Review of Group 6 by Group 5

Mei-Chih Chang Onur Veyisoglu
Sergio Roldán Lombardía Jason Friedman

...

Page limit: 18 pages.

Contents

Recall the following guidelines when writing your reports:

- *Adhere to the given templates.*
- *Refer to the security principles in the book for justification.*
- *Use clear terminology:*
  - *secure = confidential + authentic. Be clear about which properties you are writing.*
  - *Are pairwise distinct: certificate, private key, public key, archive to of certificate with private key. Please avoid mixing these up.*
- *Use a spell checker before hand-in!*

1 Background

Developers of the external system: *Markus Ding, Eric Jollès, Simon Perriard, Hédi Sassi*

Date of the review: 09.12.20

2 Report and Design Review

In this section we review the design of the system as displayed in the report of group 6.

2.1 System Architecture and Security Concepts

In the following, we assess if the chosen architecture and the security design decisions are well motivated and justified. In each of these categories we point out both positive and negative aspects.

2.1.1 Architecture

- **Components:** The setup of the system architecture adheres to the *compartmentalization* principle by using a separate machine for every different system component. Separating the Webserver, Core CA, Database, Backup and Firewall makes sure there is *no single point of failure* and increases the complexity for an attacker substantially.
- **Network Architecture:** The network is separated into Internet, Demilitarized Zone (DMZ) and the trusted CA's Private Network. This again limits the attack surface and the harm potential.

2.1.2 Security Design

- **Webserver:** The webserver is designed as a flask application served to the client directly using the development server (Werkzeug Server). This is considered bad practice because the Werkzeug server was built for the development environment only and lacks stability, scalability and security features required in production [1]. A dedicated webserver such as NGINX or Apache should be used in combination with a Webserver Gateway Interface (WSGI) to serve a python application when running publicly.

  The report did not explicitly mention any protection methodologies on client-side web application attacks such as XSS and CSRF protection. The flask web application configures Jinja2 to automatically escape all values unless explicitly told otherwise. This rules out most of the XSS problems caused in templates. For CSRF protection, it would require a one time token on every web request, this security measurement is not taken.

- **Core CA:** The overall design of the certificate authority looks very reasonable. CA resources that may be accessed from multiple users simultaneously, such as the certificate revocation list, `REVOKED_COUNTER` and `ISSUED_COUNTER` are even protected from race conditions by acquiring locks. However, the Root CA should not directly issue client certificates, for that one should use an Intermediate CA (ICA) issued by the Root CA. The reasoning behind this, is that simply revoking the old ICA and deploying a new one after compromise is much less work than deploying an entire new Root CA.

  A very good feature is that user private keys are only stored encrypted. However, we have some concerns with using symmetric keys to encrpyt the user private keys. Encrypting sensitive data with a symmetric key usually just shifts the problem to keeping the symmetric key secret. Although the symmetric encryption key in this case is stored on a different machine (not on the backup server), there is still some room for improvement e.g. by combining with public cryptography, then the private key could even be stored offline.

  Issued user certificates and private keys are wrapped inside a PKCS#12 file before they are sent to the client. The PKCS#12 standard would allow passphrase protection, but the CA server does not use this feature in the current system. Enabling this feature could additionally protect the users private keys, especially because it would make sure, that they are not stored in plaintext on an untrusted client machine.

- **Database:** Instead of letting the webserver access the remote database directly, it can access it through a limited REST API, that only allows specific SQL queries that are actually required for the system's functionality. This decreases the attack surface and follows the *Minimum Exposure*

[1] https://flask.palletsprojects.com/en/1.1.x/tutorial/deploy/

principal. Furthermore, all SQL queries are implemented as prepared statements to prevent an attacker from performing SQL-injection attacks.

- **Firewall:** There is no specification of host based rules. Even if the firewall is properly configured and strict enough, we would encourage the reviewed team to include a second layer of protection by restricting the incoming connections to each machine separately.

