67607 – Exercise 3 – HTTP Attacks

Instructed by Amit Klein, written by Agam Ebel

2024-2025

Contents

1 HTTP Response Splitting and HTTP Request Smuggling

This exercise comprises two components: a demonstration of an HTTP Request Smuggling attack and a demonstration of an HTTP Response Splitting attack.

1.1 Our Environment

We will use the same environment for both of the attacks.

The "target" environment consists of:

- A reverse proxy server
- A web server (for HTTP response splitting, it runs a simple application)

The proxy server sits in front of the web server. The web server runs a very simple "application" which is vulnerable to HTTP response splitting. Additionally and independently, the proxy server – web server combination is vulnerable to HTTP request smuggling.

The attacker is deployed by a docker container and will communicate with the proxy server.

1.2 General Information

- To log in to the environment, use username "httpEnv" and password 5260.
- All the containers are connected to a virtual LAN named `http-a-net2`
- The reverse proxy server is an Apache2.0.45 and deployed by a docker container. Its connected to a virtual LAN and its IP address is `192.168.1.202`.
- The web server is an Apache2.0.45 and deployed by a docker container. Its connected to a virtual LAN and its IP address is `192.168.1.201`.
- The attacker is deployed by a docker container. Its connected to a virtual LAN and its IP address is `192.168.1.203`.
- The "legitimate" client is deployed by a docker container. Its connected to a virtual LAN and its IP address is `192.168.1.204`.
- We can send messages to the proxy server (from the containers) by creating a request to `http://`*`proxy-server-IP-address`*`:8080/`*`the-desired-page`*`.`
- **In both of the attacks, use HTTP/1.1.**
- In both of the attacks, make sure your Host header is correct (i.e. a one that matches the attack attempt from the containers).

- **Conduct all testing and development of your attacks exclusively within the containers.** The proxy keys store cached data using the full URL, which includes the value of the Host header and the absolute URL from the request line. Consequently, testing the cache directly from the VM will result in a failed attempt, even if the attack itself is successful.

1.3 Starting the Environment

The environment for the attacks consists of a VM and 4 containers. First, download the environment at the following link.
Note – the environment disk requires at least 30GB free on your computer, make sure to have enough free space to upload the disk.
Start the virtual machine using the instructions for uploading a virtual disk to a VM which we provided in the "Environment Setup" file.

Next, start the containers:

1. Start the Apache2.0.45 web server using `docker start apache2-web-ver2`.
2. Start the Apache2.0.45 proxy server using `docker start p-ws-ver2`.
3. Start the attacker's container using `docker start attacker`, next run `docker exec -it attacker /bin/bash`.
4. Start the "legitimate" client container using `docker start client`, next run `docker exec -it client /bin/bash`.

Note that you can work on your source code file from within the VM (you should place your files under the directory `/home/httpenv/Desktop/attacker` on the VM), and you'll be able to access it in the attacker's container under `/tmp/attacker` (which is mounted to the VM's `/home/httpenv/Desktop/attacker`).

Therefore, it is enough to change the source code itself inside the "attacker" directory on the VM and the changes will also applied in the container. You must compile the source code from inside the container (with the following command: `gcc -Wall -Wextra -Werror` *`attack-name-file.c`* `-o attacker_`*`name-of-the-attack`*) and ensure that there are no compilation warnings, and run the executable from the container. This is how we will compile and run your code ourselves (in our container environment which is practically identical to yours), so this guarantees you're working with the same compilation and runtime environment we use. We will take off points for compilation warnings (and lots of points if it fails to compile or do the job).

2 HTTP Response Splitting

In this section, we will introduce the HTTP response splitting attack.

2.1 Introduction

In this attack, we will exploit a web application that embeds user data in HTTP response headers without sanitizing the data beforehand to poison the proxy server cache. This enables us to poison the cache for the page the `67607.html` and change its content to a simple, static HTML page that includes the submitters' ID (the one who submits the file in Moodle).

