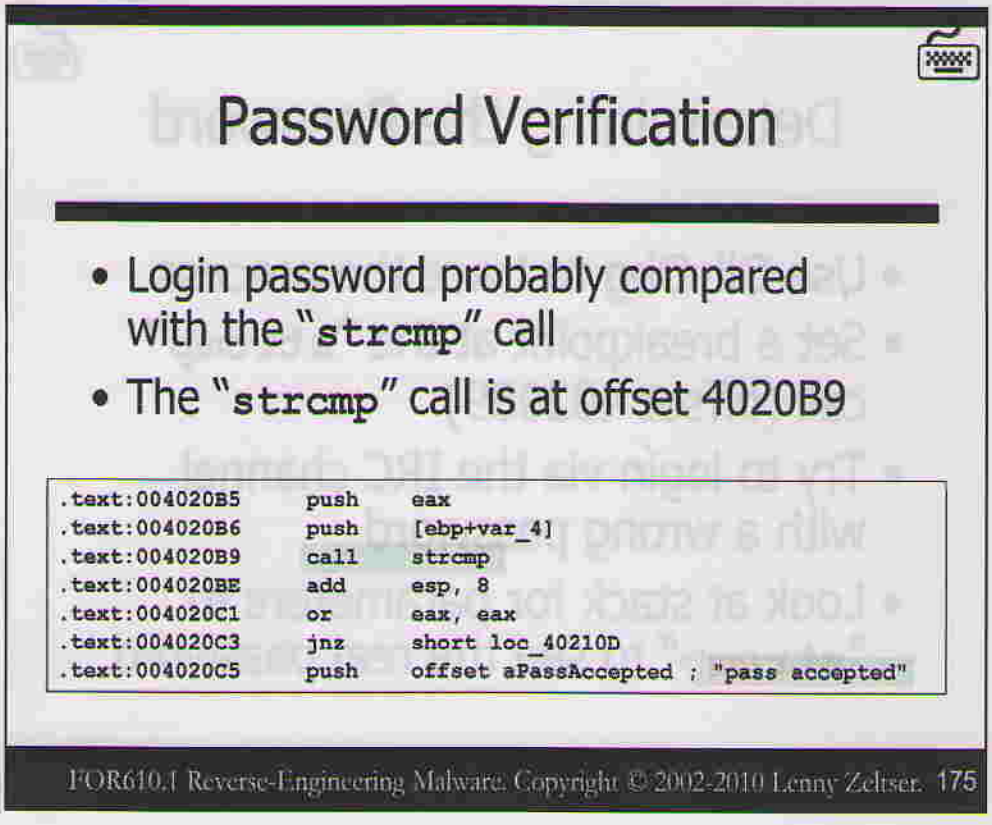


Now let's take a look at the way the program handles the "`!@login`" command, and compare it to how "`!@id`" is processed. Use Alt+T to locate the first instance of "`!@login`" in the code.

You may notice that, as we discuss on the next slide, the program seems to require a parameter to the "`!@login`" command, presumably a password. Note that after processing the parameter the program pushes the "`pass accepted`" string onto the stack.

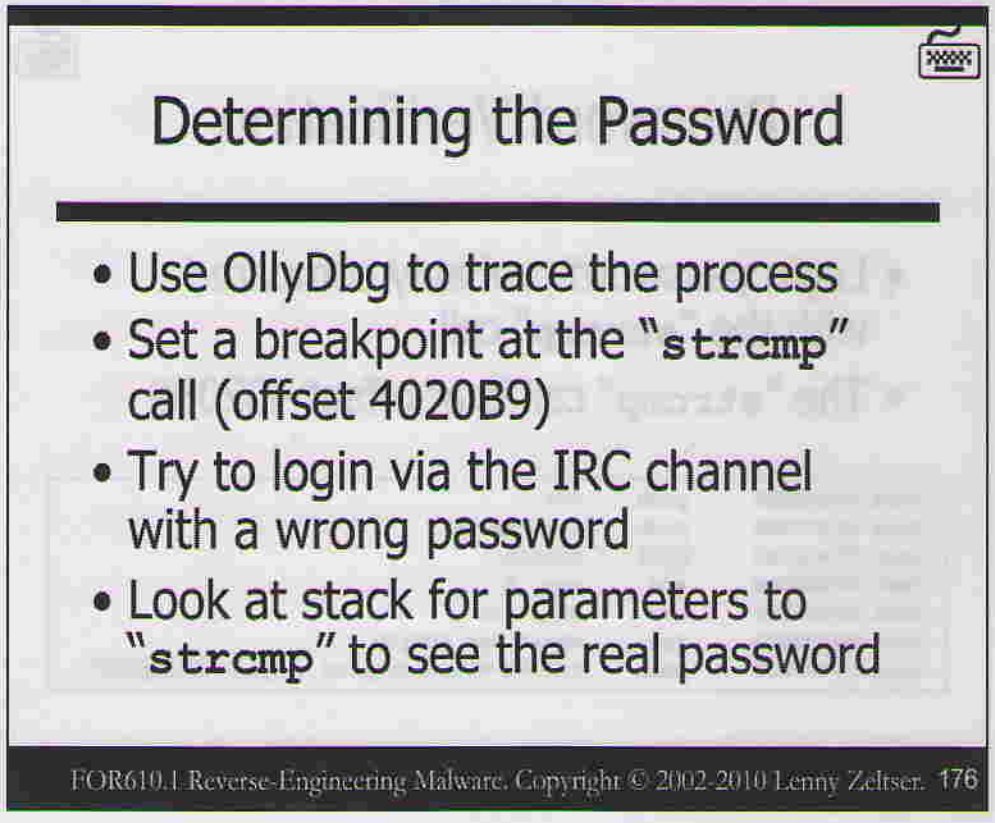


This section of the code is similar to the one that processed the "`!@id`" command. It begins by comparing the value of the supplied command to the string "`!@login`". Notice, that unlike in the "`!@id`" processing section, the program performs another comparison before executing code that is specific to command "`!@login`". (By the code specific to "`!@login`" I mean the use of the string "`pass accepted`".) We'll take a closer look at this second comparison on the next slide.

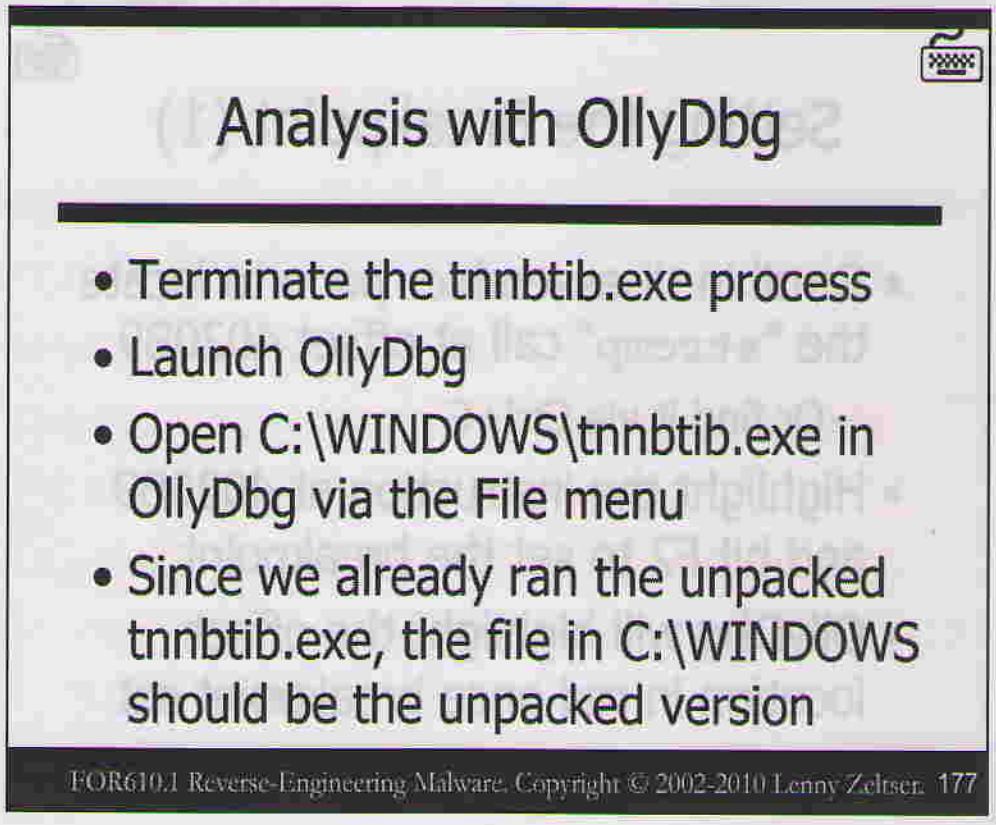


Note that right before the string "`pass accepted`" is pushed to the stack, the program performs a comparison. The jump instruction that controls whether the instruction that pushes "`pass accepted`" to the stack is reached, seems to depend on success or failure of that jump. Most likely this call to "`strcmp`" compares the password supplied by the user to the correct password, allowing the program to execute different branches of the code depending on results of this comparison.

If we could examine the program's runtime environment as it is about to perform this comparison, we might be able to see the expected login password. Please make a note of the location where the "`strcmp`" call is made, so that it is easier to refer to this crucial instruction.



This shall be our battle plan for determining the bot's login password. Use OllyDbg to trace the process, setting a breakpoint at the "`strcmp`" call that compares the valid password to the one supplied by the user. Then, we will try logging into the bot via the IRC channel using a password that we know is incorrect, in order to trigger the OllyDbg breakpoint. We will then use OllyDbg to look at the top of the stack to see which parameters are passed to the "`strcmp`" call. This should allow us to determine the real password for authenticating to the bot.

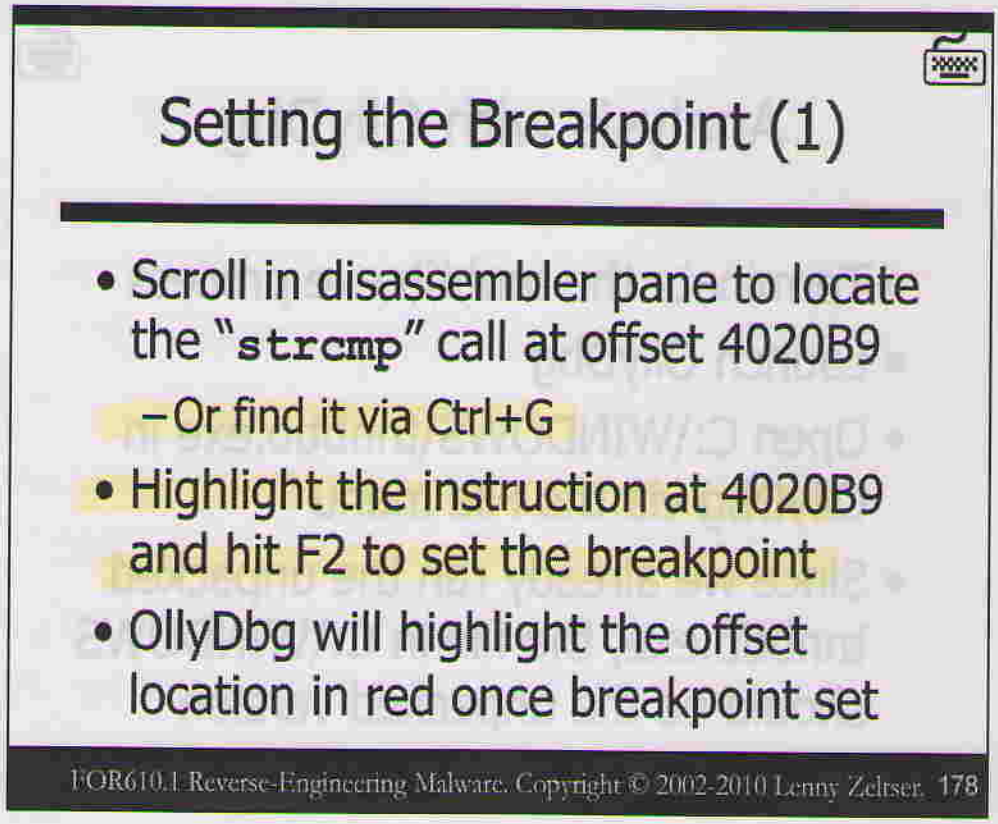


To begin this stage of the analysis, load C:\WINDOWS\tnnbtib.exe into OllyDbg.

Note that we're using the unpacked version of Tnnbtib for this example. Since you've already infected the system with the unpacked version of the specimen, the tnnbtib.exe file in the C:\WINDOWS directory should be a copy of the file you unpacked using the upx.exe utility.

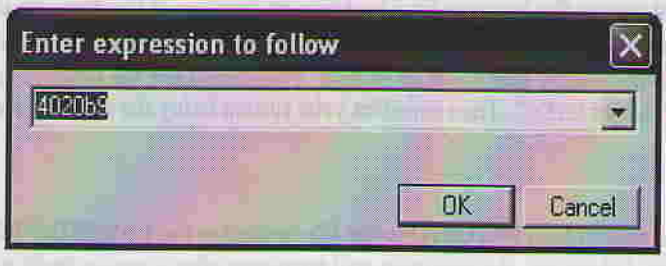
If you load the specimen into OllyDbg, and don't see the expected range of addresses that include the one you're looking for, the C:\WINDOWS\tnnbtib.exe file is probably still packed. This may occur if you forget to re-infect the system using the unpacked version of the specimen, or if the C:\WINDOWS\tnnbtib.exe file is in use by a background process. If this happens to you, terminate all existing instances of tnnbtib.exe processes, and ensure that the version of tnnbtib.exe on your Desktop has been unpacked by looking at its strings. You should see the familiar commands such as "`!@login`" and "`!@sysinfo`". Then re-infect your system using the unpacked version of tnnbtib.exe.

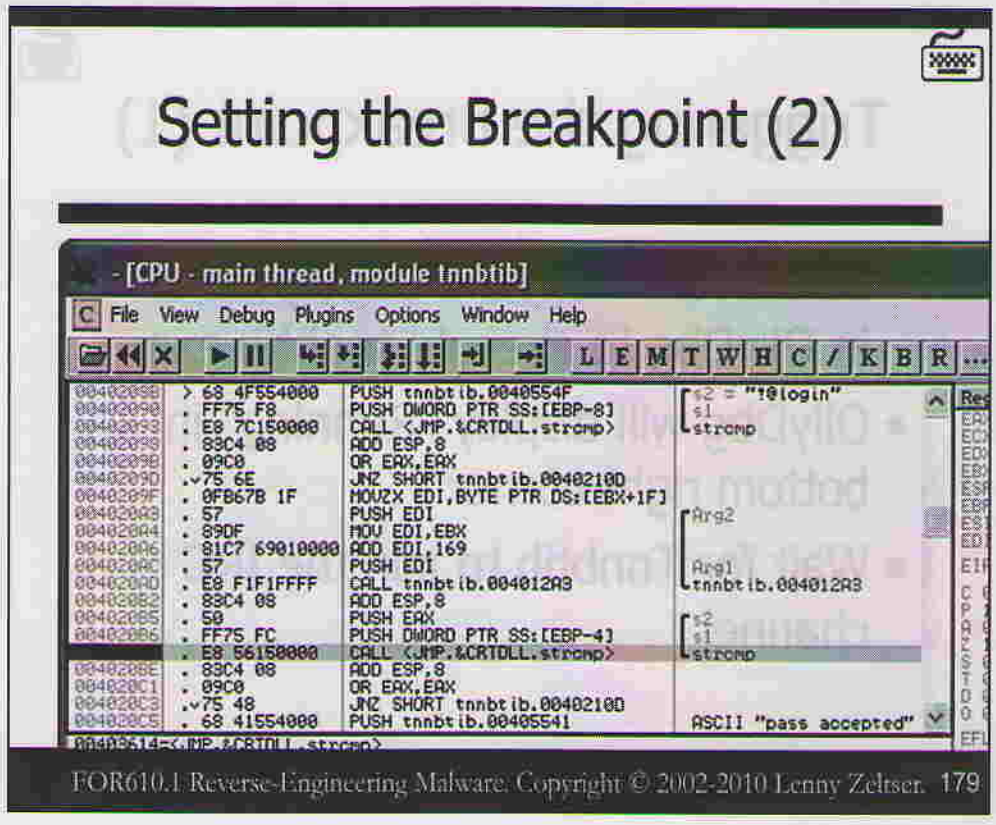
Remember that you should load into OllyDbg the file located in the C:\WINDOWS directory, not the one on your desktop. Otherwise, the specimen will terminate after it copies itself to the C:\WINDOWS directory and launches another instance of itself from that location.



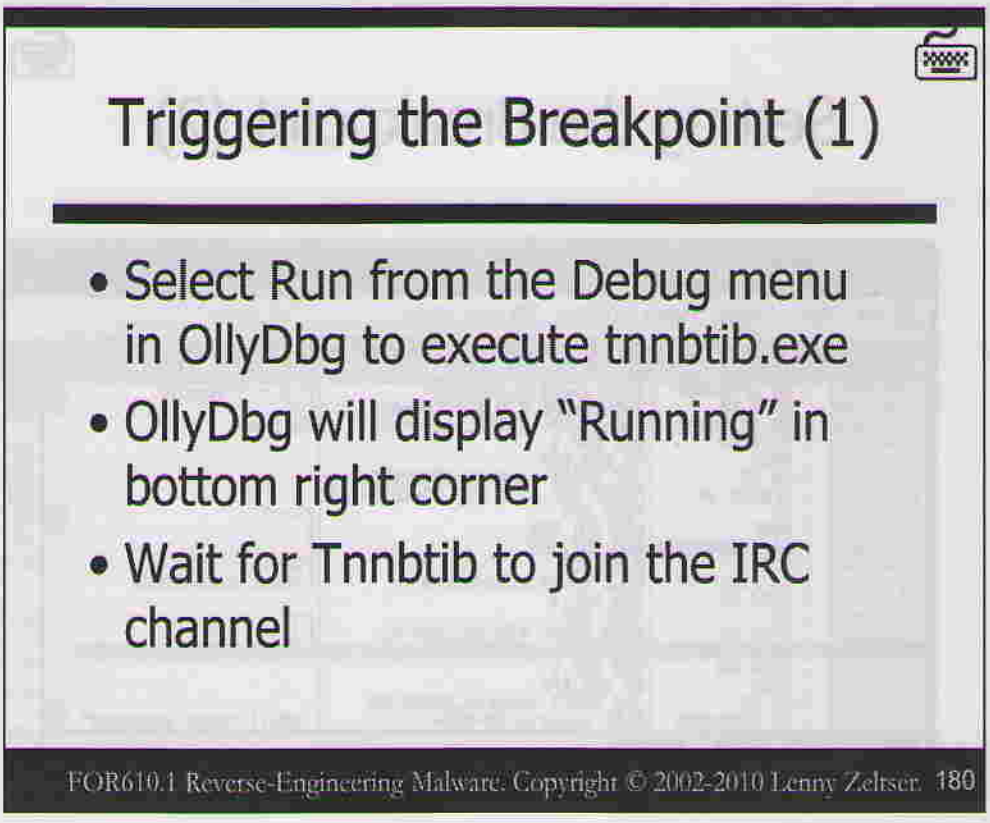
Visually locate the "`strcmp`" instruction we're interested—it's at offset 4020B9—then highlight that instruction and hit F2 to set a breakpoint there.

If you don't feel like scrolling to find the offset 4020B9, here's a shortcut: Press Ctrl+G in OllyDbg to bring up the window that will prompt you to "Enter expression to follow." Then type the desired offset into that window (4020B9) and press OK. OllyDbg will bring you to the appropriate offset, freeing you from the burden of scrolling.



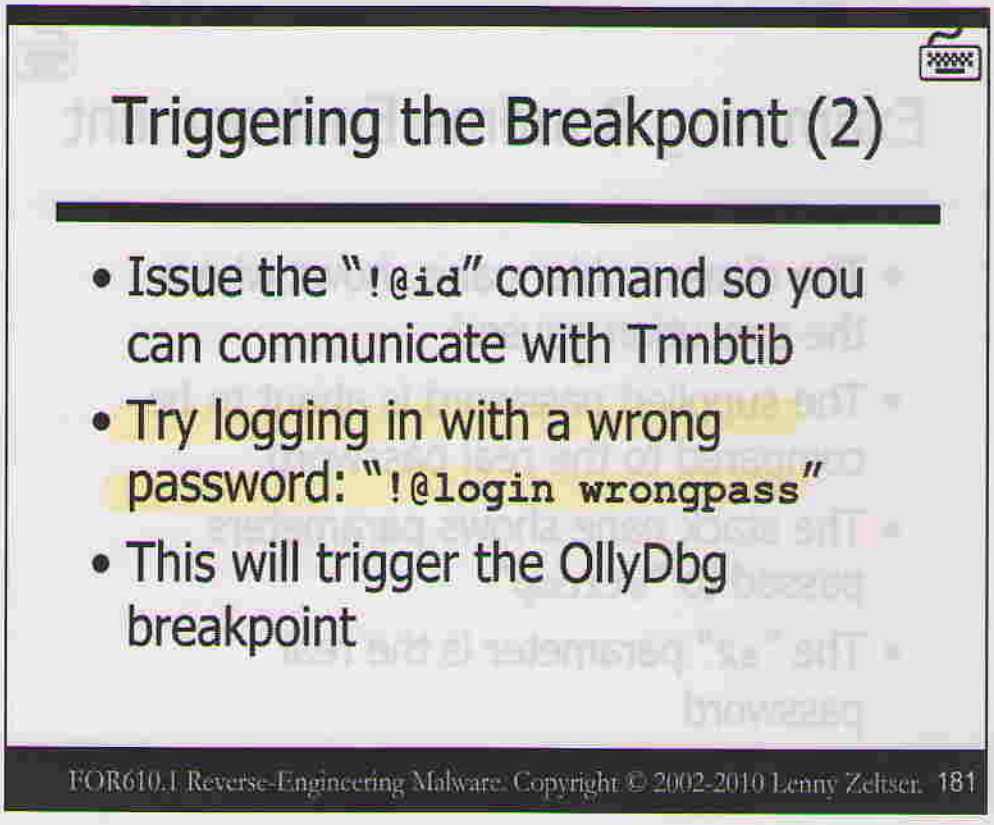


This slide includes a screenshot of OllyDbg at the point where I located the desired instruction and set the breakpoint there.



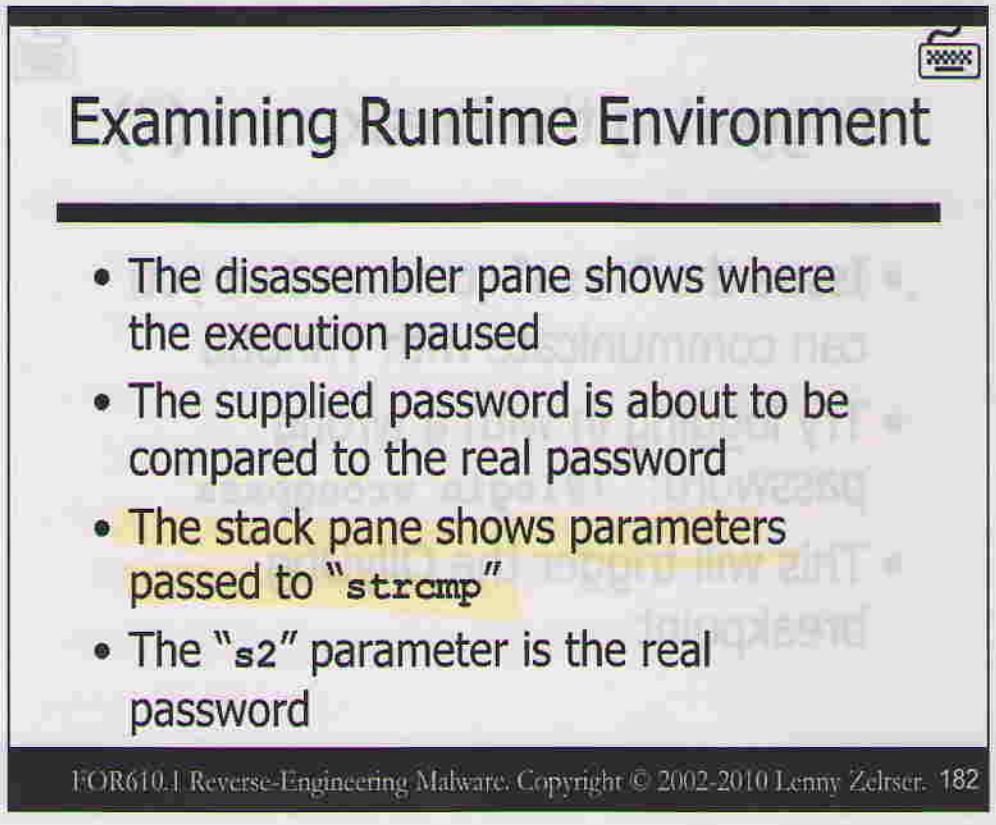
Once the OllyDbg breakpoint is set, attempt logging into Tnnbtib from the IRC channel. It doesn't matter which password you supply to the "`!@login`" command, since our goal is simply to get the program to interpret this command and reach our breakpoint.

As soon as you send the "`!@login`" command, OllyDbg should interrupt execution of the bot at the "`strcmp`" call. Examine Tnnbtib's runtime environment using OllyDbg, as we discuss on the next slide.

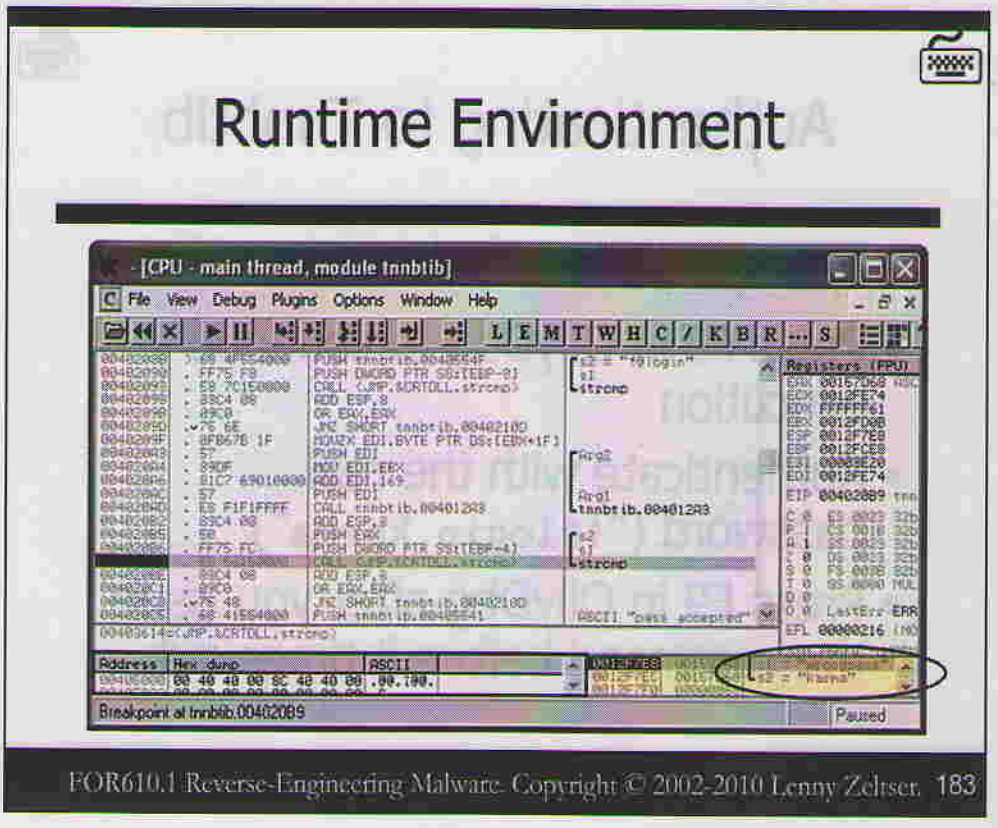


Once Tnnbtib is running within Tnnbtib, send it the "`!@id`" command through the IRC channel to make sure you can still communicate with the trojan. Then try logging into it using what you know is an incorrect password. It doesn't matter which password you supply to the "`!@login`" command, since our goal is simply to get the trojan to interpret this command and reach our breakpoint.

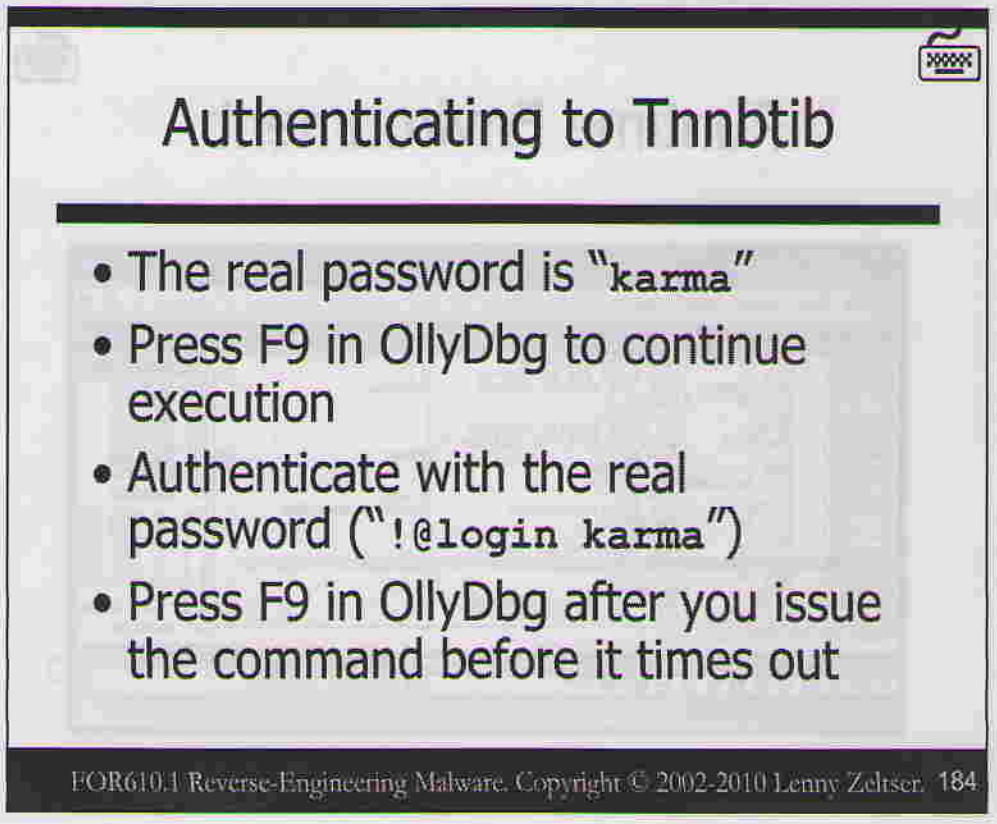
As soon as you send the "`!@login`" command, OllyDbg should interrupt execution of the trojan at the "`strcmp`" call. Examine Tnnbtib's runtime environment using OllyDbg, as we discuss on the next slide.



The screenshot on the next slide displays where the execution of the processes was interrupted. As expected, OllyDbg reached the breakpoint on instruction 4020B9, where the "`strcmp`" call resides. You can see that a few instructions further, the string "`pass accepted`" is pushed to the stack – this looks like the right place.



The Stack pane (that's the lower right corner) of OllyDbg shows parameters that are being passed to the "`strcmp`" call. One of the parameters is "`wrongpass`"–that is the fake password that we supplied to the bot when attempting to login. The other parameter is a string that we haven't seen before–this is the real password to which the program is comparing the password that we supplied to it!

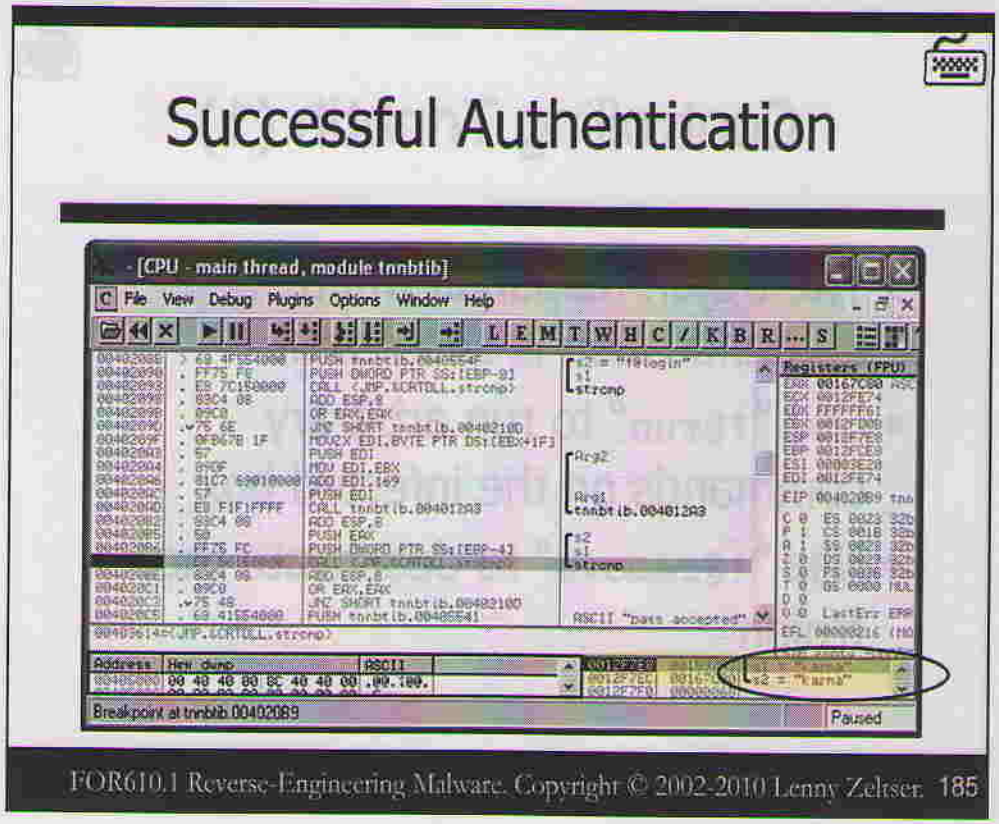


Now that you have the real login password, press F9 in OllyDbg to continue execution of the program. If you keep the process paused for too long, it will time out from the IRC server (but it will reconnect eventually).

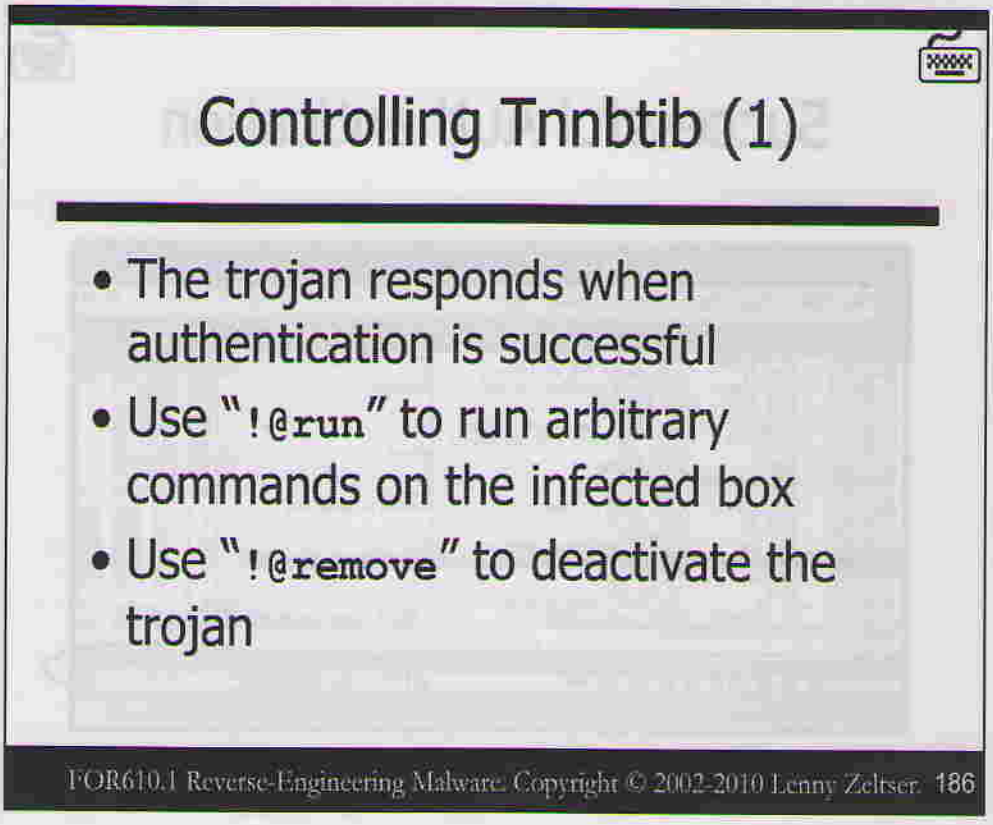
Try authenticating to the trojan using the newly discovered password ("`!@login karma`"). The trojan should respond to your command with an acknowledgement.

