Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістр)

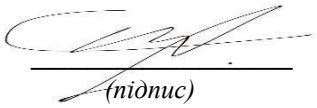
Дослідження тестувань на проникнення та безпеки веб-додатків
(Research of web application penetration testing and security)
(тема)

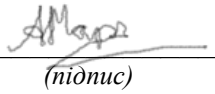
Виконав:
студент 2 курсу, групи АМСЗІім-21-1
Зеггум Суфіан
(прізвище, ініціали)
Спеціальність: 125 Кібербезпека
(код і повна назва спеціальності)
Освітня програма: Адміністративний менеджмент у сфері захисту інформації
of(повна назва освітньої програми)
Керівник: ст.викл. кафедри ІКІ ім. В.В. Поповського
Марчук А.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Лемешко О.В. .
(підпис) (прізвище, ініціали)

2023 р.

Не містить відомостей заборонених до відкритого опублікування

Студент  *Зеггум Суфіан*
(підпис) *(ініціали, прізвище)*

Керівник роботи  *А.В. Марчук*
(підпис) *(ініціали, прізвище)*

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)
Рівень вищої освіти другий (магістр)
Спеціальність 125 Кібербезпека
(код і повна назва)
Освітня програма Адміністративний менеджмент у сфері захисту інформації
(повна назва)

ЗАТВЕРДЖУЮ
Зав. кафедри_____
(підпис)
«____» _____2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Зеггум Суфіан
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження тестувань на проникнення та безпеки веб-додатків (Research of web application penetration testing and security)
затверджена наказом по університету від «_23__» _03___ 2023 р. №_291_ Ст.
2. Термін подання студентом роботи до екзаменаційної комісії _22.05._2023 р.
3. Вихідні дані до роботи: Вимоги стандарту ISO / IEC 27001 A.10.10.2, щодо виявлення атак. Допустимість використання програмного та апаратного забезпечення (стандарт ISO / IEC 27001 A.12.4.1), контроль змін в інформаційній системі (ISO / IEC 27001 A.12.5.1),), а також основні протоколи, що забезпечують функціонування веб-додатків.
4. Перелік питань, що потрібно опрацювати в роботі:
1) Аналіз загроз і вразливостей сучасних веб-додатків.
2) Основні інструменти та методи тестування на проникнення і безпеки для веб-додатків.
3) Дослідження тестування на проникнення і безпеки веб-додатків

5. Перелік графічного матеріалу із зазначенням креслень, плакатів, комп'ютерних ілюстрацій: Демонстраційний матеріал у вигляді ppt-презентації .

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		(підпис)	(дата)
Основна частина	ст.викл. Марчук Артем Володимирович		22.05.23

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	23.03.23	Виконано
2	Збір матеріалів для дослідження	01.04.23	Виконано
3	Розробка 1 розділу	20.04.23	Виконано
4	Розробка 2 розділу	01.05.23	Виконано
5	Розробка 3 розділу	17.05.23	Виконано
6	Оформлення кваліфікаційної роботи	22.05.23	Виконано

Дата видачі завдання 23 березня 2023 року

Студент _____ Зеггум Суфіан
(підпис) (прізвище, ініціали)

Керівник роботи _____ ст.викл. Марчук А.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 77 с., 47 рис., 7 табл., 17 джерел.

ІНФОРМАЦІЙНА БЕЗПЕКА, ЗАГРОЗИ, ВРАЗЛИВОСТІ, ВЕБ-ДОДАТКИ, РИЗИКИ БЕЗПЕКИ, АТАКИ, СКАНУВАННЯ, ТЕСТУВАННЯ НА ПРОНИКНЕННЯ.

Об’єкт дослідження – процес виявлення вразливостей та захист від них веб-додатків.

Предмет дослідження – методи тестування та виявлення вразливостей веб-додатків.

Мета роботи – аналіз найпоширеніших видів атак на веб-додатки, аналіз існуючих методів тестування веб-додатків для виявлення вразливостей та рекомендації щодо захисту веб-додатків.

Методи дослідження – емпіричний аналіз, порівняння.

На сьогоднішній день тема захисту веб-додатків є актуальною, оскільки в наш час життя людей не можливо представити без використання веб-додатків.

У даній кваліфікаційній роботі виконано аналіз найпоширеніших актуальних видів загроз і вразливостей веб-додатків. Розглянуто методи які дозволяють провести тестування веб-додатків для виявлення можливих вразливостей. Проведено аналіз сучасних методів захисту веб-додатків. Виконано моделювання з використанням платформи Damn Vulnerable Web Application (DVWA) та надано рекомендації щодо використання методів захисту веб-додатків.

ABSTRACT

Explanatory note: 77 pp., 47 figures, 7 tables, 17 sources.

INFORMATION SECURITY, THREATS, VULNERABILITIES, WEB APPLICATIONS, SECURITY RISKS, ATTACKS, SCANNING, PENETRATION TESTING.

The object of research is the process of detecting vulnerabilities and protecting web applications against them.

The subject of the research is methods of testing and detecting vulnerabilities of web applications.

The purpose of the work is the analysis of the most common types of attacks on web applications, the analysis of existing methods of testing web applications to identify vulnerabilities, and recommendations for the protection of web applications.

Research methods – empirical analysis, comparison.

Today, the topic of protecting web applications is relevant, because nowadays people's lives cannot be imagined without the use of web applications.

This qualification work analyzes the most common current types of threats and vulnerabilities of web applications. Methods that allow testing of web applications to detect possible vulnerabilities are considered. An analysis of modern methods of protecting web applications has been carried out. Simulations using the Damn Vulnerable Web Application (DVWA) framework are performed and recommendations are provided for the use of web application protection methods.

CONTENTS

LIST OF ABBREVIATIONS, SYMBOLS, UNITS AND TERMS

ALE - Annualized Loss Expectancy
API - Application Programming Interface
B2B – Business to business
CMS - Content Management System
CSRF - Cross-Site Request Forgery
CVE - Common Vulnerabilities and Exposures
DoS - Denial of Service
DDoS - Distributed Denial of Service
DMZ – Demilitarized Zone
FTA - Fault Tree Analysis
HTTPS - Hypertext Transfer Protocol Secure
IaaS - Infrastructure as a Service
LAN – Local Area Network
LFI - Local File Inclusion
NIST - National Institute of Standards and Technology
OWASP - Open Web Application Security Project
PaaS - Platform as a Service
PTES - Penetration Testing Execution Standard
RFI - Remote File Inclusion
SaaS - Software as a Service
SDLS - Software Development Life Cycle
SQLi - SQL Injection
SSL - Secure Sockets Layer
TLS - Transport Layer Security
WAF - Web Application Firewall
WAN – Wide Area Network
XSS - Cross-Site Scripting

INTRODUCTION

The security of online systems and applications has become more important today, since practically everything is connected to the internet. Cybercriminals are always devising new and sophisticated methods to exploit flaws in online systems and apps. As a result, people, corporations, and organizations must take preventative actions to safeguard their internet assets.

The usage of cyber security frameworks is an important way to assuring the security of online systems and applications. Cyber security frameworks offer companies with a set of principles, best practices, and standards to strengthen their cyber security posture. These frameworks assist firms in identifying and mitigating possible security threats, preventing data breaches, and ensuring regulatory compliance.

The National Institute of Standards and Technology (NIST) Cyber security Framework is a well-known cyber security framework. This paradigm provides a flexible, risk-based approach to cyber security risk management. It provides a collection of principles and procedures that may be used by companies to identify, protect, detect, respond to, and recover from possible cyber security issues.

Finally, as our dependence on online systems and applications grows, cyber security has become a critical factor. Cyber security frameworks provide an organized strategy to detecting and managing possible security risks, maintaining regulatory compliance, and safeguarding user data from cyber-attacks. To safeguard the security and privacy of their online assets, organizations must prioritize cyber security and adopt proactive actions.

1 ANALYSIS OF MODERN WEB APPLICATIONS SECURITY THREATS AND VULNERABILITIES

1.1 Overview of modern web application architecture and technologies

When a user accesses a web page, the server responds to the user's request by sending specified data to the browser. To be more precise, a web client (or user agent) may ask a web server for web resources or more widely used online documents (such as HTML, JSON, PDF, and so on). The requested information then materializes after these simple operations. Following that, a user and a website begin interacting.

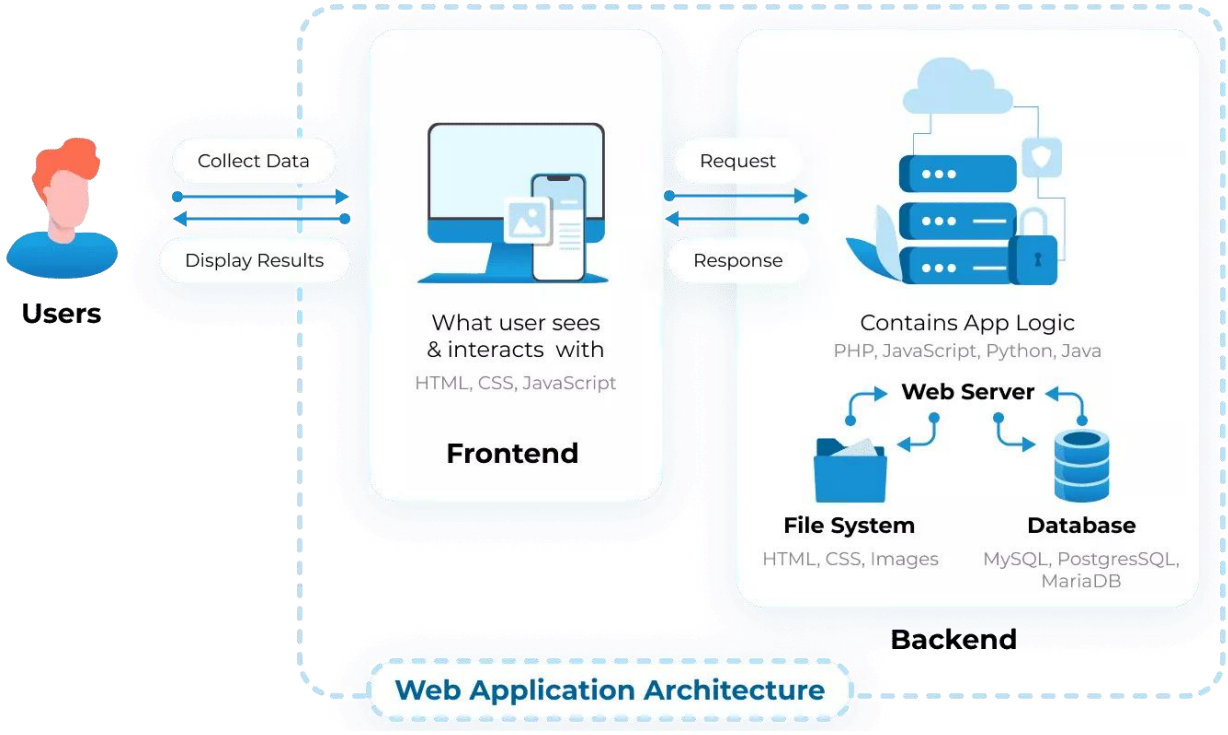


Figure 1.1 - Web application simplified architecture

Simplified diagram of web application architecture was shown on Figure 1.1. It illustrates the overall structure and components of a typical web application. The diagram showcases the interaction between the user, frontend, and backend components.

At the left side of the diagram, we have the user, who interacts with the web application through a user interface, commonly referred to as the frontend. The frontend

encompasses the visual elements, such as web pages, forms, buttons, and other interactive elements that users interact with directly. It provides the means for users to input data, navigate through the application, and view information. For web applications it is usually built on HTML, CSS and JavaScript based on some kind of frontend frameworks.

The frontend side communicates with the backend side of the web application, which handles the processing and management of data and logic behind the scenes. The backend includes several key components that work together to fulfill user requests and deliver the desired functionality.

The first component of the backend is the web server, which receives requests from the frontend and processes them. The web server is responsible for routing the requests to the appropriate components within the application and generating responses that are sent back to the frontend. It serves as the intermediary between the user and the application's functionality.

Connected to the web server, we have the file system, which stores various files and resources utilized by the web application. This includes HTML, CSS, JavaScript files, images, and other static assets required for the proper rendering and functioning of the frontend.

Another integral part of the backend is the database, which serves as a persistent storage system for the application's data. The database stores and organizes structured information, such as user accounts, configuration settings, and content specific to the application. It allows for efficient data retrieval, storage, and manipulation, ensuring the web application can provide dynamic and personalized experiences for users.

Consider the Layers of Web Application Architecture.

Modern web application architecture typically consists of several layers, each responsible for a specific aspect of the application's functionality.

Here's a simplified overview of the architecture and technologies commonly used.

1) **Presentation layer:** This layer handles the user interface and presentation of the application. It typically consists of HTML, CSS, and JavaScript, which are used to create the frontend of the application.

2) **Application layer:** This layer contains the business logic of the application. It processes user requests, communicates with the database, and generates responses. Popular programming languages for this layer include Java, Python, Ruby, and Node.js.

3) **Data layer:** This layer stores and manages the application's data. It can include both SQL and NoSQL databases, such as MySQL, PostgreSQL, Mongo DB, and Redis.

4) **API layer:** This layer provides a standardized interface for communication between the frontend and backend of the application. RESTful APIs are Common use to facilitate this communication.

5) **Cloud infrastructure:** Many modern web applications are hosted on cloud infrastructure, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. These services provide scalable and reliable infrastructure for running and managing web applications.

6) **Containerization and orchestration tools:** Tools like Docker and Kubernetes are often used to manage and deploy applications in a scalable and portable way.

7) **Security:** Modern web applications must be designed with security in mind. Common security measures include authentication and authorization, encryption, and secure coding practices.

Let’s take a look at the architecture diagram and its components.

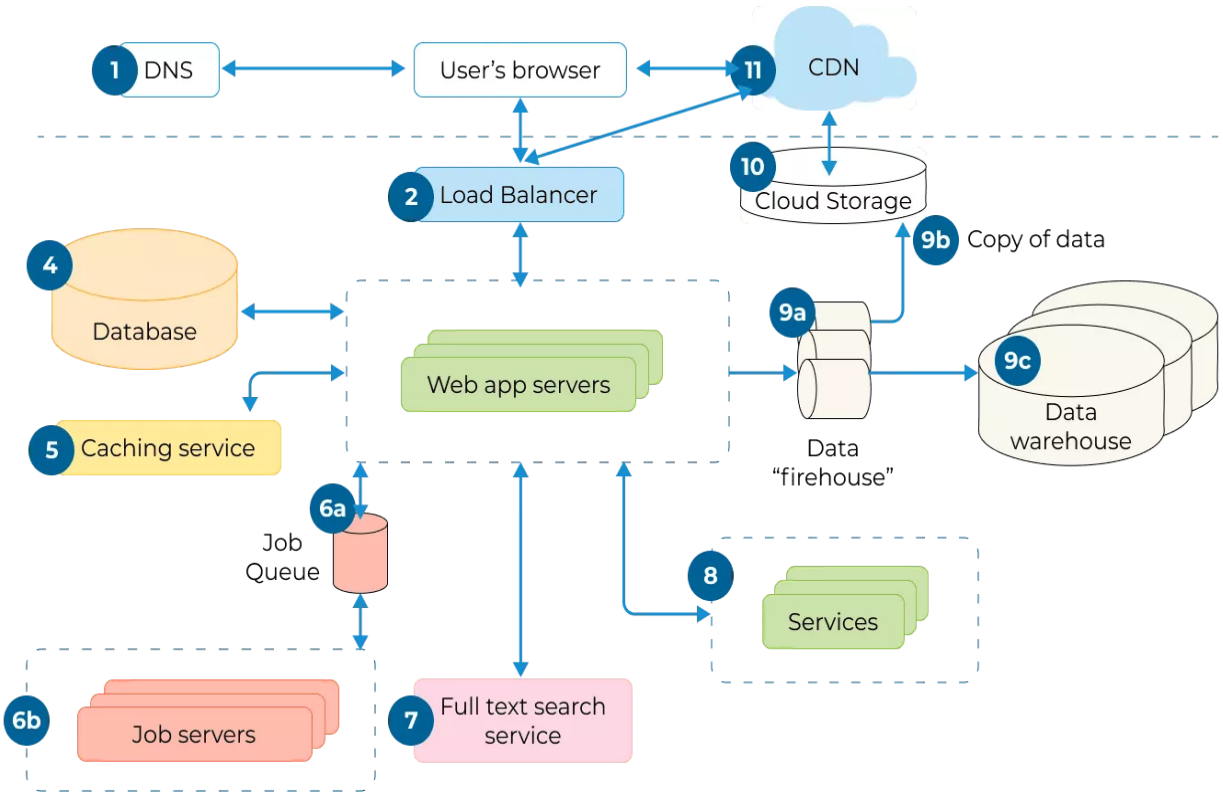


Figure 1.2 - Web application architecture diagram

Consider the elements of architecture.

1) **DNS (Domain Name System).**

DNS, or the Domain Name System, is a critical component of web application architecture. It serves as a distributed naming system that translates human-readable domain names into their corresponding IP addresses. The DNS enables users to access web applications using user-friendly URLs instead of requiring them to remember complex IP addresses. It acts as a fundamental building block by resolving domain names to the IP addresses of web application servers, facilitating the routing of user requests to the correct destination.

2) **Load Balancer.**

A Load Balancer plays a crucial role in distributing incoming web application traffic across multiple servers. Its primary function is to optimize resource utilization, enhance scalability, and ensure high availability. By intelligently distributing requests among multiple web application servers, a load balancer prevents any single server from being overwhelmed, thus improving performance and fault tolerance. Load balancers use various algorithms to determine how requests are distributed, such as round-robin, least connection, or weighted round-robin.

3) **Web App Servers.**

Web application servers are responsible for hosting and executing the code that drives the web application's logic and functionality. They receive incoming requests from users and process them by executing the necessary business logic, accessing databases, and generating responses. Web app servers typically support multiple programming languages and frameworks, enabling developers to build and deploy dynamic web applications.

4) **Databases.**

Databases serve as the central repositories for storing and managing structured data within web applications. They provide a reliable and scalable way to store user data, application configurations, content, and other critical information. Databases are designed to efficiently handle data retrieval, storage, and manipulation operations. They support query languages like SQL (Structured Query Language) and provide mechanisms for data integrity, transaction management, and data consistency.

5) **Caching Service.**

A caching service is an optional component that improves the performance and responsiveness of web applications. It stores frequently accessed data, such as database query results or rendered web pages, in a cache. By retrieving data from the cache

instead of accessing the original data source, the caching service reduces the response time and the load on the backend systems. Caching services employ strategies like expiration policies, cache invalidation mechanisms, and intelligent data fetching to ensure the freshness and consistency of cached data.

6) **Job Queue (optional).**

A job queue is an optional component used to manage and process asynchronous tasks within a web application. It allows the decoupling of time-consuming or resource-intensive tasks from immediate user requests, improving application performance and responsiveness. The job queue receives tasks or jobs from the application and processes them asynchronously, often leveraging background workers or distributed computing resources. Common use cases for job queues include sending emails, generating reports, and performing batch data processing.

7) **Full-Text Search Service (optional).**

A full-text search service is an optional component that enhances the search functionality within web applications. It enables users to search and retrieve relevant information from large volumes of unstructured or semi-structured textual data. Full-text search services employ indexing and search algorithms optimized for efficiency and relevance ranking. They enable features such as fuzzy matching, stemming, and advanced search capabilities to deliver accurate and fast search results.

8) **Services.**

Services encompass a broad category of components that offer specific functionalities or integrations within a web application architecture. Examples of services include authentication and authorization services, payment gateways, third-party API integrations, notification services, and messaging services. Services are designed to provide specialized capabilities that can be utilized across different parts of a web application, promoting code modularity, reusability, and interoperability.

9) **Data Warehouse.**

A data warehouse is a component that facilitates the consolidation and analysis of data from multiple sources within a web application. It provides a centralized repository where data from various databases and systems can be extracted, transformed, and loaded for reporting, business intelligence

10) **CDN (Content Delivery Network).**

A Content Delivery Network (CDN) is an essential component of modern web application architecture. It is a distributed network of servers strategically located in

multiple geographic locations worldwide. The primary function of a CDN is to improve the delivery and performance of web content, such as images, videos, JavaScript files, and CSS stylesheets, to end users.

Overall, modern web application architecture is designed to be scalable, flexible, and secure, using a variety of technologies to provide a robust and reliable user experience.

1.2 Types of modern web application architectures

In addition to traditional architecture, there are several other modern web application architectures that have gained popularity in recent years. These architectures offer different approaches to building web applications, each with its own advantages and use cases. Here are a few examples.

Single-Page Application (SPA) architecture is a type of web application architecture that focuses on delivering a seamless and interactive user experience by loading a single HTML page initially and dynamically updating its content without full page reloads. SPAs rely heavily on client-side rendering and JavaScript frameworks to handle the application logic and data retrieval.

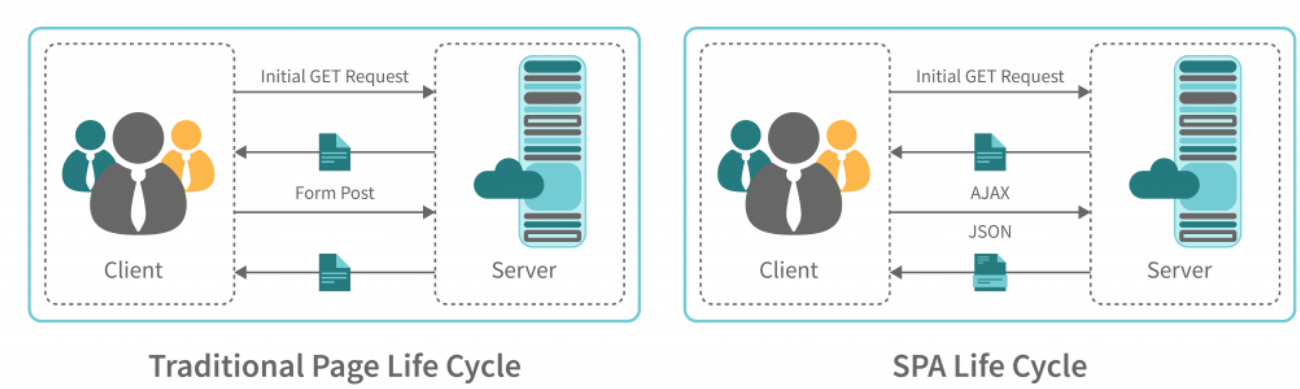


Figure 1.3 - SPA web application

Microservices architecture is a modular approach where an application is divided into small, loosely coupled services that are independently deployable. Each service focuses on a specific business capability and communicates with other services through APIs. Microservices allow for flexibility, scalability, and independent development and deployment of different application components.

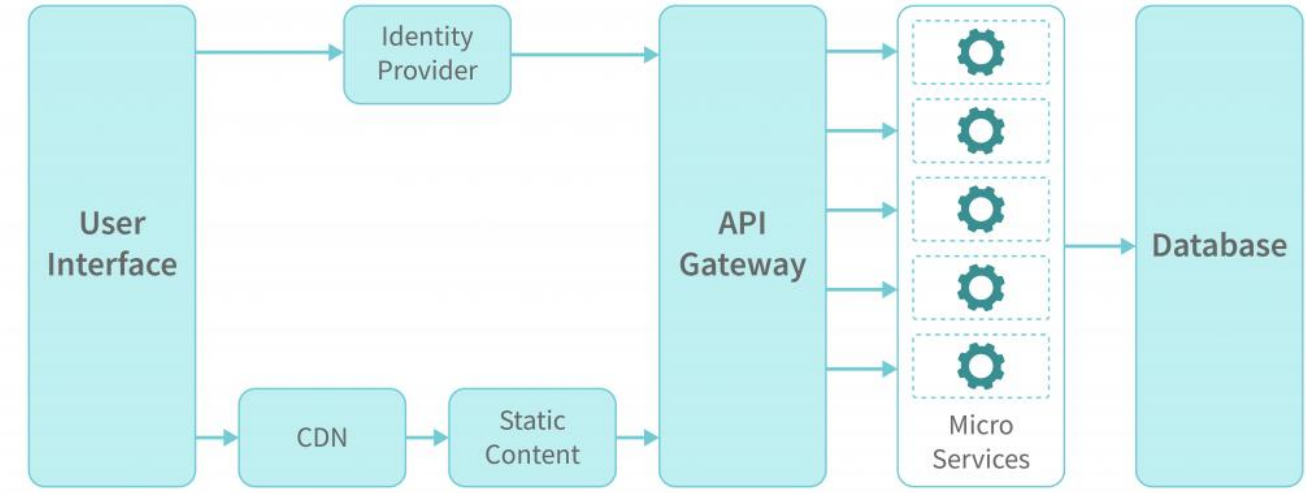


Figure 1.4 - Microservices architecture of web application

Serverless architecture, also known as Function as a Service (FaaS), allows developers to focus on writing code without managing the underlying infrastructure. In this model, applications are built using individual functions or services that are triggered by specific events or requests. The cloud provider takes care of scaling, managing resources, and charging based on actual usage, providing a cost-effective and scalable solution.

