System Description and Risk Analysis

Lucie Hoffmann Lucas Rollet Francesco Intoci
Elvric Trombert

02/12/2021

Page limit: 25 pages.

Contents

1 System Characterization

1.1 System Overview

1.1.1 Mission

The system's mission is to act as a certificate authority (CA) that will be responsible for issuing and managing certificates to the employees of IMovies. These certificates are intended to be used for secure email communications between employees and journalists.

1.1.2 Boundaries

The system includes the Web Server (front-end), the CA Server (back-end, REST API), the CA Database Server, the Backup server and the Firewall.
The system cannot guarantee the security of users personal data in the database, since there is no encryption due to legacy reasons.
The system guarantees the security of the private keys associated to the certificates issued to the employees, that are transferred encrypted and stored in the Backup Server. The confidentiality of the private keys associated with the client certificates issued to IMovies employees, or other sensitive information is not guaranteed outside of the system.
We will not consider other machines/services not included in the aforementioned list. It is important to notice that any iMovies internal employee network is considered outside of our scope, but would be physically separated from the CA DMZ through the firewall. As requested in the project specifications, the sysadmins can access the system through the internet, thus eliminating the need for a specific internal employee network - except in the case of a VPN setup, which was considered too much overhead for this project.

1.1.3 Architecture

Figure 1 shows a client (i.e user) who accesses the company's CA system via the Web Server. A user here can either be a simple IMovies employee or a special IMovies employee called a CA admin. CA admins have no special roles with respect to system management, but they have special permissions to access information about IMovies CA.
The Web Server provides the entry point for all users' actions, forwarding clients' requests to the CA Server through the Firewall. That firewall separates the IMovies intranet from the internet. On turn, the CA server fetches data from the CA Database and responds to users requests.

System data are regularly backed up in a remote server, called Backup Server. This data include logs from all the system machines, Database dumps and copies of the certificates private keys issued to the employees.
In our system, we also include one sysadmin per machine who can access the machines through authenticated connections.

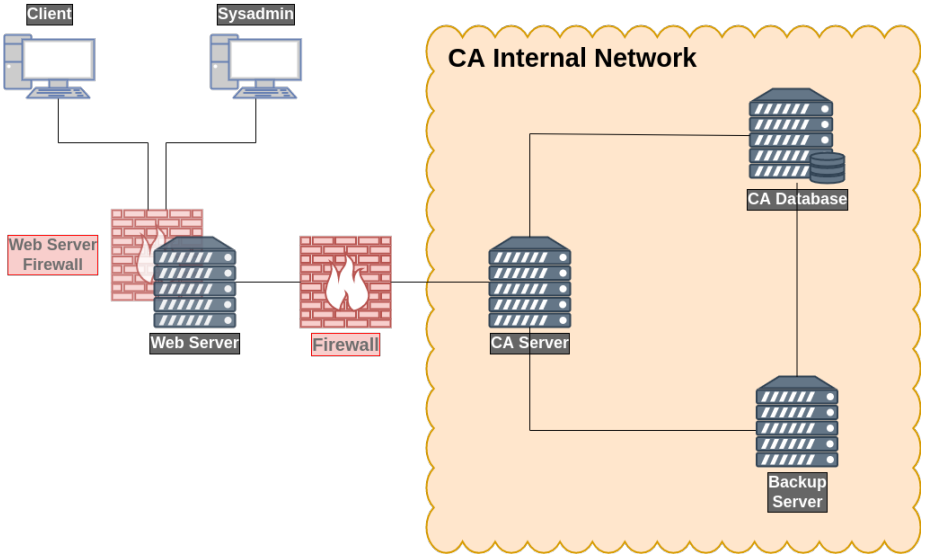

Figure 1: Network Diagram

Note that CA admins and sysadmins are logically separated: CA admins are special employees which can access information about the CA state. Sysadmins are members of IMovies engineering team which deal with the maintenance of the services and log in into the system machines using SSH (even outside of the internal network) with credentials of users in the sudoers group of the machine.

