Sniffing and Utilizing unencrypted HTTP Packages

During this lab you will be introduced to the security risks that aris when using non-secure connections. You will work with a locally hosted `Docker` and `Wireshark`. Both tools are already installed on your Kali VM or can easily be installed on both Windows and Linux. The web application hosted in the docker container, runs default without any encryption. While this is bad practice, numerous websites are still not using SSL based encryption and rather operate on unencrypted http connections. This lab will show you the risks of not securing your connections and using unsecure connections.

The hints are encoded in base64. They can be decoded by using `echo <hint> | base64 -d` or with online tools

Docker

Setup

In order to run the vulnerable Web application, download and install docker on your machine (if not yet installed). The instructions can be found on the docker website. When docker is installed and running in the background, run the following command in the command line:

```
docker run -p 8080:8080 patricklindner/unsecured_webapp:1.0
```

This command downloads the desired Docker image from the docker hub and stars a local container from it. All network traffic on port 8080 is rerouted to port 8080 of the Docker container. If you are using a Linux distribution, this command might require superuser privileges, depending on your installation. Therefore, prepend the command with `sudo`. Wait until the image has been downloaded and started, and you see the log message

```
Completed initialization in <number of milliseconds> ms
```

Afterward, the web application can be accessed using the browser on http://localhost:8080/login.

Wireshark

Setup

In order to inspect network traffic, we will use Wireshark. Download and install it on your machine (installed on Kali already). See their website for installation instructions. In order to listen to the traffic between the web application and your browser, run it as administrator and select the network loop back interface named `lo`. Type `http` in the filter bar, and access the web application at http://localhost:8080/login via the browser. You should now be able to see multiple packets, which are exchanged between the browser and the web application. Since the connection is not encrypted, we can see the content of each package. In order to

maintain a clear overview, all captured packages can be deleted from wireshark's overview. Therefore, click the green `Restart Current Capture` Button in the Wireshark toolbar.

Usage

Stealing Passwords

When HTTP traffic is not encrypted, it is very easy for attackers to steal a password during the login process. In the following steps, we will emulate such attack.

1. Connect the browsers to the web application at http://localhost:8080/login. Login with the user `joe` and the password `s3cret`. You should be logged in now and see the secured content of the web application.
2. In Wireshark, find the package with the info `POST /login HTTP/1.1`. This is the HTTP post request which transmits the username and the password to the server. In the background, the server checks the validity of the credentials, and creates a new session for the browser. Click the package and find the `HTML Form URL Encoded` entry at the left bottom panel. You should now be able to see the credentials, you inserted in the browser.

A potential attacker could use that information to login in to-, and control the, the victim's account.

Since this method is very basic, most modern web application are protected against that by not transmitting the clear text, but an encrypted version of the password.

Session Hijacking

With the given setup, it is also very easy to gain access to a user's secured account by sniffing their session ID. Since the session ID is the only thing used by the server to identify a browser (and therefore a potentially authenticated account), stealing it enabled the attacker to impersonate the victim user. This is very helpful in scenarios, where the browser encrypts the password before transmission.

1. Log in the web application at http://localhost:8080/login and capture any http packets which are sent from the browser to the server.
2. Find the http section in the wireshark's packet inspector called Hyper Transfer Protocol. In that section find the cookie section and open it. Here you can read all cookies that have been sent in that specific request. We are specifically interested in the cookie called JSESSIONID. This is the session ID of our victim, which we can use to hijack their session. Copy the value of this cookie to your clipboard.
3. Open a new browser tab in private mode. In this mode, the browser does not share any cookies with the tabs. Therefore, in private mode, the browser is not logged in the application. Verify that the private tab is not logged in by accessing http://localhost:8080. If you are not logged in, you should be redirected to the login page.
4. In the private tab, we can now change the value of the JSESSIONID cookie in order to impersonate the victim. Open the browsers developer menu, navigate to the cookies tab, and paste the copied JSESSIONID into the JSESSIONID cookie of the private tab.
5. Reload the page, you are successfully logged in as the victim.

Stealing Private Data

Not only passwords, but all data packets that are transmitted unencrypted between the browser and the server can be sniffed. In this section we try to steal form data, which the user sends to the server. This attack works very similar to the first one, where the password is sniffed. However, the goal is different. We sniff the password in order to impersonate as the victim. Logging in with the sniffed password could leave traces on the server logs of the attackers IP address. In this attack, w use the same sniffing technique for different data.

