ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

CDS201: Έλεγχος Εισβολών Δικτύων και Συστημάτων (Penetration Testing)

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

Web Application Penetration Testing

- Web Technologies and Standards
- Basic Web Attacks and exploitation
- Demo with examples
- Chained Attack example
- Putting it all together with a lab

Web Application Technologies

- HTML
- Cascading Style Sheet (CSS) – Not of a concern during pretesting
- JavaScript
  - High level interpreted Programming language primarily used to make an application interactive and dynamic.
  - Widely used in modern applications (Node.js is increasingly popular)
  - JavaScripts can give us interesting insights in behavior on the front end, as well as information about the endpoints an application is using
- Client Side
  - Operations that performed in the browser on the user’s device
  - Presenting data to the user
  - Handling user interactions
- Server Side
  - Operations performed on a web server where the application is hosted
  - Tasks like data storage and retrieval and processing

Web Application Technologies HTTP and DNS

- HTTP:
  - Request – Response Protocol : the "dialogue" between the client and the server. The client sends a request to the server and it responds.
  - Request Methods:
    - GET : Fetch the resource. The most common request that browsers send
    - POST : Submit data to be processed to a server. Mainly used in submitting form data or uploading files
    - PUT: Update or create a resource on the server.. Like update the email value of an account
    - DELETE: remove a resource from the Server
    - HEAD: like GET but without fetching the response body, just to retrieve the headers of a resource.
    - PATCH, TRACE, OPTIONS, CONNECT…
  - HTTP Response Codes: HTTP responses consists of a 3-digit code and a reason phrase
    - 1xx – Informational
    - 2xx – Success
    - 3xx – Redirection
    - 4xx - Client Error
    - 5xx – Server Error

  - HTTP is stateless. Each request is unaware of previous requests, so cookies are used

Web Application Technologies HTTP and DNS

DNS: Hierarchical and decentralized system for names in any resource used to the Internet

- Translates human readable domain names to IP addresses
- Hierarchical:
  - Subdomain[.]Second Level Domain[.]Top level domain (TLD)
- DNS Records types:
  - A (Address) Record: Maps a domain name to an IPv4 address.
    Example: example.com IN A 192.0.2.1
  - 2. AAAA (IPv6 Address) Record: Maps a domain name to an IPv6 address.
    Example: example.com IN AAAA 2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - 3. CNAME (Canonical Name) Record: Alias of one name to another: the DNS lookup will continue by retrying the lookup with the new name.
    Example: www.example.com IN CNAME example.com
  - 4. MX (Mail Exchange) Record: Directs email to a mail server.
    Example: example.com IN MX 10 mail.example.com
  - 5. TXT (Text) Record: Holds arbitrary text, often used for verification purposes and to hold SPF (Sender Policy Framework) data.
    Example: example.com IN TXT "v=spf1 include:example.com ~all“
  - ...

Web Application Security

Ensuring the security aspect of web applications is very important :

- Data Protection (Leaks, Breaches….)
- Maintain our trust with customers (is there any if we are breached several times?)
- Legal and Regulatory Compliance
  - Failing application security may lead to no compliance and legal consequences
  - External Pentest and Web application pentests have the most compliance driven aspects
- Business Continuity with no disruptions
  - Sometimes attacks can lead to downtimes, defacements or event deletion of files leading ot outages
- Avoid financial loss
  - Outages, disruptions and downtimes could lead to financial loss
  - Consequences of a breach of clients data could lead to financial damage
- Protecting our brand /reputation
  - Could take years to gain the trust of customers if you are getting breached and this certainly means financial loss

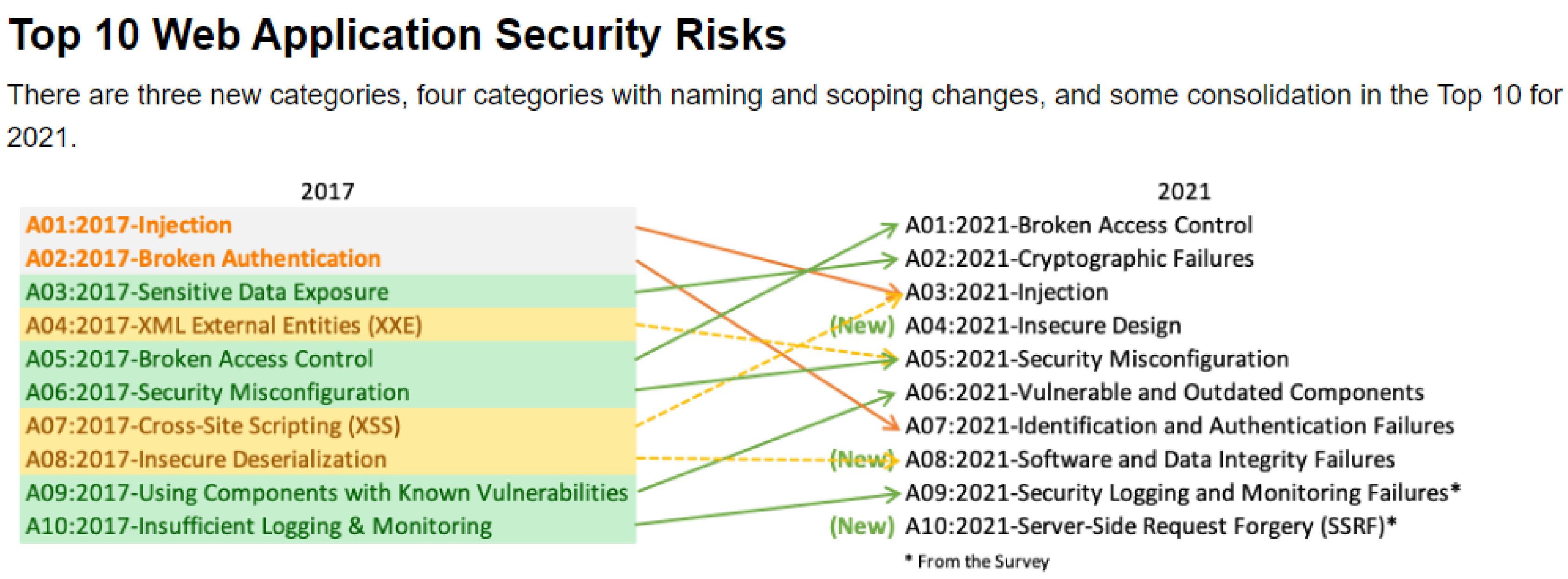
Web Application Security Best Practices

- Patching and Updating regularly
  - Patch known vulnerabilities, keep everything up to date
- Enforce the Least Privilege Principle
  - Give permissions only where they needed
- Secure Coding (Enforce security by design)
- Secure Data Storage (encrypt data at storage and at transit)
- Enforce MFA at authentication mechanisms used
- Logging and Monitoring
- Training the end user
  - Security awareness (phishing emails become more sophisticated using AI)
- ….

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

Web Application Security Standards

- OWASP TOP 10
- SANS TOP 25
    - Built on Common Weakness Enumeration (CWE)



Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.

2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

CWE **Common Weakness Enumeration**
A community-developed list of SW & HW weaknesses that can become vulnerabilities

Web Application Penetration Testing Phases

- **Planning and Preparation (Statement of Work, Rules of Engagement) :**
  - **Define Scope:** Identify the target systems, applications, and network segments.
  - **Set Objectives:** Outline the goals, such as identifying security weaknesses, assessing compliance, or testing incident response.
  - **Gather Documentation:** Obtain necessary documentation like application architecture, data flow diagrams, and security policies.
  - **Legal Authorization:** Ensure that proper permissions and legal agreements are in place to avoid legal repercussions.
  - **Payment Terms**
  - **Rules of Engagement** (Testing Team, PoCs, Testing Dates, Status updates…)

Web Application Penetration Testing Phases

- **As a pentester that has to do the job:**
  - Wait for the RoE to be signed (do not do anything before that)
  - Verify the scope of the pentest
  - Perform testing
  - Report your findings
  - Communicate everything to the client
  - If necessary, plan and do some retesting

Web Application Penetration Testing Phases

- **Reconnaissance (Information Gathering):**
  - **Passive Reconnaissance:** Collect information without interacting directly with the target. This includes WHOIS lookup, DNS enumeration, examining social media, and gathering data from public sources.
  - **Active Reconnaissance:** Engage with the target to gather more detailed information. This involves pinging the target, port scanning, and banner grabbing.
- Find all the assets (not just scratching the surface)

Web Application Penetration Testing Phases

Reconnaissance (Information Gathering):

- **Fingerprinting Web Technologies**
  - Identify technologies on the website to further look for vulns and exploits
    - https://builtwith.com – site that identifies technologies for a domain
    - Wappalyzer extention – browser extension
    - Kali command line:
      - $ curl –I <website domain name> : may gather extra info
      - $ curl –I –L <website> : follow redirects to go over cloudfronts
      - Nmap: $ nmap –p<port> -A <domain name>

***Focus on the methodology not the tool

Web Application Penetration Testing Phases

Reconnaissance (Information Gathering):

- **Directory enumeration** (Brute Forcing): Guess the directories with tools
  - Utilize wordlists with tools like:
    - ffuf : $ ffuf -w /path/to/wordlist -u https://target/FUZZ (useful flags : --recursive , -fc (filter status codes)
    - dirb : $ dirb <domain name or IP> - will scan with defaults. (-X defines extensions to test for)
    - Dirbuster: dirbuster
    - Gobuster: gobuster dir …

Web Application Penetration Testing Phases

Reconnaissance (Information Gathering):

Subdomain enumeration: Find subdomains that we would like to attack in scope

- Google dorks: site:example.com
- crt.sh
- Subfinder : command line tool
  - $ subfinder –d <domain name>
- Assetfinder: command line tool written in go
  - $ assetfinder <domain> | grep <domain> | sort -u
- Amass : The OWASP Amass Project performs network mapping of attack surfaces and external asset discovery using open source information gathering and active reconnaissance techniques.
  - $ amass enum –d domain.com
- Httpprobe : Probe websites to find if they are live:

  $ cat output_from_previous_tools | grep domain.com | sort –u | httprobe –prefer-https | grep https > domain_sites_alive.txt
- Gowitness: take screenshot of websites:
  - $ gowitness file –f domain_sites_alive.txt –P domain_pics –no-http

Web Application Penetration Testing
Web Proxies

- Tools that can be set up between a browser/mobile application and a back-end server to capture and view all the web requests being sent between both ends, essentially acting as man-in-the-middle (MITM) tools
- limited to, HTTP/80 and HTTPS/443
- Apart from tampering they can be also used for:
  - Web application vulnerability scanning
  - Web fuzzing
  - Web crawling
  - Web application mapping
  - Web request analysis
  - Web configuration testing
  - Code reviews
- Most well known and widely used web proxies:
  - Burp Suite (Burp)
  - OWASP Zed Attack Proxy (ZAP)

1. Install and run Burp Suite
2. Proxy setup
   1. Opend default browser
   2. Configure manual proxy settings (install cert http://burp, install foxyproxy)
3. Intercept requests
   1. To manipulate requests for web vulns (Injections, Authentication)
   2. Intercept Responses before they reach the browser (Proxy->Options)
4. Automatic modification for all requests (Proxy>Options>Match and Replace)
5. Repeating Requests
6. Encoding/Decoding
7. Combined with proxy tools: proxychains config, nmap, Metasploit …)
8. Web fuzzing with Burp Intruder /ZAP fuzzer
9. Vulnerability scanning with Scanner
10. Many Extensions with BAPP store

Web Application Penetration Testing Exploitation

Attacking Authentication and Authorization:

Authentication: Who your are? Your Identity! A very important component in application security

- Interesting to test for because many of the flows are logic based
    - Get through a view of the authentication flow and come up with edge cases to break it
    - Multi step login / MFA: Are MFA tokens weak and bruteforcable? Is it applied to multiple users? Does it expire? Can it be reused? …
- Attacking authentication:
    - Password Based: Brute Force Attacks, Detect logic issues
    - MFA: Phishing attacks to collect both factors

https://portswigger.net/web-security/authentication

https://appsecexplained.gitbook.io/appsecexplained/common-vulns/authentication

OWASP: https://owasp.org/www-community/controls/Multi-Factor_Authentication

Web Application Penetration Testing Exploitation

Attacking Authentication and Authorization:

Authorization: Access Control ? What we are allowed to do

- Vertical Access Controls: Restrict access to functionality to specific users
  - Example: restrict users from accessing administrative control on a Ecommerce platform and editing a product
- Horizontal Controls: Restrict access to resources for a specific user
  - Example: a user can update details on their own accounts but not on any other user's account
- Context dependent access controls: based on the current state.
  - Example: you may not allowed to checkout if you do not have anything in your cart

Broken Authorization: Any user having access to functionality they should not be able to access

- IDOR – Insecure Direct Object Reference
  - access control vulnerability that arises when an application uses user-supplied input to access objects directly
  - one example of many access control implementation mistakes
  - IDOR vulnerabilities are most commonly associated with horizontal privilege escalation
- Attacking authentication:
  - Brute Force parameter values

Web Application Penetration Testing Exploitation

Attacking Authentication and Authorization:

APIs: Application Programming Interface

- More than 80% of internet traffic
- Requests are made to endpoints to return some data that are processed client side
  - Example: a JavaScript will handle to request some data that will update a portion of our page. Not needing to refresh the page.
- Responses of API requests are usually in Jason format, our application takes these data and processes them in any way
- Developers can build applications that use APIs of known Vendors
  - Examples: Google maps API
- Check out this API Swagger example

Web Application Penetration Testing Exploitation

Attacking Authentication and Authorization:

API – JWTs:

- a standardized format for sending cryptographically signed JSON data between systems
- used to send information ("claims") about users as part of authentication, session handling, and access control mechanisms
- all of the data that a server needs is stored client-side within the JWT itself. This makes
- consists of 3 parts: a header, a payload, and a signature separated by a dot
    - Even if the token is unsigned, the payload part must still be terminated with a trailing dot.
- the security of any JWT-based mechanism is heavily reliant on the cryptographic signature because this data can be easily read or modified by anyone with access to the token

JWT Attacks: involve a user sending modified JWTs to the server in order to achieve a malicious goal.

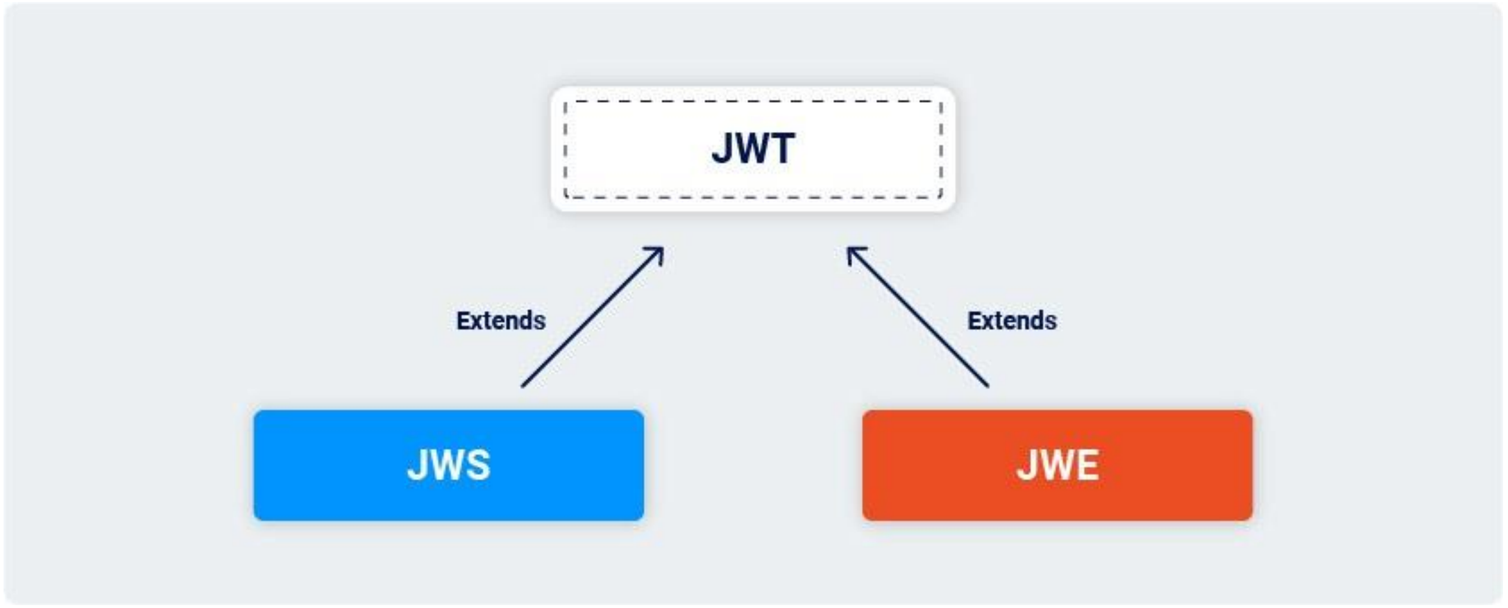
Example: Exploiting flawed JWT signature verification (Accepting arbitrary signatures | Accepting tokens with no signature)

Web Application Penetration Testing Exploitation

Attacking Authentication and Authorization:

API – JWTs:

- JWTs aren't really used as a standalone entity. The JWT spec is extended by both the JSON Web Signature (JWS) and JSON Web Encryption (JWE) specifications
- JWT always mean a JWS token. JWEs are very similar, except that the actual contents of the token are encrypted rather than just encoded



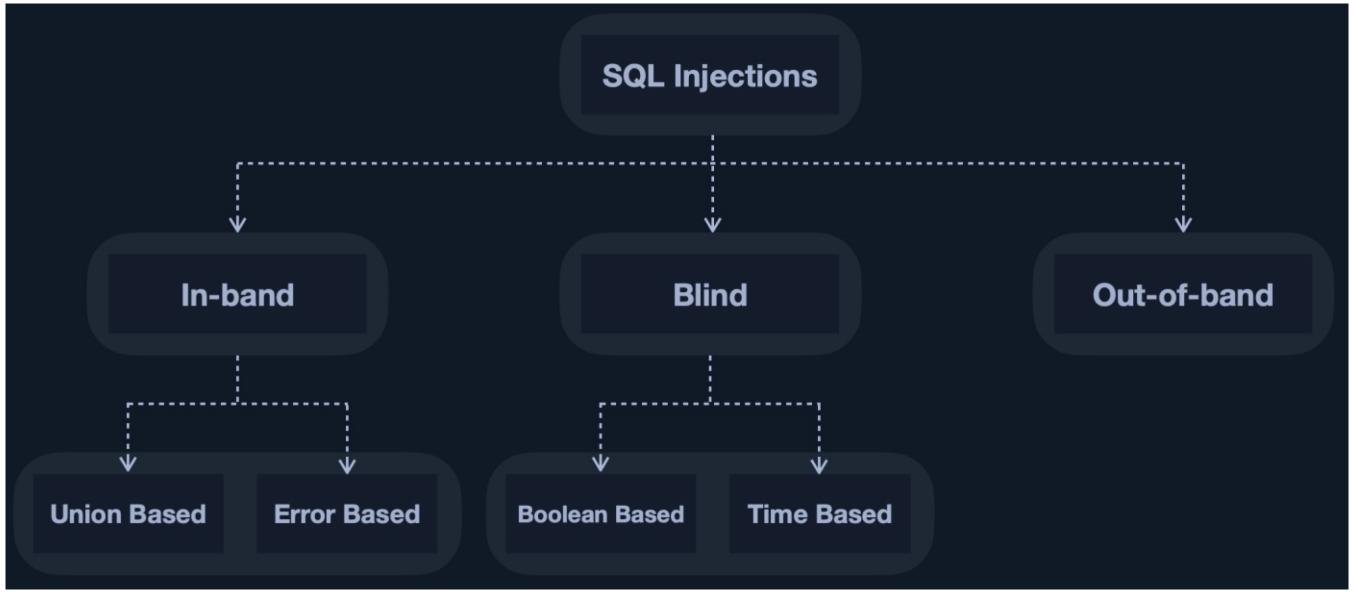
Web Application Penetration Testing Exploitation