2.2 General Information

- The proxy hostname in the VM is "localhost" and is available locally in the host machine (the VM) using port 4244.
- There are some web pages on the web server. The main 2 we need for this attack are `/67607.html` and `/cgi-bin/course_selector`.
- Asking the proxy server for some web pages (from the attacker's container) can be done in the following way: `http://`*`proxy-server-ip-address`*`:8080/`*`the-desired-page`* (Notice that for asking a request to the proxy server from the VM we need to use "localhost" instead of the proxy server IP and port 4244 instead of port 8080).
- The web server can be accessible from the VM by using the hostname `localhost` and port 4242.
- **The body of the poisoned page should be in the following format:** `<HTML>`*`submitters'-id`*`</HTML>` (without spaces between the HTML tags and the ID).
- **In contrast to what appears in the paper, trying to perform cache invalidation using `Pragma: no-cache` does not work. Meaning, we cannot invalidate the cache using an HTTP request, so make sure to clear the proxy server cache before trying your attack attempt.**

Access to the web and proxy servers from the VM is enabled solely to facilitate a better understanding of the web pages and the overall setup. However, all attack development and testing must be conducted exclusively from the containers!

2.2.1 Notes About the HTTP Protocol

HTTP (and HTTPS) is the protocol used to send requests to a web server. In this part, we will briefly introduce HTTP requests and responses.

1. HTTP Request. This is a message a client sends to a web server to request some resource (like a web page). It has 5 main parts, The HTTP (request) method, the resource location or path on the server, the HTTP version (for example, HTTP/1.1), the headers, and an optional body.

2. HTTP Response. This is the server response to the HTTP request, it contains information about the request and most of the time also the request resource or data. It includes 3 main parts: the status line (with the response HTTP version, a numeric status code, and a textual status message) the headers, and an optional response body.

There are some common HTTP Methods. In the following attack, we will use the GET method (to retrieve data from the server) and the POST method (to send data to the server for processing).

for more information about HTTP requests, responses, and the HTTP protocol, please see the TryHackMe tutorial (which also contains practice exercises).

2.2.2 HTTP Response Splitting Attack Reminders

In this attack, we have three components: a web server, a proxy server, and an attacker. To make the attack possible, we need to have a vulnerable web application on the web server, this application embeds user data in HTTP response headers without sanitizing the data beforehand.

We can exploit this application to poison the proxy server cache in the following way:

- Send a request to the proxy which is later interpreted as 2 responses from the web server. We can do it by sending an HTTP response-splitter message (with our malicious payload in a URL parameter which is later embedded in a response header).
- Next, send an innocent request for the page we want to poison in the proxy server cache.

Please note, that both requests have to be sent on the same TCP connection to the web server that runs the vulnerable application.

The attacker's goal is to poison the proxy server cache, using a response-splitting message. That way, as long as the poisonous record in the cache is still valid, anyone who requests the poisoned page (URL) from the proxy will get the attacker's malicious response instead of the original page.

There are 3 ways to handle TCP streams: the *message boundary* approach, the *buffer boundary* approach, and the *packet boundary* approach. Each proxy server uses one of those 3 approaches. Apache2.0.45 (our reverse proxy server) uses the *message boundary* approach. In this approach, the second message is assumed to start exactly where the first message ended.

2.3 Tips and Tricks

- To listen to the traffic on our virtual LAN with Wireshark from the VM, we first find the ID of our docker network using `docker network inspect http-a-net2`, check what is the ID value, and choose the `br-`*`http-a-net2 network ID`* interface to listen on.

- For analyzing our attack attempt and debugging, we recommend recording the network traffic between the proxy server and the attacker's container as well as the network traffic between the proxy server and the web server. You may use Wireshark's "display filters" to analyze the traffic data more easily.

- Note that if the data is already cached in the proxy server, the proxy server will not forward it to the web server, thus if you expect the proxy to forward the message to the web server, but you do not observe the HTTP request from the proxy, it may be the case that the resource is already cached by the proxy server and is served from the cache.

- Check the proxy and web server logs, they have some useful information. For example – the log data can indicate which resources are cached, if the message was forwarded to the web server, and if something went wrong because we had a bug in the request. We can access the logs in the following path in the containers `/usr/local/apache2/logs`

- Make sure to clean the proxy cache between one attack attempt and another. We can do it by deleting all the files in the following directory in the proxy container: `/usr/local/apache2/proxy`

- Notice that the second request should arrive immediately after the first. We recommend checking that the TCP connection we send our malicious HTTP requests to the proxy server on remains open throughout sending these requests, else – the attack may fail due to timing-related issues.

- Make sure that the `Last-Modified` HTTP response header is sent in the poisoned response (that is later cached), designates a date newer than that of the genuine page, and older than the time of the attack.