1.1.4 CA

The system contains two different CAs, namely the *Root* and *Intermediate* CA. The former is used to issue certificates and private keys for all the machines to be used for TLS connection establishment between each other, and additionally to the Intermediate CA for issuing client certificates (it will not actually be used actively in the system, it was used just in the setup phase).
The latter is used for issuing certificates to IMovies employees.

1.2 System Functionality

Here we describe the functionalities of the system as a whole, from a user perspective. For a more detailed explanation of the single components, please refer to the Components section.

1.2.1 Webserver and CAserver

To access the CA functionalities, users connect to the Webserver located at `https://web.imovies`.
Here they can login using their credentials or a certificate, if they already have one issued by the CA. The Webserver acts as a proxy and relays the requests of the users, through the firewall, to the API endpoints served by the backend server CA Server.
After logging in successfully, users are authenticated using a cookie. At this point, users can access the homepage and its session is persisted through while browsing to different pages.
From the homepage, users can:

- View and modify their personal information, including their password.
- View information about all the certificates that have been issued on their behalf. They can also decide to revoke some or all of them. Once a certificate is revoked, an user cannot login with it anymore. Users don't need to have their previous certificate to revoke it, it is intended for the situation where a certificate is stolen or a private key is lost.
- Request a new certificate. A new certificate will be generated for the logged in user. It can then be downloaded in PKCS12 format, e.g. with the private key and the public part of the certificate included.
- (For Admin only, after logging in with certificate) access the admin panel. Admins can view information about the current status of the CA and all revoked certificates.

1.2.2 Database

The CAServer sends SQL queries to the Database Server which runs a MySQL instance to retrieve or store data. The Database stores all users personal information, information regarding issued certificates and current status of the CA. See the Components section for more details.

1.2.3 Backup

CA Server and Database regularly send data to the Backup Server. CA Server sends clients' private keys as soon as they are generated, while the Database sends daily dumps from the MySQL database. Users can ask the sysadmin to recover some data they may have lost.

1.2.4 System Administration

Our system can be administrated remotely from the internet by sysadmins. Sysadmins are special users of the system. They are not involved in the CA-application logic we have described so far. They are responsible for the security and maintenance of the machines involved in the system. They have SSH access to all the machines, and they authenticate using their private key. Once logged in, they have root access to the machines. In our current setup, we can have one sysadmin per machine, since they all use different private keys. (For the sake of simplicity we just created one sysadmin with all the credentials on the client machine.)

1.3 Security Design

In this section, we will discuss more in depth about the security of the system.

1.3.1 Architecture

Following the principle of *Compartmentalization*, we will deploy services on different machines (Web Server for front-end, CA Server for back-end, DB Server for storing data, and Backup for resilience).

1.3.2 Network Security

The Network of the system articulates in two parts:

- DMZ - The DMZ hosts the web-server that is exposed to the Internet. We setup two firewalls. The first is implemented using `iptables` on the Webserver. The second one is setup on the Firewall machine through which connection from and to the machines are routed.
- CA Intranet - It contains all the internal services of the organization, namely the CA-server, the Database server and the Backup Server. In the Components section we will describe in more details firewall rules. It is important to note that in a real company situation, the internal company machines (employees, other websites, ...) would run on separate networks.

In designing the network infrastructure, we applied a *Zero Trust* security model for the network:

- Fail-Safe Default: We set up firewall rules following the Fail-Safe Default principle, i.e applying a Default-Deny set of rules based on white-listing.
- Minimize Trust: Even if connection between machines happens in the internal network, all data-in-transit is encrypted using TLS or SSH, minimizing trust in the internal network.

This model should make lateral movement trough the network more difficult in case of a breach.