- **Session Management:** Session management is handled through session cookies with support from the flask builtin session handler. However, the `SECRET_KEY` used for the creation of session tokens is set to a very weak default key ('dev'). This key can be brute-forced by an attacker to forge valid session tokens.

- **Backup:** The backup process includes mutual authentication between the Backup Server and the rest of the machines. The backup solution is a custom server instead of rsyslog, both to allow on-fly encryption of the backed-up files and to avoid vulnerabilities of rsyslog. The encryption algorithm used is AES-256 which is strong enough, and the additional physical measures to protect the symmetric keys are well thought. However, the backup process is only triggered when a certificate is issued or revoked. We think that this decision could lead to potential problems in the system, as multiple consecutive issue and revoke calls would cause multiple expensive backup processes. This could lead to a DoS attack to the backup server. Additionally, other modifications like a change in the user values won't trigger a backup, which seems quite arbitrary. We would encourage the reviewed team to schedule periodic back-ups instead.

- **System Administration and Maintenance:** The administration process using ssh is simple, secure and effective. However, detailing the physical protection of the private keys and its hand-in could have been interesting.

- **Certificate authentication:** The user authentication provided for the valid certificates is not delegated to the browser. Instead, the client and the server engage in a challenge-response protocol to prove validity of the certificate. Implementing crypto and cryptographic protocols is difficult, in a sense that any detail changed from specification to implementation could be a potential vulnerable spot [2]. With that we would encourage the reviewed team to switch to delegate this interaction to the browser to minimize any risk.

- **Access control:** Each machine runs their required software using a user which has the minimum privileges possible to execute its task, and crucial information and source code can only be modified by the root-user. The root users rely in long strong passwords and disables direct root connection

[2] https://www.vice.com/en/article/vbwz94/experts-find-serious-problems-with-switzerlands-online-voting-system-before-public-penetration-test-even-begins

using SSH. The root CA's private keys and the symmetric backup keys have the read flag set for all users, this makes it easier for an attacker to steal the sensitive keys.

- **Security of data at rest and in transit:** Data at rest is encrypted with AES-256 CTR and the crucial information such as root keys are physically protected in a secure vault, which needs three keys to open, following the "no single point of failure" principle. Data in transit is protected using TLS or SSH, and every connection is first filtered by the firewall. In addition, weak ciphers on HTTPS connections are disabled and mutual authentication is in place where possible.

- **Password Authentication:** The SHA-1 hash of the key is computed in the client side, this means that any access to the database that could leak the password hashes would give the adversary the capabilities to impersonate that user without having to crack the password first.

- **System recovery:** We have the concern that system recovery, for example of the CA, requires physical means to access the Vault. This can delay the recovery of the system or even not allow it, when one of the three members that is able to open the vault is on holidays. If those keys are widespread then the attacker could also be able to tamper with the physical keys and disable recovery. Additionally, that the recovery of a client's key must be done in person could become a problem in the current Covid situation. This converts the Vault into single point of failure for recovery in case that the system is compromised.

2.2 Risk Analysis

In this section, we will discuss the risk analysis of group 6 step by step including the assets, threat sources, risk definitions and the countermeasures proposed.

The assets have been divided in physical assets, logical assets, persons and intangible goods. For the physical assets they provide a complete list including physical protection and physical separation of the different components. They also disable input/output components which may be an entry point for the attacker, and define the mode of operation of the hardware. For the logical assets they list both the software used by each machine and the information stored. The software list may not be exhaustive but the information list includes all the required assets with a short description of their possible use. The persons list is complete but lacks one type of person that could be potentially harmful, the external employees. This kind of employees are in charge of maintenance, cleaning and even maybe security and can have physical access to high level information of other employees and can try to physically disrupt some of the companies system, as for example disconnect the energy supply. Finally,the list of intangible good is complete and well argued.

The threat sources list is complete and in this section the aforementioned external employees are included. However, a more detailed separation and de-

scription of Employees defining for each type the capabilities and risks for the company could have been a good addition.

The likelihood and impact descriptions are complete and well argued, and the resulting risk table is coherent with the descriptions.

