PUCPR - Pontifical Catholic University of Paraná

Blue Team

Implementation of Network Security Infrastructure

Bernardo Walker Leichtweis
Cassio Vieceli Filho
Enzzo Machado Silvino

Curitiba - PR

May 2025

Contents

Executive Summary

This project proposes the implementation of a network security infrastructure structured in three main layers: **firewall**, **DMZ (demilitarized zone)**, and **internal network**. The goal is to ensure the protection of the organization's information based on the pillars of information security - *confidentiality*, *integrity*, *availability*, and *non-repudiation*.

Firewall Layer

The **firewall**, based on *pfSense*, will be responsible for filtering and controlling all incoming and outgoing network traffic. For secure remote connections, *WireGuard*, a modern VPN, will be used. Real-time traffic monitoring will be performed with *Snort*, acting as an IDS/IPS to detect and mitigate intrusion attempts and denial-of-service attacks (*DoS/DDoS*).

DMZ - Demilitarized Zone

The **DMZ** will host services aimed at the external public, logically isolated from the internal network to contain potential attack vectors. A web application and a complete email server with *MailCow* will be deployed, integrating SMTP, IMAP, antispam, and antivirus functions, providing secure and auditable communication. This separation ensures that, even in the event of a compromised exposed service, internal assets remain protected.

Internal Network

The **internal network** will house the most critical services, such as:

- **MySQL**: Database storing critical web application information, with restricted access and encryption at rest for data protection.
- **SAMBA**: Network file-sharing system with user-based permission control and access auditing to prevent leaks.
- **Wazuh**: Centralized security monitoring platform, acting as SIEM, XDR, and DLP, using YARA rules for advanced threat detection.
- **Light LDAP and HashiCorp Vault**: Identity and access management. LDAP authenticates users, while Vault stores and automatically rotates passwords.
- **Restic**: Automated and encrypted backup tool with periodic scheduling, ensuring information availability and integrity in case of failures or incidents.

Conclusion

This architecture ensures a secure, organized, and resilient foundation for operations, reducing risks and ensuring service continuity in compliance with information security best practices.

Network Topology

The network is structured with the firewall as the central point, connecting the internal network, the DMZ, and external interfaces. The topology is illustrated below.

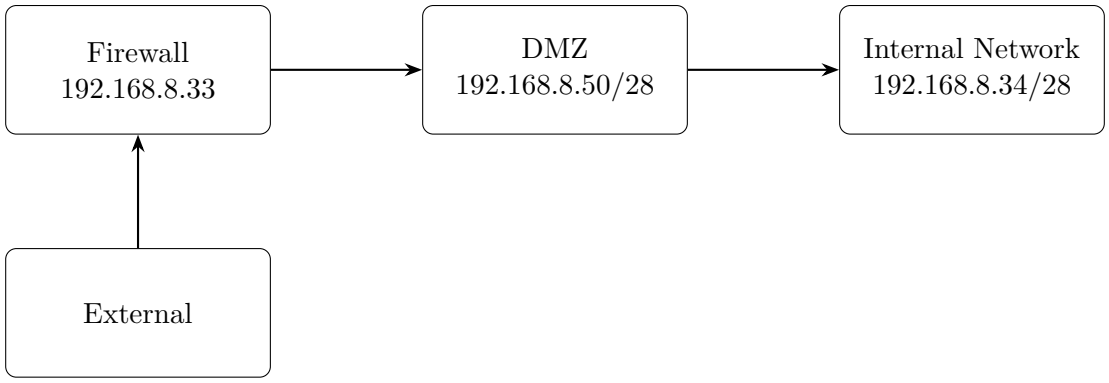

Figure 1: Network Topology Diagram

Machine Descriptions

- **Firewall (pfSense)**:
  - Address: 192.168.8.33
  - Hostname: ENE48-EP-PUCPR.home.arpa
  - Operating System: FreeBSD 14.0-CURRENT
  - pfSense Version: 2.7.2-RELEASE
- **DMZ**:
  - Address: 192.168.8.50/28
  - Hostname: ene50-ep-pucpr
  - Default Gateway: 192.168.8.49 via eth0
  - Distribution: Ubuntu 24.04.2 LTS (Noble)
  - Kernel: Linux 6.8.0-57-generic
- **Internal Network**:
  - Address: 192.168.8.34/28
  - Hostname: ene49-ep-pucpr
  - Default Gateway: 192.168.8.33 via eth0
  - Distribution: Ubuntu 24.04.2 LTS (Noble)
  - Kernel: Linux 6.8.0-57-generic

Apache Web Server

The **Apache** web server is deployed in the DMZ to host the *sitefoda.com* website. This configuration ensures secure access via HTTP and HTTPS, automatically redirecting HTTP traffic to HTTPS. Security headers and directory restrictions have been implemented to protect against common web vulnerabilities. A self-signed SSL certificate and specific permissions are used to enhance security.

Security Directives

The following directives are added to `apache2.conf` to minimize information disclosure and disable unnecessary features:

```
ServerTokens Prod
ServerSignature off
FileETag None
TraceEnable off
```

Listing 1: Security Directives in apache2.conf

- `ServerTokens Prod`: Limits server information in HTTP headers to "Apache".
- `ServerSignature off`: Disables server signature on error pages.
- `FileETag None`: Removes ETag headers to prevent inode-based attacks.
- `TraceEnable off`: Disables the TRACE method to prevent cross-site tracing.

Directory Configurations

Directory configurations restrict access and disable unnecessary features:

```
<Directory />
Options FollowSymLinks
AllowOverride None
Require all denied
</Directory>

<Directory /usr/share>
AllowOverride None
Require all granted
</Directory>

<Directory /var/www/>
Options -Indexes
AllowOverride None
Require all granted
<LimitExcept GET POST HEAD>
Require all denied
</LimitExcept>
</Directory>
```

Listing 2: Directory Configurations in apache2.conf

- Root directory (/): Denies all access except for symbolic links.
- /usr/share: Grants access to shared resources.
- /var/www/: Allows GET, POST, and HEAD methods, disables directory listing, and denies other HTTP methods.

DoS Mitigation

To reduce the risk of denial-of-service (DoS) attacks, the timeout is reduced:

```
Timeout 60
```

Listing 3: Timeout Configuration

This sets the server timeout to 60 seconds, reducing resource exhaustion risks.

Security Configuration

The file /etc/apache2/conf-enabled/security.conf contains additional security headers and rules:

```
RedirectMatch 404 /.git
Header set X-Content-Type-Options: "nosniff"
Header set X-XSS-Protection "1; mode=block"
Header always set Referrer-Policy "strict-origin"
Header always append X-Frame-Options SAMEORIGIN
```

Listing 4: Security Configuration in security.conf

- RedirectMatch 404 /.git: Prevents access to .git directories.
- X-Content-Type-Options: "nosniff": Prevents MIME type sniffing.
- X-XSS-Protection: Enables XSS filtering in browsers.
- Referrer-Policy: "strict-origin": Limits referrer information to the origin.
- X-Frame-Options SAMEORIGIN: Prevents clickjacking by restricting framing.

HTTP Virtual Host

The file /etc/apache2/sites-available/000-default.conf redirects all HTTP traffic to HTTPS:

```
<VirtualHost *:80>
ServerName www.sitefoda.com
ServerAlias sitefoda.com
Redirect permanent / https://sitefoda.com
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Listing 5: HTTP Virtual Host Configuration

This ensures all traffic is encrypted, redirecting to `https://sitefoda.com`.

HTTPS Virtual Host

The file `/etc/apache2/sites-available/ssl.conf` configures the HTTPS virtual host with SSL:

```
<VirtualHost *:443>
ServerName www.sitefoda.com
ServerAlias sitefoda.com
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
ServerAdmin webmaster@localhost

<FilesMatch ".(?:cgi|shtml|phtml|php)$">
SSLOptions +StdEnvVars
</FilesMatch>
<Directory /usr/lib/cgi-bin>
SSLOptions +StdEnvVars
</Directory>

SSLEngine on
SSLCertificateFile /etc/ssl/certs/apache.crt
SSLCertificateKeyFile /etc/ssl/private/apache.key
</VirtualHost>
```

Listing 6: HTTPS Virtual Host Configuration

- `SSLEngine on`: Enables SSL/TLS.
- `SSLCertificateFile` and `SSLCertificateKeyFile`: Specify the self-signed certificate and key.

Permissions

File and directory permissions are set to enhance security:

```
sudo chmod 750 /etc/apache2/conf*
sudo chown -R www-data:www-data /var/www
```

Listing 7: Permission Commands

- `chmod 750 /etc/apache2/conf*`: Restricts configuration files to the owner and group.
- `chown -R www-data:www-data /var/www`: Sets the web server user as the owner of the web root.

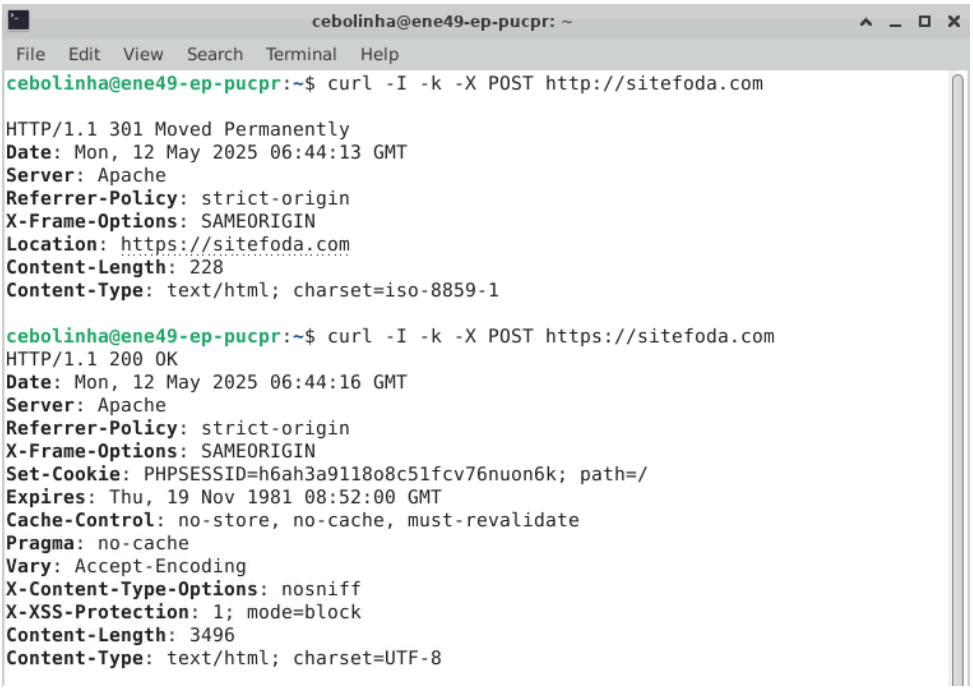

Figure 2: HTTP to HTTPS Redirect