Since OllyDbg still remembers the breakpoint that you set in it earlier, either remove the breakpoint, or hit F9 whenever OllyDbg interrupts the execution of Tnnbtib upon encountering the password comparison instruction.

Here's an OllyDbg usage tip: OllyDbg remembers breakpoints across debugging sessions, even if you exit the program. This information is saved in files that are located in OllyDbg's installation directory (for instance, C:\Program Files\OllyDbg). The names of these files begin with the name of the executable that was being debugged. To force OllyDbg to "forget" these breakpoints, simply remove these files.

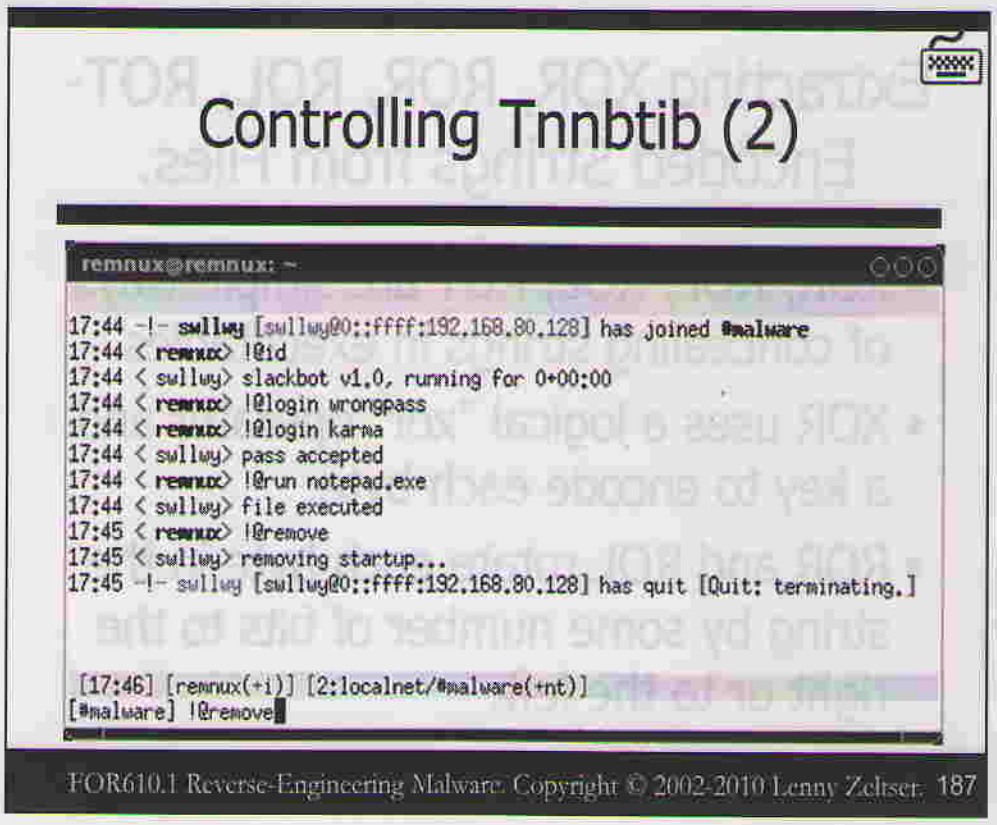


The screenshot on this slide shows OllyDbg in action. Glancing at the lower-right corner shows the reason the authentication attempt will succeed this time: both arguments to the "`strcmp`" function are the same.

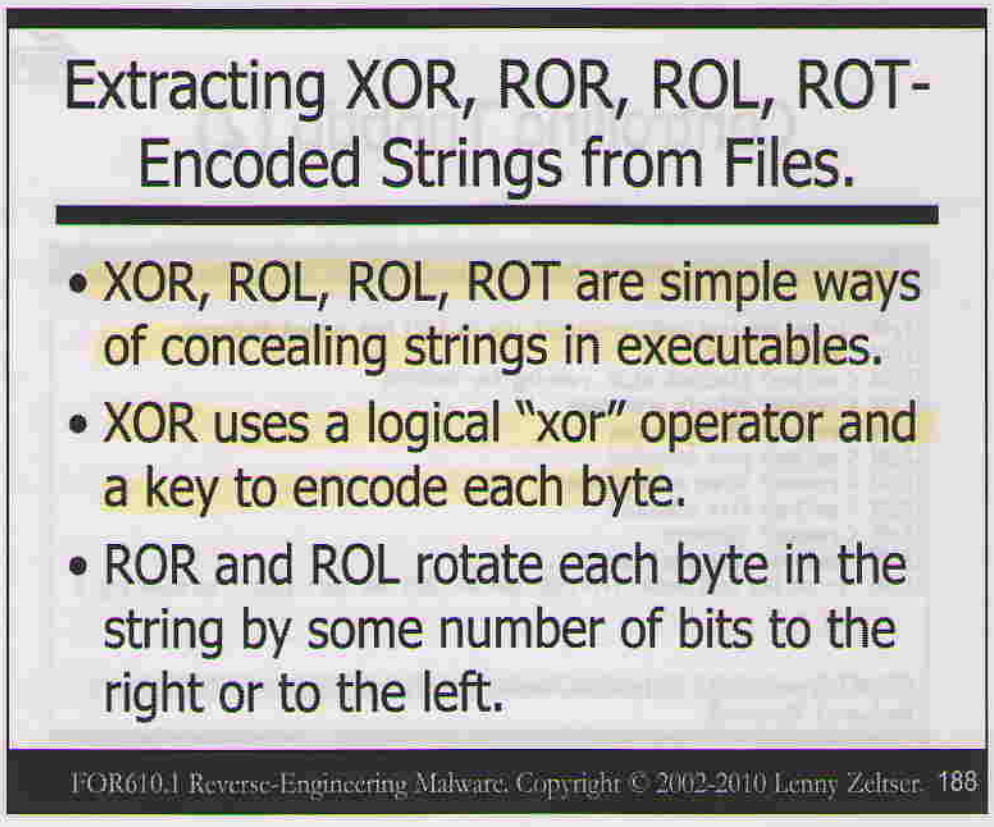


After you authenticate to Tnnbtib, it should accept from you commands that were ignored earlier. For instance, the screenshot on this slide shows that we were able to successfully launch Notepad on the infected machine by commanding the trojan to run notepad.exe.

Try experimenting with other commands that you saw embedded into the bot's executable. Keep in mind that some of them may require one or more parameters. One of the commands that may be worth trying is "`!@remove`", which may remove the trojan from the infected system.

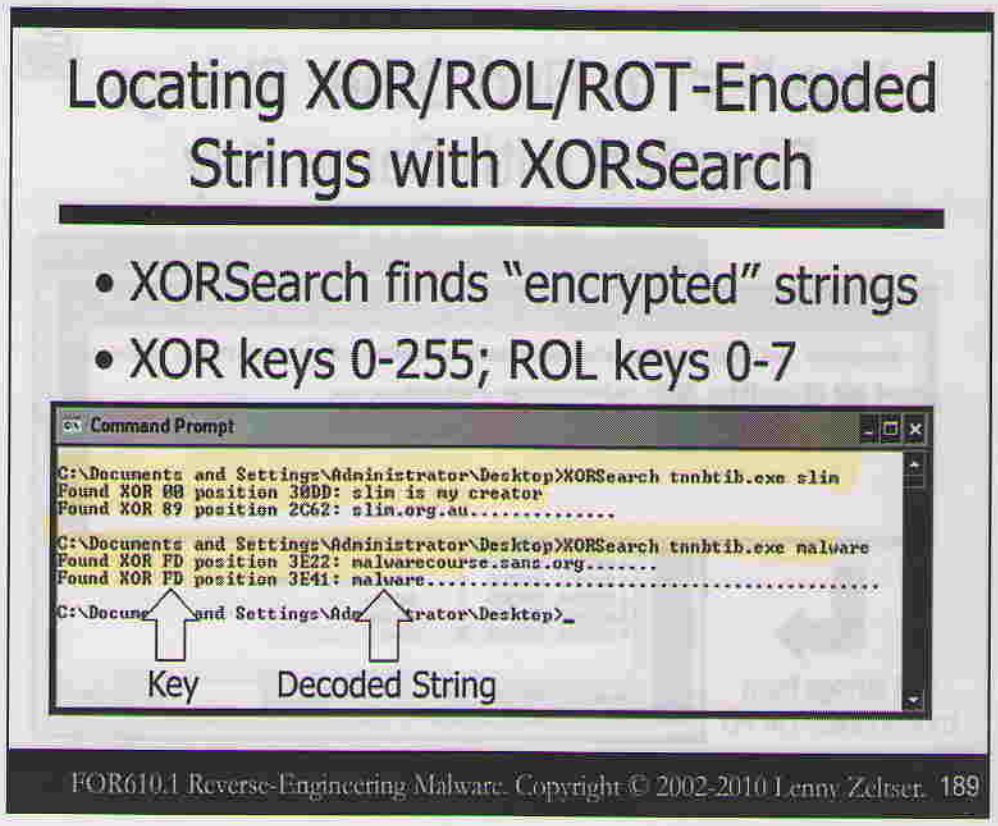


The screenshot on this slide shows my IRC session during which I authenticated to Tnnbtib and commanded it to launch notepad.exe on the infected system.



When concealing sensitive data inside an executable, malware code authors often look for a simple-to-implement mechanism for obscuring the string's original value. One way to accomplish this is to use the logical "xor" operator to change each character by "xor'ing" it with some random number (this number plays the role of the key). To reverse the process, the same operation needs to be applied to obtain the string's original value.

Another simple approach rotates each byte in the string by some number of bits to the right or to the left. This algorithm is called ROR (rotate right) or ROL (rotate left), depending on the direction in which the rotation occurs. Another common approach called ROT rotates alphabet characters (A-Z and a-z) by a certain number of positions within the alphabet.



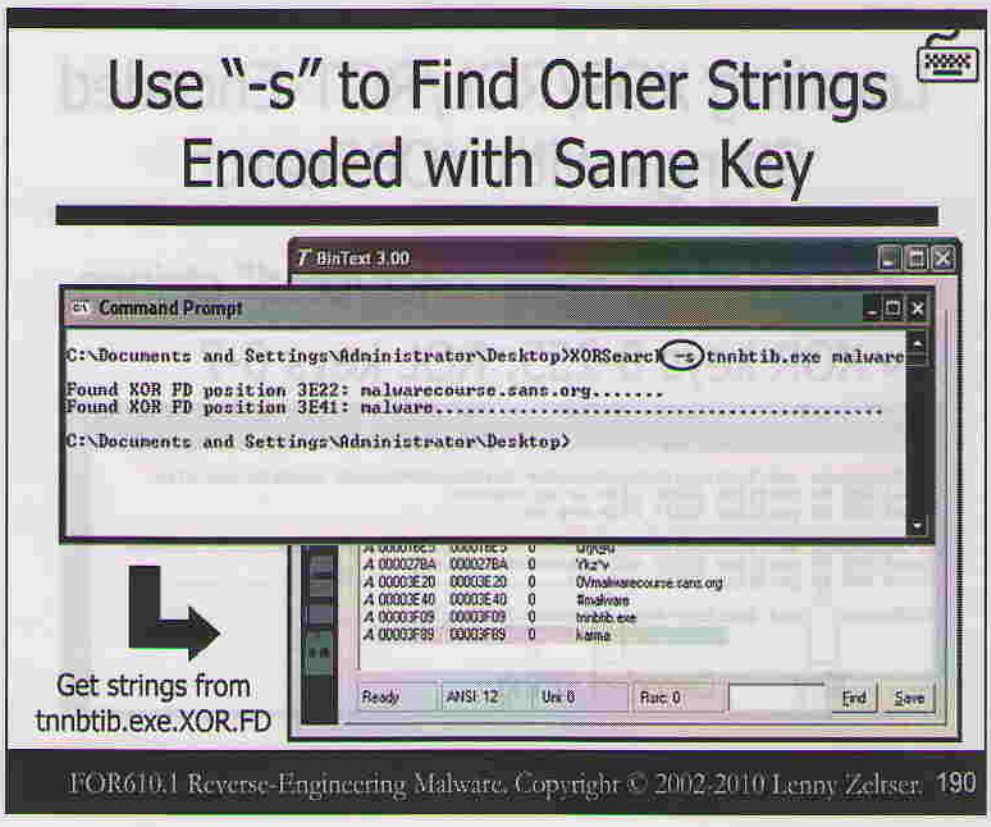
If you suspect that the malicious executable is concealing its strings using XOR/ROL/ROT, you'll find XORSearch useful. It was written by Didier Stevens, and is available as a free download from the following URL:

http://blog.didierstevens.com/programs/xorsearch/

You will also find it on the DVD for this course in the \CodeAnalysis folder.

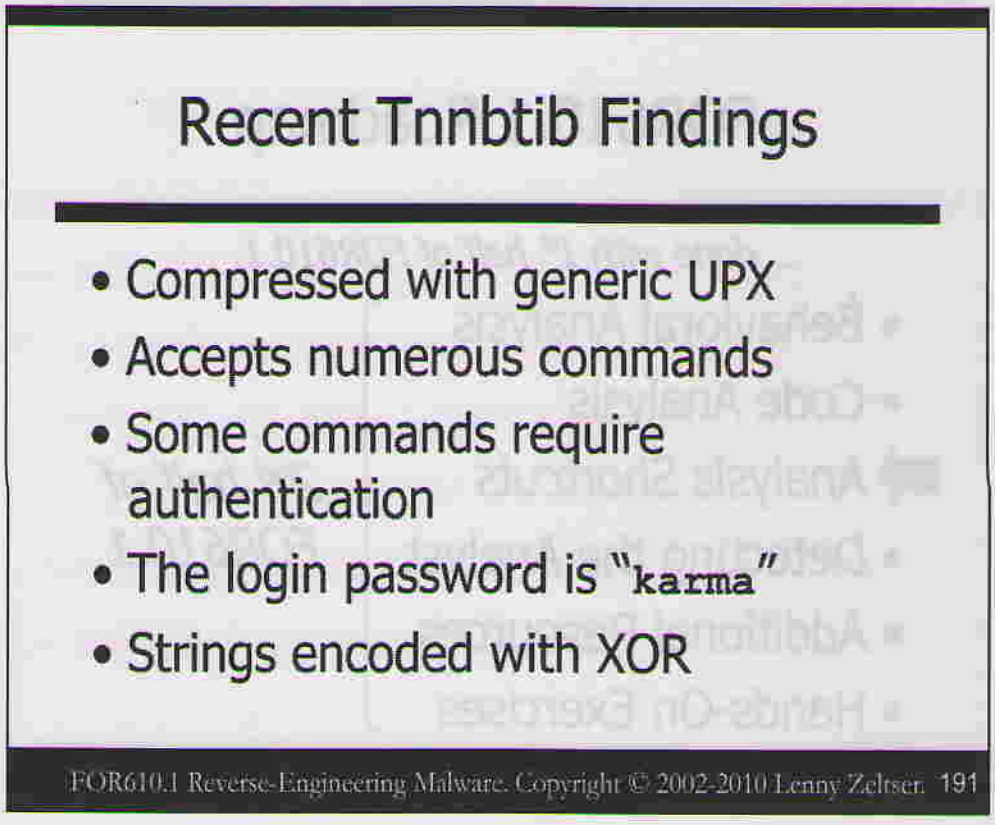
XORSearch searches the file for the specified string encoded with XOR, ROL, and ROT, attempting all possible key values in the range 0-255 for XOR, 0-7 for ROL, and 1-25 for ROT. Even though the program explicitly implements only ROL (not ROR), rotating to the left for all key values is equivalent to rotating to the right (ROR) for our purposes.

I scanned the body of the unpacked version of tnnbtib.exe with XORSearch, asking it to look for the string "`malware`". I knew to search for this string, because I witnessed it used by the program during behavioral analysis; yet, it was not visible as a regular string in the program's body. XORSearch found two strings that included "`malware`". They were each encoded with XOR using the key "`FD`". XORSearch also specified the location where these strings were embedded in the file.

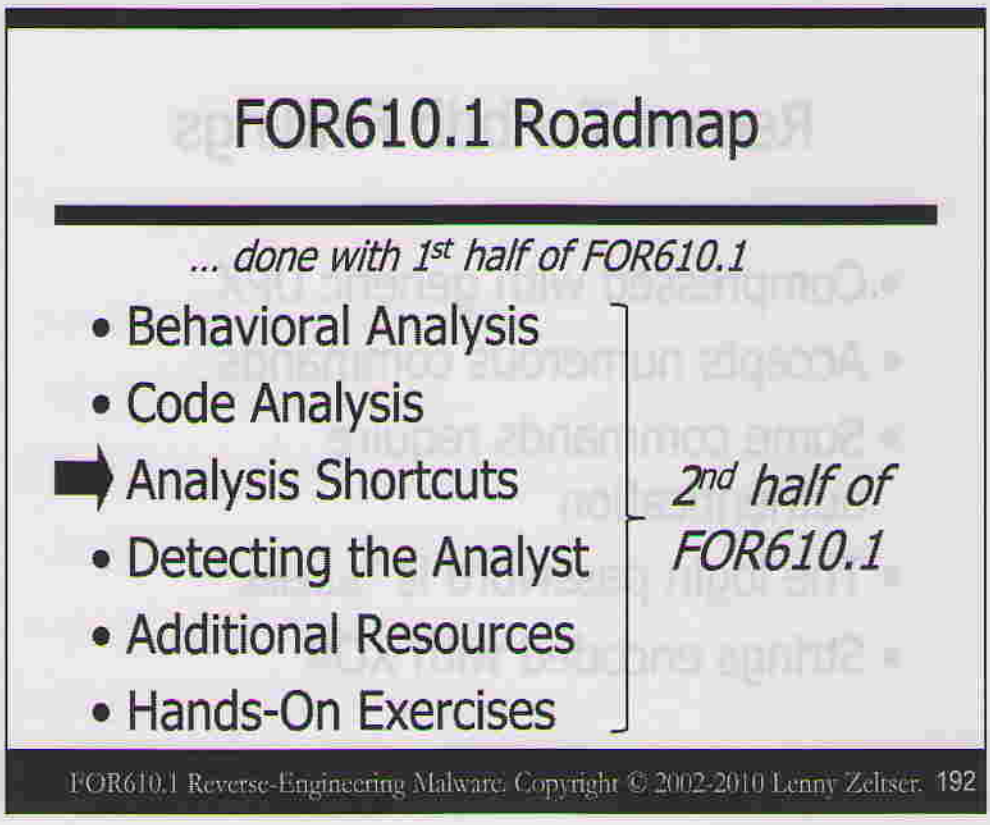


One of the nice features of XORSearch is the ability to locate other strings in the file, encoded with the same key as the string for which you searched. If you invoke it with the "`-s`" parameter, XORSearch will not only scan for obfuscated strings, but also deobfuscate all the strings it will find encoded with the same key.

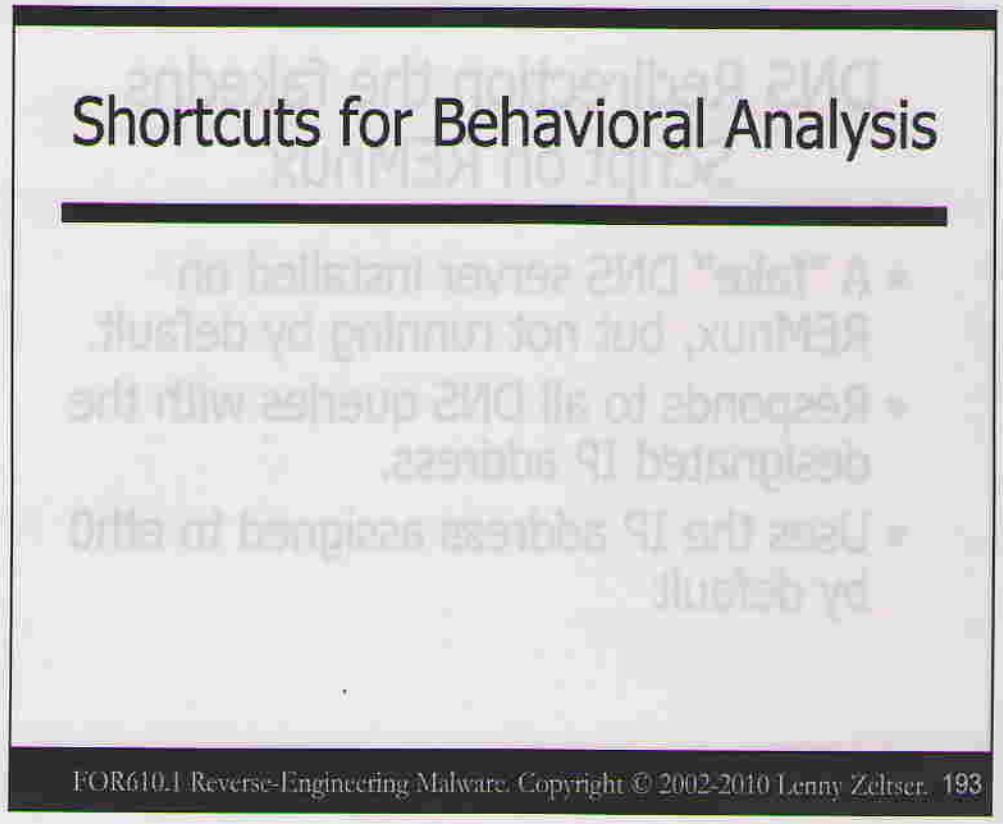
In this example, I only searched for "`malware`". XORSearch found it and the corresponding key, and created the file tnnbtib.exe.XOR.FD. I used BinText to extract strings from that file, and noticed that XORSearch not only decoded the strings that included "`malware`", but also decoded two other strings that were encoded with the same key "`FD`". These strings were "`tnnbtib.exe`" and "`karma`". This can be useful for locating the strings you have not encountered during earlier stages of your analysis.



So, what have we discovered about Tnnbtib through code analysis? We were able to find and understand a number of commands that can be sent to Tnnbtib via the IRC channel. By experimenting with these commands and analyzing relevant code segments we are able to understand how to use them and what effects they may have on the trojan's behavior. We discovered that most commands require authentication. We determined that the trojan's login password is "`karma`", and were able to successfully authenticate to the trojan to execute commands such as "`!@run`" and "`!@remove`". We also determined that some of the more sensitive strings that affect the program's behavior are encoded in it using XOR and the key 0xFD.

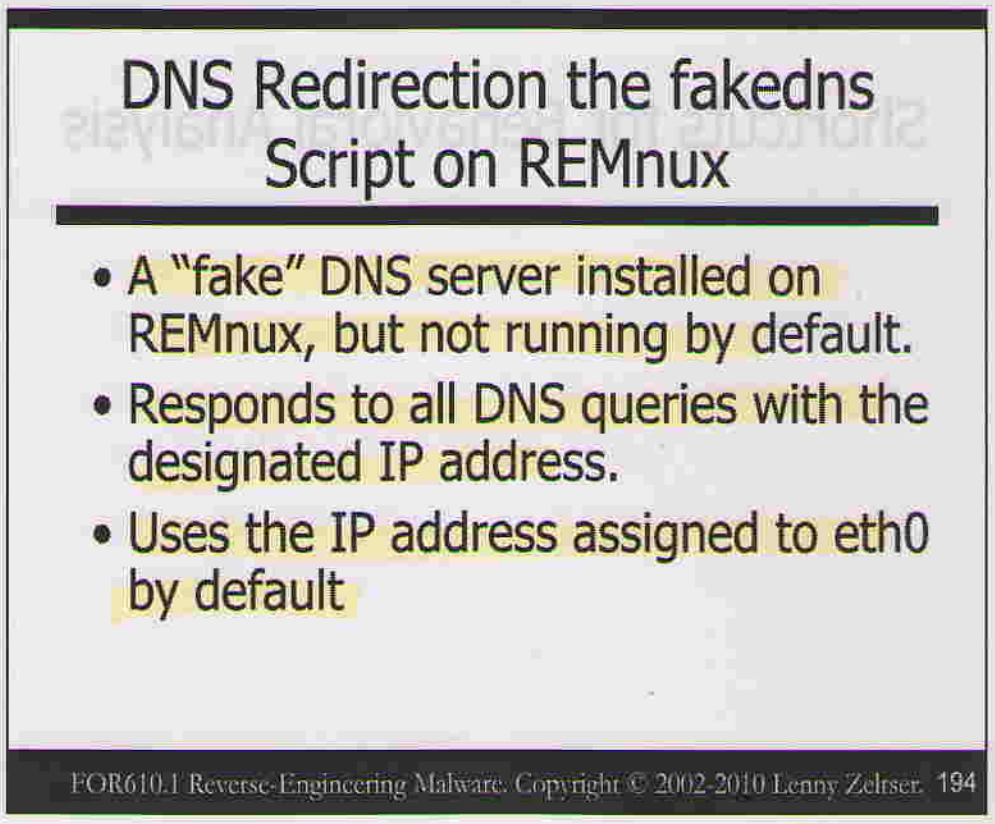


Now that we examined Tnnbtib from behavioral and code perspectives, let's look at some shortcuts for saving time when examining malware.



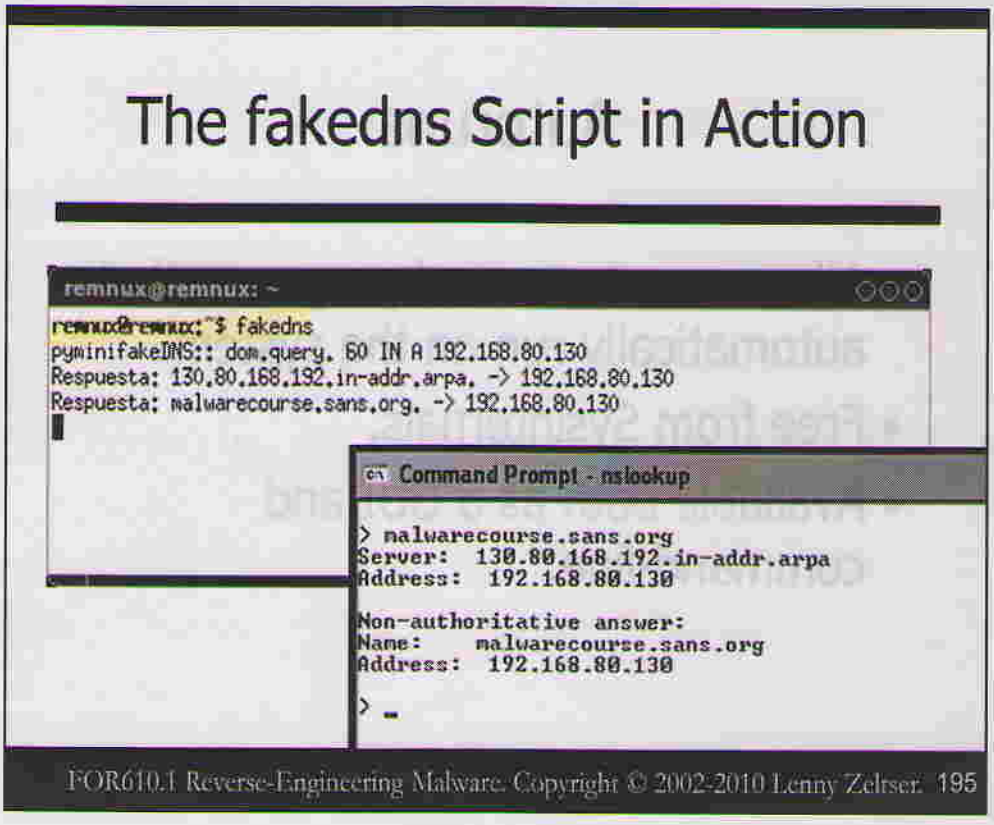
We've done our fare of analysis using standalone tools and manual techniques. Now, let's take a look at a few utilities that can save us time by automating a few steps or integrating disparate tools. These tools are very convenient, and are reliable most of the time. However, I found that on several occasions they failed to log some of the activity that I was expecting them to log (just something to keep in mind when you use them).

Let's see what these tools are about…



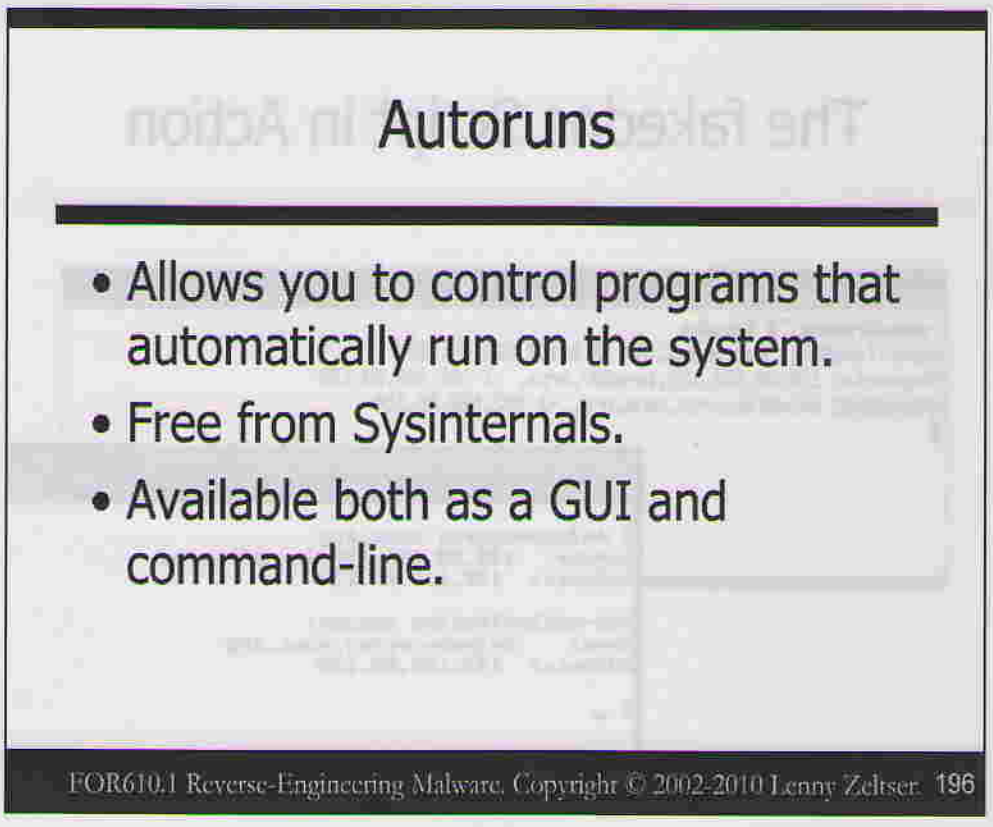
The *fakedns* script installed on REMnux provides a convenient alternative to using hosts file for DNS-based traffic redirection. The *fakedns* automatically responds to all DNS queries using the IP address that you specify. On REMnux, the script uses the IP address assigned to eth0 by default.

In this course, we've been using the more laborious "hosts" file technique, instead of the *fakedns* script, because it provides us with a great deal of control in situations where we may not want to allow the malware specimen to resolve all of its hostnames at once. However, *fakedns* provides a convenient shortcut for redirecting DNS-based traffic in the analysis lab.



To launch the *fakedns* script, type "`fakedns`". You can override the IP address with which the script responds to DNS queries; by default, it will use the IP address assigned to the eth0 network interface. In this example, the IP address is 192.168.80.130; the IP address of your REMnux virtual machine will probably be different.

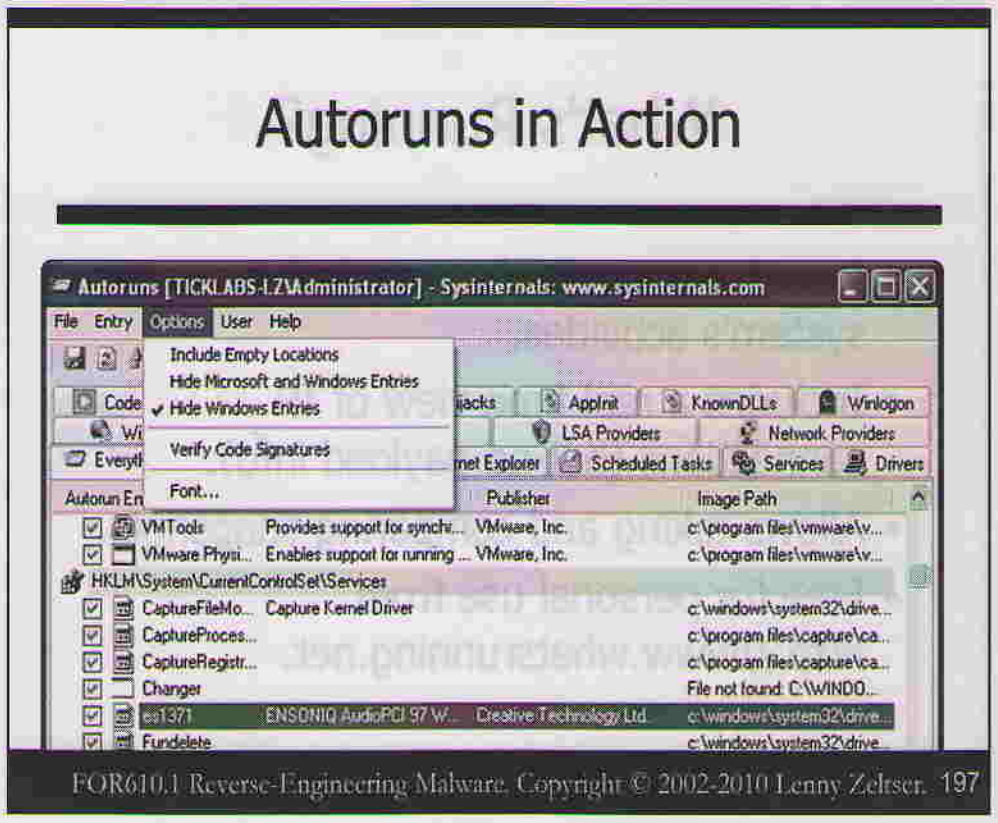
When you use *fakedns* script, don't forget to set your infected host's DNS server to the IP address of your REMnux virtual machine, as you were asked to do when setting up the lab for this course; otherwise, REMnux will not receive the infected host's DNS queries.



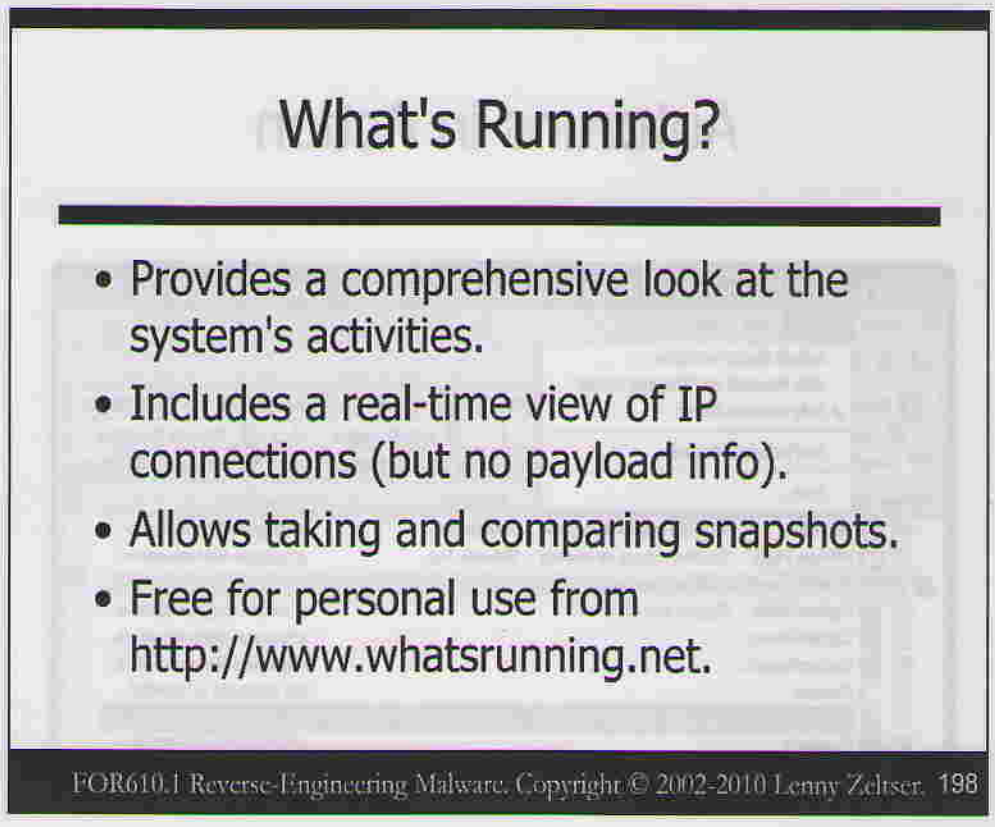
The Autoruns utility is a free tool, available from Sysinternals at the following URL:
http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx

This program allows you to easily determine which programs are set to automatically start when the system reboots, and to control their start-up behavior. Autoruns can also display Windows services, and even Internet Explorer Browser Helper Objects (BHOs).