Injection Attacks:

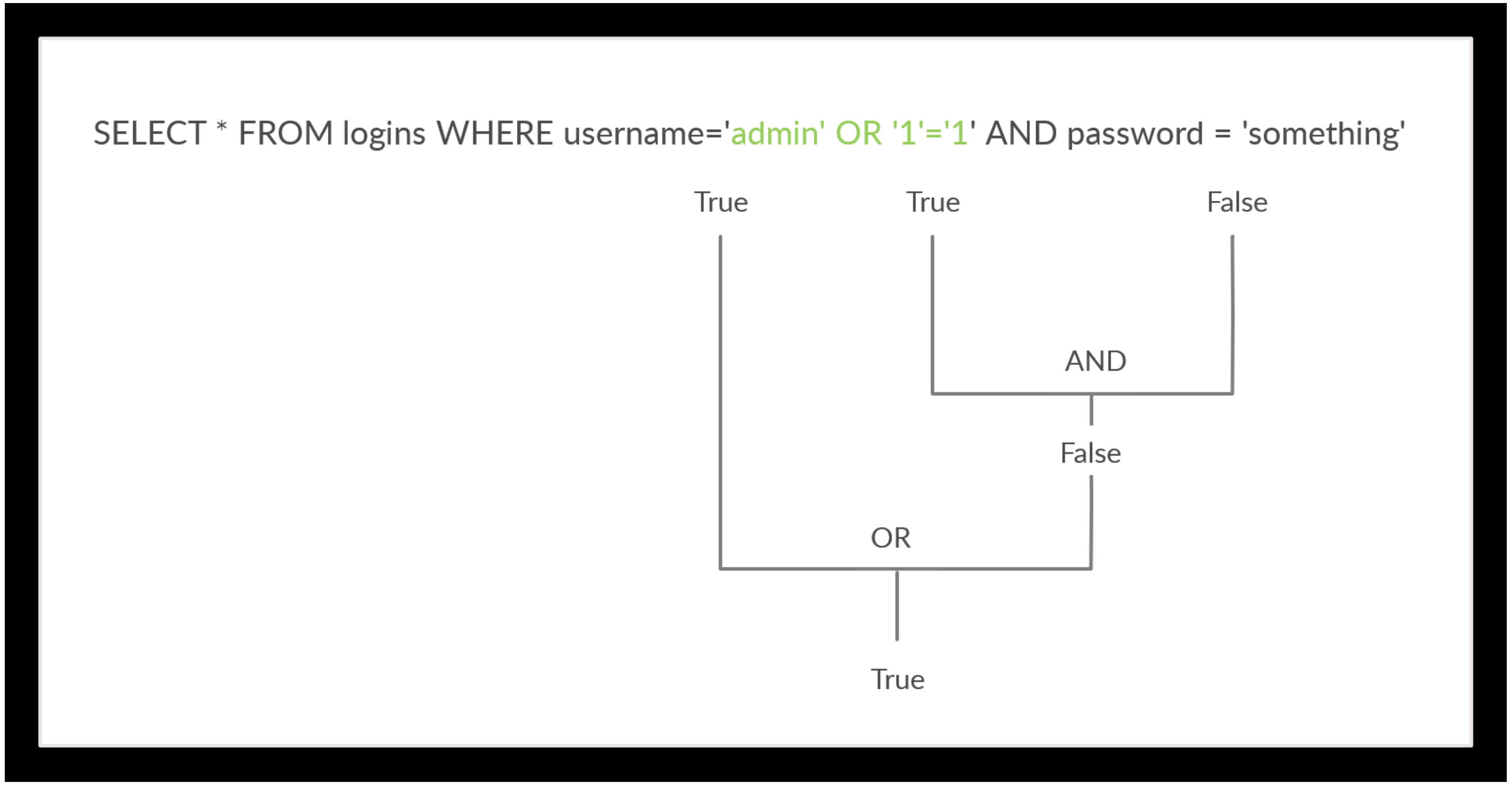
- SQL Injections
- XSS (Cross Site Scripting attacks)
- XXE (XML External Entity)
- File Inclusion (LFI/RFI)
- Command Injection
- Insecure File Uploads
- SSTI (Server Side Template Injection)

Web Application Penetration Testing Injections - SQLi

- SQLi: attacks against relational databases (MySQL, MSSQL)
- NoSQLi: injections against non-relational databases (MongoDB)
- How SQLi works: an attacker interferes with the queries that an application makes to its database. An SQL injection occurs when user-input is inputted into the SQL query string without properly sanitizing or filtering the input



Web Application Penetration Testing Injections - SQLi


```
SELECT * FROM logins WHERE username='admin' OR '1'='1' AND password = 'something'
```

Web Application Penetration Testing Injections - SQLi

- A successful sqli could even lead to server compromisation or backend infrastructure
- How to test for sqli
  - single quote character ‘ (Error-based SQLi – simple)
  - Boolean conditions such as OR 1=1 and OR 1=2 (Error Based SQLi)
  - Trigger time delays within a SQL query, and look for differences in the time taken to respond (Time-based SQLi)
  - Out-of-band application security testing using external servers and channels (Burp Collaborator with Burp Suite Pro)
- Examples:
  - Union Attacks
    - Determine the number of columns like with ORDER BY or UNION SELEC null,null,null--)
    - Fetch other data from the table or other tables

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (not the low-hanging-fruits as common sqli)

- The idea here is that you carry on with the attack and deliver the payload in the application but it is not executed immediately but later on.
- The difference with 1rst order sqli is in the way that the attacker can insert a malicious string and cause the modified code to execute immediately.
- Slightly more difficult to test: requires knowledge of backend ops of the app. There is no application response to discover the vulnerability.
- automated tools are not smart enough to identify the change in application behavior.

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (example)

```
CREATE TABLE USERS ( userId serial PRIMARY KEY, firstName TEXT )
```

Supose we have a SAFE code for receiving firstName from a form:

```
$firstname = someEscapeFunction($_POST["firstName"]);

$SQL = "INSERT INTO USERS (firstname) VALUES ('{$firstName }');";

someConnection->execute($SQL);
```

assuming that someEscapeFunction() does a fine job. So it in not possible to inject SQL. Inserting a payload like *bla'); DELETE FROM USERS; //* would not be successful

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (example)

Suppose somebody in the same system wants to transport firstName from Users to table ORDERS and does something like this:

```
$userid = 42; $SQL = "SELECT firstname FROM USERS WHERE (userId=
{$userid})"; $RS = con->fetchAll($SQL); $firstName = $RS[0]
["firstName"];
```

And then inserts into ORDERS table with :

```
$SQL = "INSERT INTO SOME VALUES ('{$firstName}');";
```

So using the trusted payload *bla'); DELETE FROM USERS; //* would end up executing:

```
INSERT INTO SOME VALUES (' bla'); DELETE FROM USERS; //
```

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

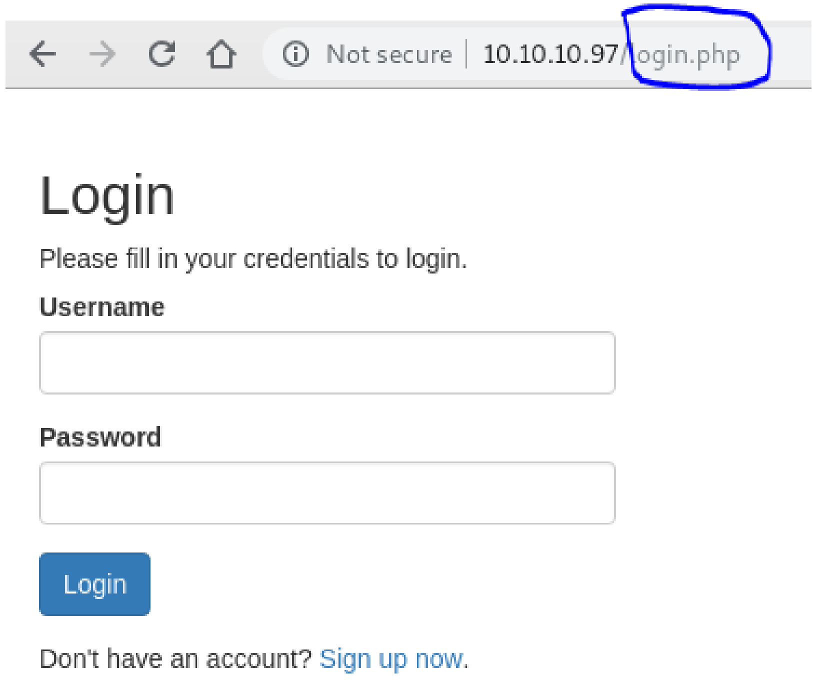
Demo of 2nd order sqli on HTB SecNotes Machine

(https://www.hackthebox.eu/) (Not the intended way!!!)

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

So we have a login with no sql injection flaw in its login params.



Not secure | 10.10.10.97/login.php

Login

Please fill in your credentials to login.

