14-742 Security in Networked Systems
Mini Project 1

Assigned: 18 February 2010
Due: 28 February 2010 11:59pm EST

1 Project Overview

This mini project will consist of implementing small attacks against TCP and SSL/DNS in C. It will consist of three parts: in part 1, you will implement a TCP reset attack against a client/server connection, in part 2, you will hijack an ongoing TCP session and in part 3, you will implement man-in-the-middle DNS spoofing attack against an SSL connection.

You will be using the `libpcap` and `libnet` libraries to read and write raw TCP and UDP packets across to an ethernet interface. Since both of these libraries require root priviledges, you will not be able to test your work on the andrew machines. We are providing the Knoppix Linux distribution for you to use, which can be booted and run from a cd, (detailed in a later section) however any reasonably modern Linux system will work also.

2 Introduction to Autolab

1. You will use the AutoLab system to download the assignment, as well as submit your finished code for grading. The autolab server is available at `http://vivendi.lab.ece.cmu.local:18731/14742-s10/autolab.pl`. Notice that this server is on CMU's local network. This means that you will not be able to access it directly from home. However, we have found that the CMU library's WebVPN service is compatible with Autolab (available at `https://www.vpn.cmu.edu/+CSCOE+/logon.html`). You can use WebVPN to access the website from off-campus.

2. Since this is your first time using AutoLab, you must Create your account before you can access its functionality. To do this, select the "Create" link along the top of the web page for this course. Use the usernames that the TAs have instructed you to use.

3. Once your account on AutoLab has been created, you will be able to log in and view all active assignments. You will see three labs listed in the Autolab home: `L1 - tcpresetlab`, `L2 - tcphijacklab` and `L2 -sslmitmlab`. Note that all of these labs are required for this mini project; we have seperated them on Autolab to make testing with the Autograder easier for you.

3 TCP Reset Attack

1. Write a simple TCP client-server application with the following characteristics: the server's receive buffer is set to at least 2^{16} bytes and the client side sends periodic bursts of data to the server. Each burst of data should consist of one line of a file which is specified on the command line.

2. Use the `libnet` library to implement the TCP Reset Attack exploit (TCP poisioning). Your program should send a stream of spoofed reset packets to the server application; each packet should cover an appropriate range of potential TCP sequence numbers.

3.1 Detailed specification for client, server and attack code

To get you started we have provided the skeleton code for the client, server and attack code, named client.c server.c and reset.c respectively. We have also provided a Makefile that you can use to compile the programs. Please do NOT change the names of these files nor the content of the Makefile, as the autograder will use the exact same Makefile to grade your solutions.

Note that this is the EXACT specification of the behavior of the server, client and attack code that we expect. Failure to follow this specification means that you will receive zero credit for each test case.

3.2 General Requirements

All output must go to stdout. Any output to stderr or to a file will be ignored.

3.3 Server

The server should be reading data from the client as described above. The server will take inputs of <server ip> <server port>, so the TAs will invoke the server program as follows:

```
./server <server ip> <server port>
```

The output of the server will be the number of bytes it reads from each read() from the socket. For example, if read() returns 928 bytes the output for the first read(), 17 bytes for the second, and 23 bytes for the third, the output should be:

```
928
17
23
```

and nothing else. The server should continuously read data from the socket and output the return value of read() on a single line, until the connection is reset.

The students are expected to output the result of read() every time, including error values and the message corresponding to the value of `errno` when a connection is reset.

3.4 Client

The client should take inputs of <server ip> <server port> <client port> <file name>, such that the client program will be invoked as follows:

```
./client <server ip> <server port> <client port> <filename>
```

The <file name> is a file that the TAs will provide, the client is supposed to read one line of the file at a time (until the newline character) and then send this data in ONE write() call (so you are sending one line at a time in order). Please note that one line of this file will not exceed 1024 bytes, including the newline character. This may be useful when choosing the sizes of your buffers.

The output of the client will be similar to the server, but in this case, it will be the number of bytes sent. For example, if the first line of the file (including the newline) is 8 characters, and the second line 239 characters, then the output should be:

```
8
239
```

and nothing else. The client will continuously send data until either the client reaches the end of file, or if the client has sent the max number of bytes as specified by this specification (800 kilobytes), or if the connection is reset in which the case the client will output the socket error code.

3.5 Attack Code

The students are to implement their attack code using libnet (we will check for this) for packet creation and injection. Note that this code will have to be run as root (please see disclaimer below in this document). The attack code should implement the TCP sequence number guessing attack and successfully reset the connection between the server and client. We do not require any output from this code as long as the output of the server and client is done correctly.

