Review of Group 9 by Group 5

Alexandre Santangelo Kaourintin Tamine
Moritz Teichner Yu-Hsin Yang

December 21, 2023

Contents

1 Background

Developers of the external system: *Kamila Babayeva, Marin Cornuot, Niels Lachat, Lin Zhang*

Date of the review: December 20, 2023

2 Report and Design Review

This part of our review analyzes the documentation provided to us by group 9. We review the system architecture and security concepts as well as their risk analysis.

2.1 System Architecture and Security Concepts

The first part of this documentation analysis deals with the general system architecture and the security concepts employed to make the system more resilient to attacks. For both aspects, we list positive and negative aspects.

2.1.1 System Architecture

Positive

- **Compartmentalization**: All specific services required for the functionality of the system are separately deployed on different machines. Furthermore, the use of a DMZ for the web server and the separation of internal and external network segments demonstrate a sound approach to compartmentalization. This reduces the attack surface and mitigates risks associated with external threats. Additionally, employing a dedicated firewall machine between the parts of the system is a good idea to fully monitor and restrict access.

Negative

- **Single Point of Failure (Firewall)**: The firewall server is the only point of entry into the system. This is of course beneficial when it comes to access control as every request can be evaluated and blocked here. But a possible problem is that when a Denial of Service attack occurs and the firewall is not responding to legitimate requests, system and ca admins can not access the system either. They would in this case have to connect to the machines physically.

2.1.2 Security Design

Positive

- **Overall**:

  - Internal Communication: Employing TLS for secure, internal communication between the different machines ensures confidentiality, integrity, and authentication. This is crucial for a system dealing with sensitive information like certificates and user data.
  - Access Control Mechanisms: The implementation of access controls and authentication mechanisms, such as TLS client authentication and SSH key-based authentication, aligns with security best practices. This contributes to a robust defense against unauthorized access.
  - Logging and Traceability: Real-time logging, secure communication (TLS) for log transmission, and regular backups contribute to traceability and adherence to security principles. This ensures that events and actions are recorded, aiding in incident response and forensics.
  - Security Principles Adherence: The security design consistently adheres to established security principles, including least privilege, defense in depth, and fail-safe defaults. This provides a solid foundation for building a secure system.

- **Web Server**:
  - Session Management: Utilizes Flask for managing user sessions securely through cryptographically signed session cookies.
  - Secure against Several Attacks: Use strict transport security (HSTS) to protect against Man-In-The-Middle attacks. Use prepared statements to mitigate the risk of SQL injection attacks. Use flask form to protect against CSRF vulnerabilities.

- **CA Server**:
  - TLS Authentication: The use of TLS for server authentication and secure communication with the web server strengthens the CA server's security.
  - Access Controls: Limited access via sudo and the requirement for X509 certificates and passwords contribute to the principle of least privilege.

- **MySQL Database**:
  - Isolation of Root Database User: The root database user is only enabled on localhost, accessible solely to the system admin after a successful SSH connection. This follows the principle of *least privilege*.
  - Access Controls: Differentiated access levels for webserver and ca-server users follow the principle of least privilege.

- **Backup Server**:

  - Secure Backups: Regular backups, encryption of private keys, and offline storage of backup drives contribute to data availability and confidentiality.
  - Backup Strategy: A client-agent backup approach is adopted, reducing the risk of a single point of failure. Critical data, including keys, certificates, databases, configurations, and logs, are backed up. Backup drives are rotated and migrated offline, providing redundancy and adhering to the principle of data recovery. Storing backups in a physically secure vault adds an extra layer of protection.
  - Syslog-ng for Log Management: Syslog-ng is used for streaming logs, which enhances traceability and monitoring. Centralized logging is a good security practice for aggregating logs from different components.
  - Dedicated user accounts for each machine used to restrict access to the backups of other machines.
  - Root Private Key Protection: Keeping the root private key offline adds an extra layer of protection, preventing potential remote attacks. Physical access is required for any operation involving the root private key.

- **Firewall**:
  - Compartmentalization: Enforcing separation between the DMZ and intranet aligns with the principle of compartmentalization.
  - Port Forwarding Configuration: The firewall implements port forwarding rules to direct specific incoming traffic to designated internal machines, such as the web server, CA server, MySQL server, and others.
  - Fail2Ban Integration: The firewall utilizes Fail2Ban for monitoring logs and automatically blocking IP addresses exhibiting malicious behavior, which is useful for mitigating brute force attacks.

