Lab9: Secure Networking: SSH and HTTPS

COM-208: Computer Networks

Objectives

- Learning about the security behind ssh, https and making use of them.

Prerequisites

- Lecture 10
- Wireshark

HTTPS

HTTPS is a marked improvement over HTTP when it comes to authentication and confidentiality. In this Lab, we will examine the following questions about HTTPS:

1. How do HTTP and HTTPS compare with respect to confidentiality? Which of these two protocols can we trust to transmit sensitive data (e.g. passwords) over the network?

2. How do web browsers (Firefox, in particular) handle the issue of Certificate Authorities (CA) with respect to HTTPS?

Confidentiality

You are going to test the level of confidentiality that is provided to you by the following websites:

- A Test Login Page

- Yahoo!

You are going to test that for each website, as follows:

- Start by checking whether the web page address begins with "https://" or "http://". Note that HTTPS (also called HTTP over TLS, HTTP over SSL, and HTTP Secure) consists of communication over HTTP, but within a connection encrypted by Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL).

- Start a Wireshark capture. Use a filter for capturing only the traffic to/from the web server: e.g. use an initial filter based on the IP address: "ip.addr == X.X.X.X". You may find the actual IP addresses using "nslookup" or any other tool (even Wireshark).

- Refresh the login page for the website.

- Submit any username and password for each website. **Do NOT provide a valid password. One of the websites is insecure!**

- Examine the Wireshark trace to identify which protocol is used to connect to the website.

- Examine the trace to see if you can discover the username and password that you provided in the login form. For this, it is better to use a additional filter for the traffic concerning the protocol that is used. You may need to add a second filter to the initial one, by using the AND operator "&&" and examine the content of the messages that are exchanged. Consider adding "http" for the test login page, and "ssl" to for the yahoo login page.

Questions

a. Which protocols does your browser use to connect to each website? What are the confidentiality guarantees of each of the two websites?

b. **IMPORTANT:** What security advice would you give to a friend/relative who is struggling with computers about this issue: What information from the browser could they use to identify an insecure website?

Certificate Authorities

To identify itself, a web server provides you with a certificate. This certificate is signed by a Certificate Authority (CA). When a web browser receives a certificate from a website, it checks whether the CA that has issued the certificate belongs to one of the authorities that the browser recognizes.

In Firefox, recognized authorities can be found in:

Menu (top-right corner of Firefox)/Preferences/Privacy & Security/View Certificates/Authorities

The response from the web server which carries the certificate is carried on a "Certificate"-type record.

Questions

    a. Use Wireshark to discover which Certificate Authority signed the certificate of Yahoo!.

    b. Is this Certificate Authority recognized by your web browser?

Managing Certificate authorities

As we said, in Firefox, recognized authorities can be found in

Menu (top-right corner of Firefox)/Preferences/Privacy & Security/View Certificates/Authorities

For demonstration purposes, we are going to examine what would happen if we removed one of those servers from the list.

- Visit https://www.auth.gr.

- Go to the list of the recognized server from Firefox. Remove every entry of the parent CA organization *"Hellenic Academic and Research Institutions Cert. Authority"*. Then, restart Firefox to make changes take effect.

- Visit https://www.auth.gr again. What is the warning message you get?

- Now, visit https://mail.yahoo.com. Why do you not get a similar warning message for the Yahoo! website?

SSH

The `ssh` tool is a powerful utility which grants secure access to a target remote machine. One of the most common use-cases for `ssh` is to run commands on the target machine as if we had direct physical access to that machine.

In this Lab, we are going to learn how to achieve the following:

    1. Connect to a target machine.

2. Execute a command on a target machine.

3. Run a graphical application on a target machine.

4. Copy files to/from a target machine.

We will also delve a bit deeper on two important security-related aspects of `ssh`:

1. Setting up public key authentication (i.e., "How can I avoid having to type my password each time I want to connect to a target machine?")

2. Fingerprints (i.e., "How do I know that the machine I am connecting to is really the machine I asked ssh to connect to?")

Setup

(You should be able to run these exercises on just about any GNU/Linux system which runs an SSH server)

For the purposes of Lab 9, we will be using one of the machines at INF3 as the target machine. The machines at INF3 can be accessible at icin3pcX.epfl.ch, where X is a value in {01, 02, 03, …, 64}. You will use your GASPAR accounts to access to those machines.

If you receive an error on connecting to one of the INF3 machines, please try to connect to some other machine by simply choosing another X value in the hostname above. The error might be due to overloading the machine with more ssh connections than the machine can handle.

SSH use cases

Connecting to a target machine

The basic syntax which makes you connect to machine "hostname" using your current username, using the **ssh** command is:

```
$ ssh hostname
```

In our case, the "hostname" is **icin3pcX.epfl.ch**, where X is a value in {01, 02, 03, …, 64}.

If you want to log in as a different user, you should provide the following command:

```
$ ssh username\@hostname
```

(NOTE: The first time you log in to the server, you will be asked whether you accept a "fingerprint". For the moment, you should type "yes" and continue with the Lab. We will examine fingerprints later in this Lab.)