Username

Password

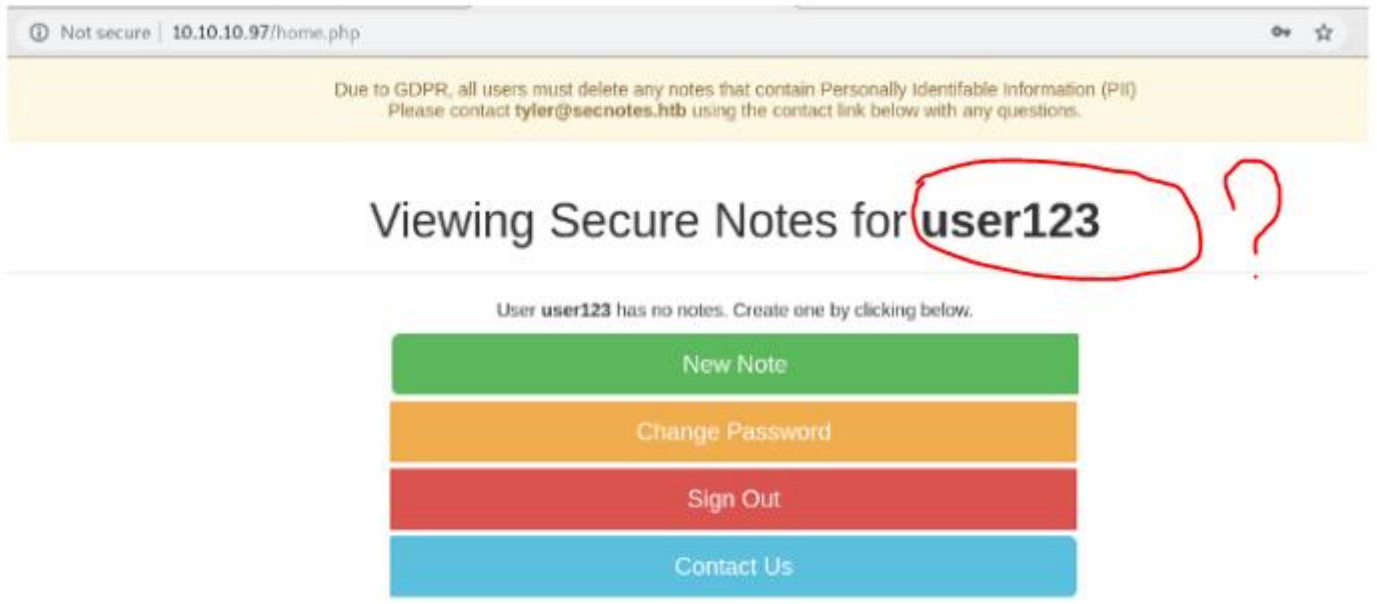
Login

Don't have an account? Sign up now.

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

We can register a user with username user123 and after login we see our notes:



Not secure | 10.10.10.97/home.php

Due to GDPR, all users must delete any notes that contain Personally Identifiable Information (PII)
Please contact **tyler@secnotes.htb** using the contact link below with any questions.

Viewing Secure Notes for **user123** ?

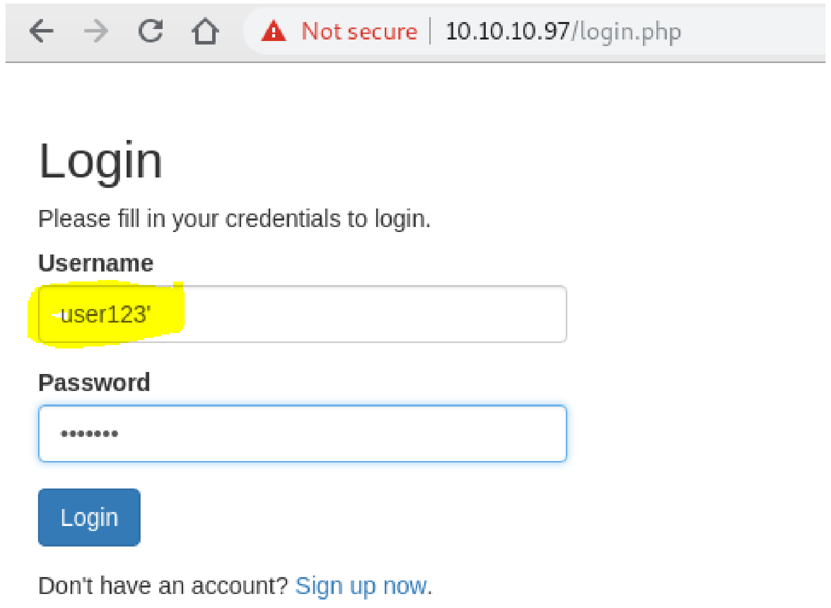
User **user123** has no notes. Create one by clicking below.

We assume the username is used to fetch the notes

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

So … we try to register a user with username user123' hoping we encounter an SQL error but we get a 500 error:


Not secure | 10.10.10.97/login.php

Login

Please fill in your credentials to login.

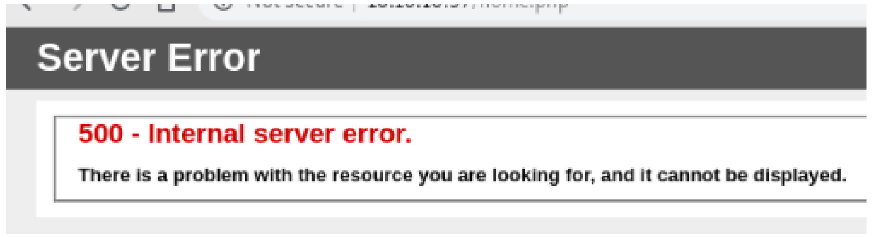
Username

user123'

Password

•••••••

Don't have an account? Sign up now.


Server Error

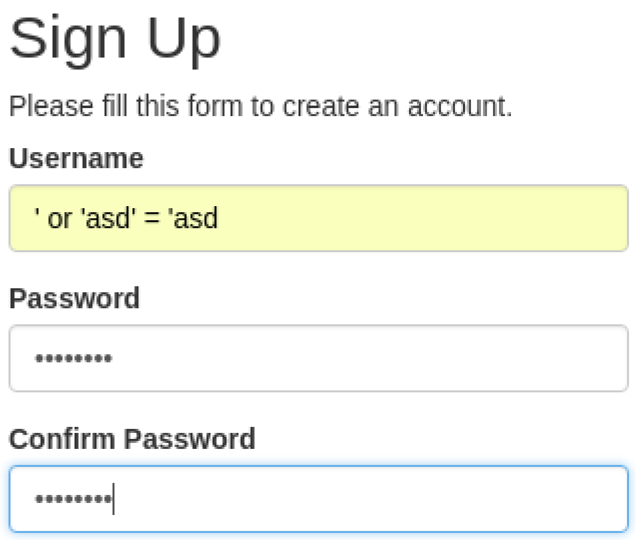
500 - Internal server error.

There is a problem with the resource you are looking for, and it cannot be displayed.

Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

OK!….. Lets create a payload to confirm that the sql command causes the error (username = ‘or ‘asd’ = ‘asd). We first sign up and the login. And …..

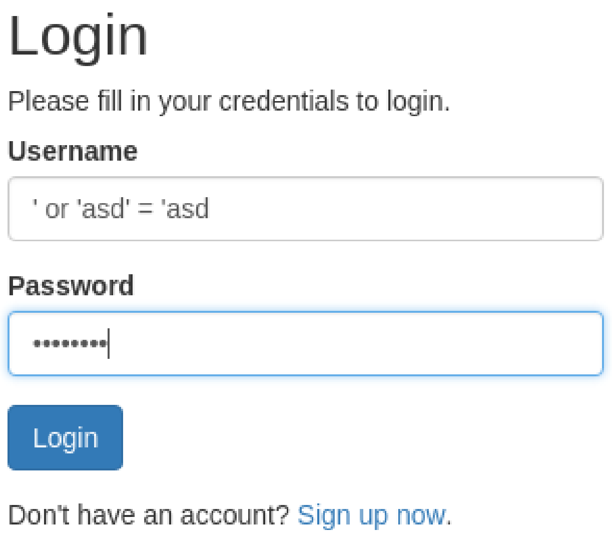

Sign Up

Please fill this form to create an account.

Username: ' or 'asd' = 'asd

Password: •••••••••

Confirm Password: •••••••••


Login

Please fill in your credentials to login.

Username: ' or 'asd' = 'asd

Password: •••••••••

Don't have an account? Sign up now.

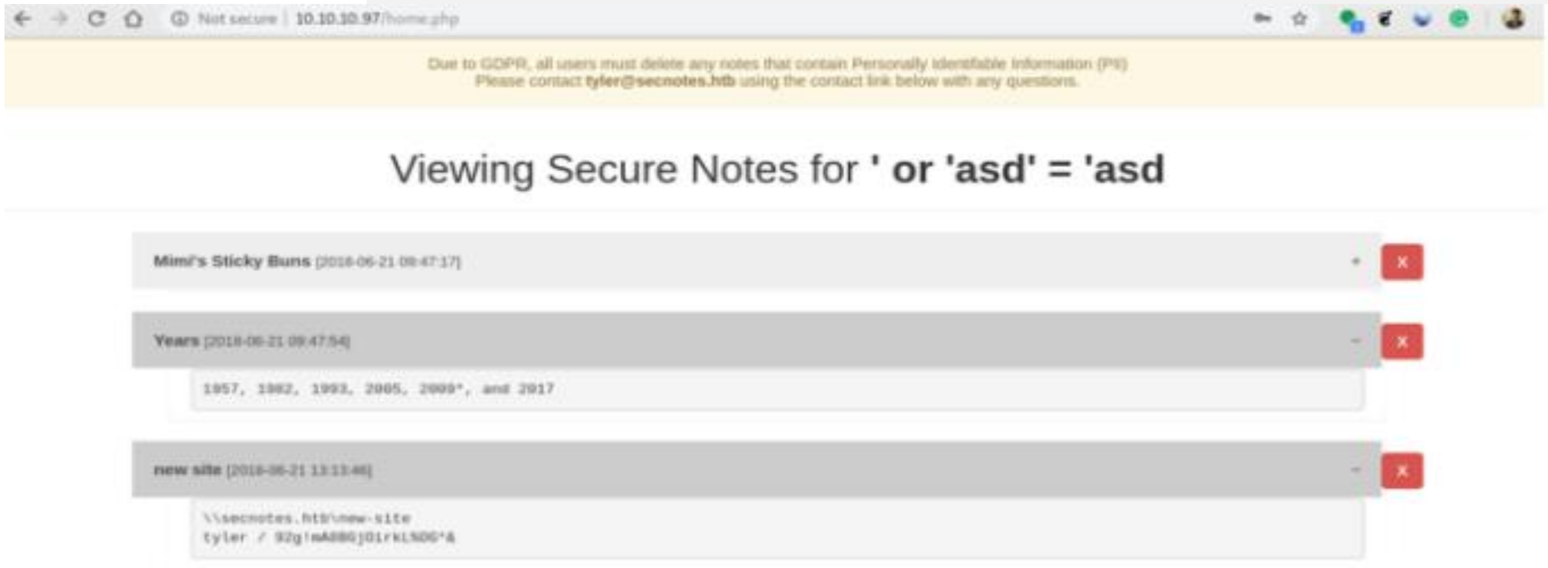
Web Application Penetration Testing Injections - SQLi

2nd order SQLi (HTB example)

And….. Booom!!!