```
cebolinha@ene49-ep-pucpr:~$ curl -I -k -X TRACE https://sitefoda.com
HTTP/1.1 405 Method Not Allowed
Date: Mon, 12 May 2025 06:47:20 GMT
Server: Apache
Referrer-Policy: strict-origin
X-Frame-Options: SAMEORIGIN
Allow:
Content-Length: 222
Content-Type: text/html; charset=iso-8859-1
```

Figure 3: TRACE Method Not Allowed

Web Application

The web application was developed with a focus on simplicity, security, and user experience. We used **HTML**, **Tailwind CSS**, **JavaScript**, and **PHP** for its construction. The application is hosted in the DMZ, adhering to the principle of layer separation and protecting sensitive data in the internal network.

General Description

The homepage displays *quotes* (motivational or inspirational phrases) for users. The quotes are dynamically generated using the system command `fortune`, executed securely on the server.

The system consists of four main pages:

- **Home**: Displays a randomly generated quote.
- **Register**: New user registration page.
- **Login**: Authentication page.

- **Profile**: User profile page, where users can save their favorite quotes.

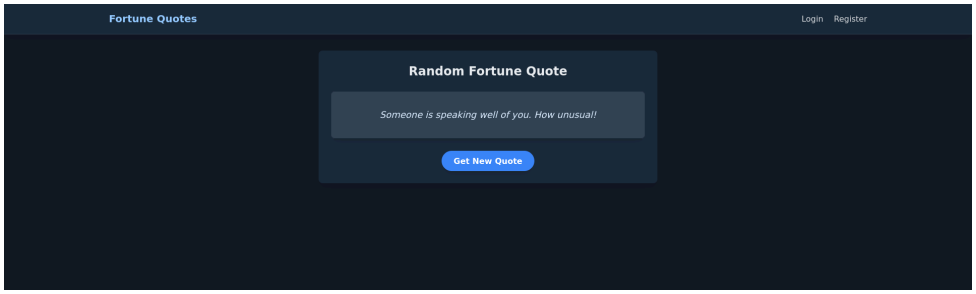

Figure 4: Home Page

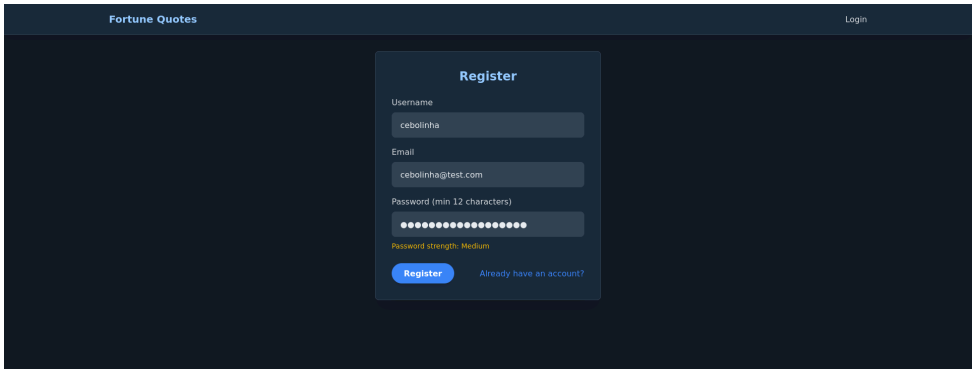

Figure 5: Register Page

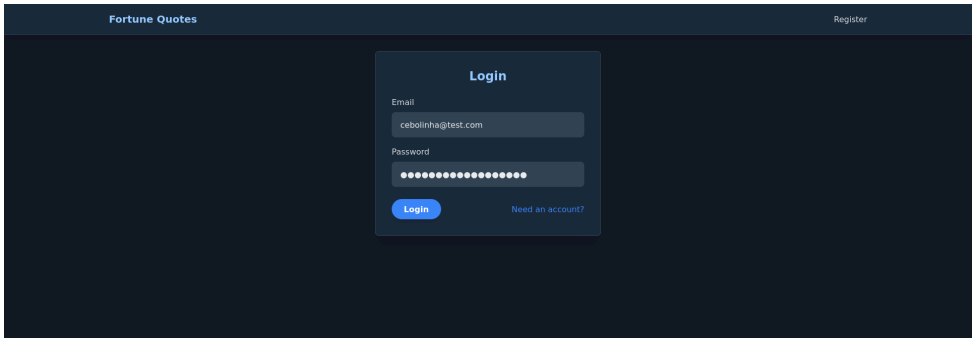

Figure 6: Login Page

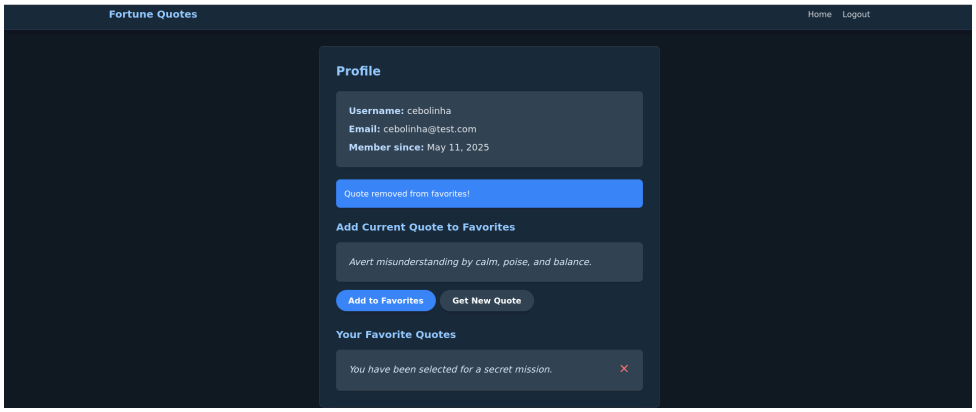

Figure 7: Profile Page

Command Execution Security

Quotes are generated with `shell_exec` and the `fortune` command, but with rigorous sanitization to prevent arbitrary execution:

```
while(empty($fortune) || strlen($fortune) >= 700) {
    if (is_executable($fortune_path)) {
        $fortune = shell_exec(escapeshellcmd($fortune_path));
    }

    if(empty($fortune) || strlen($fortune) >= 700) {
        $fortune = '';
    }
}

if (empty($fortune)) {
    $default_quotes = [...];
    $fortune = $default_quotes[array_rand($default_quotes)];
}

return htmlspecialchars(trim($fortune), ENT_QUOTES |
   ENT_SUBSTITUTE, 'utf-8');
```

Listing 8: Sanitization of shell_exec Command

Secure Password Storage

During user registration, passwords are encrypted using `bcrypt` via PHP's native function:

```
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

Listing 9: Password Hashing with bcrypt

Secure SQL with Prepared Statements

All database interactions use `prepared statements` to prevent SQL injection:

```
$stmt = $db->prepare("INSERT INTO users (username, email,
    password) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $username, $email, $hashed_password);
```

Listing 10: Insertion with Prepared Statement

Input Sanitization

Data received via forms is sanitized using appropriate filters:

```
$username = filter_input(INPUT_POST, 'username',
    FILTER_SANITIZE_SPECIAL_CHARS);
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL)
    ;
```

Listing 11: Input Sanitization

Favorites Functionality

On the profile page, users can save their favorite quotes. These actions are also protected by prepared statements and session validations.

Mailcow Email Service (Postfix + Dovecot)

To provide a complete, secure, and auditable email service, we opted for the deployment of **Mailcow**, which integrates *Postfix* as the MTA (Mail Transfer Agent) and *Dovecot* as the MDA (Mail Delivery Agent), along with antispam, antivirus, webmail, and administrative interface modules.

Installation and Configuration

Mailcow installation is performed via `Docker` using the automated `generate_config.sh` script provided by the official repository. This script generates the necessary configuration files based on the domain and administrator preferences.

By default, Mailcow uses the ports:

- 80 (HTTP)
- 443 (HTTPS)

To avoid conflicts with the Apache web server already present in the DMZ, these ports were changed in the `mailcow.conf` file:

```
1 HTTP_PORT=8080
2 HTTPS_PORT=8443
```

Listing 12: Port Changes in mailcow.conf

Security and Accounts

After installation, the default administrator account password was changed to a secure password, and a new test user account was created.

We adopted a strong password policy with the following criteria:

- Minimum of 12 characters
- Must contain letters and numbers

This policy reduces the risk of unauthorized access, even if credentials are exposed.

Email Service Operation

Mailcow offers a complete and responsive administrative interface, facilitating the management of domains, mailboxes, aliases, distribution lists, and logs. Communication between servers and clients is protected by SSL/TLS, ensuring the confidentiality and integrity of transmitted data.