For the risk evaluation, we will list the different assets and analyze the described threats and countermeasures proposed.

First the Physical Assets:

- Hardware: Includes a complete description of the most important threats with effective countermeasures as backup and physical protection countermeasures.
- CA's private Network Physical: The threats described are sensible and the countermeasures are simple but effective.
- Internet Connectivity: The likelihood of this event is considered as high but the threats sources presented for this asset are global adversaries. In order to keep the consistency with other threats with similar adversary definitions the threat should be considered medium.

Now the Logical Assets:

- Information: This asset has been misplaced and appears under Physical assets, when it should belong to Logical Assets. However, the list of threats is quite complete and the countermeasures proposed are effective to mitigate the possible attack vectors. We lack an additional threat related to the lost, theft or modification of the CA's private keys or intermediate certificates, which can have noticeable effects on the system.
- Software: The threat list and counter measures are correct and enough to protect the system. The procedure to be applied in case of system disruption is a great addition.
- Firewall: The list of threats only includes possible miss-configurations but not possible implementation or specification problems, which could be useful to have a more complete analysis of the asset. This assets is too generally evaluated.
- Backup: Similar to the last point, only the steal of the backup is contemplated not its destruction or modification which can be even more harmful than the information leak. Additionally, miss-configuration is not contemplated. This assets is too generally evaluated.
- Webserver: Similar to the last two points only the CA part is contemplated. To have a complete threat list, the team should include possible threats to the DB or the Backup server due to the incorrect behaviour of the Webserver. This assets is too generally evaluated.

- Core CA: For the core CA the threat list is more complete than for the last three points but it is still slightly too general. A more in-depth evaluation of the last four assets could be useful for a more detailed analysis of the system.

Let's now analyze the Person assets. In this case the set of threats only includes threat caused by the different types of employees but not threats that will affect the employees, for example a natural disaster or a criminal organization causing damage to an employee. This section analysis seems, therefore, incomplete. Additionally, some of the listed threats keep similarities with threats listed in the Logical Assets section including similar countermeasures.

Too finalize with this part of the evaluation we will center in the Intangible Goods Asset. This asset threat list is complete and well defined. However, the inclusion of hardening, access control and a response team could have been a good addition as countermeasures of the listed threats.

Overall the Risk evaluation is correct and quite complete but the analysis of some threats is too general and some threats are missing or have been aggregated in situations where its separation would have resulted in a more detailed and complete risk evaluation of the system.

Finally, we analyze the Risk Acceptance. All the threats considered as high risk have been mentioned and possible countermeasures proposed. The proposed counter measures seem sensible and effective to mitigate the high risk threats.

As a final note, we would encourage the deletion of the introductory paragraphs given in the template for the completion of the Risk Analysis. This would give the final report a more realistic appearance.

3 Implementation Review

3.1 Compliance with Requirements

This section discusses about whether the system meets the functional and security requirements given in the assignment and whether they are implemented correctly.

This system had a problem that the coreca TLS certificate has expired on 9/12/2020, therefore, many functions were not workable after that date. The resulting behaviour after that point is the generation of internal server error pages as can be seen in Figure 1, when we tried to interact with any function dependant of the coreca.

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 user ID and his password. And the user ID and password are verified by consulting the information stored in the database.

Figure 1: coreca TLS certificate expired causing an internal server error

- The user can log in by using a valid certificate. However, this function does not work when coreca TLS certificate expired
- The user is shown the user information stored in the database.
- The updated user information can not be kept in the database. The system can not show the new updated information.
- A certificate can be issued based on the user information. However, this function does not work when coreca TLS certificate expired.
- The user can download the new certificate, including the corresponding private key, in PKCS12 format. However, this function does not work when TLS coreca certificate expired.

3.1.2 Certificate Revocation Process

- The user can revoke his certificate. However, this function does not work when coreca TLS certificate is expired.
- 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 Administrator Interface

- The CA administrator can login with a CA administrator certificate. However, this function does not work when coreca TLS certificate is expired.
- The CA administrator interface can show the number of issued certificates, number of revoked certificates and current serial number.