We executed….

```
SELECT * from notes WHERE username = '' or 'asd' = 'asd';
```



Not secure | 10.10.10.97/home.php

Due to GDPR, all users must delete any notes that contain Personally Identifiable Information (PII)
Please contact **tyler@secnotes.htb** using the contact link below with any questions.

Viewing Secure Notes for **' or 'asd' = 'asd**

Mimi's Sticky Buns [2018-06-21 09:47:17]

Years [2018-06-21 09:47:54]

```
1957, 1982, 1993, 2005, 2009*, and 2017
```

new site [2018-06-21 13:13:46]

```
\\secnotes.htb\new-site
tyler / 92g!mA8BGjOirkL%OG*&
```

Figure 8

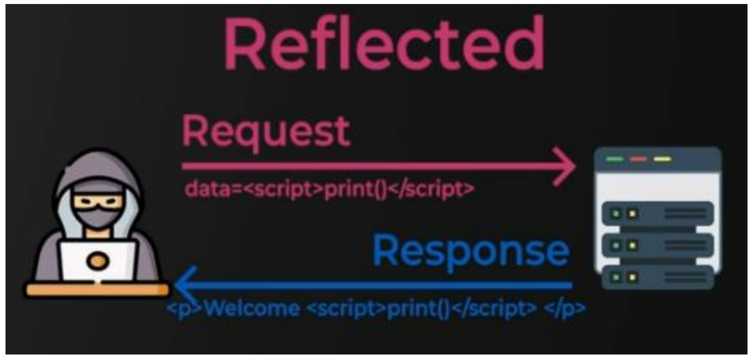
Web Application Penetration Testing Injections - XSS

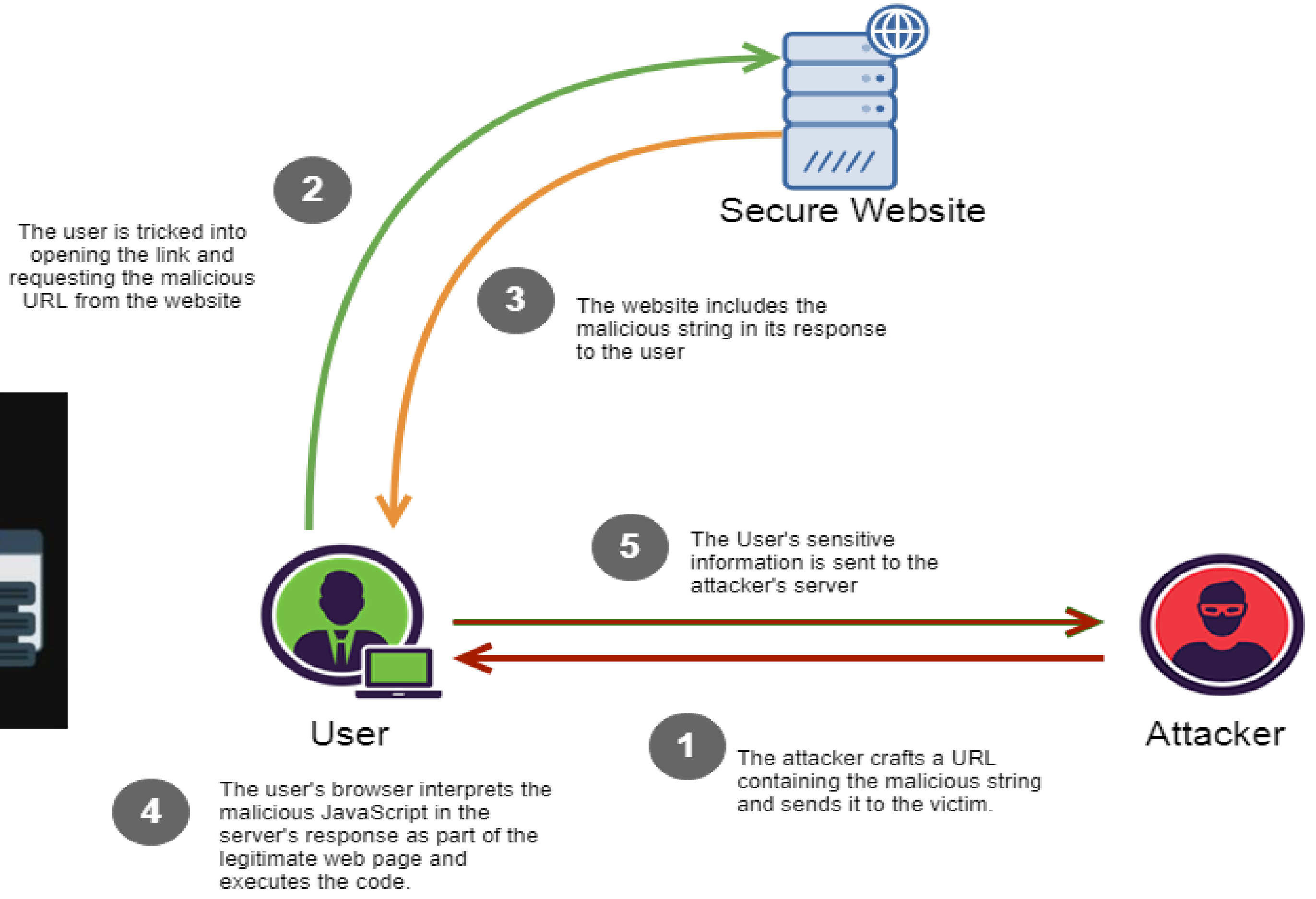
- XSS: Attacks that modify the page rendered by the web server
- It is still found in modern apps. A very flexible vulnerability comb used to chain other issues together
- In a nutshell: It lets us execute JavaScript in a victims browser and often gives control over the application for that user (in his security context).

- May affect security via:
  - Session hijacking attacks (Cookie stealing)
  - Impersonation attacks (attacker can do everything the victim can)