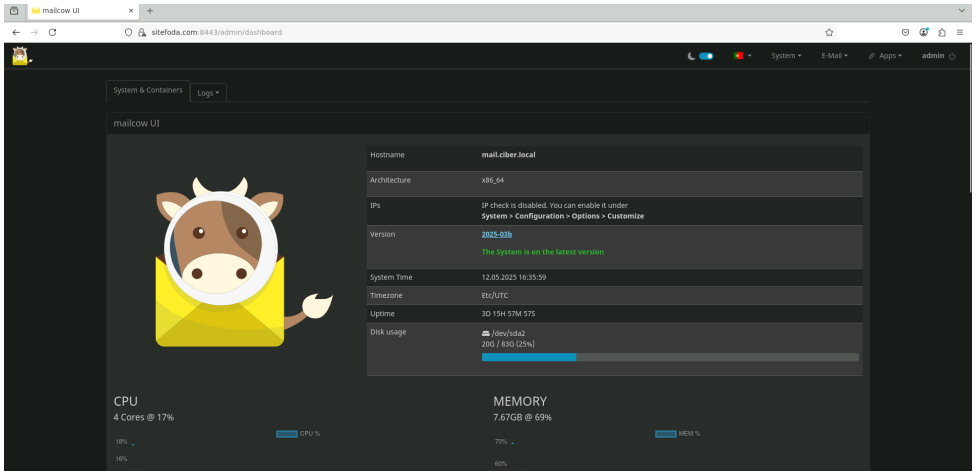

Figure 8: Mailcow Administrative Interface Dashboard

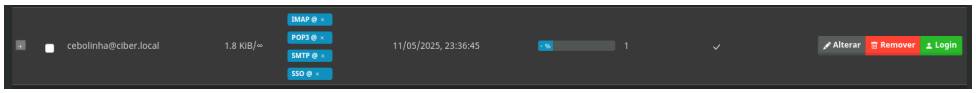

Figure 9: Test User Account Created Successfully

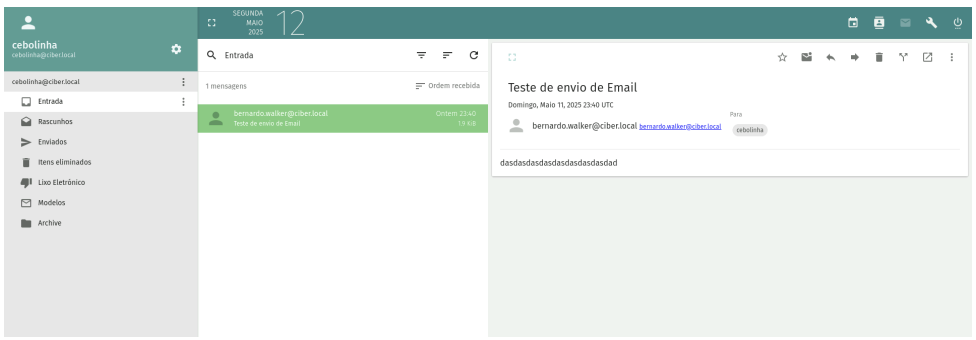

Figure 10: Email Sending and Receiving Tested Successfully

MySQL Server Database

The **MySQL Server** database is responsible for storing sensitive web application information, including user credentials and favorite quotes. It is located in the internal network for enhanced security, accessible remotely only by the application in the DMZ.

Initial Configuration

The first step was to change the administrator (`root`) password and create a new user specifically for the application to access the database remotely:

```
CREATE USER 'remote_user'@'192.168.8.50' IDENTIFIED BY '
   securePassword123';
GRANT ALL PRIVILEGES ON *.* TO 'remote_user'@'192.168.8.50';
FLUSH PRIVILEGES;
```

Listing 13: Creation of remote user with privileges

This user is permitted access only from the DMZ machine's IP (`192.168.8.50`), ensuring logical segmentation between network zones.

Database and Table Creation

The `fortune_quotes` database was created, containing two tables: one for storing user data and another for recording favorite quotes.

```
CREATE DATABASE IF NOT EXISTS fortune_quotes;
USE fortune_quotes;

-- Users table
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX (email)
);

-- Favorite quotes table
CREATE TABLE IF NOT EXISTS favorite_quotes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    quote TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX (user_id)
);
```

Listing 14: Database and Table Creation

The `favorite_quotes` table has a foreign key linked to the `users` table, ensuring referential integrity and allowing automatic deletion of quotes if a user is removed.

User with Restricted Permissions

For testing and simulations, an additional user with minimal permissions was created. This user can only perform read operations (`SELECT`) on the favorite quotes table.

This practice follows the principle of least privilege, essential for database security.

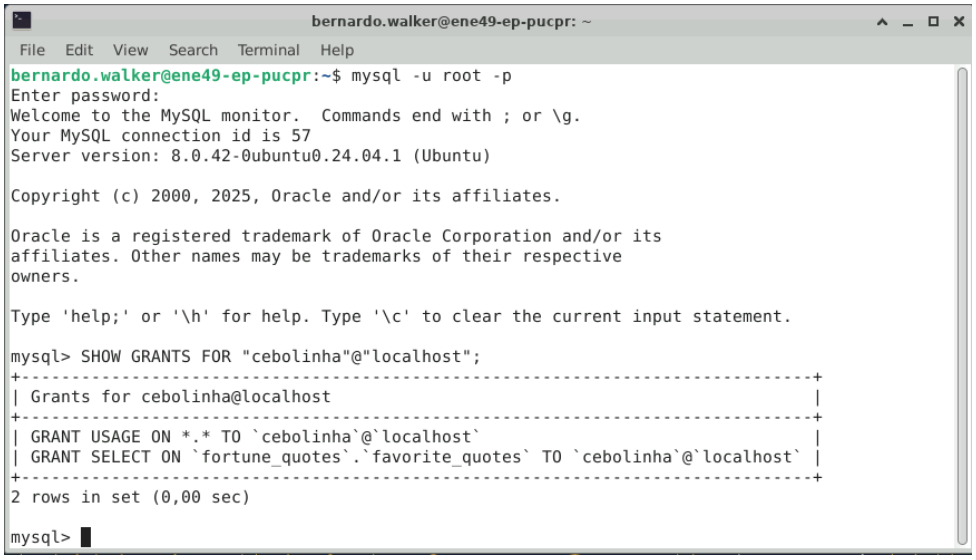

Figure 11: Test User Permissions with `SHOW GRANTS`

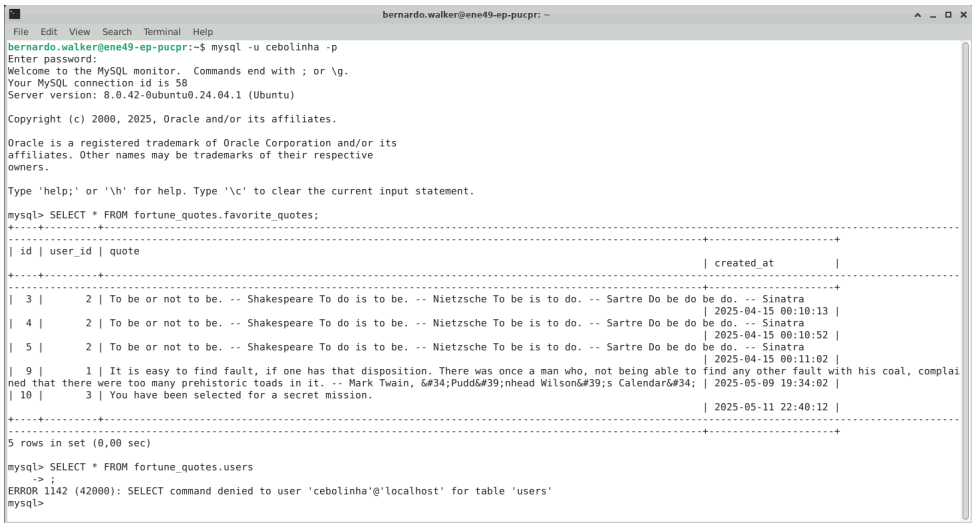

Figure 12: Query to favorite_quotes Table with Test User

SAMBA (SMB) File Sharing

The file-sharing service was implemented using **Samba** with the **SMB3** protocol, which offers native support for encryption of all connections. The following directives were configured in the `smb.conf` file to enhance security:

```
smb min protocol = 3
smb encrypt = required
```

Listing 15: SMB Protocol Security Configuration

These directives ensure that only SMB3 connections are accepted and that encryption is mandatory for all accesses to the Samba server.

Sharing Configuration

The sharing was defined as follows:

```
[InternalFiles]
path = /srv/internal_files
browseable = yes
writable = yes
valid users = cebolinha bernardo.walker enzzo.silvino filho.
   cassio
create mask = 0660
directory mask = 0770
```

Listing 16: Sharing Block in smb.conf

The `/srv/internal_files` directory was created with restrictive permissions:

- **Permissions**: `drwxrwx---` (770)
- **Purpose**: Ensure that only authenticated users have full access.

Auditing and Logging

The log level was adjusted to capture useful access information:

```
log level = 2
```

Listing 17: Log Configuration for Auditing

This allows logging of events such as connections, errors, and file accesses, aiding future audits.

Space Control with Quotas

To prevent excessive disk usage by users, quotas were implemented per group in the filesystem. A group called `smbquota` was created, and authorized users were added:

```
sudo usermod -aG smbquota filho.cassio
sudo usermod -aG smbquota bernardo.walker
sudo usermod -aG smbquota enzzo.silvino
```

```
sudo usermod -aG smbquota cebolinha
```

Listing 18: Adding users to smbquota group

The quota was applied as follows:

```
sudo setquota -g smbquota 921600 1048576 0 0 /
```

Listing 19: Applying quota to smbquota group

Defined Limits:

Limit Type	Value	Meaning
Soft Limit	900 MiB	Warning after reaching (7 days tolerance)
Hard Limit	1 GiB	Prevents writing above the limit
File Limit	Unlimited	No restriction on file count

Table 1: Quota Policy for the `smbquota` Group

Sharing and Access Tests

Practical tests of access and file movement were conducted to ensure the correct functioning of the sharing.

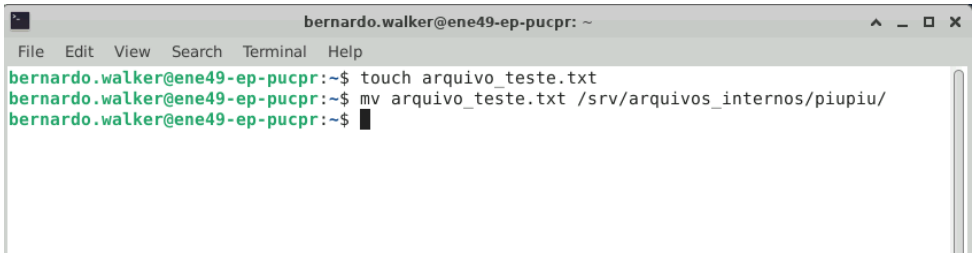

Figure 13: Creation and Movement of File to Shared Folder

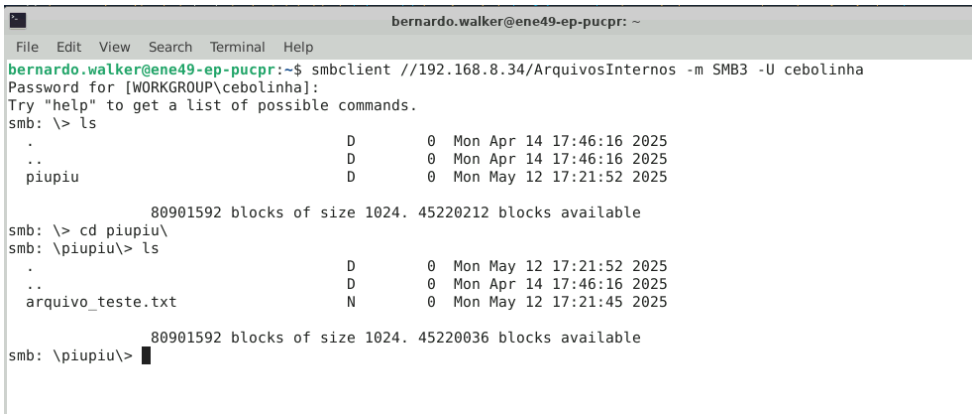

Figure 14: Access to Sharing via smbclient with Test User

Backup with Restic

The backup system was implemented using **Restic**, a modern tool that ensures security, performance, and simplicity. This mechanism performs incremental backups with native encryption and easy integration with `systemd`.