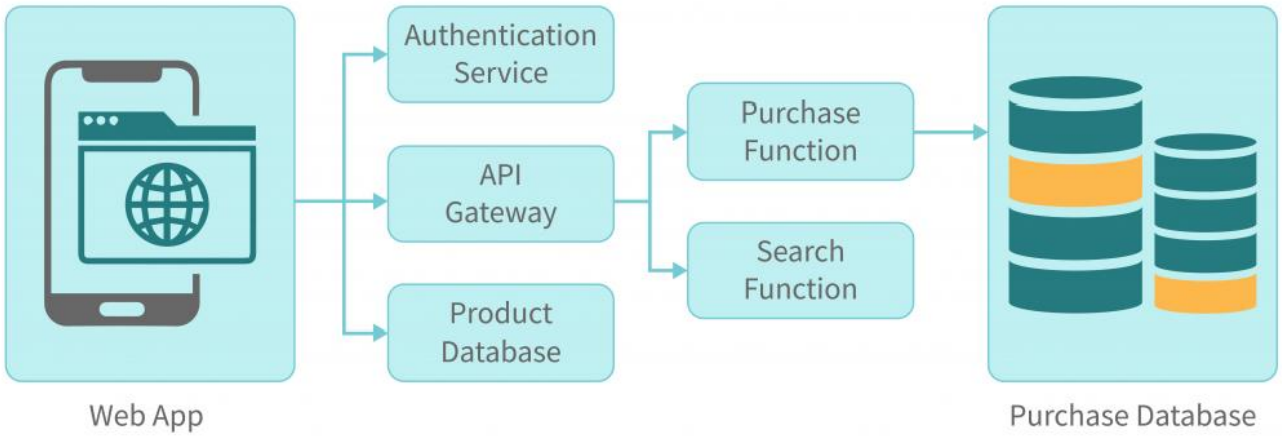


Figure 1.5 - Serverless architecture of web application

Progressive Web Applications combine the capabilities of web technologies with the user experience of native mobile apps. PWAs are designed to be reliable, fast, and engaging, offering features like offline access, push notifications, and device

hardware access. They leverage service workers to cache content and enable offline functionality, providing a seamless experience across different devices and platforms. One of the key features of PWAs is their ability to work offline or with a poor internet connection. They use service workers, a script that runs in the background, to cache essential resources and data. This enables users to access the application and continue using it even when they are offline, providing a seamless experience.

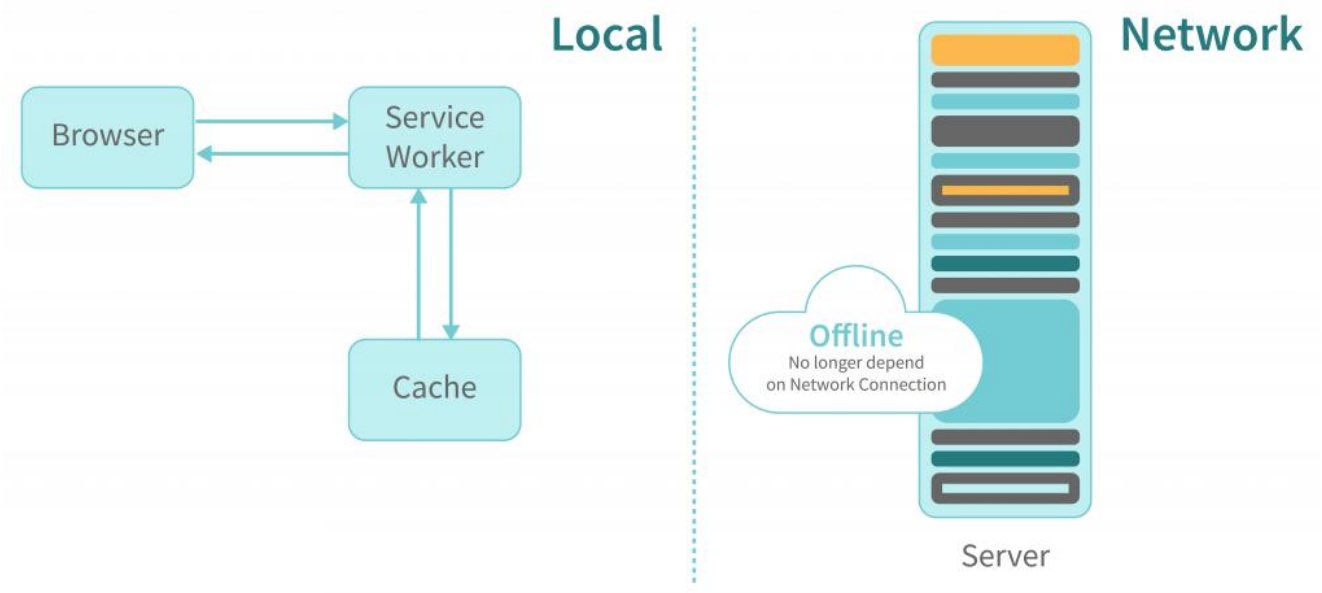


Figure 1.6 - Progressive Web Applications

Choosing the right web application architecture depends on various factors, including the nature of your project, scalability requirements, development team's expertise, timeline, and specific project goals. For example if you anticipate high traffic or expect the need to handle a large number of concurrent users, architectures like Microservices or Serverless can provide scalability and auto-scaling capabilities. Caching mechanisms and Content Delivery Networks (CDNs) can also improve performance and reduce server load. If it's just a simple website with primarily static content, a traditional architecture may suffice.

1.3 OWASP risk rating methodology

The OWASP Risk Rating Methodology is a method for determining the severity of security flaws in web applications. The Open Web Application Security Project

(OWASP), a non-profit organization dedicated to improving software security, built it.

The methodology uses a formula that considers the likelihood of exploitation, the potential impact of the exploit, and the overall severity of the vulnerability. The formula is as follows:

$$\text{Risk Rating} = \text{Probability} \times \text{Impact} \times \text{Severity}. \quad (1.1)$$

Here's an explanation of each of the three components:

Likelihood: The likelihood that the vulnerability will be exploited. It considers factors such as the ease of exploitation, the level of skill required to exploit the vulnerability, and the prevalence of the vulnerability in the code base of the application.

Impact: The potential damage caused by an exploit of the vulnerability is referred to as its impact. It considers factors such as the confidentiality, integrity, and availability of the data or systems that are vulnerable.

The severity of the vulnerability is determined by factors such as the ease of detection, the availability of a fix or workaround, and the level of effort required to exploit the vulnerability.

The resulting risk rating is expressed on a scale of 0 to 25, with 25 being the highest possible risk rating. Here's a breakdown of the risk rating levels:

- 0 - Informational only;
- 1-3 - Low Risk;
- 4-6 - Medium Risk;
- 7-9 - High Risk;
- 10-25 - Critical Risk.

In example on Figure 1.7, the cells in the table represent the overall risk severity levels based on the assigned impact and likelihood values. The severity levels can guide the prioritization of risks, with higher severity risks requiring more immediate attention and mitigation actions.

The OWASP Risk Rating Methodology is a useful tool for prioritizing security vulnerabilities and determining which ones require the most urgent attention. By using this methodology, developers can focus their efforts on fixing the most serious vulnerabilities first, reducing the overall risk of the web application.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Figure 1.7 - Determining overall risk Severity

1.4 Top Security Threats According to the OWASP Security Project

The OWASP Top 10 is a list of the top 10 security threats to online applications. OWASP Top 10 - 2022, the most recent edition, was launched in 2021 and represents the increasing threat environment in web application security.

The following ten threats are included in the OWASP Top 10 – 2022.

1) **Injection:** Injection attacks include introducing malicious code into a web site via user input, such as SQL, command, or script injection attacks.

2) **Broken Authentication and Session Management**: This refers to flaws in a web application's authentication and session management methods, such as weak passwords, session hijacking, and cross-site scripting (XSS) assaults.

3) **Broken Access Control**: This refers to flaws in a web application's access control methods, such as privilege escalation and unauthorized access to sensitive data.

4) **XML External Entities (XXE**): This refers to vulnerabilities in XML data processing, such as XML injection and remote code execution.

5) **Broken Cryptographic Storage**: This refers to flaws in the storage and security of cryptographic keys, passwords, and other sensitive information, such as inadequate key management and weak encryption methods.

6) **Insufficient Logging & Monitoring**: This entails inadequate logging and monitoring procedures, making it harder to notice and respond to security events.

7) **Cross-Site Scripting (XSS):** This refers to vulnerabilities that allow attackers, such as stored and reflected XSS attacks, to inject malicious scripts onto web pages seen by other users.

8) **Employing Components with Known Vulnerabilities**: This entails using software components with known security flaws, such as obsolete and unsupported libraries and frameworks.

9) **Inadequate Security Configuration:** This is caused by incorrectly configured security settings in the online application, such as open firewall ports, weak passwords, and untrusted SSL certificates.

10) **Improper Error Handling and Exception Management**: This concerns the web application's poor error handling and exception management, such as stack traces and extensive error messages, which might disclose critical information to attackers.

OWASP Top 10 2021		change	OWASP Top 10 2022 proposal	
A1	Injections	as is	A1	Injections
A2	Broken Authentication	as is	A2	Broken Authentication
A3	Cross-Site Scripting (XSS)	down 2	A3	Known Vulnerabilities
A4	Sensitive Data Exposure	down 2	A4	Insecure Deserialization
A5	Insecure Deserialization (merged with XXE)	up 1	A5	Cross-Site Scripting (XSS)
A6	Broken Access Control	down 1	A6	Sensitive Data Exposure
A7	Insufficient Logging & Monitoring	down 1	A7	Broken Access Control
A8	NEW: Server Side Request Forgery (SSRF)	down 1	A8	Insufficient Logging & Monitoring
A9	Known Vulnerabilities	up 6	A9	NEW: Server Side Request Forgery (SSRF)
A10	Security Misconfiguration	as is	A10	Security Misconfiguration

Figure 1.8 - OWASP 2021 VS OWASP 2022

Understanding and addressing these OWASP Top 10 security threats is crucial for developers, security professionals, and organizations to build robust and secure web applications. By implementing best practices and applying appropriate security measures, the risks associated with these threats can be significantly mitigated.

Improving security for the OWASP Top 10 security threats involves implementing best practices and adopting appropriate security measures. Here are some suggestions to enhance security for each of the OWASP Top 10 threats.

1) **Injection:**

- utilize parameterized queries or prepared statements to prevent SQL injection and other injection attacks;

- apply input validation and sanitization techniques to filter and validate user input;

- implement strong input validation rules and enforce strict data type checking.

2) **Broken Authentication and Session Management:**

- implement secure authentication mechanisms, such as multi-factor authentication (MFA) and strong password policies;

- use secure session management techniques, such as generating unique session identifiers, setting secure session cookies, and implementing session expiration;

- protect session data from unauthorized access or tampering by employing encryption and secure session storage.

3) **Broken Access Control:**

- implement role-based access control (RBAC) to ensure that users have appropriate privileges and can access only authorized resources;

- use parameterized access control methods to prevent unauthorized access and enforce proper authorization checks;

- regularly review and test access control configurations to identify and fix any mis configurations or vulnerabilities.

4) **XML External Entities (XXE):**

- disable XML external entity processing unless it is explicitly required;

- implement secure XML parsing libraries that mitigate XXE vulnerabilities;

- validate and sanitize XML input to ensure it does not contain malicious entities or external references.

5) **Broken Cryptographic Storage:**

- use strong encryption algorithms and secure hashing methods to protect sensitive data at rest;

- employ proper key management practices, such as using secure key storage and rotation mechanisms;

- encrypt sensitive data before storing it in databases or other persistent storage systems.

6) **Insufficient Logging & Monitoring:**

- implement comprehensive logging mechanisms that capture relevant security

events and activities;

- regularly review and analyze logs to detect suspicious or malicious activities;

- set up real-time monitoring and alerts to quickly identify and respond to security incidents.

7) **Cross-Site Scripting (XSS):**

- implement input validation and output encoding techniques to sanitize user input and prevent XSS attacks;

- use security libraries or frameworks that automatically sanitize user input and escape potentially dangerous characters;

- employ content security policies (CSP) to restrict the execution of untrusted scripts.

8) **Using Components with Known Vulnerabilities:**

- regularly update and patch software components to ensure they are free of known vulnerabilities;

- use software composition analysis tools to identify and manage dependencies with known security flaws;

- monitor vulnerability databases and subscribe to security alerts to stay informed about new vulnerabilities.

9) **Inadequate Security Configuration:**

- follow security best practices and harden the configuration of servers, frameworks, and web application components;

- disable unnecessary services and features to reduce the attack surface;

- regularly perform security audits and vulnerability assessments to identify and remediate configuration weaknesses.

10) **Improper Error Handling and Exception Management:**

- implement custom error messages that provide minimal details and do not disclose sensitive information;

- utilize centralized exception handling mechanisms to ensure consistent and secure error handling;

- implement logging and monitoring of error events to detect and respond to potential security incidents.

It's important to note that addressing these security threats requires a holistic and proactive approach to web application security. Regular security assessments, secure coding practices, employee awareness and training, and ongoing monitoring and

maintenance are essential for maintaining a robust and secure web application environment.

1.5 Standard ISO / IEC 27001 A.14.2.8

Standard ISO / IEC 27001 A.14.2.8 is an information security management standard that specifies the standards for system security testing. The standard requires companies to execute system security testing on a regular basis to assure the confidentiality, integrity, and availability of information handled by the system.

The standard addresses a variety of security testing techniques, such as vulnerability scanning, penetration testing, and code review. Qualified persons or groups should conduct the testing, and the findings should be documented and analyzed to identify areas for improvement.

Organizations must also create and execute a security testing policy and method, which must be evaluated and updated on a regular basis, according to ISO/IEC 27001 A.14.2.8. The purpose of this standard is to guarantee that businesses use a consistent and effective approach to security testing and that their information security posture is constantly improved.

The ISO/IEC 27001 A.14.2.8 standard can help firms demonstrate their commitment to information security while also meeting regulatory and compliance obligations.

2. BASIC PENETRATION AND SECURITY TESTING TOOLS AND TECHNIQUES TO TEST WEB APPLICATIONS

Basic penetration and security testing tools and techniques to test web applications include:

- scanners for Web Applications: Automated programs that scan web applications for known vulnerabilities and security concerns such as SQL injection and XSS. OWASP ZAP, Nessus, and Burp Suite are some web application scanners;

- manual testing involves simulating attacks and evaluating the replies to find vulnerabilities in the web application. This method enables more in-depth testing, including checking for vulnerabilities that automated tools may miss;

- fuzz testing is providing huge volumes of random data to the program in order to uncover any input validation or buffer overflow problems;

- social engineering attacks involve tricking users into disclosing sensitive information or taking activities that might affect the web application's security;

- log analysis is the process of analyzing log files created by the web application and it’s supporting systems to discover security events and evaluate the effectiveness of security measures;

- code Review: Code review is the process of evaluating a web application's source code to find vulnerabilities and security problems.

These tools and approaches should be used in concert to give a comprehensive security evaluation of a web application. It is critical to test web apps on a regular basis to discover and resolve security problems as soon as possible.

2.1 Penetration testing and its role in improving the security of modern web applications

Penetration testing is the process of evaluating a web application's security by simulating attacks and attempting to identify vulnerabilities that an attacker could exploit. It includes a full evaluation of the web application's safety record, which includes testing for known vulnerabilities, weaknesses in access controls, configuration issues, and other issues.

Penetration testing's primary goal is to identify vulnerabilities in web applications before they can be exploited by attackers. Organizations can address security risks and improve the overall security of their web applications by doing so.

Penetration testing can help to improve the security of modern web applications by:

- identifying vulnerabilities: Penetration testing helps in the identification of vulnerabilities in web applications that may have gone undetected during regular security testing, such as vulnerability scanning or code reviews. Penetration testers employ a variety of techniques and tools to simulate attacks and identify potential security flaws in applications;

- assessing the impact of security controls: Penetration testing can also be used to evaluate the efficacy of security controls implemented in a web application, such as firewalls, access controls, and encryption. Penetration testers can identify flaws in these controls and make suggestions for improvements;

- prioritizing security risks: Penetration testing can also assist in the prioritization of security risks based on their severity and potential impact on the web application and the organization. This enables organizations to focus their resources on the most critical problems risks first.

Many compliance standards, such as PCI DSS and HIPAA, require penetration testing to ensure that web applications are secure and meet regulatory requirements.

In conclusion, penetration testing is a critical component of improving the security of modern web applications. Organizations can address security risks and prevent potential security incidents by identifying vulnerabilities and assessing the effectiveness of security controls.

2.2 Penetration testing tools

There are many penetration testing tools available to security professionals to identify vulnerabilities in web applications. Here are some commonly used tools:

Burp Suite is a web application security testing suite that includes a scanner, proxy and intruder. Intruder enables security professionals to intercept and modify HTTP requests and responses, as well as identify and test the effectiveness of security controls.

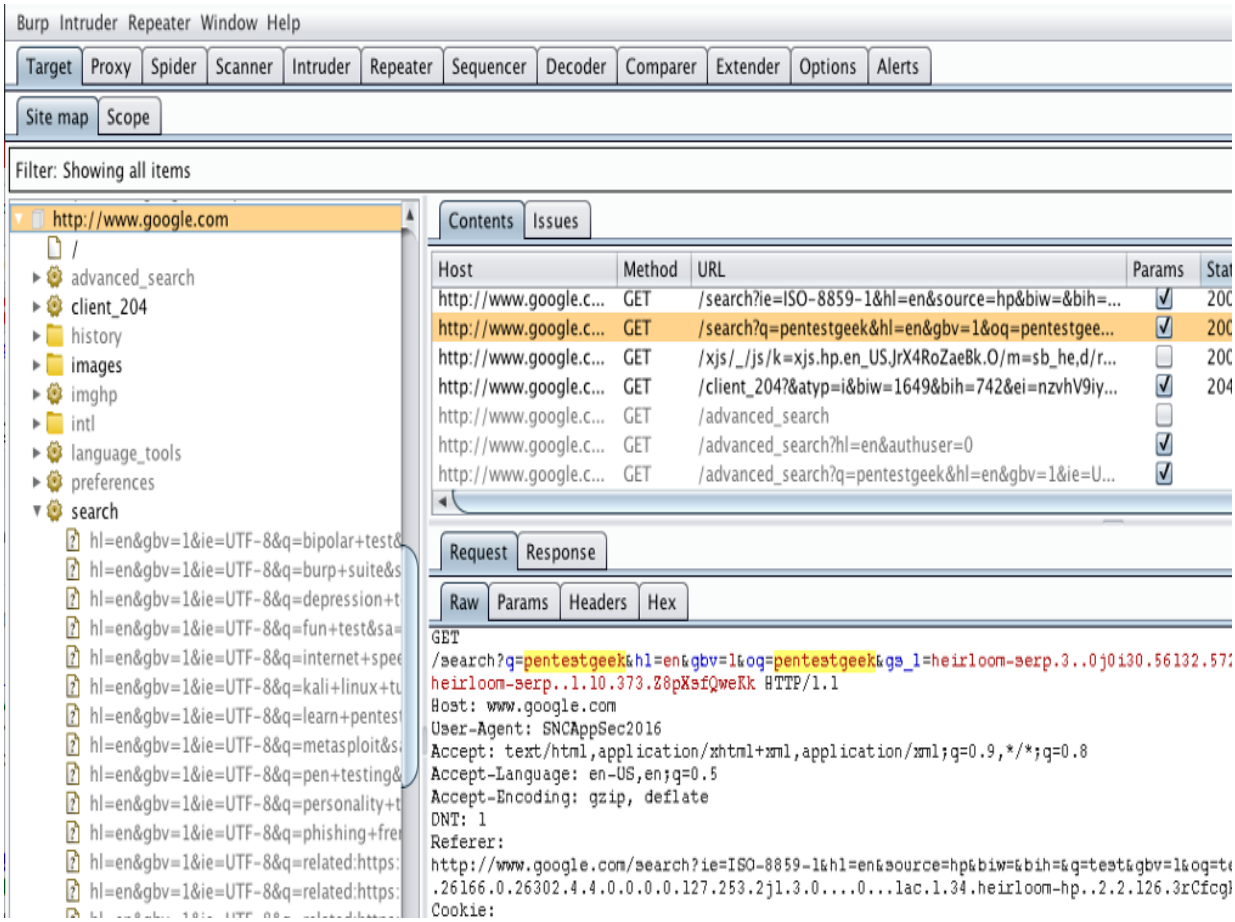


Figure 2.1 - Web application security testing Burp Suite

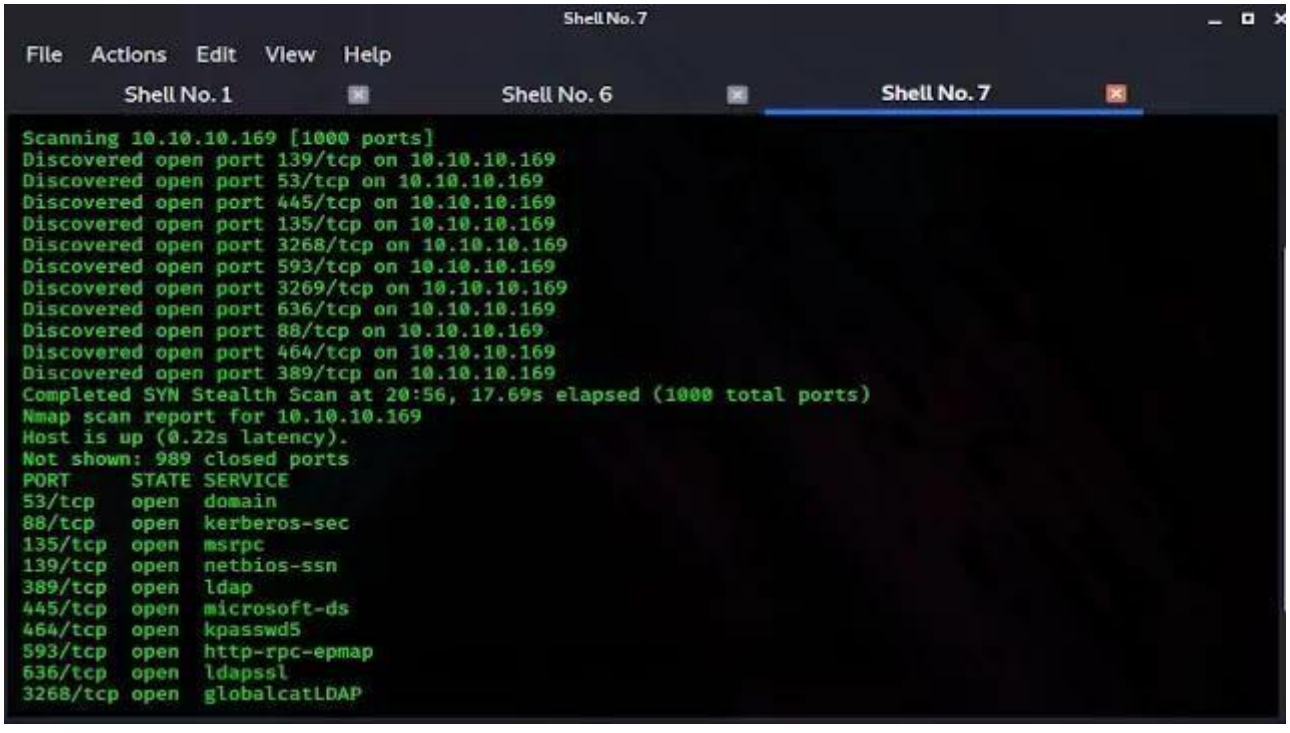


Figure 2.2 - Network scanner Nmap

Nmap is a network scanner that can detect open ports, services, and vulnerabilities in web applications. It can also perform OS fingerprinting and host discovery.