1.3.3 Security of data at rest

Data in the database will not be encrypted due to legacy reason.
The private keys issued by the CAserver are generated and then immediately sent to the Backup Server. Before sending the certificates, the CAserver encrypts them with the Backup public key.
Note that for compatibility issue with Browsers, certificates had to be generated using OpenSSL from the command line, rather than using the x509 module from python crypto library. Hence, for a brief moment, they will be stored and then removed in CAserver file system. More specifically, the Flask backend will call a bash script which will deal with the creation and encryption of the certificates and private keys.
Backups from the CA server and the database are sent on the backup server to `/data/cabackup/backup` and `/data/dbackup/backup` respectively. Both users `cabackup@backupserver` and `dbackup@backupserver` have limited remote SFTP access to these respective folders only, and don't have SSH access. The backup server then copies these backups to a different folder, non accessible from sFTP connection, in the backup server's home folder.

1.3.4 Security of data in transit

As mentioned earlier, data flowing from one services to another will be encrypted using TLS. Same holds for sending logs to the Backup server. Backup of data from CAserver and Database will rely on sFTP. Furthermore, the private keys backed up when generating a new certificate are encrypted before being sent to the backup server using sFTP.

1.3.5 Access Control

Application Level iMovies employees can access their personal data (email, name, user id, certificates) by logging in on the website with their credentials (id + password OR certificate). After a succesful login, the users are issued a JSON Web Token (JWT) containing their uid and whether or not they are able to access the admin panel.

Following the principle of Complete Mediation, every subsequent API request to the CA Server verifies that the JWT has not been tampered with, and only issues data corresponding to the authenticated user. The JWTs handle both authentication and session management, since their integrity is guaranteed through HMAC-SHA256 signature, they are persisted in the browser using cookies, and they effectively identify a single user.

Also, every access to the application is logged both on the Web Server and the CA Server, following the principle of Traceability. Furthermore, all failed access to the admin endpoints are also logged. Regarding authentication with certificates, we use TLS client authentication which ensures that the certificate used is valid and not revoked.

The CA Server implements a CORS policy which denies any cross-origin request. Furthermore any user input is checked and sanitized before using it in database queries (i.e it implements prepared statements).

The Database also implements security measures at application level: in particular, DAC (Discretionary Access Control) is enable such that only the backend application can access the database. For more information about the privileges of the user connecting from the CA server, refer to the Components section.

Machines Level On every machine there is an admin user called "vagrant" which has root access on the machine, that can only be accessed by the sysadmin. This user can only be accessed through SSH connection with the correct private key, unique for each machine. We will now detail more how each machine handles access control.

- **Web Server**: the services running here are all ran by the user "www-data", which has non-root privileges. There also exists a root of trust between this server and the CA Server to perform TLS client authentication: the signature is verified on the Web Server, and the verification of the revocation status is off-loaded to the CA server backend.
- **Firewall**: no user apart from the sysadmin can access the machine.
- **CA Server**: external users can issue requests to the backend in accordance with the API details described in the components section. The API process runs under the "www-data" user which has no access to sudo capabilities in accordance with the Least Privilege principle. This user can however read the source code, but can't edit it. Same goes for the certificates information stored by OpenSSL (serial number).
- **Database**: users can trigger SQL queries to the database via the Flask backend running on the CA Server, as intended by the system architecture. CA Server sends queries by connecting via TLS to the Database and then authenticating using the credentials of the `certmanager` user. The `certmanager` user holds SELECT,INSERT,UPDATE privileges on the tables of the database created for the CA system. Note that, following the Defence-in-Depth principle, we restrict access to the Database as `certmanager` to originate from the CA Server machine IP.
- **Backup Server**: both CA Server and Database connects to the Backup Server using SFTP. Following the principles of Least Privilege and Separation of Privilege, we defined two different users which are able to connect with SFTP but not using SSH. Those users are confined to their home directory by chroot.

1.3.6 Key management

- Key management: we deal with the following keys in the system:
  - Users' private keys: these keys correspond to the private keys related to the certificates issued by the CA to employees. They are stored encrypted on the Backup Server.
  - Root CA private key: this key is used by IMovies CA to sign certificates issued to the services. It is stored in the Backup Server, encrypted with a passphrase known only by IMovies CEO.
  - Machines TLS private keys: used for TLS connection between machines. They are stored in the associated machine and the Backup Server.
  - Intermediate CA private key: This key is used by the CA server to sign certificates issued to IMovies employees. The key is stored in the CA server and Backup Server.
  - Flask Secret Key: This key is used by Flask backend to sign the JWT for authentication. It is stored in the environment.
  - Backup decryption Key: This key is used to decrypt users' private key stored securely in the backup. It is stored in the Backup Server encrypted with a passphrase known by the sysadmin of the CA Server.