Restic Advantages

- **Security**: All data is encrypted with `AES-256` and authenticated with HMAC, ensuring protection against reading and tampering.
- **Performance**: Incremental backups avoid redundancy, saving time and space.
- **Simplicity and Portability**: Works without a daemon, ideal for scheduling via `systemd` or custom scripts.

User and Permissions

We created the `backupuser` user, exclusively for running the backup process, with secure characteristics:

- System user type (UID below 1000)
- No login shell (`/usr/sbin/nologin`)
- No `home` directory
- No access password
- Group: `nogroup`

Permissions for the repository and sensitive files:

```
sudo chown -R backupuser:nogroup /backups/restic_repo
sudo chown backupuser:nogroup /etc/restic/pass
sudo chmod 600 /etc/restic/pass
```

Listing 20: Repository and Password Permissions

Backup Script

The main script was moved to a secure location:

```
sudo mkdir -p /opt/restic
sudo mv /usr/local/sbin/restic_backup.sh /opt/restic/
   restic_backup.sh
sudo chown backupuser:nogroup /opt/restic/restic_backup.sh
sudo chmod 750 /opt/restic/restic_backup.sh
```

Listing 21: Backup Script Organization

Script Functions:

- Checks if repository disk usage exceeds 50%. If so, the backup is aborted.

- Performs incremental backup of the `/opt` folder.
- Executes intelligent retention: only the last two snapshots are kept with `restic forget --keep-last 2 --prune`.
- All execution is logged in `/var/log/restic/backup.log`.

Systemd Integration

Service: `/etc/systemd/system/restic-backup.service`

```
[Unit]
Description=Daily Restic Backup (Incremental, retention 2)

[Service]
Type=oneshot
User=backupuser
Group=nogroup
ExecStart=/opt/restic/restic_backup.sh
Nice=10
IOSchedulingClass=best-effort
IOSchedulingPriority=7
```

Listing 22: Restic systemd Service

Timer: `/etc/systemd/system/restic-backup.timer`

```
[Unit]
Description=Daily Restic Backup Scheduler

[Timer]
OnCalendar=*-*-* 04:30
Persistent=true

[Install]
WantedBy=timers.target
```

Listing 23: Restic systemd Timer

Important Paths

- `/backups/resticrepo` – Backup repository
- `/etc/restic/pass` – Repository password
- `/opt/restic/resticbackup.sh` – Execution script
- `/etc/systemd/system/restic-backup.service` – Systemd service
- `/etc/systemd/system/restic-backup.timer` – Systemd timer
- `/var/log/restic/backup.log` – Detailed execution log

Test User (Snapshot Reading)

A configuration was created so that the test user can only view snapshots, without write permissions:

```
sudo chmod -R 550 /backups/restic_repo/
sudo chown -R root:cebolinha /backups/restic_repo/
```

Listing 24: Read Permission for Test User

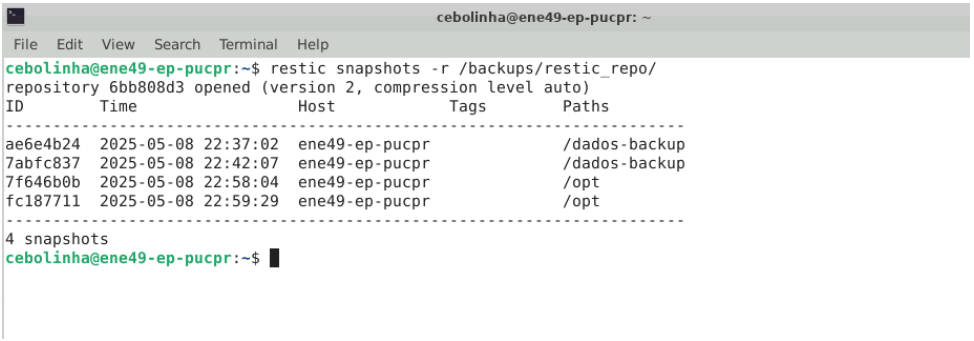

Figure 15: Test User Viewing Snapshots in Restic Repository

Secure Remote Access via SSH

The **SSH** (Secure Shell) protocol was configured with a focus on security and access control, using public key authentication and integration with multifactor authentication.

SSH Server Configuration

The main configuration file used was `/etc/ssh/sshd_config`. The main security directives applied include:

- `PubkeyAuthentication yes` – Enables public key authentication.
- `AuthorizedKeysFile .ssh/authorized_keys` – Standard path for authorized keys.
- `PasswordAuthentication no` – Disables password authentication.
- `PermitRootLogin no` – Blocks direct login as `root`.
- `UsePAM yes` – Kept enabled for Google Authenticator integration.
- `PermitEmptyPasswords no` – Prevents authentication of users with empty passwords.

Public Key Installation

Authorized users' public keys were added to their respective `~/.ssh` directories:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
echo "XXCHAVEXX" > ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Listing 25: Directory Creation and Key Installation

Bypass for Test User

For the `cebolinha` user, a Google Authenticator bypass was configured, requiring only the public key:

```
Match User cebolinha
    AuthenticationMethods publickey
```

Listing 26: Match rule for test user

Connectivity Tests

SSH connectivity tests were conducted between different network segments to verify firewall and routing policies:

- **Internal Network** → **DMZ** on port 22: **Allowed**.
- **DMZ** → **Internal Network** on port 2222: **Allowed**.
- **External Machine** → **DMZ** on port 22: **Allowed**.

- **External Machine → Internal Network** on port 2222: **Blocked** (as expected).