- Ensure that the value of the `Last-Modified` HTTP response header is set to a date close but (a bit) later than the actual `Last-Modified` date of the genuine page.

- To see how `course_selector` page works, you can navigate to the `/doomle.html` page (i.e. request it from the proxy using `http://`*`proxy-ip-address`*`:8080/doomle.html` from the containers, or from the VM by using `http://localhost:4244/doomle.html`), which contains a form that invokes `/cgi-bin/course_selector`. Fill in a value, submit the form, and see what parameters the `course_selector` page gets.

- Do not use the browser for building and verifying your payload, as it may send other requests which may interfere with the attack sequence.
- Make sure to send only a single splitter-request per experiment (to avoid cases of a 502 response from the proxy server).
- It is highly recommended to restart the proxy between experiments, you can do it using `docker restart p-ws-ver2`.
- You can check if your attack succeeds by sending another request (not including the requests you need for the attack) to the proxy for the page `/67607.html`. A successful attack should end with the injected data served from the proxy. Send this request from the client container.

2.4 Resources

1. "Divide and Conquer, HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics"
2. `https://medium.com/@S3Curiosity/http-requests-and-responses-a-beginners-guide-fc215b9ea741`
3. `https://tryhackme.com/r/room/httpindetail`

3 HTTP Request Smuggling

In this section, we will introduce the HTTP request smuggling attack, this is the second attack we will demonstrate in this exercise.

3.1 Introduction

Similar to the HTTP Response Splitting attack, in this attack we also poison the proxy cache, although we do it differently. Now – instead of sending an additional response, we will send a hidden request inside our poisoned message.

Your goal is to poison the proxy cache for the page `/page_to_poison.html` with the content of the page `/poison.html` using HTTP request smuggling. That is, the goal is to make requests to the proxy for `/page_to_poison.html` to receive the page data from `/poison.html`.

3.2 General Information

- The environment for this attack is the same one as the one of the HTTP response splitting attack.
- To recall:

  - The reverse proxy server is an Apache2.0.45 and deployed by a docker container. Its IP address is `192.168.1.202`.
  - The web server is an Apache2.0.45 and deployed by a docker container. Its IP address is `192.168.1.201`.
  - The attacker is deployed by a docker container. Its IP address is `192.168.1.203`.
  - The proxy hostname in the VM is "localhost" and is available locally in the host machine (the VM) using port 4244.
  - Asking the proxy server for some web pages (from the attacker's container) can be done in the following way: `http://`*`proxy-server-ip-address`*`:8080/`*`the-desired-page`* (Notice that for asking a request to the proxy server from the VM we need to use "localhost" instead of the proxy server IP and using port 4244 instead of port 8080).
  - The web server can be accessible from the VM by using the hostname `localhost` and port `4242`.

- There are some web pages on the web server. The 3 pages we need for this attack are `/doomle.html` (an "innocent bystander" page you can use to initiate the attack), `/page_to_poison.html` and `/poison.html`.

3.2.1 Notes about Content-Length and Transfer-Encoding

There are two mechanisms in HTTP for designating where the message body ends: `Content-Length` and `Transfer-Encoding`.

If the message body size is fully known beforehand, the sender can use the `Content-Length` header and specify the size of the data that is sent in the body of the message.
However, sometimes the body length is not known at the time the headers are sent, e.g. when a web application generates dynamic pages on the fly. In this case, the sender can use the `Transfer-Encoding: chunked` header, sending chunks of the body as they arrive. Each chunk is prefixed with its length encoded in hexadecimal notation ASCII characters followed by CR+LF. The chunk's data is then terminated with another CR+LF. The end of the body is marked by a chunk of length 0, signaled by the character 0 followed by CR+LF+CR+LF.

3.2.2 HTTP Request Smuggling Reminders

This attack is based on the observation that whenever HTTP requests that were originated from the client pass through more than one entity that parses them, those entities may process the requests differently, which can facilitate an HTTP Request Smuggling (HRS) attack.

The attack exploits differences in the way 2 devices parse HTTP requests. To recall, in this attack, each one of the devices interprets the data it gets in a different way, which leads to a mismatch in matching the request to the response in the proxy server, and leads to the

caching of wrong (i.e. malicious) data for a request.