Web Application Penetration Testing Injections – XSS types

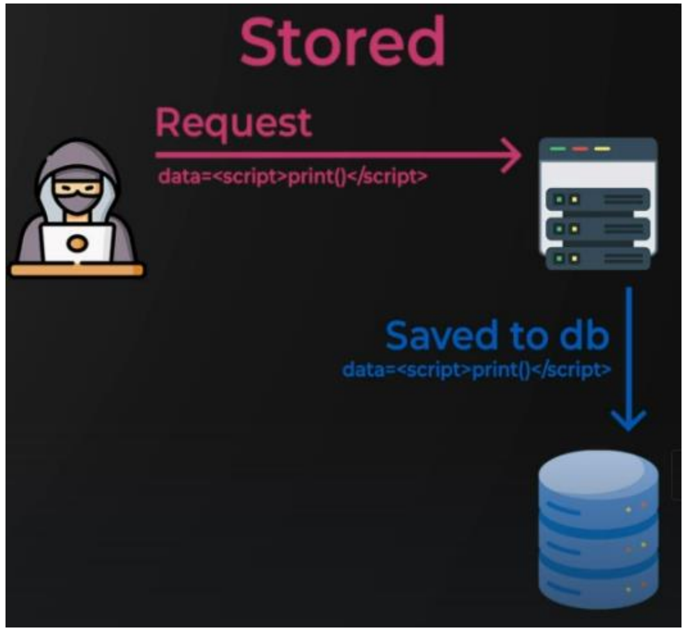
XSS Reflected

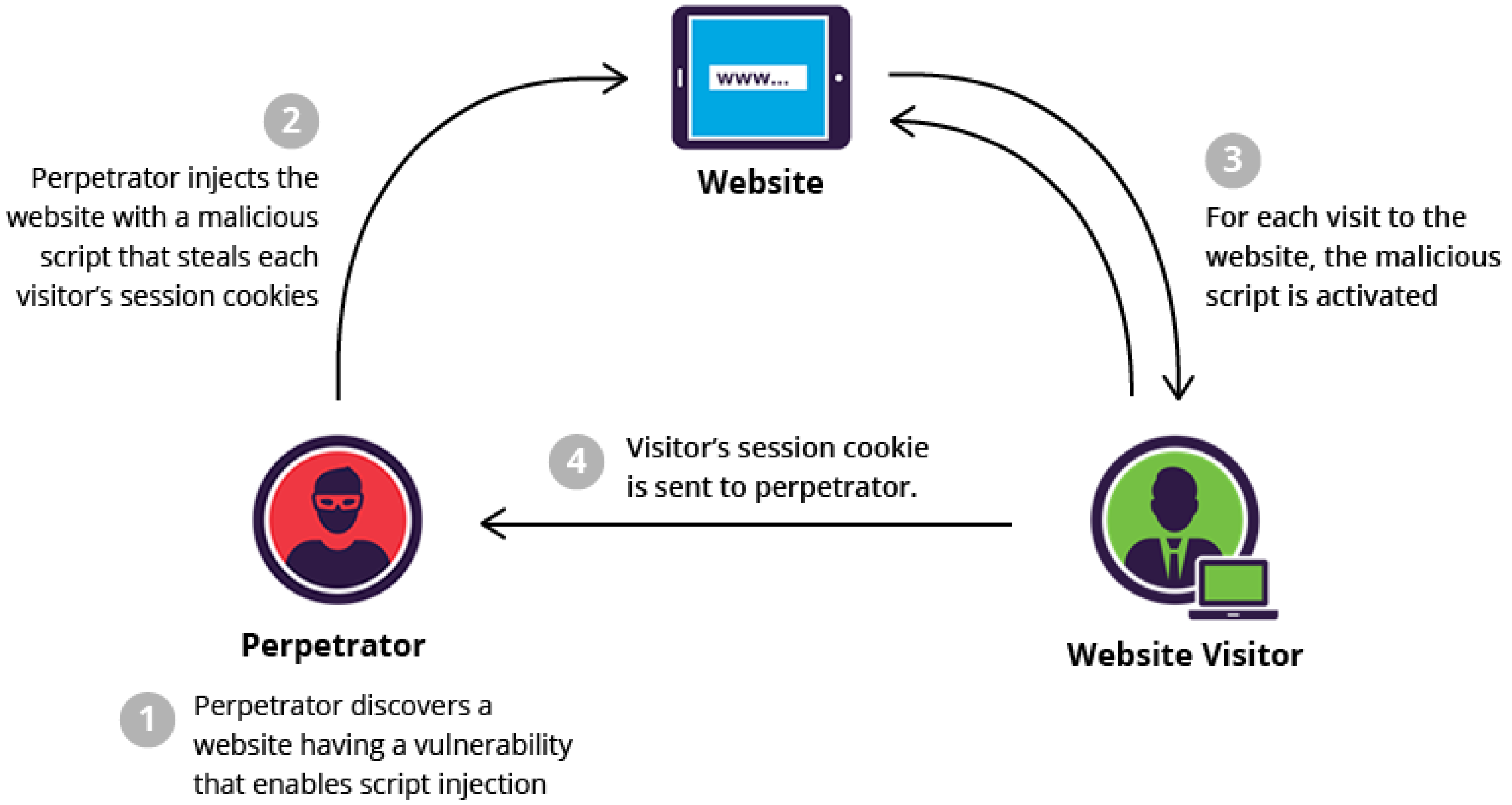

Reflected
Request
data=<script>print()</script>
Response
<p>Welcome <script>print()</script> </p>


Secure Website
2
The user is tricked into opening the link and requesting the malicious URL from the website
3
The website includes the malicious string in its response to the user
5
The User's sensitive information is sent to the attacker's server
User
Attacker
1
The attacker crafts a URL containing the malicious string and sends it to the victim.
4
The user's browser interprets the malicious JavaScript in the server's response as part of the legitimate web page and executes the code.

Web Application Penetration Testing Injections – XSS types

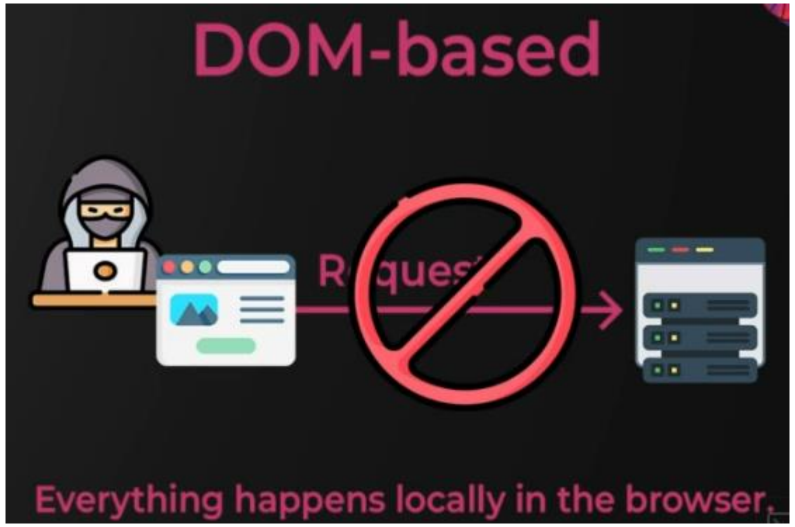
XSS Stored

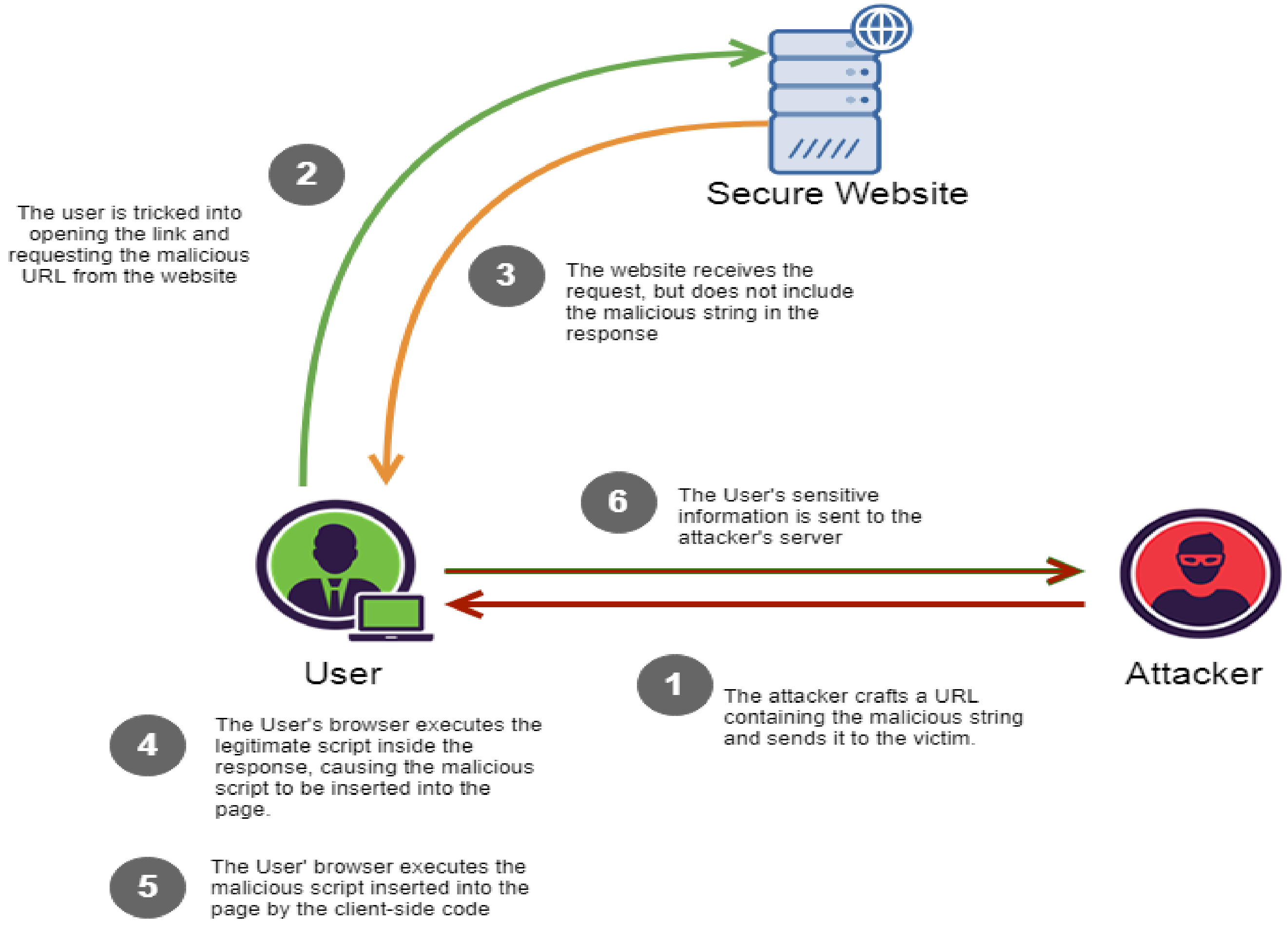



Website
www...
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having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

Web Application Penetration Testing Injections – XSS types

XSS DOM-Based




Secure Website
User
Attacker
1 The attacker crafts a URL containing the malicious string and sends it to the victim.
2 The user is tricked into opening the link and requesting the malicious URL from the website
3 The website receives the request, but does not include the malicious string in the response
4 The User's browser executes the legitimate script inside the response, causing the malicious script to be inserted into the page.
5 The User' browser executes the malicious script inserted into the page by the client-side code
6 The User's sensitive information is sent to the attacker's server

Web Application Penetration Testing Injections – XSS

- Avoid triggering with the alert() function
  - Filtered in chrome and easily detected
  - User print() or prompt() functions instead to pop an alert
  - https://portswigger.net/research/alerts-is-dead-long-live-print
- Using the http-only flag set Javascript cannot access the document.cookie

Web Application Penetration Testing Injections – Command Injection

- Very serious vulnerability. We may end up compromising the entire app and the host
- The application is taking input from the user and pass it as a parameter to a function that executes code on the system {eval(), system()}
  - Not mixing data and code is not something we took care of in such a condition
- We usually try to chain commands with operators like ;

Web Application Penetration Testing Injections – SSTI

Server Side Template Injection:

- Many web applications use templating engines to dynamically generate HTML content.
- These engines allow embedding of logic, such as loops and conditionals, within the template to render dynamic content.
- Common templating engines include Jinja2 (Python), Twig (PHP), and others.
- When user input is concatenated directly into a template, rather than passed in as data, attackers can directly inject arbitrary template directives in order to manipulate the template engine, often enabling them to take complete control of the server.

```
template = "Hello, {{ name }}!“
rendered = template.render(name=user_input)
```