Autoruns is available as a GUI program, and as a utility that you can run from a command-line. This program works on all flavors of Windows.



In order to focus on items that are most likely to be suspicious, you can select the Hide Signed Microsoft Entries option. This will prevent the program from displaying software modules that were cryptographically signed by Microsoft. Deselecting an entry in the Autoruns list disables the entry's auto-run capability.



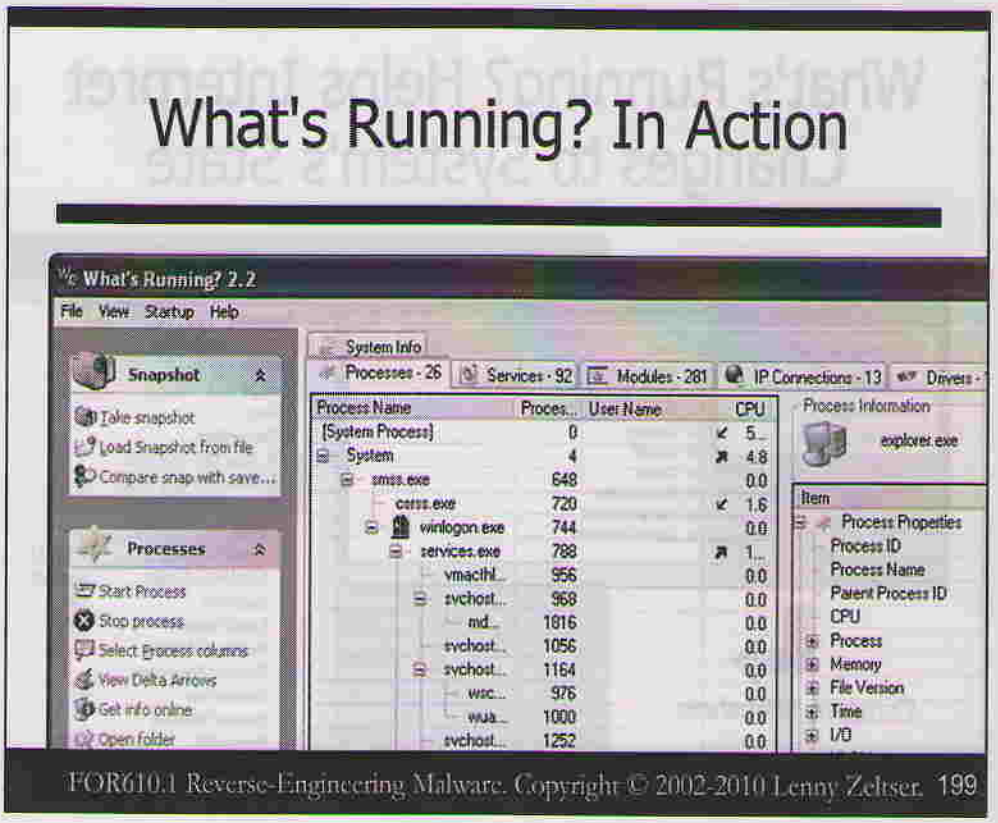
What's Running is an excellent tool for quickly examining important aspects of the infected system's runtime environment, including:

- Processes (displayed in a tree fashion)
- Services, modules, and drivers
- IP connections (excluding payload information)
- Start-up items from the registry and Startup folders

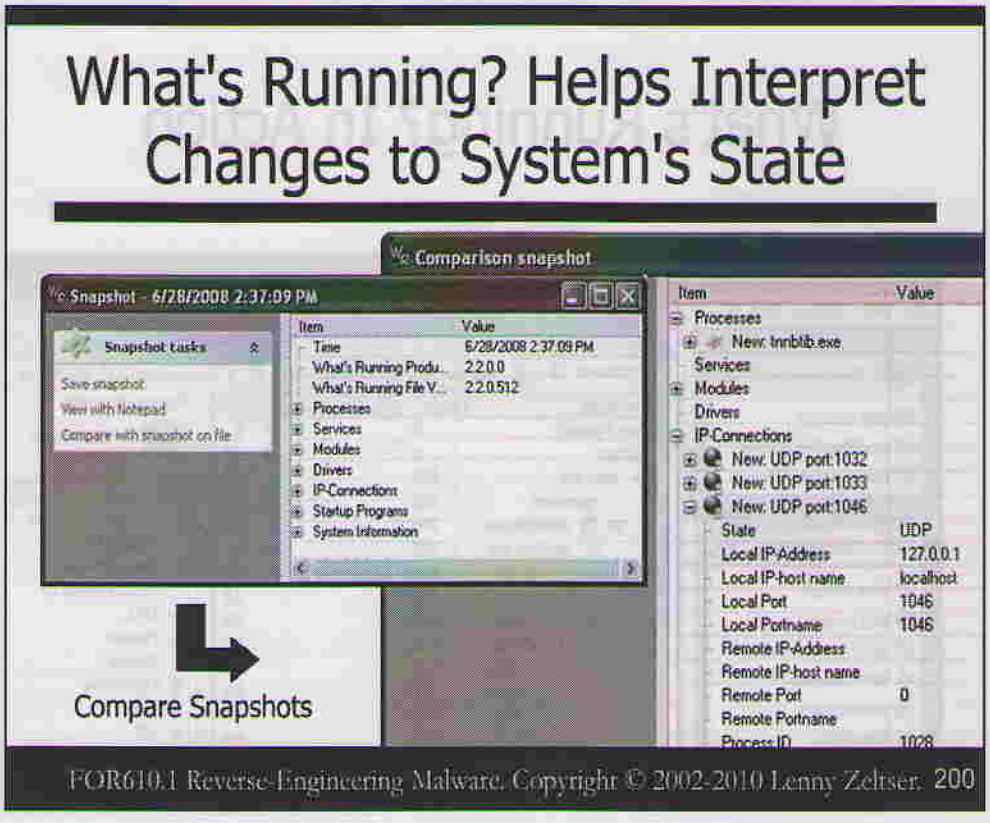
The ability to observe in real time the system's IP connections is particularly convenient. This feature allows you to see which process name is participating in network conversations, and obtain details about both sides of the communication. What's Running? displays information about both UDP and TCP connections.

This tool also provides a convenient mechanism for taking a snapshot of the current system's state, with respect to the system parameters tracked by What's Running? You can compare two snapshots to detect changes in the state.

What's Running? is free for non-commercial use, and is available at: http://www.whatsrunning.net.

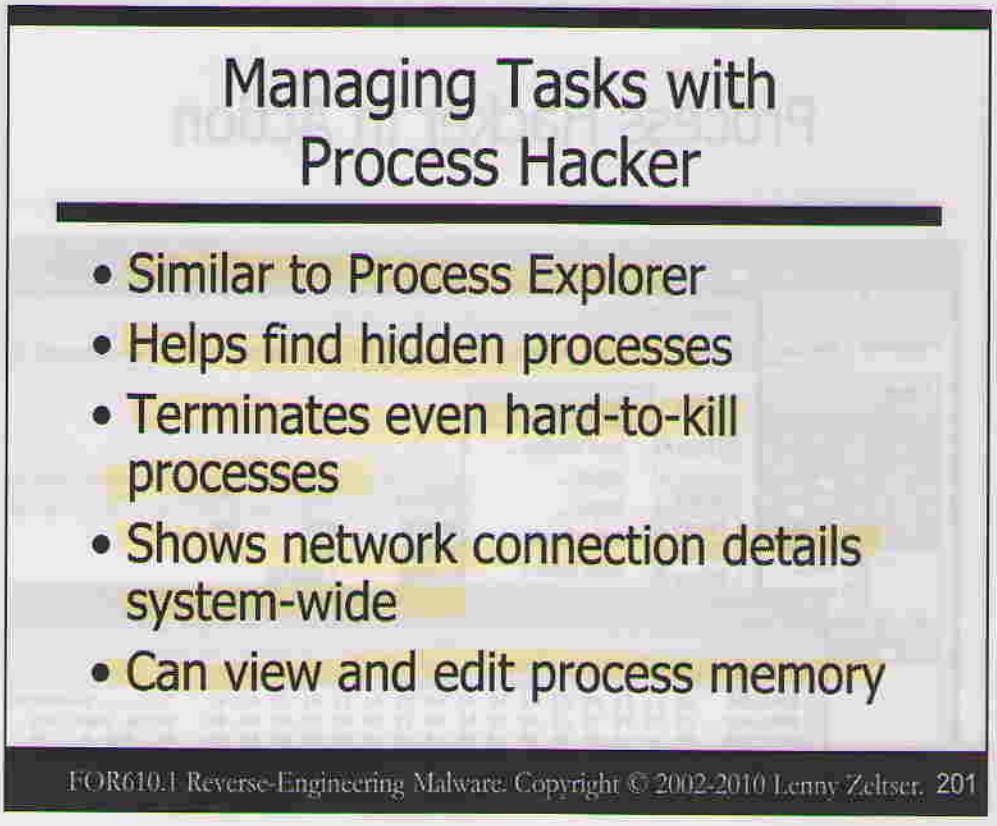


The screenshot on this slide shows What's Running? In action. The tab visible at the moment lists the processes running on the system in a tree-like fashion. Clicking on any process displays that process' information in the right-most pane on the screen, including the process' IP sockets, dynamic libraries, associated services, and so on. You can also start and stop the processes, as well as take snapshots of the system's state.



The ability to take and interpret snapshots of the system's state is, perhaps, the most useful feature of What's Running? for our purposes. While you can quickly detect changes to the system's state using RegShot, RegShot will not interpret changes to the registry for you, nor will it notice any chances to the system's running processes. What's Running? is able to not only detect changes, but also identify them as having a specific effect on the system's running processes, services, drivers, loaded modules, and so on.

To make use of the Snapshot feature in What's Running?, take the first snapshot (before infecting the system) by clicking on the "Take snapshot" link in the program's main window, then click "Save snapshot" in the window that will pop up. After infecting the system, take and compare the second snapshot by clicking on "Compare snap with saved snap" in the program's main window, then point the program to the first saved snapshot.

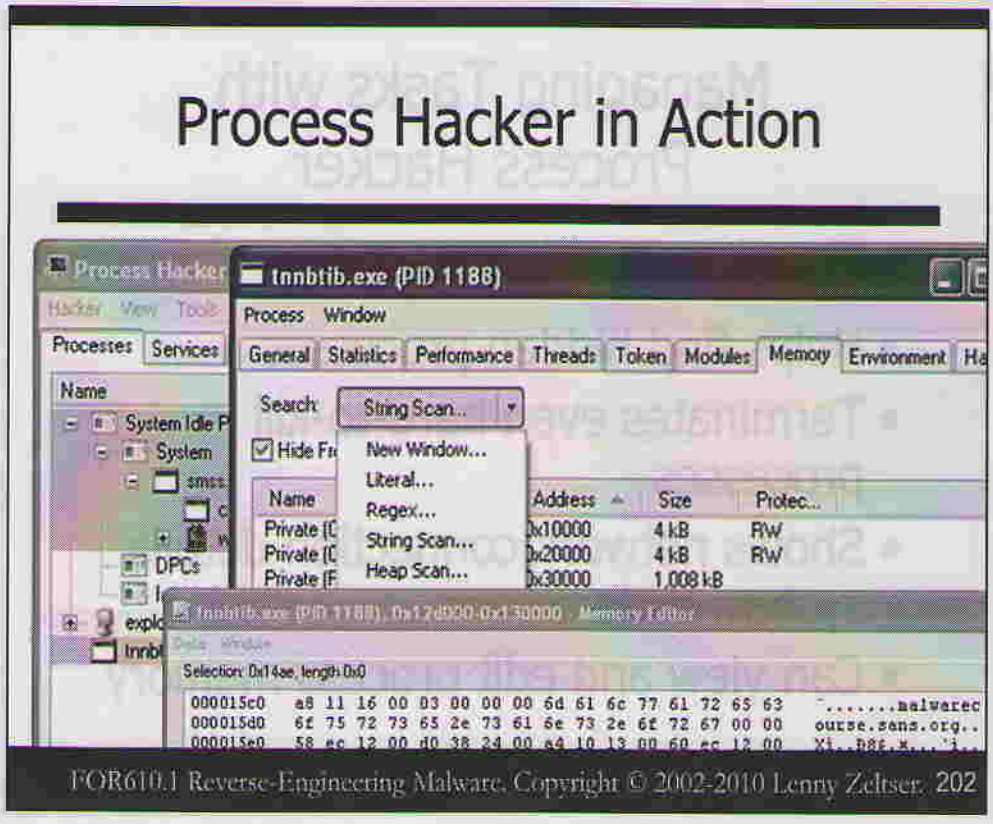


Another tool that's worth considering for your malware analysis toolkit is Process Hacker. It is a free, open-source tool that is similar in key functionality to Process Explorer; however, Process Hacker includes several features that are particularly useful when dealing with malware. For instance, Process Hacker can find hidden processes that may be associated with rootkits. Also, its built-in kernel driver allows terminating even protected processes that are usually hard to kill. Process Hacker also includes a tab for seeing network details across all processes running on the system. This tool can even allow you to look at and edit memory contents of the running processes.

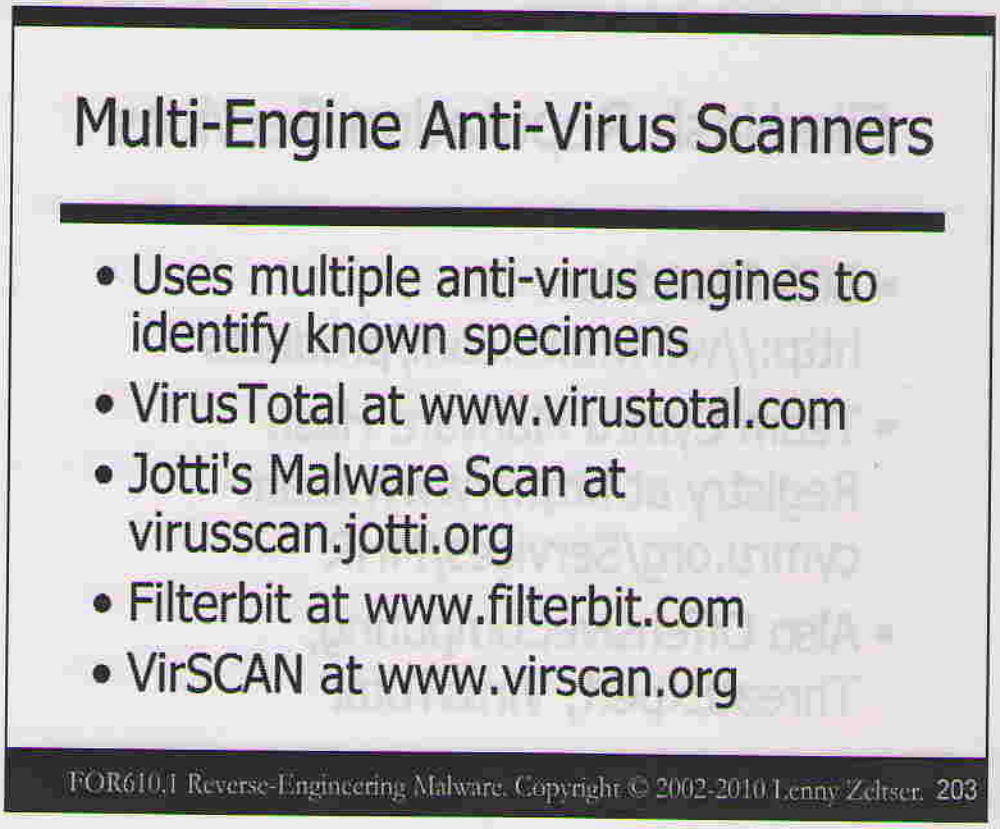
For information about Process Hacker and to download this tool, visit:

http://processhacker.sourceforge.net

Also, note that Process Hacker requires that Microsoft's .NET Framework 2.0 be installed on your system.



This slide captures just some of the many process-management features built into Process Hacker. For instance, you can scan the memory of the process (in this case, of tnnbtib.exe) for strings; double-click on an interesting string, and arrive at the built-in memory editor to tweak the process' runtime environment.



A handy tool to have as part of your toolkit is a multi-engine anti-virus scanner. Having multiple scanners at your disposal is very useful for identifying the malware specimen you are dealing with. If you can determine what kind of beast you are dealing with, you may be able to locate information about it that will speed up your analysis. Scanning the malicious file with several scanners increases the likelihood of successful identifying it. Moreover, some anti-virus products are relatively good at identifying common packing mechanisms.

It may be difficult and expensive to have multiple anti-virus programs available locally. Fortunately, there are several sites that allow you to scan the suspicious with several scanners by uploading the file to their site. Two of such services, both operating for no fee, are VirusTotal and Jotti's Malware Scan.

VirusTotal is a very convenient service, which allows you to scan your file for free using numerous anti-virus products.

Jotti's Malware Scan is another free multi-engine scanner that is a less mainstream alternative to VirusTotal. Another option, though one that seems to use fewer anti-virus engines, is Filterbit.

VirSCAN is another free multi-engine scanner, similar to the ones mentioned above.

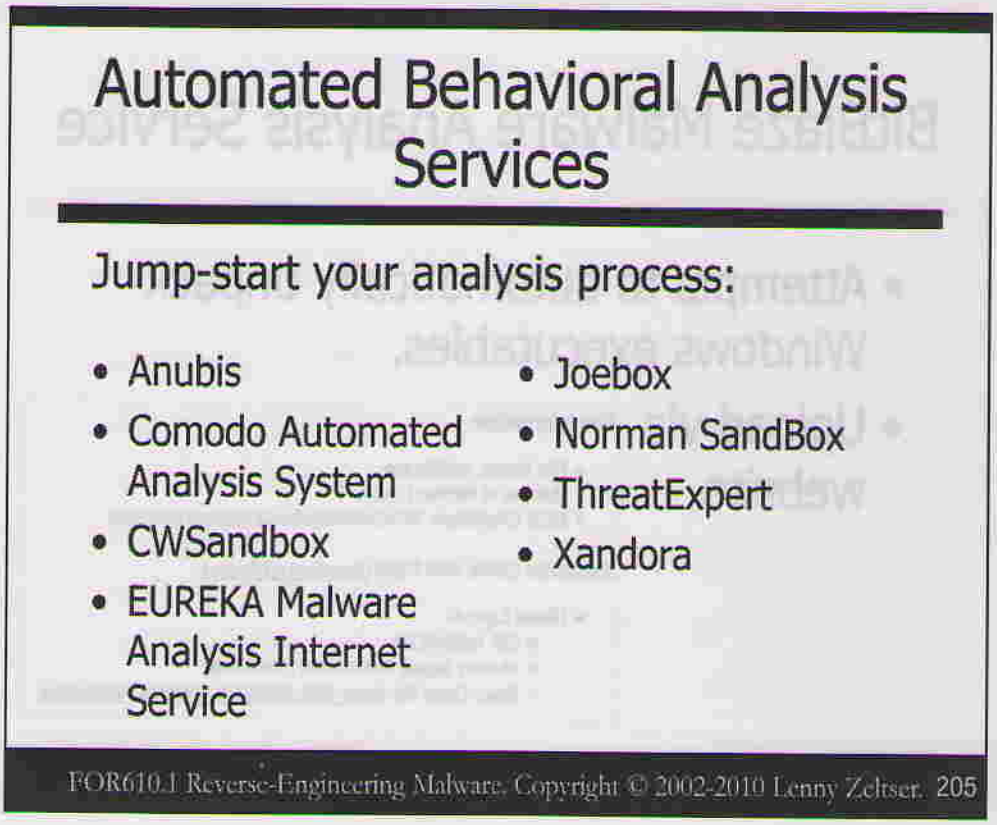
Before uploading your files to these files, consider whether you trust them with your data. In some situations, you may not wish to share the file with a third party whom you do not have a contractual agreement. Furthermore, these sites provide reports about executables that they scanned; attackers can use such features to determine when or whether their executables have been discovered by the targeted organization.

When faced with a potentially-malicious file, you can look up reputational data about it by querying the following free websites for that file's MD5 hash:

- Bit9 FileAdvisor (http://www.bit9.com/products/fileadvisor.php)
- Team Cymru Malware Hash Registry (http://www.team-cymru.org/Services/MHR)

You can also find information about malicious files based on their MD5 hashes by searching the following sites we mentioned earlier in the course:

- OffensiveComputing (http://www.offensivecomputing.net)
- VirusTotal (http://www.virustotal.com)
- ThreatExpert (http://www.threatexpert.com)

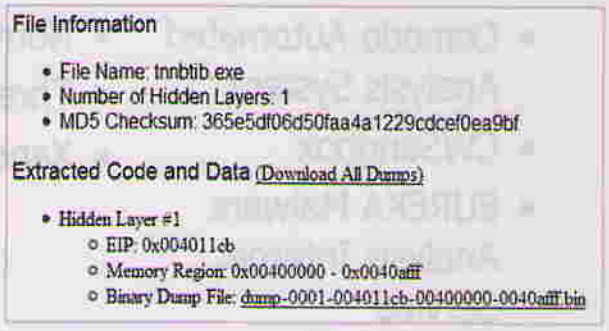


There are several free automated malware analysis services that can examine compiled Windows executables to save us time and provide a sense about the specimen's capabilities. Such service allows you to upload a malicious executable to the free website; however, rather than simply checking the file against a database of known signatures, the service attempts to automatically determine what actions the executable would take on the infected system. After scanning your file, the services provide a report that includes basic behavioral information, such as what files and registry entries the malicious program might create, and may even tell you about some network-level activity the specimen may exhibit.

- Anubis - http://anubis.iseclab.org/
- Comodo Automated Analysis System - http://camas.comodo.com/
- CWSandbox - http://www.cwsandbox.org/?page=submit, and the instance at Sunbelt Software - http://sunbeltsecurity.com/Submit.aspx?type=cwsandbox&cs=A41CD150B37359889A553671CBFD2360
- EUREKA Malware Analysis Internet Service - http://eureka.cyber-ta.org/
- Joebox - http://www.joebox.org/submit.php.
- Norman SandBox - http://norman.com/security_center/security_tools/submit_file/en
- ThreatExpert - http://www.threatexpert.com/submit.aspx
- Xandora - http://xandora.security.net.my/

BitBlaze Malware Analysis Service

- Attempts to automatically unpack Windows executables.
- Upload via website



BitBlaze is an innovative service in that it attempts to automatically unpack (dump) the Windows executable you upload via its Web interface. Once the file has been analyzed, you receive a link to its report via e-mail. This is a free service, based on research at Carnegie Mellon University. You can access it by visiting:

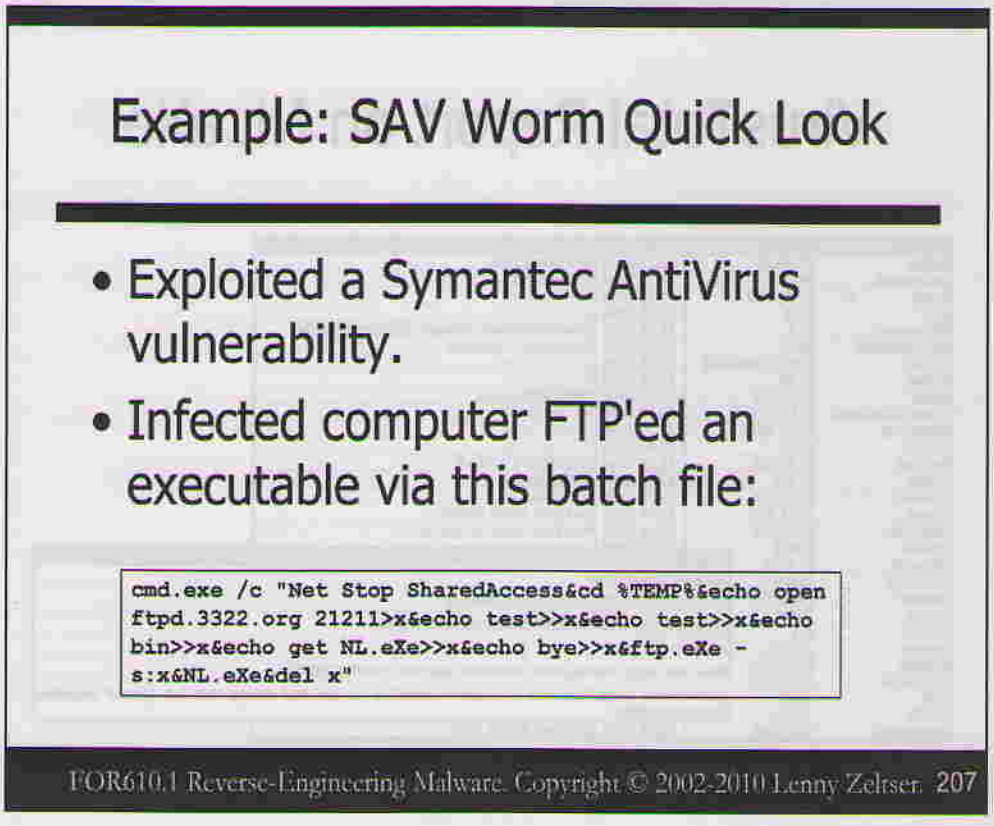
http://panorama.ece.cmu.edu

The part of the service that extracts packed code from malicious executables is called Renovo. This module "monitors program execution and memory writes at run-time, determines if the code under execution is newly generated, and then extracts the hidden code of the executable."

The technique employed by Renovo is outlined in the paper *Renovo: A Hidden Code Extractor for Packed Executables* by Min Gyung Kang, Pongsin Poosankam, and Heng Yin, which is available at:

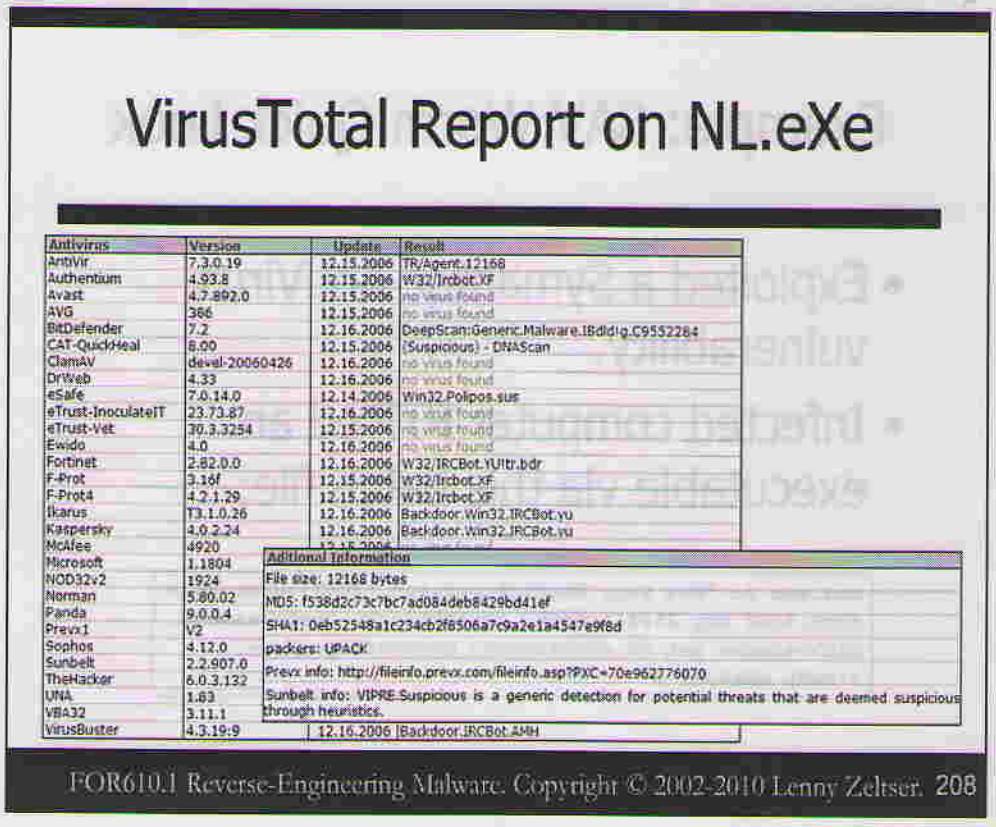
http://www.andrew.cmu.edu/user/ppoosank/papers/renovo.pdf

In the example illustrated on this slide, the service produced a downloadable dump file for tnnbtib.exe.

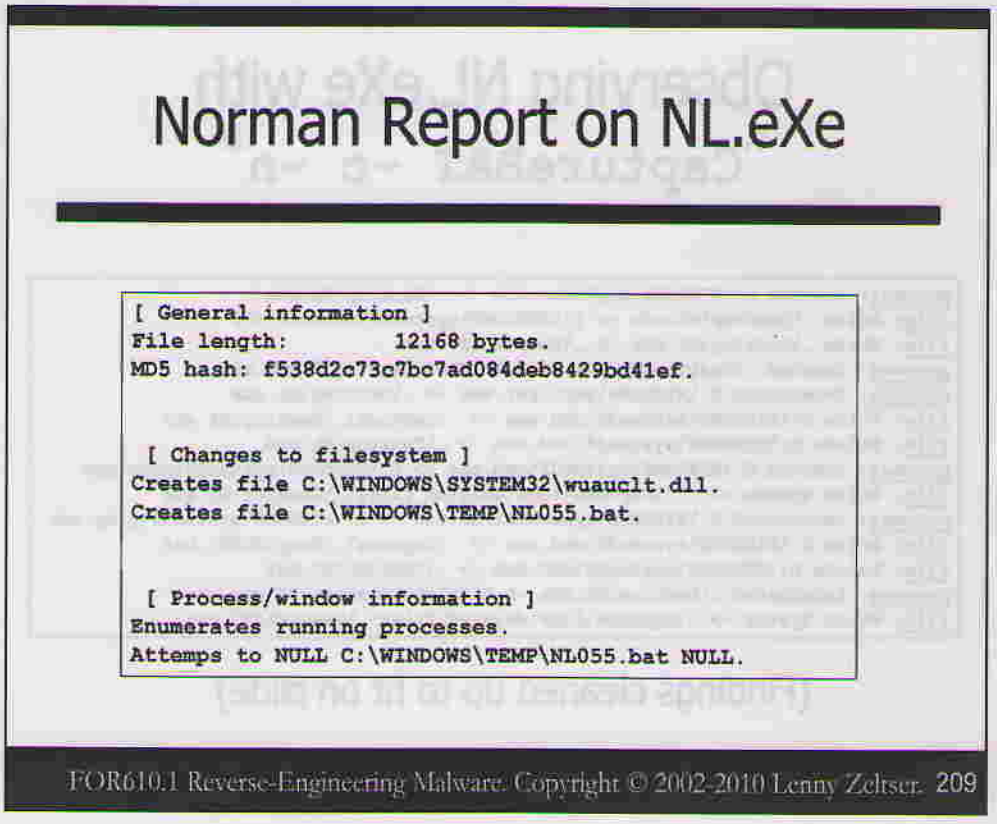


In mid-December 2006, a worm appeared that exploited a vulnerability in Symantec AntiVirus to propagate. When it infected a system, it executed a batch file presented on this slide to retrieve an executable from a remote FTP site. What would you do if you were presented with a batch file above and had only a few minutes to assess the situation?

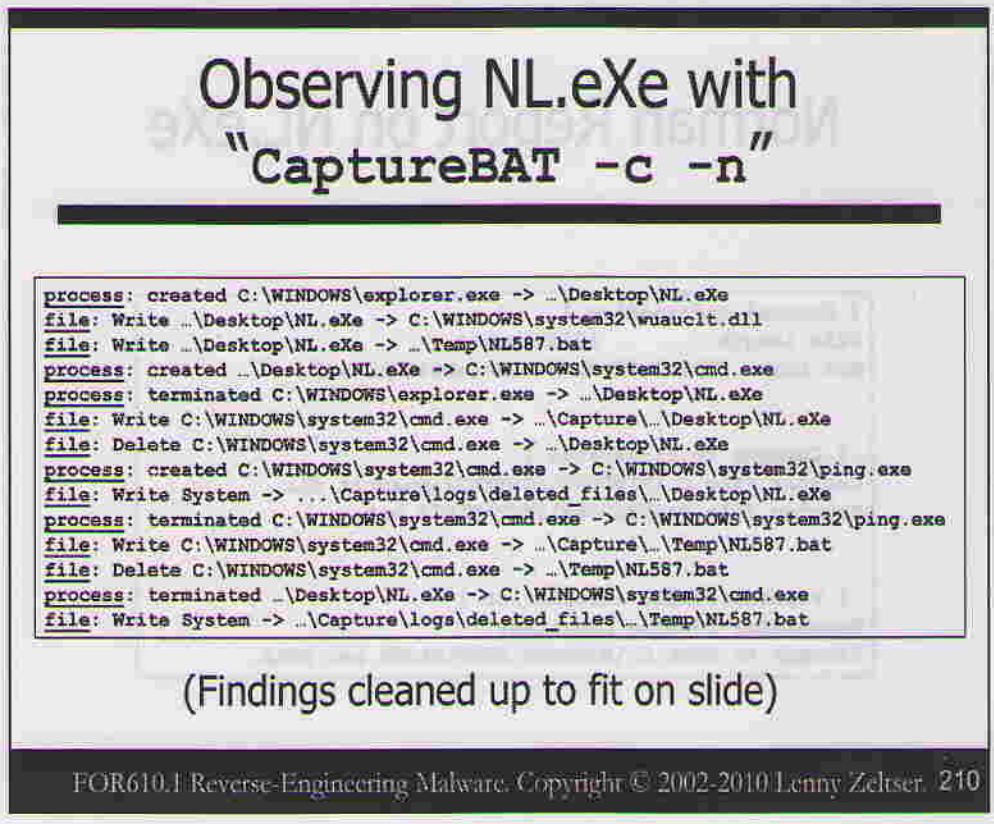
First, you would probably interpret the contents of the batch file and retrieve a copy of the malicious executable, NL.eXe. What's next?



One way to start a quick investigation of a malicious executable is to use a website such as www.virustotal.com to scan it with multiple anti-virus engines. Perhaps some of them will be able to identify the specimen, which will allow you to read up on it to understand it better. As you can see, several anti-virus engines identified this specimen, but it looks like they used generic detection, so we don't have many pointers for gathering additional information. According to some of the names presented by the VirusTotal report, it looks like we're dealing with an IRC bot.



Let's see whether we can gather additional information via a behavioral analysis scan. In this case, Norman Sandbox Live provides a report that tells us which the executable creates and attempts to delete when it infects a system. Now, when we infect our lab system, we have a good sense of what to look for.