Exercise

- Connect to one of the INF3 machines using your GASPAR account.

- Use Wireshark to capture one ssh session. Which transport protocol is used? Which port does the ssh server use?

Executing commands remotely

Once you have connected to a remote machine, this launches a terminal prompt. You can use this prompt to issue commands to that machine, as if you had direct physical access to it.

If you are only interested in running a single command and then quitting the SSH session, you can specify that command directly to the **ssh** program:

```
$ ssh username\@hostname command
```

When to use

- The latter way to invoke the **ssh** command is useful if you want to perform actions on a remote machine as part of an automated script. This capability becomes even more powerful if you have set up public key authentication (covered later in this Lab).

Exercise

- Practice both ways to run a command on one of the machines at INF3 (e.g., the date command).

Running graphical applications on a remote machine (X11 forwarding)

You can also use **ssh** to run graphical applications remotely. In order to do that, you need to instruct **ssh** to forward the display server session (X11) to your current machine. This is commonly referred to as X11 forwarding.

In order to perform X11 forwarding you must provide the right flag to the **ssh** command when you instruct it to connect to the target machine. Read the **man** page of **ssh** and find out the appropriate flag for X11 forwarding.

When to use

- You don't have sufficient rights to install your favorite application on the machine you are using.

- The machine you are using is too weak to support the application you want to run. Thus, you are tapping in to the resources of the remote machine.

Exercise

a. Connect to one of the INF3 machines (without X11 forwarding) and try to run a graphical application (e.g., firefox). What is the error message you get?

b. Reconnect to one of the INF3 machines with X11 forwarding enabled this time. Try to run the same graphical application, again.

Copying files using scp

The `scp` tool (Secure Copy) to copy files between two different hosts, just like the `cp` ↪ command is used to copy files between different directories. In fact, the two tools are very similar in syntax (e.g. the -r to copy directory subtrees recursively). The basic syntax for the `scp` command is:

```
$ scp file1 file2 ... username@hostname:/path/to/target/folder/
```

where … means that you can specify multiple files to be copied from your machine (in the same way that **cp** can copy multiple files from the source directory).

Exercise

- While connected to your machine, use scp to copy a file from your machine to one of the INF3 machines. To do that, you can create a few temporary files by using Linux's touch command. Moreover, do not forget that your files on an INF3 machine is kept only until your session is closed. Hence, copying files from your machine to one of the INF machines by scp will copy the files, but also delete right after the command is executed as the scp session completes right after the copying operation is done.

  To see that the files are actually copied, you can open another terminal and ssh to your target machine. Then, you can do the copy operation from your host machine to the target machine by using the previous terminal you have opened. Lastly, you can go back to the terminal you newly opened, and check if the files are actually copied to the target machine (by ls command).

SSH security

Setting up public key authentication

By this time in the Lab, you might have realized that it can get quite tedious, having to retype your password every time you want to connect to a specific remote machine. Luckily, there is a way to automate the authentication process (i.e., so that you don't have to type your password every time) without giving up on security. You can achieve that using public key authentication.

The Bigger Picture (and the theory behind it)

Let's say that your machine is Alice, and that the target machine (in this case, one of the INF3 machines) is Bob. Your goal is to set Alice and Bob up in a way such that Alice can securely authenticate herself to Bob every time she tries to connect (i.e., perform an **ssh connection**). We can use public key authentication to assist us with this task.

Let's assume that Alice has generated a pair of keys: the private key Ka- and the public key Ka+. Alice keeps Ka- (the private key) to herself. Alice, then, needs to notify Bob about Ka+ (the public key). Since both Alice and Bob are machines under our control, we can find a way to copy the public key from Alice to Bob in some secure way.

Finally, we instruct Bob to allow Alice to authenticate herself using public key authentication, (instead of checking whether Alice can provide the right password). Since Alice

is the only machine with access to Ka- (the private key), we know that we can trust this method of authentication.

Note that if Eve manages to steal Ka- (e.g.., by stealing Alice's machine and reading the filesystem) she will also be able to authenticate herself as Alice. As an added measure of precaution, Alice could also encrypt Ka- using a passphrase. Now, when the user wants to make Alice connect to Bob, they will first have to provide the passphrase to decrypt Ka-.

From a usability perspective, this throws us back to the initial problem, where the user has to provide some kind of a password every time Alice tries to connect to Bob. Luckily, we can configure Alice in such a way, so that we have to decrypt the Ka- only once per session. For the reminder of the session, Alice can retain the decrypted key in memory without sacrificing security.

In the rest of this section, we are going to learn how to link security theory to SSH practice.

Generating a key pair

We use the `ssh-keygen` command to generate a public-key/private-key pair for SSH. We recommend that you invoke the command `ssh-keygen` in the following way:

```
$ ssh-keygen -t rsa -b 4096
```

Once you invoke the command, you will presented with a prompt to provide:

- Where to store your key-pair. You should press ENTER, and the key-pair will be stored in files `/home/username/.ssh/id_rsa` (private key) and `/home/username/.ssh` ↪ `/id_rsa.pub` (public key).