- **System Admin Access**:
  - A System Administrator uses a private key to connect to the machines via SSH. To further increase security, the admin needs to input a machine specific password to gain root access. This is a form of Multi Factor Authentication and it's definitely reasonable.

Negative

- **Web Server**:
  - Data Storage During Certificate Issuance: Temporary storage introduces a potential risk, and the duration and security measures during this temporary storage are not explicitly mentioned.

  - HTTP to HTTPS Redirection: While the redirection from HTTP to HTTPS is implemented, the fact that port 80 is still open might expose the system to potential attacks.

- **CA Server**:
  - Single Point of Failure: The CA server being a single point of failure poses a potential risk. Design considerations for redundancy and failover are crucial for continuous CA functionality.
  - Limited Information on Key Management: In their report, there is a lack of information on how the CA server securely stores its private keys, how to distribute CA server's public keys, and how to protect its root private key. This raises concerns about the clarity and robustness of key-related practices.
  - Lack of Intermediate CA: In their report, it seems that there's only one root CA for issuing the client certificates. Yet, from our point of view, one should use an Intermediate CA (ICA) that's issued by the root CA to do this job as if the CA used for issuing certificates is compromised, it's less work to revoke the old ICA than to deploy a completely new root CA.

- **MySQL Database**:
  - Plain Text Storage of Database: Storing the MySQL database in plain text, even with hashed passwords, might expose the data to potential risks. Encrypting the entire database or employing database-level encryption could provide an additional layer of protection. (This is not a critic on the team as we were instructed not to encrypt the database due to storage restrictions.)

- **Firewall**:
  - Single Point of Failure for SSH Access: The firewall allows SSH access to all internal machines using a single set of credentials (public key). While the use of public key authentication enhances security, having a single set of credentials for accessing multiple internal machines poses a risk. If these credentials are compromised, an attacker gains widespread access, contradicting the principle of least privilege.

2.2 Risk Analysis

In this section, we will discuss the risk analysis of group 9 step by step, including the relevant assets, threat sources, risk definitions, and countermeasures.

2.2.1 Assets

- **Physical Assets**: The list is almost comprehensive, detailing physical protection measures. Yet, some important security properties of the assets

are missing. For example, Confidentiality and Integrity are missing for the Internet Connectivity. For instance, allowing an attacker to observe all the traffic would possibly allow tracking of employees via ip addresses.

- **Logical Assets - Software**: The list is comprehensive.
- **Logical Assets - Information**: Focusing mostly on the keys in the system. Yet, failed to mention several kinds of data, for example, Backups and Logs, the certificate list and the CA state, which are an essential part of the Information asset.
- **Persons**: All the involved parties are listed and therefore we confirm the completeness.
- **Intangible good**: This part is missing from the report. This asset is important since it affects the organization's operations, reputation, and competitive position.

2.2.2 Threat sources

- The list of threat sources is quite complete. Yet, adding threats specific to the company iMovie can be an improvement. For instance, parties targeted by investigative journalism could be a very specific threat source.

2.2.3 Risk definitions

- The likelihood and impact descriptions are well-argued, and the resulting risk table aligns coherently.

2.2.4 Risk Evaluation - Physical Assets

- **Key safe**: Well-stated identification of threats and countermeasures.
- **Backup storage**: Well-stated identification of threats and countermeasures
- **Backup server**: The threats described are sensible and the countermeasures are simple but effective.
- **Internet access**: For Threat 5, the countermeasure mentions no specific action. A more detailed countermeasure or alternative plan could be outlined.

2.2.5 Risk Evaluation - Logical Assets

- **Web server software**: Comprehensive identification of potential threats, including script kiddie attacks, zero-day vulnerabilities, brute-force attempts, exploitation of login pages, and phishing attacks. The countermeasures are also robust.

- **CA software**: The threat analysis is specific, focusing on a DoS attack related to continuous certificate generation and revocation. They can consider expanding the threat analysis to include other threats specific to CA server software, such as certificate compromise or unauthorized access.
- **Backup server software**: Identification of a potential threat related to a script kiddie attempting numerous requests. Similar to CA software, they should consider expanding the threat analysis to include other potential threats specific to backup server software, such as unauthorized access or data tampering.
- **Firewall/Router software**: Comprehensive identification of potential threats and countermeasures are correct and seem sufficient to protect the system.
- **Employees' data**: Comprehensive identification of potential threats and countermeasures are correct and seem sufficient to protect the system.
- **CA private key**: The threat analysis focuses on unauthorized access to the CA private key by skilled adversaries. Similar to CA software, they should expand the threat analysis to include other potential threats specific to CA private keys, such as key theft, loss, or compromise during key generation.