Metasploit is a penetration testing framework that includes a database of known vulnerabilities and exploits. It enables security professionals to test web applications for known vulnerabilities and to automate the exploitation process.

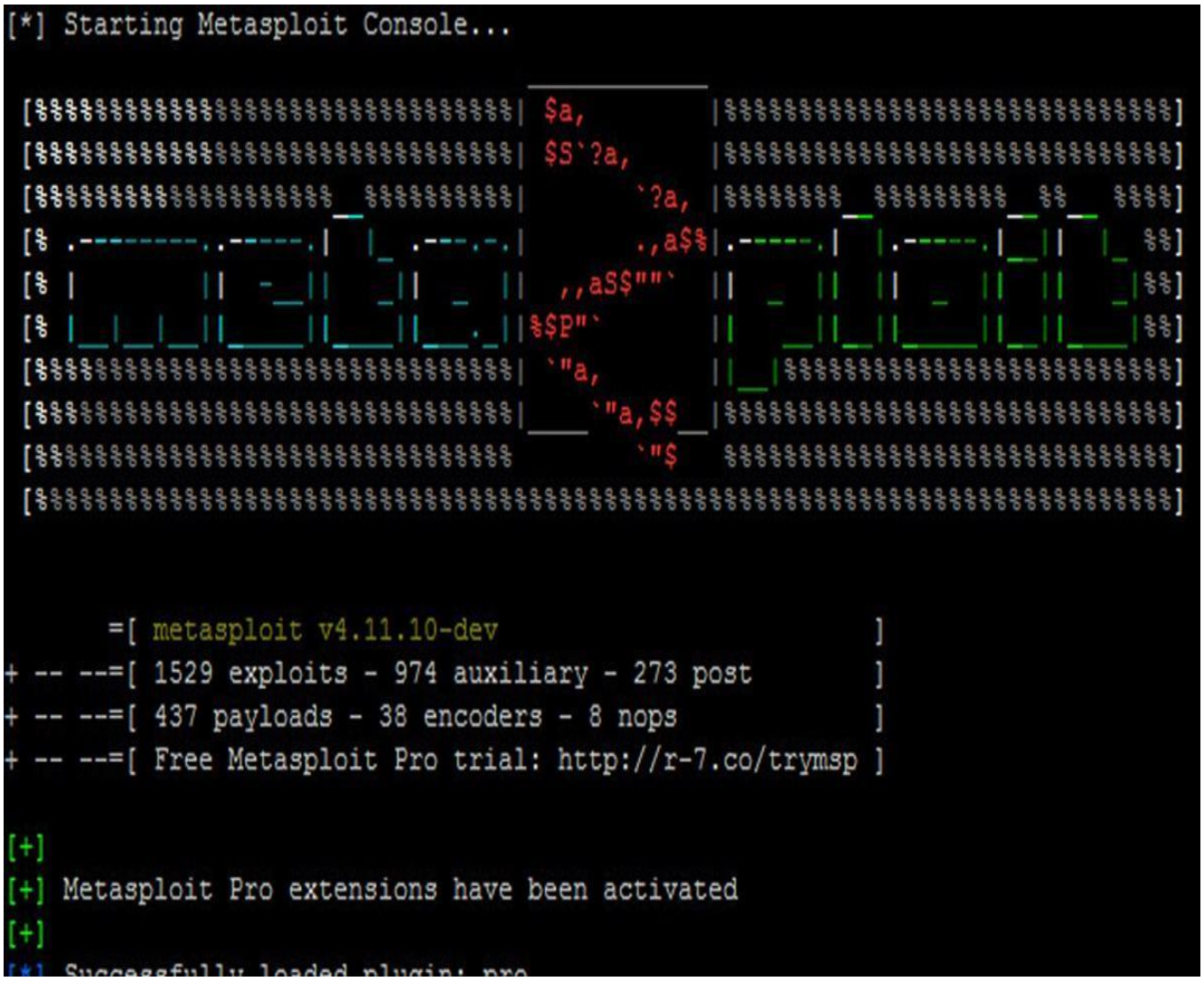


Figure 2.3 - Penetration testing framework Metasploit

OWASP ZAP is an open-source web application security testing tool that includes an intercepting proxy, a scanner, and automated vulnerability testing. Based on the OWASP Top 10 list, it is intended to detect vulnerabilities in web applications.

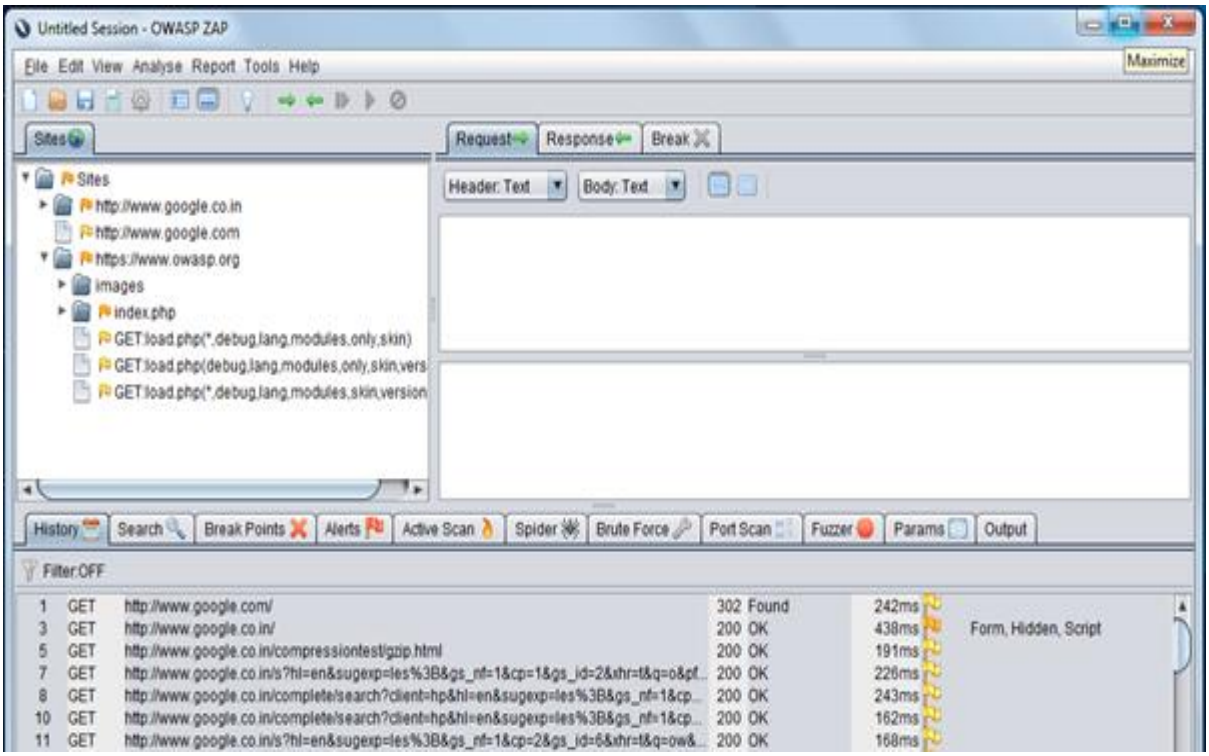


Figure 2.4 - Open-source web application security testing tool OWASP ZAP

Nikto is a web application scanner that looks for known vulnerabilities, misconfigurations, and other security issues on web servers. It is intended to be used in conjunction with other security tools and provides detailed reports on identified vulnerabilities.

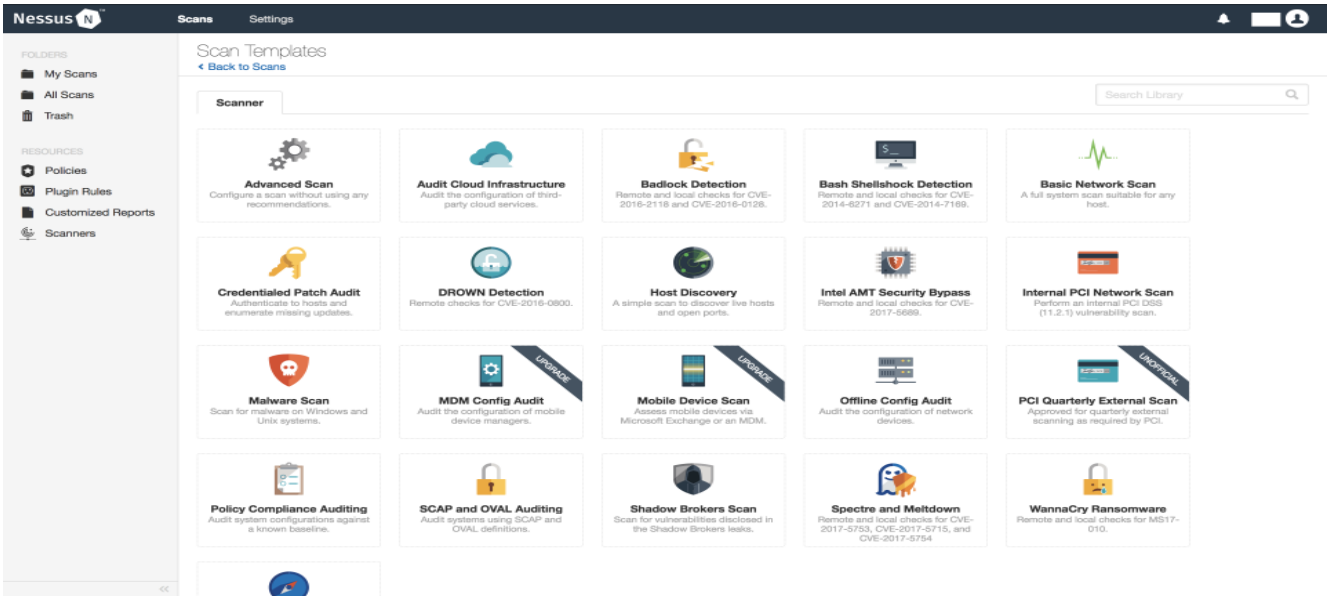


Figure 2.5 – Web application scanner Nessus

Nessus is a vulnerability scanner that can be used to find flaws in web applications as well as other network devices and systems. It includes a large database of known vulnerabilities as well as detailed reports on vulnerabilities discovered.

Aircrack-ng is a tool used for wireless network penetration testing, including testing the security of Wi-Fi networks.

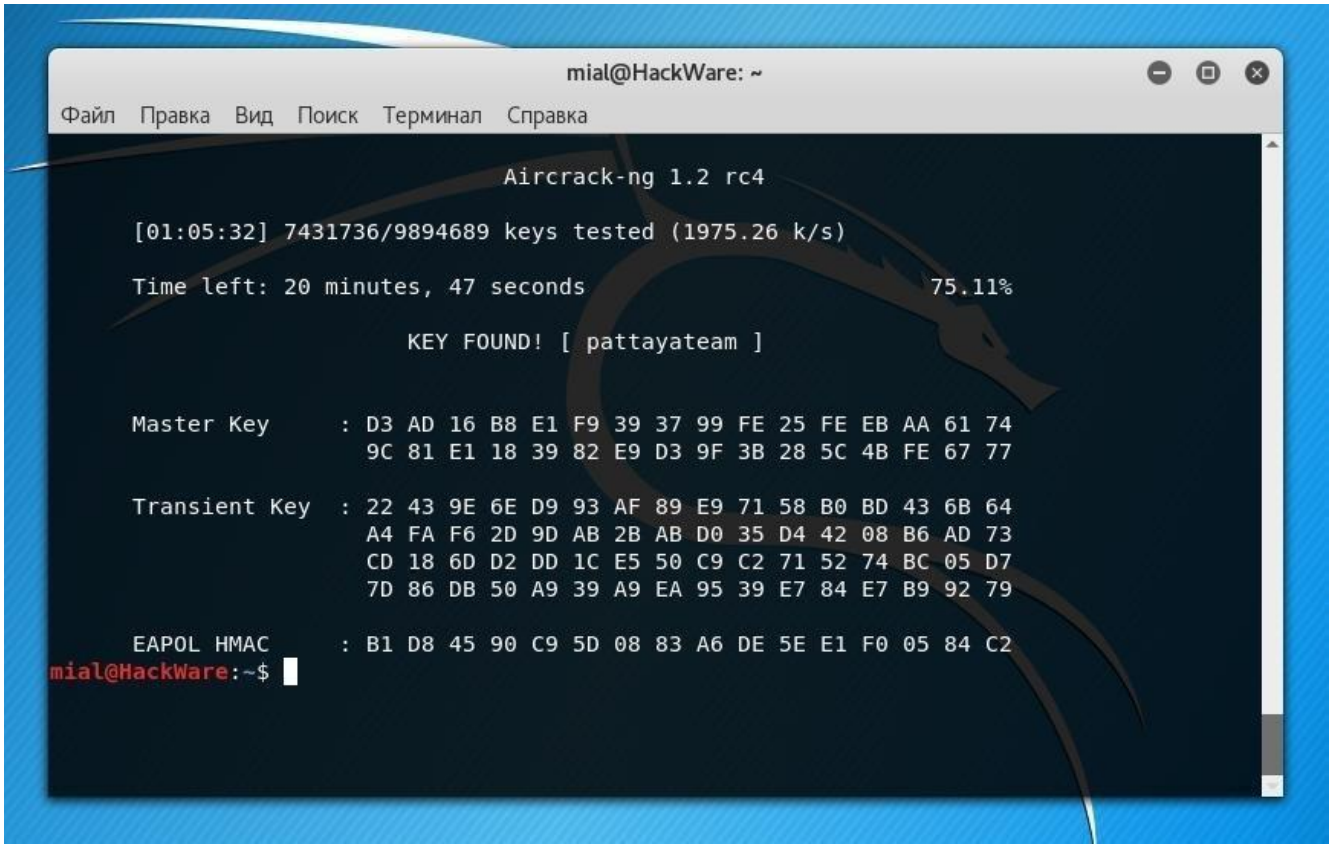


Figure 2.6 - Tool used for wireless network penetration testing Aircrack-ng

Example of steps for a web application penetration testing.

The following are some steps for a web application penetration testing engagement:

2.2.1 Reconnaissance

The tester begins by learning about the target online application and the company that controls it. This might entail looking through the company's website, social media pages, and other publicly available information regarding the application.

There are various tools that can be used for reconnaissance in a penetration testing process, here are some common tools.

1) Shodan: A search engine that allows you to look for internet-connected devices and systems, as well as information on open ports, services, and vulnerabilities.

2) Recon-ng is an open-source reconnaissance program that automates the process of acquiring target information such as DNS information, email addresses, and network topology.

3) The Harvester: A tool for collecting email addresses, subdomains, and other information about a target from search engines, social networks, and other publicly available sources.

4) Maltego is a graphical application for visualizing and analyzing complicated networks, such as social networks, DNS information, and other sources of target information.

5) OSINT Framework: A complete set of open-source intelligence (OSINT) tools and resources for reconnaissance and various phases of penetration testing.

2.2.2 Mapping

The tester maps out the various areas of the program and identifies any access points, such as login pages, user profile pages, and forms, using tools.

Many tools are available for mapping during a web application penetration testing engagement.

1) Burp Suite: Burp Suite is a comprehensive web application security testing toolkit. It comes with a mapping tool that lets you crawl a website and map out all its pages and forms.

2) Nmap is a network scanner that can be used to map all the many ports and services that are operating on a web server. This can be useful for detecting services that utilize non-standard ports or are vulnerable to attack.

3) Sqlmap: SQL map is a tool for automating the detection and exploitation of SQL injection vulnerabilities. It may be used to diagram the various tables and columns in a database.

4) OWASP ZAP: The OWASP ZAP web application security scanner is free and open source. It contains a spider that can explore a website and map out all the different pages, which may be utilized for mapping the web application.

Google Dork is a search phrase that may be used to locate categories of material on the internet. It may be used to locate sensitive material on the internet, such as login sites, backups, or test pages.

2.2.3 Attack simulation

The tester uses a variety of tools and techniques to attempt to attack known application vulnerabilities such as SQL injection and cross-site scripting (XSS). In addition, the tester will employ automated techniques to identify any new vulnerabilities.

An example of steps for a web application penetration testing engagement could be as follows:

- Metasploit is a framework for creating and running exploit code. It may be used to carry out a variety of attacks, including remote code execution and privilege escalation;

- OWASP ZAP: The OWASP ZAP web application security scanner is free and open source. It may be used to do both automatic and manual web application vulnerability scans;

- Vega: Vega is an online vulnerability scanner that can detect a wide range of vulnerabilities, including XSS and SQL injection;

- w3af: w3af is a web application attack and audit framework that may be used to detect and exploit many sorts of vulnerabilities, such as XSS and SQL injection.

2.2.4 Evaluation

The tester examines the attack simulation findings to assess the implications of any discovered vulnerabilities, such as the potential to access sensitive data or take control of the program. In addition, the tester will assess the effectiveness of the application's security safeguards.

During a web application penetration testing engagement, various tools can be used to evaluate the impact of vulnerabilities that have been identified. Some popular tools include:

- Metasploit. By executing exploit code, Metasploit may be used to assess the effect of vulnerabilities. This allows you to test the application's response to various forms of assaults to evaluate the possible effect of a vulnerability;

- Nessus. Nessus is a vulnerability scanner that may be used to do automated vulnerability assessments to assess the impact of vulnerabilities. It may offer specific information about the found vulnerabilities, including the possible effect of each issue;

- Nmap. Nmap may be used to assess the impact of vulnerabilities by mapping out the many ports and services that are active on a web server. This can be useful for detecting services that utilize non-standard ports or are vulnerable to attack.

2.2.5 Reporting

The tester will then write a report that details the vulnerabilities discovered, their effect, and suggestions for remedy. This report will subsequently be delivered to the entity in charge of the online application.

It is crucial to remember that the testing process may include additional phases or be more detailed depending on the size, complexity, and security needs of the application. Furthermore, testing should be done ethically and legally with the website's consent.

2.3 Limitations of penetration testing

Penetration testing (PT) is one of the most widely used techniques for identifying vulnerabilities in networks and systems.

However, the effectiveness of PT as a security assessment method is limited by several factors, such as the complexity of network infrastructure, the expertise of the tester, and the scope of the test. Paper [9] presents a survey and analysis of the limitations of PT in network security, based on a review of the relevant literature. The paper identifies six major limitations of PT: 1 - limited scope of testing, 2 - dependence on manual techniques, 3 - lack of visibility into the underlying system, 4 - insufficient knowledge of attacker techniques, 5 - limited ability to detect zero-day vulnerabilities, and 6 - the dynamic nature of the threat landscape.

The paper [9] discusses each of these limitations in detail and proposes some possible solutions to address them.

1) Limited testing scope: PT has limitations in terms of testing scope since it is unable to protect all potential attack scenarios and vulnerabilities.

2) Reliance on manual techniques: Physical therapy is heavily reliant on manual techniques, which can result in errors and errors in the testing process.

3) Lack of visibility into the underlying system: Because PT has limited visibility into the underlying system, false positives and false negatives can occur.

4) Inadequate knowledge of attacker techniques: PT testers may have little actual knowledge of attacker techniques, limiting their ability to identify vulnerabilities.

5) Restricted ability to detect zero-day vulnerabilities: PT may be limited in its ability to detect zero-day vulnerabilities, which are vulnerabilities that are unknown to the public because there are no patches or fixes available.

6) The threat landscape's dynamic nature: The threat landscape is constantly evolving, making it difficult for PT to keep up with the latest threats and vulnerabilities.

3 RESEARCH OF WEB APPLICATION PENETRATION TESTING AND SECURITY

3.1 Penetration testing and Web Application Security research based on the DVWA platform

DVWA (Damn Vulnerable Web Application) is a web application that is intentionally designed to be vulnerable to various types of attacks. It is commonly used as a training tool for learning about web application security.

Here are some of the vulnerabilities that can be found in the DVWA lab.

3.1.1 Brute Force attack simulation

This vulnerability occurs when an attacker can use automated tools to guess a user's password by trying different combinations of characters.

Here an example of a brute force attack.

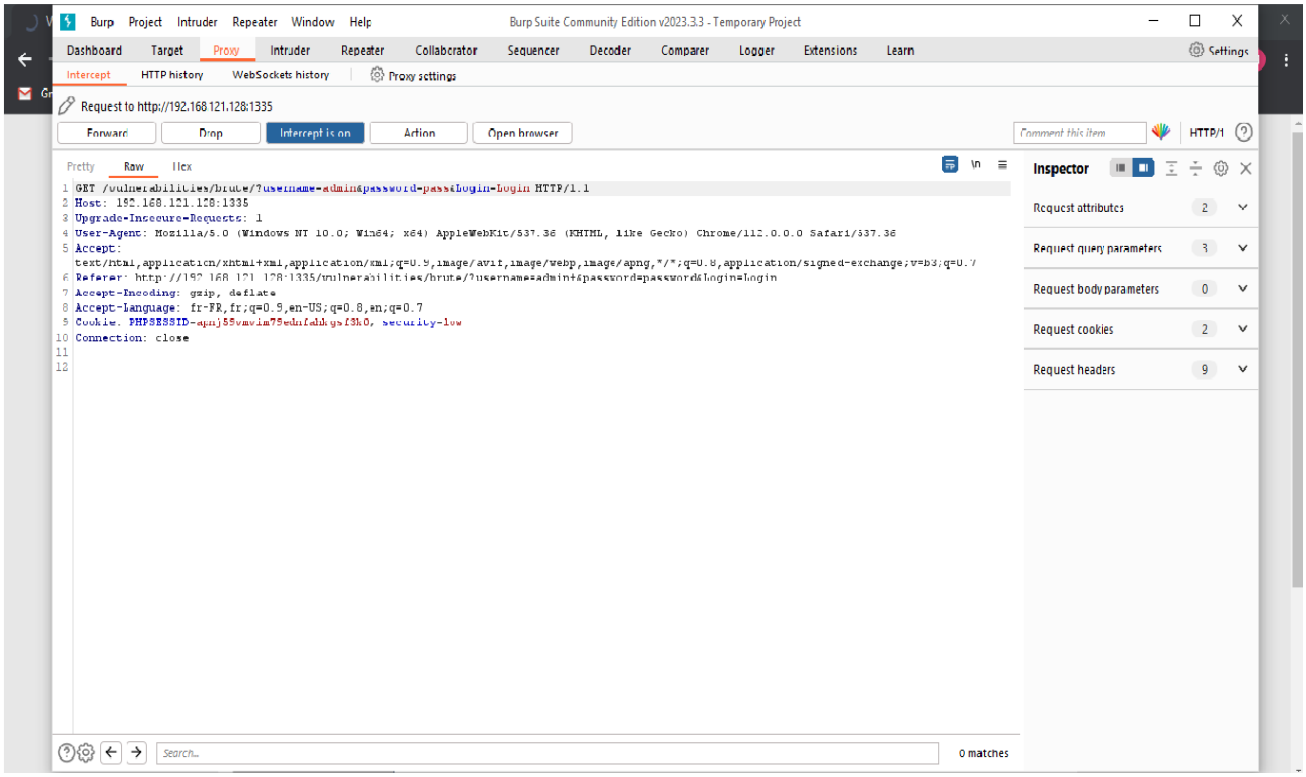


Figure 3.1 - Intercepting in burp suite

Any requests made from our browser will be intercepted by the proxy server if interceptor is enabled. The request can then be inspected, modified, dropped, or forwarded.