1. Log in as the user joe and listen in wireshark to any http packets on the loopback interface.
2. After login, you will find a textbox, which asks for a secret. This secret can be sniffed by wireshark in the same way as the password from the first attack. Fill some text into the textbox and hit the `Save` button.
3. After saving, you should be able to see the transmitted http package starting with `POST /secured`. In that package, the secret private data from the textbox, can easily be red by the attacker.

Secured Connection

We now know, how easily unencrypted HTTP packets can be sniffed by a potential attacker. Let us prevent the three attacks by encrypting the data traffic. In order to encrypt HTTP packets, we make use of Transport Layer Security (TLS), the newer version of Secured Socket Layer (SSL). Using this encryption, we switch from the HTTP, to the HTTPS protocol.

Since TLS makes use of asynchronous encryption, the application server requires a public and a private key. In order to create those keys, we make use of javas keytool command line tool. We invoke the following command:

```
keytool -genkeypair -alias <a name for your key> -keyalg RSA -keysize 2048 -
storetype PKCS12 -keystore keystore.p12 -validity 3650
```

This command creates an RSA key pair with your chosen alias and stores it in the output file keystore.p12 in industry standard PKCS12 format.

After running the command, you have to define a password, that encrypts the keystore file. Furthermore, additional information, like the creators name and the organizations name are requested. All of those information can be left blank, since we host the application locally.

When the keystore has been created, we can restart the docker application with the keystore.

Run the following command from the folder where `keystore.p12` is located:

```
docker run -v .:/cert -p 8443:8443 patricklindner/unsecured_webapp:1.0
   --spring.profiles.active=https
   --server.ssl.key-store=cert/keystore.p12
   --server.ssl.key-store-password=<password to keystore.p12>
   --server.ssl.key-alias=<your alias>
```

For some systems: replace .:/cert with "$(pwd)":/cert

The HTTP traffic of the application is now encrypted and unreadable by potential attackers. Verify this by repeating the three attacks. The application is now accessible at https://localhost:8443. When accessing the app, your browser should issue a security warning about the used TLS certificate. Browsers do only trust certificates which have been signed by a trusted central authority (CA). These CSa are hardcoded into the browser. Since our key is not signed by any CA, but by itself, the browser does not trust it. This does not make the connection less secure. The browser merely warns you that the origin of the certificate is not known. It could potentially be a faked website. When using HTTPS in production, you can let your key be signed by a central authority in order to make your certificate trustworthy. Read more over CAs here.

In Wireshark, instead of filtering by http, change the filter setting to tls. You can now see all encrypted data packets. Since wireshark (the potential attacker) does not know the encryption key, the contents of all data packets can not be read.

Exercises

Reading Data

Q: How is the password presented? (format: ....: "password" = ...)

Q: Follow the TCP trace of the request, what is the class of the button with value="Logout"? (format:"... ...-.......")

> Hint: UmlnaHQtY2xpY2sgb24gdGhlIHBhY2thZ2UgPiBGb2xsb3cgPiBUQ1Agc3RyZWFtLg==

Session Hijacking

Q: How is the session cookie presented? (format: ....: JSESSIONID=*)

Q: What is the max age of the cookie?

> Hint: VXNlIGRldmVsb3BlciB0b29scywgY2xpY2sgb24gdGhlIGNvb2tpZSBhbmQgZXhwbG9yZSB0aGUgYWRkaXRpb25hbCBpbmZv

Q: What happens with the unauthorized session when you logout in the hijacked browser?

Stealing Private Data

Q: What is the key of the form item?

Q: What is the protocol used to communicate the secret?

Q: What is the HTTP method used to communicate the secret?

Secured Connection

Q: What country code is requested to create the keypair? (format: ...-.....)

Q: How many days is the generated key valid?

Q: What is the Signature Algorithm of the certificate? (format: ...-... ....)

> Hint:
> R28gdG8gdGhlIGJyb3dzZXIsIGluIHRoZSBhZGRyZXNzIGJhciBjbGljayBvbiB0aGUgbG9jay4gVmlldyB0aGUgY2VydGlmaWNhdGUu

Q: Redo the attacks and read the data. What is the protocol now?

Additional material

- Certifate Authority
- TLS vs SSL
- FTP