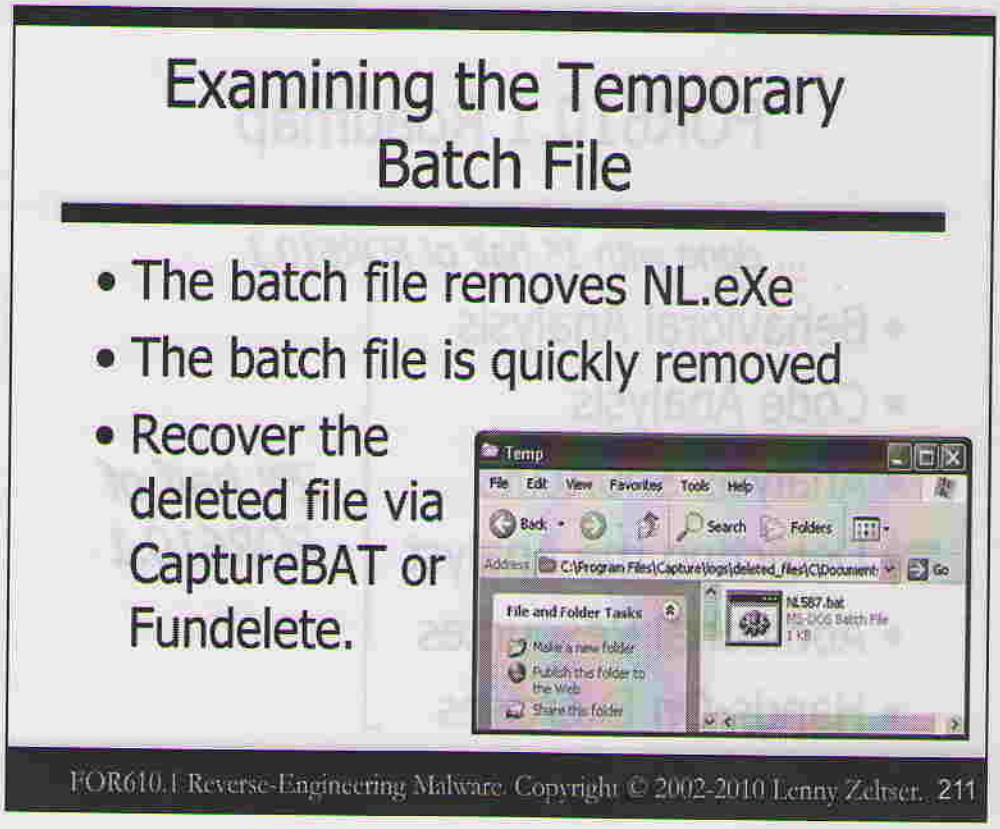


You can also get a sense for the program's activities by infecting a laboratory system with NL.eXe while running Capture BAT in the background. As In this example, we launched CaptureBAT.exe with the "-c" parameter to capture any deleted and modified files, and with the "-n" parameter to capture network activity to and from the infected system.

As you can see on this slide, NL.eXe created wuauclt.dll and NL587.bat files. It also launched cmd.exe, which, in turn, executed ping.exe and then terminated it. The cmd.exe process also deleted NL.eXE and NL587.bat files. You can also observe events that copied the deleted files to Capture BAT's directory; this occurred because we specified the "-c" parameter when invoking CaptureBAT.exe.

Capture BAT also created a .pcap file in its log directory, which we can view using a network sniffer, such as Wireshark. As you can see, the infected system attempts to resolve the hostname "nameless.3322.org". It gets a "port unreachable" message in response, because there is no DNS server in this environment.

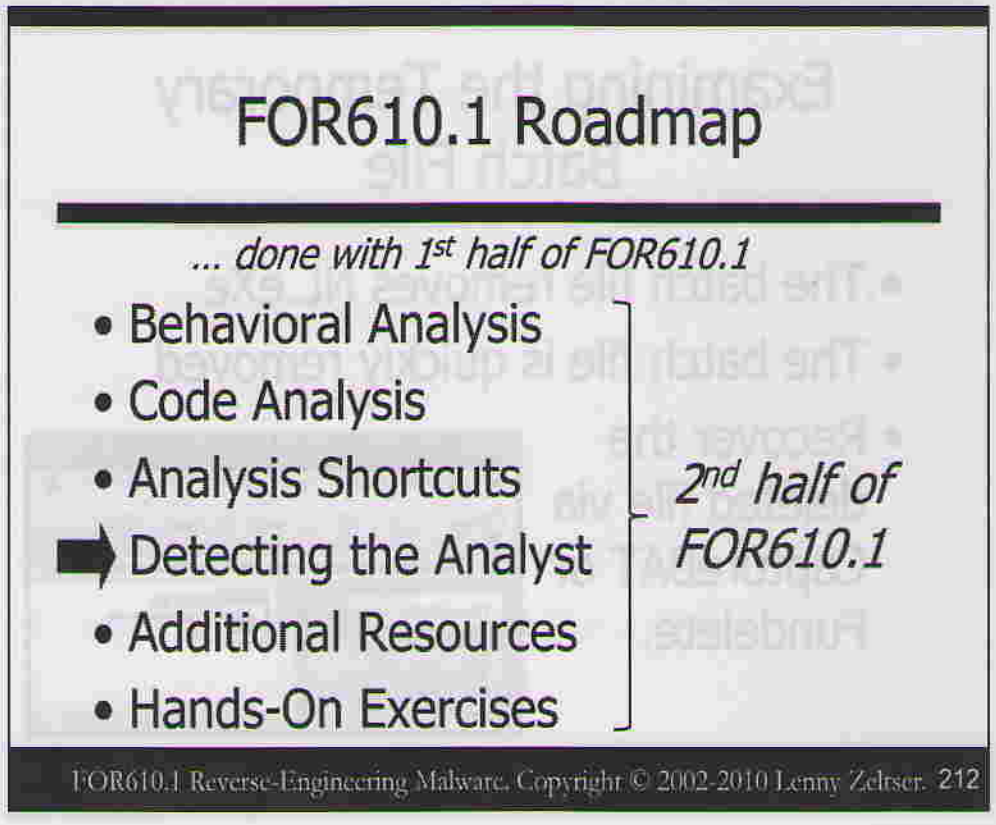
Source	Destination	Protocol	Info
192.168.179.129	192.168.179.1	DNS	Standard query A NameLess.3322.org
192.168.179.1	192.168.179.129	ICMP	Destination unreachable (Port unreachable)



You can now infect your lab system and have a good sense of what to expect. That's great, we don't have much time for this analysis, so we want to keep the surprise factor to a minimum. If you infect a system with NL.eXe, you will notice the two files being created. The wuauclt.dll file turns out to be a regular executable, so you'd need to analyze it as such. We won't go into this analysis, because the goal of these few slides is to focus on on-line analysis tools. You will also find a batch file on the system, but you have to be fast about grabbing a copy of it, because it will get deleted in a matter of seconds. Here are the contents of that file:

```
@ECHO OFF
:Repeat
DEL "C:\Documents and Settings\Administrator\Desktop\NL.eXe"
Ping 0.0.0.0
IF EXIST "C:\Documents and Settings\Administrator\Desktop\NL.eXe" GOTO
Repeat
DEL "%0"
```

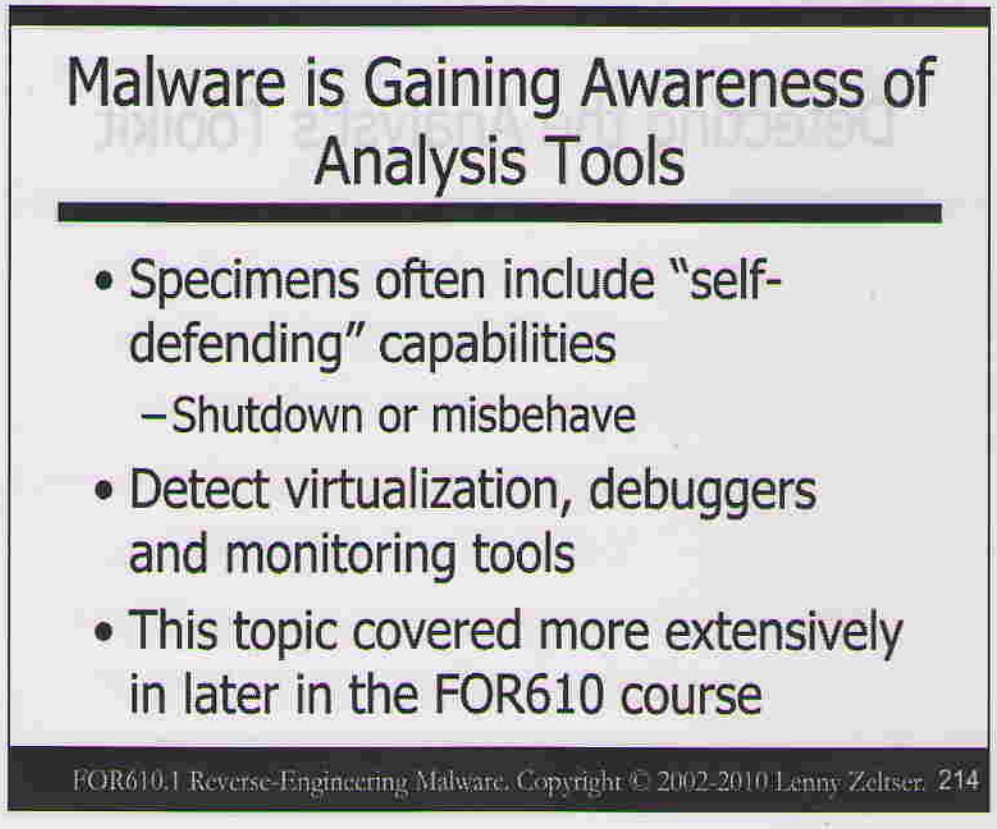
It looks like the purpose of the batch file is to remove the original NL.eXe file from wherever it originally ran from. The command "`Ping 0.0.0.0`" is used to slow down the execution of the batch file before attempting the removal operation again.



The next section discusses some of the ways in which malware can detect presence of the analyst's tools.

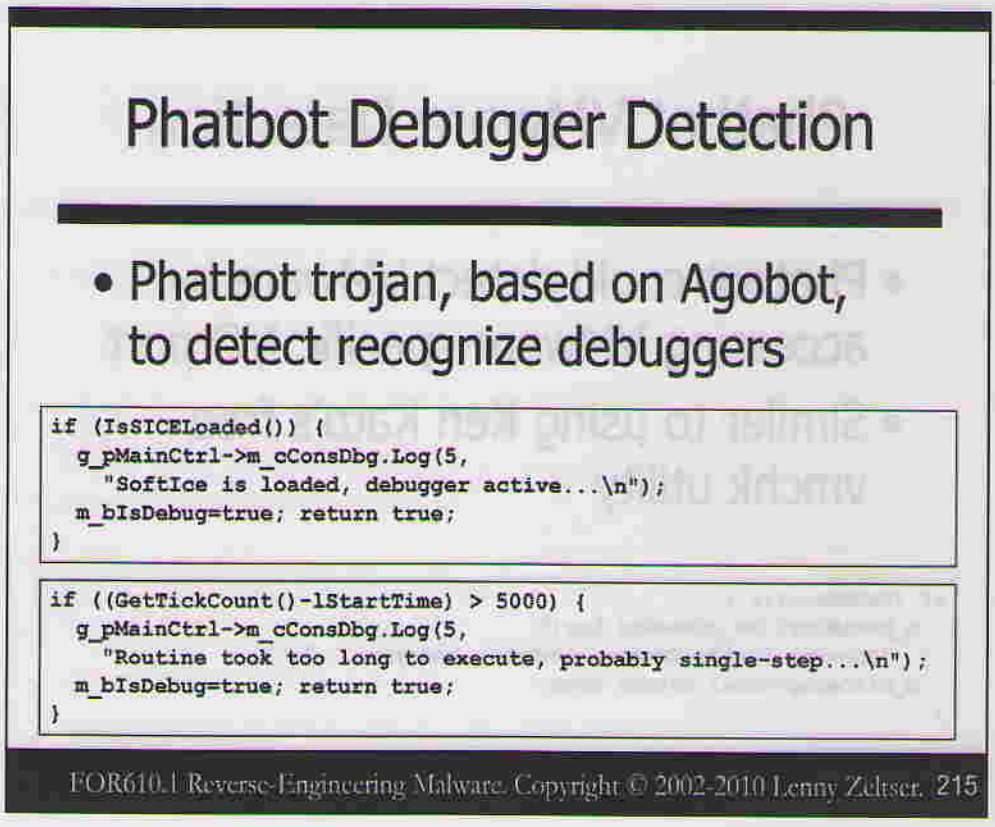
Detecting the Analyst's Toolkit

This page intentionally left blank



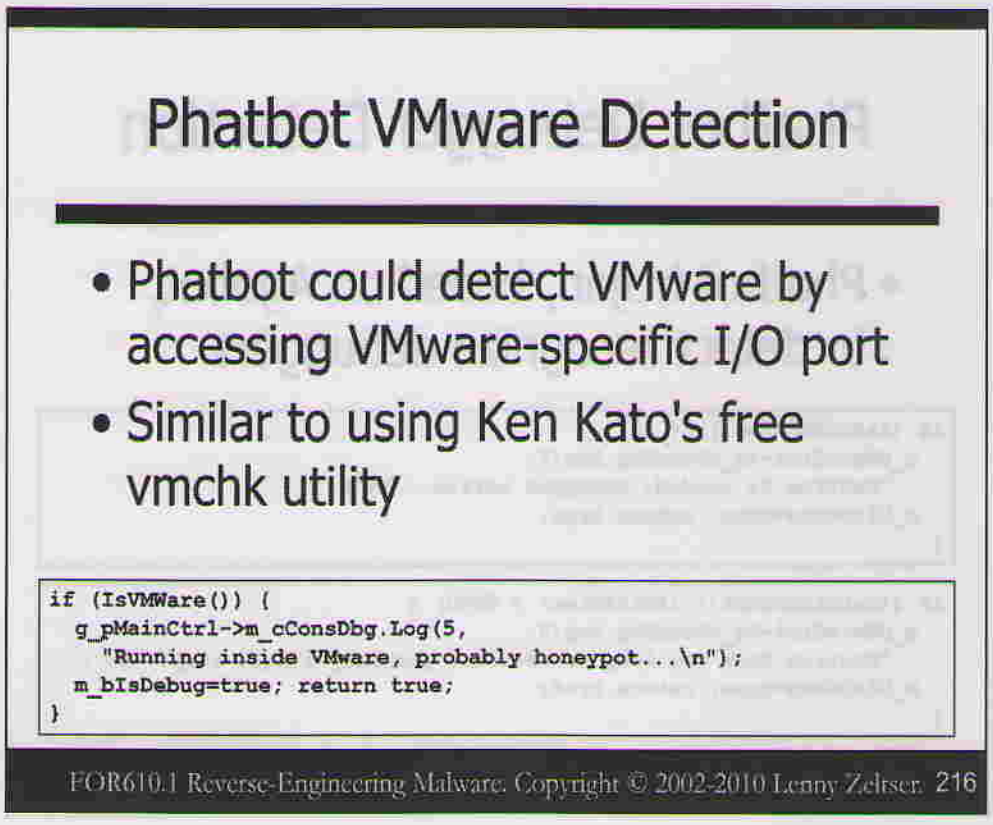
Malware authors increasingly include "self-defending" capabilities into their creations, designed to detect the presence of analyst's tools. For instance, they can check for the presence of VMware, OllyDbg, System Monitor, and other utilities that are part of your toolkit. If malware believes it's being analyzed, it may try to terminate the offending processes, refuse to run, or behave differently than it would have on a regular victim's system.

This topic is examined in greater detail in the second half of the FOR610 course. However, let us take a brief look at the types of functionality you may need to deal with.



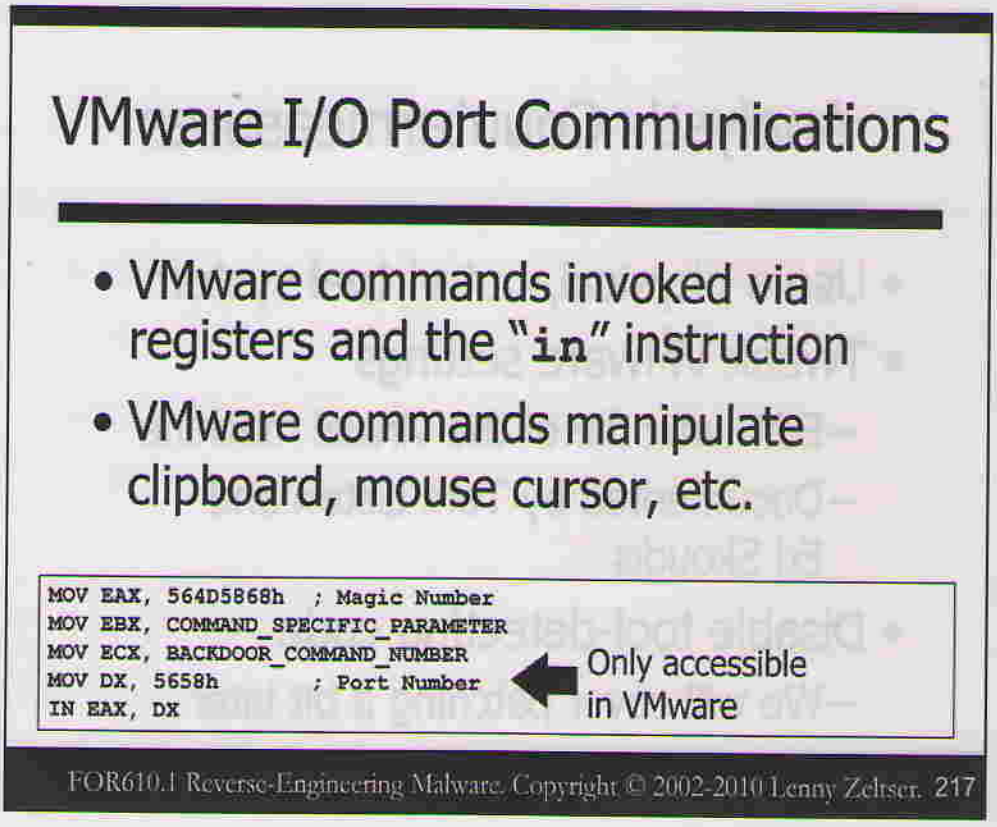
As the use of packers demonstrates, malware authors are taking measures to make the analysis of their executables more difficult. Some specimens have the ability to detect whether a debugger is running on the system and, if so, either refuse to run or attempt to disable the debugger.

On this slide you see a code fragment from a Phatbot trojan variant. These are some of the ways in which Phatbot can detect the presence of SoftICE—a once popular debugger—as well as the fact that someone may be stepping through the program using a generic debugger.



Phatbot also has code for detecting whether it is operating within a VMware environment. VMware detection is performed by the IsVMWare() routine. The implementation of that routine is very similar to the way a freely available tool called vmchk is able to detect the presence of malware.

You can download vmchk and related utilities for free from the following URL:
http://chitchat.at.infoseek.co.jp/vmware/vmtools.html



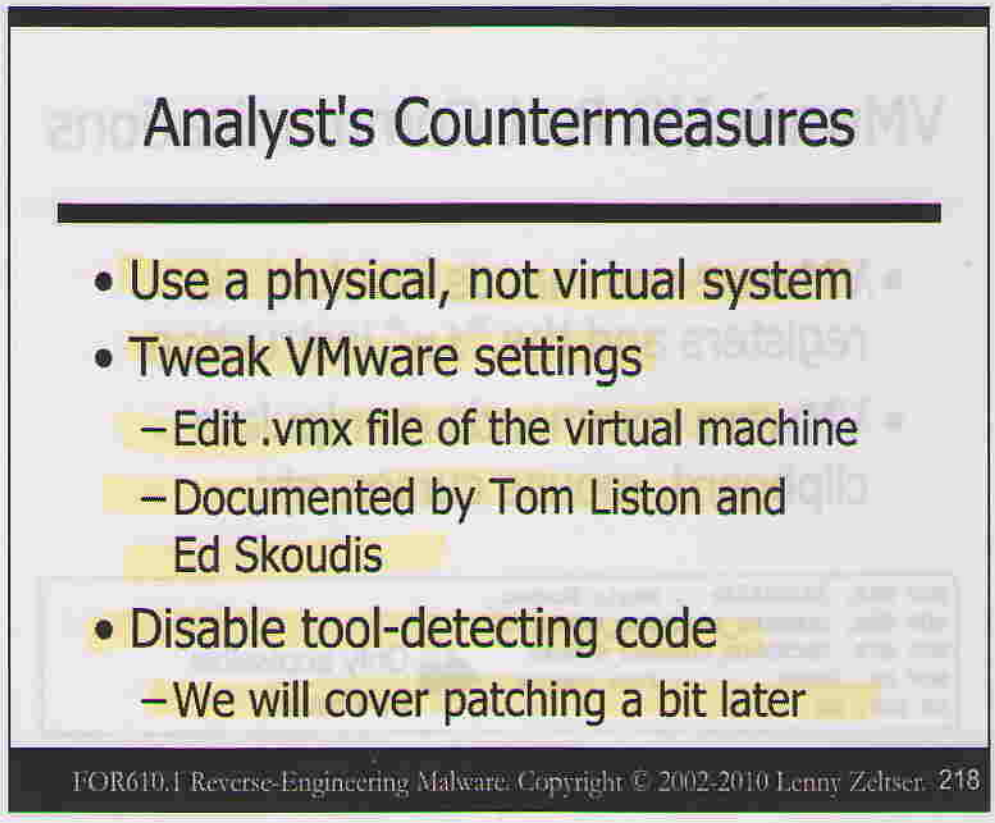
VMware virtual machines exchange VMware-specific information with each other using a special I/O port that doesn't exist on non-VMware systems. Some of the VMware functions that take advantage of this communication mechanism are mouse cursor positioning, the exchange of clipboard data, device configuration, and so on.

The assembly code fragment displayed on this slide can, when executed in a VMware virtual machine, control VMware-specific functions that I mentioned in the previous paragraph. I obtained this code from the following URL, which also contains a detailed explanation of VMware I/O port communication mechanisms.

http://chitchat.at.infoseek.co.jp/vmware/backdoor.html

The "`in`" instruction is a way for assembly code to copy data from the I/O port specified by the second operand (the DX register, in this case), to the destination register (EAX, in this case). A regular "`in`" command would not look at the contents of EAX, EBX, and ECX registers; however, the "`in`" command emulated by VMware is able to interpret the contents of those registers and process them in a VMware-specific way.

By employing the assembly code such as the fragment displayed on this slide, malware authors can detect the presence of the VMware I/O port to detect when their code executes within a VMware-based sandbox.

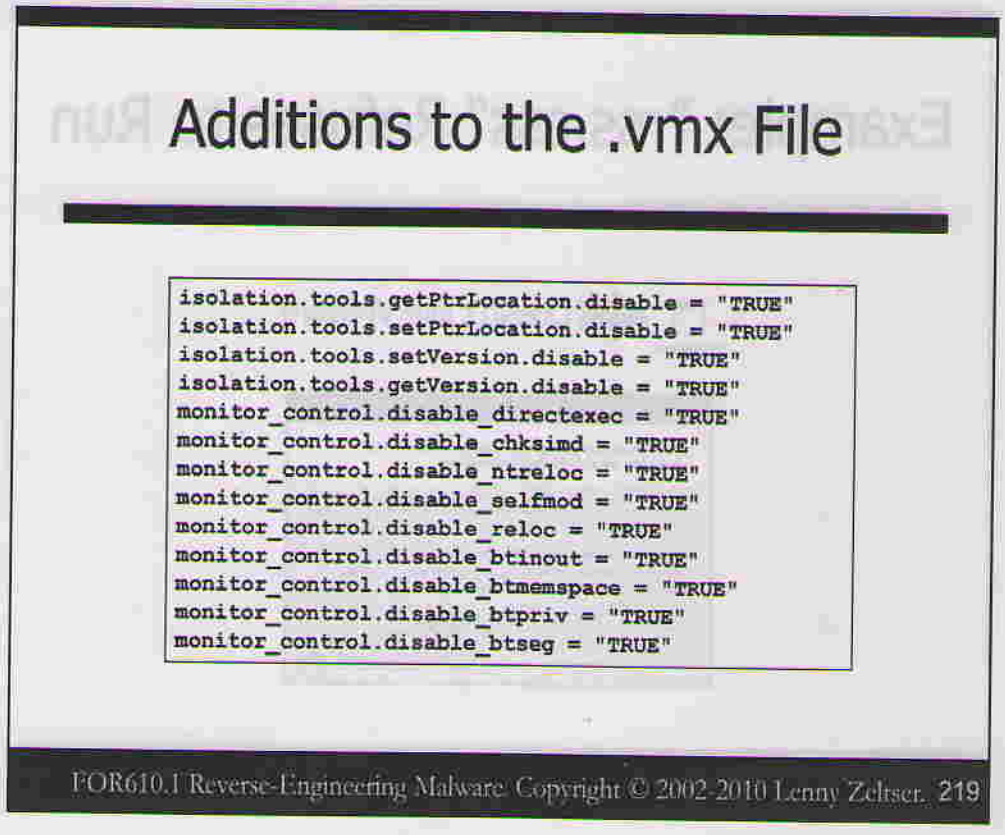


There are several ways around mechanisms that detect the presence of our malware analysis tools. For instance, you could use a dedicated physical system for performing your analysis, instead of using VMware.

Another option is to edit the malicious executable, so that the code that detects VMware never has a chance to run. This technique can remove debugger-detecting functionality from the specimen as well, which makes it particularly powerful. Editing compiled executables is known as patching—we will cover this topic in another section of the course.

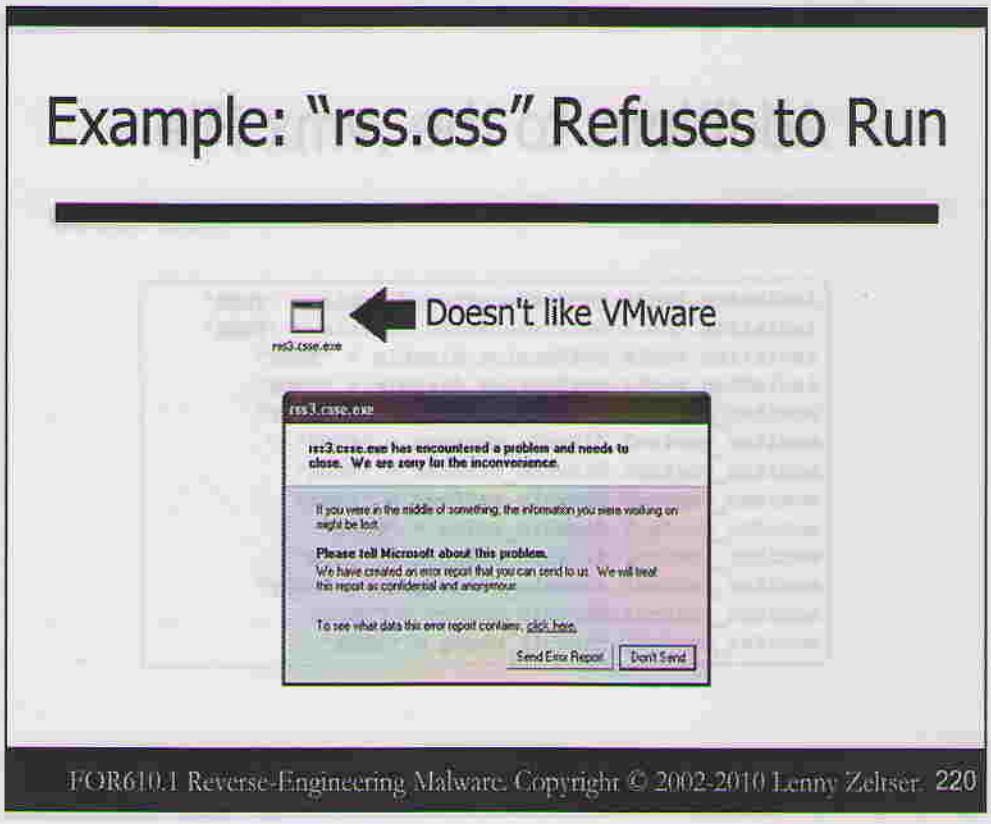
You have another option for VMware-aware code. Tom Liston and Ed Skoudis documented unofficial settings you can add to the .vmx file of the virtual machine to make it much more difficult for malicious code to detect the presence of VMware. Tom and Ed have documented these options, and various techniques for detecting malware, in the following paper:

http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf



These are the options that Tom and Ed documented in their paper. To take advantage of these settings, locate the .vmx file that defines how your virtual machine should behave. This file will be located on your physical host in the directory that stores the other files that belong to the virtual machine; the file's name will be something like "Windows XP Professional.vmx", although the exact name may differ. You can use a regular text editor, such as Notepad, to edit the .vmx file. Simply append the lines shown on this slide to the bottom of the .vmx file.

Be sure to shut down your virtual machine before editing its .vmx file. Also, I recommend making a backup copy of your original .vmx file, to make it easier to revert to the earlier configuration. Finally, be aware that adding these settings to the .vmx file is not supported by VMware. VMware tools will refuse to run on the virtual machine, and the machine's performance will slow down significantly.



Let's take a quick look at a situation where editing the .vmx file in this manner can come in handy. In this situation, I came across a file called rss.css. This file was being downloaded to the victims' computers using an exploit that took advantage of a vulnerability in Microsoft Data Access Components (MDAC).

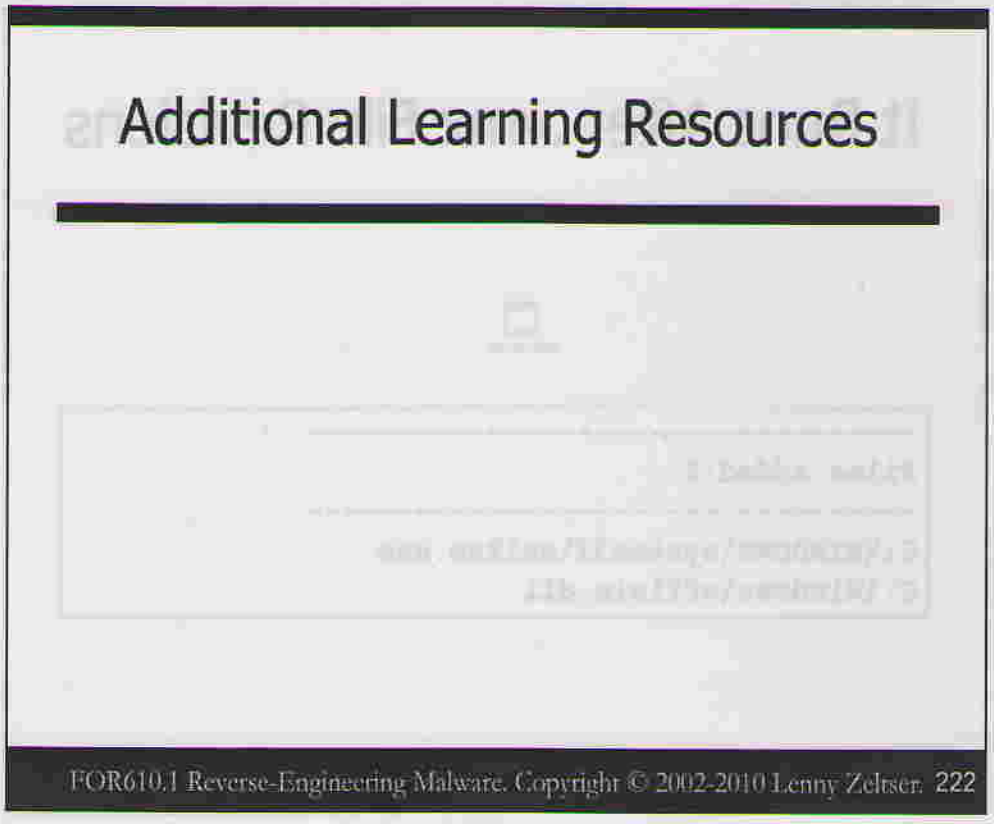
Even though the file was called rss.css, it was just a regular Windows executable. When I attempted to run it in my VMware-based lab, the executable refused to run. It crashed with the unfriendly error shown on this slide. What to do?

It Runs After .vmx File Revisions

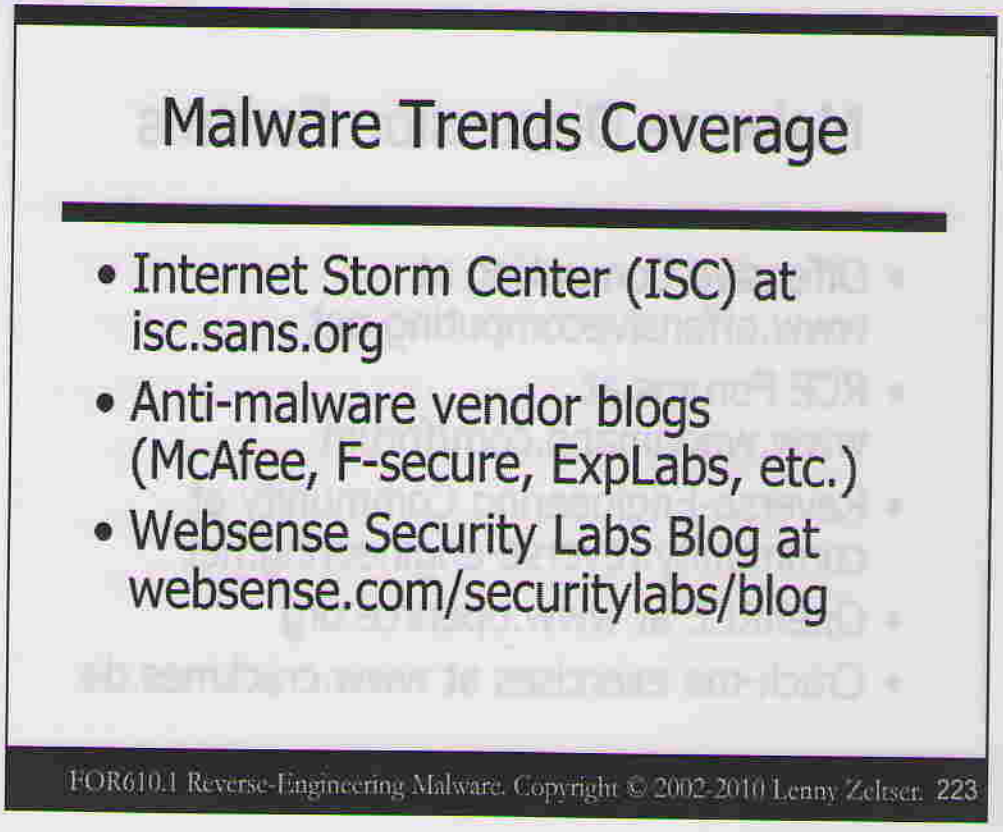

```
--------------------------------
Files added:2
--------------------------------
C:\WINDOWS\system32\on1Exe.exe
C:\Windows\off1win.dll
```


I edited the .vmx file for the virtual machine I tried to infect, appending to it the lines I showed a few slides ago. I booted into the virtual machine and tried infecting it again. It worked! As you can see in the excerpt from the RegShot report, shown on this slide, the malicious executable placed a couple of files on the infected system.

That's as much as I'd like to say about the rss.css file. It turned out to be a relatively obscure spyware specimen, which was particularly popular in Asia. The reason I wanted to cover it briefly in this section was to demonstrate the effectiveness of the .vmx file-editing technique.

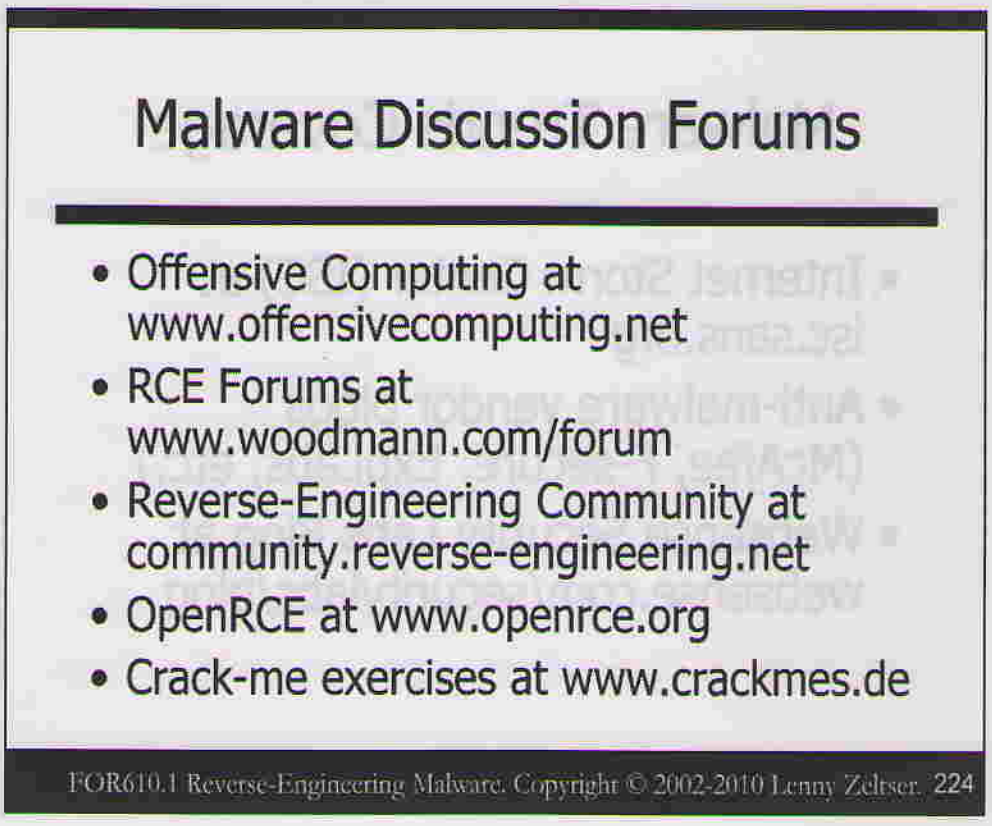


Let's take a brief look at a few information resources that come in handy when analyzing malware.