  Additionally the keys that are stored in the Backup Server are stored in a encrypted zip archive which can be accessed only by the Backup sysadmin. Moreover, Root CA private key and Backup decryption key are encrypted as stated before.

1.3.7 Defenses against common web vulnerabilities

- **Information leakage:** being able to know what technologies are used on a website can help the attacker target specific vulnerabilities. To make this reconnaissance step harder, we hardened the NGINX config to remove the server header which could leak the NGINX service as well as its version. We also made sure to not use the default NGinx error pages. There is also no leakage that we are using Flask as a backend.

- **Cookie stealing:** a logged in user's JWT is stored as a cookie in the browser. We hardened this cookie as much as possible to make it impossible to steal in case of an XSS attack. We set the following flags:
  - Secure: the cookie will only be sent with HTTPS requests
  - HttpOnly: the cookie cannot be accessed through client side scripts.
  Furthermore, we enabled the default CORS policy which prevents any requests to other domains than iMovies, as well as the Same Origin policy. Even if there is an XSS, the cookie cannot be exflitrated.

- **XSS:** the use of React for the frontend protects us against most XSS payloads. React escapes all the rendered HTML and displays it as plaintext, making it impossible to inject script tags.
- **CSRF:** since we are using cookies for authentication, our application could be vulnerable to CSRF attacks. To remediate to it we enforced the "strict" SameSite policy on the JWT cookie. This policy ensures that the cookie won't be sent to other domains than web.imovies, making it impossible to exfiltrate. Another CSRF attack would be possible if an attacker got a logged in user to click on a link redirecting to a web.imovies page performing some action like revoking all certificates. This could lead the user to perform unwanted actions, even though it wouldn't lead to any data leak. Our fix was to only perform state changing actions such as issuing or revoking certificates through POST requests, which cannot be triggered without user action.
- **SQL Injection:** we use prepared statements in the backend to escape queries and prevent SQLi.
- **DOS:** some API endpoints performing longer tasks such as writing to the filesystem can be easily vulnerable to DOS attacks. To prevent against that, we added backend rate limiting to our app. For instance we limited the certificate generation to 3 requests per minute.

1.4 Components

In this section, we will describe in depth all the components interacting in our system, as well as their interface and the services they are running.

As a side note, all of our components run the same Ubuntu Server 20.04 operating system, and are then hardened according to their security needs. Furthermore, all communications between components are protected using TLS.

1.4.1 Web Server

IP: 172.26.0.2 - Hostname: webserver

The web-server is the first entry point to our system for any external user. It is located outside of the internal network in a DMZ, and is used to serve static HTML and Javascript files to the client. The user *www-data* runs a NGINX reverse proxy (v.1.18.0) on ports 443 to answer requests made to *https://web.imovies*. The frontend static files are coded using React, a JavaScript library designed to create single-page dynamic websites. When accessing the website, all the frontend code is sent to the client, and the request are performed directly from the client's browser using AJAX. All the functionalities of our frontend interface are described in the System Functionality section. The main advantage of React (aside from developer comfort) is that it provides default protections against injections such as XSS, thus securing the frontend drastically. Note that the

frontend interface is completely separated from the backend, the React app and the Flask app act as two separate applications, with the frontend only interacting with the backend through API calls to retrieve data, authenticating itself with the use of access tokens.

The other main function of this server is to perform TLS client authentication. Note that the web-server relies on the backend to check with the database that the certificate has not been revoked.
Apart from that, no logic is performed directly on this host, all API requests made by the client are proxied using HTTPS to the CA Server through the firewall, which will handle logic and database access. Every request sent to *https://web.imovies/api/** is proxied to the CA server. Every request sent to *https://web.imovies/* only sends the React static files.