3.2 System Security Testing

We inspected the system by performing both black box and white box testing. For that, we applied various information gathering and analysis tools from the course book.

3.2.1 Black Box Testing

Information Gathering: We tried to use the port scanner Nmap to gather some initial information about what ports are open from the clients perspective. In combination with Nmap we used Telnet, Ping and Netcat to look at if and what the ports where responding. However, we were not able to find anything interesting, as it seems portscanning protection is in place. We were still able to identify open ports for https connections on port 5000 instead of the default 443 and. This was to be expected though, as the groups instruction manual told us that the iMovies webpage is accessible on port 5000. Also the port 22 is open for ssh connections, but this was also expected.

Vulnerability Scanning: We used the OpenVAS vulnerability scanner to make an automated and coarse-grained assessment on the system host. The scan gave us some valuable information about the machine, but it was not successful in finding many vulnerabilities. However, it found a vulnerability classified with 'Medium' severity, namely `SSL/TLS: Missing ‘secure‘ Cookie Attribute`. The flaw here is that the 'secure' attribute is not set for session cookies, this would allow a client to pass the cookie to the server over non-secure channels (http) and thus would allow an attacker to conduct session hijacking attacks. For further hardening of the system we could set the 'secure' flag and thus ensure the session cookie will only be transmitted on an https connection.

Functionality Testing: In a last step we looked at the functionality of the webpage by clicking through the pages and looking if everything acted the way it was expected. After we had our first glimpse at how the different components of the web application fit together, we started Burp Suite to look in more detail at what exactly is sent in the different requests. From the analysis with Burp Suite we could tell that the session management is achieved through session cookies stored at the client. We also looked at the javascript code that was sent to us from the webserver and especially at how the challenge response mechanism in the certificate authentication works. It seemed that the front end takes a .p12 file, depackages it into user private key and certificate and then sends the certificate to the webserver for further processing. However to make further investigations we need to switch to whitebox testing.

3.2.2 White Box Testing

Running Services: To check which services are running on the different machines, we used the command `ps -aux`. Also we checked directories such as

`/etc/init.d` and `/etc/systemd` to find services that are started each time the machine boots up. We could find a python service on the webserver machine running a flask server. A similar python service was also running on the CA server, both of them are started through their starting scripts in init.d. From these starting scripts we could tell that e.g. the webserver is started using a flask development server instead of a dedicated webserver such as NGINX or Apache. On all machines there is also a backup agent service running that takes care of file encryption and transfers the resulting data bundle to the backup server. We also checked with which privileges the found services are running and found that all services are started as an unprivileged user.

Code Analysis: By looking at the source code of the separate services, we were able to understand the features and functionalities of the different components in more detail and could also assess the correctness of their implementation. We focused on the python code of the two flask servers (webserver and ca server) and on the `core_ca` python module that is dedicated to the handling of the certificate authority. We payed special attention on how the session handling (user and admin privileges) is implemented and checked if security parameters such as the flask `secret_key` are set correctly.

Access Control: By using the command `ls la` we checked how the access rights on the sensitive resources are set. The access rights are set to reasonable values allowing modifications only to the file owners. However, the root ca private key on the ca server and the symmetric backup keys have the read flag set for all users, this is not necessary in our opinion and stronger access control mechanisms could be put in place here.

Log Analysis: We performed log analysis to see if we could find any hints on possible security vulnerabilities or unusual system errors. By investigating this we realised that the logs are not stored append-only. Although logs are backed up and encrypted by the backup agents, we think that adding the append-only attribute to the log files, would further increase security.

3.3 Backdoors

In this section we describe the backdoors we found on the system during our investigations.

3.3.1 Privilege Escalation Attack

When doing the code review from the webserver, we noticed a suspicious function that seemed to escalate a users privileges on some specific condition.

```
def check_phone_user_agent(user_agent):
    if "Android" in user_agent or "iPhone" in user_agent
    or "Phone" in user_agent:
```

```
        set_responsive_design()
    else:
        set_normal_design()

def set_responsive_design():
    session["phone"] = True
    g.admin = True
```

Given the last function, when the end point /auth/stats is called by a normal user, the application redirects user to admin login page and displays an error indicating that current session is not an admin session.