Web Application Penetration Testing Injections – LFI/RFI

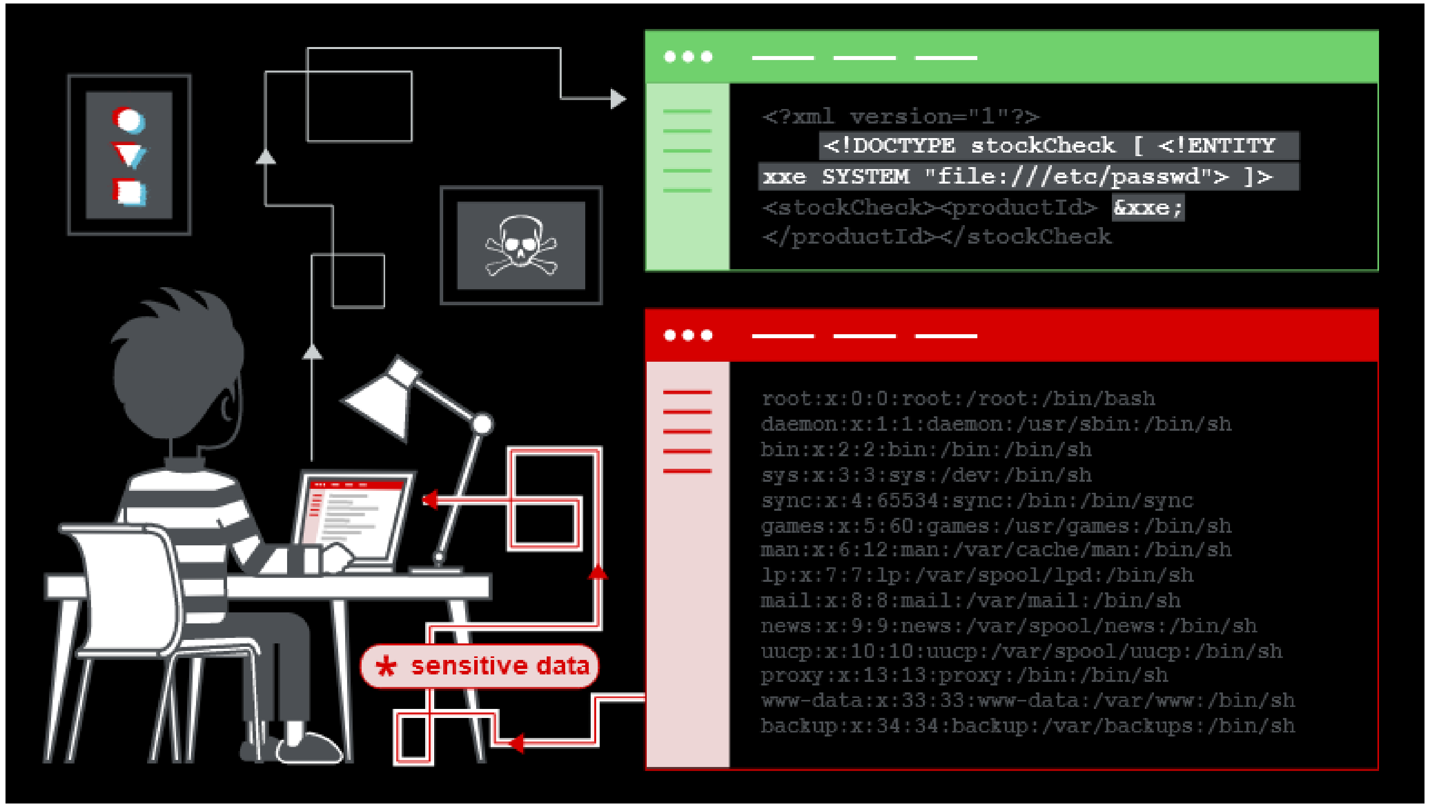
2 Types of File Inclusion Attacks

- Local File Inclusion: allows a target to include a file that is already present on the target server
  - Example : they might export the vulnerability to view sensitive configuration file containing credentials
- Remote File Inclusion: allows an attacker to include a malicious file from an external location
  - Could lead to arbitrary code execution on the vulnerable server and give us a way in.
- Mostly happen because an application takes an input from the user and using that path to a file without properly validating
- We have to know what is the target host (Linux | Windows)
- Search some payloads on PayloadAllTheThings github repo
- Try some filter bypasses (like double url encoding) or php filters (wrappers) for base64 encoding
- We could also again try fuzzing user entry points with payloads (like Intruder of Burp Suite)

Web Application Penetration Testing Injections – XXE Injection

XML External Entity Injection

- Some apps use XML to transfer data
- a web security vulnerability that allows an attacker to interfere with an application's processing of XML data
- 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.
- We could try XXE attacks on API endpoints that accept json files to see how they behave



```
<?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
```

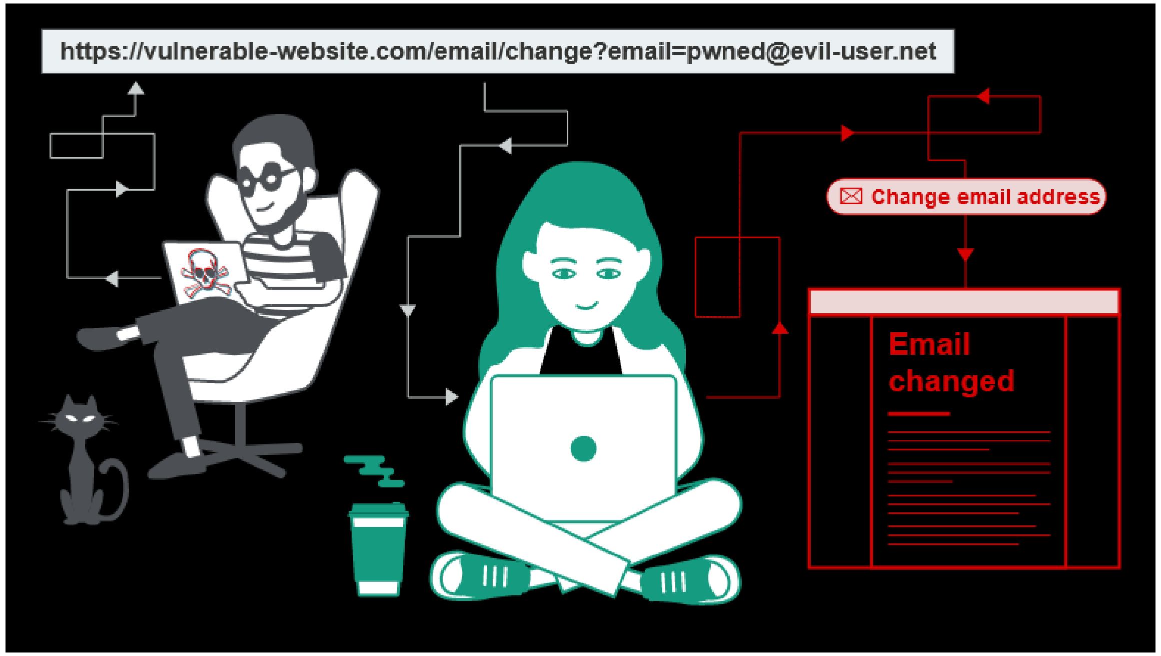
https://portswigger.net/web-security/xxe

Web Application Penetration Testing Injections – Insecure File Uploading

- A very straightforward attack: Trying to achieve some code execution after we have managed to upload a file.
- Could even lead to overriding files or DOS Attacks
- Weak filters may apply to both client side and server side
- Serve side filters: on file extensions (.png and not .php)
  - We could try file.php%00.png – using the null byte terminator
  - We could try file.php.png -
  - We could play with the magic bytes of files (usually the first bytes)
    - $ head file.png
  - Use other valid file extensions (like .php3, .php4, .pht)

Web Application Penetration Testing Injections – CSRF

An attack where we can check unsuspected users into performing an action within a web app that that user is authenticated to. (changing account settings, posting messages on forums, or even worse transferring funds to other bank accounts)


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

Web Application Penetration Testing Injections – CSRF

Conditions for a CSRF Attack to work:

- **A relevant action**. There is an action within the application that the attacker has a reason to induce. This might be a privileged action (such as modifying permissions for other users) or any action on user-specific data (such as changing the user's own password).
- **Cookie-based session handling**. Performing the action involves issuing one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests. There is no other mechanism in place for tracking sessions or validating user requests.
- **No unpredictable request parameters**. The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess. For example, when causing a user to change their password, the function is not vulnerable if an attacker needs to know the value of the existing password.