When you have your attack code working (it is able to reset the client-server connection), use the tcpdump utility–and your ingenuity–to count the number of spoofed packets required to reset the connection over at least five trials. We will compute the average length of time your attack code takes to reset the TCP connection and your

final grade will depend on the duration your code takes to excute compared to the rest of the class.

The attack code should take inputs of

```
<client ip> <client port> <server ip> <server port>
```

, such that the attack rogram will be invoked as follows:

```
sudo ./reset  <client ip> <client port> <server ip> <server port>
```

4 TCP Session Hijacking Attack

1. Use the same client as you wrote for Part 1. However this time you would be using the server binary that is included in the tarball.
2. Use the `libpcap` library to sniff the connection packets.
3. Use the `libnet` library to craft and transmit a well-formed TCP response in order to hijack the session. The crafted TCP packet should have data as

   ```
   0 kind server, <andrewid> needs your blessings.
   ```

4. The above message makes the server reply with a secret key. Sniff this response and print out the data part.

4.1 Detailed specification for client, server and attack code

To get you started we have provided the skeleton code for the client and attack code, named client.c and hijack.c respectively. The server binary is provided which you should be able to execute on the image we provide. We have also provided a Makefile that you can use to compile the client.c and hijack.c. Please do NOT change the names of these files nor the content of the Makefile, as the autograder will use the exact same Makefile to grade your solutions.

Note that this is the EXACT specification of the behavior of the server, client and attack code that we expect. Failure to follow this specification means that you will receive zero credit for each test case.

4.2 General Requirements

All output must go to stdout. Any output to stderr or to a file will be ignored.

4.3 Server

The server binary provided would be reading data from the client as described above. The server will take inputs of <server port> <andrew-id>, so the TAs will invoke the server program as follows:

```
./server <server port> <andrew-id>
```

4.4 Client

The client for this part can be the same client as for Part 1. The client should take inputs of <server ip> <server port> <client port> <file name>, such that the client program will be invoked as follows:

```
./client <server ip> <server port> <client port> <filename>
```

The <file name> is a file that the TAs will provide, the client is supposed to read one line of the file at a time (until the newline character) and then send this data in ONE write() call (so you are sending one line at a time in order). Please note that one line of this file will not exceed 1024 bytes, including the newline character. This may be useful when choosing the sizes of your buffers. The client will continuously send data until either the client reaches the end of file, or if the client has sent the max number of bytes as specified by this specification (800 kilobytes).

4.5 Attack Code

The students are to implement their attack code using libpcap and libnet (we will check for this). Note that this code will have to be run as root (please see disclaimer below in this document). The server we have given you accepts connection from a client and reads its messages. When this server receives a specific message (containing your andrew id), it will reply with a secret response. The client that you will use will never generate and send this specific message to server. So, your job is to implement an attack that hijacks the ongoing session between client and server and sends this message to the server (impersonating as client). To achieve this, you will need to sniff packets (to obtain sequence no. information) and using the information you sniffed, create a forged packet.

The attack code should be able to obtain the server's reply.

The attack code should take inputs of

```
<client ip> <client port> <server ip> <server port> <andrew-id>
```

, such that the attack rogram will be invoked as follows:

```
sudo ./hijack  <client ip> <client port> <server ip> <server port> <andrew-id>
```

Start your `hijack` program after server and client. Your hijack program should only produce single line of output showing the reply you got from the server (only the 'data' part of the packet). You may not need to use all the command line arguments supplied to the hijack program. Use only what you need, but make sure to parse all of them.

5 SSL "Man in the Middle" Attack

For this part of the project, you will perform an attack against SSL by injecting a fake response to a DNS request for example.com, thereby redirecting https requests to a malicous web server, and bypassing SSL security.

This attack will consist of two parts:

1. Use the `libpcap` library to sniff outgoing DNS request packets from the victim. Observe each packet, searching for a DNS query for `example.com`. Once the query is observed, it should be used as the trigger for the DNS packet injection.

2. Use the `libnet` library to craft and transmit a well-formed DNS response to the victim's query, which directs the victim to a malicous web server, rather than `example.com`. If the spoofed DNS response arrives first, the subsequent legitimate response will be disregarded.

5.1 General Requirements

All output must go to stdout. Any output to stderr or to a file will be ignored.

5.2 Specification for attack enviornment and code

5.2.1 Source Code

We have provided skeleton code, sslattack.c and sslattack.h, that you will need to modify to execute your attack. We have also provided a Makefile that you can use to compile the programs. Please do NOT change the names of these files nor the content of the Makefile, as the autograder will use the exact same Makefile to grade your solutions.