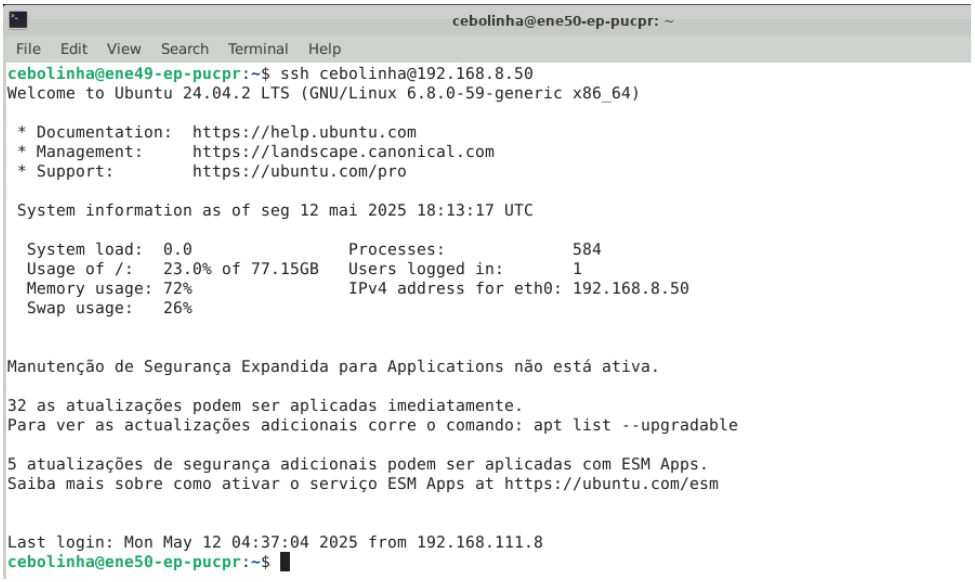

Figure 16: Connection Established from Internal Network to DMZ on Port 22

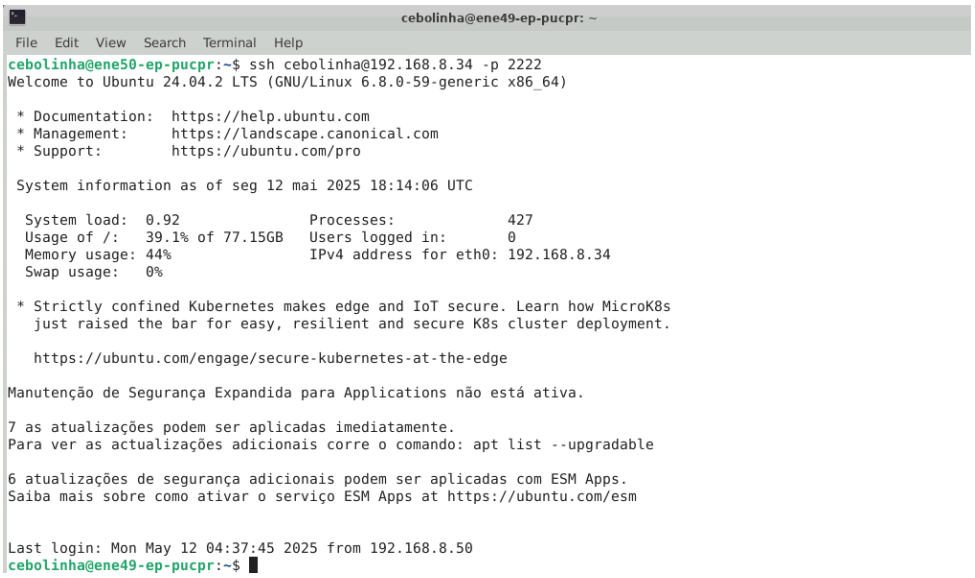

Figure 17: Connection Established from DMZ to Internal Network on Port 2222

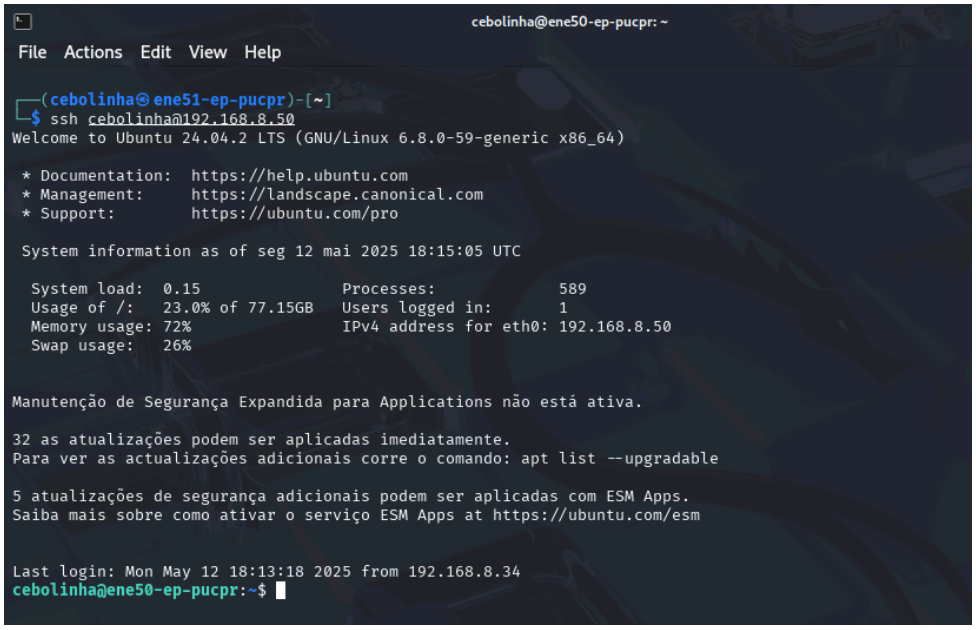

Figure 18: Connection Established from External Machine to DMZ on Port 22

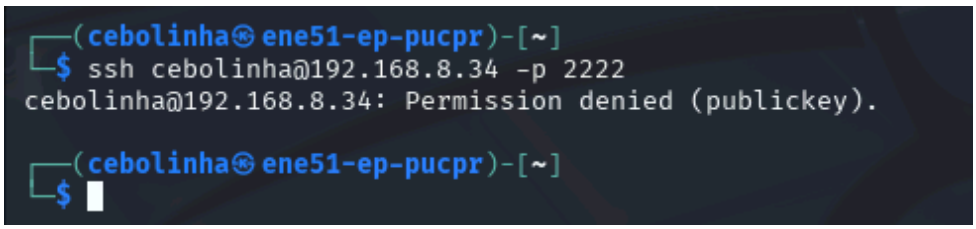

Figure 19: Connection Attempt from External Machine to Internal Network on Port 2222 - Access Denied

Snort (IDS/IPS)

Snort was configured as an Intrusion Detection and Prevention System (IDS/IPS), operating in real-time to protect the Internal Network and DMZ against scans, unauthorized access, and suspicious connections.

Rule Sources

Several rule sources were enabled to maximize threat detection coverage:

- **Snort VRT Rules**: Requires registration on Snort.org and provision of the *Oinkmaster Code*.
- **Snort GPLv2 Community Rules**: Community-maintained rules.
- **Emerging Threats Open Rules (ET Open)**: Free set maintained by Proofpoint.
- **OpenAppID Detectors**: Application detection by signature.
- **Feodo Tracker Botnet C2 IP Rules**: Blocks connections to command and control (C2) IPs.

Rule Update Interval: Daily at 04:00.
Automatic IP Unblocking: After 1 hour.

Manual Rule Update

An initial manual update was performed in the Updates tab to ensure all sources were synchronized:

- Snort Subscriber Ruleset
- Snort GPLv2 Community Rules
- Emerging Threats Open Rules
- Snort OpenAppID Detectors
- Snort AppID Open Text Rules
- Feodo Tracker Botnet C2 IP Rules

Pass Lists (Exclusion List)

To avoid blocking essential internal IPs, an exclusion list was created with the following IPs:

- 192.168.8.50 (Internal Network Server)
- 192.168.8.34 (DMZ Server)

Monitored Interfaces

The configured interfaces for monitoring are:

- **LAN33 (Internal Network)**

- **LAN49 (DMZ)**

For both:

- Snort enabled on the interface
- *Block Offenders* enabled (legacy mode)
- *Kill States* enabled
- Detection mode: **AC-BNFA**, with optimized search
- Passlist applied

Rule Policy

The **Balanced Policy** was chosen, providing a balance between security and connectivity. The **Connectivity** policy, though less restrictive, allowed unwanted scans and tests.

Preprocessors

- **SSH Detection**:
  - LAN33: port 2222
  - LAN49: port 22
- **Portscan Detection**:
  - LAN33: **High** sensitivity
  - LAN49: **Medium** sensitivity
- **ARP Spoofing Detection**: Enabled on both interfaces

Detection Tests

To validate Snort's functionality, the following tests were conducted:

- **Nmap scan from external machine to DMZ**:

```
nmap -sV -O 192.168.8.50
```

Listing 27: Nmap Scan

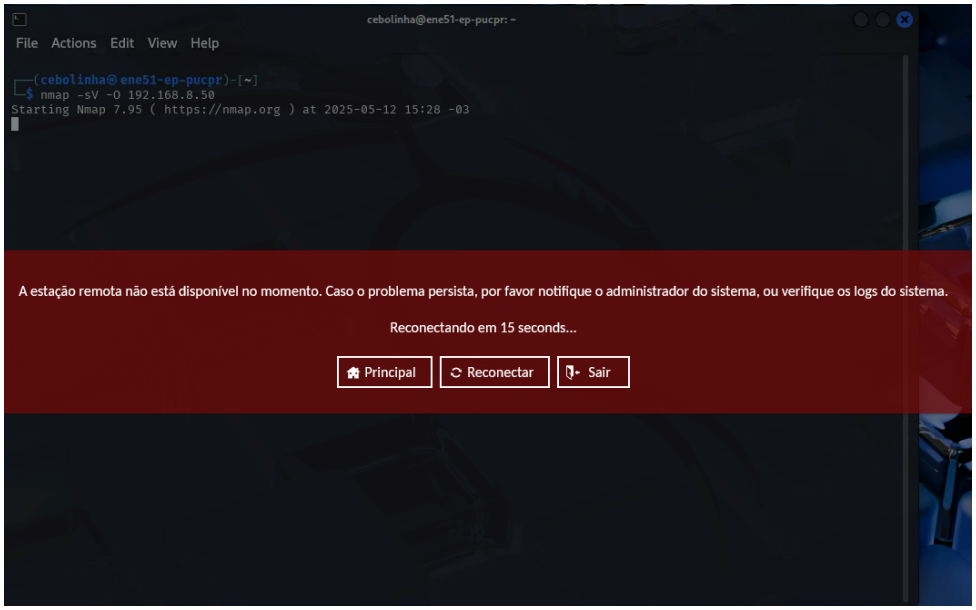

Figure 20: Nmap Scan Blocked and Stopped

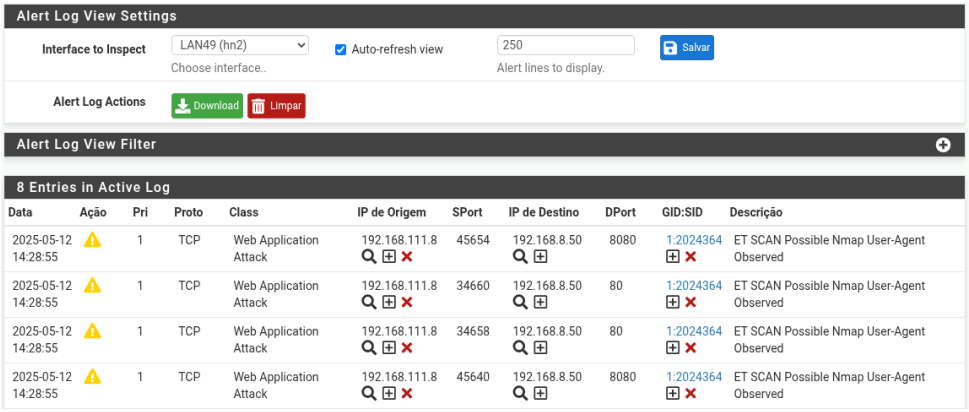

Figure 21: Alert Generated by Snort During Intrusion Attempt

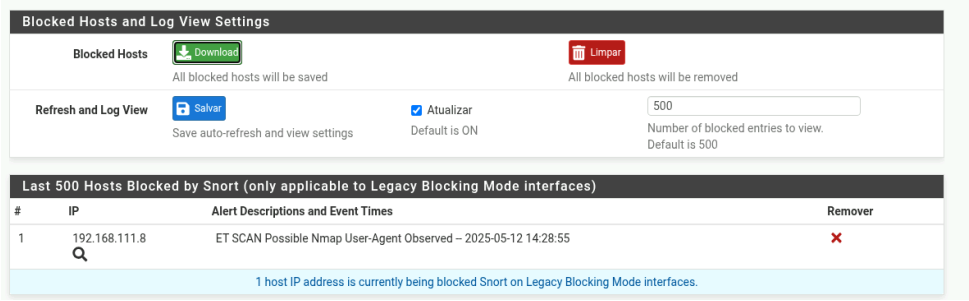

Figure 22: External IP Automatically Blocked by Snort

Light LDAP (LLDAP)

The **LLDAP (Light LDAP)** service was used as a lightweight and modern directory solution, supporting LDAPS (LDAP over TLS) and a web administrative interface. It was implemented via `Docker`, ensuring portability and simplicity in management.

Configuration via Docker Compose

The service was configured with the following `docker-compose.yml` file, exposing port 6360 for LDAPS connections and 17170 for the web interface:

```
version: "3"

services:
  lldap:
    image: lldap/lldap:stable
    ports:
      - "6360:6360"  # LDAPS
      - "17170:17170"  # Web interface
    volumes:
      - lldap_data:/data
      - ./certs/lldap.crt:/app/lldap.crt
      - ./certs/lldap.key:/app/lldap.key
    environment:
      - LLDAP_LDAPS_OPTIONS__ENABLED=true
      - LLDAP_LDAPS_OPTIONS__CERT_FILE=/app/lldap.crt
      - LLDAP_LDAPS_OPTIONS__KEY_FILE=/app/lldap.key
      - LLDAP_JWT_SECRET=XXXXXX
      - LLDAP_KEY_SEED=XXXXX
      - LLDAP_LDAP_BASE_DN=dc=example,dc=com
      - LLDAP_LDAP_USER_PASS=XXXXX
    restart: unless-stopped

volumes:
  lldap_data:
    driver: local
```

Listing 28: LLDAP docker-compose.yml File

Web Administrative Interface

After deployment, the administrative interface can be accessed via browser at:

`http://localhost:17170`

Through this interface, the following actions were performed:

- Created a new user with administrative privileges.
- Created the `cebolinha` user and added it to the `LLDAP-STRICT-READONLY` group, ensuring restricted read-only access.

