OWASP

Open Web Application Security Project

OWASP Top 10 – 2017

The Ten Most Critical Web Application Security Risks

Null - An open security community

JAY PATEL

Hello!

I am Jay Patel

I am here to **learn** something new and to **share** the knowledge.

Final year **Computer Science Engineering Student** at DIT University

You can find me at **@jay13patel**

linkedin.com/in/jay13patel

github.com/jay13patel

Agenda

1. Purpose of this Session

- **Provide an overview of Web Application Security threats and Defenses.**

- **Using the Open Web Application Security Project (OWASP) Top 10 List we will:**
  - **Define the vulnerabilities**
  - **Illustrate the Web Application Vulnerabilities using OWASP JUICE SHOP**
  - **Explain how to protect against the vulnerabilities.**

A Company encountered with a huge security breach in 2018 in which nearly

50,000,000

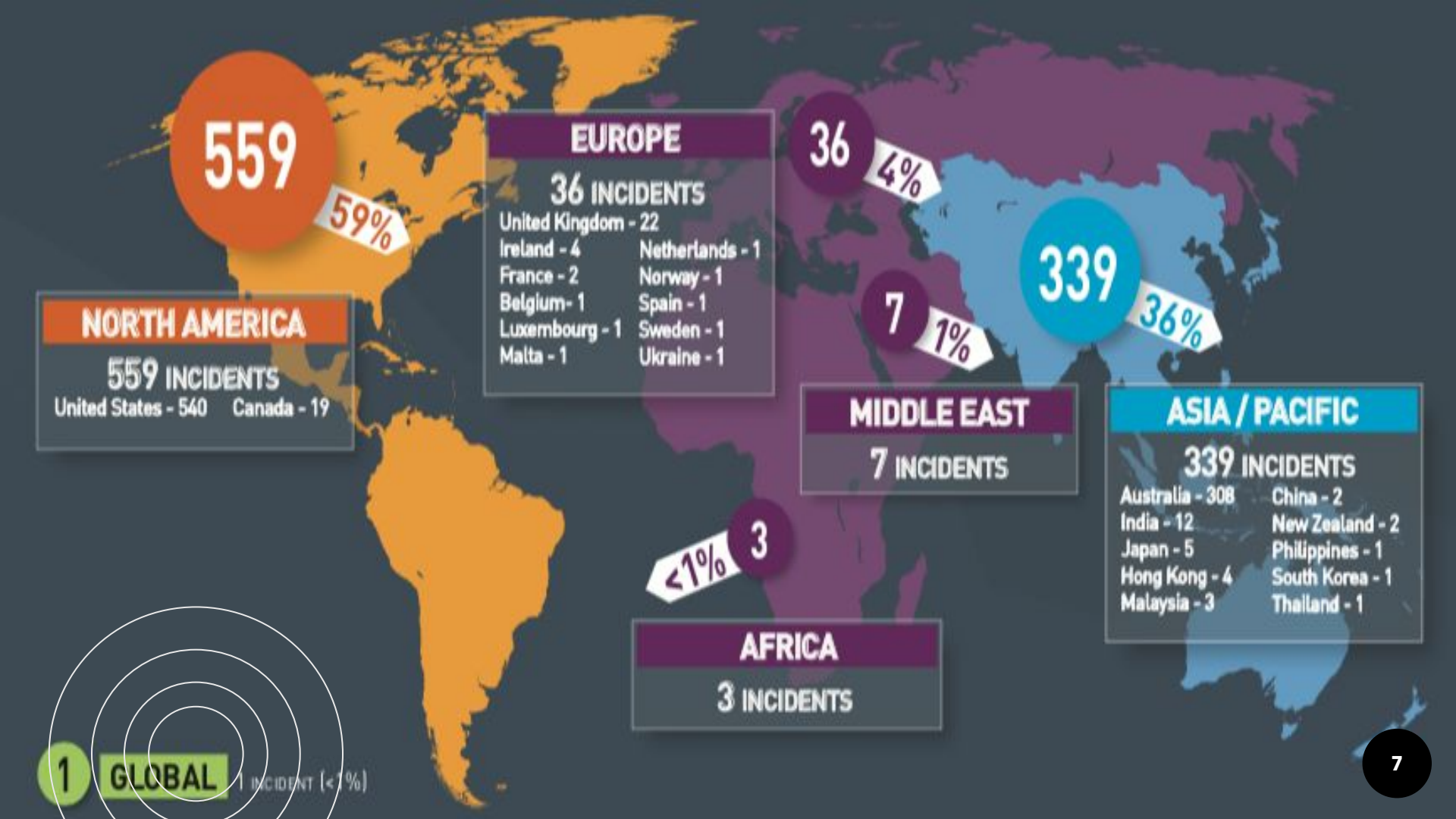
users were compromised .


facebook

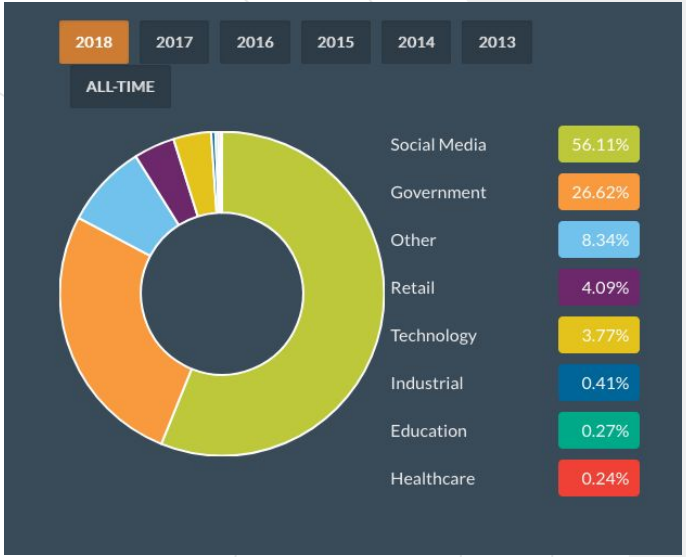
DID YOU KNOW ?

Facebook says attackers exploited a “vulnerability” in Facebook’s code that impacted **“View As”**, a feature that lets people see what their own profile looks like to someone else. This allowed them to **steal Facebook access tokens** which they could then use to take over people’s accounts.”

Broken Access Control
Or
Broken Authentication


559
59%
NORTH AMERICA
559 INCIDENTS
United States - 540 Canada - 19
EUROPE
36 INCIDENTS
United Kingdom - 22
Ireland - 4
France - 2
Belgium- 1
Luxembourg - 1
Malta - 1
Netherlands - 1
Norway - 1
Spain - 1
Sweden - 1
Ukraine - 1
36
4%
7
1%
MIDDLE EAST
7 INCIDENTS
339
36%
ASIA / PACIFIC
339 INCIDENTS
Australia - 308
India - 12
Japan - 5
Hong Kong - 4
Malaysia - 3
China - 2
New Zealand - 2
Philippines - 1
South Korea - 1
Thailand - 1
<1%
3
AFRICA
3 INCIDENTS
1
GLOBAL
1 INCIDENT (<1%)

Organization Breached	Records Breached	Date of Breach	Type of Breach	Source of Breach	Location	Industry	Risk Score
Facebook	2,200,000,000	04/04/18	Identity Theft	Malicious Outsider	United States	Social Media	10.0
Government of India/Aadhaar	1,200,000,000	01/03/18	Identity Theft	Malicious Outsider	India	Government	10.0
Equifax	147,900,000	07/15/17	Identity Theft	Malicious Outsider	United States	Financial	10.0
Reliance Jio	120,000,000	07/10/17	Account Access	Malicious Outsider	India	Technology	10.0
Friend Finder Networks	412,214,295	10/16/16	Existential Data	Malicious Outsider	United States	Entertainment	10.0
Anthem Insurance Companies (Anthem Blue Cross)	78,800,000	01/27/15	Identity Theft	State Sponsored	United States	Healthcare	10.0
Yahoo	500,000,000	12/01/14	Account Access	State Sponsored	United States	Technology	10.0
Home Depot	109,000,000	09/02/14	Financial Access	Malicious Outsider	United States	Retail	10.0
JPMorgan Chase	83,000,000	08/27/14	Identity Theft	Malicious Outsider	United States	Financial	10.0
CyberVor	1,200,000,000	08/05/14	Account Access	Malicious Outsider	Global	Technology	10.0

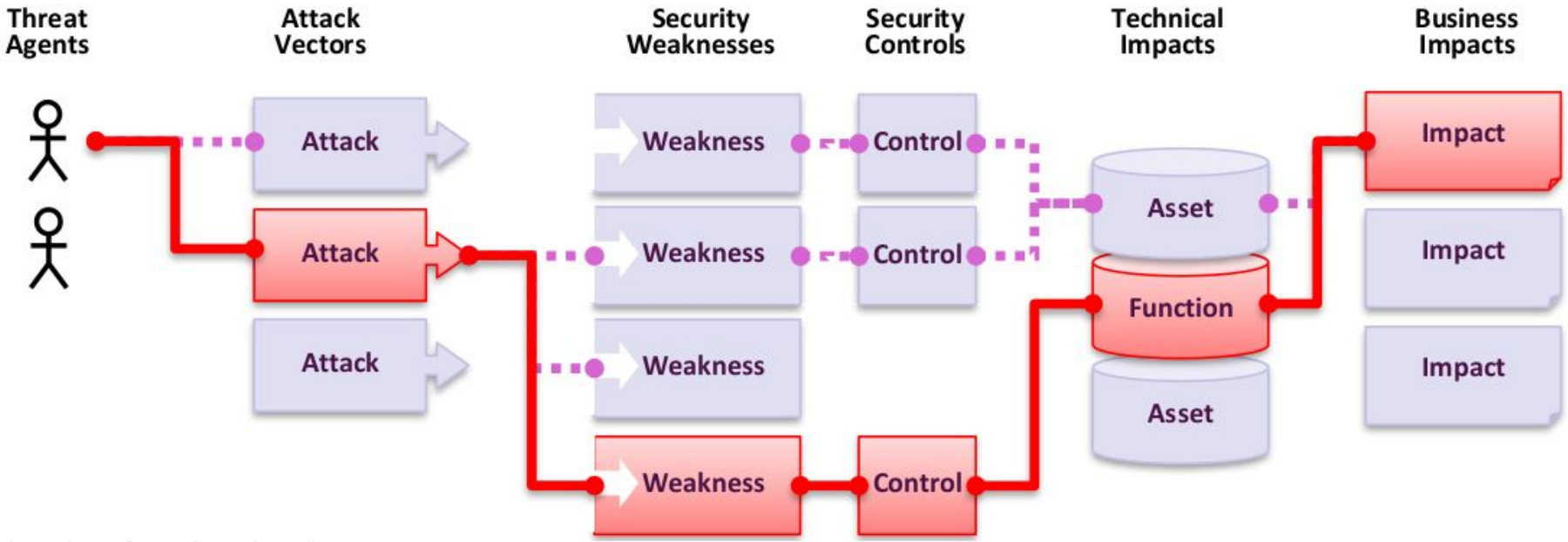


The number of records stolen in the first half of 2018 reached an astonishing 4,553,172,708, an increase of 133 percent over the first half of 2017 according to a report from Gemalto.

