CS155: Computer and Network Security Spring 2005

Programming Project #3

Due: Thursday, June 2nd, 2005, 11:59 pm (no late days)

1 Introduction

1.1 Summary

The aim of this project is to familiarize you with a few basic network hacking techniques. You will start by manually sniffing packets using standard network monitoring tools. In the second phase of the project, you will use a packet capture library to programmatically capture packets and extract useful information. In the third phase, you will go one step further and actually inject spurious packets on the network. In the final phase, you will put on your white hat and create a basic intrusion detection system.

You may either work alone or in pairs. You should not collaborate with others outside of your group.

You will use the Boxes system again. Remember to test your code in a closedbox.

1.2 The Setup

The assignment tarball is available on the class website and at /usr/class/cs155/projects/pp3/.

You will be working with four separate Boxes system instantiations, and we will provide you with a copy-on-write file-system image for each of them.

1. **Client** - This machine is home to several users who use the network to access services on the server. You do not initially know any usernames or passwords on this machine. The image **clientcow** is used for the client. It should be started up with the IP address **10.64.64.64**

2. **Server** - This machine runs several network daemons which provide services for the users on the clientcow. You do not initially know any usernames or passwords on this machine. The image **servercow**

is used for the server. It should be started up with the IP address **10.64.64.65**

3. **Attacker** - This machine is controlled by a malicious user (you) with the intent of eavesdropping and performing network attacks on the client and the server. You have access to user:user and root:root on this machine. Preinstalled in this image are: The network monitoring utilities that you will use in Phase 1 and the TCP SYN flood and port scanning utilities that you will use in Phase 4. We have put the starter code for the programmatic exploits in /home/user/pp3. You are provided with two programs - sniff.c and inject.c - to be used for phases 2 and 3 of the project respectively. The image **attackcow** is used for the attacker. It should be started up with the IP address **10.64.64.66**

4. **Monitor** - This machine is controlled by a good user (you) with the intent of monitoring the network for SYN floods and port scans directed at hosts on this network. You have access to user:user and root:root on this machine. We have put the starter code for the intrusion detection system in /home/user/pp3. You are provided with one program - ids.c - to be used for phase 4 of the project. The image **monitorcow** is used for the intrusion detection system. It should be started up with the IP address **10.64.64.67**

All four machines will be connected to each other on the Ethernet. Try pinging the client and server from the attack virtual console. Also, the four machines are connected via a (virtual) hub, so that the attacker machine can actually see the packets sent from the client to server and the monitor machine can see all packets on the network. **Important:** In order for the network to function as a hub, you must use the version of **string** that is included in the assignment tarball, **NOT** the version in /var/boxes/.

1.3 Recommended Reading

For an overview of the various network protocols, you should use the site

`http://www.networksorcery.com`

It gives a short explanation of all the protocols which you will have to deal with during the course of the project. We will provide references to relevant material as and when required while describing the requirements for each phase of the project.

2 The Assignment

2.1 Phase One : Manual Packet Sniffing

Your goal is to use the packet sniffing software installed on the **attackcow** image to learn information about the traffic on the local network. We have installed tcpdump, tcpflow and tethereal for you. Make sure to read all of the manpages so that you know which program is most appropriate for each task.

You will use these programs to answer the following questions (in your README)

1. Which protocols do you see being used for the communication between the client and the server?

2. For each of these protocols, describe what information you can learn about the client or server just by passive network sniffing.

3. For the protocols that use cleartext passwords for logins, list the username/password pairs that you were able obtain through sniffing. For each pair that you list, you need to give us the sniffer command you used to get it.

You might want to check out the well-known ports listed on **networksorcery**, while figuring out which protocol is used. For some basic background on packet sniffers, the following articles by Karen Frederick would be worth reading :

```
http://www.securityfocus.com/infocus/1221 (part 1)
http://www.securityfocus.com/infocus/1222 (part 2)
http://www.securityfocus.com/infocus/1223 (part 3)
```

2.2 Phase Two : Programmatic Network Sniffing

For this phase of the project, you will use the **libpcap** packet capture library to programmatically reconstruct sniffed FTP data between the client and the server.

