Exercise 2:
Sniffing

Contents

1 Tasks

1.1 Identifying and Exploiting the Vulnerability

The exercise description suggests that there is a vulnerability in the implementation of the SSL library. Also, in the hint, there is a link for a file that we need to download, named **heartbleed.pcapng**.

By analyzing the **heartbleed.pcapng** file, we can also see that although the client sends 9 cipher suites that provide *Perfect Forward Secrecy*, the server still chooses a weak cipher suite, namely *TLS_RSA_WITH_AES_256_CBC_SHA*. This is important, because even if we are able to exploit the server and exfiltrate the private key, we would not be able to decrypt the traffic if *Perfect Forward Secrecy* is enabled.

Based on this information, we can guess that the vulnerability is, in fact, the popular bug named *Heartbleed*. We can quickly check if our hypothesis is correct by utilizing *nmap* [1]:

```
1 nmap -sV --script=ssl-heartbleed -p443 10.0.23.19
```

Listing 1: Check if the server is vulnerable to the Heartbleed bug

The results are positive and the server is indeed vulnerable to this particular bug. But in order make use of it, we would need to write our own exploit (as suggested in the exercise).

By analyzing the packet structure of a *Heartbleed* exploit on *Github* [2], as well as by analyzing one *ClientHello* and one *ServerHello* packet from the **heartbleed.pcapng** file, we can make a few tweaks in order to minimize the packet size.

1.1.1 ClientHello Packet

The new *ClientHello* packet will contain the following info:

- **Record Header**
  - *Packet Type (1 byte)* - in this case, a *Handshake*, which has the value of $0x16$ (22)
  - *TLS Version (2 bytes)* - we will use TLSv1.2, which has the value of $0x0303$
  - *Packet Length (2 bytes)* - this will be the length of the whole packet minus five, so we will set it to $0x0000$ at the moment
- **Handshake Header**
  - *Handshake Type (1 byte)* - a *ClientHello* type, which has the value of $0x01$
  - *Handshake Length (3 bytes)* - this will be the length of the whole packet minus nine, so we will set it to $0x000000$ at the moment
- **Client Version**
  - *TLS Version (2 bytes)* - it will once again take the value of $0x0303$, meaning TLSv1.2
- **Client Random**
  - *Random (32 bytes)* - it can contain any value, but we will set it to values from $0x00$ to $0x1f$
- **Session ID**

- *Session ID Length (1 byte)* - because no session ID is needed, we will set it to $0x00$

- **Cipher Suites**
  - *Cipher Suites Length (2 bytes)* - based on the *ServerHello* packet in *Wireshark*, the used cipher suite is already known, so we can therefore set the length of the cipher suites to $0x02$
  - *Cipher Suite (2 bytes)* - we will use the cipher suite *TLS_RSA_WITH_AES_256_CBC_SHA* ($0x0035$), which we know the server supports

- **Compression Methods**
  - *Compression Methods Length (1 byte)* - again, based on the *ServerHello* packet in *Wireshark*, we can set the compression methods length to $0x01$
  - *Compression Method (1 byte)* - we will also set the compression method to $0x00$, which means no compression

- **Extensions**
  - *Extensions Length (2 bytes)* - because we will only be using one extension, we will set the extension length to the length of the *heartbeat* extension, which is $0x05$
  - *Extension (5 bytes)* - we will use the *heartbeat* extension, which will allow us to send heartbeat packets later on

We can now sum the number of total bytes, which is 57, and replace both the *Packet Length* and the *Handshake Length*:

$$\begin{aligned} PacketLength: &\quad 57 - 5 = 52 = 0x34 \\ HanshakeLength: &\quad 57 - 9 = 48 = 0x30 \end{aligned}$$

1.1.2 Heartbeat Packet

The next thing that we need to do is construct the *Heartbeat* packet. This is a lot easier to do because it only contains a total of 8 bytes split into two parts:

- **Record Header**
  - *Packet Type (1 byte)* - in our case, a *Heartbeat*, which has the value of $0x18$ (24)
  - *TLS Version (2 bytes)* - as in the newly constructed *ClientHello* packet, we will use TLSv1.2 ($0x0303$)
  - *Packet Length (2 bytes)* - because we won't provide any payload (see *Heartbeat Message* below), we will set this to $0x0003$

- **Heartbeat Message**
  - *Heartbeat Type (1 byte)* - we will be sending a request, so this will take the value of $0x01$
  - *Payload Length (2 bytes)* - we will set this the maximum value allowed, which is $16,384 = 0x4000$

The *Payload Length* with the value of $16,384$ will allow us to exploit the *Heartbleed* bug. Because we are not supplying any payload, the server will return $16,384$ bytes data from some random place in the memory. The data can include credentials, private keys, etc.

1.1.3 Exploit

The last thing that we need to do is connect to the IP address given in the example (**10.0.23.19**) on port **443** by using the *socket* library in Python, send the *ClientHello* packet in order to initiate a handshake, and send two *Heartbeat* packets after the handshake in order to exfiltrate the private key. We can then extract the private key by using a simple regular expression.

The full, working code can be found inside the files **heartbleed.py** and **tls.py**. The **heartbleed.py** file is used to create an instance of the class *TLSConnection*, call its methods and extract the private key, while the **tls.py** file contains the *ClientHello* and *Heartbeat* packets, together with the *TLSConnection* class.

1.2 Decrypting the HTTPS Traffic

After successfully exfiltrating the private key, it will automatically be saved into a file named **private.key**. What we now need to do is open the **heartbleed.pcapng** file in Wireshark in order to decrypt the HTTPS traffic.

After opening the file, we can click on *Edit* → *Preferences*, click on the arrow left of *Protocols* in order to expand it and select *TLS*. We can then click on *Edit* RSA keys list, click on the green plus and in the *Key File* column select the file **private.key**. We can then click on all *OK* buttons provided and the traffic will be decrypted.

1.3 Finding the Login Credentials

In order to easily find the login credentials, we can use the following filter in Wireshark:

$$http.request.method == POST$$

If we select the top packet and expand the *HTML Form URL Encoded*, we will see both the username (**d4rkh4xx0r**) and the password (**Y0uW1llN3v3rG3tM3**).

1.4 Reading the Emails

We can navigate to https://10.0.23.19/squirrelmail, enter the found credentials and we will find an email with the following content:

```
Wer routet so spaet durch Nacht und Wind?
Es ist der Router, er routet geschwind!
Bald routet er hier, bald routet er dort
Jedoch die Pakete, sie kommen nicht fort.
...
```

Listing 2: Content of email with subject *Remember this!*

1.5 Mitigation

In order to prevent the decryption of past TLS packets, *Perfect Forward Secrecy* must be enabled. It should once again be noted that the cipher suite *TLS_RSA_WITH_AES_256_CBC_SHA*, chosen by the server, does not support *Perfect Forward Secrecy* [3].

2 Requirements

The one and only requirement is *Wireshark*. All libraries used in the Python code should already be installed on *Ubuntu 20.04 LTS*.

In order to install *Wireshark*, execute the following command:

```
sudo apt install -y wireshark-qt
```

Listing 3: Install required packages

References

1. https://hakin9.org/detecting-and-exploiting-the-openssl-heartbleed-vulnerability/
2. https://gist.github.com/eelsivart/10174134
3. https://ciphersuite.info/cs/TLS_RSA_WITH_AES_256_CBC_SHA