2.2.6 Risk Evaluation - Persons and Intangible Goods

- **Persons**: The threats only focus on one specific threat (phishing attack), which is too specific. They should expand the threat analysis to include other potential threats to employees, such as physical security breaches, insider threats, or social engineering attacks beyond phishing.
- **Intangible Goods**: Lack of risk analysis on intangible goods

2.2.7 Risk Acceptance

- All high-risk threats are identified, and proposed countermeasures seem sensible and effective.

3 Implementation Review

3.1 Compliance with Requirements

In this section, we will discuss whether the system meets the functional and security requirements given in the assignment and whether they are implemented correctly.

The biggest issue we faced while examining the system was the setup process. As the system is not stable we need to reboot different machines several times to get everything working. Yet, in the end, we managed to find a workaround

with the help of Group 9. The only problem remaining is that each request requires quite some time (5 to 10 seconds) to receive the response. In addition to that, we were not able to properly SSH into the backup machine and had to use the VM interface to look at the machine directly.

The responses from the VMs are sluggish, meaning something like taking a request, replaying it in burpsuite and modifying it multiple times takes minutes instead of seconds.

3.1.1 Certificate Issuing Process

- The system keeps a database as required.
- The user can log in via a web form by entering his credentials (ID and password). The credential is verified by consulting the information stored in the database.
- The user's information stored in the database is shown.
- The user can correct the information and the changes can be applied to the database.
- The user's certificate can be issued based on the user's information.
- The user can download the new certificate, including the corresponding private key, in PKCS12 format.

3.1.2 Certificate Revoking Process

- The user can revoke his certificate.
- The new certificate revocation list can be generated and published on the web server. Login with revoked certificates is not possible.

3.1.3 CA Administrative Interface

- The CA administrator can log in with a CA administrator certificate. Yet, this function won't work if you've previously logged in as a normal user. You need to relaunch the web page to log in as a CA administrator.
- The CA administrator interface can show the number of issued certificates, the number of revoked certificates, and the current serial number.

3.1.4 System Administrative and Maintenance

- The system administration of this system is implemented on the user-GUI machine. This user can ssh to all the machines in the system
- An automated backup solution is implemented on the backup server.

3.2 System Security Testing

We divide the system security testing into two parts. First, we look at the system from the outside, thinking of it as a black box. Second, we take a deep dive into the implementations of security measures directly on the machines, the white box test.

3.2.1 Blackbox Testing

- **Information Gathering**: We employed different tools to gather information about the system in review. These tools are: Nmap, Telnet and Netcat. Using these, we were able to gather information about the running services on the internet-facing machine, the firewall.

Port	Status	Service	Version
22	open	ssh	OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
443	open	https	Apache httpd 2.4.57 (Debian)
2002	open	globe	OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
2003	open	finger	OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
2004	filtered	mailbox	OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
2005	open	deslogin	OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)

Table 1: Port information

  When trying to connect to the different services, we only managed to get reasonable output from the HTTP server. The other SSH services require an ssh key to connect to them. We suspect that these ports are used to connect to the other machines, e.g. for maintenance purposes. The white box testing revealed that our suspicion was correct. We also identified that the database runs MariaDB.

- **Vulnerabilities Scanning**: To further assess if the system is vulnerable to attacks, we decided to employ automatic tools for vulnerability detection. The tools are: OpenVAS, Skipfish, Wapiti and Nikto. When scanning the system with Skipfish, we discovered that the system indeed deploys DDoS detection as the tool reported that the responses to certain queries vary. The output was a "Too many requests" error stating that only 10 per minute are allowed.
  A Wapitit scan claimed to have found a vulnerability. The tool reports that the Content Security Policy (CSP) is not set. CSP is an extra layer of security to prevent some types of attacks, including cross-site scripting (XSS). While we were not able to exploit this vulnerability, we still recommend adding a CSP header for additional security.

- **Manual Testing**: In addition to automated testing tools, we manually explore the website, familiarise ourselves with the system and manually

search for vulnerabilities. Furthermore, we employ a crawler, specifically burpsuite, where we supply usernames and credentials for enhanced testing.