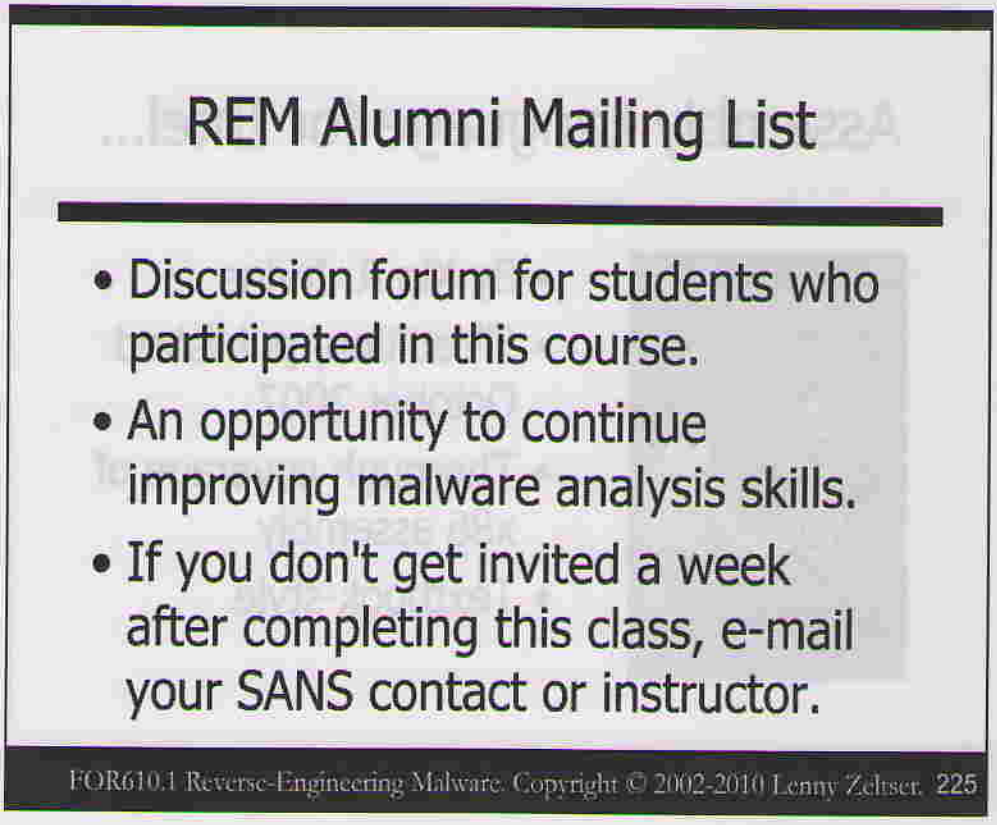
Once you know something about the malware specimen you'd like to understand, it is often a good idea to search the Web in case someone has already analyzed it. In addition to using your favorite search engine, you should also visit specialized sites, such as those listed on this and the following slides.

Many anti-malware companies maintain blogs that offer timely information regarding malware-related activity and trends.

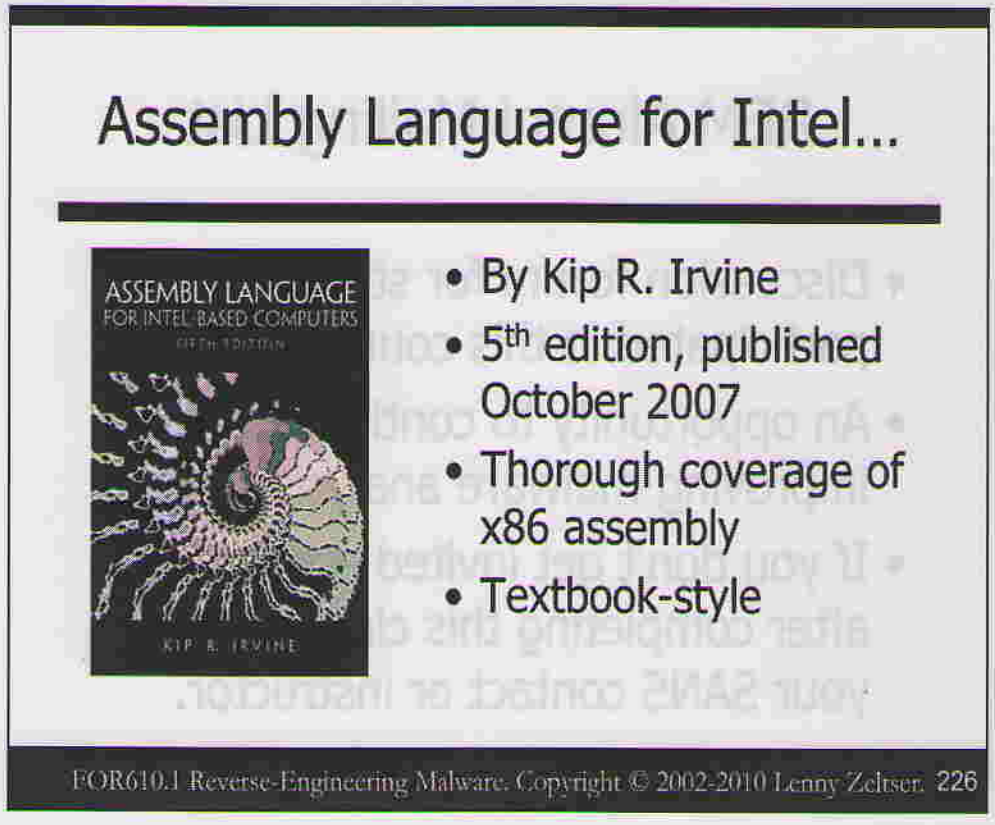


To stay current with malware developments, as well as to have a forum for asking questions and offering answer, I recommend keeping an eye on the following discussion forums:

- The Offensive Computing site at http://www.offensivecomputing.net
- RCE Forums at www.woodmann.com/forum
- The Reverse-Engineering Community forum at http://community.reverse-engineering.net
- The OpenRCE site at http://www.openrce.org
- Crack-me exercises at http://www.crackmes.de

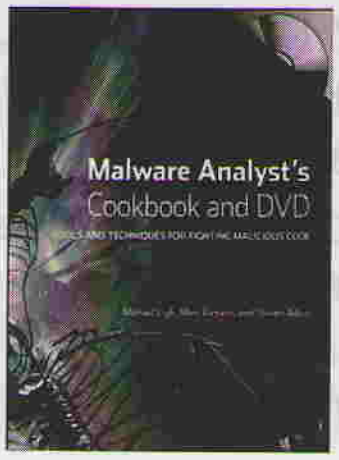


SANS also set up the REM mailing list for individuals who participated in this course, to give you the forum for continuing to improve malware analysis skills after completing this class.



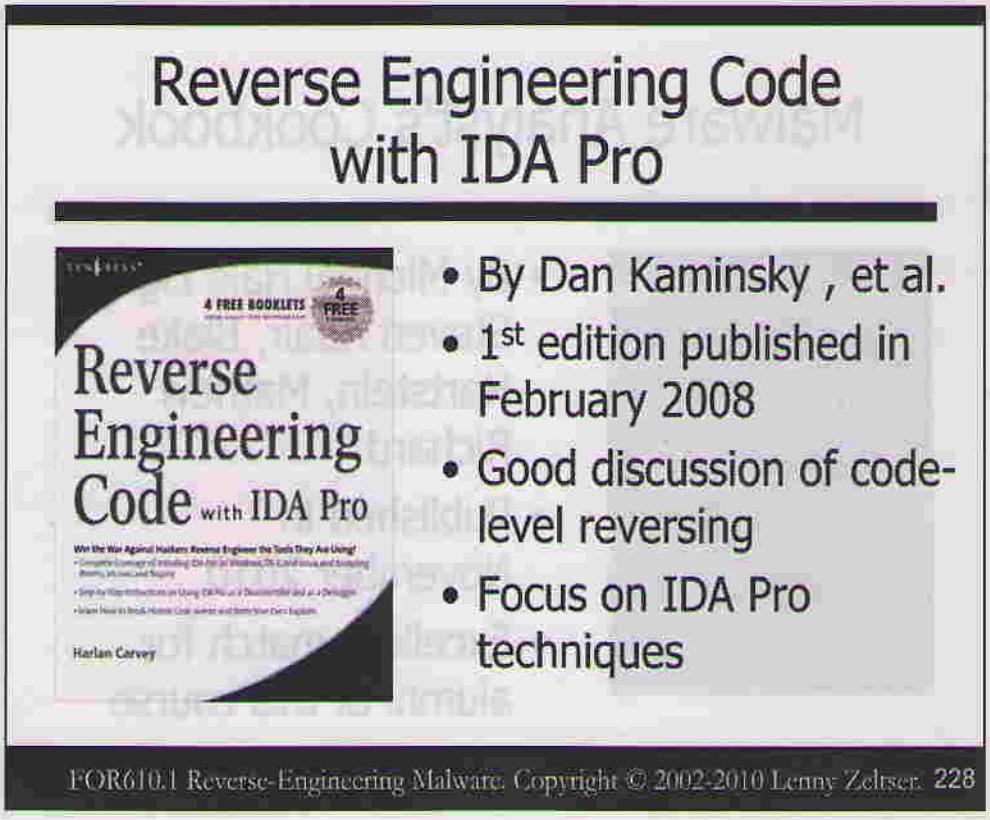
Assembly Language for Intel-Based Computers, by Kip R Irvine, is a good text-book style volume learning x86 assembly. It's well regarded, thorough, and I like the way it presents the material. This book is pricey, though.

Malware Analyst's Cookbook



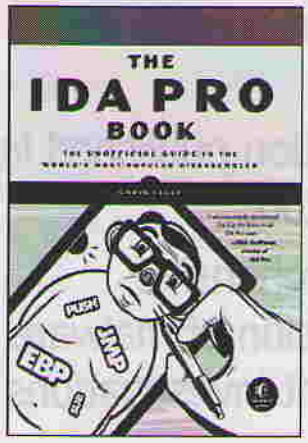
- By Michael Hale Ligh, Steven Adair, Blake Hartstein, Mathew Richard
- Published in November 2010
- Excellent match for alumni of this course

Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code is presents numerous practical tips, tools and techniques for analyzing malicious software. It is highly accurate and innovative, and is an excellent match for alumni of this course.



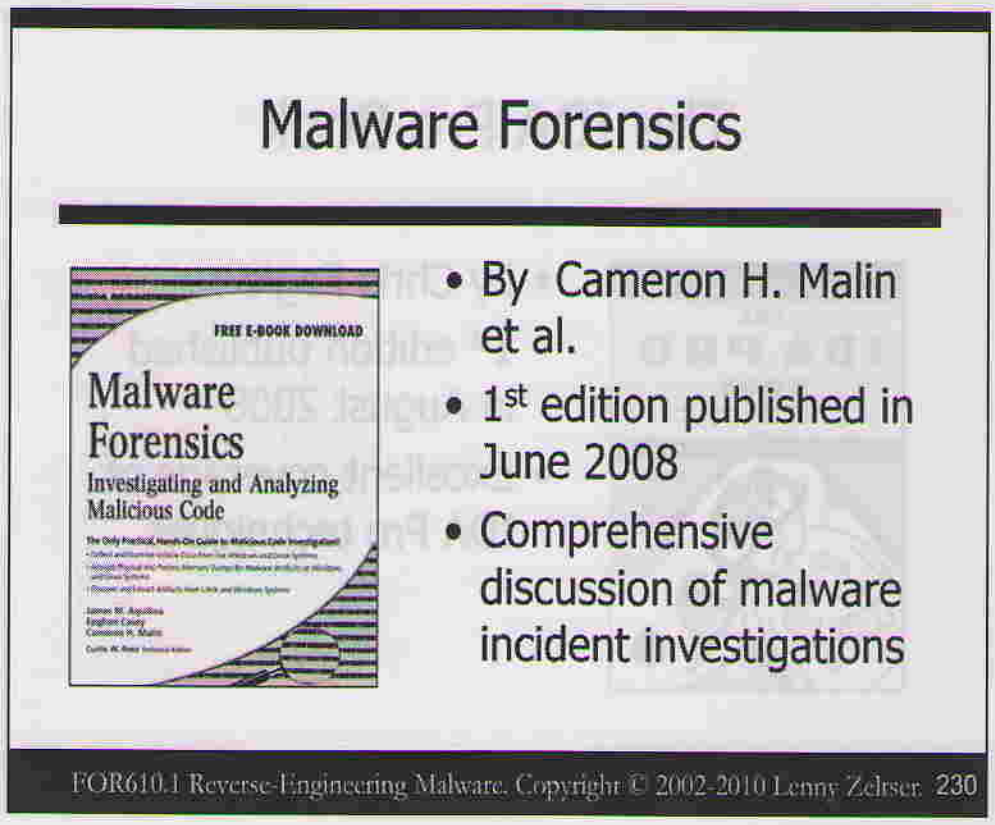
Reverse Engineering Code with IDA Pro, by Dan Kaminsky, Justin Ferguson, Jason Larsen, Luis Miras, and Walter Pearce provides excellent coverage for using IDA Pro for reverse-engineering software, and does so in the context of malware. You may also find it useful for its discussion of x86 assembly fundamentals.

The IDA Pro Book

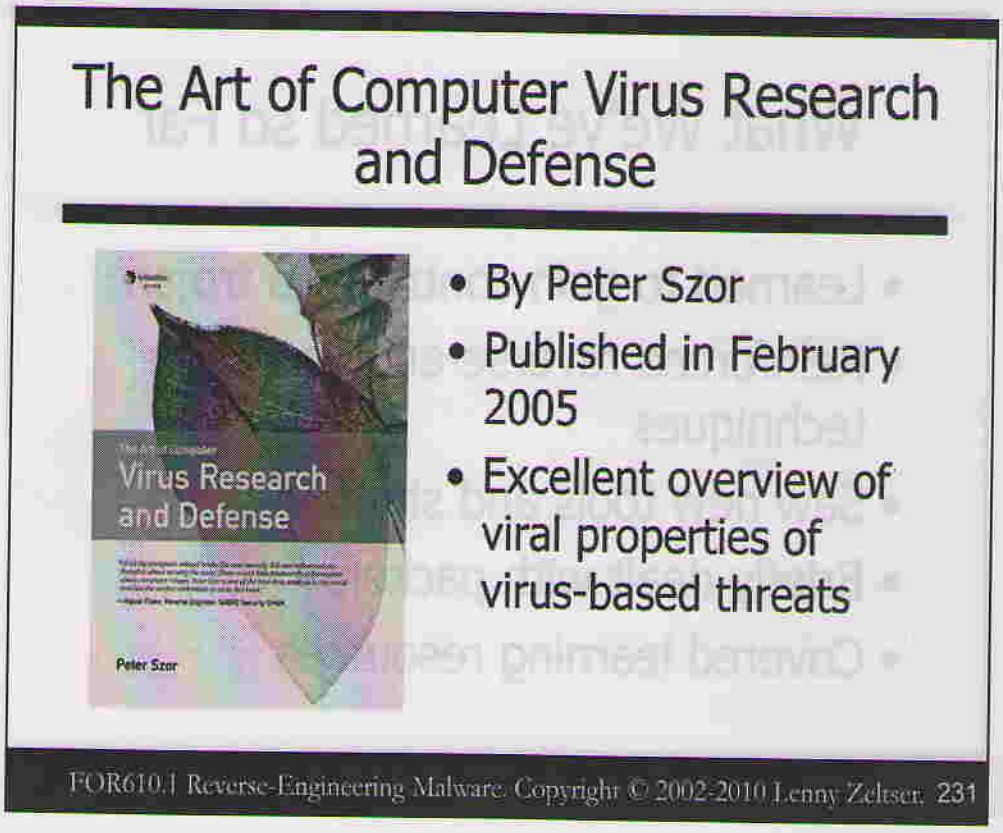


- by Chris Eagle
- 1st edition published in August 2008
- Excellent coverage of IDA Pro techniques

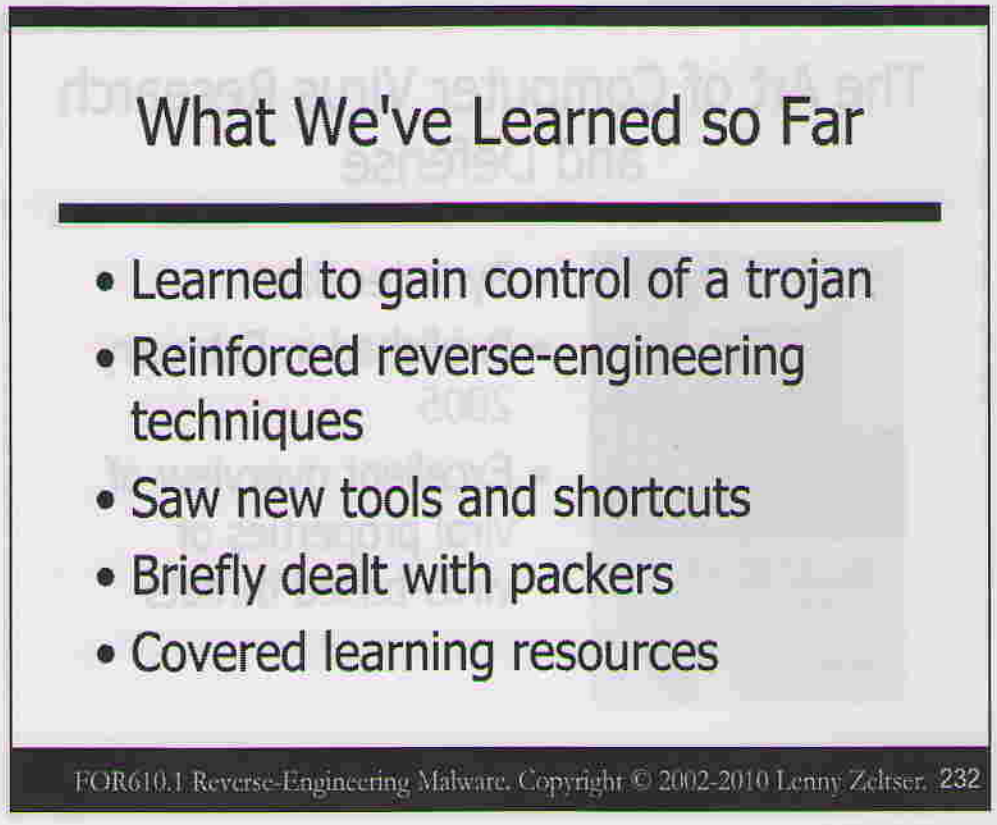
The IDA Pro Book by Chris Eagle is lauded for its coverage of IDA Pro, and serves as the "missing manual" for the popular disassembler.



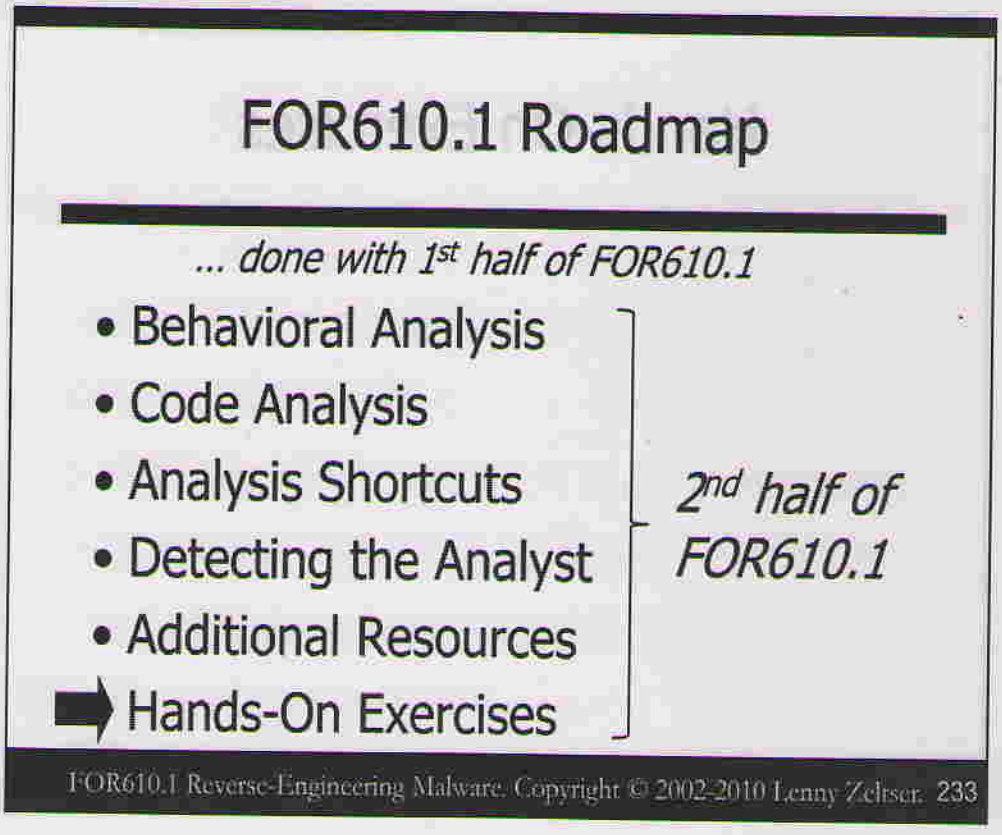
Malware Forensics: Investigating and Analyzing Malicious Code by Cameron H. Malin, Eoghan Casey, and James M. Aquilina offers a comprehensive discussion of topics related to malware incident investigations for both Windows and Unix platforms.



The Art of Computer Virus Research and Defense, by Peter Szor, offers an excellent, and deeply technical overview of viral properties of virus-based malicious code. I highly recommend this book for those who want to understand more about infection mechanisms of malware.



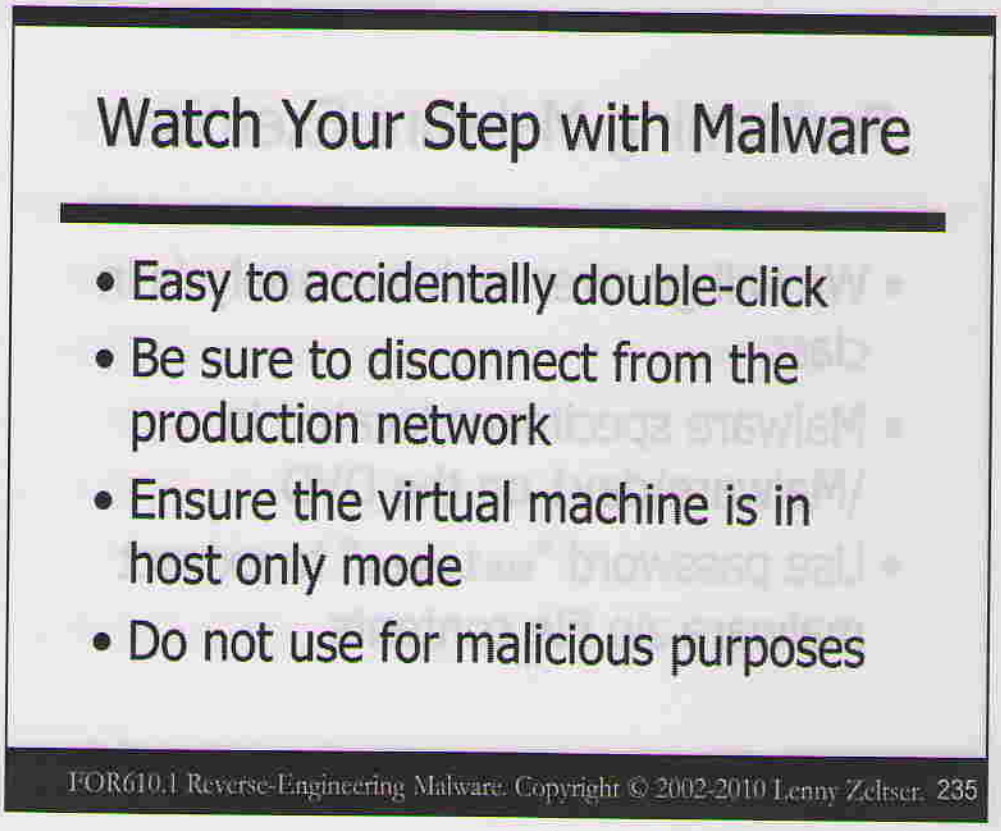
This brings us to the end of the second half of FOR610.1. In this section of the course we focused on techniques for gaining control over a trojan running in our lab, and reinforced the reverse-engineering methodology introduced in the previous section. We also had our first encounter with a UPX-packed executable; in the next section we will spend more time dealing with packed executables. Finally, we took a look at additional tools and resources that can assist us with malware analysis.



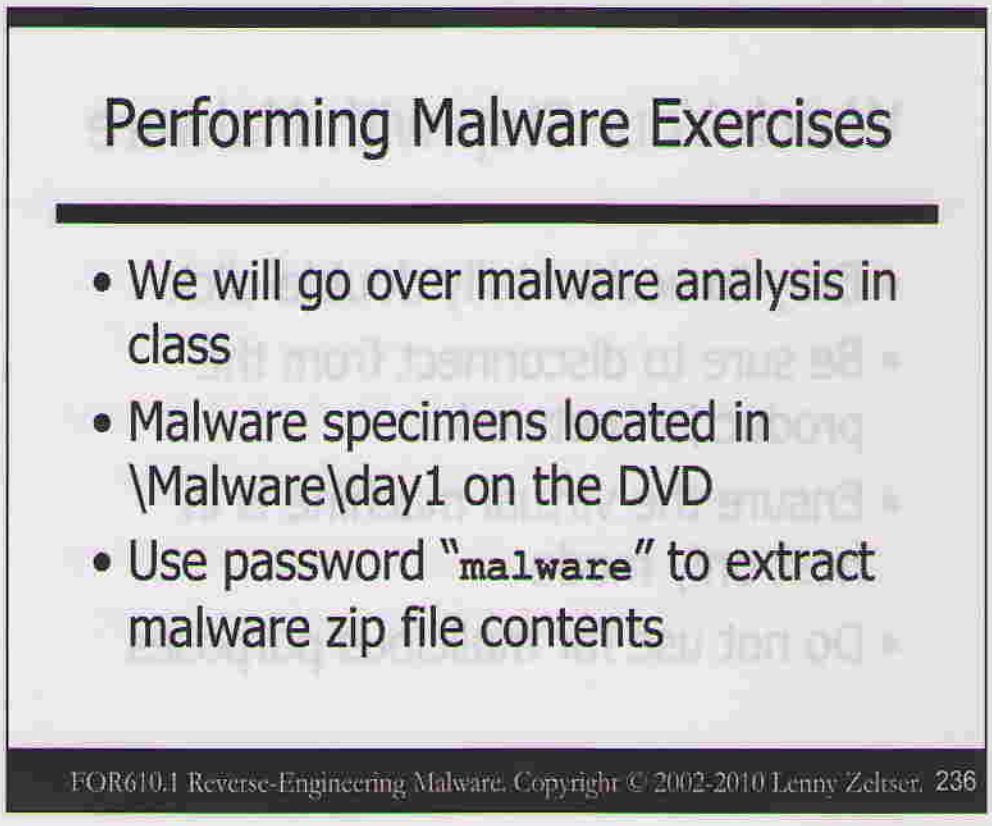
The next section discusses some of the ways in which malware can detect presence of the analyst's tools.

Hands-On Exercises

This page intentionally left blank.



Hands-on exercises for this course involve real-world malware code that is dangerous and needs to be handled with care. Please follow isolation guidelines and common sense precautions to ensure that you do not accidentally impact your production environment.



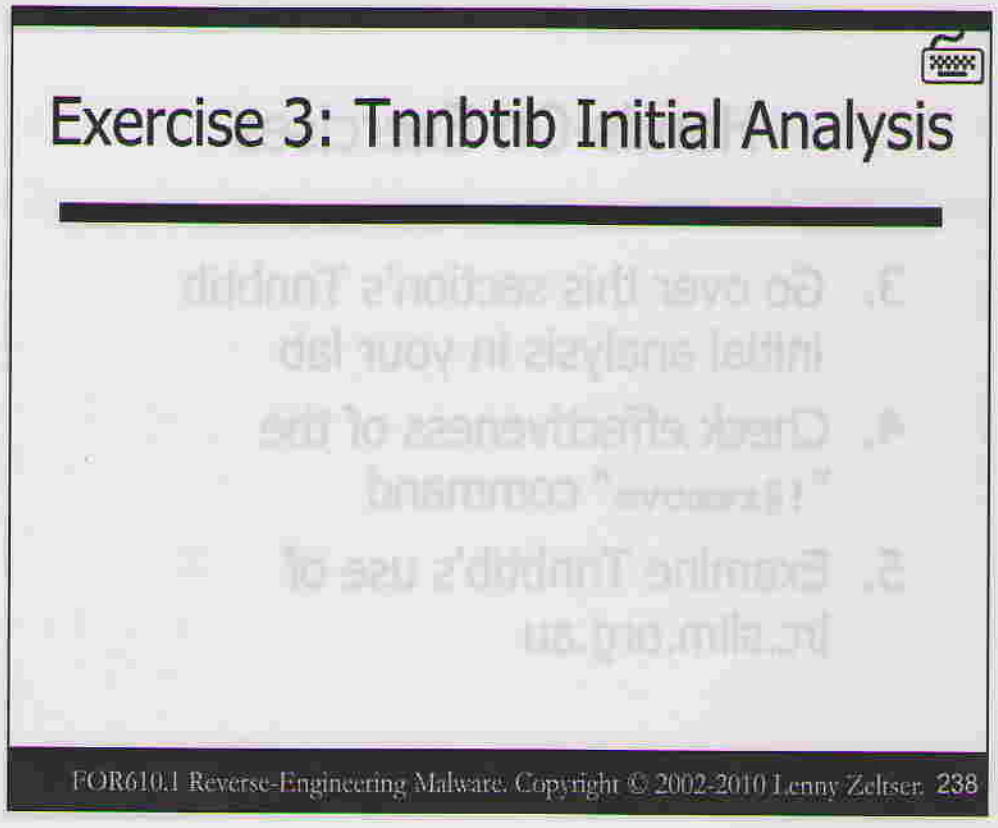
We will go over the solutions to these exercises in class.

The malware specimens used for these exercises are located in the \Malware\day1 directory on the DVD you received for this course. Each specimen is in a dedicated zip file that is protected with the password "`malware`" to help prevent accidental execution and anti-virus detection.

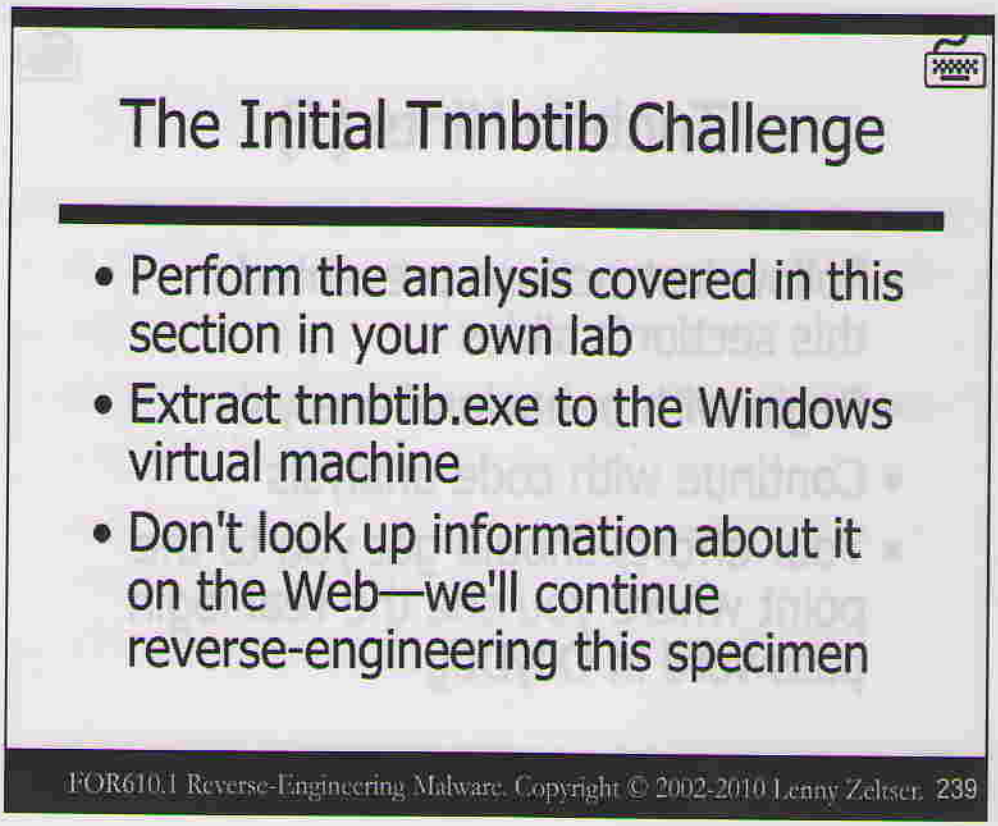
Hands-On Exercises

3. Go over this section's Tnnbtib initial analysis in your lab
4. Check effectiveness of the "`!@remove`" command
5. Examine Tnnbtib's use of irc.slim.org.au

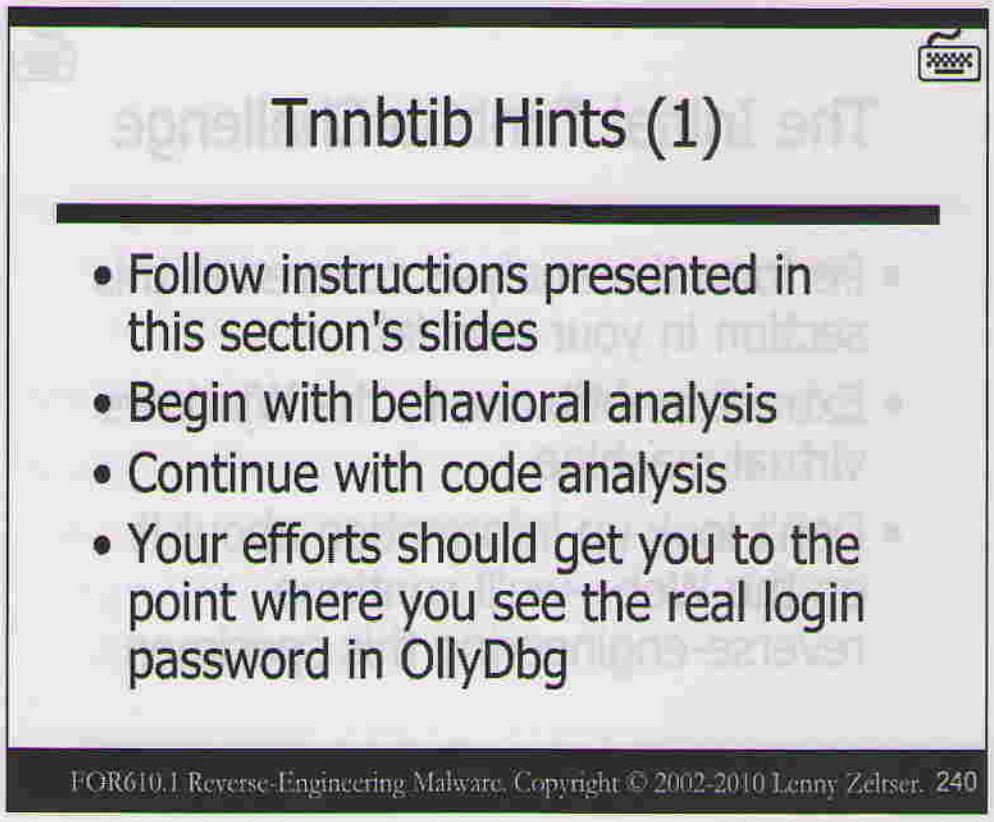
Please complete these exercises before starting the next section of this course. These exercises are described in greater detail in subsequent slides.