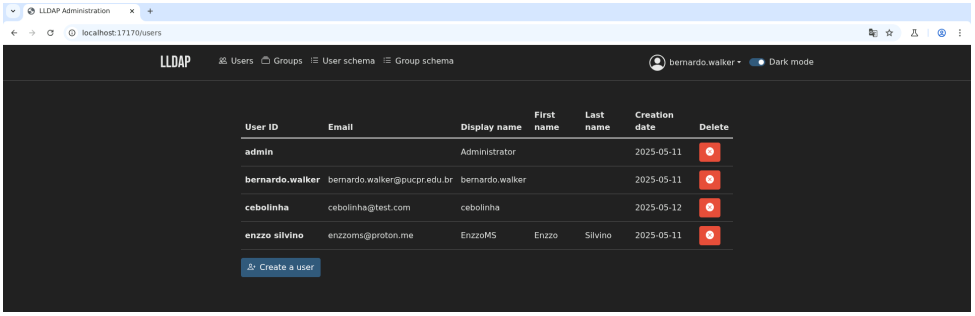

Figure 23: LLDAP Administration Screen

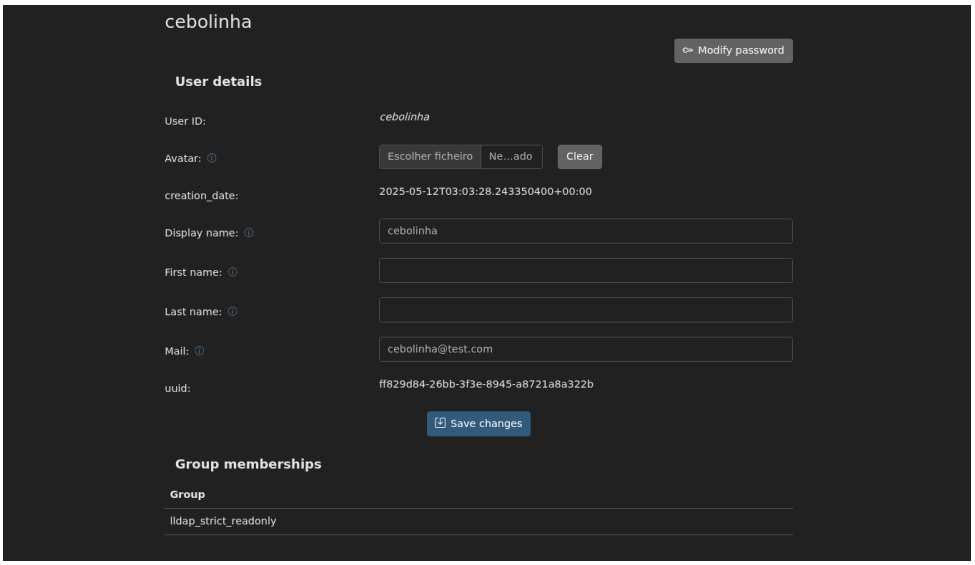

Figure 24: View of Test User (cebolinha)

LDAPSEARCH Query

The `ldapsearch` command was used to validate the structure and read permissions assigned to the test user. Below are examples of searches performed:

```
ldapsearch -x -H ldaps://localhost:6360 -D "uid=cebolinha,ou=
    people,dc=example,dc=com" -W -b "dc=example,dc=com" "(cn=
    cebolinha)"
```

Listing 29: LDAP Query for cebolinha

```
ldapsearch -x -H ldaps://localhost:6360 -D "uid=cebolinha,ou=
    people,dc=example,dc=com" -W -b "dc=example,dc=com" "(cn=admin)
    "
```

Listing 30: LDAP Query for admin user

Figure 25: LDAP Query for cebolinha using `ldapsearch`

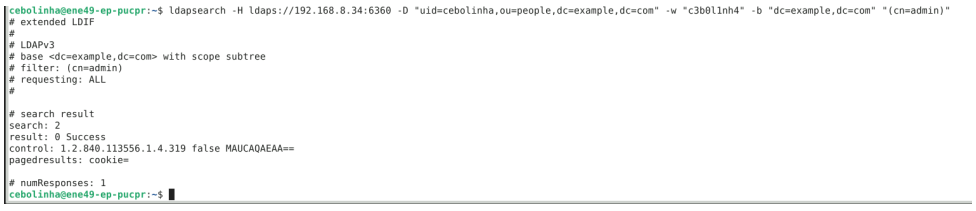

Figure 26: LDAP Query for admin using `ldapsearch`

Wazuh (XDR + SIEM)

Wazuh is a security platform that functions as a *SIEM* (Security Information and Event Management) and *XDR* (Extended Detection and Response) solution. It enables log collection, analysis, and correlation, intrusion detection, integrity monitoring, incident response, and more.

Wazuh Manager Installation

The **Wazuh Manager** installation was performed using the official script:

```
curl -sO https://packages.wazuh.com/4.11/wazuh-install.sh
chmod +x wazuh-install.sh
./wazuh-install.sh
```

Listing 31: Wazuh Manager Installation

Note: The `admin` user password is automatically generated during installation and displayed in the terminal.

Wazuh Agent Installation

On a client machine (such as in the DMZ or internal network), the **Wazuh Agent** was installed with the commands below, using the Wazuh Manager's IP:

```
wget https://packages.wazuh.com/4.x/apt/pool/main/w/wazuh-agent/
   wazuh-agent_4.11.2-1_amd64.deb
sudo WAZUH_MANAGER='192.168.8.34' WAZUH_AGENT_NAME='DMZ' dpkg -i
   ./wazuh-agent_4.11.2-1_amd64.deb
sudo systemctl daemon-reload
sudo systemctl enable wazuh-agent
sudo systemctl start wazuh-agent
```

Listing 32: Wazuh Agent Installation

Wazuh Agent Configuration

To ensure audit logs are monitored, the agent configuration file was edited:

```
<localfile>
  <log_format>audit</log_format>
  <location>/var/log/audit/audit.log</location>
</localfile>
```

Listing 33: Audit Log Monitoring Addition

Audit Rules for Privileged Commands

To ensure visibility into administrative privilege usage, rules were added to `auditd`:

```
-a exit,always -F arch=b64 -F euid=0 -S execve -k audit-wazuh
-a exit,always -F arch=b32 -F euid=0 -S execve -k audit-wazuh
```

Listing 34: Audit Rules in /etc/audit/rules.d/audit.rules

These rules ensure that all `execve()` system calls performed by users with effective ID 0 (`root`) are logged and processed by Wazuh.