This server also runs a simple firewall configured through IP tables to isolate it from the outside, the only ports open are 443 for HTTPS and 22 for SSH. It cannot access any machine in the internal network directly, every request it makes has to be forwarded and checked through the inner firewall machine.

We specifically designed this machine to make the attack surface as small as possible, we installed the minimum number of libraries and allowed the minimum number of services to run. It doesn't run Node.JS or NPM to build the React static files. These files have to be built locally on the developer's computer and then sent to the web server as static files. We made the choice to only allow connections to the website through HTTPS thanks to HSTS, and disabled port 80 (HTTP). The only services running are NGINX, iptables, and rsyslog for logging. All the accesses are logged by the NGINX logs and synchronized with our backups. They contain all the HTTPS requests made to the server, as well as potential NGINX errors. They are the most sensitive data contained on this machine.

1.4.2 Firewall

IPs: 172.27.0.254, 172.26.0.254 - Hostname: firewall
The Firewall is the entry point to the internal network. It doesn't run anything other than iptables configured with the principle of least privilege in mind, to provide minimal exposure. It is probably one of the most sensitive parts of our system, since taking control of it could allow an adversary to open ports. We applied a whitelisting approach, dropping every requests except:

- SSH connections made on port 22 from the outside network are forwarded to the internal network
- HTTPS request made on port 443 by the Web Server (and the web server only) are forwarded to the CA server.
- SSH connection made on port 22 of this machine are accepted to allow administration.

It was considered out of the scope for this project, but this host could also ideally run additional security defenses such as a Web Application Firewall or an Intrusion Detection System.
It is located at the perfect place to perform most of the security checks before entering the sensitive internal network.

1.4.3 CA Server

IP: 172.27.0.2 - Hostname: caserver

The CA server is the most important building block of our system, it is where all the logic of our application is performed. Its main purpose is to expose a REST API to handle all the requests coming from the web server. This API task will be to perform actions such as logging in, issuing certificates, ... A more detailed description of the endpoints can be found below. This component interacts directly with the database by performing SQL queries, as well as with the Web Server by answering API calls proxied through Nginx at the web-server and the firewall.

This server runs a simple NGINX reverse proxy on port 443 to proxy requests towards the API running on a uWSGI socket setup by user *www-data*. Our REST API is developed in Python3, using Flask (v2.0.2). It needs the following pip3 packages to run: bcrypt, Flask, pyjwt, cryptography, mysql-connector-python, Flask-limiter. uWSGI's goal is to run the Flask webserver in a dedicated socket so that it can be compatible with the NGINX reverse proxy.
The Flask server logs every request received, as well as the id of users logging in and out, the certificates that were issued, and the attempted accesses to the admin page. In the end, the only services running are: NGINX on port 443, uWSGI internally, as well as ssh and rsyslog. This server interacts directly with: Backup Server, Web Server (forwarded through firewall), Database.

API Description In this section, you'll find a detailed summary of every API endpoint accessible in the CA server, what they do, and what services they run. The API follows stateless REST principles, every request is treated independently and the data received and sent back uses the JSON format. It is important to note that while most of our endpoints execute SQL queries, they all use **SQL prepared statements** to protect them from SQL injections.

- **POST /api/is_logged_in** - No data
  Verify if the request contains a "token" cookie, checks its signature, and verify in database if the user exists. If all these conditions are met, returns True.
- **POST /api/login** - Data: { "uid": str, "pwd": str }
  Check the uid and password against the database. If the login is succesful, return a successful response as a "token" cookie containing the JSON Web Token used for authentication of later requests.

- **POST /api/login_with_cert** - No data
  This endpoint is called when a user connects to the CA using certificate authentication. For this login to work, the user must have installed a valid certificate in his browser and selected it in the pop-up displayed by the browser during the TLS client authentication process with the Web Server. During this process, the Web Server should have set a X-Custom-Referrer header containing the serial of the certificate. We then check in database if the serial corresponds to any certificate issued, and then generate a JWT for the associated user.