According to their documentation, they specify a limit of "100 requests per hour" before the denial-of-service protection blocks the endpoint. However, in practice, we have managed to make over 600 requests before encountering a block. Although the imposed limitation remains, it appears that the stated specifications are inaccurate and do not align with the observed behavior.

We intercepted and manipulated the POST requests responsible for login, receiving a certificate, admin login, etc.. The first interesting result is that there seems to be a comment on the main HTML page that might serves as a hint for one of the implemented back doors.

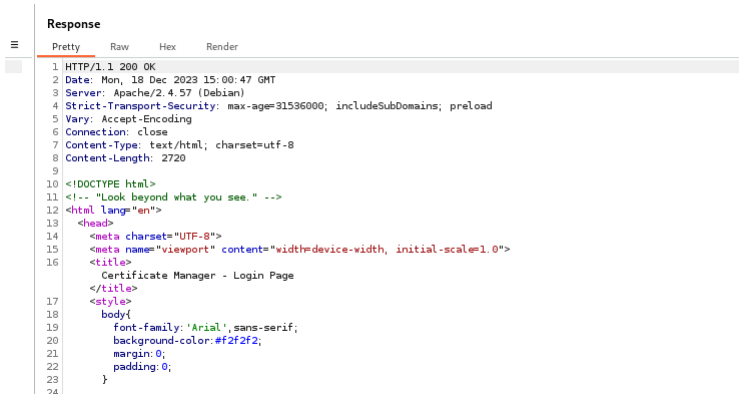

Figure 1: Main HTML page including a hint for the backdoor

Additionally, we tried using the burpsuite repeater and manually submitted classic SQL requests to trigger a crash. Just like the automated tools reported, the server seems safe against SQL injections on the login parameters.

Furthermore, we inspected the different requests made to the web server in more detail. The requests seem to be in order. For instance, the login request is a classic post request containing username and password. Getting access using brute force is limited due to the rate limiting described above. The cookie does not look identifiable in any way. It is a very long hash and we have not succeeded in determining any similarities or indications of how the cookies are constructed. User input seems to be sanitized as an input of the character " results in an error.

During this part of the testing, we were able to find one possible security risk. When we logged out of the account, we noticed that we were still able to make requests, as if we were logged in. The system even logged

us in again automatically, indicating that the cookie was not invalidated after logging out. In a scenario where a user is on a public computer, other people could make use of this flaw to get access to the user's account. As in the current setup you can freely change passwords, the user could lose his account and his certificate. Thus, we identify this as a security finding of HIGH importance and suggest mitigating it by invalidating user cookies upon logout.

3.2.2 Whitebox Testing

As part of the Whitebox Testing, we analyzed the running services, file access control and the code. There are three codebases made available to us. The **webserver**, the **ca-server backend** and the **log server**.

- **Running Services:** We inspected the running services on the machines to determine if there are vulnerabilities that could be exploited. While we were not able to find vulnerable services running, we noticed that the team installed the service "failed2ban" on the firewall machine which is used to protect log files and prevent brute force attacks. This is a good idea to enhance security on the machine that is most vulnerable as every connection is handled by the firewall before handing them to the other machines.
- **File Access Control:** Inspecting the files and the permissions using the simple command "ls -l" we were not able to find any vulnerabilities in the system. Every file that seems to be important for the security of the system is protected by only allowing the owner of the file to access it.
- **Webserver and Front-end:** The SQL server query variables are currently exposed in plaintext within the code. Consider securing them by storing the sensitive information in a .env file. This recommendation applies universally across source code files, as they inadvertently disclose CA addresses. This disclosure could potentially expand the attack surface.

```
MYSQL_HOST = "10.0.0.5"
MYSQL_PORT = 3306
MYSQL_USER = "webserver"
MYSQL_PASSWORD = "}DqG3mZ8neKPp?#Uc?49K&W2"
MYSQL_DATABASE = "imovies"
MYSQL_CLIENT_CERT_PATH="/etc/ssl/certs/webserver-intranet.crt"
MYSQL_CLIENT_KEY_PATH="/etc/ssl/private/webserver-intranet.key"
CA_CERT_PATH="/etc/ssl/certs/cacert.pem"
```