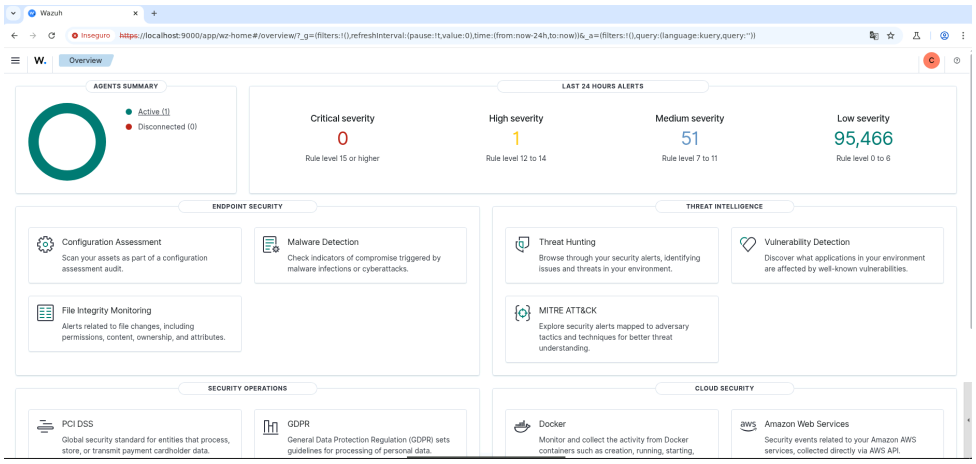

Figure 27: Wazuh Dashboard with Active Agent in DMZ

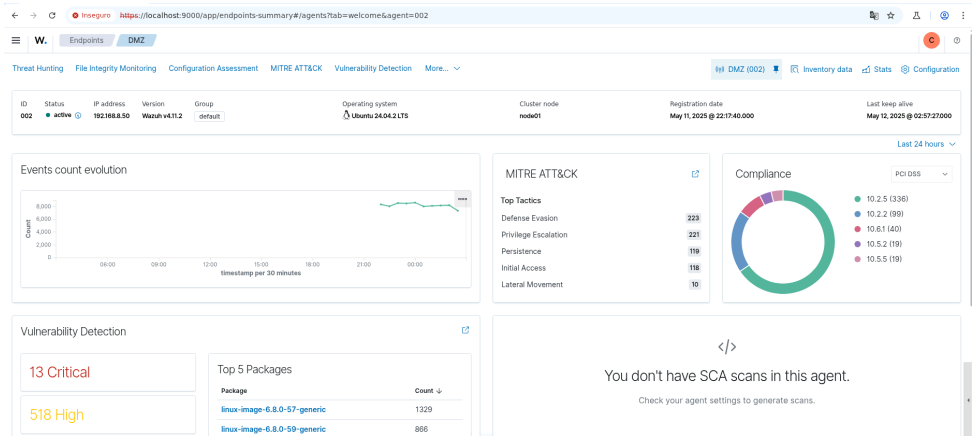

Figure 28: Wazuh Agent Interface Running on DMZ Machine

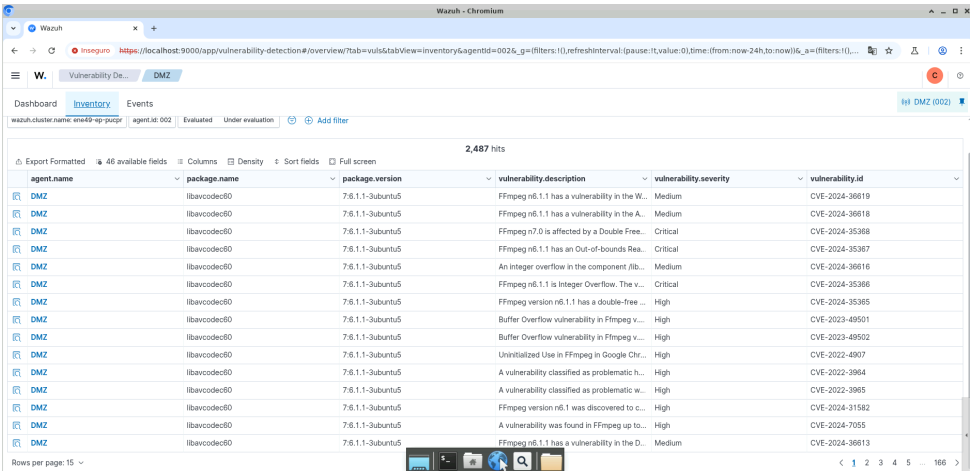

Figure 29: Vulnerability Inventory Detected

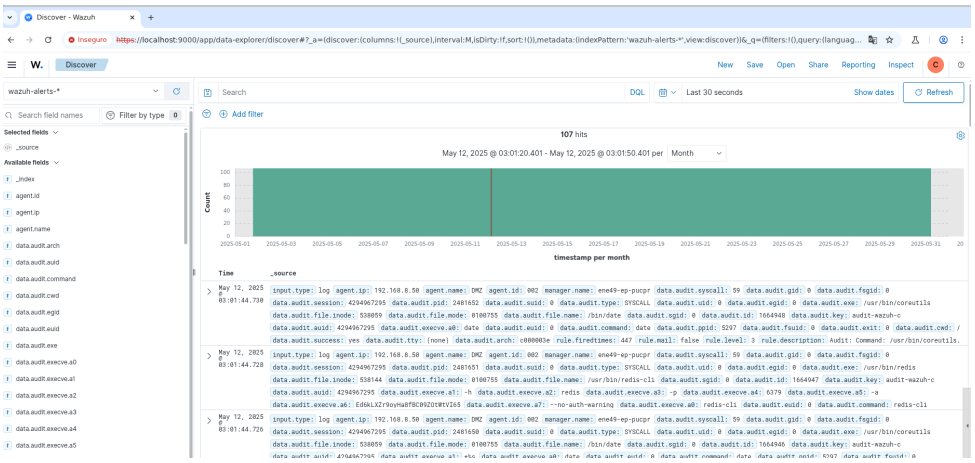

Figure 30: Wazuh Alert Center with Security Events

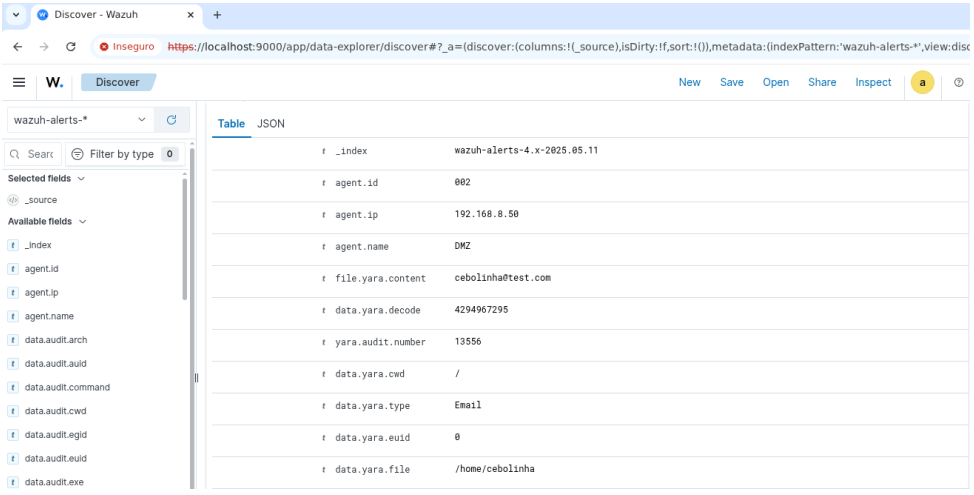

Figure 31: Event of Execution with Root Permissions Captured

YARA Integration with Wazuh (DLP)

To enhance **Data Loss Prevention (DLP)** capabilities, integration between **YARA** and the **Wazuh** agent was performed. YARA is a powerful tool for identifying patterns and sensitive content in files.

YARA Rules

Rules were defined in the `/opt/dlp/rules/dlp.yar` file. Below are some of the rules used:

```
rule Detect_PrivateKey_PEM {
  meta:
    description = "Detects private keys in PEM format"
  strings:
    $pem1 = "-----BEGIN PRIVATE KEY-----"
    $pem2 = "-----END PRIVATE KEY-----"
  condition:
    $pem1 and $pem2
}

rule Detect_Email {
  meta:
    description = "Detects email addresses"
  strings:
    $email = /[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}/
  condition:
    $email
}

rule Detect_PhoneNumber_BR {
  meta:
    description = "Detects Brazilian phone numbers"
  strings:
    $phone = /\(?\d{2}\)?\s?\d{4,5}-\d{4}/
  condition:
    $phone
}
```

Listing 35: YARA Rules for Sensitive Data Detection

Scanning Script and Scheduling

A `yara_scan.sh` script was created to scan defined directories, executing YARA rules and logging the output:

```
#!/bin/bash

echo "=== [$(date)] Scan Start ===" >> /var/log/yara-dlp.log
yara -r /opt/dlp/rules/dlp.yar /home/* >> /var/log/yara-dlp.log
   2>&1
```

```
echo "=== Scan finished ===" >> /var/log/yara-dlp.log
echo >> /var/log/yara-dlp.log
```

Listing 36: Excerpt from yara_scan.sh Script

This script is automatically executed every 4 hours via a scheduled `cron` task:

```
0 */4 * * * root /usr/local/bin/yara_scan.sh
```

Listing 37: Crontab Scheduling

Integration with Wazuh Agent

To enable Wazuh to monitor YARA detections, the agent was configured to track the log file:

```
<localfile>
  <log_format>full_command</log_format>
  <location>/var/log/yara-dlp.log</location>
</localfile>
```

Listing 38: YARA Log Monitoring in Wazuh Agent

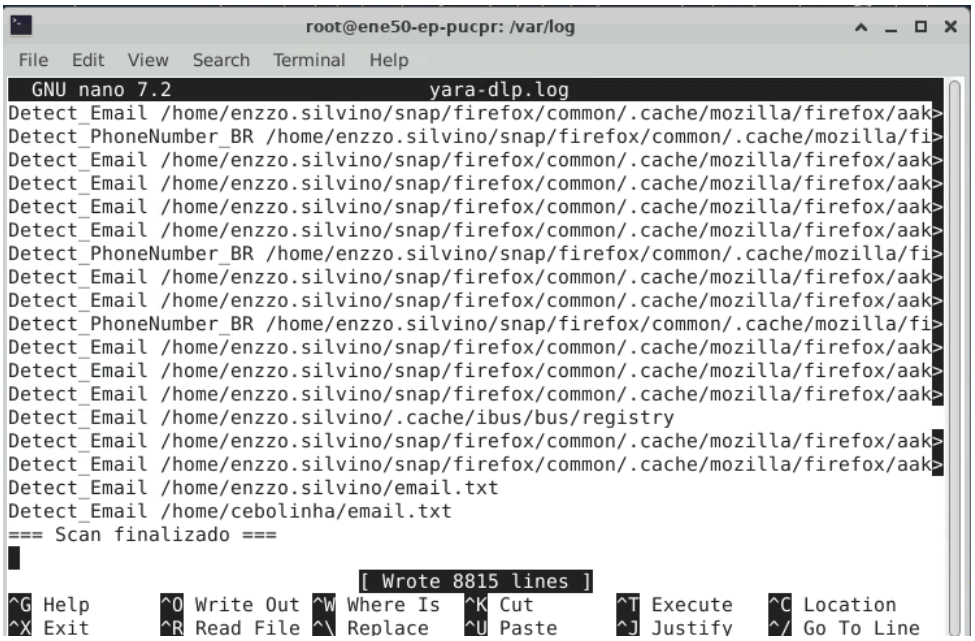

Figure 32: Log Indicating Email Presence in Test User's File

HashiCorp Vault (PAM)

To ensure secure credential delivery between systems, the **HashiCorp Vault** tool was used, implementing the **Privileged Access Management (PAM)** standard. The solution dynamically distributes MySQL database credentials from the internal network (192.168.8.34) to a PHP application hosted on Apache in the DMZ (192.168.8.50).

Workflow

- The **Vault** stores credentials in the `KV v2` engine.
- DMZ machine authentication is performed via `AppRole` with the **Vault Agent**.
- The agent dynamically generates the `/etc/app/db.env` file with username and password.
- Apache is automatically reloaded when the template is updated.
- Communication between the DMZ and Vault is protected by TLS with a self-signed certificate.

Vault Configuration (host 192.168.8.34)

File `vault.hcl`:

```
ui = true
storage "file" {
  path = "/opt/vault/data"
}
listener "tcp" {
  address = "0.0.0.0:8200"
  tls_cert_file = "/opt/vault/tls/tls.crt"
  tls_key_file = "/opt/vault/tls/tls.key"
  tls_disable = false
}
api_addr = "https://192.168.8.34:8200"
```

Listing 39: vault.hcl

- Certificate with SAN: `vault.interno.local` + IP.
- Initialization: `vault operator init -key-shares=5 -key-threshold=3`.

Enable KV v2 engine:

```
vault secrets enable -path=secret kv-v2
vault kv put secret/db-credentials username="remote_user"
   password="######"
```

db-read.hcl Policy:

```
path "secret/data/db-credentials" {
  capabilities = ["read"]
}
```

```
vault policy write app-db-read db-read.hcl
```

AppRole Configuration:

```
vault auth enable approle
vault write auth/approle/role/app-role \
  token_policies="app-db-read" \
  token_ttl=1h token_max_ttl=4h \
  token_bound_cidrs="192.168.8.50/32" \
  secret_id_num_uses=0 secret_id_ttl=0

vault read -field=role_id auth/approle/role/app-role/role-id > /etc/vault/role_id
vault write -field=secret_id -f auth/approle/role/app-role/secret-id > /etc/vault/secret_id
```

Vault Agent Configuration (host 192.168.8.50)

Environment file template: `/etc/vault/db-template.tpl`

```
DB_USER={{ with secret "secret/data/db-credentials" }}{{ .Data.data.username }}{{ end }}
DB_PASS={{ with secret "secret/data/db-credentials" }}{{ .Data.data.password }}{{ end }}
```

Agent Configuration: `/etc/vault/agent.hcl`

```
vault {
  address = "https://192.168.8.34:8200"
  tls_skip_verify = false
}

auto_auth {
  method "approle" {
    mount_path = "approle"
    config = {
      role_id_file_path = "/etc/vault/role_id"
      secret_id_file_path = "/etc/vault/secret_id"
    }
  }
  sink "file" {
    config = {
      path = "/etc/vault/.vault-token"
    }
  }
}

template {
  source      = "/etc/vault/db-template.tpl"
  destination = "/etc/app/db.env"
  perms       = "0640"
```

```
  command     = "systemctl reload apache2"
}
```