Starter code for this phase is in the file **sniff.c** This code does the work of setting up the packet capture process. Your objective is to write code which will

- identify a new FTP data transfer on the wire
- save all the captured data to local file by the name **cap_file**
- identify when the data transfer is complete, close the file and exit

Here are some simplifying assumptions

- You do not have to deal with IP fragmentation
- You can assume that all TCP packets are eavesdropped in order
- You can assume that there is only one FTP user present in the network.

For a very detailed description of the FTP protocol you will need to check out the RFC :

`http://www.w3.org/Protocols/rfc959/Overview.html`

However, there is more information here than you need for this project. A better option is to first check out the FTP description on **networksorcery** and then use some of the sniffer information from the phase one to figure out which are the useful packets for your purpose.

Knowing which packets are useful to you is only half of the job. You will also need to know how to extract the data out of the packet. The starter code parses a packet up to the TCP header – you will need to go through the TCP header format to be able to determine exactly how much data each packet contains and where in the packet the data is located. Again, **networksorcery** is the best reference.

The following tutorial on **libpcap** is useful :

`http://www.cet.nau.edu/~mc8/Socket/Tutorials/section1.html`

Once you are done, use **md5sum** to compare the integrity of the sniffed file with that of the file actually being transferred to the client. (Hint: you may need to use the information that you obtained in Phase 1 in order to determine if you sniffed the file correctly.)

2.3 Phase Three : Active Packet Injection Attack

In this phase of the project, we will combine packet capture techniques learned above with a packet construction and injection library to perform

an insertion attack on an existing RLOGIN session.

You will be working under the following scenario :

- The client logs in to the server using the RLOGIN protocol. Any authentication of the client is done during this phase. After this setup, the client and server use a TCP connection for the rest of the session. The TCP connection is connected to port 513 on the server.
- You come into the picture after the initial authentication has been done and the client/server are already communicating over the TCP connection. (Note: since the simulated RLOGIN user will logout and re-login every few minutes, it is acceptable if you manually observe the network traffic to determine if a RLOGIN session has been established before starting your program.)
- The RLOGIN client regularly sends the **date** command to the server. This generates new TCP traffic on the network, which you are ever-ready to sniff.
- Your aim is to gather enough spoofing data at the TCP, IP and the Ethernet layers so that you can construct a packet which looks to the server like it came from the client. Obviously, the payload of this packet is not a simple newline, but your EGG containing a malicious command.

  For the purpose of this project we require that your EGG should touch a file by the name of your Leland username (either of the 2 partners) on the server. (Although it could be anything which the client would normally be able to execute using the remote login session to the server - maybe **rm -rf ***)

Here are some hints and simplifying assumptions

- You do not have to deal with IP fragmentation.
- You can assume that there is only one rlogin session present in network.
- It may be that case that a succesful attack will cause the rlogin session on the client to stop responding. If this happens, see if you can kill the rlogin process on the client using information from Phase 1. (Hint: What are some common mistakes that users make when choosing passwords?)

We use the Libnet Packet Construction library written by Mike Schiffman among others to construct packets and inject them onto the network. We will be using the latest stable version 1.1.1 of the library. The project homepage is

`http://www.packetfactory.net/Projects/Libnet/`

Starter code for this phase is in the file **inject.c** Again, we have set up the pcap capture process for you. Your objective is to write code which will

- capture packets on the targeted TCP connection
- determine the required information for constructing a spurious packet
- when all information has been gathered, inject your packet on the network

You must familiarize yourself with using the Libnet library. The following tutorial might be helpful

`http://www.security-labs.org/index.php3?page=libnet`

Also, go ahead and download the Libnet source tree. Under the sample/ directory you will find several examples of constructing TCP packets.

A good approach to this problem is to create your own connection to the rlogin server and use the network monitoring tools from Phase 1 to examine the packets. (Hint: you will need information from Phase 1 in order to gain access to the server.) You should be able to filter out all extraneous network traffic by carefully choosing the options to the networking monitoring program.

Connect your client to the server using the rlogin command. Start sending newlines or other characters from the client window. Sniff and analyze what traffic each character generates on the network. Since we are dealing with a remote login session (which is interactive), you will see the values being sent from the client to the server and then being echoed back to the client for display on the screen.

