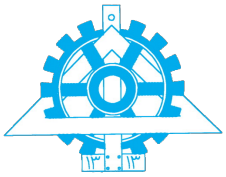
In the name of Allah

بسم الله‌الرحمن الرحیم

Network management and security Laboratory Manual



University of Tehran
دانشگاه تهران

School of Electrical and Computer Engineering
دانشکده مهندسی برق و کامپیوتر

Computer Network Lab
آزمایشگاه شبکه‌های کامپیوتری

Dr. Ahmad Khonsari - احمد خونساری
a_khonsari@ut.ac.ir

Amir Haji Ali Khamseh'i - امیر حاجی علی خمسه‌ء
khamse@ut.ac.ir

Sina Kashi pazha - سینا کاشی پزها
sina_kashipazha@ut.ac.ir

Mohammad Ali Shahsavand - محمد علی شاهسوند
mashahsavand@ut.ac.ir

Amirahmad Khordadi - امیر احمد خردادی
a.a.khordadi@ut.ac.ir

December 1, 2018

۱۰ آذر ۱۳۹۷

Exercises on secure applications

1 Man in the Middle

In this excercise we will study security vulnerability of ftp and telnet protocol . To do so, wi will create mininet topology with single hub[1] connected to three hosts then we will connect from h1 to h2 through ftp and telnet connection and capture h1 password on h3. Let's do it.

1. Start pox controller with below command to force mininet switches act like hub.

   ```
   $ python pox.py opeenflow.of_01 --address=127.0.0.1 --port=6337 forwarding.hub
   ```

2. Run below command to start mininet with one single switch and three hosts and connect it to pox controller.

   ```
   $ sudo mn --topo single,3 --controller remote,ip=127.0.0.1,port=6633
   ```

3. start ftp server on h2 with:

   h2> `/usr/sbin/vsftpd`

4. Run `wireshark &` on h3

5. Login to h2 and then run ftp from h1

   h1> `ftp mininet@10.0.0.2`

6. Capture h1 password on wireshark output

Repeat the above experiment, but use telnet to connect from h1 to h2 and capture h1 password on h3. [2]

Lab Report

1. Can you see the login ID and the password in the `FTP` experiment? Submit the two packets you captured.
2. Can you see the login ID and the password in the `TELNET` experiment? Submit the packets you captured.
3. What is the difference between `FTP` and `TELNET` in their transmission of user ID's and passwords? Which one is more secure?

2 Secure Transfer

Run previous mininet topology and connect it to pox controller but rather than using ftp and telnet use ssh and `sftp` as described in below steps.

1. Do step 1 and 2 frome previous section
2. restart ssh service on h2 to enable `ssh` and `sftp` service on it with:

   h2> `service ssh restart`
   h2> `/usr/lib/openssl/sftp-server`

3. Run `wireshark &` on h3
4. Login to h2 `sftp` from h1 by:

   h1> `sftp mininet@10.0.0.2`

5. Capture packets on wireshark output

Repeat the above experiment, but use ssh and save the wireshark output for lab report.

[1] hub forwards incoming packets to all of its ports, which means it always floods packets
[2] Don't foget to restart xinetd with `'/etc/init.d/xinetd restart'` on h2 to start telnet server.

Lab Report

1. In each experiment, can you extract the password from the tcpdump output? Can you read the `IP, TCP, SSH` headers? Can you read the TCP data?
2. What is the client protocol (and version) used in both cases?
3. What is the port number used by the `ssh` server? What is the port number used by the `sftp` server? Justify your answer using the wireshark output and the `/etc/services` file.

Exercises on Firewalls and Iptables

3 Firewall basic

Start mininet with default topology and Execute `iptables -L -v` on h1 and h2 to list the existing rules in the filter table. Save the output for the lab report.

Append a rule to the end of the `INPUT` chain, by executing

h2> `iptables -A INPUT -v -p TCP --dport 23 -j DROP`

on h2. Run iptables -L -v again on both hosts to display the filter table. Save the output.

1. Start telnet server on h2 with `‘/etc/init.d/xinetd restart‘`.
2. Capture packets on both hosts with wireshark
3. Try to login with telnet from h1 to h2

3.1 Lab Report

1. Can you telnet to the host from the remote machine?
2. From the wireshark output, how many retries did `telnet` make? Explain the exponential `backoff` algorithm of `TCP` timeout and retransmission.

4 Excercise Four

Keep previous mininet running and delete the rule created in the last exercise on h2, by:

h2> `iptables -D INPUT -v -p TCP --dport 23 -j DROP`

Then, append a new rule to the `INPUT` chain:

h2> `iptables -A INPUT -v -p TCP --dport 23 -j REJECT --reject-with tcp-reset`

Execute **iptables -L -v** to display the new rule. On both machines in your topology, restart wireshark output, and then telnet from h1 to h2. Save the wireshark output for the lab report.