There are many ways to exploit those differences in the HTTP request parsing (and some methods may not work for all types of servers), based on multiple/combinations of `Transfer-Encoding` and/or `Content-Length` headers. For example (involving ancient servers, but still), for a POST request with a double `Content-Length` header, SunONE server (6.1 SP1) interprets the request according to the first `Content-Length` header and SunONE proxy (3.6 SP4) interprets the request according to the last `Content-Length` header.

Other methods to perform an HTTP request smuggling can use a combination of `Transfer-Encoding: chunked` and `Content-Length` headers. As mentioned in the paper, we can perform HTTP request smuggling when using those headers when the proxy server and the web server are Apache 2.0.45.

As stated in the paper: "Apache 2.0.45 was found to react interestingly to this anomaly. A request which arrives with both headers [`Transfer-Encoding: chunked` and `Content-Length`] is assumed to have a chunked-encoded body. This body is read in full by Apache, which reassembles it into regular (non-chunked) request. For some reason, Apache does not add its own Content-Length header, nor does it replace an existing Content-Length header (if there is one). The net result is that the request is forwarded with the original Content-Length header (if there is one), without the "Transfer-Encoding: chunked" header, and with a body which is the aggregation of all the body chunks in the original request."

3.3 Tips and Tricks

- For different web and proxy servers, we might need to use some different combinations of `Transfer-Encoding: chunked` and `Content-Length` headers. Pay attention to the technical details and nuances involved in the attack technique you use, specifically how the proxy server and the web server process the requests, in order to perform the attack successfully. You can find more details in subsubsection 3.2.2
- Ensure that the size specified in the "Content-Length" header of the **smuggled request** (the second request the web server sees) corresponds to the total size of the entire smuggled request (if you use the POST method for the smuggled request).
- You can check if your attack succeeds by sending another request (not including the requests you need for the attack) to the proxy for the page `/page_to_poison.html`. A successful attack should end with the data from `/poison.html` served from the proxy. Send this request from the client container.

- Do not forget to clean the proxy server cache between one attack attempt and another. We can do it by deleting all files in the following path in the proxy container: `/usr/local/apache2/proxy`.

3.4 Resources

- "HTTP Request Smuggling" – `https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf`.

4 What to Submit

Please follow these instructions carefully to ensure full credit for your submission. Failure to meet any of these requirements will result in a deduction of points.

4.1 Important Remarks

- Using the `libcurl` library is not allowed.
- When compiling your attack files, use `gcc -Wall -Wextra -Werror` *`program-name.c`* `-o` *`desired-compiled-file-name`*
  Compile your source code from within the attacker's container and make sure there are no compilation warnings! We will take points off for any compilation warnings.
- When running the compiled code, run it without any command line arguments (for each of your files).
- Make sure your code does not print anything to the screen before submitting your attack attempt.
- The exit code from your code should be 0 (for both of the attacks).
- Make sure that the body of the poisoned page (in the HTTP response splitting attack) is in the following format: `<HTML>`*`submitters'-id`*`</HTML>` (without spaces between the HTML tags and the ID).
- Please clean up your code before submission – remove redundant code, add useful comments, use reasonable names for variables and functions, etc. – but please double-check that everything still works afterwards!

4.2 Submission

The submission is done via Moodle in pairs. Please submit a **single zip file named "ex3.zip"** which contains the following files:

1. Your HTTP response splitting attack code (in `C`). The file should be named "ex3_splitting.c".
   When compiling the file, use `gcc -Wall -Wextra -Werror` *`program-name`*`.c -o` *`desired-compiled-file-name`*
   Compile your source code from within the attacker's container, and make sure there are no compilation warnings! We will take points off for any compilation warnings.

2. Your HTTP request smuggling attack code (in `C`). The file should be named "ex3_smuggling.c".
   When compiling the file, use `gcc -Wall -Wextra -Werror` *`program-name`*`.c -o` *`desired-compiled-file-name`*
   Compile your source code from within the attacker's container, and make sure there are no compilation warnings! We will take points off for any compilation warnings.

3. A file containing a short explanation of each attack (a short explanation for the HTTP request smuggling and another short explanation for the HTTP response splitting attack).
   The file should be named "explanation.txt".

4. A TXT file contains a single line with the submitters' IDs (separated by a comma. i.e. 123456789,987654321). If you submit the exercise alone (after getting approval from the course staff), enter your ID only (without a comma, i.e. 123456789).
   The file should be named "readme.txt".

5. Please pay attention to file names (typos...) and do not submit any folders or extra files – just a "flat" zip file with those four files only.