Figure 2: Some variables that might be better in a .env

Now, we check again for SQL injections.

```python
def db_auth(user_id, password, logger):
    try:
        conn = MySQLConnection(user=MYSQL_USER, password=MYSQL_PASSWORD,
                               host=MYSQL_HOST, port=MYSQL_PORT, database=MYSQL_DATABASE,
                               ssl_ca=CA_CERT_PATH, ssl_cert=MYSQL_CLIENT_CERT_PATH,
                               ssl_key=MYSQL_CLIENT_KEY_PATH,
                               ssl_verify_cert=True)
    except mysql.connector.Error as err:
        logger.error("Connection to DB for authentification failed: {}".format(err))
        return None
```

Figure 3: Python code to authenticate to the DB

Verification through online forums and documentation affirms that the password and user fields will be treated strictly as string literals. Therefore, as observed in the blackbox testing, it appears unlikely to exploit SQL injection in this context. Additionally, the code for the front-end server follows conventional patterns and does not exhibit any apparent vulnerabilities.

- **CA server:** As the CA server backups are the most important ones, we wanted to inspect the scripts responsible for that in more detail. Looking at the backup scripts and their authorizations:

```
-r-x------ 1 root root 633 Nov 29 17:05 backup_ca_config.sh
-rw-r--r-- 1 root root 361 Nov 29 17:05 cron_setup_backup.sh
```

config.sh: root has read and exec permissions, group and others have no permissions. This seems appropriate for a backup script. For the cron setup we could consider restricting to the same auth as config.sh.

In config.sh, we noticed that the option **StrictHostKeyChecking=no** is not enabled during the SFTP connection. Online documentation suggests that this could lead to possible man in the middle attacks so it could be enabled.

The source code for the CA is mainly contained in the file ca_server.py and ca_database.py. We do not see any main flaws in both of these implementations as they seem to follow the classic setup without deviations.

- **Backup server:** Every other machine has a specific user account on this machine. In that way, the team is able to restrict access to the logs to the specific machine responsible for them.

It appears that the SSH connection with the VM experiences occasional disruptions, possibly due to interactions with other entities. To circumvent this issue, we opted to use the terminal GUI for file access, albeit with more limited interactions. We conducted a thorough examination of the log content to ensure its alignment with the provided information.

3.3 Backdoors

3.3.1 Easy Backdoor

The first hint that leads us to the easy backdoor was the comment on the main html page saying "Look beyond what you see.". In the beginning, we tried looking at the html files in more detail but were not able to find anything useful. The essential idea was to inspect the logo the other team provided. We started to look at the picture in more detail, changed the colors and contrast and in the end inspected the meta data. There we found an unexpected Base64 encoded string in the comment data:

```
"QmFja3VwIG1hY2hpbmUgY3JlZHM6IGRlYnVnIC8gQ29uZ3JhdHVsYXRpb25zIVkw"
```

Decoding this string leads us to the following message that showed that we were successful:

```
"Backup machine creds: debug / Congratulations!Y0uF0undTh3Ea5y8ackd0or:+1:"
```

3.3.2 Hard Backdoor

Unfortunately, regardless of the effort we put in finding the other backdoor, we were not able to find it. The team really did a great job hiding it.

4 Comparison

4.1 Similarities of the Two Systems

Apart from the functional requirements, both systems share the following characteristics.

- **Similarity 1:** We employed the same compartmentalization of tasks to machines.
- **Similarity 2:** We both had one machine as the single point of entry into the system.
- **Similarity 3:** Both systems only implement a root CA responsible for all certificates issued.

4.2 Highlights of Group 9's System

- **Highlight 1:** The logs are very complete, they include elements of the VM startups, elements of the VM parameters, authentications, logs, kernel logs etc. This might be helpful in the case of a system issue or security leak for the incident response team.
- **Highlight 2:** Additionally running "failed2ban" on the internet facing machine is an important security measure.
- **Highlight 3:** As system admins not only need an SSh key but also a machine-specific password to access the machines and perform actions with root privileges, implement Multi Factor Authentication.
- **Highlight 4:** The team implemented advanced DoS protection which is crucial to keep the system available.

4.3 Highlights of Our System

- **Highlight 1:** Our system works out of the box with no issues, it is very fast. The initial setup is easy and streamlined, making it easy both for users and testers to have a running system.
- **Highlight 2:** Our risk assessment is very complete. We have carefully considered the use of HSM's, the locations of our servers, the risk formations that have to be given to our employees as well as the classic security risk to our applications. Mentioning pentesting and bug bounties on our application is also a nice addition.
- **Highlight 3:** Our use of ansible to spool up the VMs allows us to easily scale up our systems if necessary.
- **Highlight 4:** Use Sveltekit for building front-end server, which provides protections for common web vulnerabilities.