Note that this is the EXACT specification of the behavior of the attack code that we expect. Failure to follow this specification means that you will receive zero credit for each test case.

5.2.2 Attack Enviornment

You will need to run the attack code with root privileges on the system you wish to attack. If you want, you may use the Knoppix enviornment provided, which is detailed in the next section of this document. Once your attack code is running, any DNS requests to example.com should resolve to the malicous IP address. You can generate DNS requests using a web browser or with the command `wget --no-check-certificate https://example.com`, which is what the autograder will use to evaluate your output. The `--no-check-certificate` flag is necessary because SSL can detect when an invalid or misnamed certificate is being used, as is the case for our attack. This option will ignore the problem, and is analogous to the victim accepting a warning in their browser. Also note that when you are testing this outside of Autolab, you should not use `example.com`, because there is no SSL server there. Instead, use a website like `paypal.com`, which does have an SSL server.

5.2.3 DNS Configuration

Since this attack is essentially a timing attack against your local DNS server, it is necessary for your forged DNS response to arrive at your host before the legitimate response. However, if the address is cached in your network's internal DNS server, this may not be the case, even if you are running your attack on your local machine. Therefore, you should reconfigure your development environment to always query an

external DNS server. In Linux, this can be done by changing the file `/etc/resolv.conf` to:

`nameserver <IP of a DNS server located far away>`

For a comprehensive listing of DNS servers in exotic locales, visit: `http://www.dnsserverlist.org`. The TA's personal favorite is `4.2.2.2`.

5.3 Attack Code: sslattack.c

The sslattack executable should take inputs of

`<spoof address> <spoof site name>`

, such that the client program will be invoked as follows:

`sudo ./sslattack <spoof address> <spoof site name>`

The <spoof address> should be the IP address of the malicous web server to which you want to redirect your DNS request. For your development, you may use any IP that you know represents a web server running SSL.

The <spoof site name> should be the site name for which you want to trigger your DNS injection on. Again, you should use any website using SSL, e.g. `https://www.paypal.com`.

The functions in sslattack.c which you will need to modify are `main`, `analyze_packet` and `spoof_dns`. In `main`, you will need to add code to read in packets from the pcap device. In `analyze_packet`, you will need to parse the packet for DNS requests to the specified address. Finally, in `spoof_dns` you will need to craft and transmit a fraudulent DNS response.

6 Disclaimer

Please note that we know that your attack code has to be run with root privileges. Any attempt to hack, destroy, snoop or attack the grading machine is in direct violation of CMU's Computing Policy. The TAs have enabled procedures such as syscall tracing and logging to track down the responsible user(s) should malicious activity occur. This type of offense may be dealt with by measures up to and including expulsion, in accordance with the CMU Computing Policy.

7 Knoppix Introduction

In order to facilitate this project, we have packaged a Linux "live cd" (Knoppix) for students who do not have access to a Linux machine with root. The ISO image is available here: `http://vivendi.lab.ece.cmu.local:18731/14742-s10/www/KNOPPIX_V5.1.1CD-2007-01-04-EN-PLUS-libnet-libpcap.iso` **Note:** do not just download the Knoppix ISO from another mirror, because this Knoppix ISO is specially packaged with libnet and libpcap included (that you need for this project).

Knoppix is a bootable CD that you can use to boot your computer, under the best circumstances, it *should* boot you straight into KDE and successfully DHCP

(this means that under the best circumstances when you boot from the Knoppix cd, Xwindows and the network connection should be working).

For the remainder of this section, I have included some basic Knoppix commands you can try when things aren't working.

If for some reason knoppix does not boot into KDE, this is a simple command that might help. During the boot prompt of knoppix, type:

knoppix screen="<screen resolution>", for example if you want to boot to 800x600, this would be knoppix screen="800x600", then hit <enter>.

Another simple command to get your Internet network connection working if you are using DHCP (and your network card is eth0) is (run this after your machine boots and you get a boot prompt):

pump -i eth0

Wireless on Linux is often difficult to configure. We recommend that you use Ethernet connections when possible. If you need to use your wireless card, we suggest using google to look up configuration information for your particular wireless card.