When the User-Agent header is modified to match one of the conditions, it tries to fetch /auth/stats page but it tries to interact with the coreca component. Because of the expiration of the certificate we get an internal server error instead of the intended page.

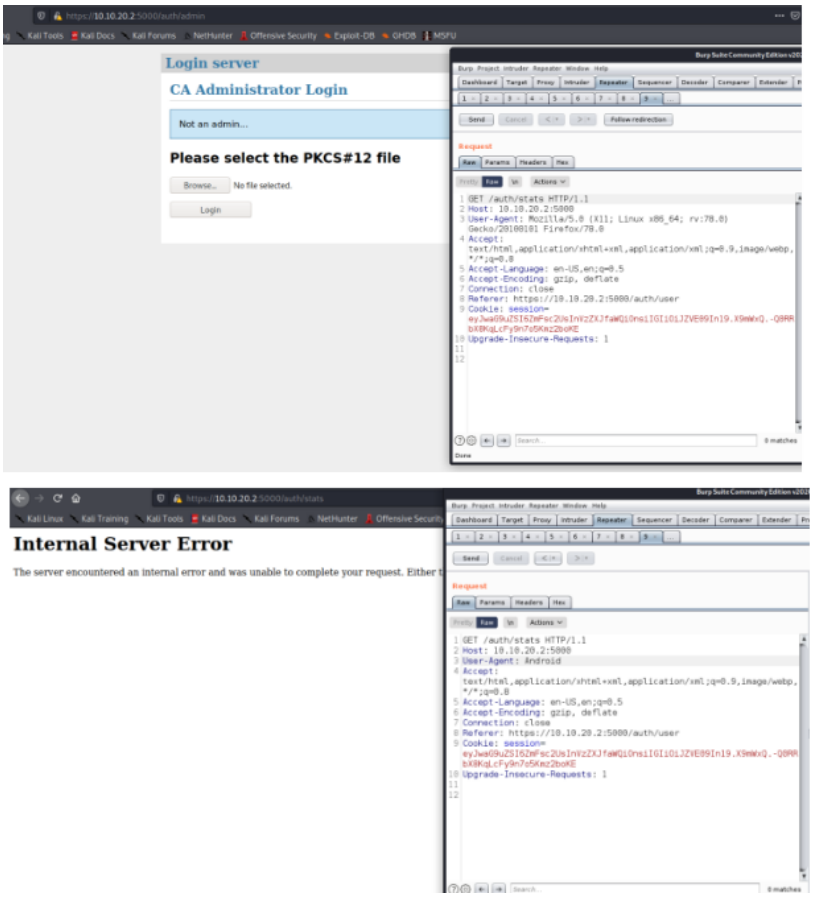

Figure 2: Backdoor exploitation

A visual representation of the exploitation can be seen in Figure 2 above. As we can see this vulnerability has been introduced deliberately and its mitigation is straightforward.

3.3.2 Session Stealing Attack

Attack: The secret key for the flask application serving the client is set to a weak default password ('dev'). An attacker could easily brute-force this, as 'dev' is a common parameter used for the flask secret key, when performing session debugging in a development environment. In fact this parameter is used in the flask tutorial for setting up a simple flask application, thus we can assume that an attacker will try this value with high probability.

```
app = Flask(__name__, instance_relative_config=True)
app.config.from_mapping(
    SECRET_KEY='dev',
)
```