SOURCE : SOFTPEDIA NEWS

Open Web Application Security Project (OWASP)

What Are Application Security Risks?



OWASP Top 10 Vulnerabilities Changes from 2013 to 2017

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

1 A1-Injection

CWE-74

Allowing untrusted data to be sent as part of a command or query

* **Common Weakness Enumeration**

WHAT IS SQL INJECTION?

A SQL query is one way an application talks to the database.

SQL injection occurs when an application fails to sanitize untrusted data (such as data in web form fields) in a database query.

An attacker can use specially-crafted SQL commands to trick the application into asking the database to execute unexpected commands.

Let's simplify SQL Injection

If a website, app, or device incorporates user input within a command, an attacker can insert a "payload" command directly into said input. If that input is not verified, an attacker then "injects" and runs their own commands.

The most important property violated here is "Access Control" Data is leaked to people without any access. Access should only be given to system administrators.

Technical implementations :

- **Incorrectly filtered escape characters**
- **Blind SQL injection**
- **Second order SQL injection**

Incorrectly filtered escape characters

Original query =

statement = "SELECT * FROM users WHERE name = '" + userName + "';"

Payload = ' OR '1'='1

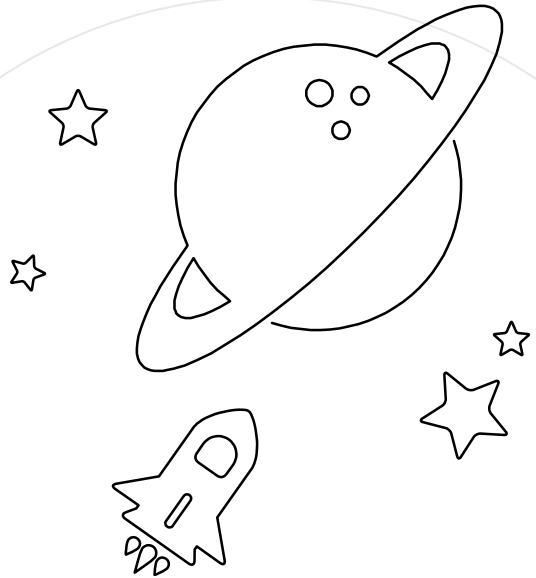
or using comments to even block the rest of the query

' OR '1'='1' -- ' OR '1'='1' { ' OR '1'='1' /*

Injection = payload + original Query

SELECT * FROM users WHERE name = '' OR '1'='1';

SELECT * FROM users WHERE name = '' OR '1'='1' -- ';



Big concept

HOW DOES IT WORK?

Query running in backend

error: {...}
message: "SQLITE_ERROR: unrecognized token:\"3590cb8af0bbb9e78c343b52b93773c9\""
name: "SequelizeDatabaseError"
original: {...}
code: "SQLITE_ERROR"
errno: 1

sql: "SELECT * FROM Users WHERE email = ' ' ' AND password '3590cb8af0bbb9e78c343b52b93773c9' AND deletedAt IS NULL"

Now it's time to exploit the vulnerability

Exploit the vulnerability using sql injection

ORIGINAL QUERY

SELECT * FROM Users WHERE email = ' “email-id” ' AND password = '3590cb8af0bbb9e78c343b52b93773c9' AND deletedAt IS NULL

MODIFIED QUERY BY ADDING ' OR '1'='1' -- IN THE TEXT BOX

SELECT * FROM Users WHERE email = ' ' OR '1'='1' -- ' AND password = '3590cb8af0bbb9e78c343b52b93773c9' AND deletedAt IS NULL


DEMO TIME !!!

How to Prevent ?

- For server products and libraries, keep up with the latest bug reports for the products you are using.
- Periodically scan your web site with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications.
- Size checking on all input.
- Ensure the web application runs with only the privileges it absolutely needs to perform its function.

#1

SQL injection was leveraged in the infamous Sony Pictures hack of 2014, when suspected North Korean operatives gained access to confidential data. According to US-CERT, the attackers used a Server Message Block Worm Tool to install several malicious components,including a backdoor and other destructive tools.

2

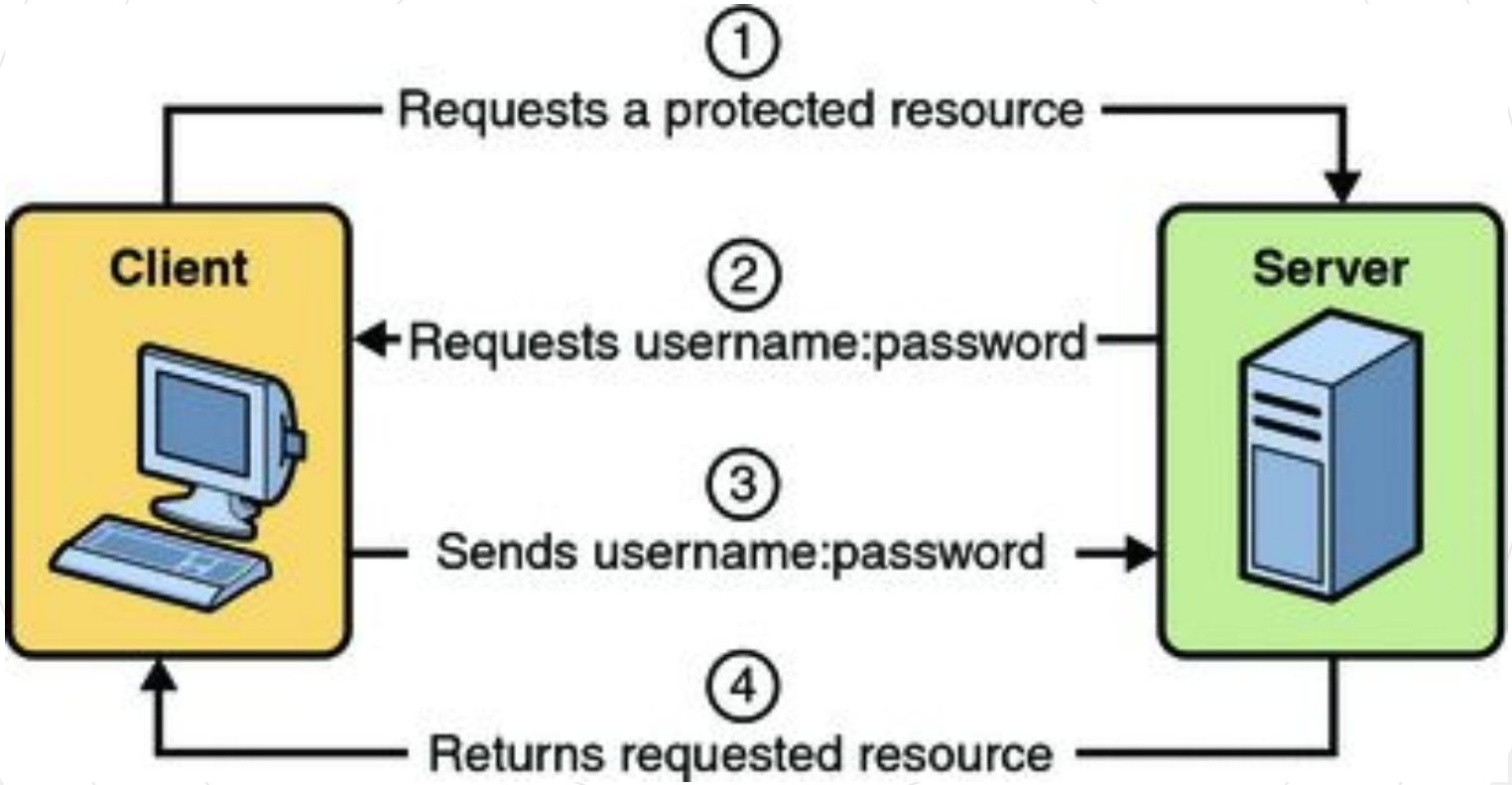
A2-BROKEN AUTHENTICATION

CWE-287,CWE-352

Incorrectly implemented authentication and session management functions

* **Common Weakness Enumeration**

What is authentication ?

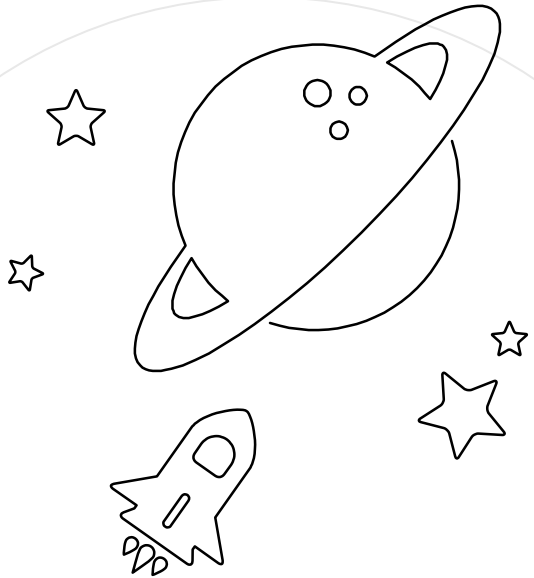


How we can break the authentication ?

