

Programming Project #3

Due: Thursday, May 27th, 2004, 11:59 pm

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.

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 three 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. The network monitoring utilities that you will use in Phase 1 are preinstalled on this image. 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**

All three machines will be connected to each other on the Ethernet. Try pinging the client and server from the attack virtual console. Also, the three machines are connected via a (virtual) hub, so that the attacker machine can actually see the packets sent from the client to server. **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.

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 Leland submit script,

```
/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. If you want to include any information about Phases 2 and 3, 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.