- **GET /api/info** - No data - Authentication required
  This endpoint simply returns all the info (uid, firstname, lastname, email) associated to the token's user account.

- **POST /api/modify** - Data: { "lastName": str, "firstName": str, "email": str, "password": str } - Authentication Required
  Updates the info for the authed user in database. All the given fields are checked: they must not be empty, passwords should be more than 6 characters long (for compatibility reasons with the legacy database), and e-mails should be correctly formatted.

- **POST /api/certificate** - No data - Authentication required
  Generate a new certificate and private key for the token's user, and signs it. Returns the corresponding PKCS12 file to be downloaded. The certificate generation process is detailed more in the next section.

- **POST /api/get_certs** - No data - Authentication required
  Returns a list of all issued certificates for the authed user.

- **POST /api/revoke** - Data: { "serial": str } - Authentication required
  Revokes the certificate with the serial given in the request body if it is assigned to the authed user. The revokation is done by setting the certificate to "revoked" in the database.

- **POST /api/revoke_all** - No data - Authentication required
  Revokes all certificates issued for the authed user.

- **GET /api/admin** - No data - ADMIN Authentication required
  Fetched the current CA state from the database and returns the number of certificates issued, the current serial, the number of certificates revoked, as well a a list of all the revoked certificates in PEM format.

Certificate issuance The certificates are generated using openssl command line instructions from a shell script. This shell script is run by the /api/certificate endpoint on request. It takes the uid of the user requesting a certificate as well as the serial number as parameters. The serial number is retrieved from the database. These parameters are sanitized to prevent any code injection. During the generation process, some intermediate files are generated such as the

client key, the pem certificate, and the CSR. These files are destroyed straight after the PKCS12 certificate is generated. Instantly after the client's private key is generated, it is sent encrypted to the backup server through sFTP, as the specification requires. A serialized version is then stored in the database, with only the publicly available informations in PEM format, along with the user id and the serial number. Once all these steps are executed, the certificate creation is logged in Flask logs and the PKCS12 file is sent to the client.

1.4.4 Database Server

IP: 172.27.0.3 - Hostname: database

The database server only runs a MySQL v8.0.27 service on the default port (3306) as well as SSH (port 22) and rsyslog. This server is only queried by the CA server and sends its backups and logs directly to the backup server. Communications between the CA server and the database are encrypted using TLS and communication from the database and CA server to the backup server is done via SFTP, so it is not possible to decrypt traffic. A CRON job runs daily at 9 am a shell script dumping the database content and sending it to the Backup server using SFTP.
The database dumps are created by the user `dbackup@localhost` having only the necessary privileges to dump the database. A schema of this database can be seen on Figure 2. It stores:

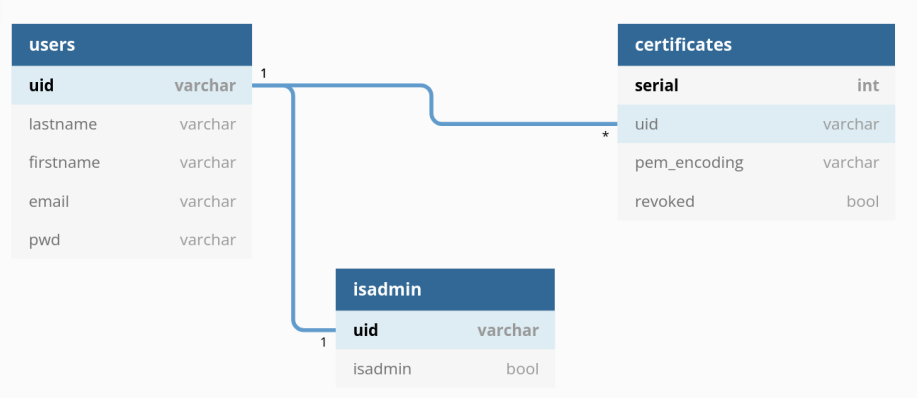

Figure 2: Database Schema

- User information, including the identifier, name, email, and hash of password.
- Information whether the user is a CA admin or not.
- Information about every certificate issued: to which user it belongs, its unique serial number, a base64 encoding of its PEM format, and if it has

been revoked or not.