In the simplest attacks, passwords can be guessed or stolen if left Unprotected. As complexities are added, attackers can find other areas where user credentials or sessions have inadequate protections and then hijack a user's access, and eventually their data.

Technical implementations :

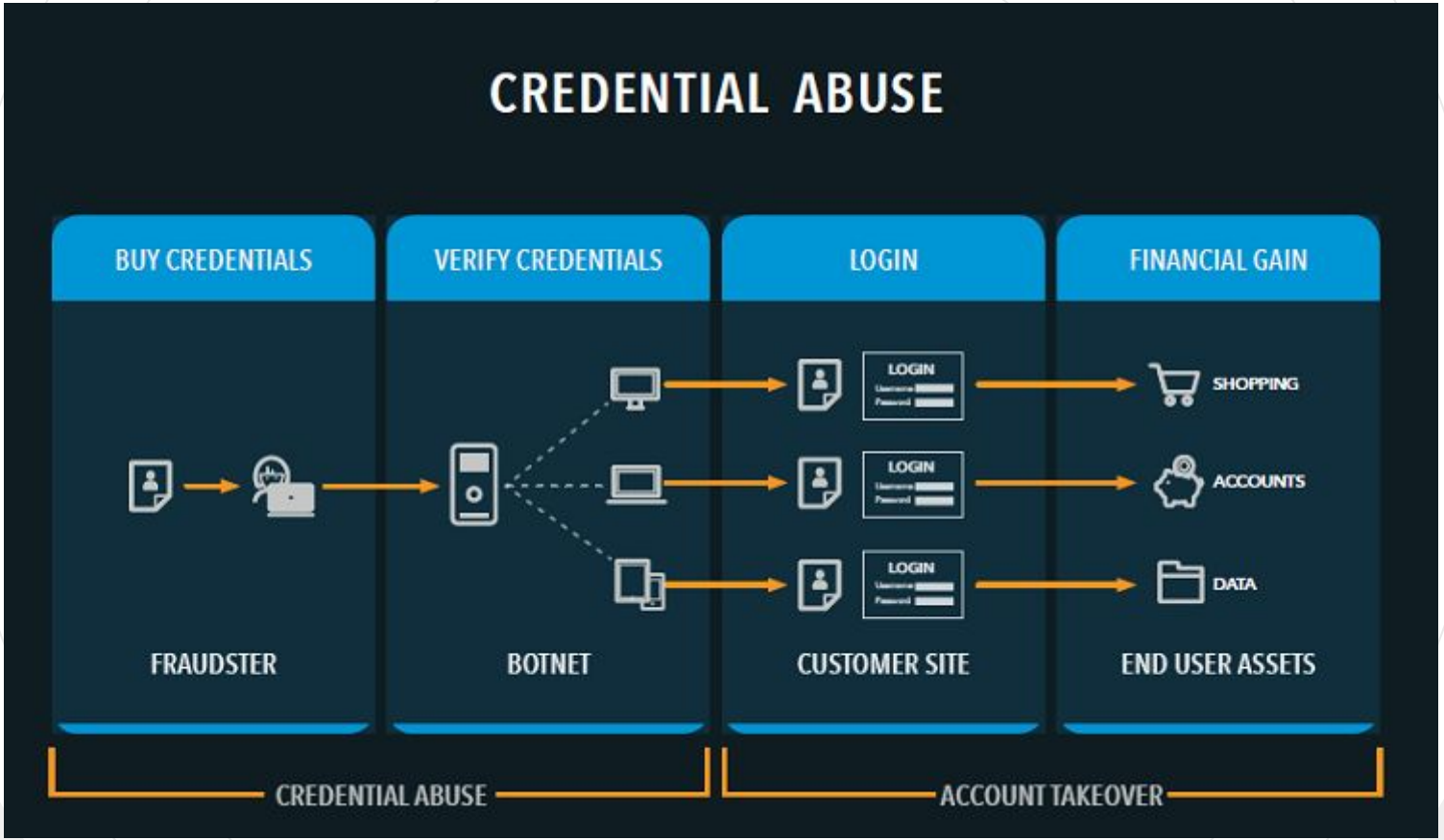
- Credentials stuffing
- Automated attacks
- Default password
- Session Hijacking



Big concept

HOW DOES IT WORK?

Credentials Stuffing




DEMO TIME !!!

How to Prevent ?

- Where possible, implement multi-factor authentication to prevent credential stuffing, brute force, automated, and stolen credential attacks.
- Do not ship or deploy with any default credentials, particularly for admin users
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Implement weak password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.

#2

The simplest examples of this vulnerability are either storing user credentials without encryption or allowing them to be easily guessed. Other examples include using session IDs in the URL and enabling unreasonably long session timeouts.

3

A3-SENSITIVE DATA EXPOSURE

CWE-200, CWE-327, CWE-328, CWE-548

Many web technologies weren’t designed to handle financial or personal data transfers

* Common Weakness Enumeration

Sensitive data, such as credit card numbers, health data, or passwords, should have extra protection given the potential of damage if it falls into the wrong hands. There are even regulations and standards designed to protect sensitive data. But, if sensitive data is stored, transmitted, or protected by inadequate methods, it can be exposed to attackers.

Why it is ranked as 3 out of 10 in OWASP TOP 10 ?

How data is exposed which is sensitive ?

If data is stored or transferred as plain text, if older/weaker encryption is used, or if data is decrypted carelessly, attackers can gain access and exploit the data.

- **Data stored in plain text, such as passwords or credit card data**
- **Lack of HTTPS on authenticated pages**
- **Hashed passwords with lack of salt, making the password easily cracked**
- **Tokens disclosed in public source code**


DEMO TIME !!!

How to Prevent ?

- **Classify your Data**
- **Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).**
- **Make sure to encrypt all sensitive data at rest.**
- **Don’t store sensitive data unnecessarily. Discard it as soon as possible.**

#3

Wireless routers offer notoriously weak data protections. Researchers recently found that the cryptography protecting WPA2, the industry standard, exposes data and allows it to be read or manipulated as it's wirelessly transferred .

4

A4-XML EXTERNAL ENTITIES

CWE-611

XML “entities” can be used to request local data or files

* **Common Weakness Enumeration**

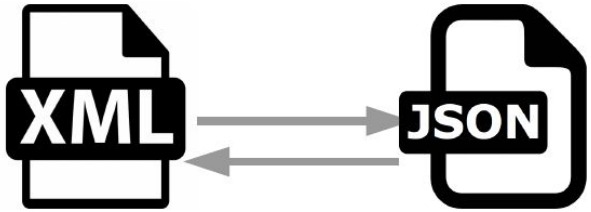
What is XML ?

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

 <book category="cooking">
       <title lang="en">Everyday Italian</title>
       <author>Giada De Laurentiis</author>
        <year>2005</year>
       <price>30.00</price>
 </book>

 <book category="children">
       <title lang="en">Harry Potter</title>
       <author>J K. Rowling</author>
        <year>2005</year>
       <price>29.99</price>
 </book>