In this exercise we will review the Tnnbtib analysis presented in this section.

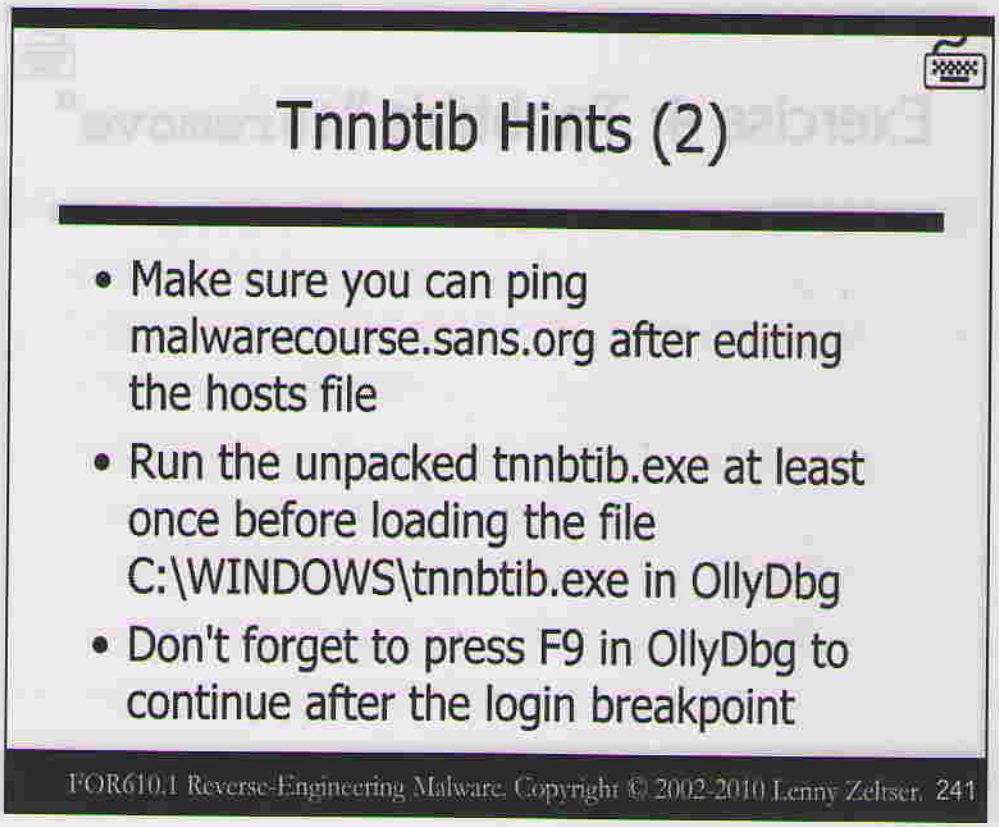


In this section I demonstrated several techniques for analyzing the capabilities of Tnnbtib. Please go over this section's slides and implement these analysis steps in your own lab.



Use the Tnnbtib slides from this section as a step-by-step guide for analyzing Tnnbtib. Begin with the behavioral analysis stage, and continue with code analysis. At the end, you should end up with OllyDbg intercepting the authentication attempt and revealing the trojan's login password.

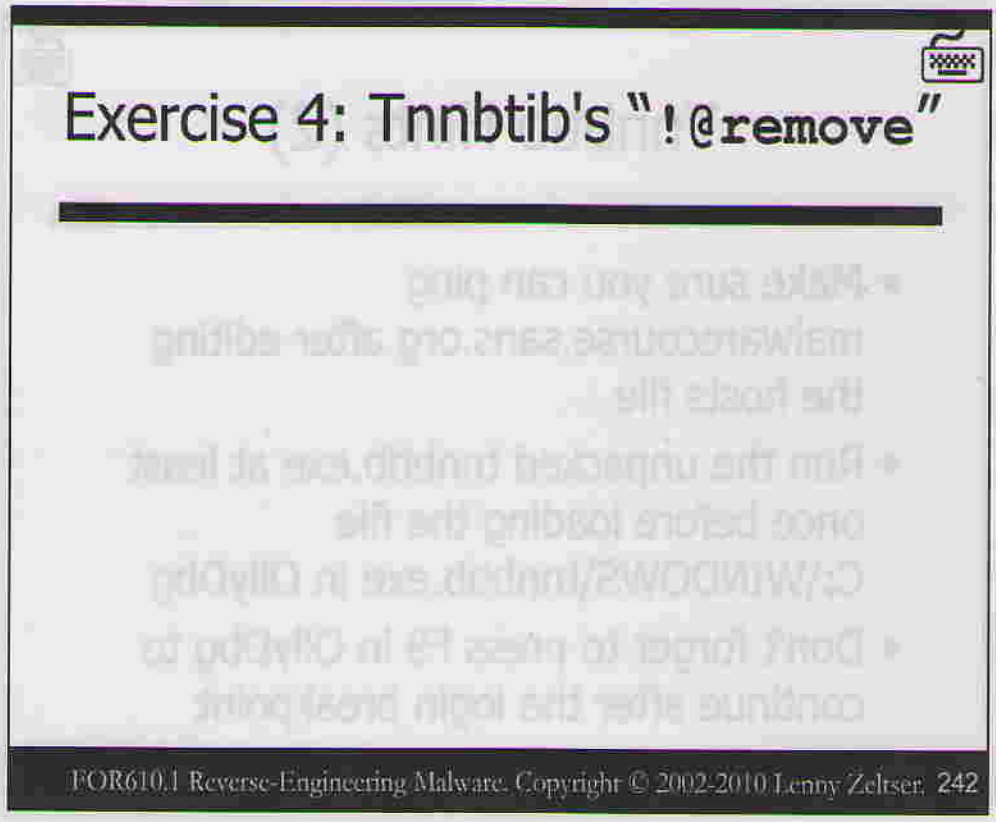
If you wish to experiment with Tnnbtib, authenticate to it with the real password and see what you can do with it.



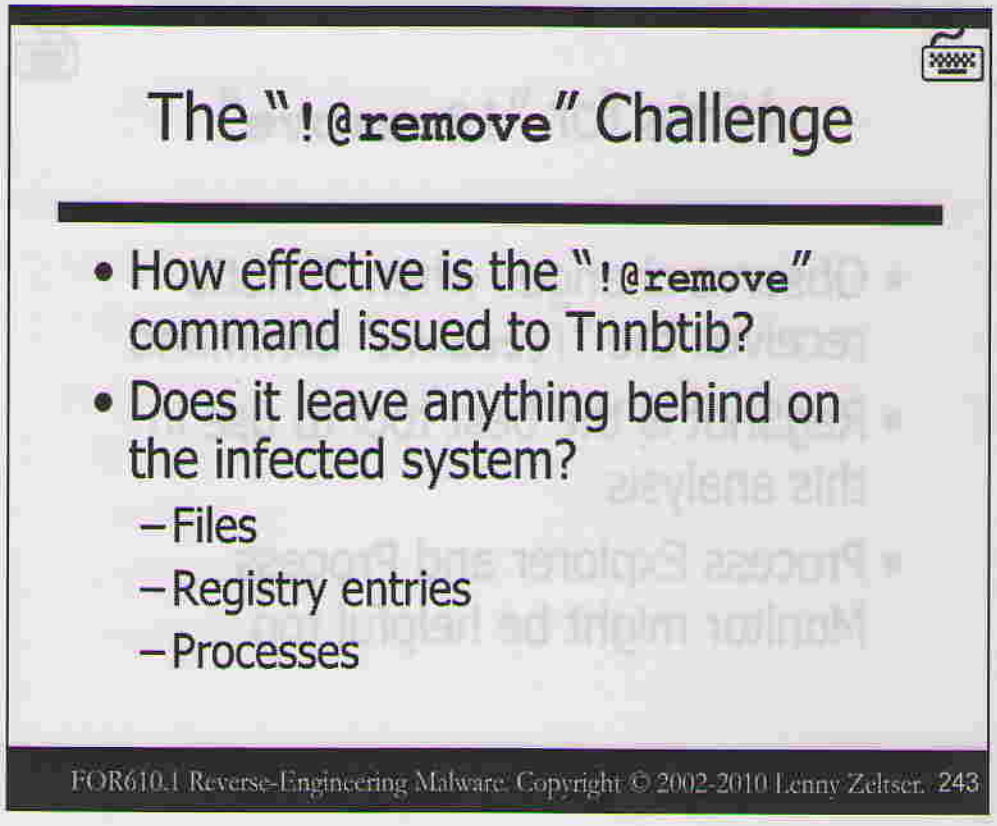
After editing the hosts file on the infected system, check to make sure you didn't make a typo, by pinging the hostname you just defined in the hosts file. Only once you get ping to work should you launch the malware specimen again.

Make sure that you're analyzing the unpacked version of Tnnbtib that we obtained via the "`upx -d`" command. An easy way to do this is to run the decompressed version once; it will then place itself, in the decompressed form, into C:\Windows.

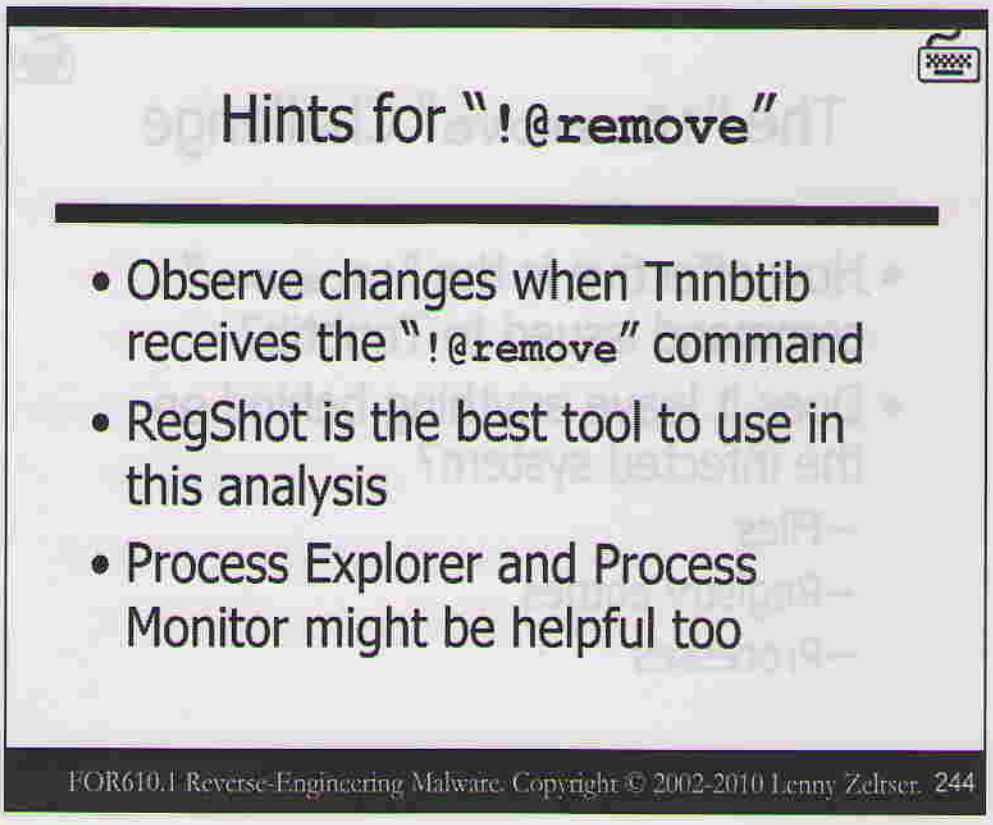
Don't forget to hit F9 in OllyDbg after each authentication attempt. OllyDbg will pause the execution of Tnnbtib when it reaches the breakpoint, and you will need to press F9 before Tnnbtib times out from the IRC session.



In this exercise we will make use of the newfound control over Tnnbtib, and attempt to have it automatically clean the system it infected.

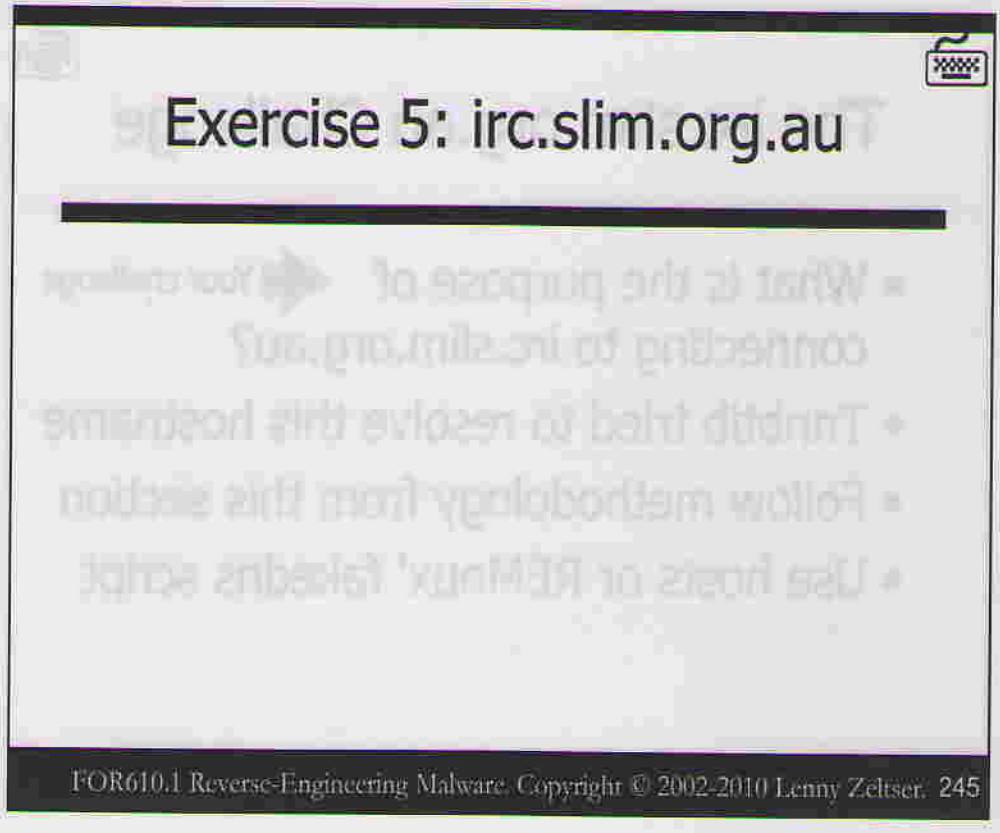


Tnnbtib seems to support the "`!@remove`" command. This command's name suggests that it will remove Tnnbtib from the infected computer. Test whether this is, indeed, the case. Does Tnnbtib cleanly uninstall itself, or does it leave something behind?

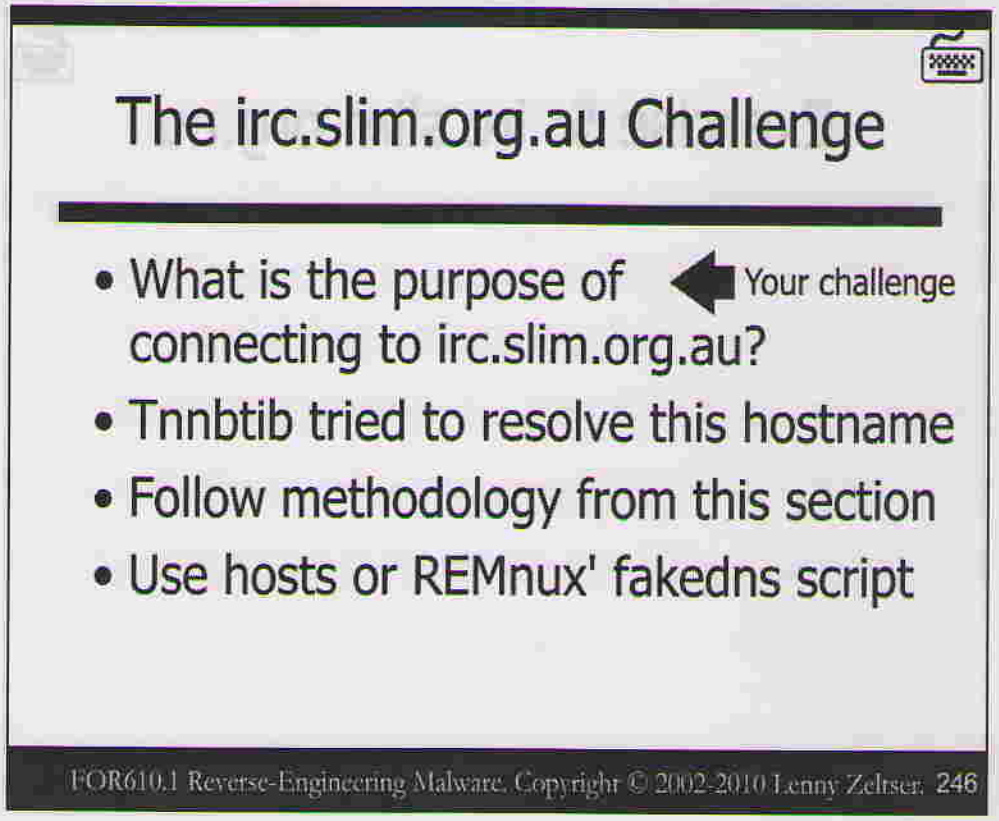


To assess the effectiveness of the "`!@remove`" command, first have Tnnbtib access its IRC channel, then connect to the channel yourself and authenticate to the trojan.

Use system monitoring tools to peek at Tnnbtib's activity when you issue the "`!@remove`" command to it over the IRC session.



In this exercise we will examine the purpose of Tnnbtib's attempt to reach irc.slim.org.au.

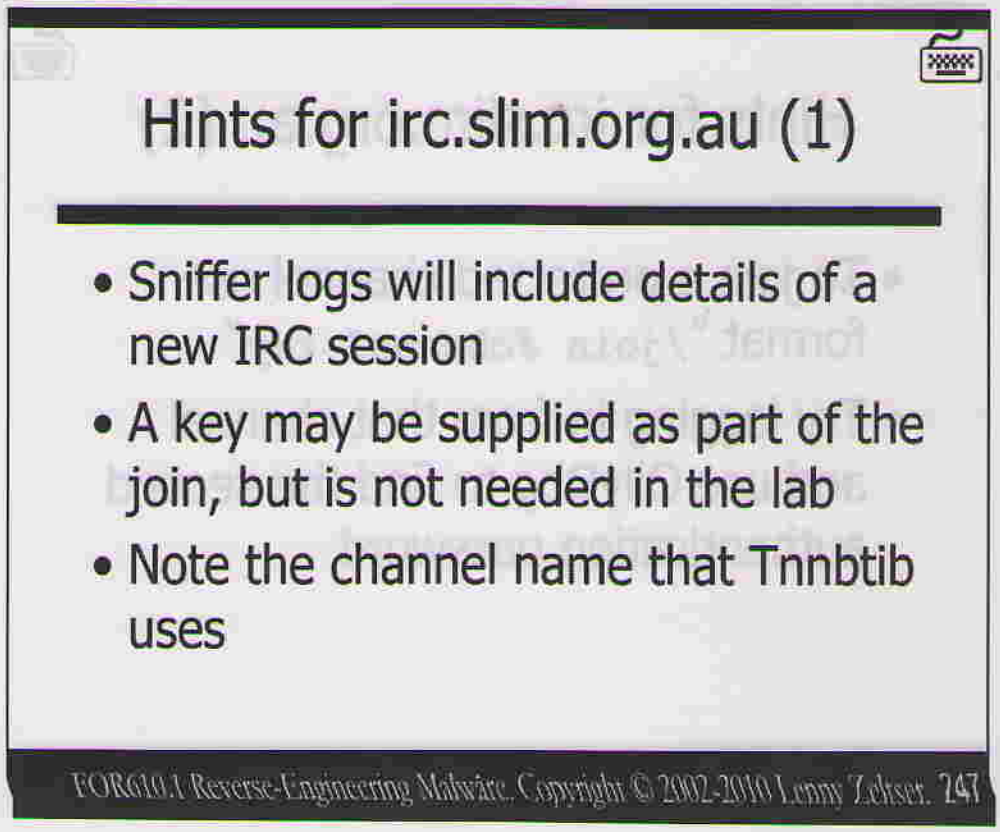


During the analysis of Tnnbtib in this section I mentioned that it tried to resolve hostname of irc.slim.org.au. Please determine the purpose of this connection attempt.

Follow the same procedures we used to determine the reasons for connecting to malwarecourse.sans.org.

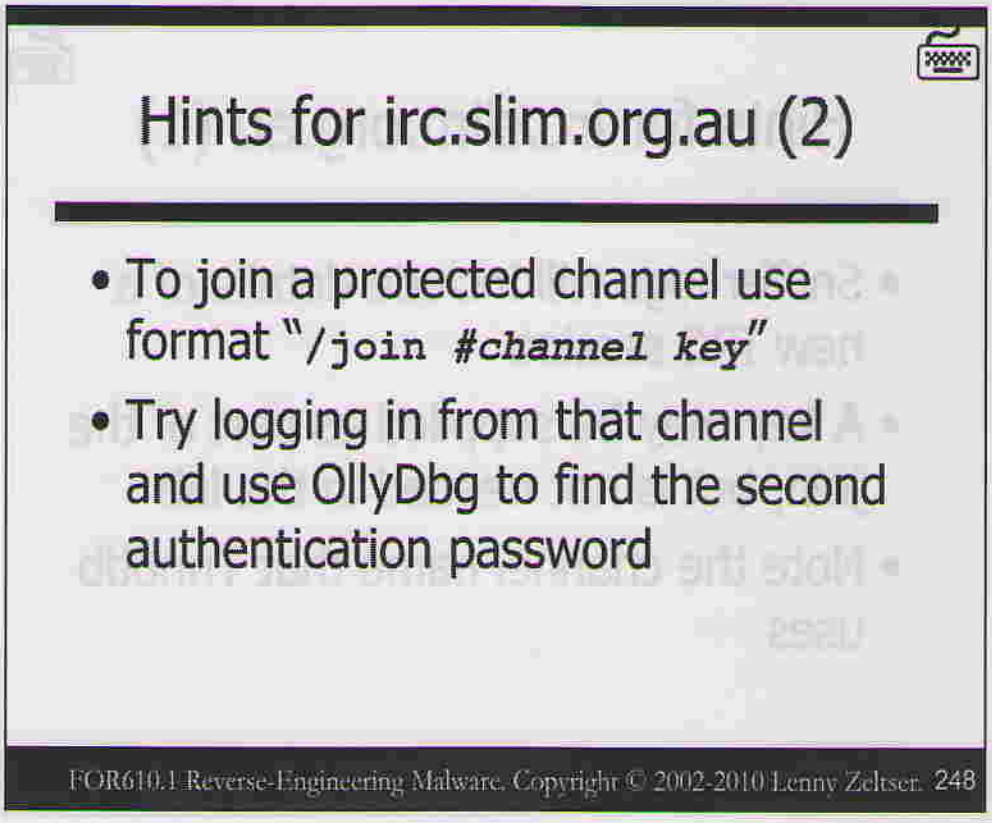
Use this exercise as an opportunity to experiment with the *fakedns* script on REMnux, if you wish. Alternatively, you can perform the exercise using the traditional methods we've been using so far, and then try it again with *fakedns* installed on REMnux. (Remember, to start *fakedns* on REMnux, type "`fakedns`".)

If you decide to use *fakedns*, I recommend verifying with *nslookup* and *ping* that domain resolution works as you expect it to work before launching Tnnbtib.



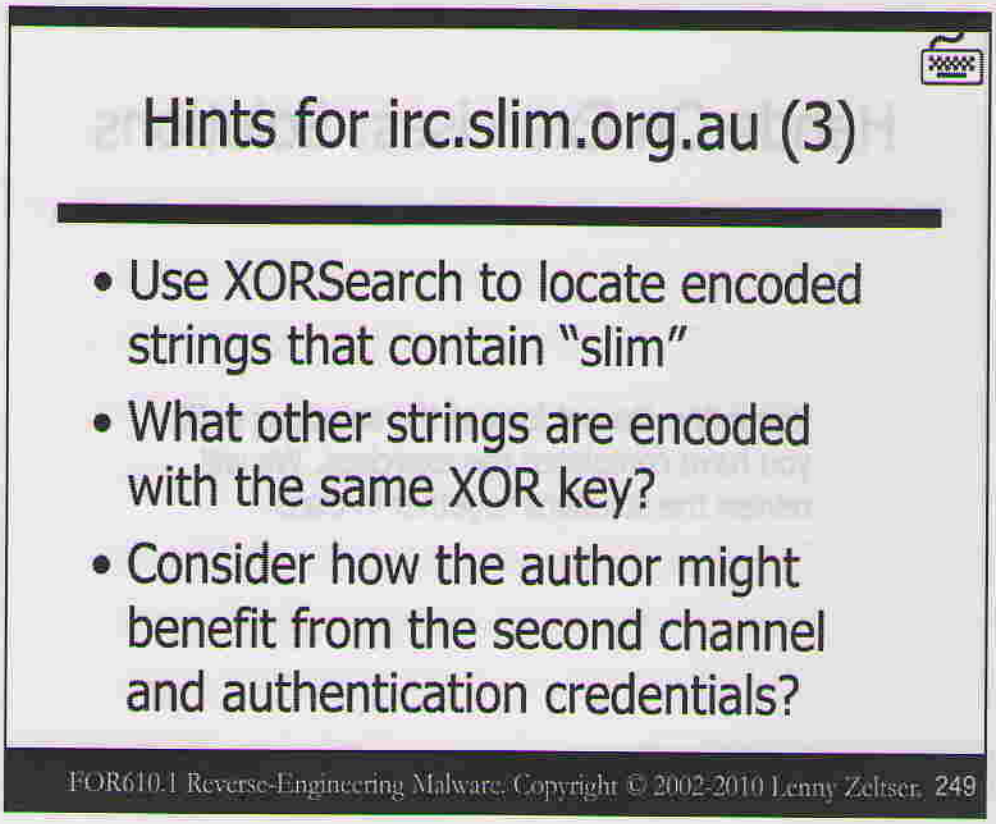
You should edit the hosts file so that the infected system points irc.slim.org.au to your REMnux virtual machine. Once the trojan can resolve this hostname, sniffer logs will reveal useful details about this connection.

You will notice an attempt to reach an unfamiliar channel. Following the channel name will be another word, which will be the key necessary to connect to this channel. IRC allows channel operators to password-protect the channel with a key, to limit who can connect to the channel. However, in your lab, the key would not be required to join a channel, because you never password-protected any channels; in this case, any password supplied as part of the join command is optional and will be ignored.



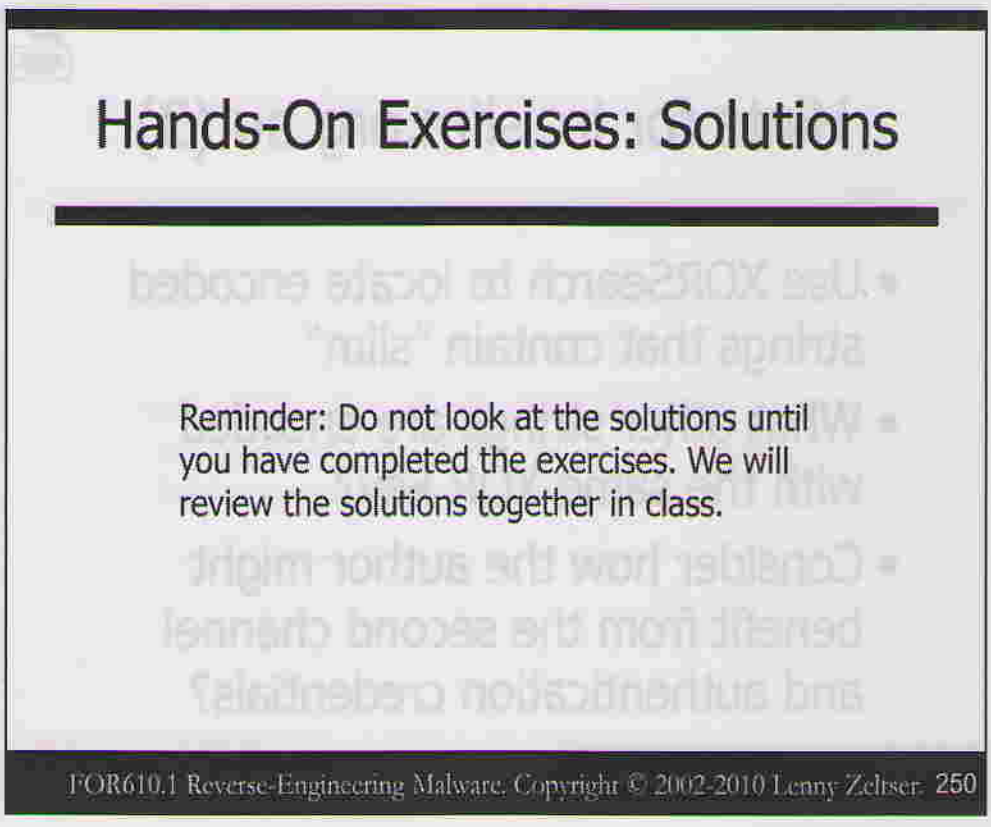
Use the "`/join`" command to connect to the desired channel. You can supply the channel key after the channel name, as shown on this slide.

Use the OllyDbg password interception technique discussed in this section to determine the second password for authenticating to Tnnbtib. To do this, follow the same procedures I demonstrated in class, but instead of logging in from the "`#malware`" channel, try authenticating from the new channel.

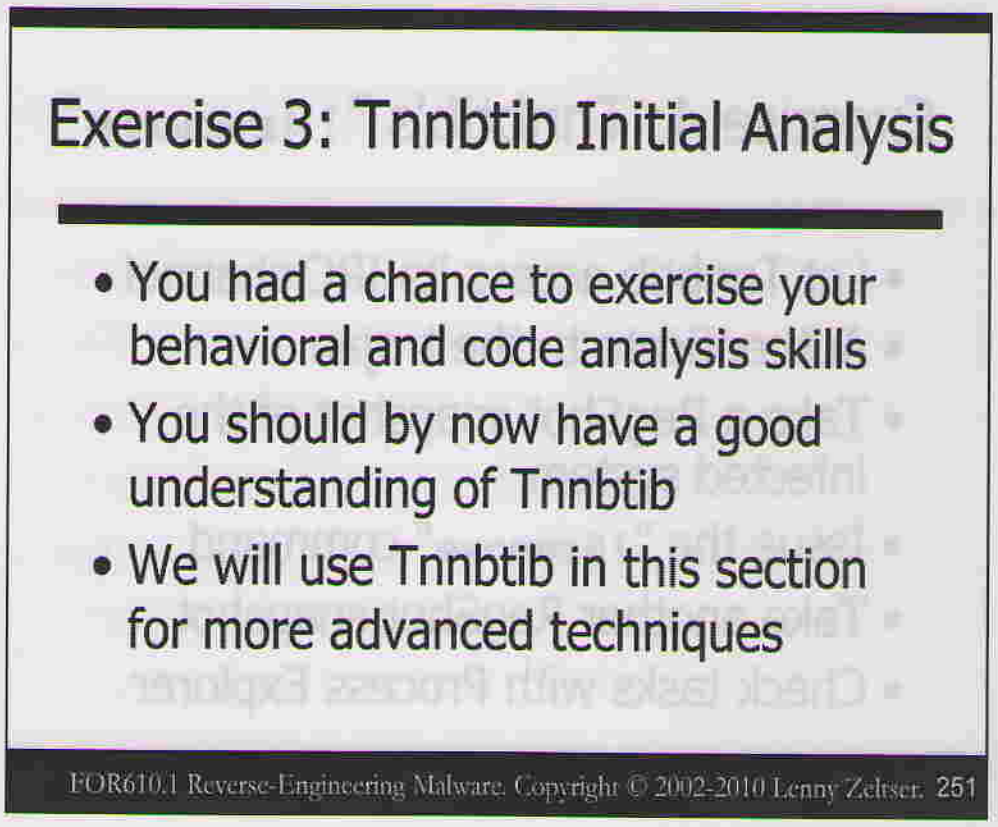


You can discover the necessary connection properties associated with irc.slim.org.au without relying on XORSearch. However, you might find it interesting to use XORSearch to locate the obfuscated strings associated with this server. Start by searching the unpacked version of tnnbtib.exe for the strings that contain the word "`slim`". You will notice that XORSearch will find two such strings: one stored without any encodings (that's the same as using XOR with the key 0); another encoded with an actual key. Use "`XORSearch -s`" and look at tnnbtib.exe.XOR.89 with BinText to locate other strings encoded with the key 89.

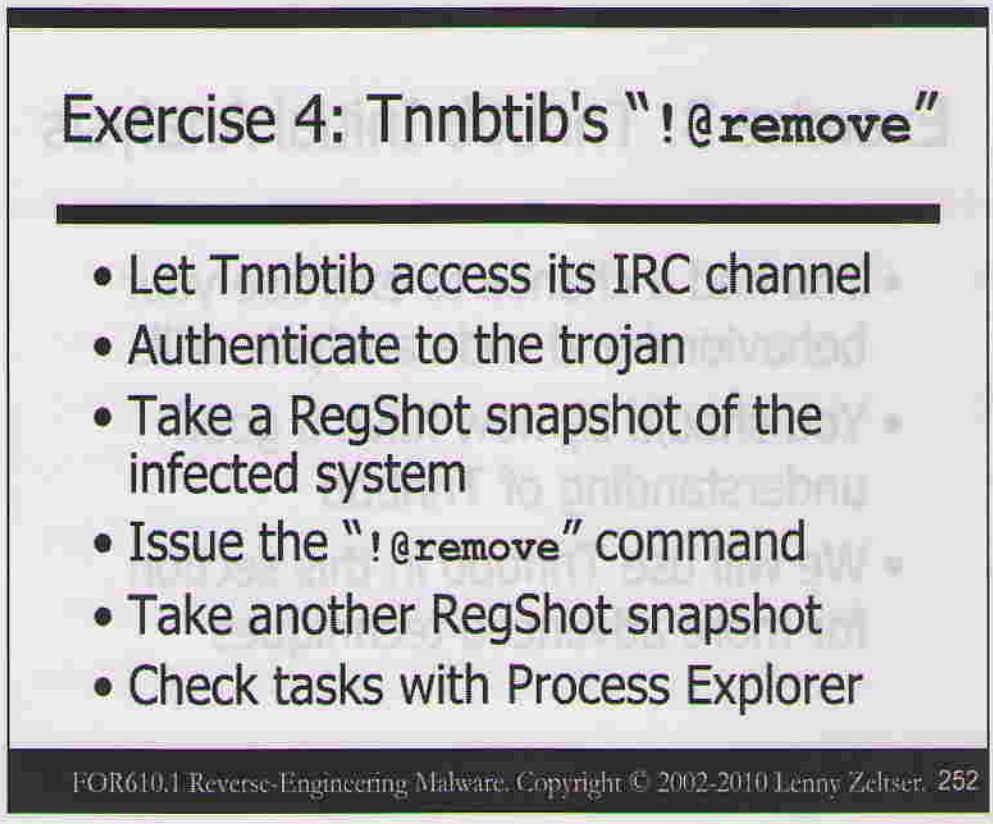
Once you discovered connection properties associated with irc.slim.org.au, consider how the trojan's author might benefit from this connection, as well as from having a second authentication password. We will discuss these considerations in the beginning of the next session.



Let's go over the solutions to this section's hands-on exercises. Reminder: Please do not look at these solutions until you have completed all exercises for this section. We will review the solutions together in class.