Flask by default uses a mechanism called 'signed cookies', which is a way of storing the current session data on the client (rather than the server) in such a way that it cannot be tampered with. Such a signed session cookie declares the username of a user, if he is currently logged in and also if he is an admin user. In a normal case an attacker has no capabilities to forge the signature attached to the cookie and thus cannot modify his session state. However, if an attacker knows the secret key used to sign the session cookie (as in our case), he can simply modify the content of the session cookie and add a new signature. This allows an attacker to either change his username to another user (session stealing) or even make himself admin user of the web application (privilege escalation). When the signature of the cookie matches, the server has no reason to assume that the attacker is illegitimate and will grant access. An example script to forge a session cookie can be seen in Figure 3.

```
session = {'logged_in': True}
secret = 'CHANGEME'

print(URLSafeTimedSerializer(
    secret_key=secret,
    salt='cookie-session',
    serializer=TaggedJSONSerializer(),
    signer=TimestampSigner,
    signer_kwargs={
        'key_derivation': 'hmac',
        'digest_method': hashlib.sha1
    }
).dumps(session))
```

Figure 3: Script to forge session cookie

In our case this allows an attacker to request the CA statistics without being an administrator. It also allows an attacker to look at the user information of any iMovies user. An important note here is that in the current implementation design of the system the attacker cannot issue or revoke a certificate of another user, as this features are additionally password protected (webserver checks if user is logged in and if so still asks for password as an additional security measure).

Mitigation An easy mitigation for this vulnerability is to set a strong random byte sequence as `SECRET_KEY`. Also we can rely on the `is_admin` field of the database each time we check if a user is an admin rather than just checking the entry of the session cookie. This has the benefit that the database is a trusted resource and might be less easily tampered with.

4 Comparison

System architecture of both teams follow a similar structure, both including a firewall to compartmentalize the network and the same machines with the same purposes, except for administration.

For the System Architecture and Security Concepts section our team and the reviewing team have opposite points of view on how to handle administration. Even though both teams rely on SSH for secure remote administration, we think that the privileged user should disable password-based login and only be accessible through SSH using a more secure private key. The reviewed team argues that it is better to disable the direct connection to the root user such that administrators need to additionally now the key to change to this user. In our view allowing password-based login adds possible risk for privilege escalation as public crypto should be much more secure than a password in any case. Even though adding a second security layer is always a good decision we think that firewall, host allowed list and public crypto should be able to reduce the possible risk of gaining control to any component of the system. Additionally, our team decided to allow a unique point for remote administration using the Ansiblemaster which root user has access to the key material to administer the rest of the machines. This limits the entry point for the internal machines that only recognize this machine as login point. However, this could cause that the subversion of this machine ends in the subversion of the whole system, for this purpose the connection to this machine should be protected by an strong password and a secure SSH connection with physically protected key material.

Another big difference on our system is the use of intermediate certificate authority in the CA to allow fast and better recovery in case a confidential key is leaked. Its always a good practice to introduce a chain of certificates instead of using the Root CA directly.

Also, we rely on the browser and the standard server configuration to perform mutual authenticated login using a client certificate, while the reviewed team decided to implement the challenge-response protocol by themselves.

The rest of the applications and security concepts, like confidentiality or access control related decisions follow a similar reasoning, which indicates that both groups follow a similar thought line to address the same problem, validating both team's decisions.

For the Risk Analysis both teams used the same structure. The assets description and threat sources are quite similar as both teams have followed similar assumptions and a similar system design. However, our team has generalized less in these phases which ends in a longer asset and threat list. This has the benefits of a more detailed description of the threats and assets but could lead to some repetition, mainly for the threat sources. An intermediate approach could lead to a brief yet detailed version of those sections.

The Risk Definition is highly similar as both team followed a similar version of the definitions provided in the Course Book.

For the risk evaluation, we face a similar situation than the one described for the assets and threat sources. Our team listed less general sources and tried to specify each possible case scenario. The result of this is a longer evaluation list, our team has 44 different entries while the reviewed team only has 24. In this case we think our evaluation is more complete as we think that generalization could be problematic in this section. However, for some assets we listed similar threats, counter measures and risk which makes sense as the system design is highly similar in some aspects.

Finally the Risk Acceptance is longer for our team due to the longer threat source. In this case we think that the reviewed teams acceptance is more convenient as our team ended up with similar problems and similar countermeasures leading to repetition. Thus, generalization could have been interesting for this last section of the Risk Analysis process.