</bookstore>
```

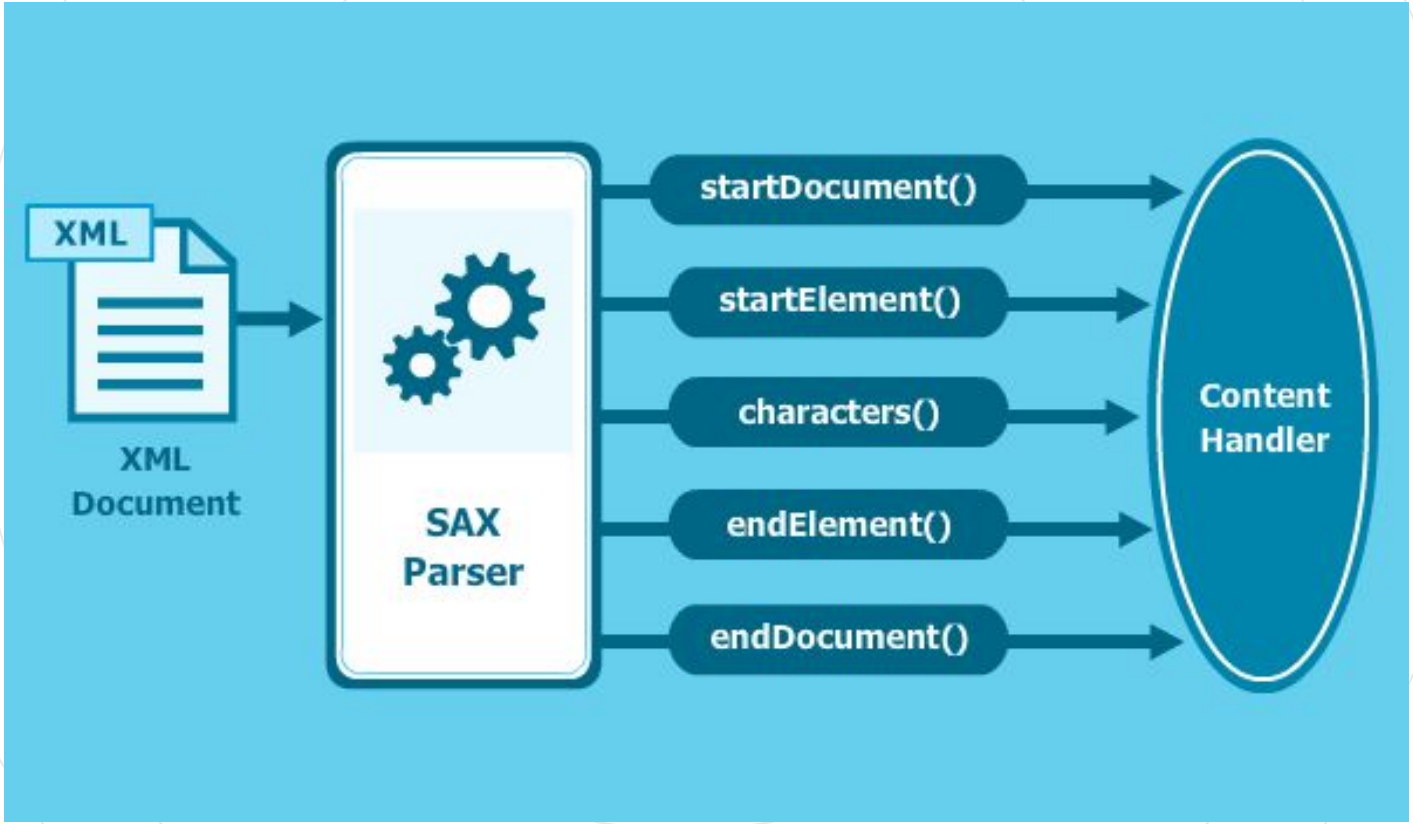
Why we use XML ?

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

Alternative to XML is JSON .



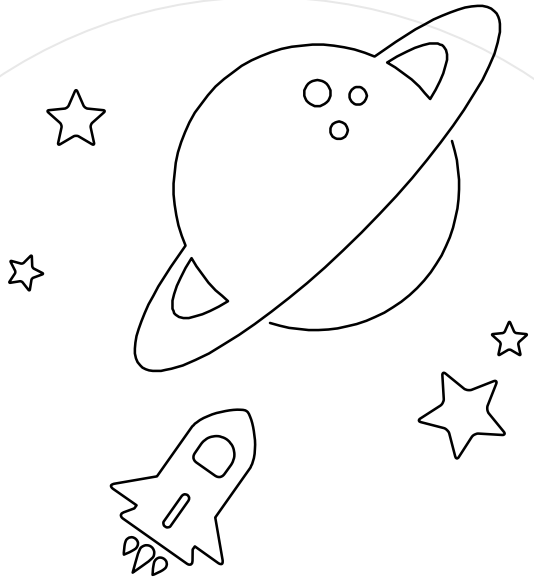
What is XML Parser ?



XXE-XML EXTERNAL ENTITIES

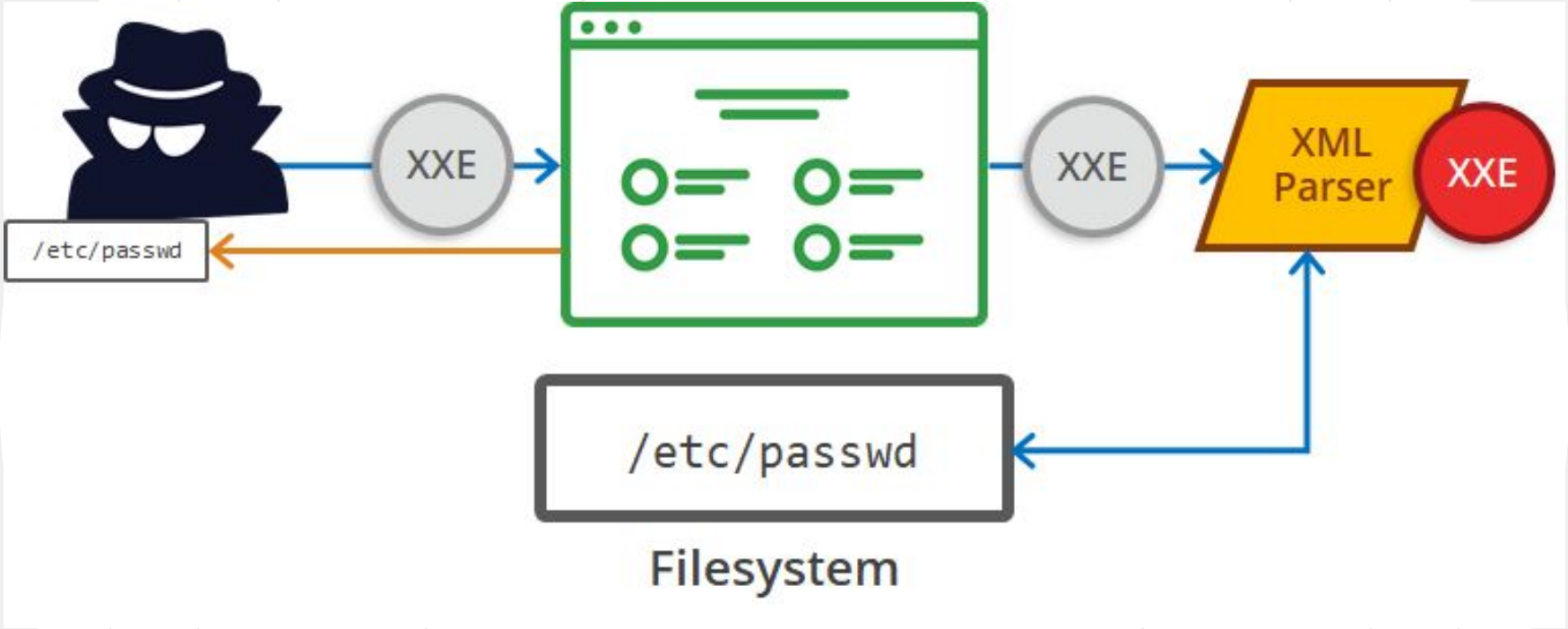
An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

```
<?xml version="1.0"?>
<!DOCTYPE Rohit [
<!ENTITY entityex SYSTEM "file:///folder/file" >
]>
<abc>&entityex;</abc>
```



Big concept

HOW DOES IT WORK?



Attackers can gain access to any data stored locally, or can further pivot to attack other internal systems.

How XXE can be used to gain access to any data stored locally ?

Request

```
POST http://example.com/xml HTTP/1.1
<?xml version="1.0"
encoding="ISO-8859-1"?>
<!DOCTYPE foo [
      <!ELEMENT foo ANY>
      <!ENTITY xxe SYSTEM
      "file:///etc/passwd">
]>
<foo>
      &xxe;
</foo>
```

Response

HTTP/1.0 200 OK