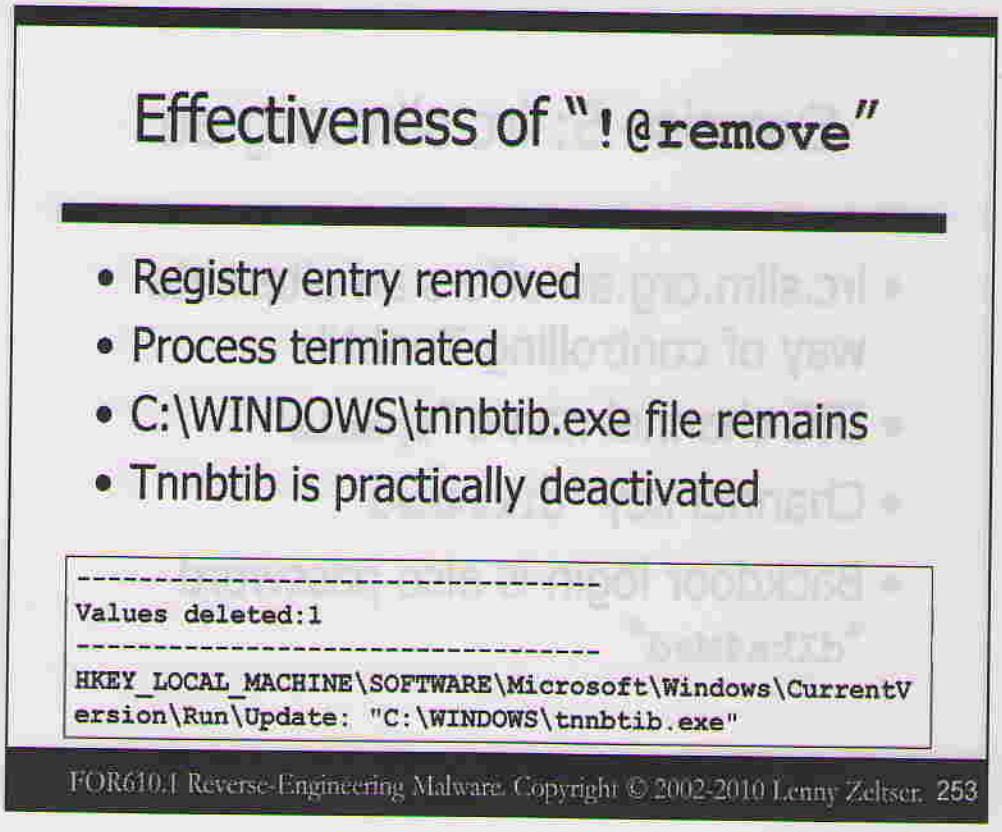


Exercise 1 asked you to perform the analysis that we went over in the previous section, giving you an opportunity to exercise your behavioral and code analysis skills. As a result, you should be very familiar with Tnnbtib by now—we will make use of your knowledge in this section, as we take an even closer look at Tnnbtib.

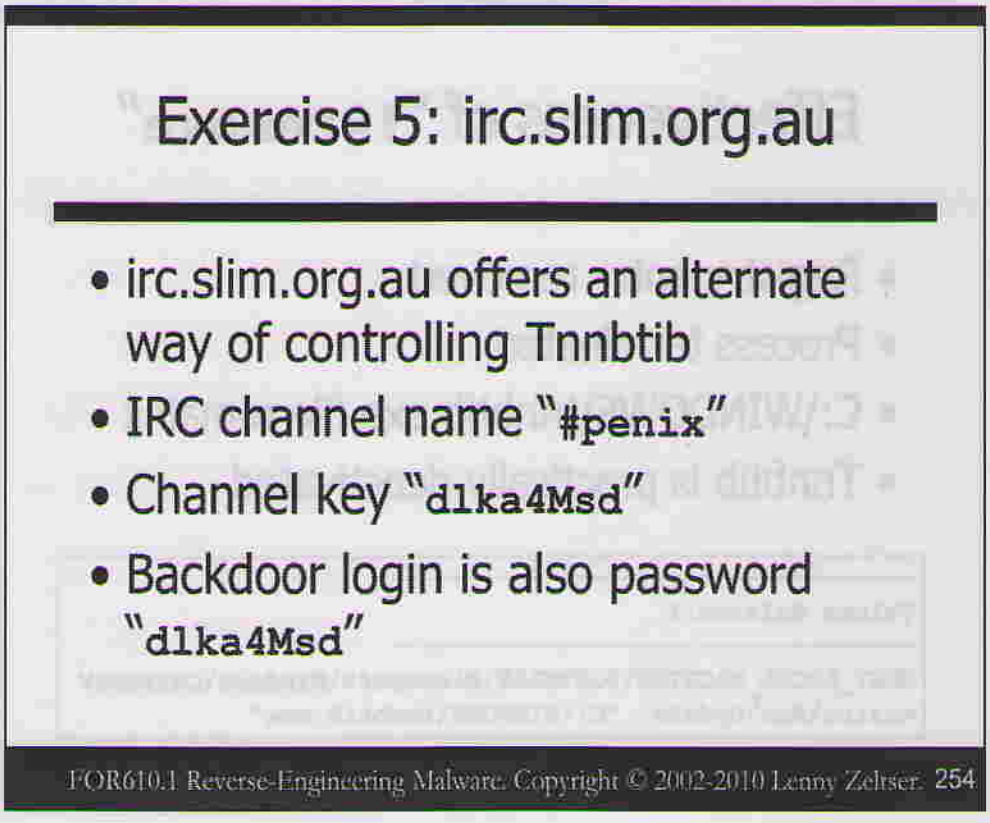


Exercise 2 asked you to assess the effectiveness of the "`!@remove`" command that Tnnbtib supports. To perform this analysis, you need to allow the tnnbtib.exe process to access its IRC channel. You then need to authenticate to the trojan's backdoor, so that you have the ability to invoke the "`!@remove`" command.

The simplest way of testing how well the "`!@remove`" command works is take a RegShot snapshot of the infected system before and after issuing this command. You also need to look at the processes running on the infected system to ensure that the tnnbtib.exe process disappears. It also makes sense to keep an eye on the infected system via Process Monitor, to verify results reported by RegShot.

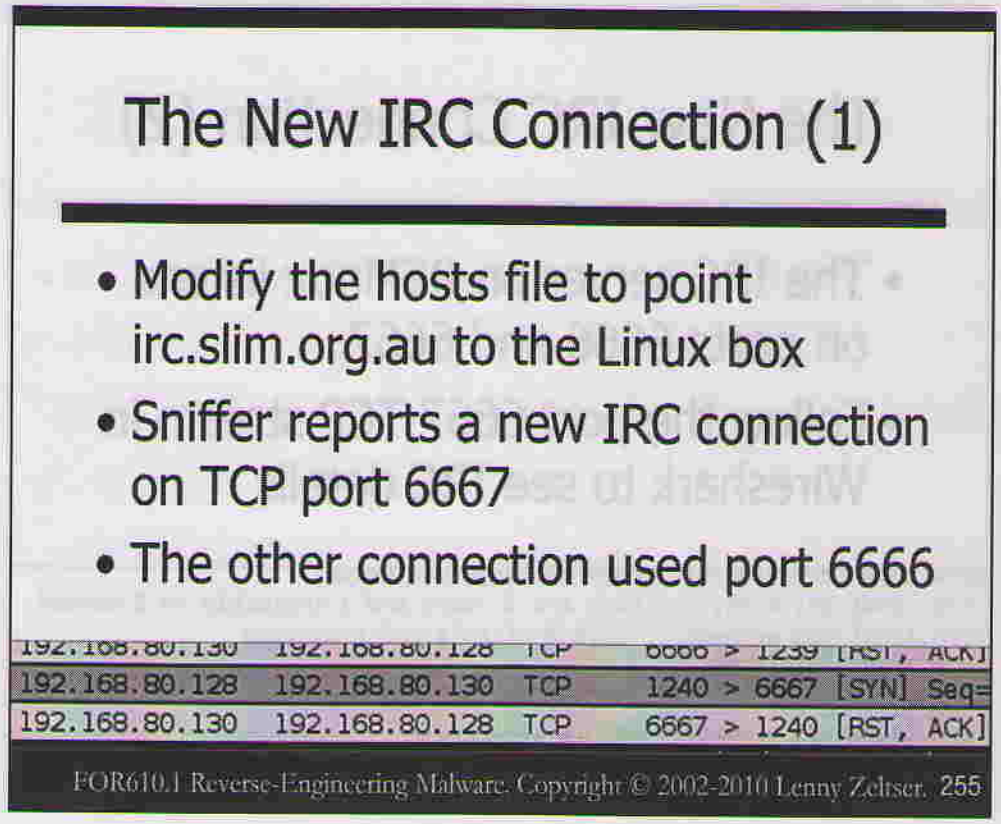


Your analysis should have revealed that the "`!@remove`" command removes the trojan's registry entry and terminates its process. For all intents and purposes, it deactivates Tnnbtib; however, it leaves the tnnbtib.exe file behind in the C:\WINDOWS directory.

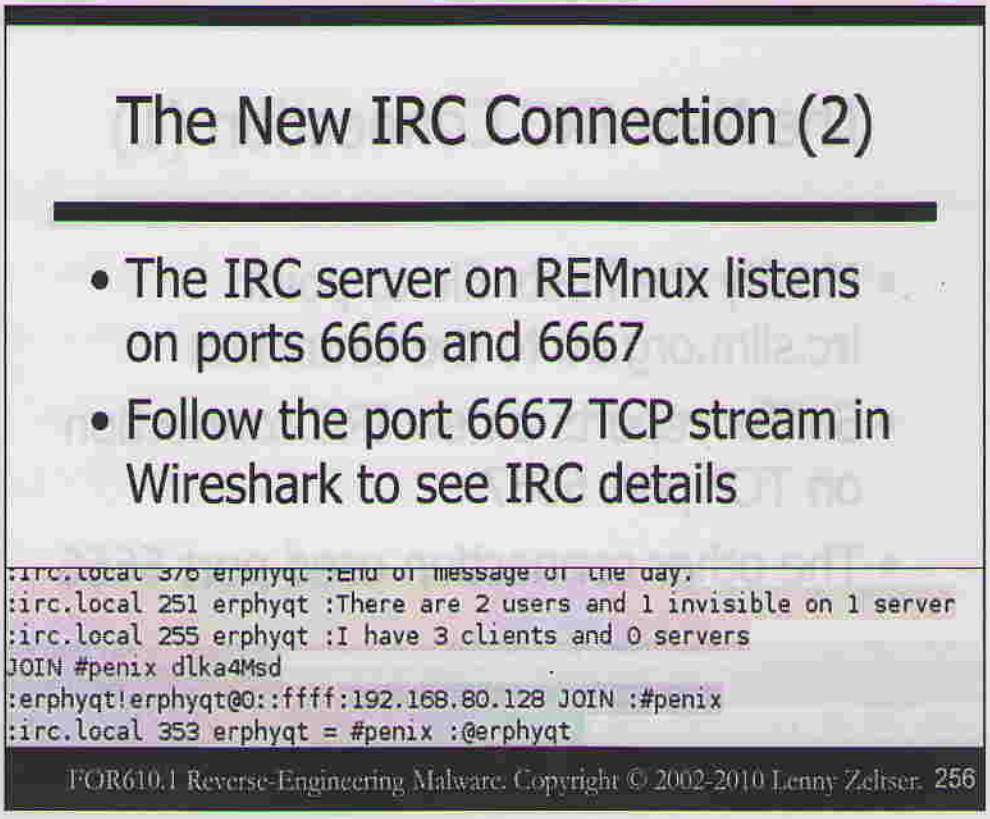


This exercise asked you to determine the purpose of the trojan trying to connect to irc.slim.org.au. Performing the analysis, which I will briefly go over in the following slides, should have revealed that Tnnbtib has a backdoor of its own! The specimen connects to this IRC server and joins channel "`#penix`" with the key "`dlk4Msd`". If you are communicating with the trojan through this channel, you can use the password "`dlk4Msd`" to login to its backdoor.

The following slides demonstrate the key steps in performing the analysis that would tell you about irc.slim.org.au and related functionality.

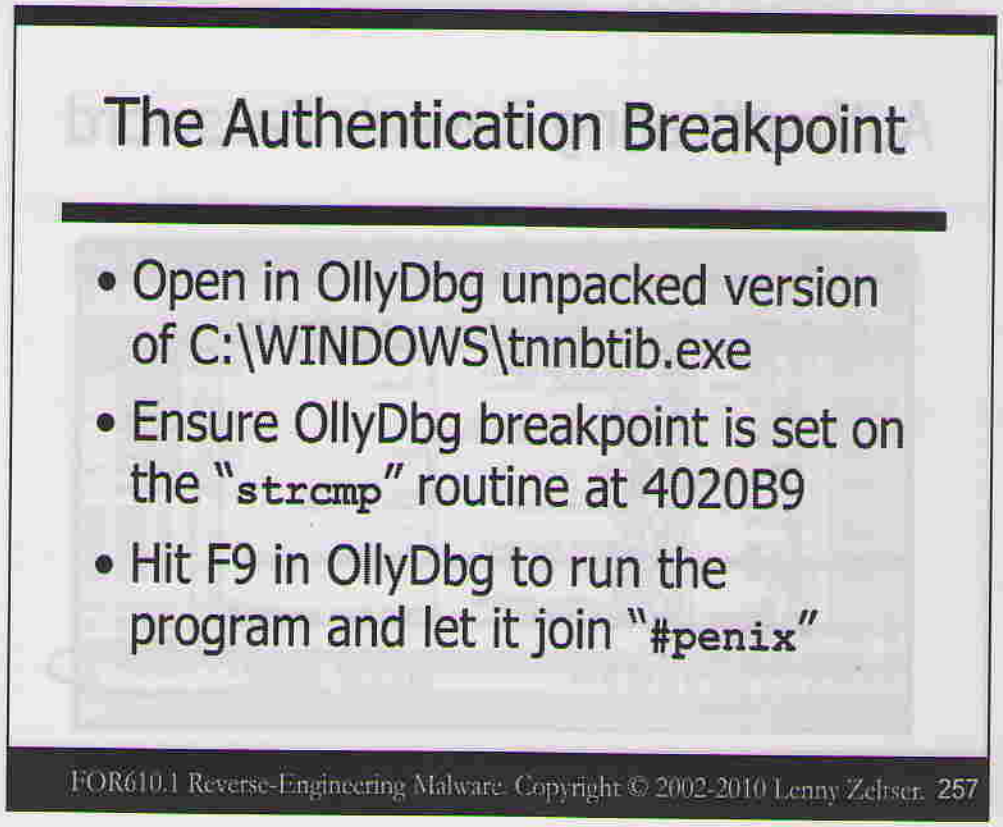


Having noticed the specimen's attempts to resolve the IP address of irc.slim.org.au, you would modify the hosts file on the infected machine to point this hostname to the IP address of the Linux box in your laboratory. You would then use a sniffer to detect that Tnnbtib is trying to connect to TCP port 6667 on that server. (Note that when connecting to malwarecourse.sans.org, the trojan used TCP port 6666 instead.)



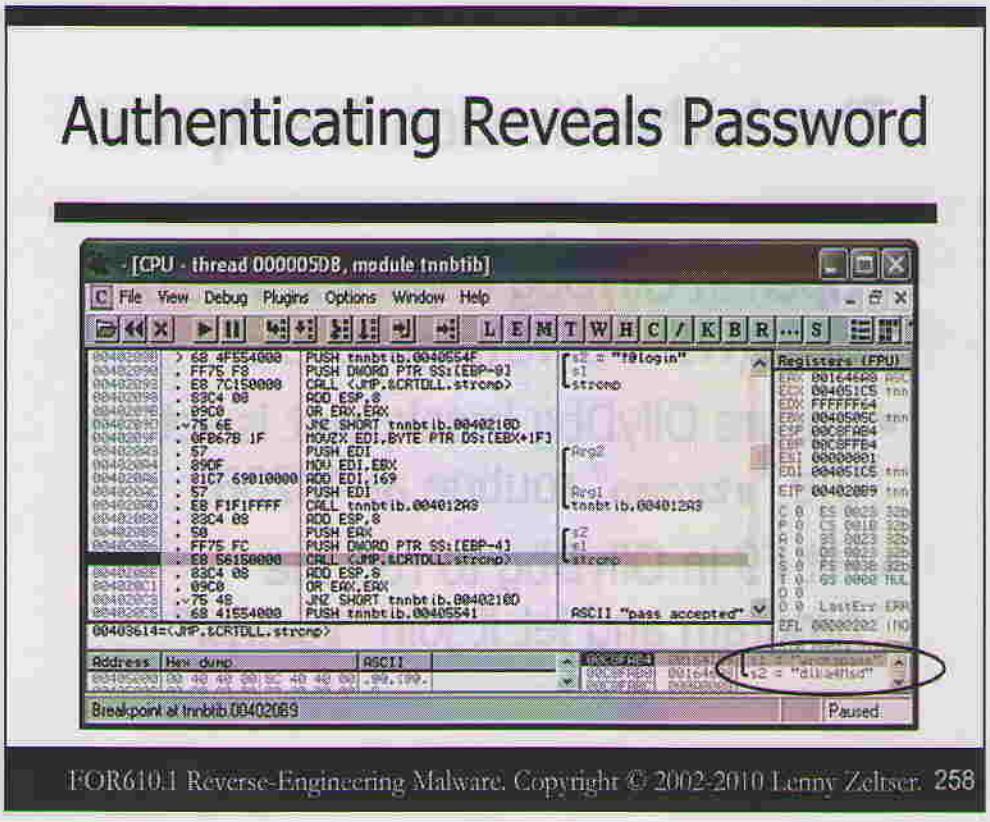
The IRC server on REMnux listens on TCP port 6666 and 6667, so the trojan would be able to connect to the IRC server as soon as you modified the hosts file on the infected system.

To see the relevant IRC connection details, locate a packet in the Wireshark capture that uses TCP port 6667, right-click on it, and select "Follow TCP Stream". You can see the nickname that Tnnbtib uses after connecting to irc.slim.org.au. Run this experiment a few times to see whether it is randomly generated or hard-coded. You can also see channel connection details: it joins "`#penix`" with the key "`dlk4Msd`".

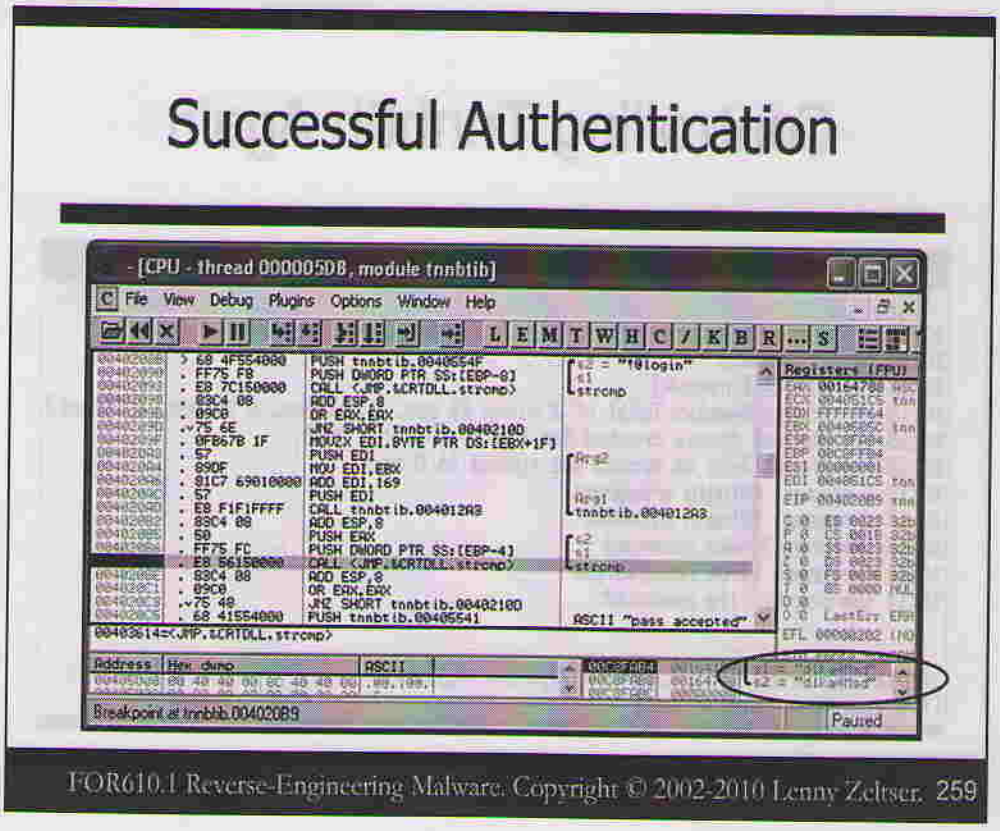


The process for intercepting this backdoor password is very similar to the one we discussed in the previous section. You need to open the unpacked version of tnnbtib.exe in OllyDbg, and set the breakpoint on the same "`strcmp`" routine as last time; it is at offset 4020B9. If you notice that OllyDbg already set a breakpoint at that offset for you, don't be surprised—OllyDbg remembers the breakpoints between sessions, and must have remembered the breakpoint you set when examining Tnnbtib last time.

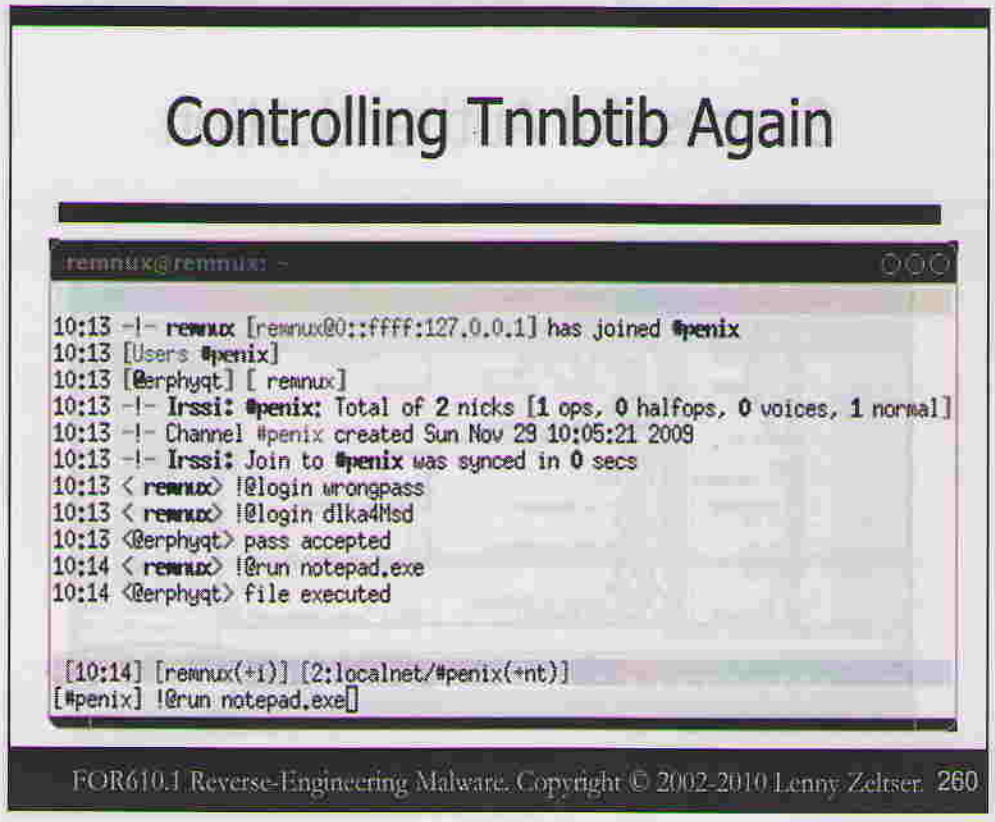
Hit the F9 key in OllyDbg to run the trojan, and let it join the "`#penix`" channel. You should access the same channel using your IRC client, and attempt authenticating to the trojan using some fake password via the "`!@login`" command.



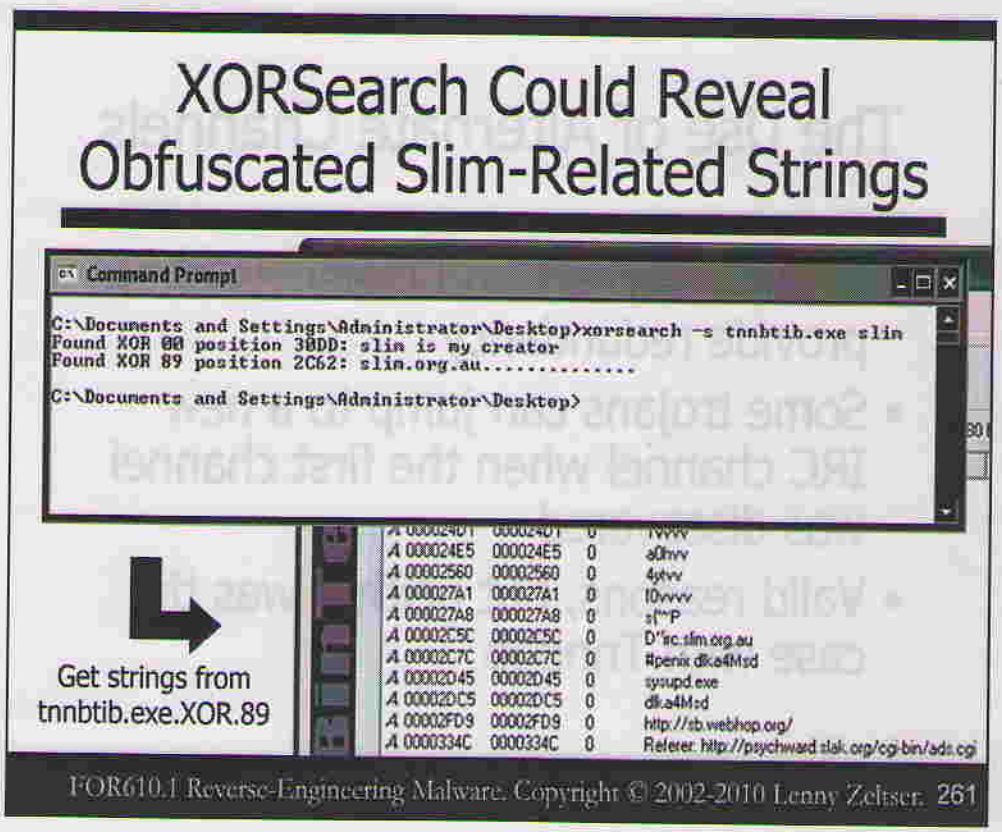
Once OllyDbg pauses the execution of Tnnbtib at the breakpoint, you can glance at the bottom right corner of the screen to see the parameters being passed to the "`strcmp`" routine. Notice that one of the parameters is the fake password that you supplied when trying to authenticate; the other parameter is the real password for this backdoor: it's "`dlka4Msd`".



Once you know the real password for this backdoor, you can log in. Notice that OllyDbg reaches the breakpoint once again, pausing the execution of Tnnbtib. In this case, the two passwords supplied to the "`strcmp`" routine are equal, and the authentication attempt will succeed.



Once you authenticate to the trojan, you can execute privileged commands. In this example, I was able to launch Notepad on the infected system via the "`!@run`" command.



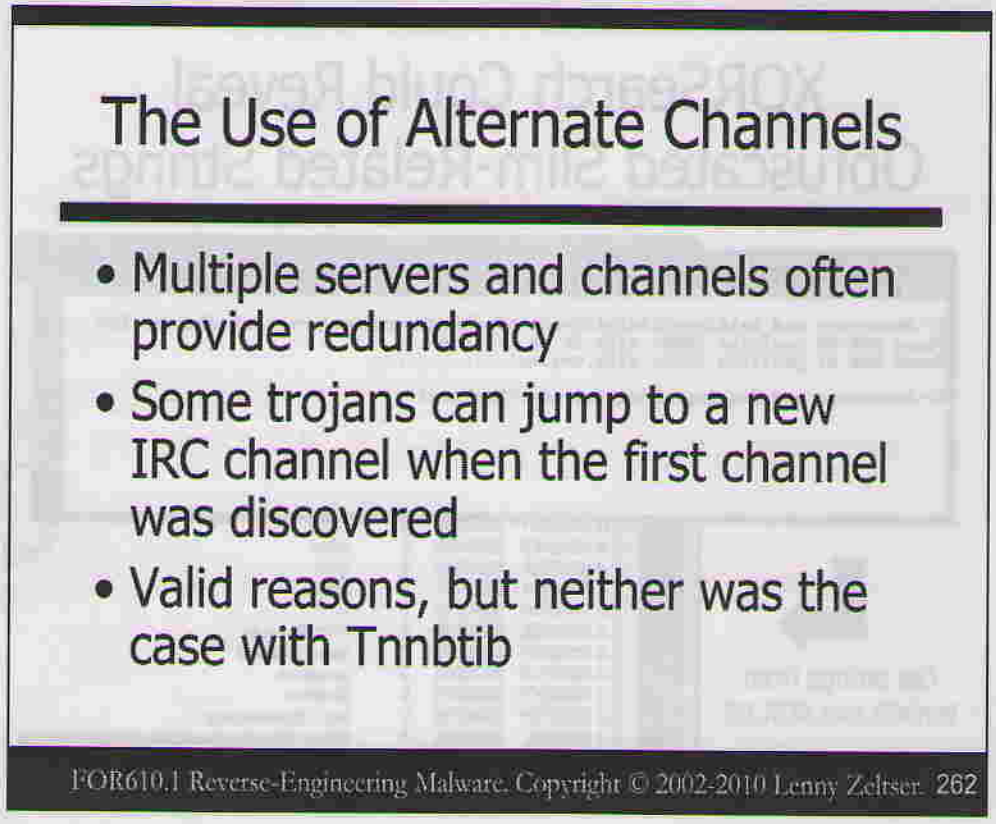
You may have found it useful to use XORSearch to locate strings associated with the hostname "`irc.slim.org.au`". I would search for a short substring, such as "`slim`", to see whether XORSearch would find anything interesting.

Indeed, XORSearch found two strings in the unpacked version of tnnbtib.exe that included "`slim`". There is "`slim is my creator`". You can see that it's encoded with the XOR key "`00`", which means it's not encoded at all, but is stored in clear text. There's also "`slim.org.au`", which is obfuscated with the XOR key "`89`".

You can invoke XORSearch with the "`-s`" parameter to deobfuscate the strings encoded with these keys.

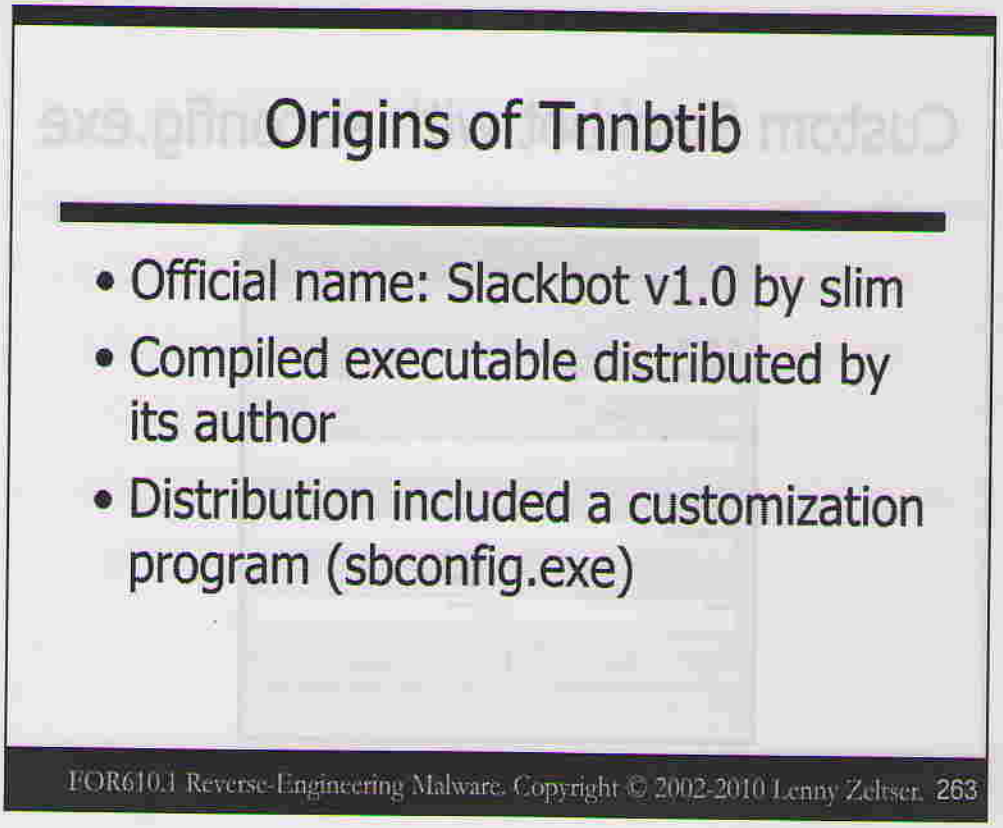
In this case, XORSearch would create two files: tnnbtib.exe.XOR.00 and tnnbtib.exe.XOR.89, for each corresponding key. The file tnnbtib.exe.XOR.00 would be a copy of tnnbtib.exe, since the key "00" does not change any characters when used with the XOR algorithm.

On the other hand, tnnbtib.XOR.89 would contain other interesting strings encoded with the key "89", such as "`#penix dlk4Msd`", the URLs we encountered earlier, and the filename "`sysupd.exe`", which we actually have not seen before.

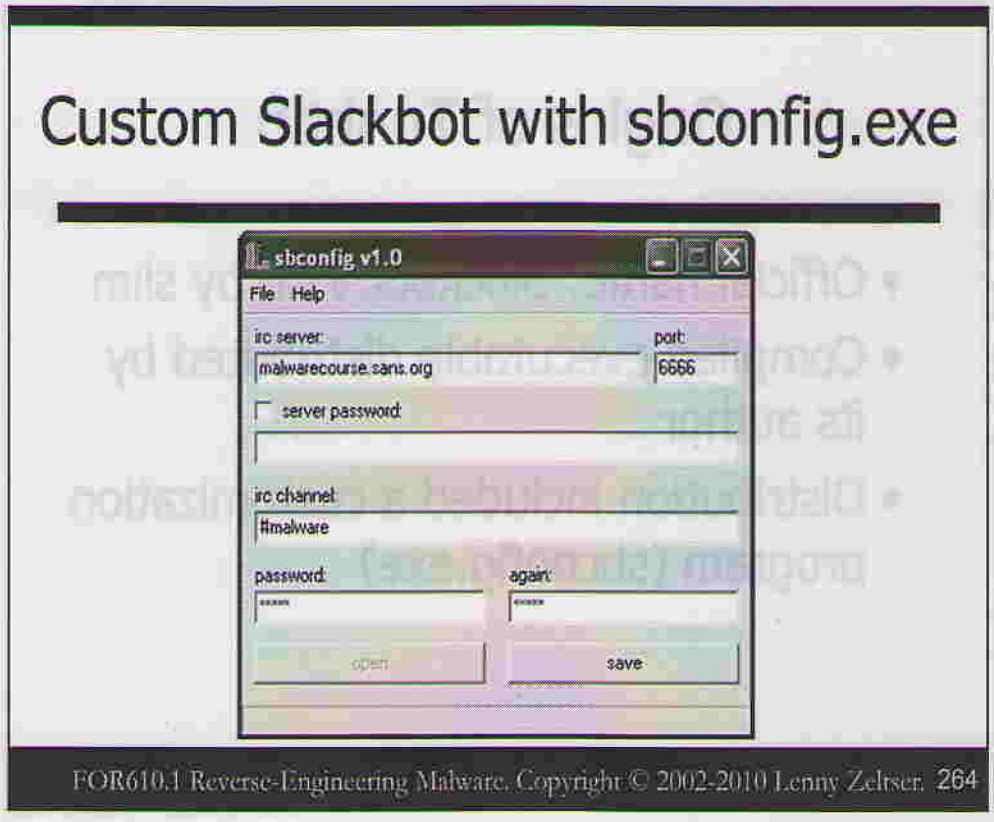


As you can see, the Tnnbtib trojan uses not only multiple IRC servers, but also multiple IRC channels. This is not uncommon. As we already discussed, making the specimen aware of multiple IRC servers allows the author to add a measure of redundancy to the communication mechanism, in case some of the servers become inaccessible.