- A passphrase can be used to encrypt your key. If you specify an empty passphrase, anybody that can read the private key file (e.g. the root) will be able to authenticate as you on your target machine. Instead, if you specify a non-empty passphrase, you need to provide the same passphrase every time you connect. We will examine a secure way to overcome this restriction.

Exercise

- Generate a key pair using `ssh-keygen`. You should specify a non-empty passphrase.

Enabling public key authentication at the target

When your machine tries to connect to the target machine, the target machine checks whether it has enabled public key authentication for your machine. An SSH server performs this check by reading file `/home/username/.ssh/authorized_keys` (on the server's filesystem), if it exists, and checking whether it contains your machine's public key.

In order to enable the target machine to authenticate your machine using public key authentication, you should append the contents of your public key file (`/home/username` ↪ `/.ssh/id_rsa.pub` from your machine) to `/home/username/.ssh/authorized_keys` (on the target machine).

Exercise

- Copy the contents of id_rsa.pub from your machine to authorized_keys on the target machine (e.g. by using scp or using a graphical text editor). If the .ssh folder or the authorized_keys file do not already exist, you will have to create them.

- Test whether public key authentication works. If you have set everything up properly by now, you will be asked to provide a passphrase when you connect to the target machine. Just like in Exercise 2.1.4, the machines in INF3 will delete your files after the latest live session has expired on the target machine. Hence, keep a live session to your target machine in another terminal to prevent the authorized_keys file to be deleted.

Remembering the decrypted private keys

Moving from password based authentication to public key authentication does not provide much improvement if you have to type the passphrase every time. Fortunately, the process can be automated with the help of **ssh-agent**, which remembers the passphrase and provides it to **ssh** the next time public key authentication is used. In the INF3 configuration, **ssh-agent** is enabled by default.

However, if you need to run **ssh-agent** manually, you can do so with the following commands:

```
$ ssh-agent $SHELL
$ ssh-add /home/username/.ssh/id_rsa
```

Exercise

- Kill the **ssh-agent** process with the **pkill ssh-agent** command.

- Restart the **ssh-agent** process with the commands listed in this section, and give your passphrase. Then, try to connect to the target machine. If you have configured everything properly, you will manage to connect to the target machine without having to provide a password. Try to logout and reconnect to the target machine. You may verify that no password is needed.

Fingerprints

So far, we have seen how Alice can authenticate herself to Bob. In order for this to be secure, Alice should also have some way to verify that she is authenticating herself to Bob, and not to Persa (somebody that tries to impersonate Bob).

Known hosts

SSH handles this feature in the following way. When Alice tries to connect to Bob, Bob sends his public host key to Alice. `ssh` stores host key pairs in directory `/etc/ssh`.

If this is not the first time Alice connects to Bob, Alice will be able to recall Bob's public key from last time; she will only continue with the authentication if what she remembers about Bob matches what Bob told her about him. For the purposes of SSH, Alice keeps track of the public key she knows about Bob in file `/home/username/.ssh/known_hosts`,

Exercise

- Check whether your machine trusts the target machine. You should open file `/ ↪ home/username/.ssh/known_hosts` on your machine and compare it to the public host key file on the target machine. (`/etc/ssh/ssh_host_ecdsa_key.pub`).

Learning new hosts

If this is the first time that Alice connects to Bob, Alice will have to decide whether she trusts that the person she is receiving the public key from is indeed Bob. If she decides she wants to trust Bob, she will store his public key and continue with the authentication. SSH uses fingerprints to handle this process

A fingerprint is a summary of Bob's public key. The proper way to verify that the machine you are connecting to is indeed the machine you think it to be is to obtain the fingerprint in some other way (e.g. make a phone call to the administrator of that machine).

Exercise

- Connect to the host machine. Once you connect to the host machine, execute the following command:

```
$ ssh-keygen -l -f /etc/ssh/ssh_host\_ecdsa\_key.pub
```

Save the output to somewhere, and go back to your host machine (either by exiting from the ssh connection, or opening a new terminal on your local machine).

- Make your machine forget all known hosts it has encountered (i.e., delete file `/home ↪ /username/.ssh/known_hosts` on your local machine). Then, reconnect to the host machine and check whether the fingerprint it advertises to you, matches the one you obtained in the previous step.

Food for thought

Now that you know a bit more about the inner workings of the `ssh` tool, you should be able to answer the following questions:

Questions

1. Compare the security of using public key authentication to using passwords in the following scenario:

Suppose that you own two laptops (laptop1 and laptop2), which you use to connect to your server. At some point, laptop1 gets stolen, and the attacker is able to deduce the password and/or the private key of the laptop (depending on which authentication message is used) from information stored on the filesystem.

   a. If you were using password-based authentication, what steps would you take to prevent that attacker from accessing your server. Would that change anything in the way that you are using laptop2 to access the server?

   b. How would your response change if you were using public-key authentication to access your server? Would you have to change anything on laptop2?

2. Use `ssh` in the verbose mode (option `-v`). Describe the message exchange.

Solution (where the host machine is icin3pc03 and the target machine icin3pc07):