7.1 Saving your work to a USB Key

In order to save your work to a USB Key, please see this link for saving your work to a USB Key in KDE: `http://www.shockfamily.net/cedric/knoppix/#usb`

If you want to use the command line to mount your USB key, please use the following steps:

First, insert the USB key into your USB slot. Then, type mkdir /mnt/usbkey (as root)

Now, type: `dmesg` and look at the last few lines of the output for lines such as:

```
usb 1-6: new high speed USB device using address 3
hub 1-6:1.0: USB hub found
hub 1-6:1.0: 1 port detected
usb 1-6.1: new high speed USB device using address 4
Initializing USB Mass Storage driver...
scsi3 : SCSI emulation for USB Mass Storage devices
  Vendor: USB 2.0   Model: Flash Disk        Rev: PROL
  Type:   Direct-Access                      ANSI SCSI revision: 02
SCSI device sdb: 256000 512-byte hdwr sectors (131 MB)
```

Now from the above you can see that the USB key is mounted on the SCSI device sdb, so you type (as root):

mount /dev/sdb /mnt/usbkey

If you look in /mnt/usbkey, your files should now be there.

7.2 Saving your work over the network

If your Internet connection is working, you can just use scp (secure cp) to copy the files you are working on to and from any other machines (such as unix.andrew.cmu.edu). The command to copy the file lab01.tgz from the current directory to your home directory on unix.andrew.cmu.edu is:

```
scp ./lab01.tgz <your andrew_id>@unix.andrew.cmu.edu:~
```

7.3 Useful Links

Below are some useful links I found for knoppix novices:
`http://unit.aist.go.jp/itri/knoppix/knowing-knoppix/main/index3.html` `http://www.knoppix.net` (click on documentation)

8 Assumptions and Hints

- Assume you will deploy your client, server, and attack module on one machine in the security lab. Hence you will be connecting to 127.0.0.1 most of the time. You will definitely want to run tcpdump to test/verify your code.
- You can install libnet (packages libnet1 and libnet1-dev) and libpcap (packages libpcap0.8 and libpcap0.8-dev) on your own linux machines and work on it too. But make sure your solutions work on autolab too.
- Haven't programmed with sockets before? There are plenty of good tutorials online. See `http://beej.us/guide/net/` for example.
- You do not want your client to simply loop through the input file reading lines and sending them as fast as it can.
- You will know your TCP reset attack code is working when you see something like this on the server machine:

```
928
928
928
-1
```

- And you will see something like this on the client machine:

```
928
928
928
928
Connection reset by peer: write()
```

- You will know if your TCP Session Hijacking code is working if you see correct message received at the server and your hijack binary prints the reply from server.

```
    SERVER'S OUTPUT:
$ ./server 9999 andrewid
18 : Hello from client.
18 : Hello from client.
18 : Hello from client.
18 : Hello from client.
18 : Hello from client.
18 : Hello from client.
18 : Hello from client.
45 : 0 kind server, andrewid needs your blessings.  /* This message is
from your attack code */

    ATTACK CODE'S OUTPUT:
$ sudo ./hijack 127.0.0.1 7777 127.0.0.1 9999 andrewid
Server's Reply:
Server's Reply: Yo andrewid ! Here is your secret 'f7733c3bc84b63575416ea482af23a85'

/* Your hijack client should terminate after printing the server's response.
 * DO NOT LEAVE YOUR HIJACK PROGRAM LOOPING INFINITELY. */

    CLIENT'S OUTPUT:
$ ./client 127.0.0.1 9999 7777 junk
18
18
18
18
18
18
18
18
...
```

- You will know your SSL MITM attack code is working when wget (or your web browser) reports that the SSL certificate it receives does not match the page it requested. When running in autolab, you will receive output similar to this:

```
sandbox: wget: --20:56:43--  https://www.example.com/
sandbox: wget:            => 'output'
sandbox: wget: Resolving www.example.com... 172.19.132.149
sandbox: wget: Connecting to www.example.com|172.19.132.149|:443... connected.
sandbox: wget: WARNING: Certificate verification error for
                     www.example.com: self signed certificate
sandbox: wget: HTTP request sent, awaiting response... 200 OK
sandbox: wget: Length: 128 [text/html]
sandbox: wget:
```

```
sandbox: wget:     0K                                  100%    4.88 MB/s
sandbox: wget:
sandbox: wget: 20:56:43 (4.88 MB/s) - ‘output’ saved [128/128]
sandbox: wget:
sandbox: killing reset...
sandbox: reset exited.
sandbox: file: <html><body><h1>Go ahead... enter your credit card number.
               You’re secure.  You’re using SSL, right ;)</h1></body></html>
Downloaded the hacker file!
```

The second to last line displays the file that was downloaded. The one displayed in the example is on the attacker's server. Otherwise, it will display a message like:

```
sandbox: file: <html><body><h1>This is the real
                example.com page.</h1></body></html>
```

*** Good Luck ! ***