```
root:x:0:0:root:/root:/bin/
bash
daemon:x:1:1:daemon:/u
sr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/s
h
(...)
```


DEMO TIME !!!

How to Prevent ?

- **Disable XML external entity and DTD processing in all XML parsers in your application**
- **Implement positive ("white listing") input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.**
- **Patch or upgrade all the latest XML processors and libraries in use by the app or on the underlying operating system.**
- **Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.**

#4

The classic “billion laughs” attack exploits XXE by Defining 10 elements that refer to each other, quickly exceeding any available memory and disrupting entire services.

5

A5-BROKEN ACCESS CONTROL

CWE-22, CWE-285, CWE-639

Improper enforcement of what authenticated users are allowed to do

* Common Weakness Enumeration

Authentication and Authorisation

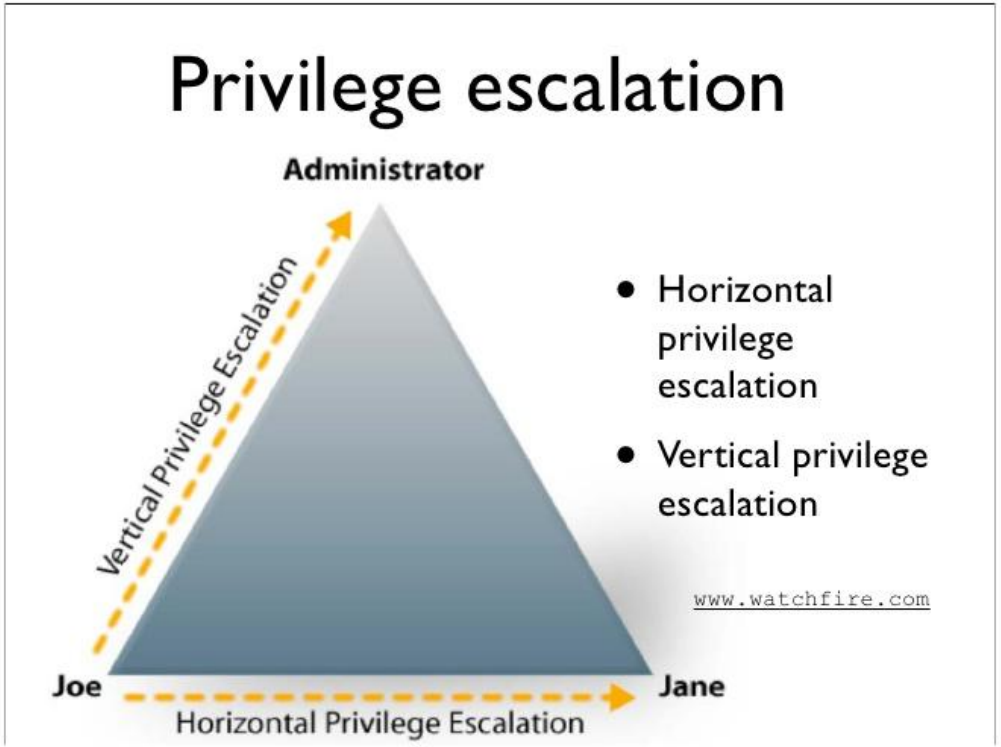
Authentication

Who you are

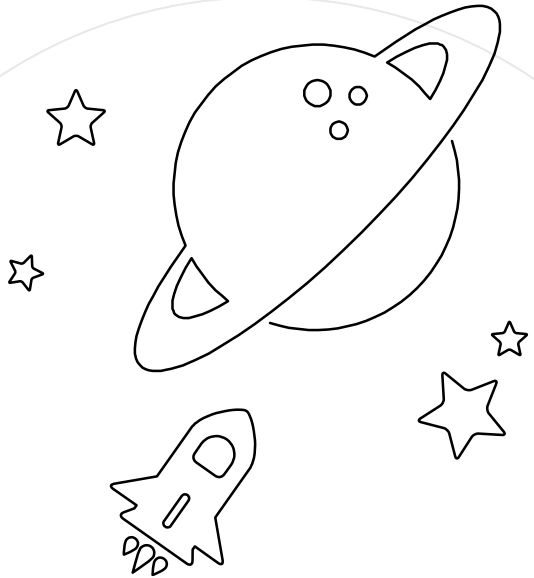
Authorization

What you can do

What is Privilege escalation ?



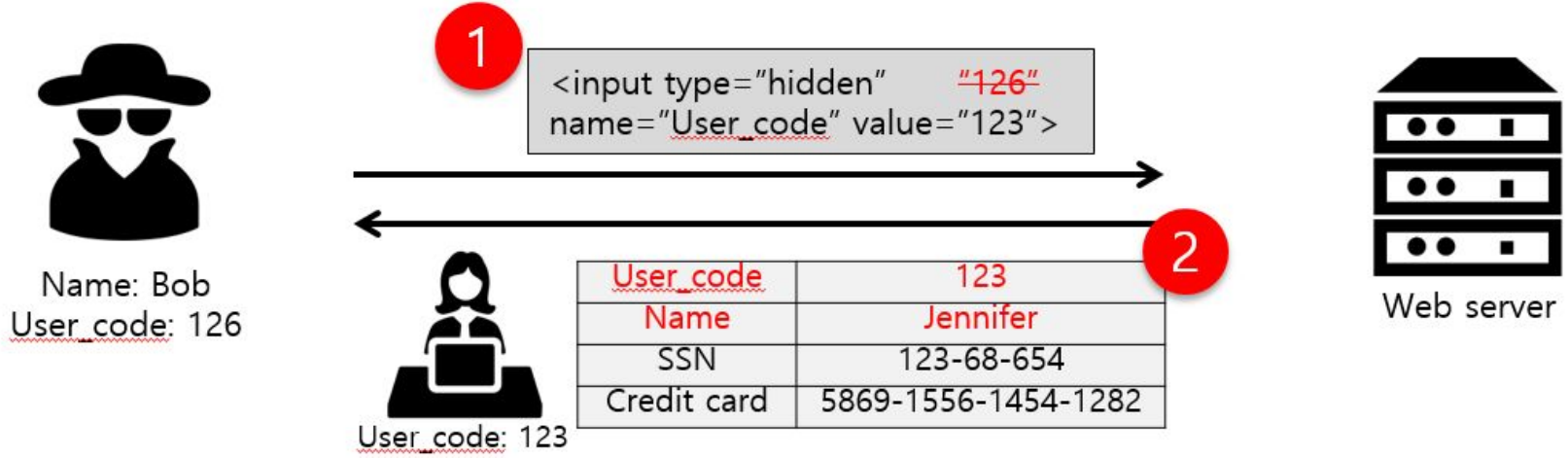
Sometimes, gaining unauthorized access is as simple as manually entering an unlinked URL in a browser, such as http://example.com/admin.



Big concept

HOW DOES IT WORK?

Exploiting Broken access control



As with other vulnerabilities, attackers can gain access to (and modify) data, accounts, and functions that they shouldn't.


DEMO TIME !!!

How to Prevent ?

- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update or delete any record.
- Implement access control mechanisms once and re-use them throughout the application.
- With the exception of public resources, deny by default.
- Disable web server directory listing, and ensure file metadata such (e.g. .git) is not present within web roots

#5

A web meeting platform, Fuze, enabled meeting access via a simple URL ending with an incrementing seven-digit number. Using any number provided access to replays of the corresponding meeting. Since the URLs were unprotected, the content was then indexed by — and searchable through — popular search engines.

6

A6-SECURITY MISCONFIGURATION

CWE-209

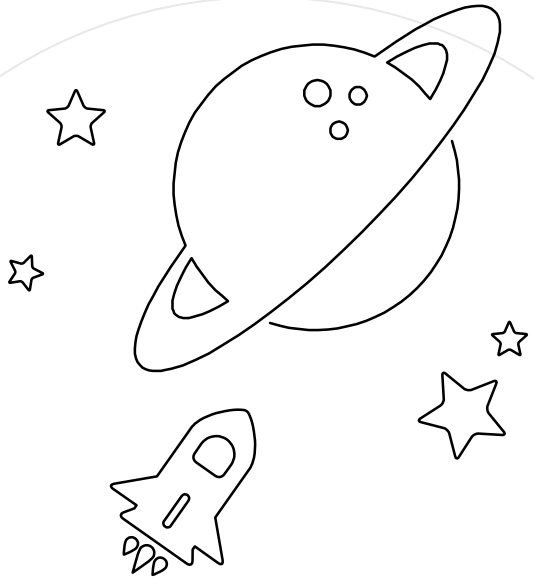
Manual, ad hoc, insecure, or lack of security configurations that enable unauthorized access

* Common Weakness Enumeration

Am I Vulnerable to Security Misconfig?

Exactly what its name implies, security misconfiguration is when you've overlooked some vulnerabilities. This includes using default credentials, leaving files unprotected on public servers, having known-but-unpatched flaws, and more, and at any layer of the software stack. In other words, it's your fault.