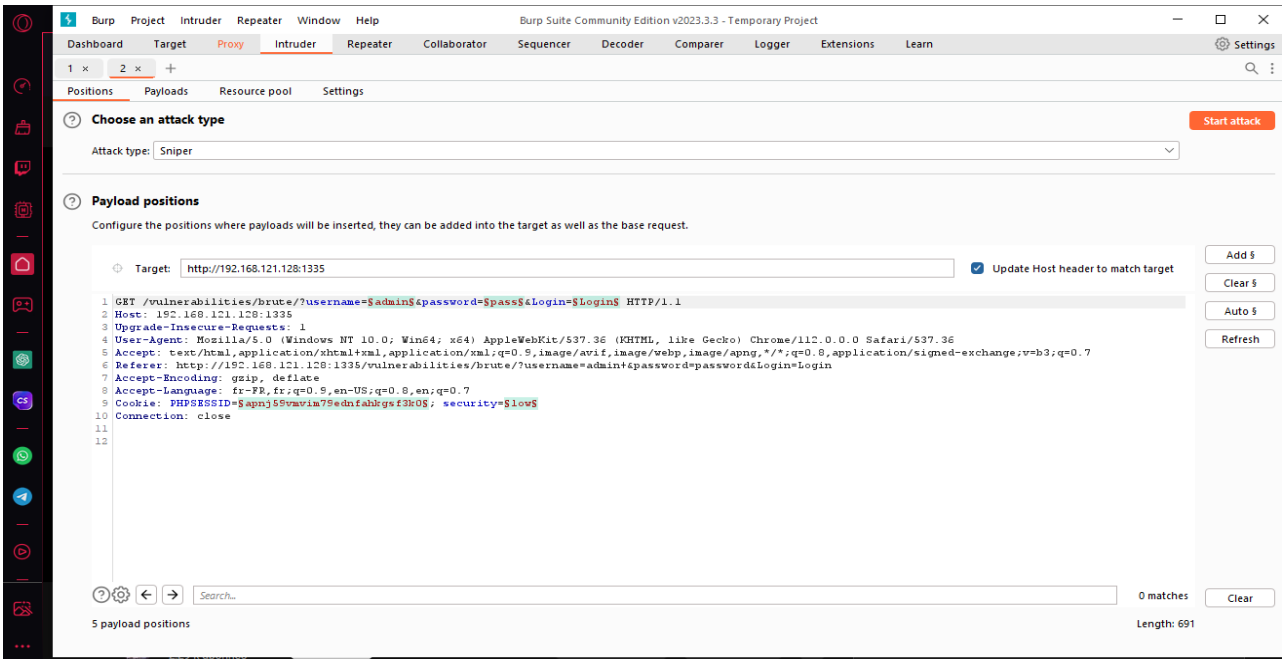


Figure 3.2 - Send it to intruder

Change the attack type to a Cluster bomb by clicking on the "Intruder" tab now, click the clear button and enter only the username and password.

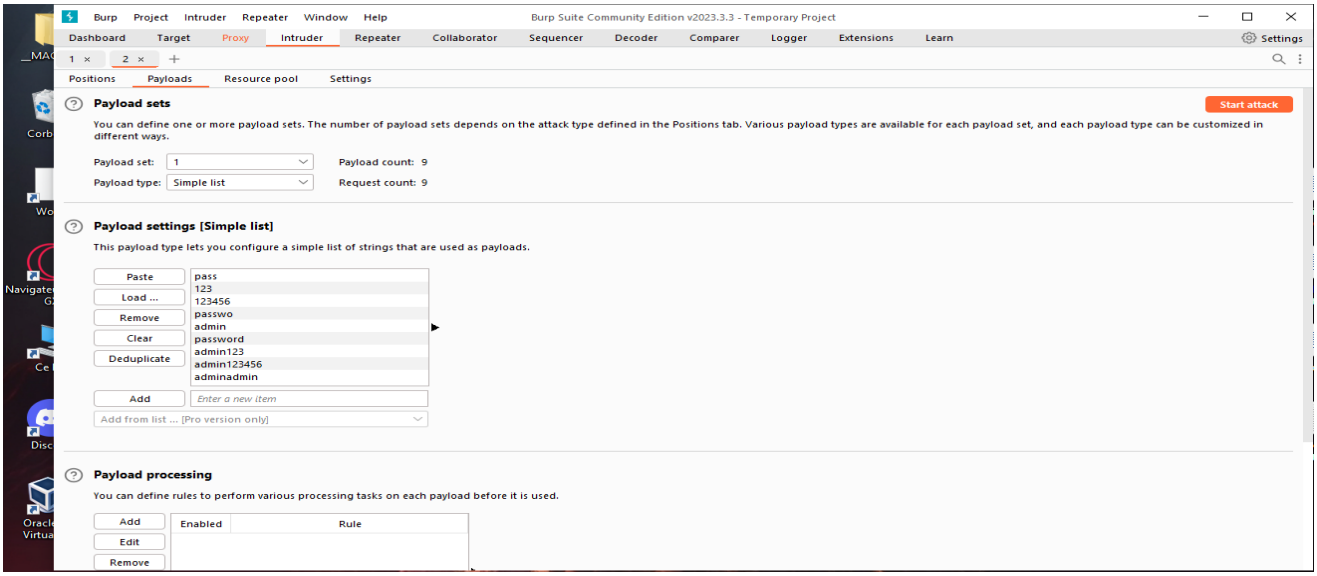


Figure 3.3 - Setting the payloads

Navigate to the Payloads tab in the Intruder tab. Fill in the blank field with some possible usernames.

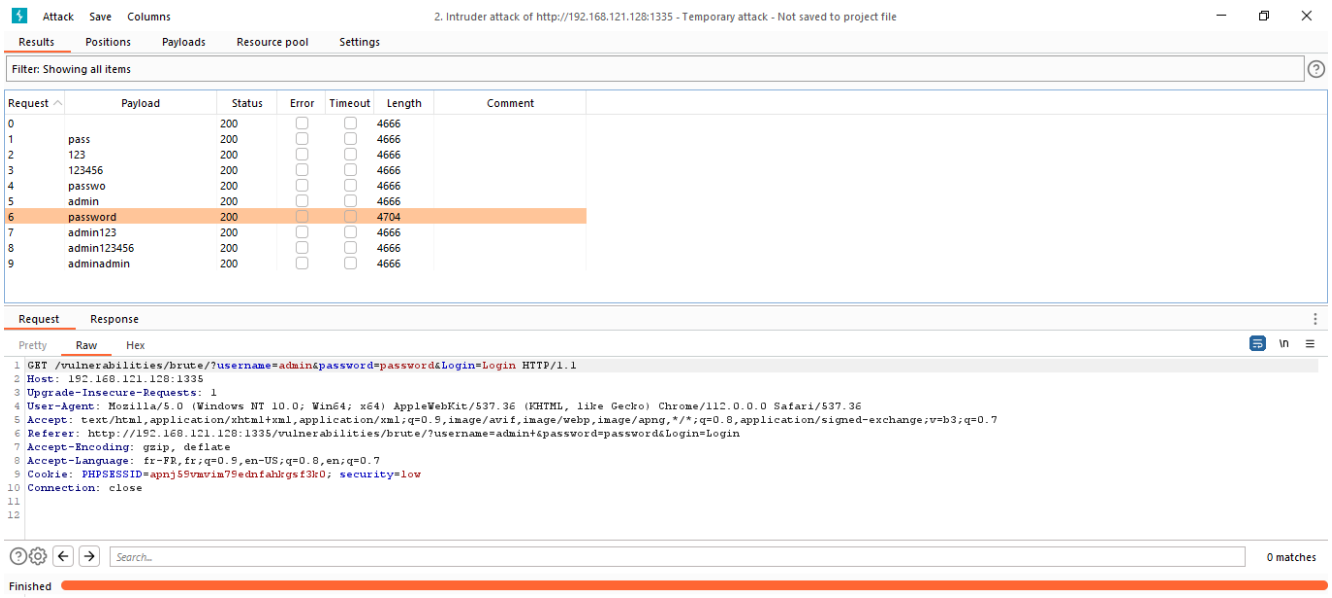


Figure 3.4 - Brute force attack

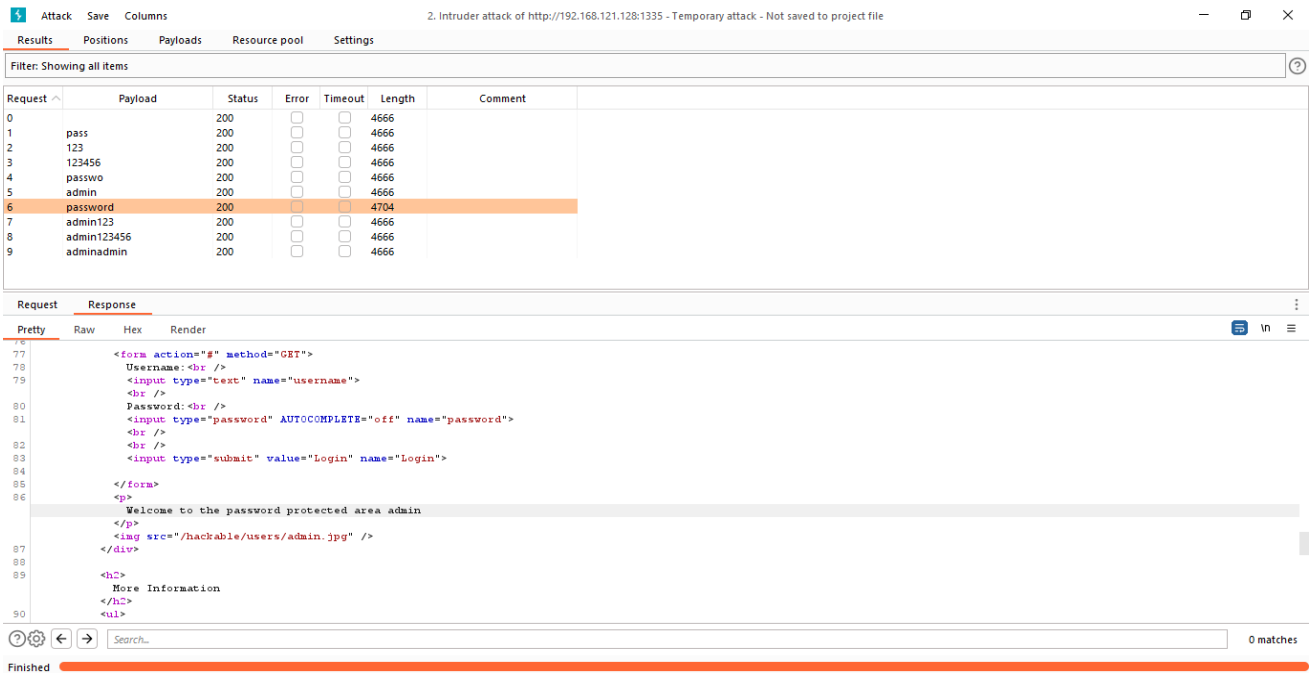


Figure 3.5 - Response for the password

The user is now logged in. As a result, the attack was successful.

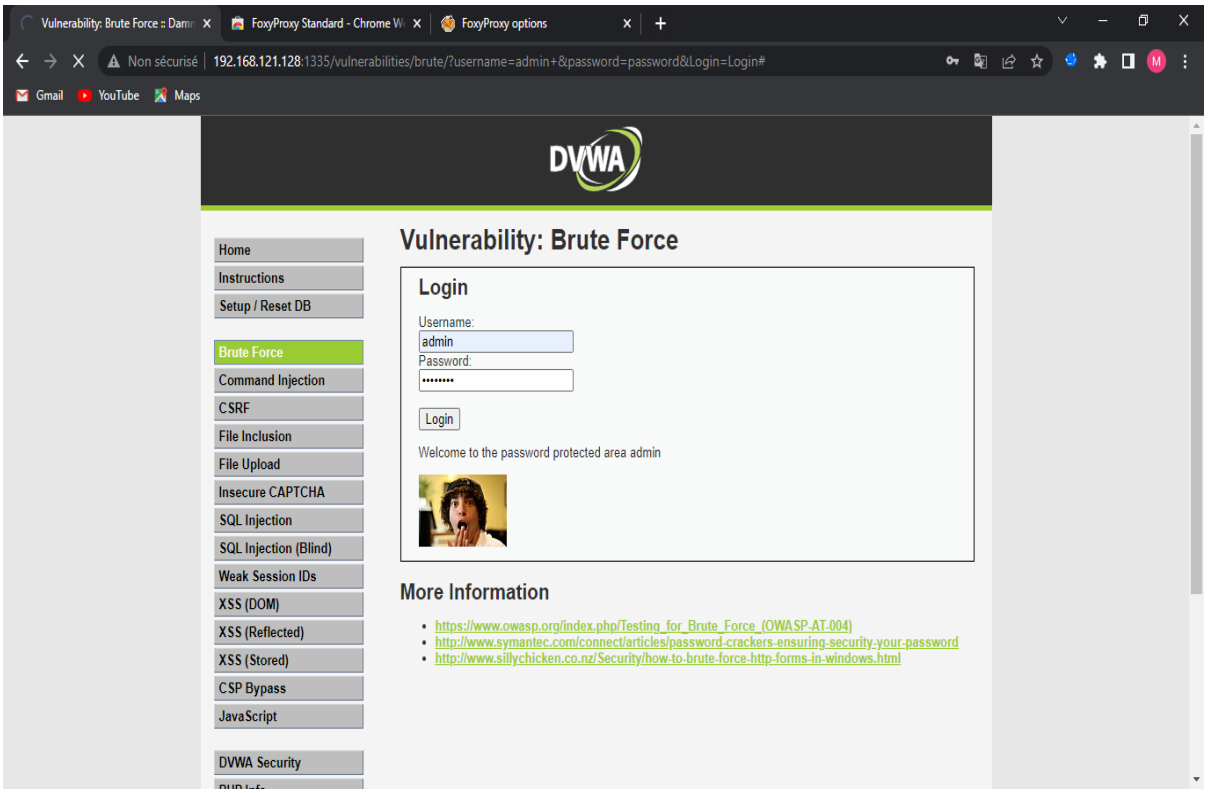


Figure 3.6 - Password Set

Problems brought on by Brute Force Attack:

- To gain access to the account, the attacker can use a variety of username and password combinations;

- if the attacker obtains the username and password, the personal data can be obtained.

The following recommendations can be made to reduce the impact of attacks:

- use Secure Passwords;

- make use of CAPTCHAs;

- implement Two-Factor Authentication (2FA);

- rename the default username 'admin' and replace it with a strong username.

3.1.2 Simulating SQL Malicious Code Injection

This is a vulnerability where an attacker can inject malicious SQL code into a web application's input fields. This can lead to unauthorized access to the application's database or the ability to execute arbitrary SQL commands.

We are now going to perform a manual SQL Injection attack on the DVWA page to obtain information about the database and the information that it contains regarding the column headings, to work out where the user information is sitting.

Step 1.

Click on the Tab for SQL Injection.

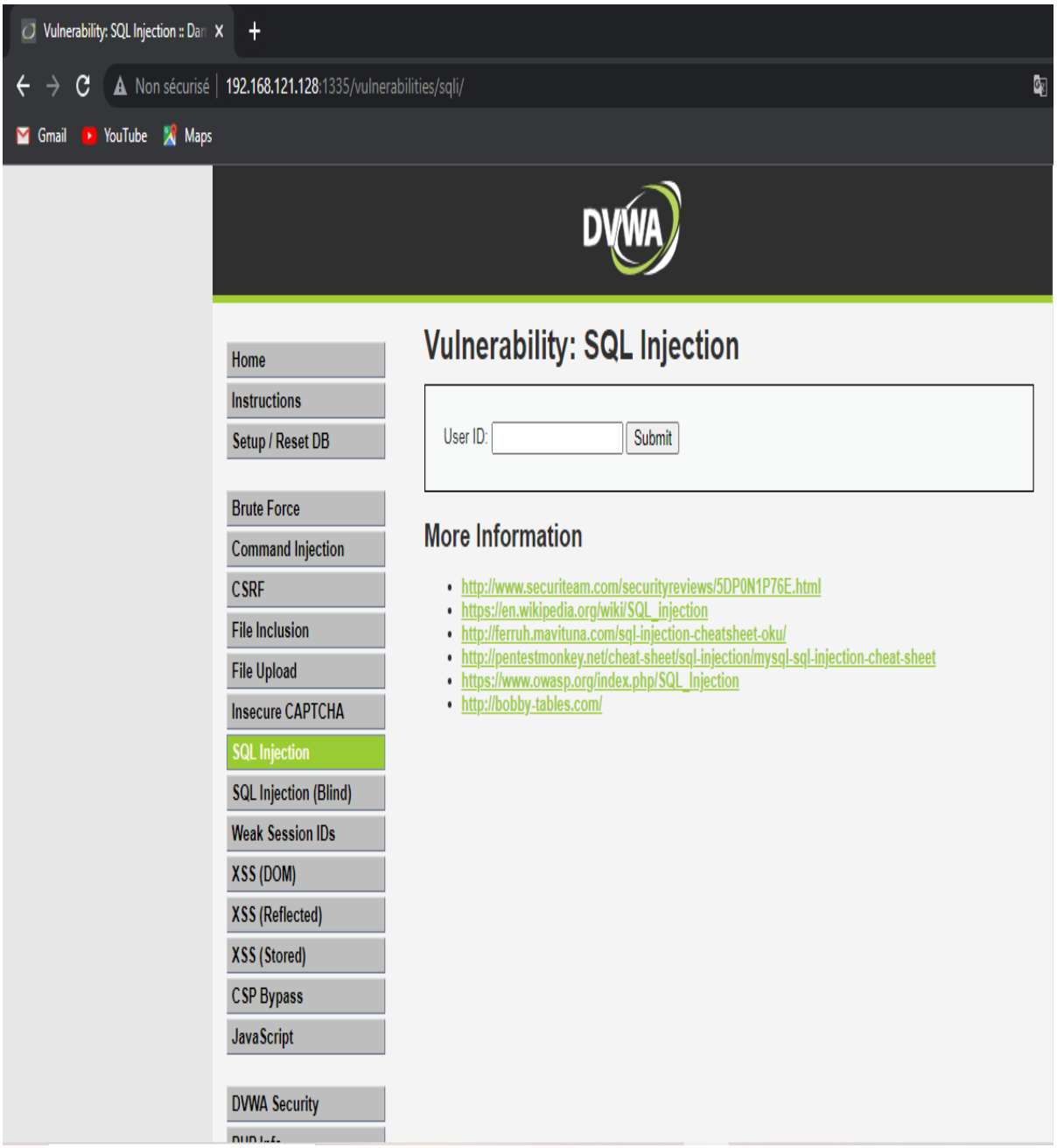


Figure 3.7 - Screenshot DVWA SQL Injection

Now we'll enter some specifics.

Step 2.

A space for entering a User ID appears. Let's start with some basic credentials. User ID: 2.

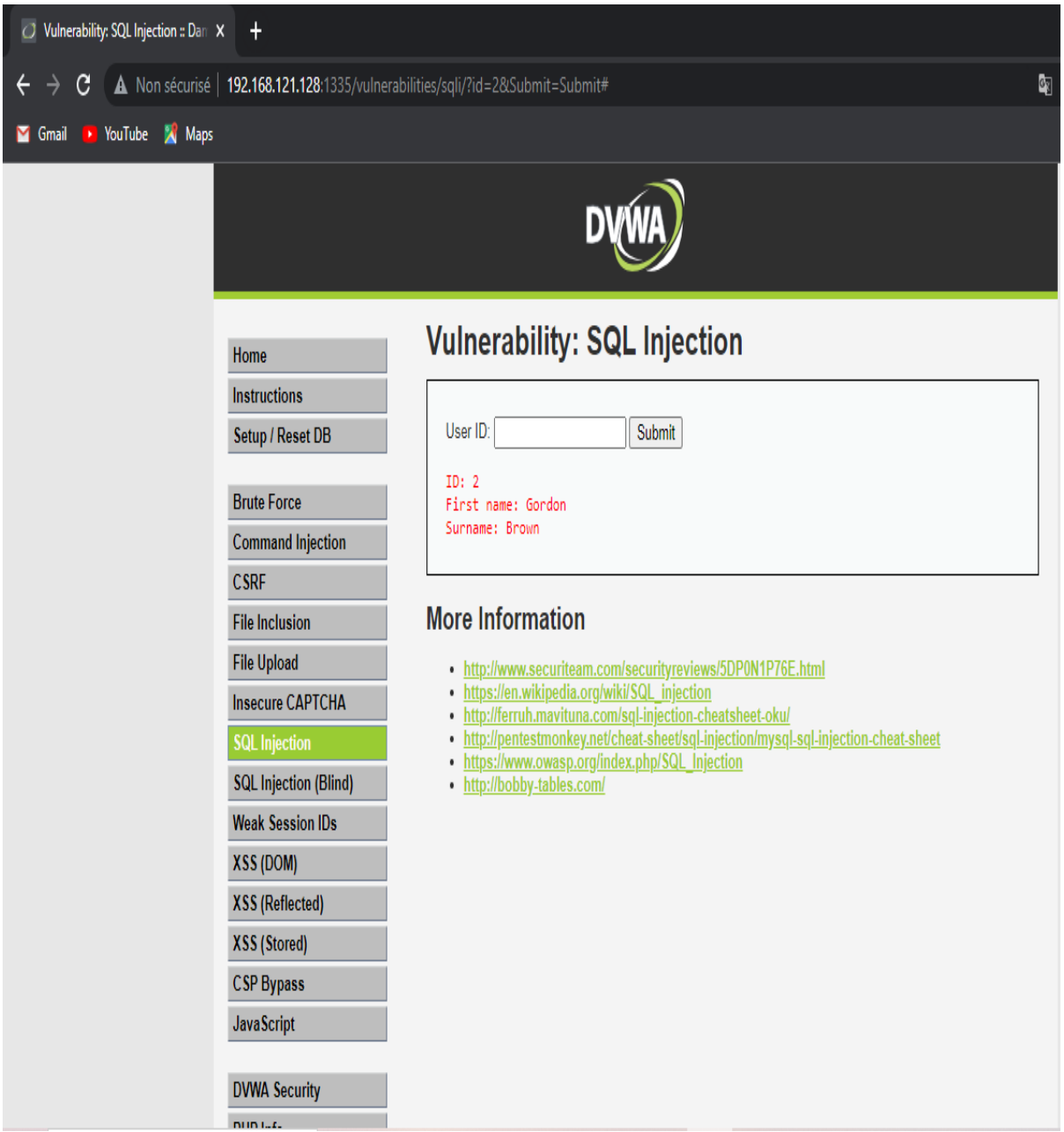


Figure 3.8 - DVWA SQL Injection ID results

You will be given the following information. First name: Gordon. Brown is a surname.

Step 3

If we type (3) into the User ID Field.

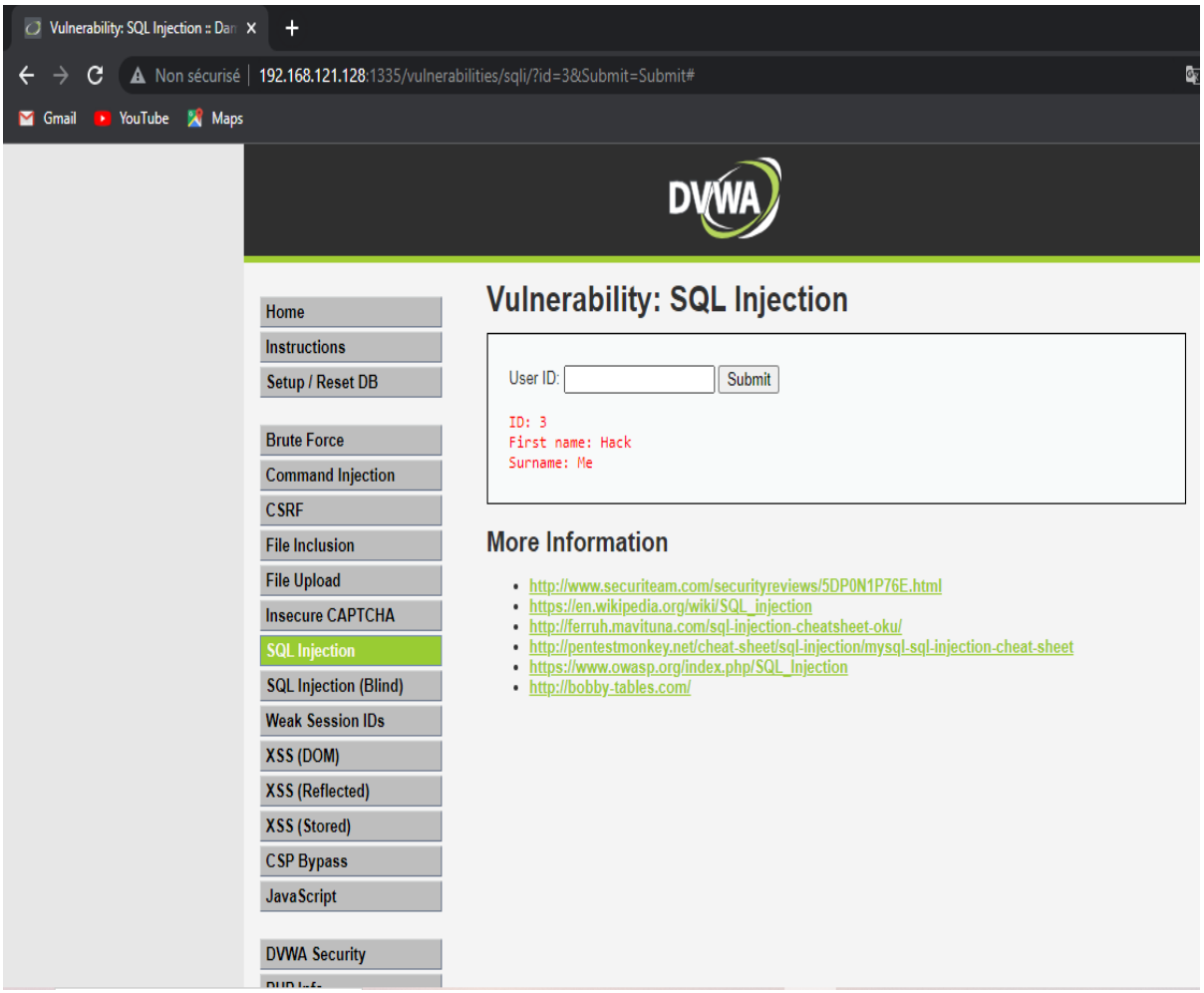


Figure 3.9 - DVWA SQL Injection ID results

We see results of: ID: 3. First Name: Hack. Surname: Me.