Systemd Service: `vault-agent.service`

```
[Unit]
Description=Vault Agent - AppRole + template
After=network-online.target

[Service]
ExecStart=/usr/bin/vault agent -config=/etc/vault/agent.hcl
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
```

Apache/PHP Integration

The PHP code directly reads the file:

```
$creds = parse_ini_file('/etc/app/db.env');
$mysqli = new mysqli('192.168.8.34', $creds['DB_USER'], $creds['
    DB_PASS'], 'fortune_quotes');
```

Permissions:

- `/etc/app`: 750
- `db.env`: 640
- Group: `www-data`

Figure 33: Password Update via `vault kv patch`

Figure 34: Vault Agent Logs Indicating Successful Rendering

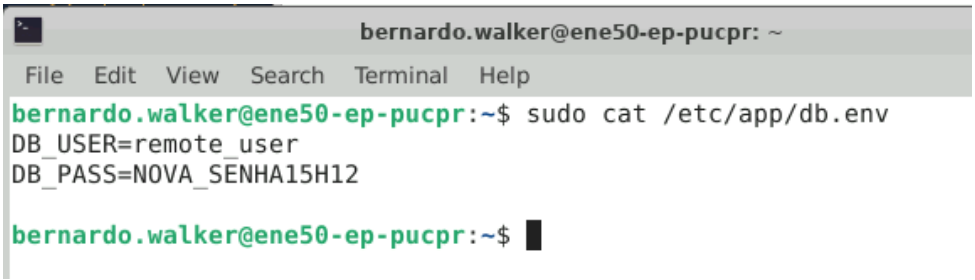

Figure 35: Generated Content in `db.env` File

```php
/**
 * Connect to the database
 *
 * @return mysqli Database connection
 */
function connectDB() {
    $host = '192.168.8.34:3307';
    $database = 'fortune_quotes';

    $envFile = '/etc/app/db.env';
    if (is_readable($envFile)) {
            $creds = parse_ini_file($envFile);
            if ($creds !== false && isset($creds['DB_USER'], $creds['DB_PASS'])) {
                    $username = $creds['DB_USER'];
                    $password = $creds['DB_PASS'];
            }
    }
    error_log("connectDB: using DB_USER={$username} DB_PASS={$password}");


    $db = new mysqli($host, $username, $password, $database);

    if ($db->connect_error) {
        die('Database connection failed: ' . $db->connect_error);
    }

    return $db;
}
```

Figure 36: Web Application Code Accessing Database

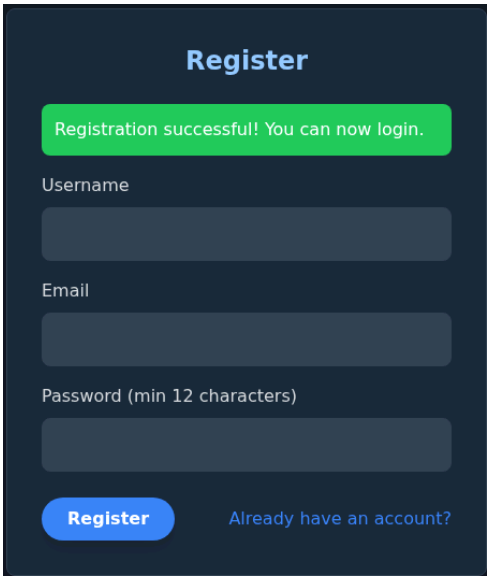

Figure 37: Web Application Registering Normally with Dynamic Credentials

Firewall Rules

Traffic control between network zones was achieved through restrictive firewall rules, ensuring only necessary flows are allowed, thus minimizing the attack surface. The rules were applied to routers/firewalls separating the zones: **Internal Network**, **DMZ**, and **External**.

Blocking Internal Network to External Machine

To prevent data theft and unauthorized access from within the network to the external world, the Internal Network was blocked from initiating any direct connections to the external machine.

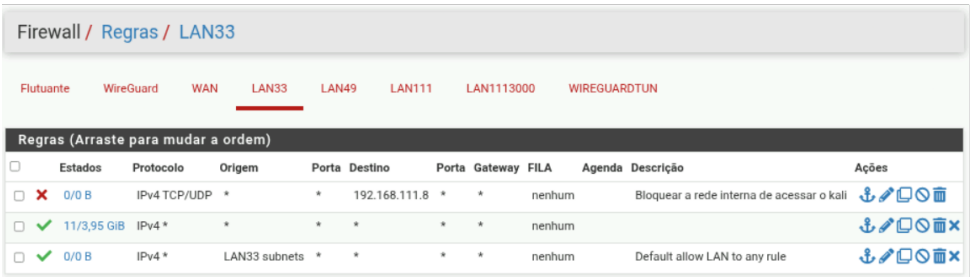

Figure 38: Blocking All Ports from Internal Network to External Machine

DMZ to Internal Network Access Rules

Communication from the DMZ to the Internal Network is strictly controlled. Only pre-authorized services are allowed, such as MySQL database access (port 3306) and Vault authentication (port 8200).

External Machine Restrictions

The External Machine is limited to accessing only public services exposed in the DMZ, such as:

- Web Application (`HTTP/HTTPS`) – Ports 80/443
- Email Server (SMTP/IMAP/Submission) – Ports 25, 587, 993

Direct connection attempts to the Internal Network are completely blocked. Administrative access is performed in two steps:

1. SSH from the external machine to the DMZ (port 22)
2. SSH from the DMZ to the Internal Network (port 2222)

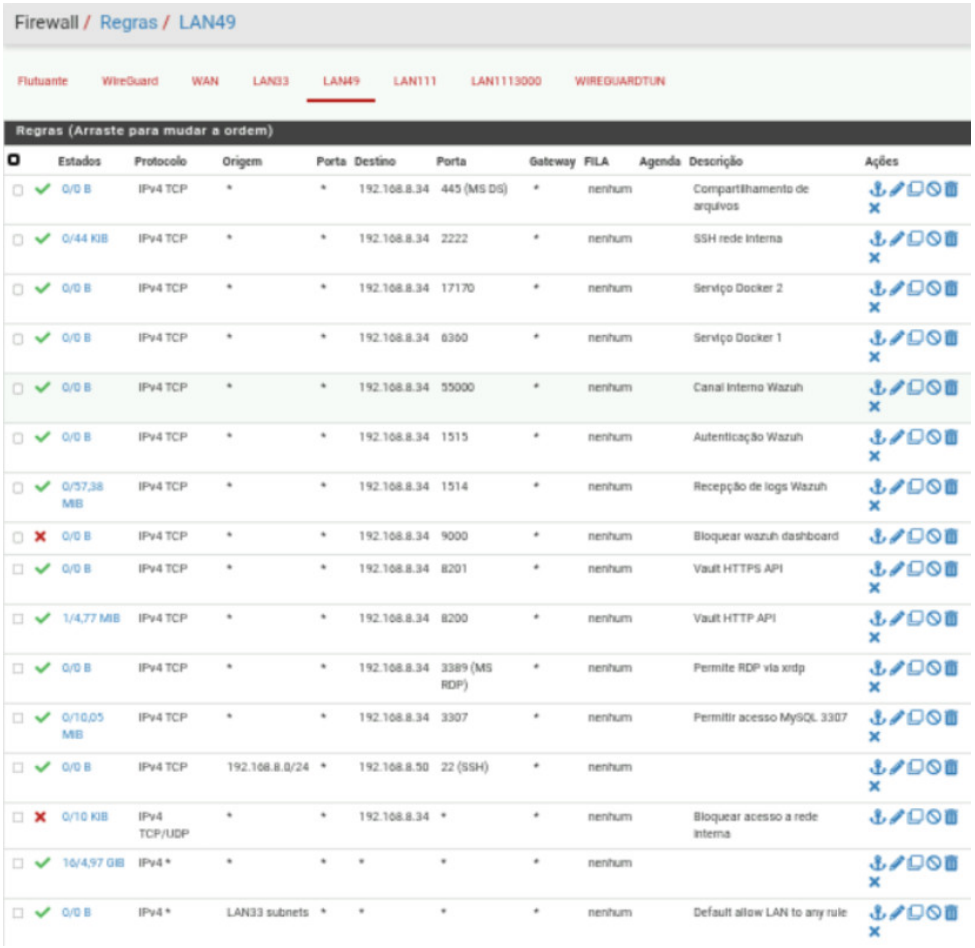

Figure 39: DMZ to Internal Network Rules: Only Authorized Connections Allowed

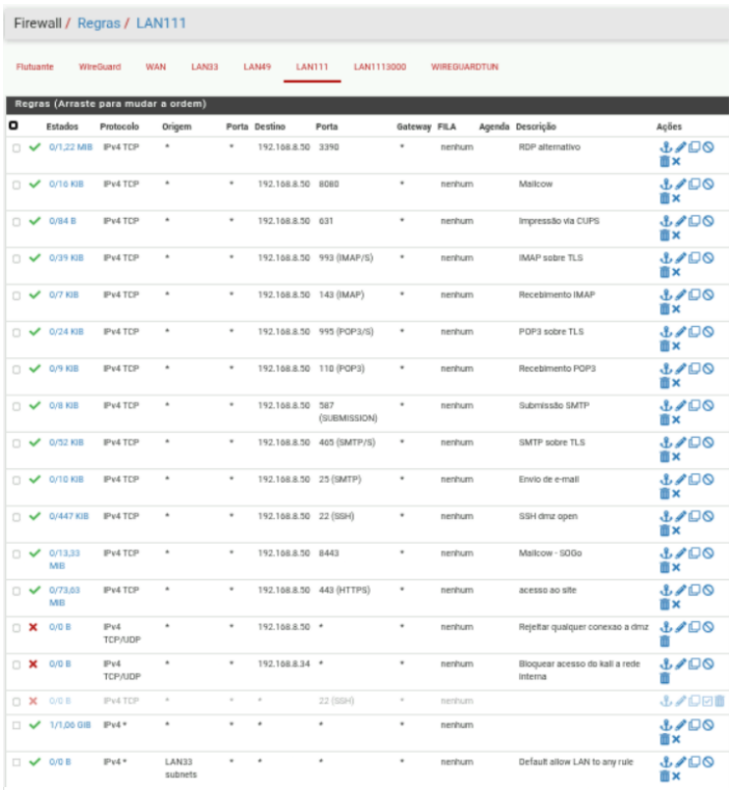

Figure 40: External Machine Only Accesses Public DMZ Services; Internal Access Requires SSH Proxy