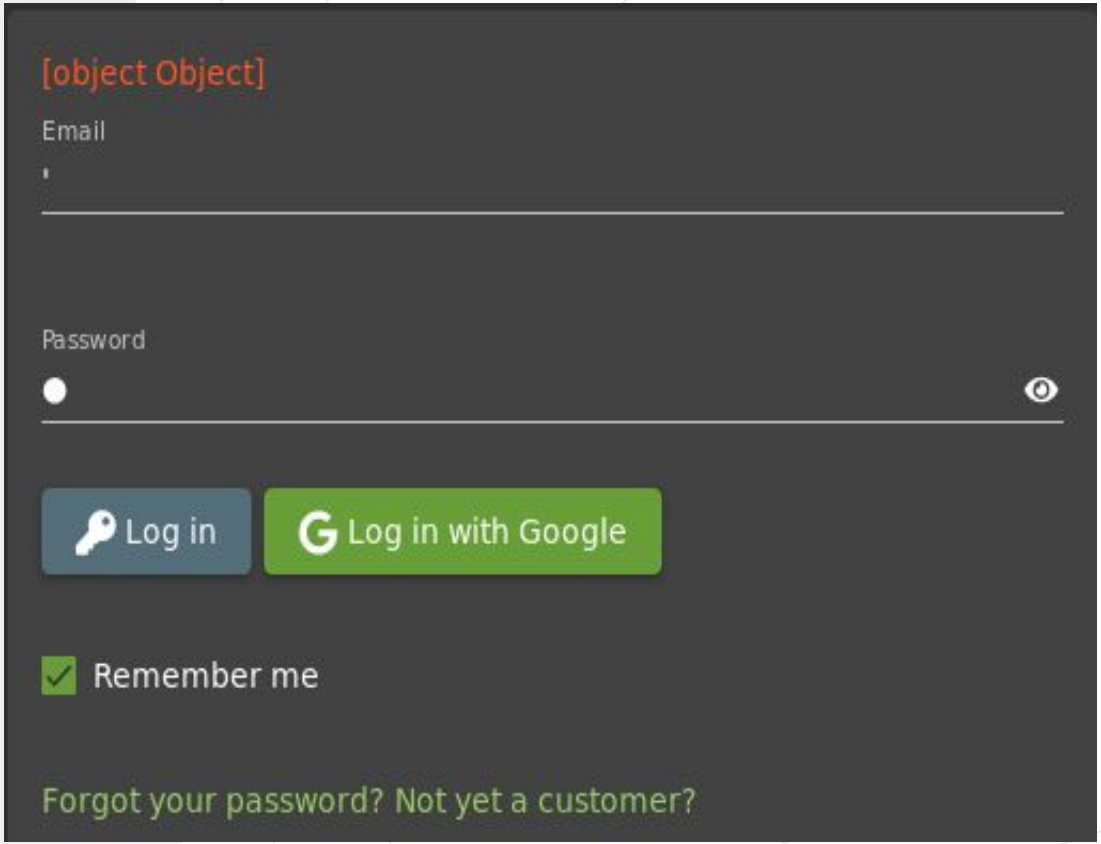
- Are any unnecessary features enabled or installed (e.g. ports, services, pages, accounts, privileges)?
- Are default accounts and their passwords still enabled and unchanged?
- Does your error handling reveal stack traces or other overly informative error messages to users?
- Do you still use ancient configs with updated software? Do you continue to support obsolete backward compatibility?



Big concept

HOW DOES IT WORK?

What happens when Error Handling is not done properly ?



Any request that **cannot be properly handled** by the server will eventually be passed to a global error handling component that sends an **error page** to the client that includes a stack trace and other s**ensitive information**. The restful API behaves in a similar way, passing back a **JSON error object** with sensitive data, such as **SQL query strings.**


DEMO TIME !!!

How to Prevent ?

- A minimal platform without any unnecessary features,components, documentation, and samples. Remove or do not install unused features and frameworks

- Check if you application is using components with known vulnerabilities

- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.

- A strong application architecture that provides effective, secure separation between components, with segmentation, containerization, or cloud security groups (ACLs).

#6

The infamous Mirai botnet of 2016 relied on unchanged default credentials (such as a login of “admin” and a password of “1234”) of just over 60 specific IoT devices. When exploited, it eventually infected nearly 400,000 units of just those 60 unprotected devices.

7

A7-CROSS-SITE SCRIPTING (XSS)

CWE-79

A web application includes untrusted data in a new web page without proper validation

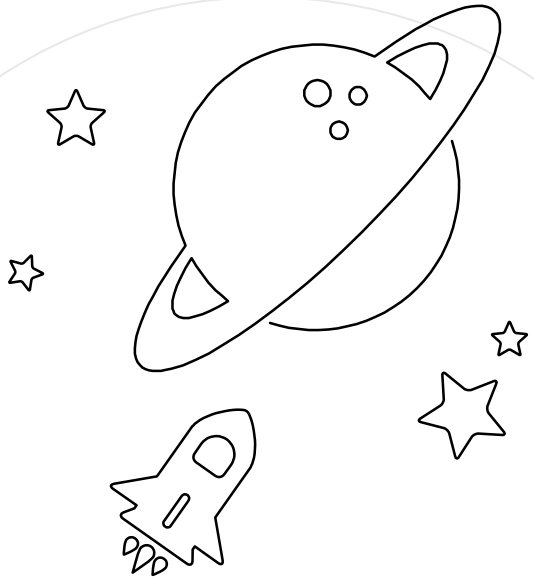
* **Common Weakness Enumeration**

What is XSS ?

Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.

Types of XSS

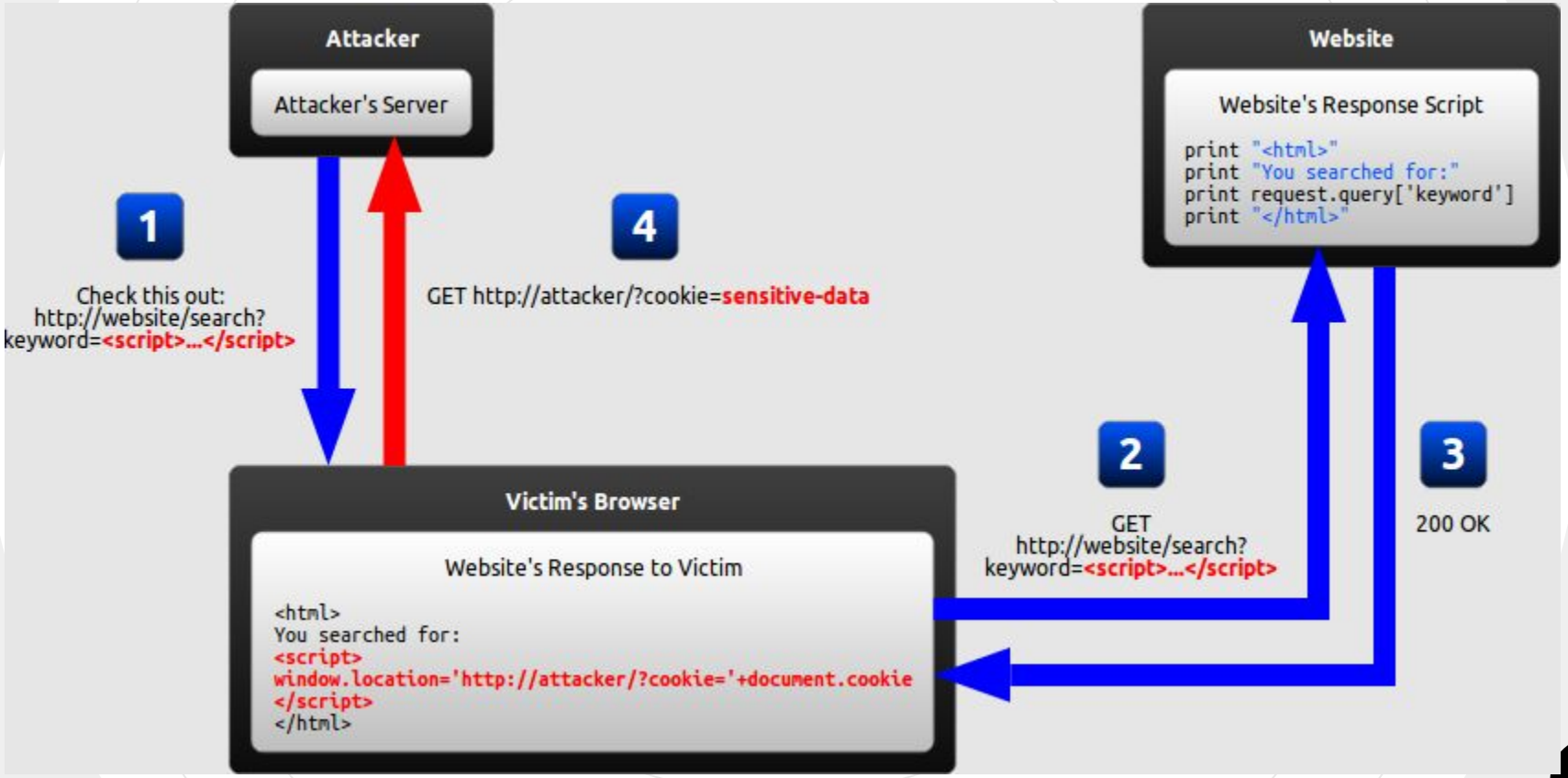
- Persistent XSS, where the malicious string originates from the website's database.
- Reflected XSS, where the malicious string originates from the victim's request.
- DOM-based XSS, where the vulnerability is in the client-side code rather than the server-side code.



Big concept

HOW DOES IT WORK?

Reflected XSS



Exploit the reflected XSS ?

Attacker payload

http://127.0.0.1:3000/#/track-result?id=%3Ciframe%20src%3D%22javascript:alert(document.cookie)%22%3E

Or

<iframe src="javascript:alert(document.cookie)">

1. The attacker crafts a URL containing a malicious string and sends it to the victim.
2. The victim is tricked by the attacker into requesting the URL from the website.