Using multiple IRC channels can provide redundancy in a similar manner. In particular, if the author detects that someone discovered his or her initial communication channel, the author can command infected systems to join another channel instead. However, this is not the reason for Tnnbtib attempting to join the "`#penix`" channel on "`irc.slim.org.au`"...



One of the most interesting aspects of Tnnbtib is that it contains an undocumented backdoor. In fact, the program you've been analyzing is officially called Slackbot v1.0 by "slim". It is dated April 18, 2000, and for a long time was available as a free download (without source code) from the author's website. In addition to distributing the compiled version of Slackbot, the author also provided a customization program called sbconfig.exe, which we discuss on the next slide.



This slide shows a screenshot of sbconfig.exe, which can be used to customize the compiled version of Slackbot. As you can see, the person who downloaded Slackbot can specify his own IRC server and channel details, along with the login password. The sbconfig.exe program requires the user to load the compiled Slackbot executable, and then modifies it when the user presses the "save" button.

The next slide contains usage information that Slackbot's author supplied with the Slackbot download. Most commands should be familiar to you already.

Slackbot Usage Information

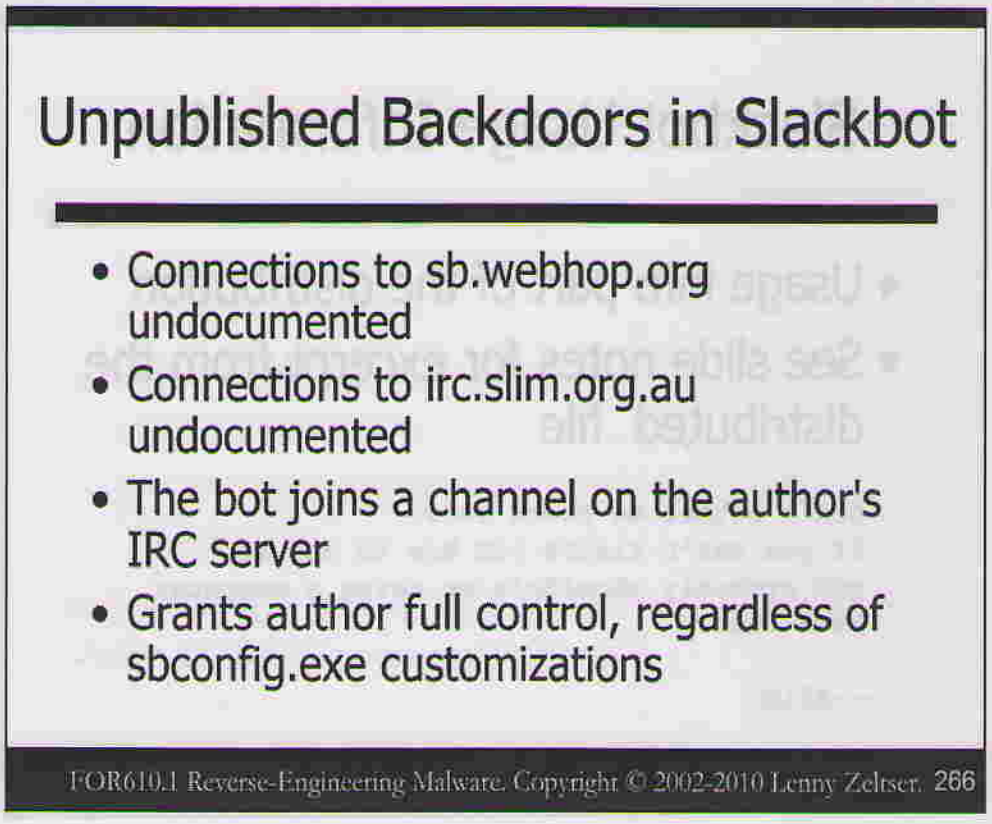
- Usage info part of the distribution
- See slide notes for excerpt from the distributed file

```
don't bother me about this,
if you can't figure out how to use it,
you probably shouldn't be using a computer.
have fun.

--slim
```



```
!@id                          :identifies the bot version
!@sysinfo                     :prints system info
!@login <password>            :sets you as the master
!@raw <whatever>              :raw irc commands
!@say <dest> <msg>            :says stuff
!@join <channel> <key>        :joins a channel
!@part <channel> <message>    :parts a channel
!@nick <nick>                 :changes the nick
!@rndnick                     :changes the nick to something random
!@cycle <number_of_times_to_cycle> <irc_channel>
                              :cycles a channel
!@clone <number_of_clones> <irc_server> <irc_port> <irc_channel>
                              :sends another bot to somewhere
!@quit <msg>                  :quits the irc server and does not
come back
!@udp <host> <number_of_packets> <delay_between_packets>
<packet_size> :sends udp packets to a host
!@visit <number_of_times_to_visit> <url>    :visits a url
!@webdl <url> <path_to_save_to>
                              :downloads a file from a webserver
```

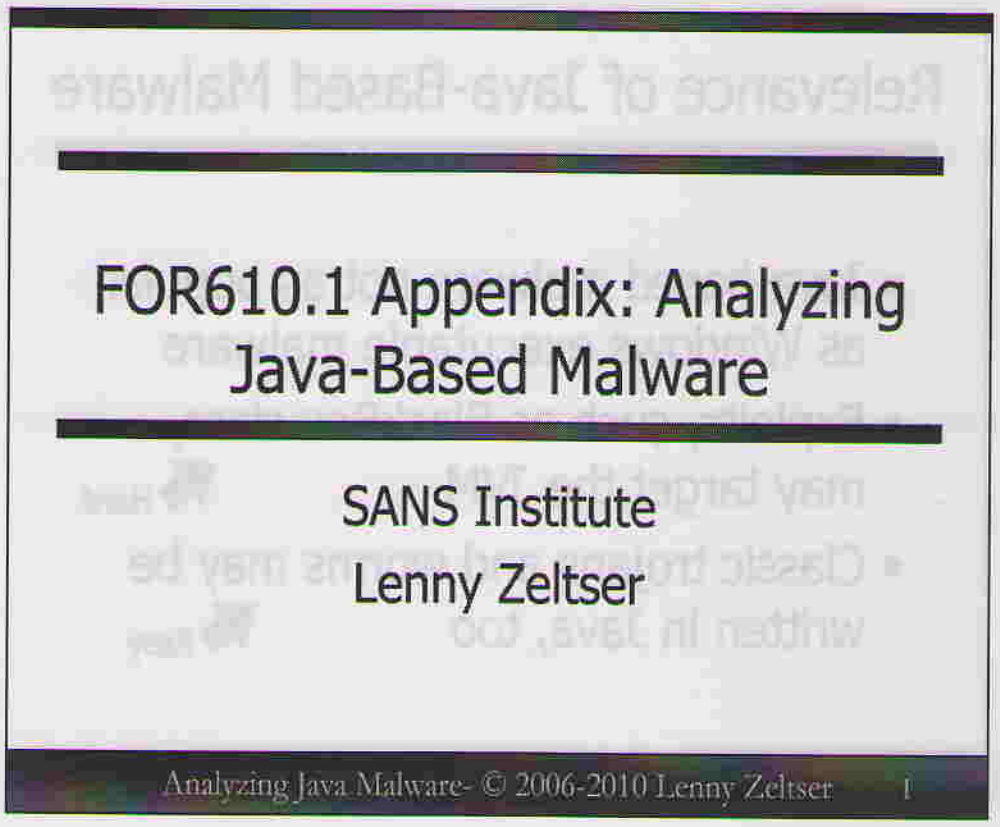


As you've learned for yourself, Slackbot contains an undocumented backdoor, with an alternate password hard-coded into the program, allowing the author to take over all Slackbot instances connected to the Internet. Those who download its compiled version and customize it with sbconfig.exe also do not know that the bot makes an HTTP connection to sb.webhop.org, as we discovered earlier.

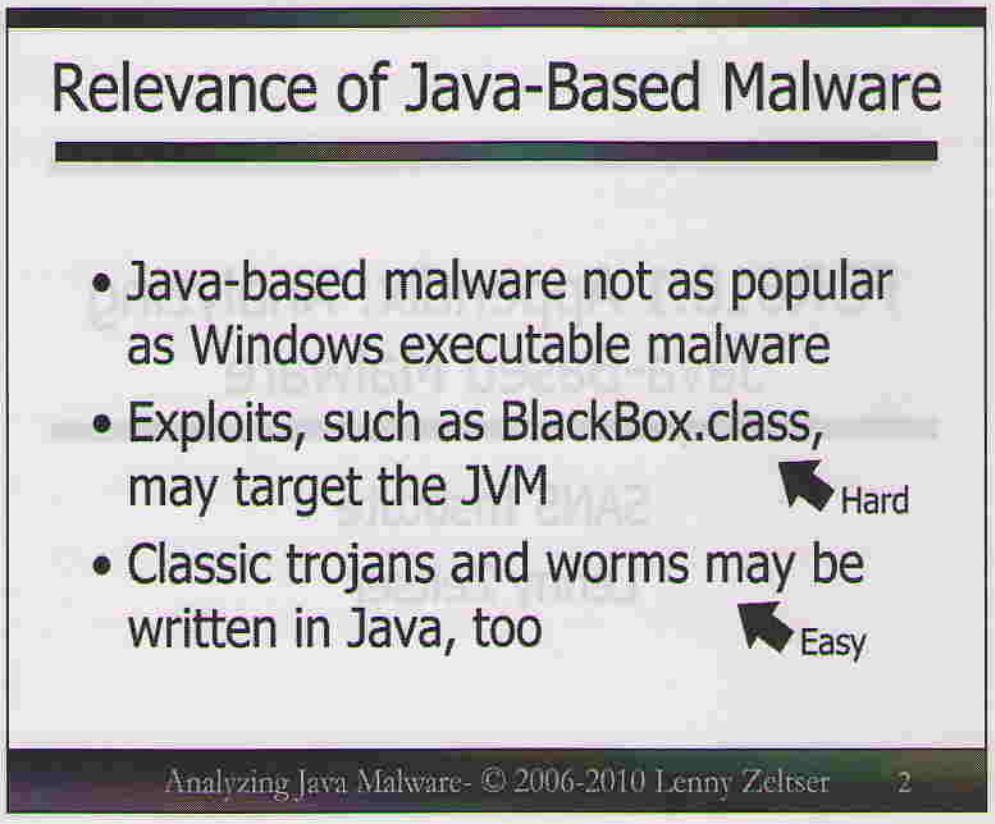
End of FOR610.1

- You have now completed FOR610.1.

Congratulations! You have reached the end of FOR610.1.



This appendix to 610.1 discusses approaches to analyzing malicious software written in Java.

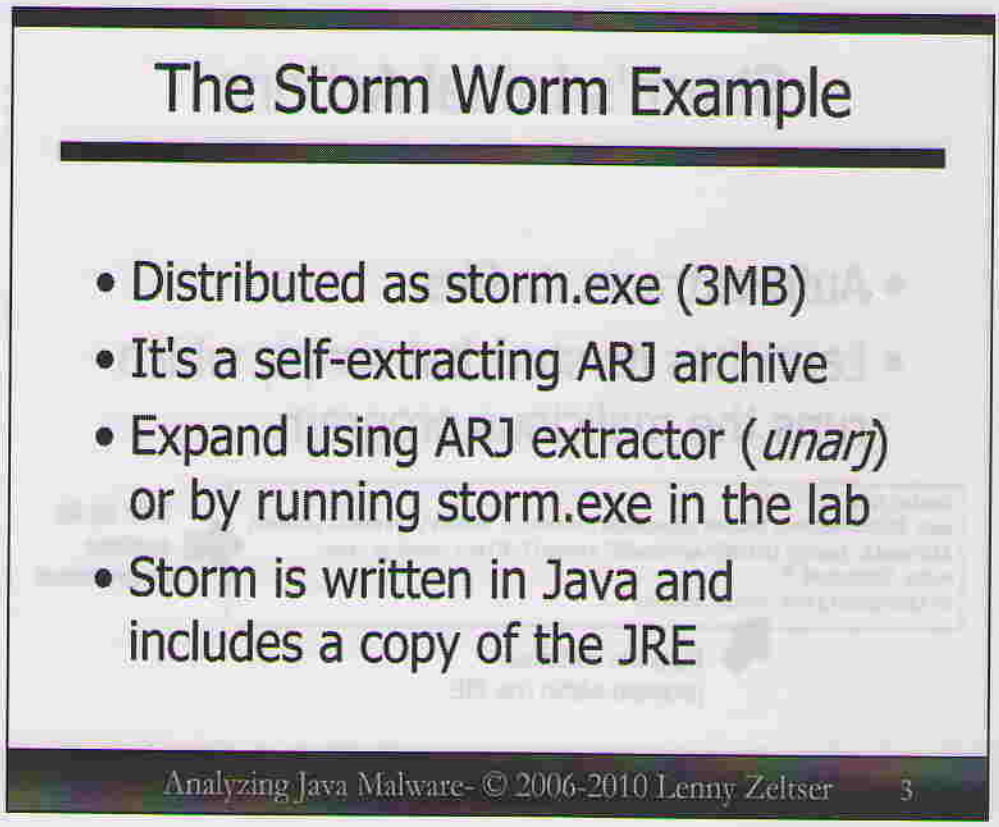


It is less common to see malicious code that was written in Java, in comparison to code that runs natively on x86 systems. One of the reasons for this is, perhaps, the dependence that Java-based code has on the presence of a Java Virtual Machine (JVM) that is part of the Java Runtime Environment (JRE) on the victim's system. Code written in Java cannot natively run on Windows or Linux, but has to execute with a JVM. However, Java is a popular programming language, and you do come across Java-based malware periodically.

I've seen two types of Java malicious code. One category includes Java-based exploits, such as BlackBox.class that we saw in the previous slides. Such exploits typically target a vulnerability within a JVM, usually with the goal of executing arbitrary code on the victim's machine. Writing such exploits is relatively difficult, because they tend to take advantage of advanced features in the implementation of a JVM, and may incorporate low-level Java bytecode instructions. For the same reasons, analyzing such exploits is relatively hard.

Another category includes specimens that implement classic trojans as worms, such as those that we've seen in earlier section of this course. It just so happens that the malicious program in this case was written in Java. Analyzing such specimens is relatively easy; in fact, it is significantly easier than analyzing compiled native executables. The reason for this that, unlike with compiled native executables, it is easy to decompile Java programs into the form that looks like their original source code.

Let's take a look at an example, to see how we would analyze one such malware specimen.

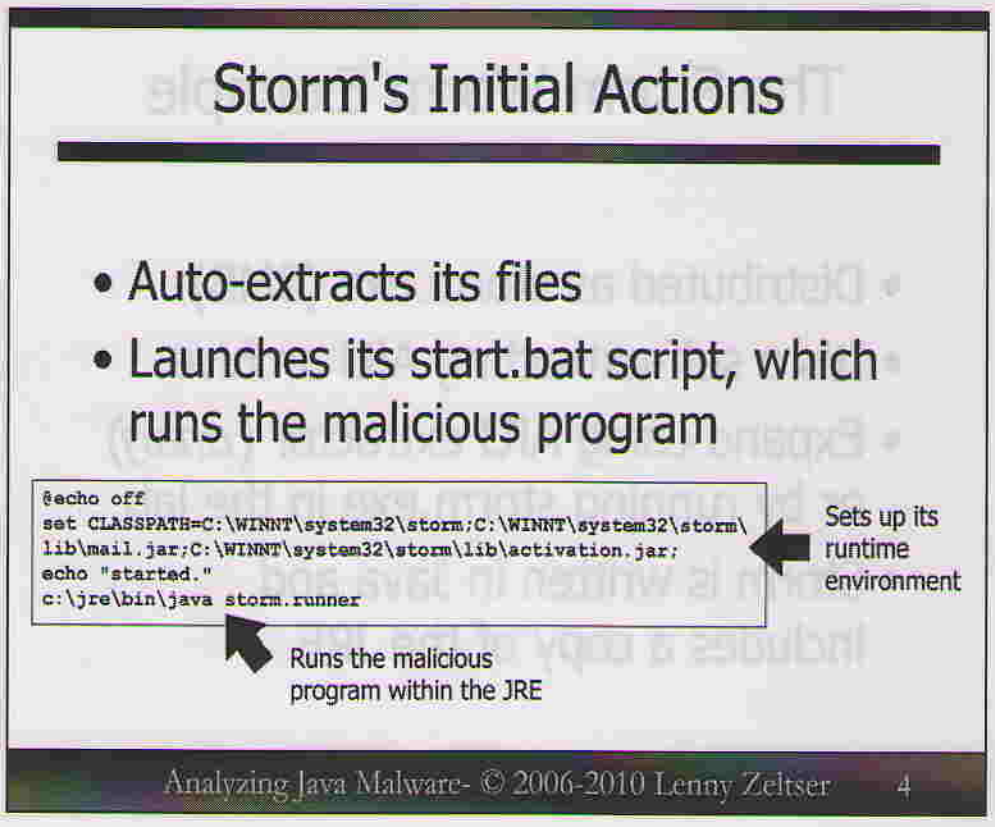


I recall being surprised when I first encountered this specimen due to its large size: it weighed approximately 3MB! That's a lot for most malicious programs, since malware authors value the advantages of having a lightweight file that is fast to transport over the network.

This file was named storm.exe. The file turned out to be a self-extracting ARJ archive that was packed with UPX. ARJ is just a file compression format, similar to ZIP; you can extract files from an ARJ archive using a free utility called unarj, which is available at http://www.arjsoft.com.

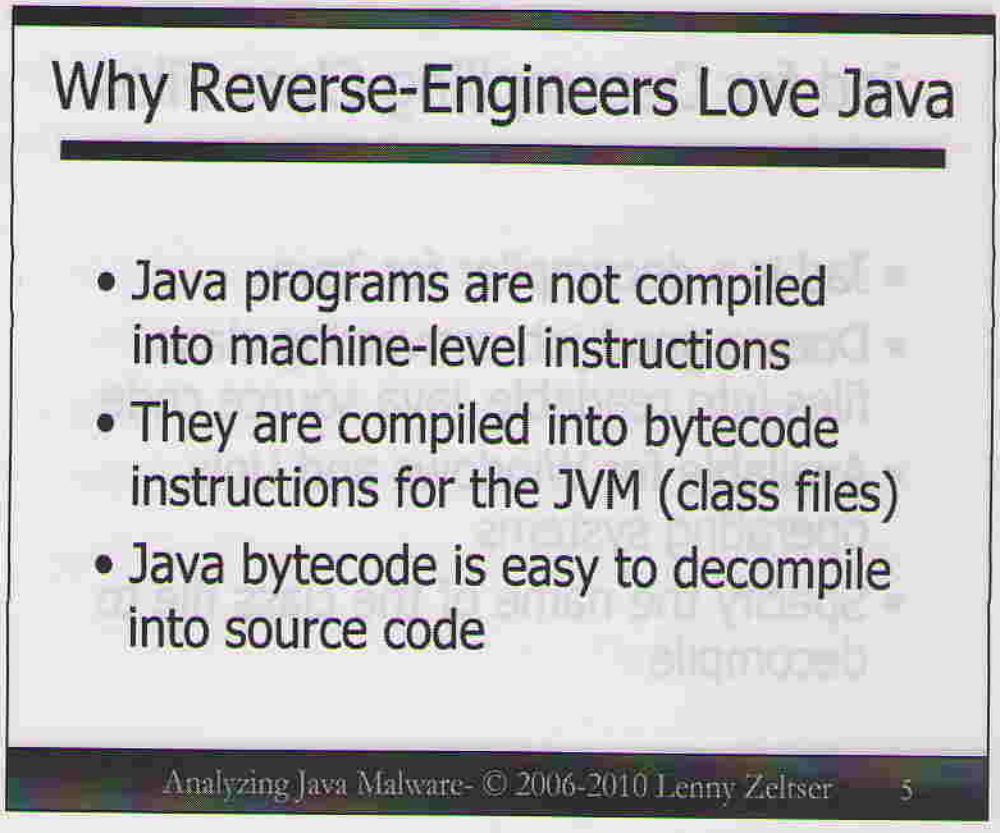
I chose to extract storm.exe using unarj, instead of running it to let it auto-extract itself. I felt that this gave me more control over the process. After extracting the file's contents I learned the reason for its hefty size: it was written in Java and came with its own copy of Java Runtime Environment (JRE). Apparently, the author did not wish to rely on the victim having the JRE installed, and decided to ship one along with the malicious program.

For the background about this specimen, please see http://www.symantec.com/security_response/writeup.jsp?docid=2001-060615-1534-99. This is the "original" Storm worm, and predates the more popular bot/worm that carried the same name and captured headlines in 2007 and 2008.

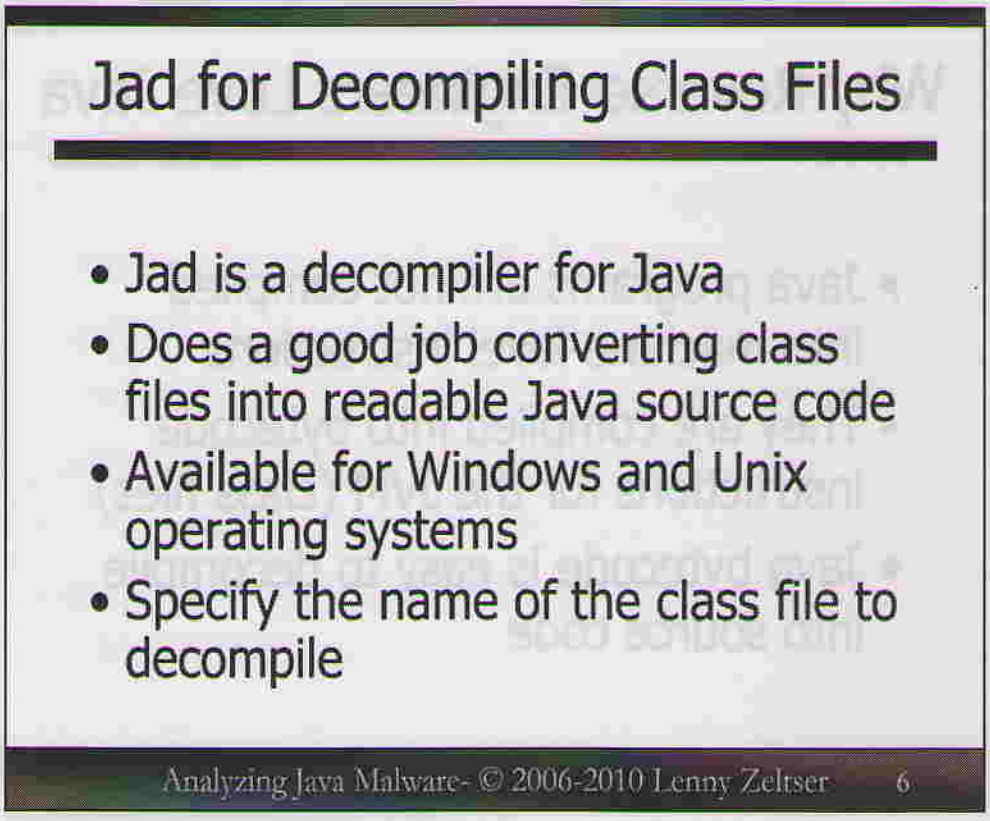


The self-extracting archive was configured to launch its start.bat script after depositing its files, along with the JRE, on the victim's file system. You can see the contents of start.bat at the bottom of this slide. The line that begins with "`set`" sets-up the environment for running the malicious program; the CLASSPATH variable tells the JRE where it can find the necessary files.

The malicious program is started at the last line of start.bat, where the JVM is invoked as "`java`" and is told to run the "`storm.runner`" class. Now that we know which Java class to start our analysis from, let's look at why we should be pleased that this is a Java-based malicious program, and not a native Windows executable.

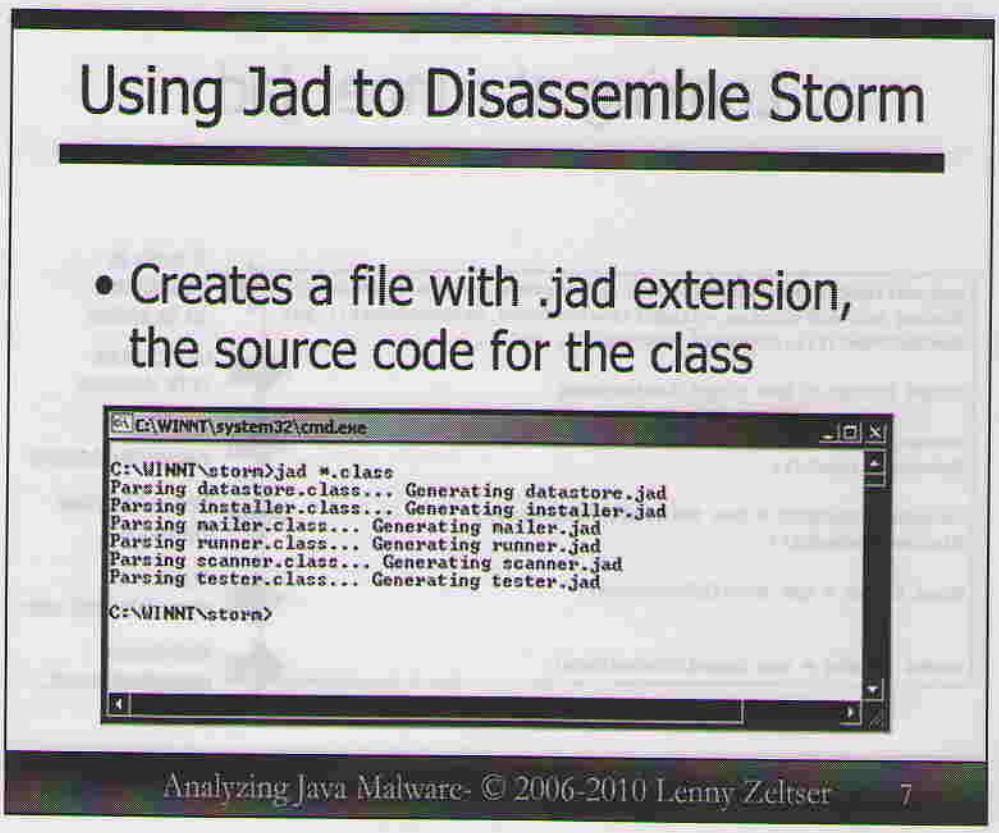


Unlike native Windows executables, Java programs are not compiled into machine-level assembly instructions. Instead, code written in Java is compiled into bytecode instructions that the JVM can execute. Files that store such bytecode instructions are called Java class files, and have to execute within a JVM. Fortunately for us, Java bytecode is easy to decompile into original source code, especially with the help of a tool that we cover in the next slide.

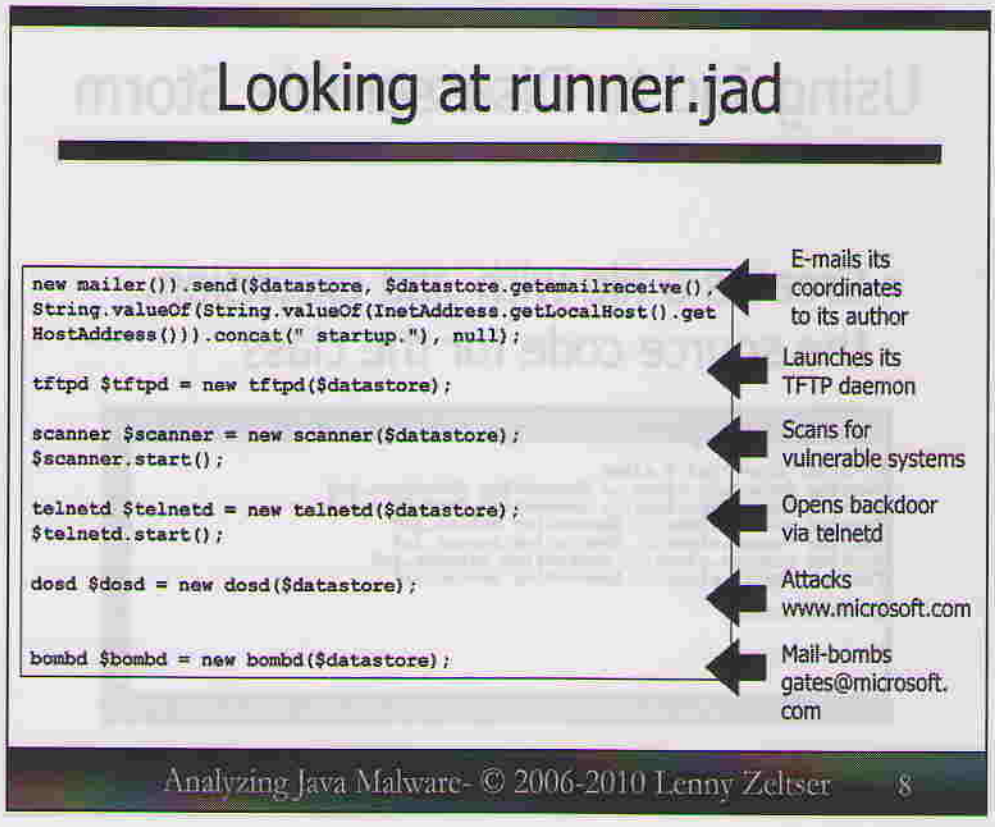


Jad is my favorite tool for decompiling Java class files into source code. It runs on Windows and Unix, and it is free for non-commercial use. You can download Jad from http://www.kpdus.com/jad.html , or use the copy I included on the DVD that you received for this course. Jad is also installed on REMnux. (Update: The website that used to host Jad seems to be no longer available.)

Jad is a command-line utility. To use it, simply supply the name of the Java class files you wish to disassemble. You can see Jad in action on the following slide.



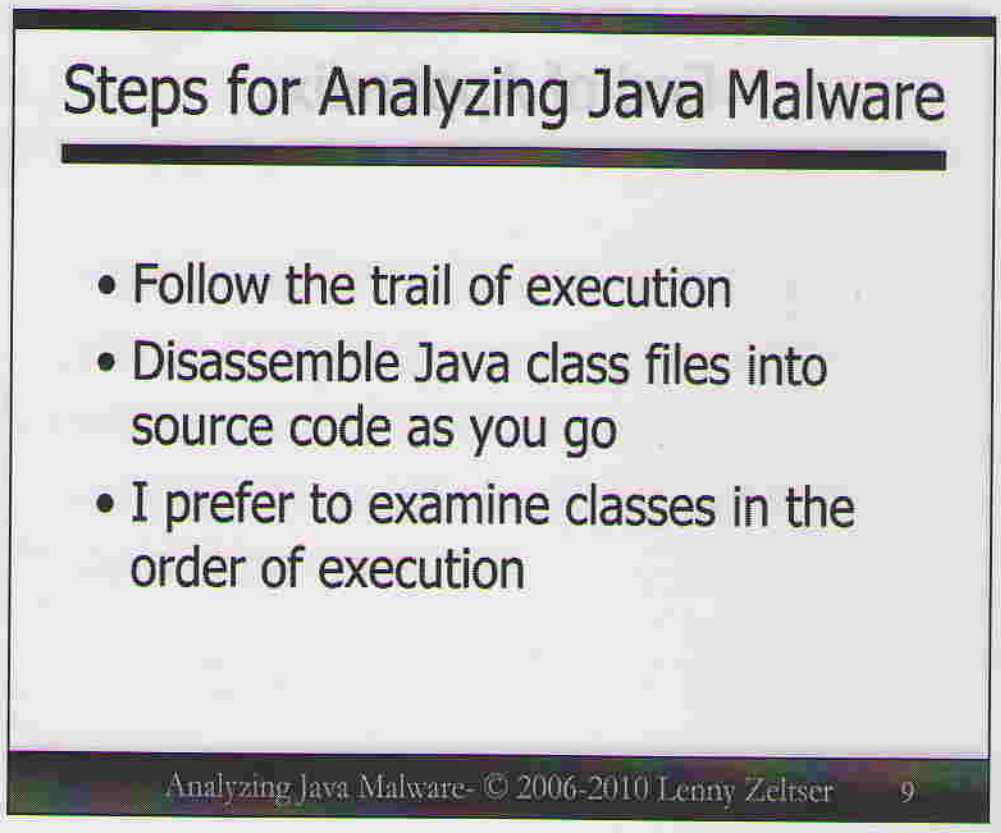
The Storm specimen that we're analyzing in this example was comprised of several class files. To disassemble multiple Storm files in one shot, I invoked Jad using the "`jad *.class`" command. As you can see in the screenshot on this slide, Jad parsed each class file and generated a corresponding Java source file. By default, Jad assigns the .jad extension to the files that it generates, but these are regular Java source files, which you can open using your favorite text editor.



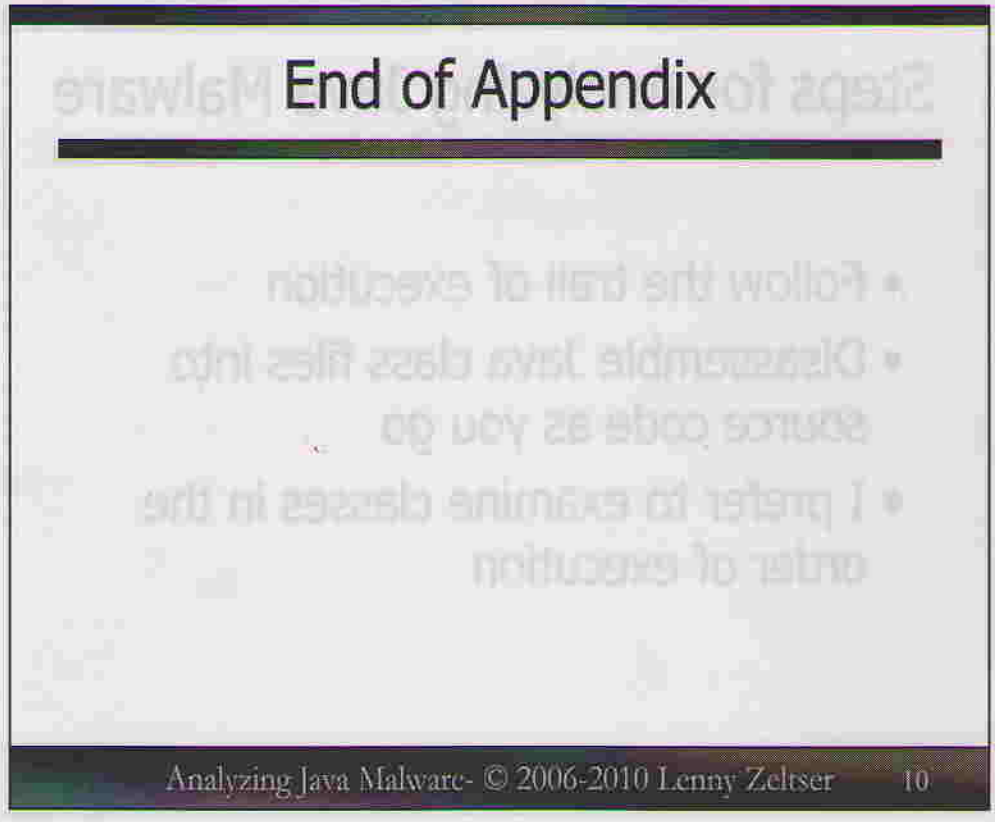
Let's take a look at a code fragment of the "`runner`" class that Jad kindly disassembled for us. The first line that you see here attempts to e-mail the author of this specimen, to notify him or her that the infection took place and where the victim's system is located. Next, "`runner`" instantiates the TFTP daemon, which allows the specimen to transfer files to and from the infected machine.

The malicious program also instantiates a scanner that allow this specimen, which turned out to be a worm, to scan the network for vulnerable systems. The worm also opens a backdoor on the victim's machine, that allows the author to connect to the system using telnet. The "`$datastore`" parameter that you see passed to these classes is used to supply them with configuration parameters.

Additionally, it instantiates the "`dosd`" class, that is programmed to launch a denial of service attack against the www.microsoft.com server. Finally, the program contains functionality that allows Storm to mail-bomb the address gates@microsoft.com.



Let's sum up the process of analyzing Java-based malicious code. The general idea is to follow the program's trail of execution, starting with the first class, and iteratively examining every class that gets called. You can use Jad, or a similar program, to disassemble Java class files as you go, or you can disassemble them all at once. This process isn't very complicated, especially in comparison to examining compiled native executables, because we get the privilege of seeing the source code of such Java-based malware specimens.



You've reached the end of this Appendix to the FOR610.1 course section.