Web Application Penetration Testing Injections – CSRF

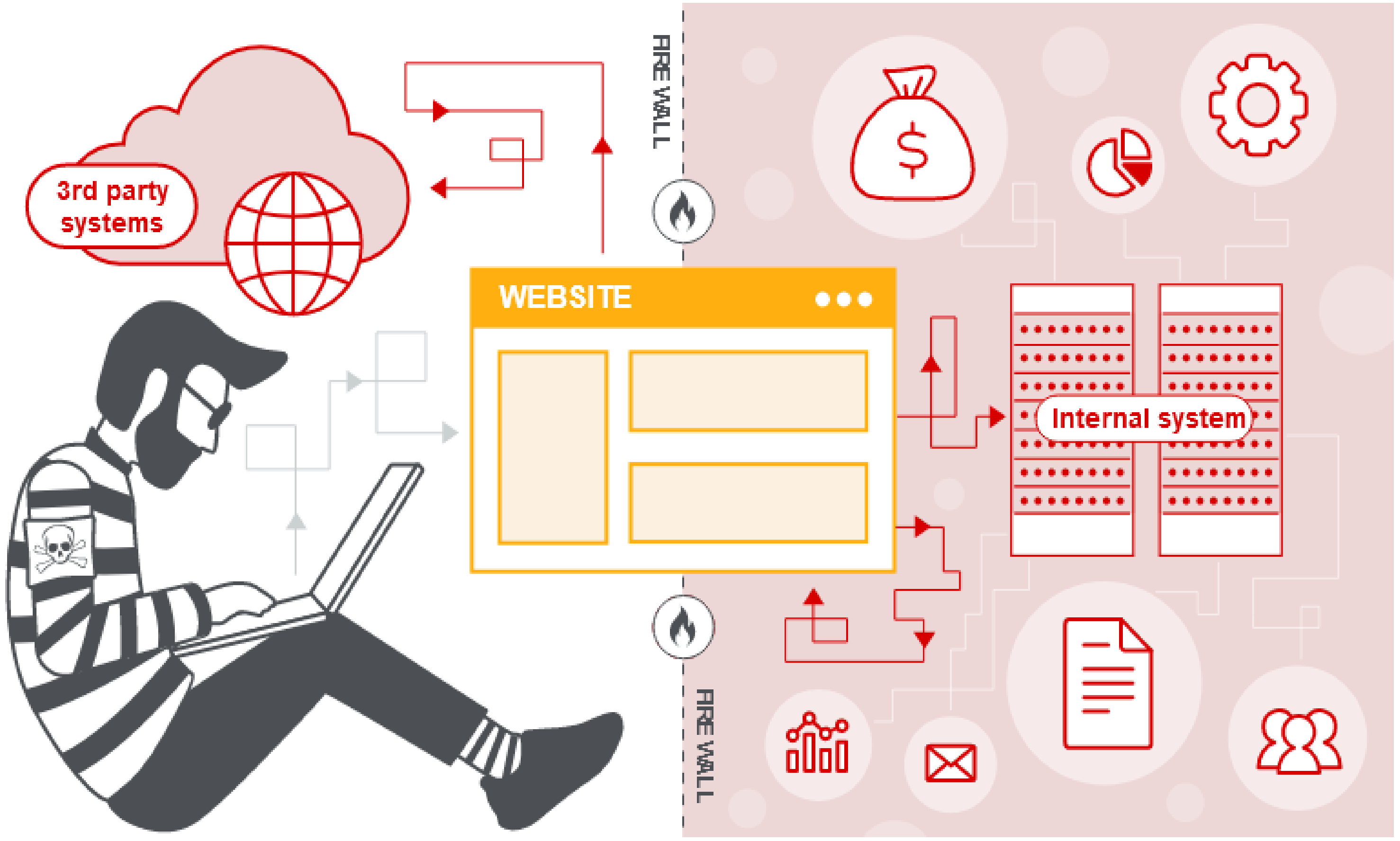
Common defences:

- **CSRF tokens:** a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client.
  - What to check for when we see csrf tokens in use?
    - Is it vulnerable to Cross Site Scripting ? Can we get a token from the app and then use it to our post request?
    - If I submit an old token would it still work?
    - What if I submit any token ? Is the logic of the app correct to check the value of the token or it checks that it just exists?
    - Checklist: AppSecExplained - CSRF
- **SameSite cookies:** a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. SameSite restrictions may prevent an attacker from triggering sensitive actions cross-site.
- **Referer-based validation cookies:** the HTTP Referer header to attempt to defend against CSRF attacks, normally by verifying that the request originated from the application's own domain

Web Application Penetration Testing Injections – SSRF

- A vulnerability that allows us to induce the server-side app to make requests on our behalf.
  - Target internal services and resources
  - Connections to external arbitrary systems
  - Could leak sensitive data
- Blind SSRF: harder but also harmful

Example: an application that a user can add a profile picture using url instead of uploading it. We could try to provide an internal url ….



https://portswigger.net/web-security/ssrf

Web Application Penetration Testing
Other Attacks - Subdomain Takeovers

An attacker gains control over a subdomain of a target domain. A subdomain points to an external service that is no longer in use or has been misconfigured, but the DNS records have not been updated or removed

1. DNS Configuration:

- A subdomain (sub.example.com) is configured in DNS to point to an external service (e.g., a cloud provider, content delivery network, or SaaS platform). This is done via DNS records such as CNAME, A, or ALIAS that link the subdomain to the external service.

2. The external service is discontinued, either because the organization stops using it, the service is shut down, or the subscription expires, however, the DNS record for the subdomain is not updated or removed.

3. Takeover:

  - An attacker discovers the dangling DNS record pointing to a non-existent or inactive resource.
  - Signs up for the same external service and claim the subdomain (e.g., setting up a new resource with the name sub.example.com on the same platform).
  - Once the attacker controls the subdomain he can serve malicious content, conduct phishing and social engineering attacks , distribute malware to the organization, cause Reputation damage, breach data ….

Web Application Penetration Testing Other Attacks – Open Redirects

When a web application accepts a user-controlled input that specifies a URL and then redirects the user's browser to that URL without sufficient validation.

1. A web application might have a functionality where it redirects users to different URLs based on user input, such as
   - http://example.com/redirect?url=http://example.com/home
2. An attacker manipulates the URL parameter to point to a malicious site
   - http://example.com/redirect?url=http://malicious-site.com
3. Redirection: The application processes the input without validating it, causing the user to be redirected to the malicious site.

Impact: Phishing Attacks, Bypassing Security Controls (Content Security Policy, Same Origin Policy) that rely on trusted domains, Reputation damage, Session Hijacking (capturing session tokens in a redirecting controlled website), XSS attacks

Defences:

- Validate/Encode URLs
- Avoid User-Controlled Redirects – (use fixed redirects or session-based state to manage navigation)

Web Application Penetration Testing Other Attacks

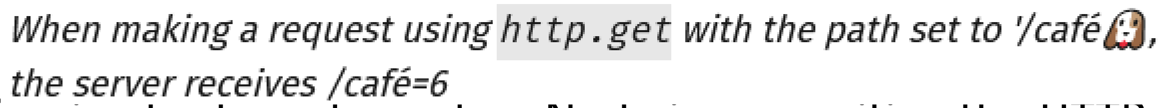
Chained Attack Scenario – Weather App HTB Web Challenge

- Inspect/review the code
  - Inspect the logic
- Find for vulns and login flaws
  - Find versions and related bugs and vulns
  - Find other vulnerabilities
- Chain them in a PoC exploit code

Web Application Penetration Testing Other Attacks

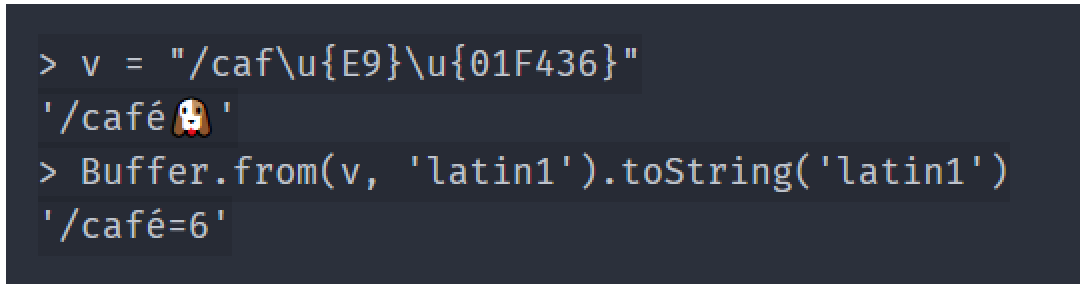
Chained Attack Scenario – Weather App HTB Web Challenge

- The bug: corruption of unicode characters in HTTP request path
    - A bug against Node.js http module



When making a request using `http.get` with the path set to '/café🐶', the server receives /café=6

    - issue was caused by a lossy encoding of unicode characters when Node.js was writing the HTTP request out to the wire
    - Node.js ultimately converts http request strings as raw bytes which means it must apply an appropriate encoding algorithm
    - Node.js (for requests with no body) defaults to 'latin1' – a single byte encoding that cannot represent high-numbered chars such as the emojis. Such characters are instead truncated to just their lowest byte of their internal JavaScript representation



```
> v = "/caf\u{E9}\u{01F436}"
'/café🐶'
> Buffer.from(v, 'latin1').toString('latin1')
'/café=6'
```

https://www.rfk.id.au/blog/entry/security-bugs-ssrf-via-request-splitting/

Web Application Penetration Testing Other Attacks

Chained Attack Scenario – Weather App HTB Web Challenge

- The vulnerability: SSRF via Request Splitting – CVE-2018-12116

  If Node.js can be convinced to use unsanitized user-provided Unicode data for the `path` option of an HTTP request, then data can be provided which will trigger a second, unexpected, and user-defined HTTP request to made to the same server.

```
GET /private-api?q=<user-input-here> HTTP/1.1
Authorization: server-secret-key
```

```
"x HTTP/1.1\r\n\r\nDELETE /private-api HTTP/1.1\r\n"
```

```
GET /private-api?q=x HTTP/1.1

DELETE /private-api
Authorization: server-secret-key
```

https://www.rfk.id.au/blog/entry/security-bugs-ssrf-via-request-splitting/

Web Application Penetration Testing Other Attacks

Chained Attack Scenario – Weather App HTB Web Challenge

- Good-quality HTTP libraries will typically include mitigations to prevent this behaviour, and Node.js is no exception: if you attempt to make an outbound HTTP request with control characters in the path, they will be percent-escaped before being written out to the wire.

- When Node.js version 8 or lower makes a GET request to this URL, it doesn't escape them because they're not HTTP control characters:

```
> Buffer.from('http://example.com/\u{010D}\u{010A}/test', 'latin1').toString()
'http://example.com/\r\n/test'
```

- Our path is the ‘endpoint’ variable in the /register post api request

%27	‘
%22	“
\u0120	(space)
\u010D	\r
\u010A	\n

So our chain is:

Request Splitting + Corruption of Unicode chars →

→SSRF – to bypass the 127.0.0.1 restriction → SQL injection found from code review

https://www.rfk.id.au/blog/entry/security-bugs-ssrf-via-request-splitting/

Web Application Penetration Testing Other Attacks – Vulnerable Components

- It is common for developers to incorporate 3rd party components like libraries, frameworks, plugins into the application. Building new products would be way to time consuming and difficult if we didn’t use them.
- However, they also introduce potential vulnerabilities if they are outdated, misconfigured or inherently insecure.
- We have to enumerate and list all third party components and check for vulnerabilities.

- Workshop for Demo -

Questions?