3. The website includes the malicious string from the URL in the response.
4. The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.


DEMO TIME !!!

How to Prevent ?

- **Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript,CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities.**

- **Using frameworks that automatically escape XSS by design,such as the latest Ruby on Rails, React JS.**

- **Applying context sensitive encoding when modifying the browser document on the client side acts against DOM XSS.**

#7

XSS exploits have been reported for more than 20 years, and have impacted Twitter, Facebook, YouTube, and many, many others. It's showing no signs of waining, however, as both Adobe and WordPress patched XSS vulnerabilities as recently as November 2017.

8

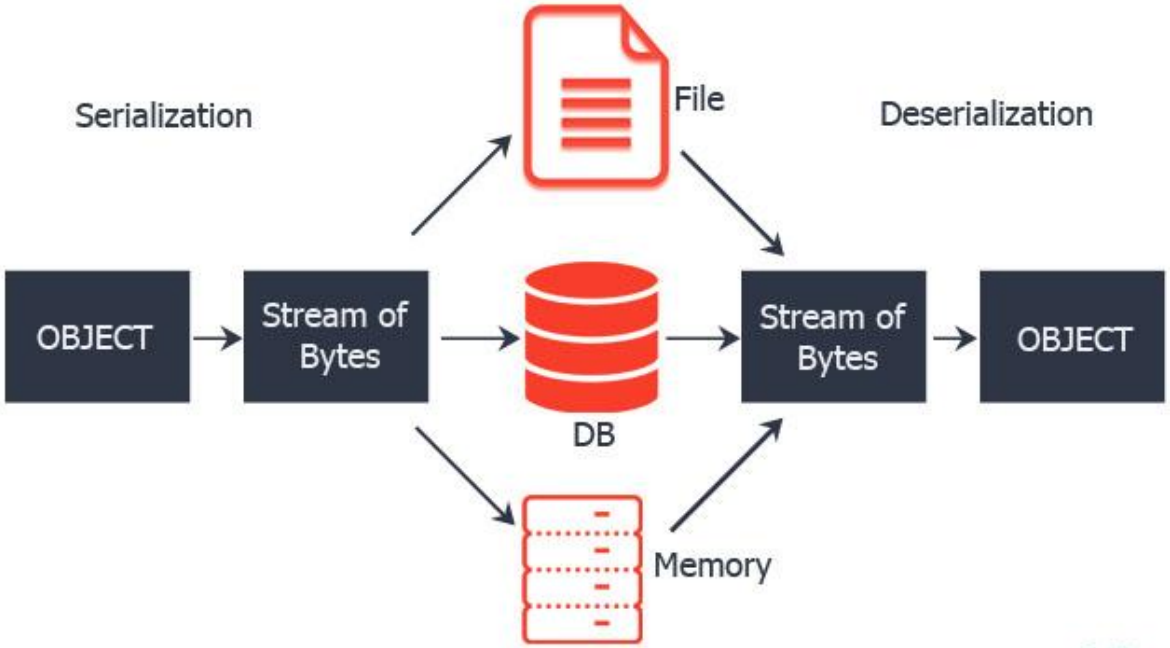
A8-INSECURE DESERIALIZATION

CWE-502

Receipt of hostile serialized objects resulting in remote code execution

* Common Weakness Enumeration

What is Serialization and Deserialization ?



Why do we serialize stuff?

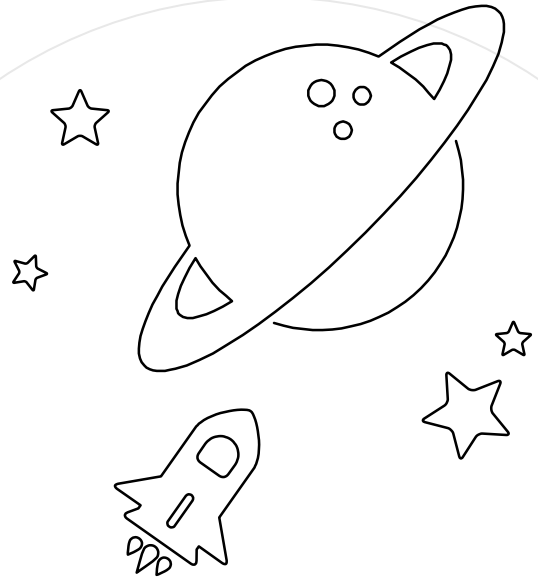
- **Interoperability**
- **Transfer over networks**
- **Storage**

Today, the most popular data format for serializing data is JSON.

How deserialization can be insecure to perform RCE ?

Deserialized data can be modified to include malicious code, which is likely to cause issues if the application does not verify the data's source or contents before deserialization.

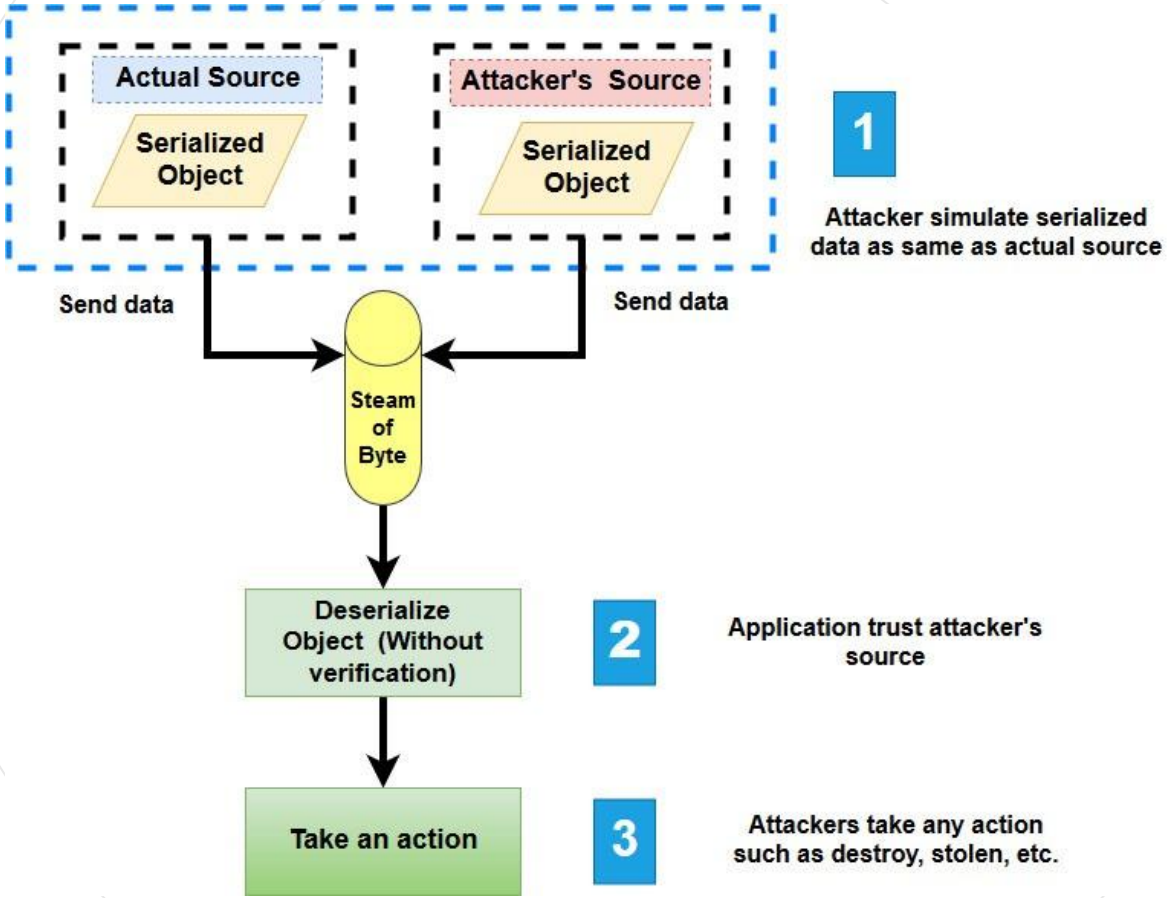
However, many programming languages offer a native capability for serializing objects. These native formats usually offer more features than JSON or XML, including customizability of the serialization process. Unfortunately, the features of these native deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data. Attacks against deserializers have been found to allow denial-of-service, access control, and remote code execution attacks.



Big concept

HOW DOES IT WORK?

Let's simplify the insecure deserialization




DEMO TIME !!!

How to Prevent ?

- The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Implementing integrity checks such as digital signatures
- Monitoring deserialization, alerting if a user deserializes constantly.

#8

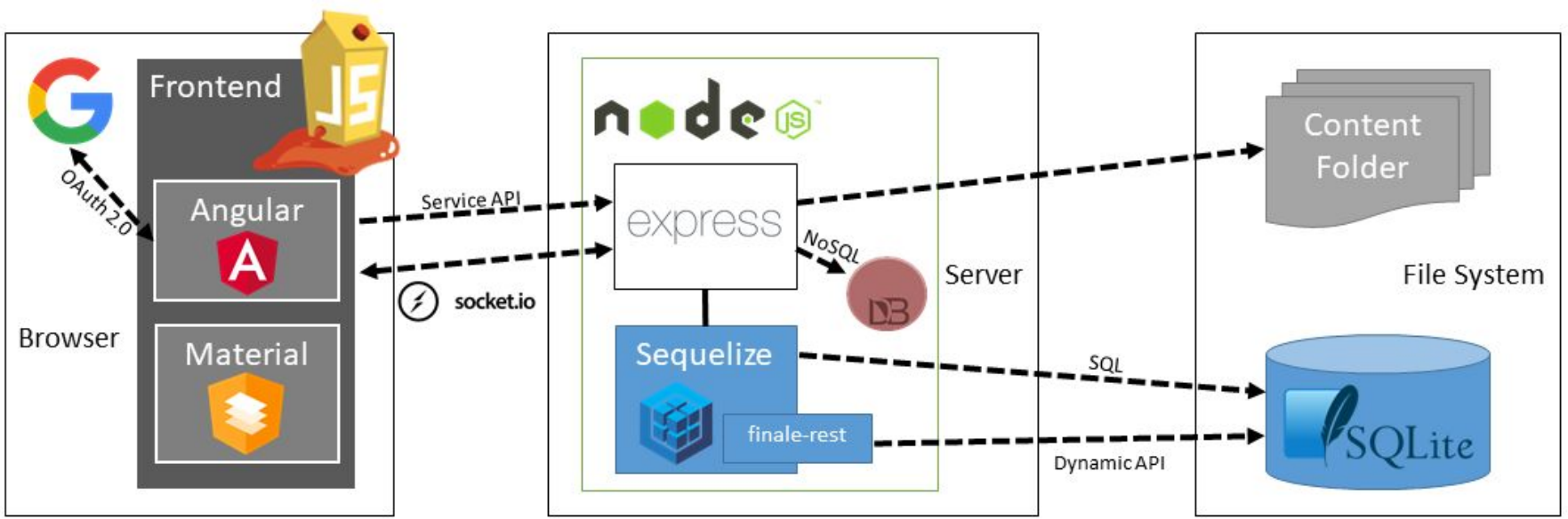
During 2015 and 2016, insecure deserialization was found in so many Java applications, including one at PayPal, the wave of vulnerabilities was dubbed the “Java Deserialization Apocalypse”.

9

A9-USING COMPONENTS WITH KNOWN VULNERABILITIES

Finding and exploiting already-known vulnerabilities before they are fixed

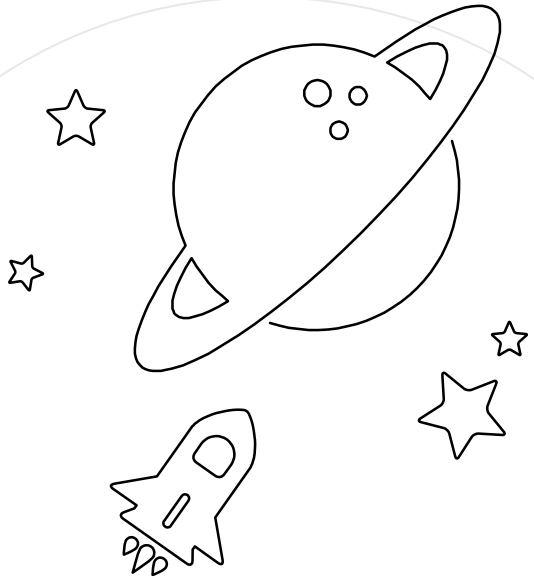