Now play around with the tcp samples in the libnet src - inject packets in the network and analyze them using the network monitoring tools. Do they look similar to what you want? What parameters do you need to modify and where? What information cannot be hardcoded and what information

needs to be sniffed out of the rlogin connection each time?

We have provided hints in the starter code on the data which you will need to gather. The starter code also outlines the packet construction process.

2.4 Phase Four : Intrusion Detection System

For this phase of the project, you will again be using **libpcap** to capture packets in order to conduct your analysis.

Starter code for this phase is in the file **ids.c** Your goal is to construct an intrusion detection system which detects: a TCP SYN flood directed at any one of the hosts on your network and a port scan directed at hosts on your network.

A TCP SYN flooding attack occurs if n SYN packets are sent to a particular destination IP and port over time t, by hosts that do not respond to corresponding SYN-ACK packets. Typically we are interested in values of n and t that cause the connection queue on the receiving host to fill up. For this to occur, n must be bigger than the queue size, and t must be less than the timeout time for half-open connections.

A TCP SYN flood is only meaningful when conducted against a listening (or open) port. A listening TCP port is one which, upon receipt of a TCP SYN packet, will respond with a SYN-ACK packet. This is as contrasted to a non-listening (closed) TCP port which will respond to a SYN packet with either a reset (RST) packet or nothing at all.

The mechanism of a TCP SYN flood relies on the fact that finite buffers are allocated for new TCP connections. In normal TCP operation, one party (the client) will send a TCP SYN to a second party (the server). When the server receives this SYN, it generates a SYN-ACK in response and allocates buffers for this pending connection. Then when the client receives the server's SYN-ACK, the client responds with an ACK to the server. When the server receives the client's ACK, the three-way handshake is complete: those allocated buffers are freed up and the connection moves to the ESTABLISHED state.

What happens when a TCP host receives a SYN-ACK for a SYN that this host never sent? A well-behaving TCP host would respond with a reset

(RST) to inform the sender that this host is not aware of any connection between the two. Then the receiver of that RST would deallocate any buffers it had reserved for this connection.

The idea of the attack should be clear from this context: the malicious party sends multiple TCP SYNs to a host specifying some port P. Generally the attacker will "spoof" the source IP address of these packets so that he can be sure that the SYN-ACK will not reach a valid host. Why is this important? As above, if the SYN-ACK reached a valid host, that host would respond with an RST (since that host did not generate the SYN for which he just received a SYN-ACK).

So the first part of phase 4 is to build a TCP SYN flood detection mechanism. In your counting of TCP SYNs sent to a particular host (IP) and port, be sure **not** to include either: (1) SYNs sent to non-listening ports (since no buffers are allocated on receipt of such SYNs) or (2) SYNs sent from valid hosts (that is, those which are part of a 3-way handshake which completes successfully). Including either of these would result in your intrusion detection system generating false positives, which is not a desirable outcome.

Note as well that your system will have to learn dynamically which TCP ports are listening and which are not.

More details on SYN floods and on **neptune**, which is a utility that generates TCP SYN floods (and is available on **attackcow** in /usr/bin) can be found here:

`http://www.phrack.org/phrack/48/P48-13`

Your objective is to write code which will

- identify a TCP SYN flood parametrized by the given number of TCP SYNs in the given time (in seconds)
- log the victim machine and port and the time each participating SYN packet was received
- then exit after logging this SYN flood information to a file

The second part of Phase 4 is port scan detection.

A port scan is parametrized by the number of "suspicious-looking packets" sent to a network and the time over which these are sent. Port scans are network reconnaissance; they are conducted to learn: which hosts are up (pingable), which TCP ports on those hosts are open or listening, and what software is running on these hosts and/or ports.

A port scan occurs if n TCP-information-gathering and/or ICMP echo request packets are sent to any IP on the network over time t. What is a TCP-information-gathering packet? Any packet which results in the victim machine revealing whether a particular TCP port is listening or not with the notable exception of such TCP packets which are part of legitimate TCP connections.