We will move onto more advanced SQL terms now to work out the columns. If we type into the User ID Field and click Submit: 3'.

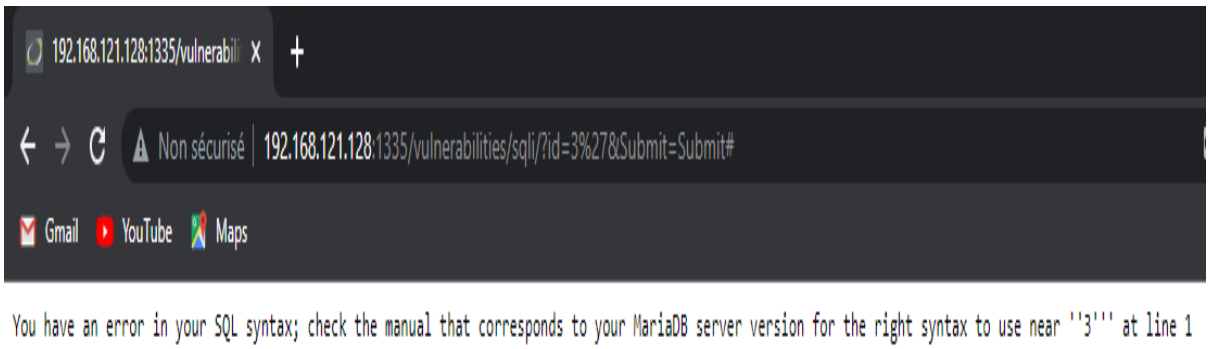


Figure 3.10 - DVWA error screen

This error message is a good sign because it indicates that this website is vulnerable to SQL Injection.

Step 4

If we type into the User ID Field and click Submit: 3’ and 1=1 #

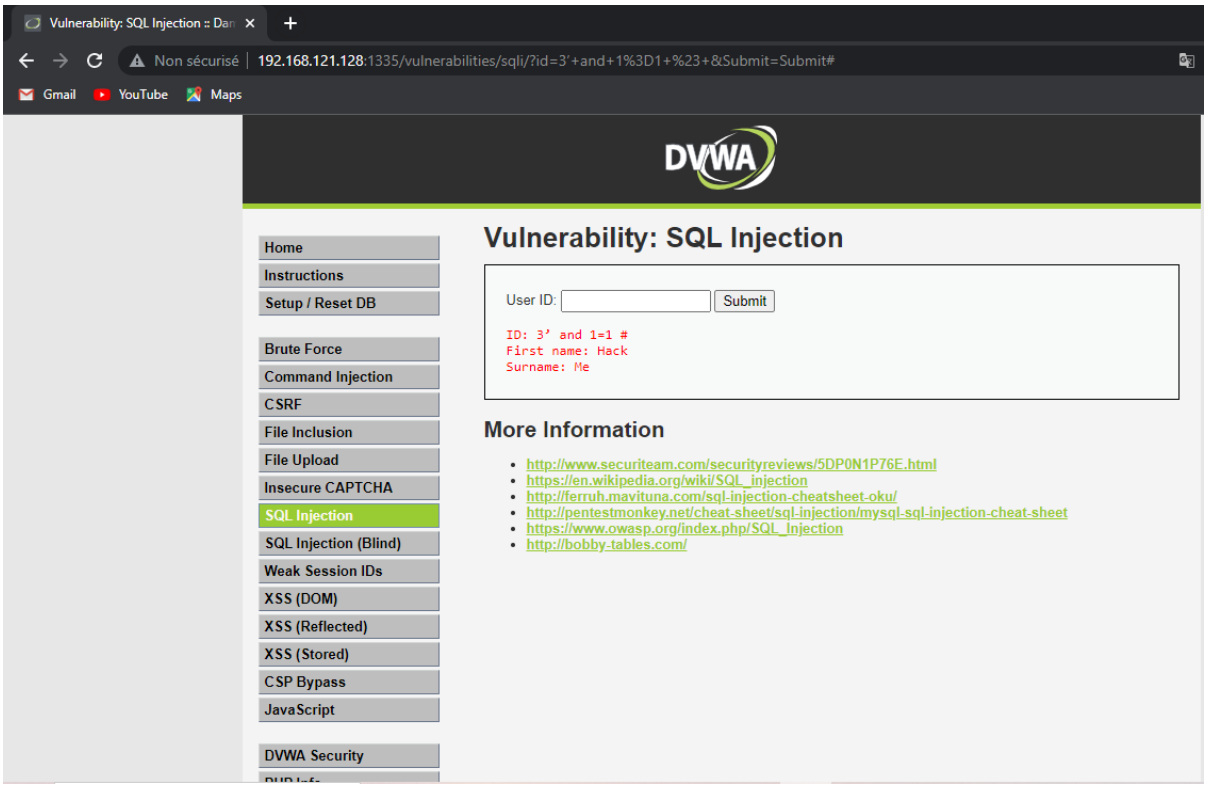


Figure 3.11 - DVWA SQL Injection results

A. Based on always true value.

We can find all the data in the database by using a value that will always be true. '1'='1' or '2'='2' are examples of always true values. So, it will look like this in a query.

Now if we type into the User ID Field: a' OR '1'='1'

```
SELECT first_name, last_name FROM users WHERE user_id = '<anything>' OR <always true value>
```

For the example, I’m going to replace *<anything>* with *‘a’* and *<always true value>* with *‘1’=’1'*

```
SELECT first_name, last_name FROM users WHERE user_id = 'a' OR '1'='1'
```

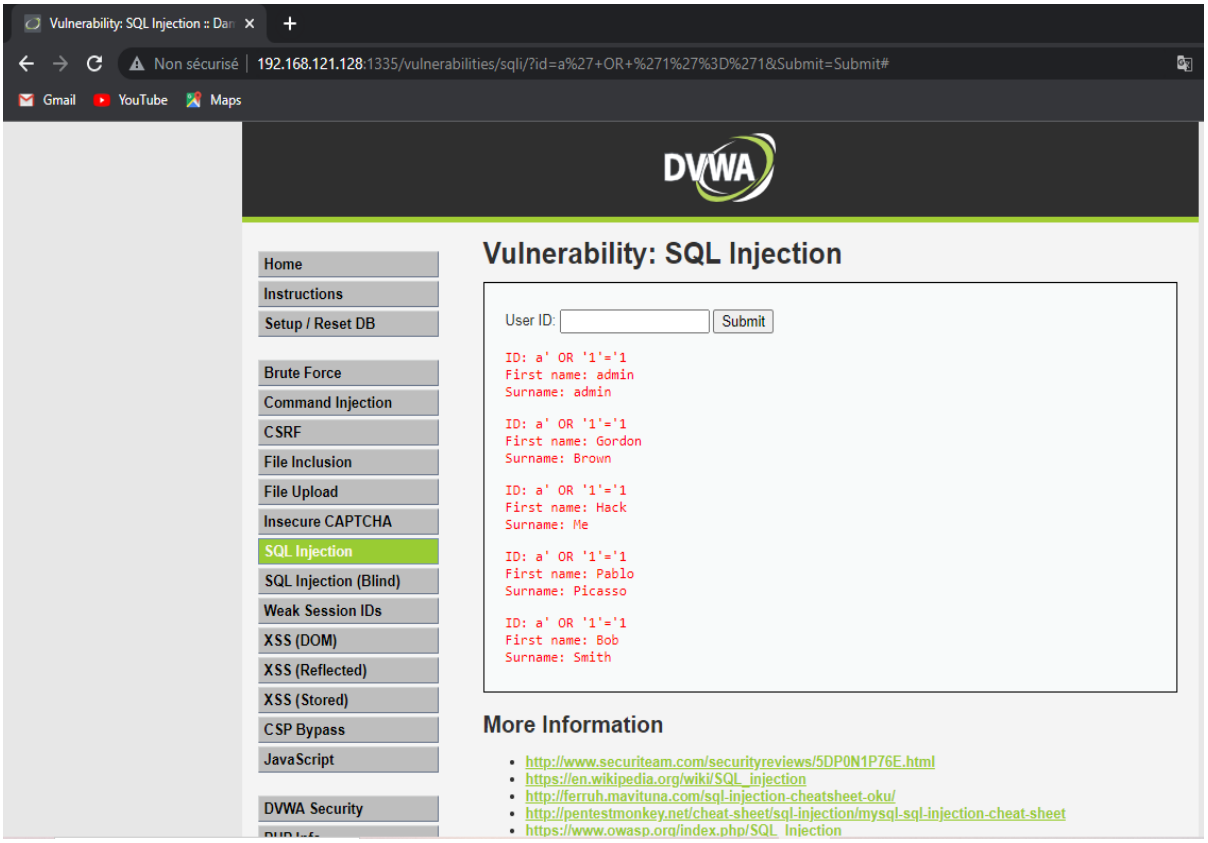


Figure 3.12 - DVWA SQL Injection results

As we can see, it shows ID, first name, and surname for all IDs in database.

B. Union-based.

Attackers can use the Union clause to execute additional Select queries and add the results to the original query. It can be used to retrieve information from other tables. However, you must ensure that:

Each query gives the same number of columns.

Each column's data types must be compatible across queries.

I will attempt to look up the columns in the Users table. To accomplish this, I will assign `'' and '1'='1' UNION CONCAT (table_name,0x0a,column_name) FROM information_schema.columns SELECT null WHERE table_name = 'users' #` to variable `$id`, resulting in the following query:

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '' and '1'='1' UNION SELECT null, CONCAT(table_name,0x0a,column_name) FROM information_schema.columns WHERE table_name = 'users' #';";
```

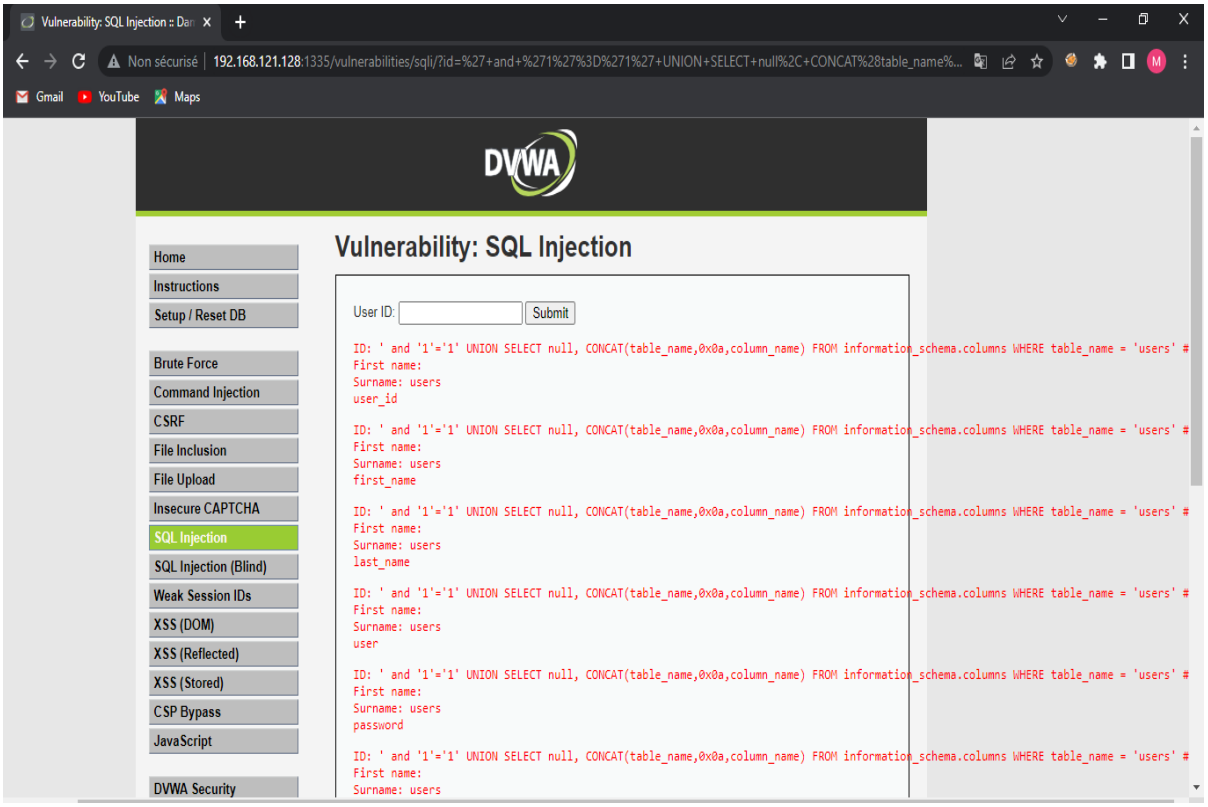


Figure 3.13 - Result of the query execution

It shows that Users table have 10 columns: `user_id (the identifier), first_name, last_name, user, password, avatar, last_login, failed_login, CURRENT_CONNECTIONS,` and `TOTAL_CONNECTIONS`.

Then we'll try to get all users' `user_id, first_name, last_name, user, password, avatar, last_login, and failed_login`. I separate the data into two `CONCAT` clauses. The first `CONCAT` contains the following values: `user_id, first_name, last_name, user, and password`, while the second `CONCAT` contains the following values: avatar, last_login, and failed_login. The query that was run is shown below:

```
$query = "SELECT first_name, last_name FROM users WHERE
user_id = '' or '1'='1' UNION SELECT
CONCAT(user_id,0x0a,first_name,0x0a,last_name,0x0a,user,0x0a,pas
sword),CONCAT(avatar,0x0a,last_login,0x0a,failed_login) FROM
users #';";
```

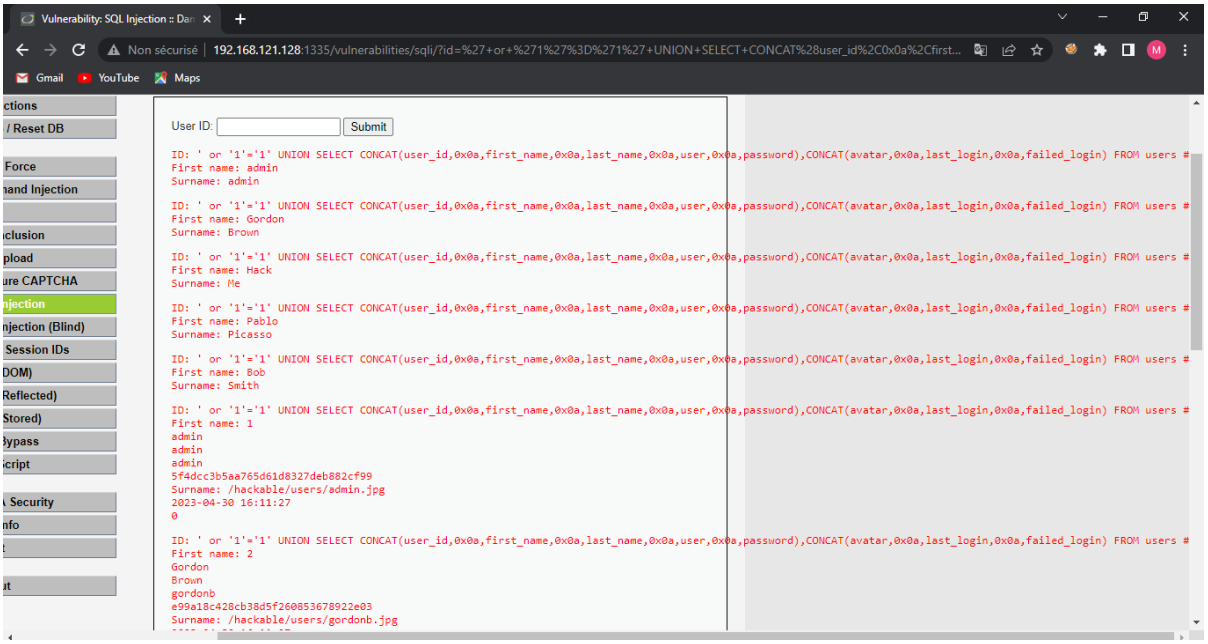


Figure 3.14 - The result of the query

We can even get the users' passwords from the data above, which is extremely sensitive and should not be exposed.

How Can SQL Injection Be Avoided?

Check to see if the input is numeric IDs in databases, are typically integers. We can check the input type and ignore the input if it is not an integer to avoid additional commands in the inputs. We can use the code below to accomplish this.

```
if(is_numeric($id))
```

As a result of query execution, ensure that there is only one record.

The web form actually requires users to enter only one user ID in order for it to return the corresponding data. However, the attacks allow attackers to run a query that returns information for more, if not all, user IDs. To avoid this vulnerability, we can use LIMIT.

```
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;
```

I've also read that there are more secure code fixes, such as:

Using prepared statements in conjunction with parameterized queries. When we use prepared statements, an attacker cannot change the query's intent. If an attacker enters tom' or '1'='1, the parameterized query will look for a username that matches the string tom' or '1'='1.

Using previously saved procedures. SQL statements with parameters are also used, but the SQL code for a stored procedure is defined and saved in the database.

Validation of allow-list input. This validation ensures that only valid user input is used.

To summarize, we learned about SQL injection, source code with low level security, and examples of attacks caused by low level security in source code, and how to improve the code's security.

3.1.3 Modeling XSS Cross-Site Scripting

This vulnerability occurs when an attacker can inject malicious code into a web page that is viewed by other users. This can be used to steal sensitive information or to perform actions on behalf of the victim.

DOM Based Cross Site Scripting (XSS).

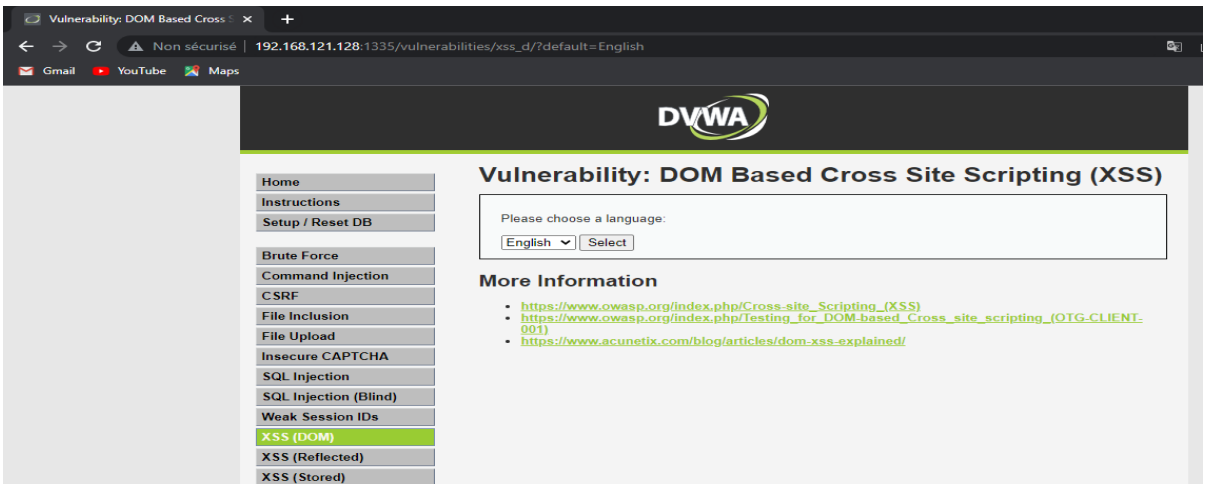


Figure 3.15 - DOM BASED XSS

Watch the URL after pressing the "Select" button. The parameter "default=English" was used. Changing the select box sets the default parameter. What happens if we change the default parameter to something else, such as "Soufiane"?

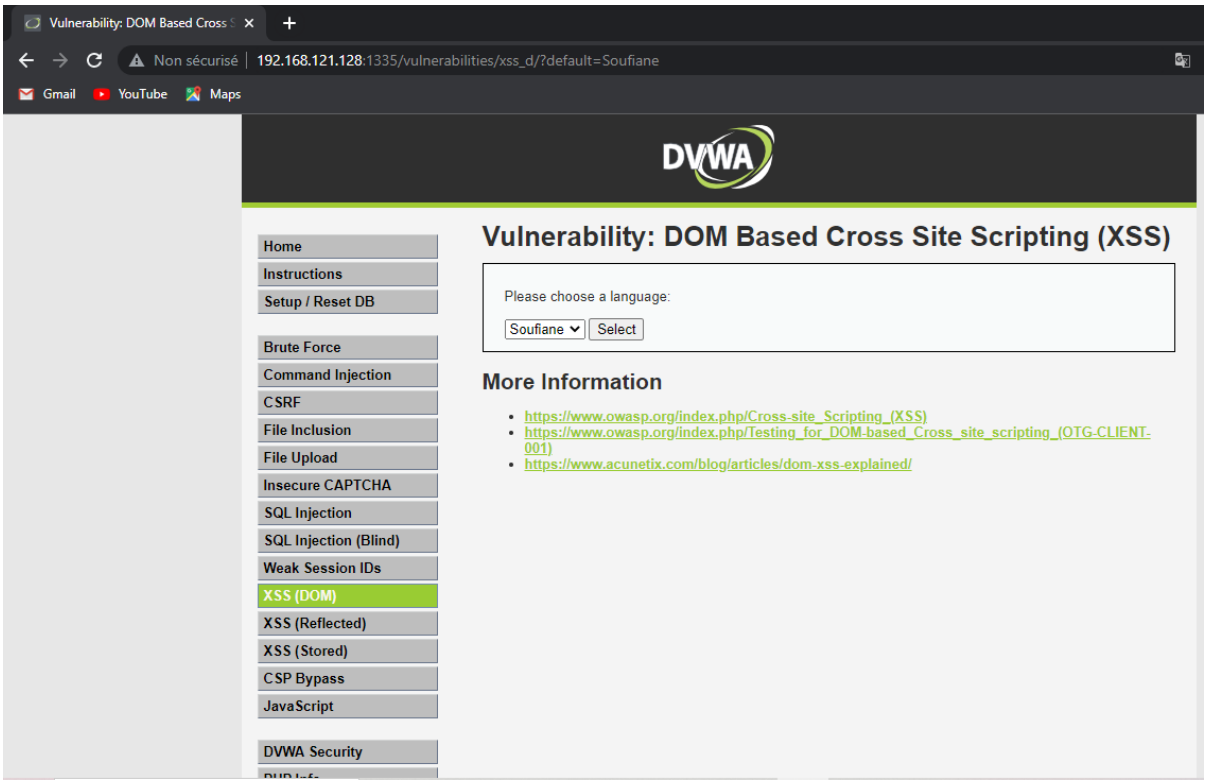


Figure 3.16 - Default parameter changes

Let's now add our own JavaScript to the request param, something simple like an array.

`alert “<script>alert(“Houston, we have a problem!!”)</script>”`:

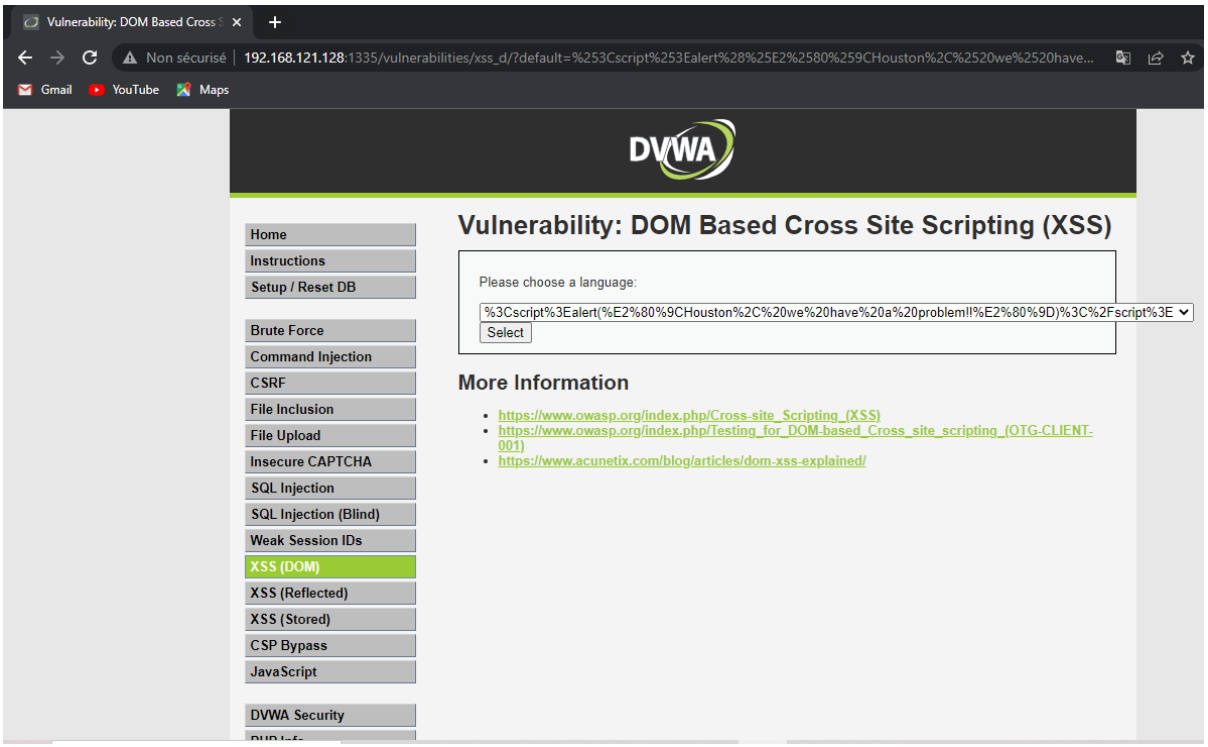


Figure 3.17 - XSS script

We discovered that the site is vulnerable to XSS. Let's use this script to get the cookie.

“<script>alert(document.cookie)</script>”

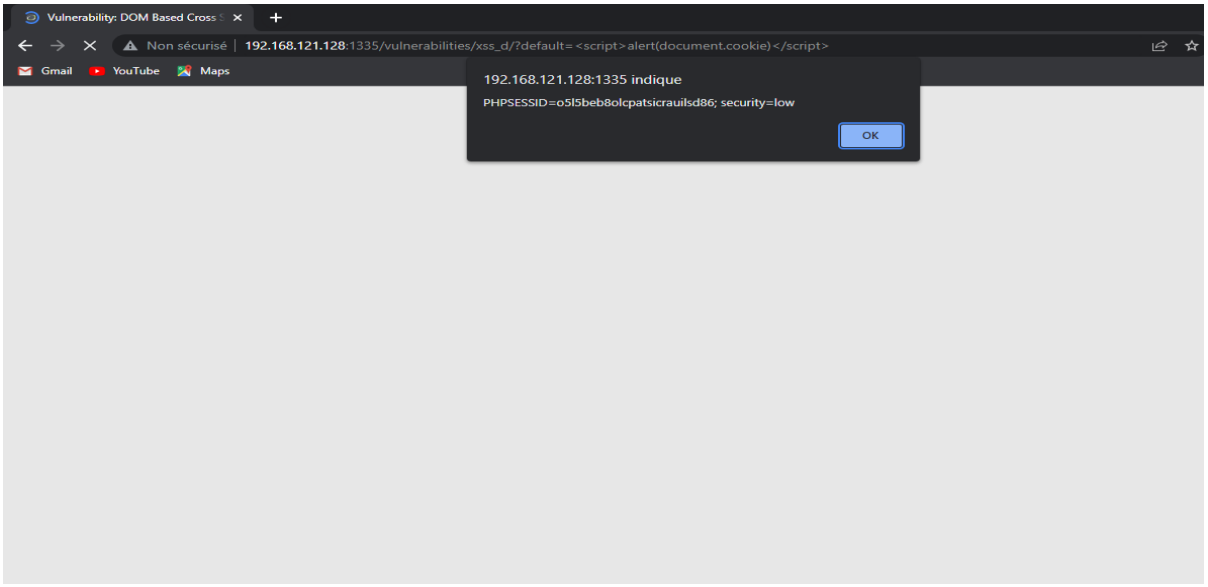


Figure 3.18 - Get the cookie with script

Reflected Cross Site Scripting (XSS).

The exploit's goal is to make the page execute arbitrary code, allowing the attacker to steal cookies or perform malicious activities using browser exploitation tools such as Beef.

If the website executes our code, in this case, a simple alert popup, we can consider the level passed.

At this level, we can see an input text field into which we can type a name.

To find out if it's a vulnerable field, we can insert some custom HTML and see if the tags are filtered in any way.

Let's give it a shot with this input:

```
<h1>Soufiane</h1>
```

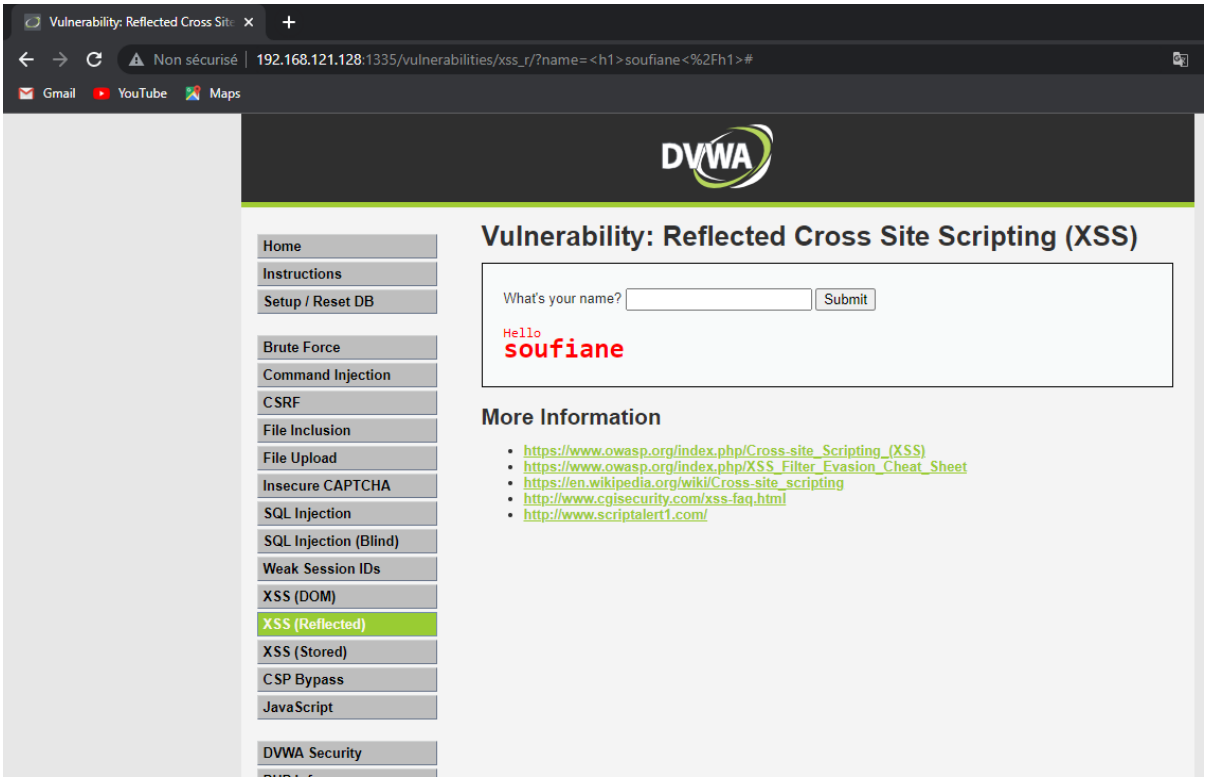


Figure 3.19 - H1 input result

For further confirmation by pressing CTRL+U, we can see the source code.

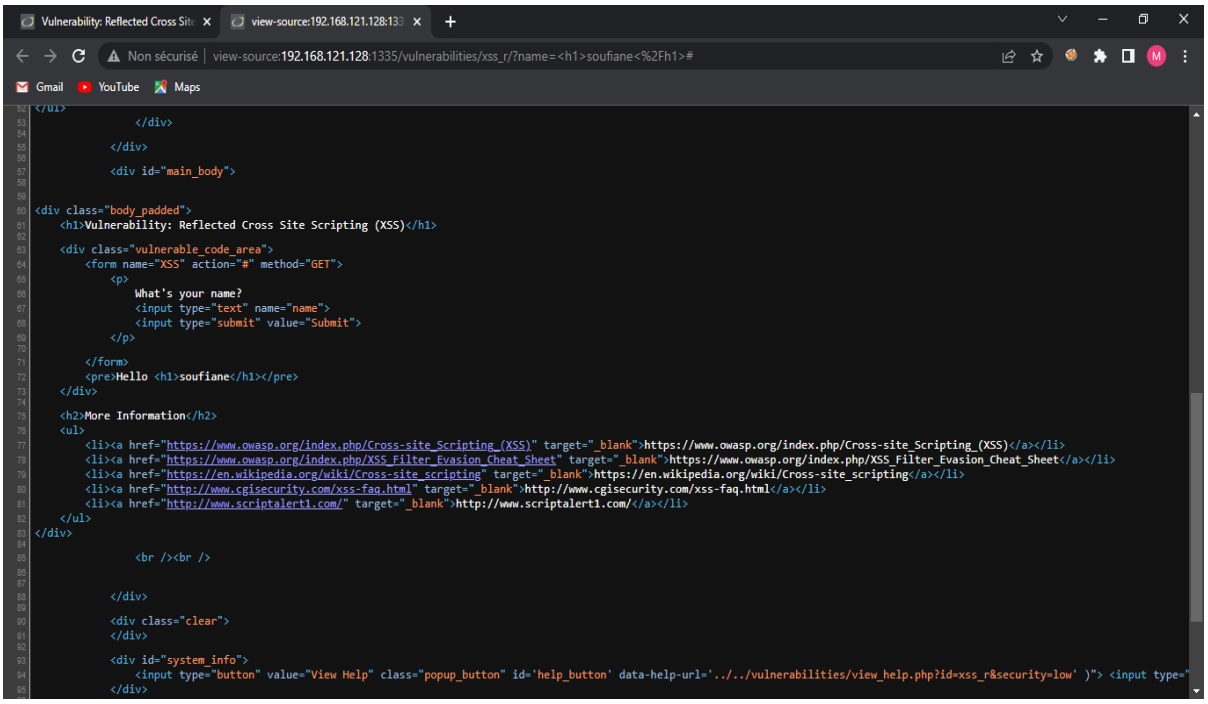


Figure 3.20 - Source Code

I simply selected the code of interest, and everything we wrote is reflected on the page without any elaboration. It's obviously a good candidate for our heist! We can try to submit this script this time.

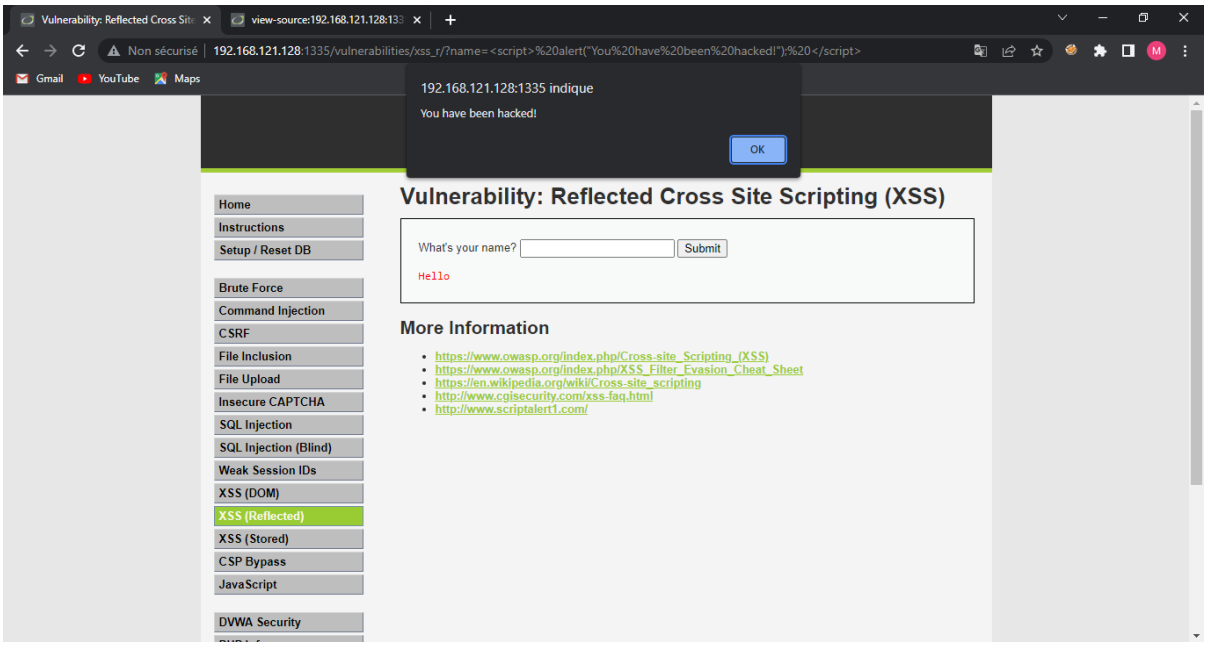


Figure 3.21 - The alert popup

Stored Cross Site Scripting (XSS).

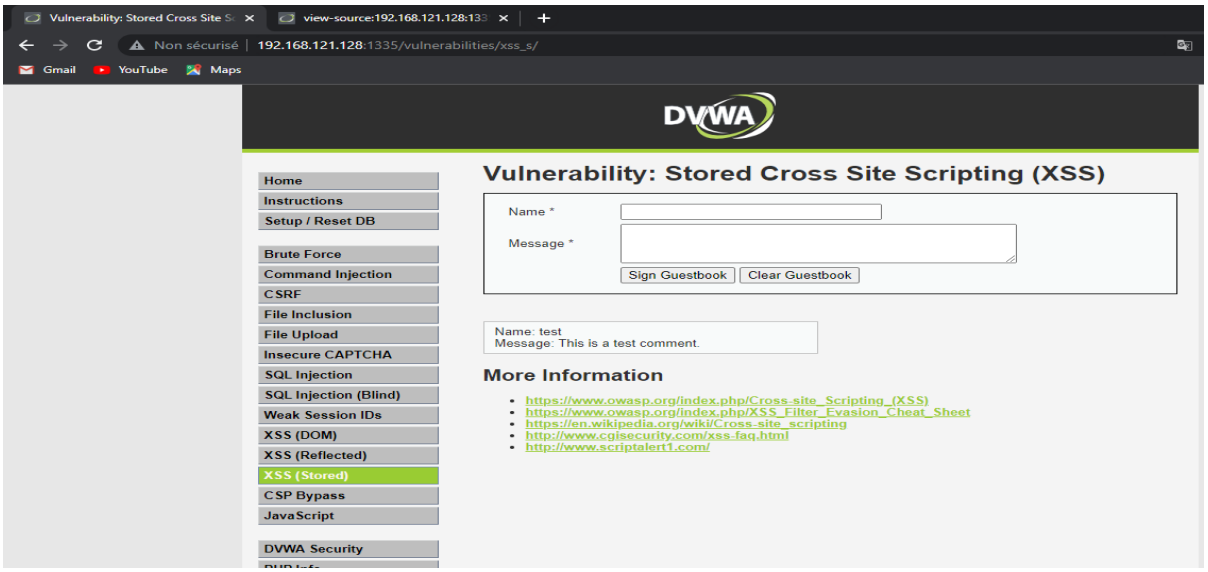


Figure 3.22 - DVWA XSS STORED PAGE

So, we can try to type just the basic exploit into the textbox “Message”. Let’s write:

```
<script>alert('You have been hacked!');</script>
```

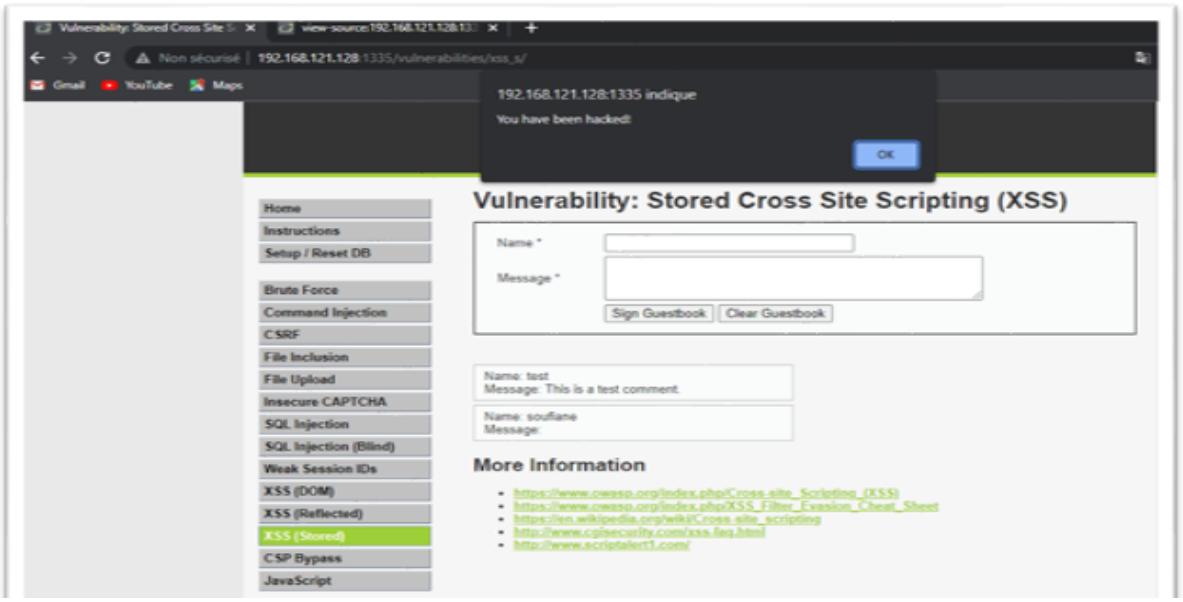


Figure 3.23 - The alert popup

When we reload the page, the alert popup remains because the script is stored in a guestbook comment; this is the real difference with Reflected XSS.

To protect against all types of XSS (Cross-Site Scripting) attacks, you can follow these general guidelines:

- input Validation and Sanitization: Ensure that all user input, including data entered in Arabic or any other language, is properly validated and sanitized. Implement server-side input validation to reject or sanitize any input that contains suspicious or malicious characters;

- output Encoding: Properly encode and sanitize all dynamic data that is displayed on web pages. Use output encoding techniques specific to the context where the data is being used (HTML, URL, JavaScript, etc.). This helps prevent malicious scripts from being executed when the data is rendered.

Content Security Policy (CSP): Implement a Content Security Policy that defines the allowed sources of content on your website. This policy can restrict the execution of scripts from unauthorized sources and help mitigate XSS attacks.

Contextual Output Filtering: Apply context-specific output filtering to prevent any user-supplied data from being treated as executable code. Validate and filter user-generated content based on the expected format and context of its usage.

Use Security Libraries and Frameworks: Utilize security libraries or frameworks that provide built-in protection against XSS attacks. These tools often include automatic output encoding, input validation, and other security features.

Regular Security Updates: Keep your web application and server software up to date with the latest security patches and updates. This helps address any known vulnerabilities that could be exploited by attackers.

Security Testing: Conduct regular security testing, including penetration testing and vulnerability assessments, to identify and remediate any XSS vulnerabilities in your application.

User Education: Educate your users about the risks of XSS attacks and encourage safe browsing habits. Teach them to be cautious about clicking on suspicious links, downloading files from untrusted sources, and sharing sensitive information.

By implementing these measures, you can significantly reduce the risk of XSS attacks and enhance the security of a web application.

3.1.4 Cross-Site Request Forgery attack

Cross-Site Request Forgery (CSRF), also known as XSRF attacks or "one-click attacks," are common vulnerabilities in web applications. This is a type of malicious exploitation where an attacker tricks a user into unknowingly performing unwanted actions on a targeted web application. The attacker forges requests in the context of the victim, skillfully leveraging the trust between the user and the target website.

The process of a CSRF attack can be divided into several steps:

- user Authentication: The user logs into a legitimate website and receives an authentication token;

- crafting Malicious Request: The attacker creates a malicious website or injects malicious code into a legitimate website visited by the victim;

- deceiving the User: The malicious code triggers a request to the legitimate website, using the user's authentication token;

- unauthorized Action: The legitimate website, assuming the request is legitimate, performs the action requested by the attacker.

Let’s consider example of such attack, with diagram is depicted on Figure 3.24.

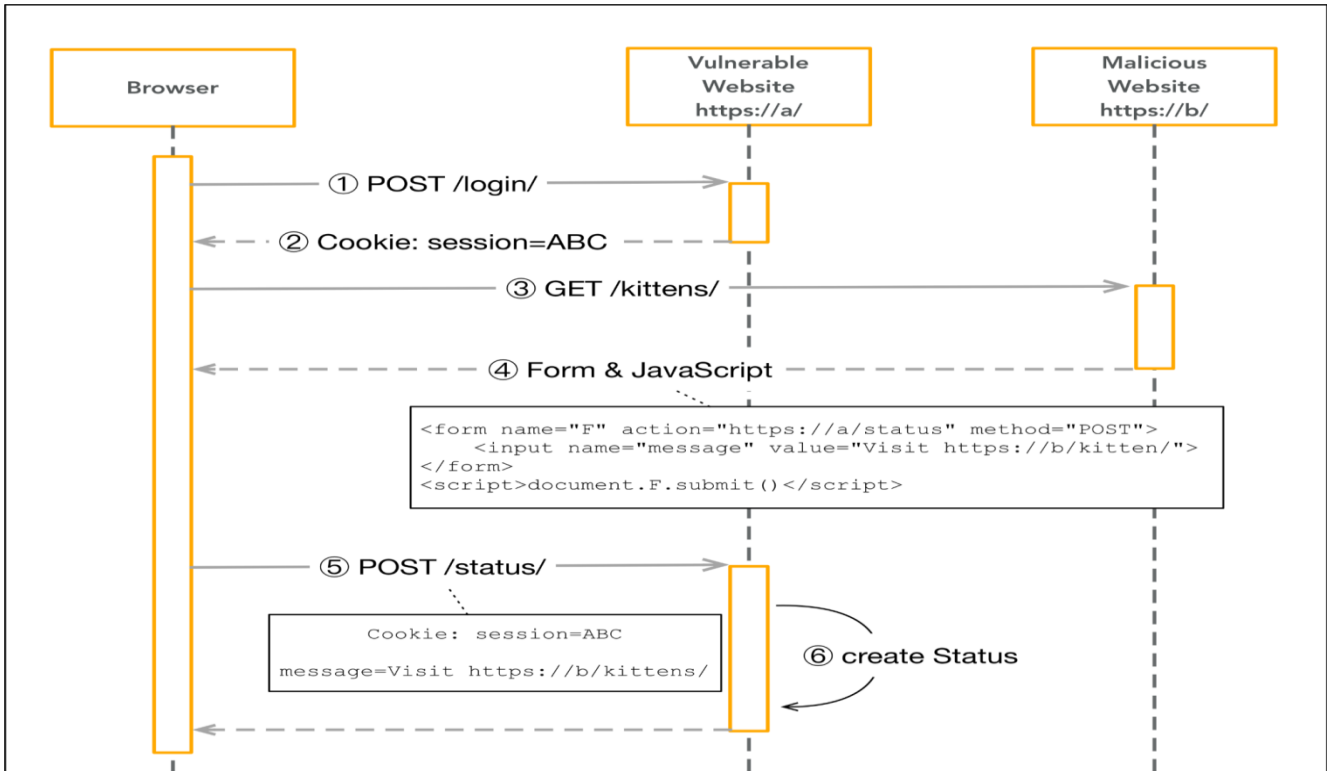


Figure 3.24 - Example of CSRF attack

The attack consists of several steps.

1) User enters your website at https://a.

2) A session cookie is returned from your server.

3) The user is tricked into accessing a website that the attacker controls, such as https://b/kittens/. Who can blame the user for not enjoying kittens?

4) The website of the attacker returns a form with the action https://a/status/ and some JavaScript that submits the form automatically.

5) JavaScript is run by the user's browser, and the form is POSTed to your server. This post request is authorized since it includes the user's session cookies.

6) The spam message created by the attacker is included in the POST request that is sent to your server. It creates a status with the spam message and has no means of knowing that it wasn't triggered by the user.

Let’s use the DVWA application to simulate a simple CSRF attack.

First of all we navigate to the CSRF attack vulnerable page example which contains the form for Changing Password.

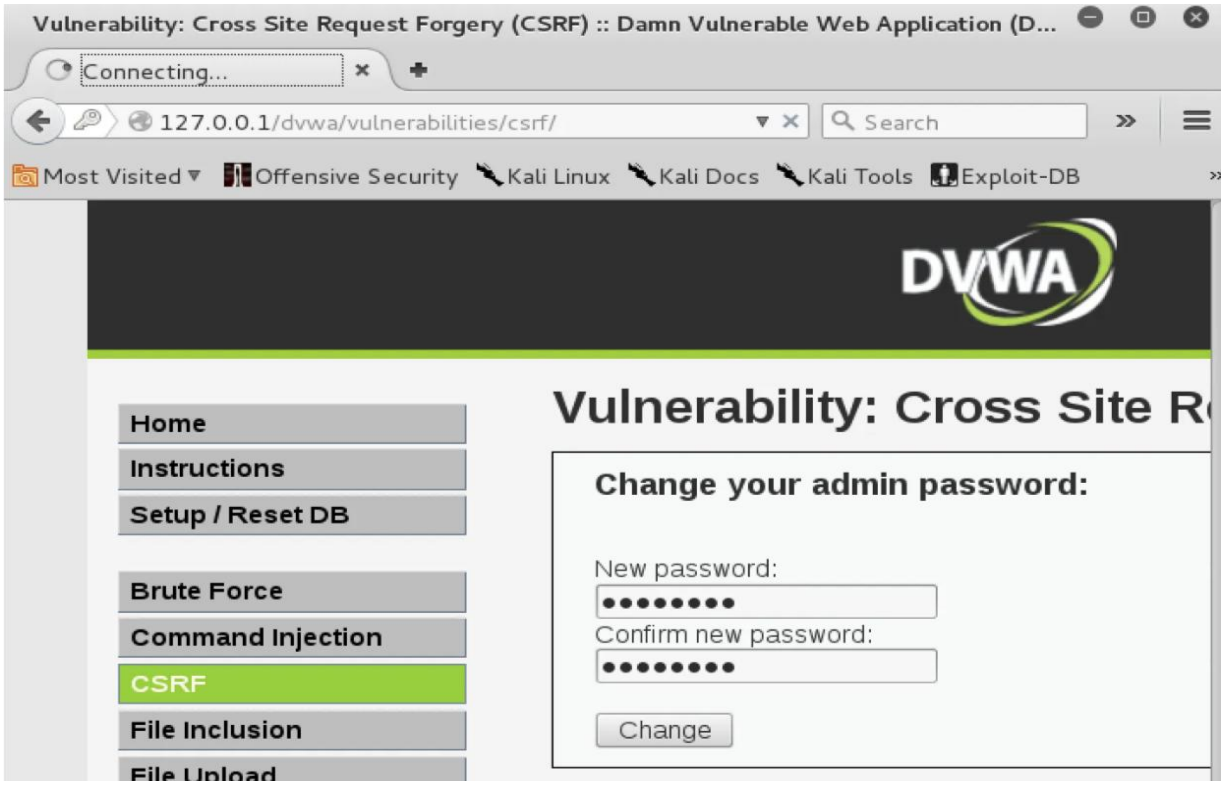


Figure 3.25 - CSRF attack vulnerable page example

We need to check the HTTP request that this form will send to the server side after we fill all the fields and press the Change button. For example we can use the Burp Suite tool to see the data on request.

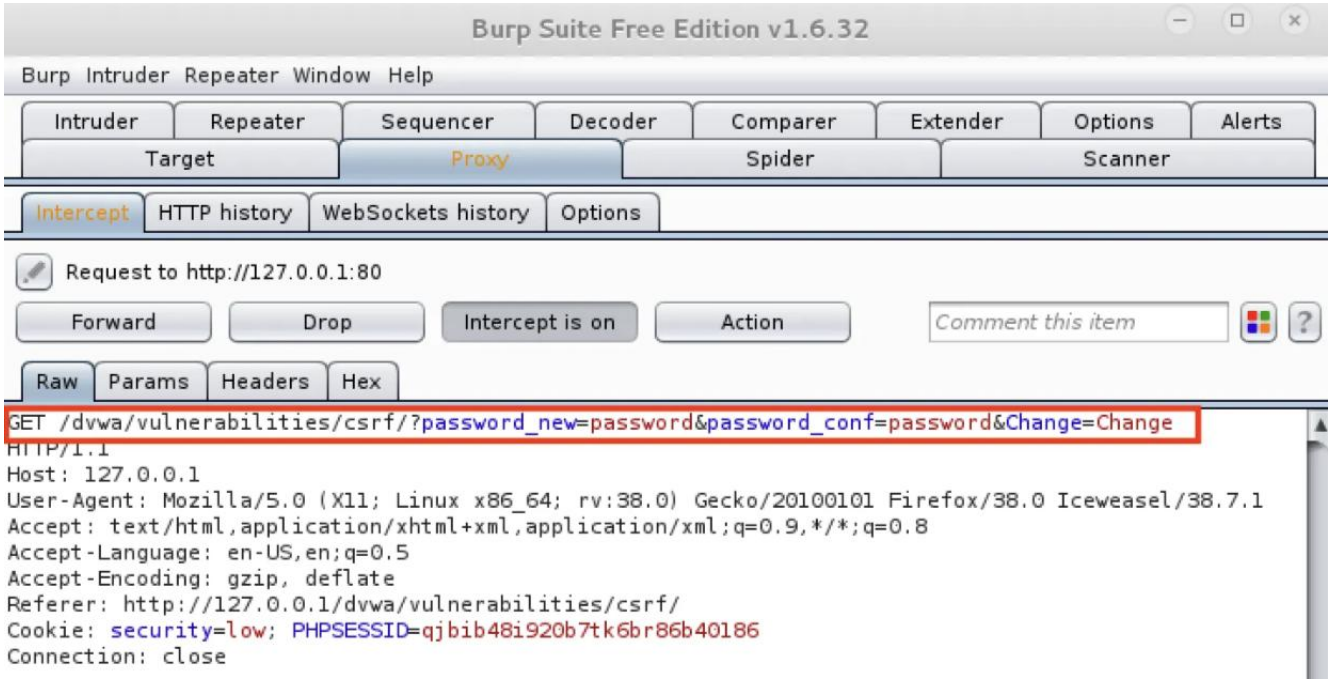


Figure 3.26 - Using to check request data

As you can see, in this simple example our form uses a GET request to the server side with all needed parameters. In this case we don't even need to create a duplicate false password change page - our password information was included in the GET request. All we need to do is create a link and force users to click it. Figure 3.27 shows how we created an HTML page with this fake link, and Figure 4 shows how this page will look like.

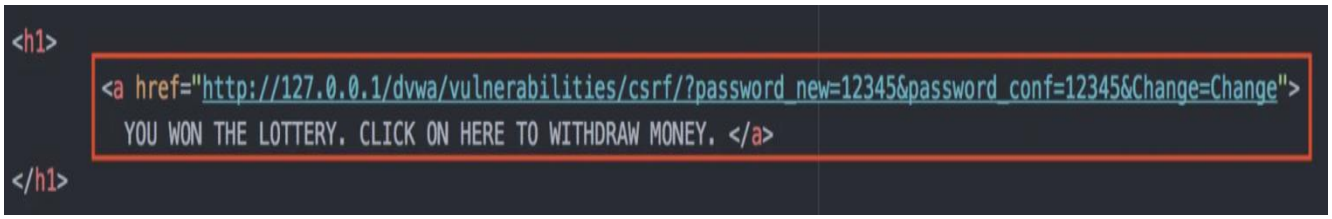


Figure 3.27 - Fake link for CSRF attack

Figure 3.28 - Fake page for CSRF attack

Now, if our victim clicks the link: http://127.0.0.1/dvwa/vulnerabilities/csrf/?password_new=12345&password_conf=12345&Change=Change

A CSRF attack will take place and successfully change password to “12345”.

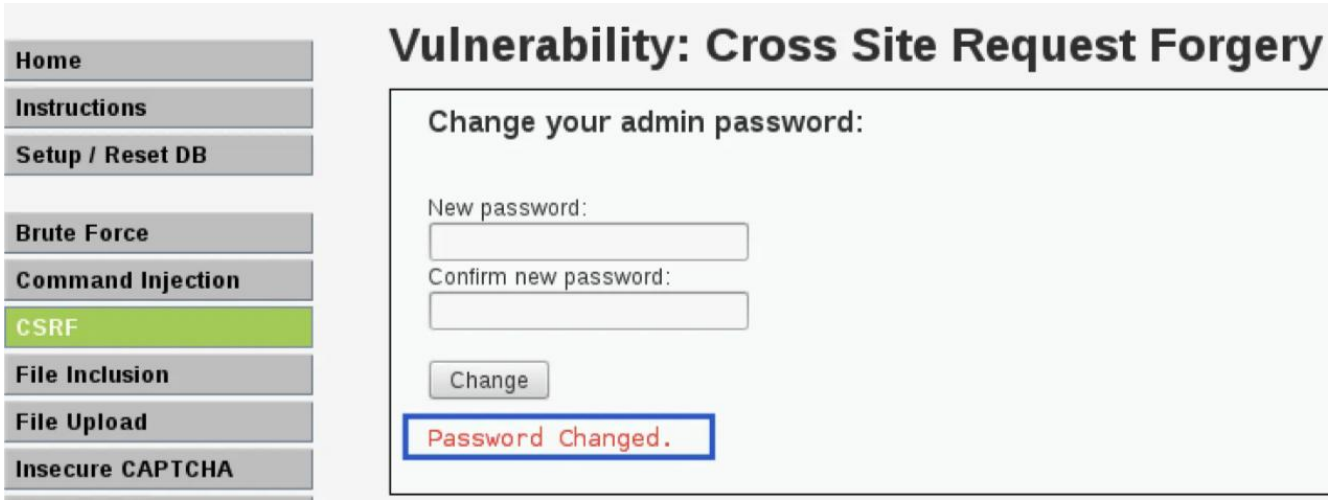


Figure 3.29 - CSRF attack successfully changed password

To protect a website from Cross-Site Request Forgery (CSRF) attacks, there are several effective countermeasures that can be implemented.

1) Use Anti-CSRF Tokens: Implement the use of anti-CSRF tokens, also known as synchronizer tokens, within your web application. These tokens are unique values generated for each user session and are included in forms or requests. They help validate that the request originates from a legitimate source by comparing the token sent with the token stored on the server.

2) SameSite Cookie Attribute: Set the SameSite attribute on cookies to restrict their cross-site usage. By setting the SameSite attribute to "Strict" or "Lax," you can prevent cookies from being sent in cross-site requests, effectively mitigating the risk of CSRF attacks.

3) Check Referer Header: Verify the Referer header of incoming requests to ensure they originate from your own website. Although the Referer header can be manipulated, it can still provide an additional layer of protection against CSRF attacks.

4) Implement Double-Submit Cookie: Generate and set a random CSRF token as a cookie value during user authentication. Then, include the same token in form submissions or request headers. Upon receiving a request, compare the token in the cookie with the token in the request to verify its legitimacy.

5) Limit Privileges and Actions: Assign appropriate privileges and access levels to users. Restrict or validate sensitive actions that can be performed, especially those with potential security implications, to prevent unauthorized changes.

6) CAPTCHA Challenges: Implement CAPTCHA challenges for critical or sensitive actions to ensure user involvement in the process. CAPTCHAs require human interaction, making it difficult for automated CSRF attacks to succeed.

7) Time-Limited Tokens: Set an expiration time for anti-CSRF tokens to limit their validity. This prevents attackers from reusing or replaying tokens obtained from previous sessions.

8) Educate Users: Educate your users about the risks of CSRF attacks and encourage them to be cautious when clicking on links or submitting forms on unfamiliar websites. Awareness helps users recognize potential threats and take appropriate actions.

9) Security Testing: Regularly conduct security testing, including vulnerability assessments and penetration testing, to identify and address any potential vulnerabilities within your web application.

Implementing a combination of these countermeasures significantly reduces the risk of CSRF attacks. However, it is important to note that there is no one-size-fits-all solution, and the specific implementation may vary based on the technology stack and requirements of your web application.

3.2 The Importance of Needing Risk Research for Penetration Testing of Web Applications

Risk is defined as the multiplication of the probability of an undesirable event occurring and its consequences.

The risk assessment methodology allows you to assess the potential consequences and the likelihood of exploitation of each vulnerability detected by the scanner or during penetration testing. This helps determine how severe the vulnerabilities are and what security measures need to be taken.

Since we still need to understand how risky what the scanner found, let's use modified OWASP methodologies for quantitative risk assessment.

Risk analysis is a process that assesses, manages, and communicates risk information to aid decision-making. Risk assessment entails identifying and quantifying the risks of adverse situations (or hazards or threats) by determining all likelihoods of their causes and the severity of their consequences. The risk measures obtained can help with cost-benefit analysis for resource planning to prevent, eliminate, or mitigate these undesirable situations. These are risk management activities. Many risk assessment techniques exist (e.g., FMEA, HAZOP (Hazard and Operatability), FTA (Fault Tree Analysis) (Haimes, 2004), but they have primarily been applied to hardware rather than software artwork. Risks associated with software (or software risks) are those that take into account the unintended consequences of software behavior in the system in which the software is used.

Security and safety are two related perspectives on risk-acceptance value judgments. Security is concerned with the protection of asset ownership and privacy, while safety is more concerned with the protection of human lives and environments, factors that cannot be easily quantified.

Some assets in web applications are also examples, such as trust and a company's reputation. As opposed to safety, sources of security threats include both malicious (e.g., deliberate attack, code injection) and non-malicious (e.g., natural disaster, human error, software system failure) actions that may result in a security breach.

Security attacks, whether planned or unplanned, can originate from both internal and external threats, depending on the degree of asset exposure and the system's vulnerability to attack. A system's vulnerability refers to a weakness (i.e., a specific attribute/environment) that makes a system more vulnerable to attack or increases the likelihood of an attack succeeding.

Vulnerabilities that are exploited can jeopardize security.

In finance, risks are estimated in terms of annualized loss expectancy (ALE), which is a product of the annual rate of occurrence and single loss anticipation. The former represents probability, while the latter represents long-term consequences. Risks

are estimated in security risk analysis using adapted ALE formulae. The basic probability factors are related to the degrees of exposure, vulnerabilities, and threats, while the basic consequence factors are measured by asset values. In order the security risk of a given threat/adverse action can be roughly estimated by:

$$\text{Risk} = \text{threat} \times \text{exposure} \times \text{vulnerability} \times \text{severity}, \quad (3.1)$$

where threat measures the threat likelihood.

The first three contribute to probability measures, while the last one measures its outcome. We omit threat in this paper because it is easily integrated into our analysis at a later stage of the design. While the ALE concept is effective in general financial and security risks, it may not be applicable in cases such as terrorist attacks, which are rare or may never occur.

3.3 Methodology for quantitative risk assessment

Quantitative risk assessment methodologies, such as the ALE-based approach, provide useful mechanisms for risk ranking in risk management decision-making. However, they are frequently corrected for producing results that are too general (statistical-like) to be useful for specific decision-making, as well as producing results that are insufficiently credible due to their reliance on subjective value assignments.

The proposed methodology [10] has two goals: first - to estimate security risks of a web-based software system based on its early design configuration, and second - to estimate probability measures more objectively. The latter is accomplished using heuristics based on relevant design attributes (more on this later). Because web applications typically have multiple architectural layers and trust boundaries, our risk methodology must include relevant attributes from multiple design levels (e.g., component, network, and hardware platforms).

The methodology consists of several main stages.

Inputs: Software design, hardware deployment design, likelihood of each scenario. Let sk be the likelihood of scenario k.

Outputs: Security risks of a system to be built and security risks of each component.

1) **Determine interaction rates in each scenario:** For each use case, for each scenario *k*, for any two components *i* and *j*, compute $I_{ijk} = n_{ijk} / N_{ik}$, where I_{ijk} is an interaction rate from *i* to *j* in *k*, n_{ijk} is the number of interactions from *i* to *j* in *k*, and N_{ik} is the number of all interactions out of *i* in *k*.

2) **Determine transition rates in all scenarios:** Construct a dependency graph of components where component transitions are based on interaction rates from all scenarios. Let t_{ij} be a transition rate from components *i* to *j*. Then $t_{ij} = \sum_k s_k I_{ijk}$. Normalize t_{ij} so that a total sum of transition rates from each component is one.

3) **Estimate component vulnerability:** By using hardware deployment, determine *Vi* , an estimated vulnerability ratio of component *i*. Note that a total sum of vulnerability ratios over all system components is one.

4) **Combine likelihood estimates:** Determine probability of security breach of each component by identifying all access paths to the component. For each path, estimate the likelihood by using the transition rates (t_{ij}'s) and vulnerability ratios (*Vi*'s) of components along the path. Combine the likelihoods obtained for each alternative path to the component.

5) **Severity Analysis:** Determine severity of each adverse action in each component. Categorize the severity and select the worst case in each component.

6) **Compute security risks:** for each component and the overall system using results obtained from 4) and 5).

A software design and a hardware deployment design are required as inputs to the methodology. We assume that the software design is expressed in terms of an extended activity or workflow diagram, in which the software components of web applications and use case actors, as well as activities in use cases, are represented. All use cases, each of which may have one or more scenarios, are included in the software design.

The hardware deployment design describes hardware platforms such as a host or server, their attributes (such as services and operating environments), and network attributes (such as IP addresses for network configuration and LAN, WAN, or DMZ connectivity). This information is used by security scanning tools to identify vulnerabilities in each host in the system. The hardware deployment plan also specifies which software components will be used or run on which hosts.

As illustrated in methodology, the first two steps employ software design characteristics as heuristic measures of risk probability factors.

Step 1 computes interaction rates among software components based on the design workflow for each scenario. For the sake of simplicity, we assume that each outgoing interaction from the same component occurs equally likely (appropriate weights can be applied when insights into interaction likelihoods are available).

Step 2 creates a graph with nodes representing software components. Transition rates among components are calculated by combining interaction rates from each scenario while accounting for scenario likelihoods. In the context of security risk, scenario likelihoods are heuristic measures of the probabilities of potential threats (e.g., when scenarios include adverse actions), whereas interaction rates are heuristic measures that partially measure degrees of component exposure.

Step 3 employs hardware deployment design characteristics as a heuristic measure of an additional probability factor of security risk. We begin by identifying each host's vulnerability sources. A table lookup procedure on publicly available vulnerability databases for vulnerability sources based on relevant host and network attributes (e.g., IP addresses, services) can be used to accomplish this. Apache Chunked-Code software on web servers, Windows XP SP2 and JRE 1.4.2 for running JAVA programs on application servers, and Oracle and TNS Listener software on database servers are examples of vulnerability sources. When the hardware platforms to be deployed are already operational, scanning tools can easily identify such vulnerabilities and their sources.

The following step is to assess the vulnerabilities of each software component. Each component may necessitate the use of one or more hosts. A component running on one application server, for example, may need to query data on another database server. We can determine all hosts required by each component based on the deployment design. Let *Hi* represent a collection of hosts required by software component *i,* and *vh* represent a collection of vulnerabilities in host *h*. Several vulnerabilities, vi, and their normalization to a vulnerability ratio, for component *i Vi* are calculated as follows:

$$v_i = \sum_{h \in H_i} v_h \text{ and } V_i = v_i / \sum_i v_i \qquad (3.2)$$

Here, vulnerability ratios of software components are heuristic measures of security risk likelihood factors. The more vulnerable a software component is, the more likely it is that the system's security will be jeopardized.

Step 4 combines all the probability factors obtained in Steps 2) and 3) to estimate the likelihood of each component being attacked (or experiencing a security breach). This likelihood is interpreted here as the possibility of a component under investigation being the first to be attacked in the system. As a result, all components and links along the path to the component are secure (not vulnerable to attack and not exploited).

The likelihood of a node being secured is calculated by subtracting its exposure degree, which is calculated as its weaknesses (i.e., vulnerability ratio), from one.

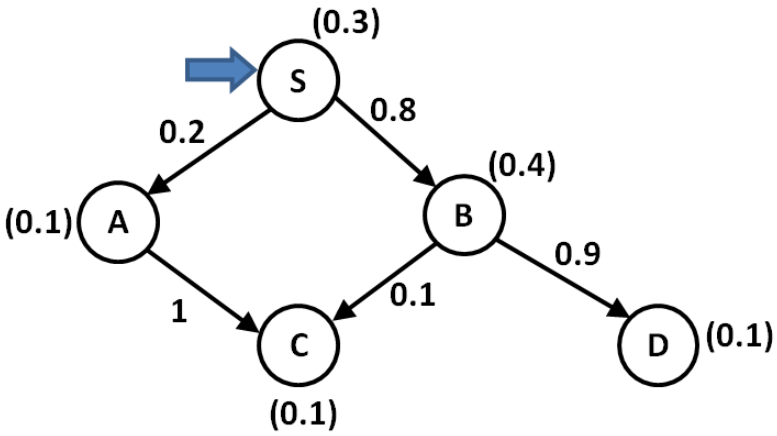


Figure 3.30 - Likelihoods of security breach in components

This step is illustrated in Figure 3.30 by a small example of a dependency graph with nodes S, A, and B.

B, C, and D are software design components. A double-lined arrow indicates that S is a software system execution entry point. Every component and transition are each annotated with a vulnerability ratio (in bold) and a transition rate. As shown in Figure 3.30, the vulnerability ratio of S determines its likelihood of breaching security. The probability of A being attacked (via S →A) is calculated by dividing its probability of being reached without attacks (i.e., S being secured and S transiting to A) by its exposure factor (weaknesses)

The former is determined by the probabilities of S being secured and S transiting to A, whereas the latter is determined by A's vulnerability ratio. As a result, the probability of A being attacked is estimated (1-0,3)x0,2x0,1 = 0.014.

Similarly, the probability of a security breach in B is 0.224. The chance of C being attacked is calculated by adding the chances of C being attacked via S → A → C (i.e., (1 – 0.3) x 0.2 x (1 - 0.1) x 1 x 0.1 = 0.0126) and via S → B → C (i.e., (1 - 0.3) x 0.8 x (1 - 0.4) x 0.1 x 0.1 = 0.00336)

This results in 0.01596.

In general, the estimated probability along an access path is analogous to the Bayes Rule-based propagation of probabilistic effects in a Bayesian network (Pearl,

1997) [10]. The transition rate functions similarly to the conditional probability of a destination being attacked given the presence of a source. We do not explicitly represent access privilege for the sake of clarity. We assume that any activity within the same component can be accessed equally. However, not every component deployed by the same host is equally accessible.

We also presume that network connections between hosts are not vulnerable. Such flaws are easily accommodated in our proposed approach. Because all probability factors (transition rate, vulnerability ratio) have a maximum value of one, the longer the path to a component is, the more difficult it is to attack (i.e., its chance of being attacked decreases). We consider the maximum likelihood that can be obtained for a path that contains cycles. As a result, the heuristic function we use to estimate likelihoods is intuitive in security contexts.