4.1 Lab Report

1. Explain the difference between the wireshark outputs of this exercise and the previous exercise. How many attempts did `TCP` make this time?

Exercises on secure Apache server

In the exercises in this section you don't need to create mininet topology, run command on your ubuntu terminal.

5 Raw HTTP

Create two files, `index.html` and `success.html`, in `/var/www/html` directory:
$ sudo nano index.html

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<body>
<form action="/success.html"  method="POST">
Username:<br>
<input type="text" value="a.a.khordadi">
<br>
Password:<br>
<input type="password" value="1234">
<br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

$ sudo nano success.html

```
<!-- success.html -->
<!DOCTYPE html>
<html>
<body>
<h2>Goog Job!</h2>
<a href="/">back to login form!</a>
</body>
</html>
```

Now open your browser and enter your VM's IP address in URL bar and submit the login form:
`http://172.18.133.XX`
Save **wireshark** output for your lab report.

6 Secure HTTP

6.1 Activate the SSL Module

Enable the module by typing:
$ sudo a2enmod ssl
After you have enabled SSL, you'll have to restart the web server for the change to be recognized:
$ sudo service apache2 restart
With that, our web server is now able to handle SSL if we configure it to do so.

6.2 Create a Self-Signed SSL Certificate

Let's start off by creating a subdirectory within Apache's configuration hierarchy to place the certificate files that we will be making:
$ sudo mkdir /etc/apache2/ssl

Now that we have a location to place our key and certificate, we can create them both in one step by typing:
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt
Let's go over exactly what this means.

- **openssl**: This is the basic command line tool provided by OpenSSL to create and manage certificates, keys, signing requests, etc.
- **req**: This specifies a subcommand for X.509 certificate signing request (CSR) management. X.509 is a public key infrastructure standard that SSL adheres to for its key and certificate managment. Since we are wanting to create a new X.509 certificate, this is what we want.
- **-x509**: This option specifies that we want to make a self-signed certificate file instead of generating a certificate request.
- **-nodes**: This option tells OpenSSL that we do not wish to secure our key file with a passphrase. Having a password protected key file would get in the way of Apache starting automatically as we would have to enter the password every time the service restarts.
- **-days 365**: This specifies that the certificate we are creating will be valid for one year.
- **-newkey rsa:2048**: This option will create the certificate request and a new private key at the same time. This is necessary since we didn't create a private key in advance. The rsa:2048 tells OpenSSL to generate an RSA key that is 2048 bits long.
- **-keyout**: This parameter names the output file for the private key file that is being created.
- **-out**: This option names the output file for the certificate that we are generating.

When you hit `ENTER`, you will be asked a number of questions.
The questions portion looks something like this:

```
Country Name (2 letter code) [AU]:IR
State or Province Name (full name) [Some-State]:Tehran
Locality Name (eg, city) []:Tehran
Organization Name (eg, company) [Internet Widgits Pty Ltd]:University of Tehran
Organizational Unit Name (eg, section) []:ECE Department
Common Name (e.g. server FQDN or YOUR name) []:ece.ut.ac.ir
Email Address []:a.a.khordadi@ut.ac.ir
```

The key and certificate will be created and placed in your `/etc/apache2/ssl` directory.

6.3 Configure Apache to Use SSL

Open the file with root privileges now:
$ sudo nano /etc/apache2/sites-available/default-ssl.conf
You should make some changes to it.
In the end, it will look something like this. The entries in red were modified from the original file:

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
ServerAdmin admin@example.com
ServerName ece.ut.ac.ir
DocumentRoot /var/www/html
...
SSLCertificateFile /etc/apache2/ssl/apache.crt
```

```
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
...
</VirtualHost>
</IfModule>
```

Save and exit the file when you are finished.

6.4 Activate the SSL

Now that we have configured our SSL-enabled server, we need to enable it.
We can do this by typing:
$ sudo a2ensite default-ssl.conf
We then need to restart Apache to load our new file:
$ sudo service apache2 restart
This should enable your server, which will serve encrypted content using the SSL certificate you created.

6.5 Test your Setup

Now that you have everything prepared, you can test your configuration by visiting your server's IP address after specifying the https:// protocol:
`https://172.18.133.XX`
Now submit the login form. Use **wireahark** output and examine the operation of SSL.

Lab Report

1. What is the port number used by the secure Apache server?
2. Compare the general information of the received certificate with the make output saved in the last exercise. Are they consistent?
3. What is the Subject of the received certificate? Who is the Issuer of this certificate? Are they the same?
4. What is the Certificate Signature Algorithm used to generate and distribute this certificate?
5. When was the certificate signed? When will it expire?