(We define legitimate TCP connections as those which complete the 3-way handshake - though the intrusion detection system may not be running when a particular TCP connection completes its handshake so you will need to devise other signatures of valid TCP connections so that you do not mistakenly classify legitimate TCP traffic as malicious.)

A nice high-level introduction to port scanning can be found here:

`http://nob.cs.ucdavis.edu/~bishop/talks/Pdf/2000portscan.pdf`

Basic port scans entail the use of ICMP echo requests (pings) to hosts. This of course reveals which hosts are up or, more accurately, which hosts are responding to pings. After determining which hosts are pingable, a port scan might then move on to sending TCP SYN packets. The goal here is not to flood the destination but rather to learn which TCP ports are open on any given host.

Since many intrusion detection systems detect SYNs sent to a variety of ports, attackers have moved on to more stealthy TCP packets which would reveal the same information (is that port listening or not?) but which might be more likely to escape detection. Examples of such packets are the TCP FIN packet, the TCP SYN-FIN packet, the TCP NULL packet (which has no flags set), the TCP XMAS packet, and others.

More details on port scanning and so-called stealth port scan packets can be found here:

`http://www.securityfocus.org/excerpts/syngress-3/1`

`http://www.securityfocus.org/excerpts/syngress-3/2`
`http://www.securitydocs.com/library/3012`

A crucial component of this task will be to build logic which can dynamically identify which traffic is valid: that is, between legitimate hosts. Given this logic, you will be able to determine whether a TCP FIN packet, for example, is part of this legitimate traffic or is, instead, a reconnaissance maneuver.

Whereas with TCP SYN floods, obfuscating the source of the flood was a prerogative, in port scanning the objective is to learn. The attacker cannot learn which ports are open unless he receives the SYN-ACKs or other packets sent by the victim in response to the attacker's port scan packets. Therefore, for your intrusion detection system, logging the source IPs of port scan packets will be critical for identifying the source of the scans after the fact.

Your objective is to write code which will

- detect a port scan on hosts on your virtual network parametrized by number of packets and time (in seconds)
- log information about each packet believed to be part of this port scan (source IP, source port, destination IP, destination port, packet type (e.g. SYN, FIN), and time received)
- then exit after logging this detection information to a file
- Your README should briefly describe your design and testing of the port scan detection mechanism: how you differentiated between legitimate and likely nefarious TCP traffic; data structures used etc.

Here are some simplifying assumptions

- The four hosts on this network (client, server, attacker, monitor) are the only targets of a port scan attack that your intrusion detection system needs to be concerned about.
- Only ICMP echo requests and TCP packets which could learn whether a port is open or not are to be considered port scan packets
- Your intrusion detection system will be operating in either TCP SYN flood detection or port scan detection mode at any given time: not both simultaneously.

nmap can be used to conduct a variety of port scans (and is available on **attackcow** in /usr/bin); **nmap**'s manpage can be found here:

`http://www.insecure.org/nmap/data/nmap_manpage.html`

3 Miscellaneous

3.1 Restrictions

There might be other vulnerabilities which could exploit to obtain username/password pairs for Phase 1. However, you are not allowed to use such methods.

3.2 Halting Boxes

In order to avoid corrupting your Boxes filesystems when exiting, you must cleanly halt the program. This is made difficult by the fact that you are not given root access to **clientcow** or **servercow**. To solve this problem, we have installed **/sbin/halt** as setuid-root. Now all that you need to do is sniff one username/password pair...

3.3 Deliverables

As in the previous programming assignments, you will use the online submit script **from a saga**,

`/usr/class/cs155/bin/submit.`

This is pp3. The directory which you submit must contain the Makefile and all source files necessary when the make command is issued.

Along with your submission, you must include a file called ID which contains, on a single line for each person in your team, the following: your SUID number; your Leland username; and your name, in the format last name, comma, first name. An example:

```
$cat ./ID
3133757 binky Clown, Binky The
$
```

If you work alone, ID should have exactly one line. If you work with a partner, ID should have exactly two lines. You should have a README file which answers the questions asked in Phase 1 and describes design and

testing for Phase 4 (as above). If you want to include any other information about Phases 2, 3, and 4, you are free to do so. Finally, you may want to include a TA-README file with comments about your experiences, or suggestions for improving the assignment.