Step 5 in methodology estimates the consequences of each adverse action in each software component using methods similar to HAZOP (Hazard and Operability) or FMEA (Failure Mode and Effect Analysis). We will concentrate on probability estimation in this work. As a result, we use a standard severity analysis technique to categorize and quantify component consequences in accordance with security standards (ISO, 2002). We can estimate the security risks of each component of the system being designed by using the maximum severity obtained in each component.

We estimate the likelihood of a system to safely terminate by multiplying the likelihood of a system to safely terminate by a vulnerability ratio of overall system components and the system severity.

We estimate system severity by taking most severity categories of system components.

3.4 Web Application Risk Research

3.4.1 Web Application Design

A web-based material requirements planning (MRP) system accepts product orders from customers via the B2B web front and performs material requirements planning. The system's goal is to respond quickly and effectively to changing requirements to minimize production and product delivery delays. The system updates orders automatically and predicts daily planned orders, which can be replaced by a manager.

The daily planned orders are delivered to a job shop, which simulates real-time scheduling and production. Meanwhile, the system determines the materials needed for the daily planned orders and sends raw material orders to appropriate suppliers. The job shop simulator sends finished parts/products to the MRP system, where the manager can change some initial parameters (e.g., lead time, lot size) for scheduling production plans to meet customer order requirements. A supplier can view the status of a material order and acknowledge receipt of the order.

MRP, MPS (Master Production Schedule), BOM (Bill of Material), JOB (Job shop simulator), CI (Customer Interface), MI (Manager Interface), and SI (Supplier Interface) are the seven software components that comprise the web application design. MRP validates logins, records product orders, and bills customers. MPS generates an appropriate daily production plan based on a given product demand (for example, a customer order) and other input parameters. The specifics of MPS are outside the scope of this paper. BOM determines the parts needed for each product to be manufactured and sends raw material orders to suppliers.

JOB is a job shop component that simulates real-time manufacturing plant scheduling and production.

Web interfaces for various users are made possible by CI, MI, and SI.

To concentrate our illustration on risk methodology, we simplify a B2B process by assuming that all payments between a customer and a manufacturer, or a manufacturer and a supplier, are made offline. To make our analysis approach more understandable, we present the workflow in three use cases, each with one scenario: customer orders products, manager adjusts demands/requirements, and supplier acknowledges material order are the three use cases. We assume that the likelihood of each scenario occurring is 0.4, 0.2, and 0.4, respectively.

Figure 3.31 depicts a workflow diagram of a process for handling product orders from customers. We assume customers have established business relationships with the manufacturing enterprise, so MRP is only available to these customers. Customers and suppliers must thus login through a web front. The workflow begins with a dark circle and concludes with a white circle.

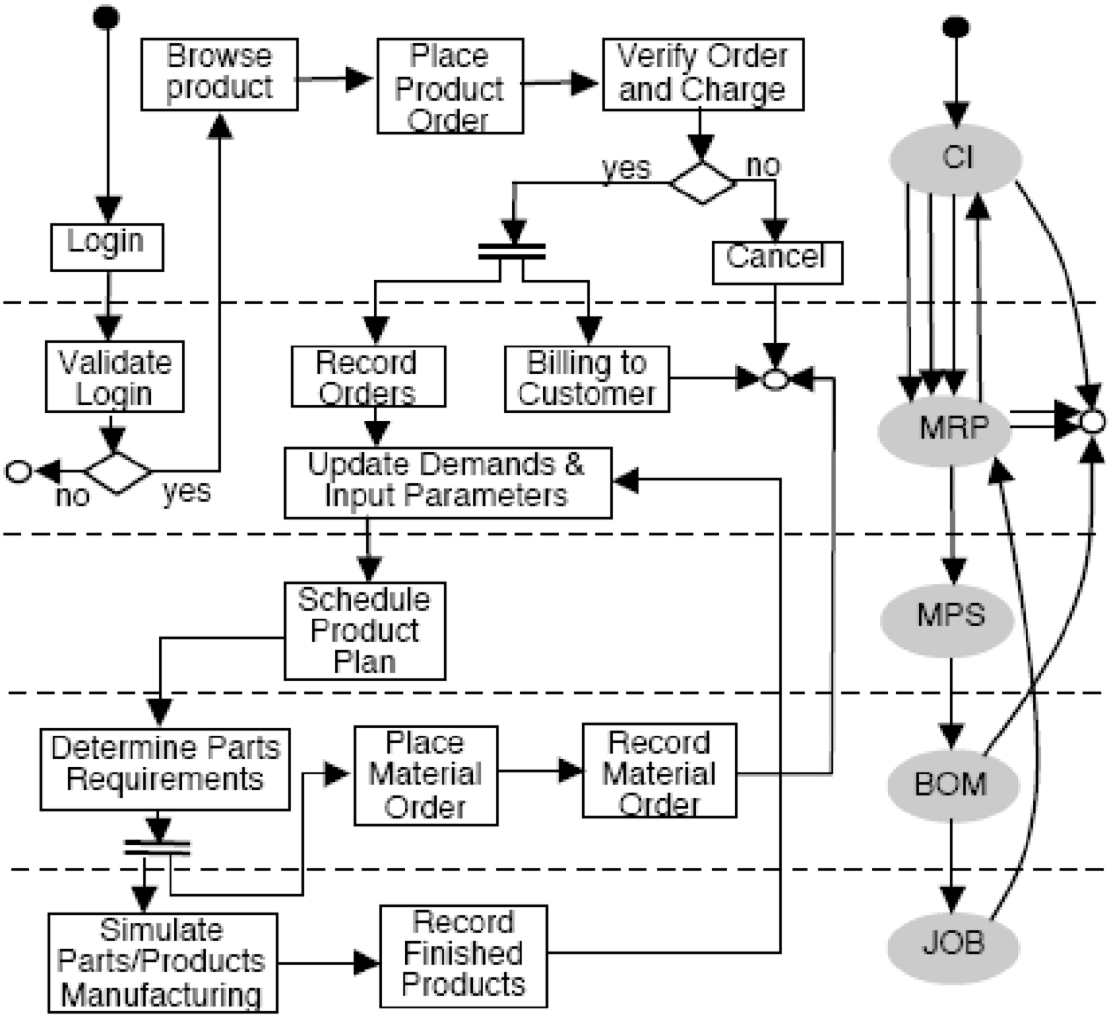


Figure 3.31 - Workflow diagram of a customer use case.

It depicts both sequential (via the directed link) and concurrent (via the double bar) activity among various components. For example, after determining the parts needed for product production, the BOM sends the results to the job shop to simulate product order production while concurrently placing raw material orders with appropriate suppliers. The workflow diagram is expanded to include software components that are in charge of the functions/executions of these activities. The components are depicted in the figure as oval shapes on the right. The arrows connecting the components represent their interactions.

The second component of the design is a hardware deployment strategy. Figure 3.32 depicts the required hardware platform design. The MRP system, like many web applications, has a three-tier architecture. The presentation layer necessitates the use of a web server, W. There are three application servers in the business logic layer: A1, A2,

and A3. Finally, a data layer employs four database servers: D1 - D4, each of which facilitates data retrieval from various databases, as shown at the bottom of the figure. The servers/hosts are linked via LAN, with a firewall separating the web server W, and the three application servers' LAN access.

The deployment plan annotates each server with software components that rely on its services, which appear in the form of a square box on top of each server. As shown in Figure 3.32, a manager can log in remotely to MRP via MI, both of which are hosted on application server A1. MRP and MI enable him to monitor production status and adjust product demand. MRP must retrieve password data because it is also in charge of login validation. As a result, unlike other application servers, A1 must be linked to D1. The analysis for assessing security risks associated with each of these software components from a given design is described in the following section.

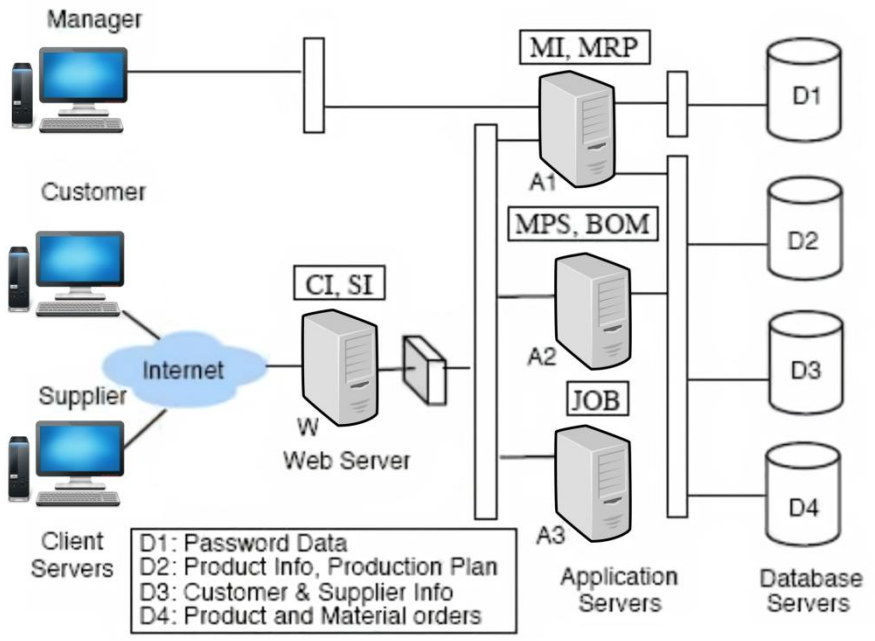


Figure 3.32 - Hardware architecture and deployment plan.

3.4.2 Security risk assessment

The proposed risk methodology begins with calculating interaction rates for each scenario of the web application design. As illustrated in Figure 3.33, MRP has one interaction with MPS but four interactions with it (two from MRP to a terminal state, one to CI, and one to MPS). As shown in table 3.1, the interaction rate from MRP to

MPS is 1/4. M1 represents component interaction rates obtained from a "customer orders product" scenario, where T is a workflow terminal state. Tables 3.2 and 3.3 represent M2 and M3 about the interaction rates in the manager and supplier scenarios, respectively.

Table 3.1 - M1 represents component interaction rates obtained from a «customer orders product» scenario

M1	CI	MRP	MPS	BOM	JOB	T
CI	0	3/4	0	0	0	1/4
MRP	1/4	0	1/4	0	0	1/2
MPS	0	0	0	1	0	0
BOM	0	0	0	0	1/2	1/2
JOB	0	1	0	0	0	0

Table 3.2 – M2 represents component interaction rates obtained from a «manager» scenario

M2	MI	MRP	MPS	BOM	JOB	T
MI	0	2/3	0	0	0	1/3
MRP	1/3	0	1/3	0	0	1/3
MPS	0	0	0	1	0	0
BOM	0	0	0	0	1/2	1/2
JOB	0	1	0	0	0	0

Table 3.3 – M3 represents component interaction rates obtained from a «supplier» scenario

M3	SI	MRP	BOM	T
SI	0	1/2	1/2	0
MRP	1/2	0	0	1/2
BOM	0	0	0	1

Step 2 creates a software component dependency graph. As shown in Figure 1, we compute a transition rate for each pair of components using the corresponding interaction rates obtained from each scenario (M1, M2, and M3) and a given likelihood of each scenario.

Determine hardware platform vulnerabilities from a given deployment plan shown in Figure 3.32 in Step 3 by using a scanning tool (if the systems required host and network configurations are already in place) or a table lookup procedure to public vulnerability databases. Table 3.4 displays the vulnerability sources and counts on our application deployment plan. As previously stated, the table lookup for vulnerability sources is based on the software and environments required on each host to provide their services.

Table 3.4 - Vulnerabilities from hardware deployment plan

Host	Vulnerability Sources	Count
W	Apache -Chunk ,PHP 4.2	2
A1	Jboss ,JRE 1.4.2,Windows, Tomcat -3.2.1	4
A2	Jboss ,JRE 1.4.2,Windows	3
A3	Telnetd	1
D1	Oracle ,TNS Listener	2
D2	Oracle ,TNS Listener	2
D3	Oracle ,TNS Listener	2
D4	Oracle ,TNS Listener	2

Next, we determine the host services required by the component to calculate its vulnerability ratio, Vi. The customer interface, CI, for example, requires the web server W for login and ordering products, as well as query retrievals of product information from the database D2 for browsing. Table 3.5 summarizes the results for other components that yielded similar results. The number of vulnerabilities associated with each component is shown in the second to last column. This is calculated by totaling the

number of vulnerabilities in each of the hosts required by the component. The final column displays vulnerability ratios, with an overall system vulnerability ratio of one.

Table 3.5 - Component demands and vulnerability ratios

	W	A1	A2	A3	D1	D2	D3	D4	#Vuls	Vi
CI	1					1			4	0.09
SI	1						1	1	6	0.14
MI		1				1		1	8	0.2
MRP		1			1	1	1	1	10	0.23
MPS			1			1	1		7	0.16
BOM			1				1	1	7	0.16
JOB				1					1	0.02

Figure 3.33 represents the resulting graph from Step 2), with annotated transition rates and vulnerability ratios (in bold), heuristic estimates of the design components' vulnerability. We assume that the system has no vulnerability at the starting point S (a solid circle) but has a vulnerability ratio of one at the terminating point T (a white circle).

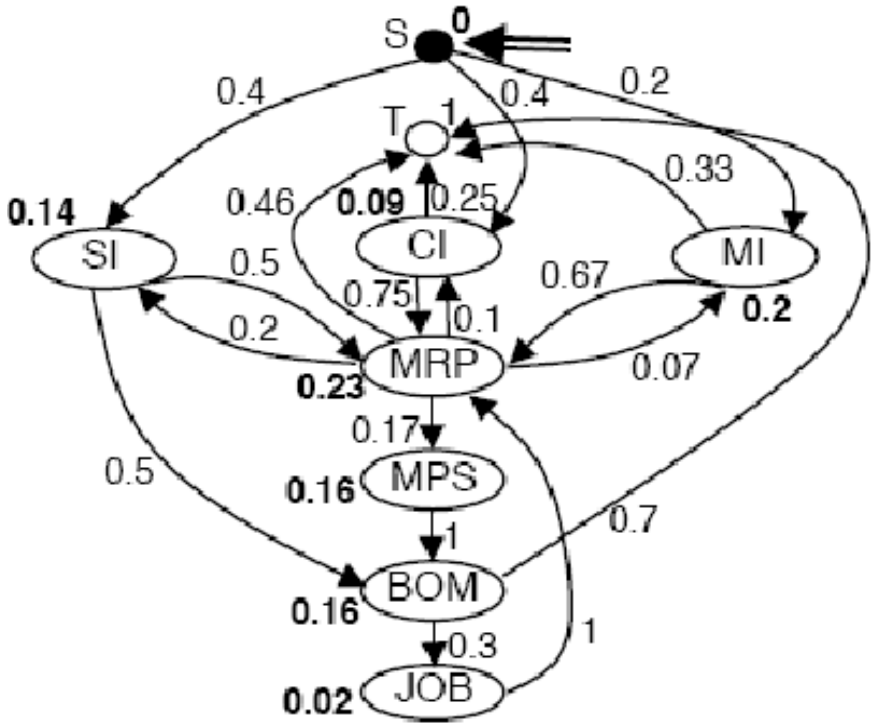


Figure 3.33 - A dependency graph of software components

Step 4 uses the approach described in Figure 3.30 to determine the likelihoods of attacks on (or security breaches in) each component. For example, the probability of CI being attacked (via S → CI) is 1 x 0.4 x 0.09 = 0.036. It should be noted that the path to CI from MRP (via S → CI → MRP → CI) is ignored because it produces a lower likelihood and thus does not produce the desired worst case. The probability of MRP being attacked is calculated as the sum of the probabilities of MRP being reached from CI, MI, SI, and JOB, i.e., 0.06+0.02+0.04+0.01= 0.14. The first row of Table 3.7 contains a summary of component security breach likelihoods. In Step 5, we estimate the consequences of adverse actions in each component by categorizing them into five groups: CT (catastrophic: 0.95), CR (critical: 0.75), MG (marginal: 0.5), and MN (minor: 0.25) (ISO, 2002). Table 3.6 shows the estimated severity for each component. We can calculate the risk of each component by taking the maximum severity obtained in each component and applying standard quantification to it, as shown in Table 3.7. MRP is the most dangerous component, with 34.31% risk, while MI is the least dangerous, with 2.64% risk.

Table 3.6 - Severity analysis of the web application design

SW units	Malicious actions	Consequences	Severity
CI	change product order increase access level	order unverified increase vulnerabilities	MN **CR**
MRP	change product order change demands change charges access customer info access product info	incorrect charge ,plan over /under stock lose profits lose financial info lose product info	MG MG CT **CT** MG
MPS	change schedule access product info access supplier info	incorrect plans lose product info lose supplier info	**CR** MG MG
BOM	change parts increase access level access product info access supplier info	incorrect product plan increase vulnerabilities lose product info lose supplier info	**CR** MG MG MG
JOB	change finished time access product info	late delivery ,lose cust . lose product info	MG **MG**
MI	change prod .status increase access level	wrong update increase vulnerabilities	MN **MN**
SI	increase access level	increase vulnerabilities	**MG**

The overall system risk is calculated by multiplying the likelihood of reaching the terminating point, T, with its vulnerability and severity. In this case, we'll use CR, which is a severity category for most of all components. Table 3.4 shows that the overall system's security risk is about 50%, which can be used as a baseline to compare

with alternatives. These risks provide relative measures to aid decision-making on security options from various designs.

Table 3.7 - Risk analysis results

	CI	SI	MI	MRP	MPS	BOM	JOB	T
Likelihoods	0.04	0.06	0.04	0.14	0.07	0.08	0.13	0.665
Severity	0.75	0.5	0.25	0.95	0.75	0.75	0.5	0.75
Risk	0.03	0.03	0.01	0.13	0.05	0.06	0.07	0.499
Total Risk, %	7.13	7.39	2.64	34.31	14.51	16.77	17.25	

3.5 Analysis of the results of modeling attacks on Web Applications

Penetration testing research is an essential procedure that may assist in identifying and correcting security flaws in online applications. Installing an exploratory web application for this purpose can be a difficult task, but certain procedures must be taken to guarantee that the testing is successful, efficient, and ethical.

The first stage is to specify the scope and goals of the penetration testing investigation. This will assist guide the web application's development and deployment, as well as the testing technique. The scope may, for example, be confined to specific sections of the web application, such as the login system or payment processing, or it may comprise the whole web site. The goals may include finding typical vulnerabilities such as SQL injection or cross-site scripting, or testing for specific threats that the web application is likely to face.

After defining the scope and objectives, the following step is to select an appropriate platform for hosting the web application. Depending on the demands of the study, this might be a cloud service provider, a virtual private server, or a physical server. It is critical to verify that the platform fulfills the web application's technical needs, such as the programming language and database system employed.

The online application should be built using a penetration-testing-friendly framework, such as OWASP ZAP or Burp Suite. These frameworks provide automated

tools for testing web applications and identifying possible vulnerabilities in a systematic manner. Moreover, the online application should be setup to emulate actual circumstances such as user authentication, session management, and input validation that the web application may experience in production. This will make the testing process more realistic and aid in the identification of possible vulnerabilities that may be overlooked in a simpler setting.

After developing and configuring the web application, it should be deployed and made available to the testing team. Configuring firewalls, DNS settings, or other network settings to allow external access to the web application may be required. The testing procedure may begin once the web application is accessible. A range of tools and techniques, including as vulnerability scanning, exploitation, and manual testing, should be used to accomplish the testing. All discoveries should be documented and prioritized according to their seriousness and possible effect.

Lastly, any vulnerabilities discovered throughout the testing process should be addressed and validated as repaired. Further testing should be performed to confirm that the web application is secure and satisfies the study objectives.

While doing penetration testing research, it is critical to comply to industry best practices and ethical norms. Obtaining sufficient authority and informed consent, ensuring confidentiality and privacy, and minimizing harm to the target system or data are all part of this. You may assist guarantee that your exploratory web application is successful and ethical in detecting and fixing security risks by following these principles and using a methodical approach.

CONCLUSION

The importance of web application security cannot be overstated in today's digital age. With online applications being used for everything from personal banking to e-commerce to social networking, the risks of cyber threats are high.

A number of techniques are used to improve the security of web applications. Penetration testing is a valuable tool for identifying web application vulnerabilities. Penetration testing can help organizations prioritize remediation efforts, avoid costly data breaches, and ensure the security and privacy of user data.

In addition to penetration testing, organizations can also take other measures to improve web application security. For example, implementing a strong authentication system, such as two-factor authentication, can help prevent unauthorized access. Using encryption to protect sensitive data can also help improve security. Regular software updates and patching can also reduce the risk of attacks. Moreover, education and training are essential in ensuring web application security. Regular security awareness training can also help employees stay up to date with the latest trends and best practices in web application security.

In this work the architecture of web applications was analyzed. The architecture of a web application usually consists of a front-end built with HTML, CSS and JavaScript and a back-end that consists of a web server, file storage, and a database. In addition to a web server, this can also be DNS, a load balancer, Content Delivery Network, various additional services, etc. From the analysis of this architecture, it follows that protecting a web application is not a trivial task and requires a complicated complex solution.

The analysis of the main vulnerabilities of web applications was carried out based on international standards and using well known Penetration Testing tools.

To estimate the security level of web applications, in the first phase it is proposed to perform penetration testing.

Based on the Damn Vulnerable Web Application (DVWA) platform, a real simulation of the 4 most dangerous types of attacks was performed: Brute Force, SQL injection, XSS and CSRF attacks.

The obtained Results of Penetration Testing can be used to assess the resistance of a web application to certain types of attacks and for the subsequent development of recommendations for protection against them.

In the second part of the work, a quantitative assessment of the security risks of the components and the entire web application was carried out for one example using the modified OWASP risk rating methodology.

The calculation results can be used to develop protection for individual software components of a web application, as well as recommendations for securing the entire web application.

The MRP (Material Requirements Planning) component is subject to the greatest risk in the considered example of web application, the risk is the highest and amounts to 34.31%, and the risk of damage to the MI (Manager Interface) component is the smallest and amounts to 2.64%. It follows that the protection of the MRP component requires the most attention.

The risk for the entire application was about 50%. This result may not satisfy web application users. To increase the security level of the entire application, we should use the results of risk calculations of individual components to determine the most vulnerable application components. We have three of them: JOB, BOM (Build of Material) and MPS (Master Production Schedule).

REFERENCE

1. OWASP. OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/. (2020).
2. OWASP. (2022). OWASP Top 10 - 2022. URL: https://owasp.org/Top10/.
3. International Organization for Standardization. ISO/IEC 27001:2013 Information technology — Security techniques — Information security management systems — Requirements. URL:https://www.iso.org/standard/54534.html. (2013).
4. Gamaledin, Y., Alsadoon, A., & Hussain, A. J. Web Application Penetration Testing: A Comprehensive Survey. *IEEE Access*. 2020. No 8. P. 55601-55619.
5. Ali, M. T., Abuagoub, A., & Zeki, A. M. Web application security testing: A systematic literature review. *Journal of Network and Computer Applications*. 2017. Vol.87. P.49-73.
6. Gietzmann, M., & Meinel, C. The state of the art in web security testing. *Computing*. 2019. Vol. 101. No. 1. P. 59-86.
7. Khan, Z. U., Soomro, T. R., & Lakhan, A. A comprehensive survey of web application security vulnerabilities and countermeasures. *Journal of Ambient Intelligence and Humanized Computing*. 2021. Vol. 12. No 9. P. 8723-8744.
8. The Web Application Hacker's Handbook. URL: https://www.wiley.com/en-us/The+Web+Application+Hacker%27s+Handbook%3A+Finding+and+Exploiting+Security+Flaws%2C+2nd+Edition-p-9781118026472
9. Shravan, K., Neha, B., & Pawan, B. Penetration Testing : A Review. *Compusoft*. 2014. Vol 3. P.752-757.
10. Pearl, J. Graphical models for probabilistic and causal reasoning. *The Computer Science and Engineering Handbook*. 1997. P.697-714.
11. Georgia Weidman. Penetration Testing: A Hands-On Introduction to Hacking. San Francisco: No Starch Press, 2014. 528 p.
12. Yacoub, S. M., Cukic, B., and Ammar, H. H. Scenario-based reliability analysis of componentbased software. *ISSRE '99*: Proceedings of the 10th International Symposium on Software Reliability Engineering. Washington, DC, USA. IEEE Computer Society. 1999. P. 22.

13. Justin Seitz. Black Hat Python: Python Programming for Hackers and Pentesters. San Francisco: No Starch Press, 2015. 192 p.

14. V. Krishnamoorthy, K. A. M. Hassan, and N. M. M. Khamad. Information and Network Security (AINS). *IEEE Conference on Application*. 2019. P. 345-348.

15. ISO (2002). Risk management vocabulary guidelines for use in standards. *ISO Copyright Office*, Geneva. 2002.

16. Barna, P., Frasincar, F., and Houben, G.-J. A workflow-driven design of web information systems. *ICWE '06*: Proceedings of the 6th international conference on Web engineering. New York, NY, USA. ACM Press. 2006. P. 321–328.

17. Qiang, L., Khong, T. C., San, W. Y., Jianguo, W., and Choy, C. A web-based material requirements planning integrated application. *EDOC '01*: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing. Washington, DC, USA. IEEE Computer Society. 2001. P. 14.