Architecture overview of OWASP Juice Shop



When vulnerabilities become known, vendors generally fix them with a patch or update. The process of updating the software eliminates or mitigates said vulnerability.

Am I Vulnerable to Known Vulnerabilities?

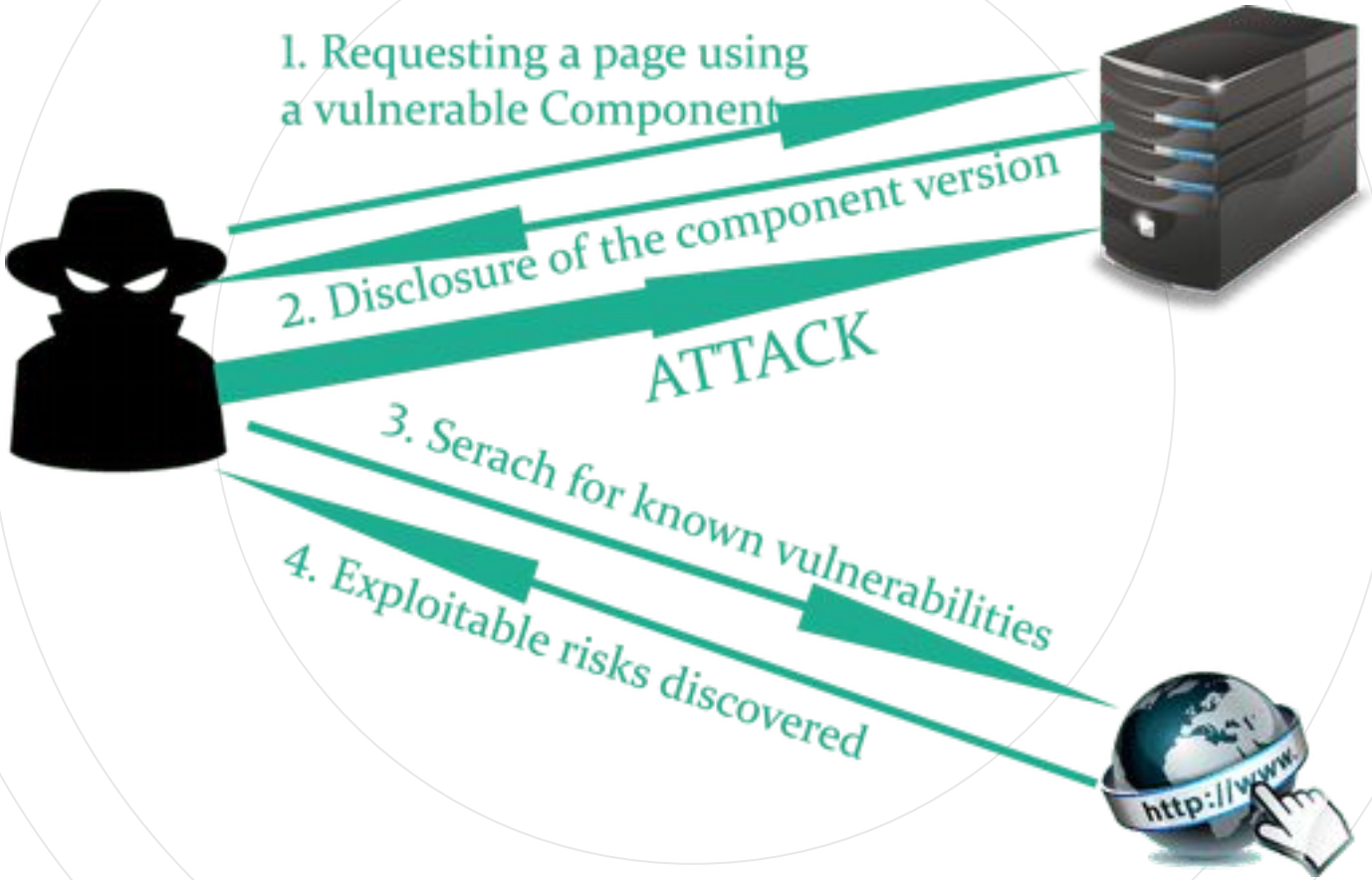
- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If any of your software out of date?
- If you do not know if they are vulnerable?
- If you do not fix nor upgrade the underlying platform,frameworks and dependencies in a timely fashion.
- If you do not secure the components' configurations



Big concept

HOW DOES IT WORK?

How to exploit known vulnerabilities ?




DEMO TIME !!!

How to Prevent ?

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process.
- Continuously inventory the versions of both client-side and server-side components and their dependencies using tools like versions, DependencyCheck, retire.js, etc.

#9

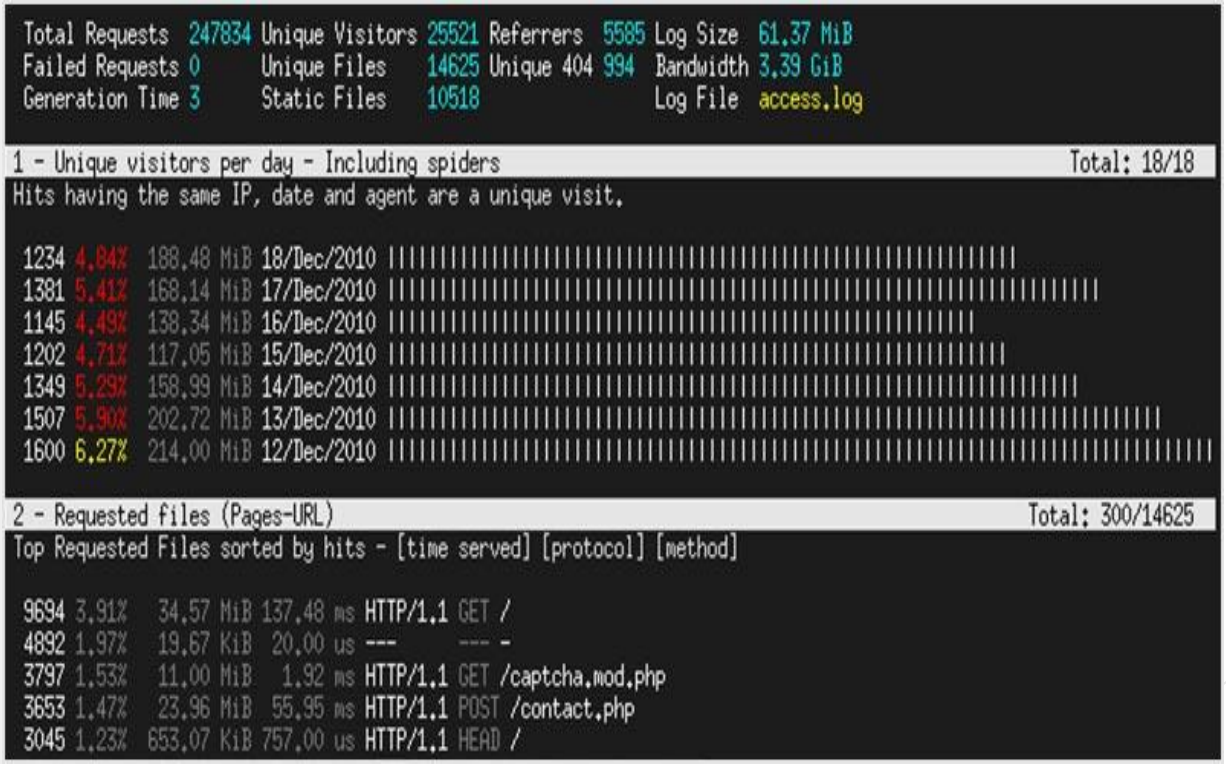
The former CEO of Equifax, while testifying to Congress regarding their infamous 2017 breach, blamed it on someone in IT, stating “The human error was that the individual who’s responsible for communicating in the organization to apply the patch, did not.”

10

A10–INSUFFICIENT LOGGING & MONITORING

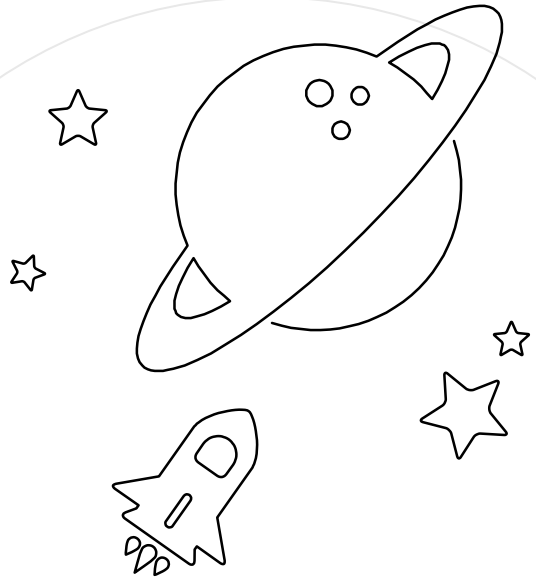
Insufficient monitoring allows attackers to work unnoticed

Logging and Monitoring



Logging-Recording Events as they happen Captures all failures and transactions detecting malicious activity

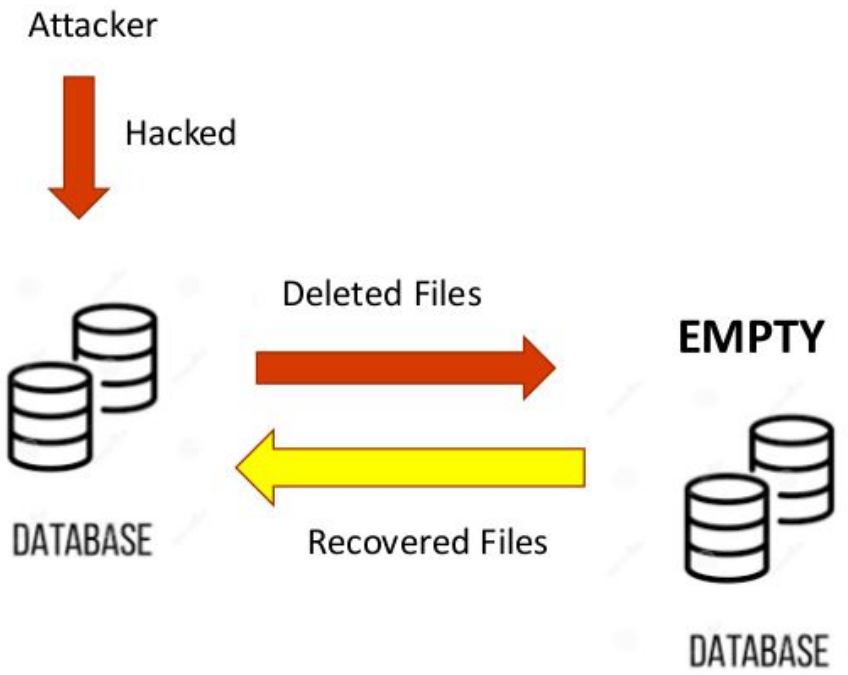
Monitoring-Checking Logs to detect Malicious Activity
Monitoring Log to look for suspicious activity



Big concept

HOW DOES IT WORK?

Scenario #1



- An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version
- Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach


DEMO TIME !!!

How to Prevent ?

- **Ensure all login, access control failures, input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.**

- **Web Defacement and Intrusion Monitoring Tool: WDIMT.**

- **Establish effective monitoring and alerting such that suspicious activities are detected and responded within acceptable time periods.**

#10

Logging isn't just important for identifying attacks in progress; it can assist with the forensic analysis after an attack has succeeded.

Summary

By using the OWASP Top 10 List we saw how to:

- **Define the vulnerabilities**
- **Illustrate the Web Application Vulnerabilities**
- **Explain how to protect against the vulnerabilities.**

Thanks!

Any questions?

You can find me

Jay Patel

OCJA,OCJP,RHCSA,RHCE,CCNA

jay.patel1325@gmail.com

+91-7665293148

github.com/jay13patel

linkedin.com/in/jay13patel