1.4.5 Backup server

IP: 172.27.0.4 - Hostname: backupserver
The backup server holds the root CA certificate along with its private key. It also contains all the certificates and associated private keys used for the https connection with the WebServer, for the https connection with the CAserver and for the TLS connection used when sending the logs from the other machines to the backup server. It also holds the public and private key used to encrypt the clients private keys generated when creating their Certificates. All these are stored in a passphrase protected zipfile. Furthermore the private key of the root CA along as the private key used to decrypt the private keys of clients are themselves protected with a passphrase as well. We also store the configuration files of our machines in an other passphrase protected zip file.

- Logging: machines send logs to the remote Backup using the rsyslog over TLS, which runs on port 6514. Every access to the machines, including ssh, sudo calls, or connections handled by Nginx, are sent to the backup server. The logging happens "in real time" for every machine.

- Data backup: the database creates daily dumps of all its tables and send them to the backup server for it to store them. Database dumps are put into a special folder accessible by a dbackup user via SFTP on port 22. For legacy reason, the Database backup on the Backup Server are not encrypted, as in the Database machine. The backing-up happens daily (at 9:00am) for the Database, while the private keys are backed-up as soon as they are generated. A daily cron job launched at 9:05am moves the received backups to an inner directory inaccessible via SFTP.
  Similarly, the CA server sends the private keys associated to client certificates as they are generated to the Backup in a specific folder accessible by the `cabackup` user via SFTP on port 22. Clients' private keys are encrypted on the CA server before being sent, and they are stored encrypted in the Backup server which holds the private decryption key. Every time new private keys are added to the backup server, they are copied to a different folder on the backup server inaccessible via SFTP.

1.5 Backdoors

1.5.1 JWT "None" Attack (Easy):

The first backdoor is a well known web vulnerability: the attacker is able to set the alg field in the JWT token to "none", so to evade the HMAC signature check and provide the application with a tampered token. The attacker is thus able to log in as an admin. To see the backdoor in action, follow the following steps:

- 1. Go to https://web.imovies without any valid connection, you should be redirected to /login.
- 2. Craft a tampered JWT token with a valid user id, and the algorithm field set to none. An example of such a token could be:
  Header = {"alg" : "$none$"}
  Payload = {"uid" : "ps","$admin$" : 1}
  The encoded form of this token is
  $eyJhbGciOiJub25lIn0.eyJ1aWQiOiJwcyIsImFkbWluIjowfQ.$
- 3. Set the cookie "token" to the encoded forged token in the browser.
- 4. Refresh the webpage, we are now logged in as the user "ps" which is the CA admin !

1.5.2 Kleptography (Difficult):

Under the pressure of NSA, IMovies is forced to insert a super secret backdoor in their system. In particular, NSA wants to be able to forge certificates using the Root CA private key for surveillance purposes.
Hence, NSA provides IMovies with a script to generate backdoored RSA key pairs: while the bit-length of the key parameters seems right, the .pem file hides a secret. The backdoor works as following:

SETUP

- 1. Generate a small $|\delta| < |n|/4$ odd number which is also invertible $mod(\phi(n))$
- 2. Compute $\epsilon = \delta^{-1} mod(\phi(n))$
- 3. Encrypt ϵ using AES_ECB_128 under a key known by NSA.
- 4. Generate RSA components as usual.
- 5. Append the encrypted ϵ and the key used to the .pem certificate of the root CA. Note that common libraries such as OpenSSL will ignore the comment and parse the certificate as usual. Also, note that this certificate is of public knowledge as it comes pre-installed in users browser to connect via TLS to the web-server.
- 6. Release the public key.

ATTACK

- 1. Get ϵ by decrypting the secret with the known key.
- 2. Apply Wiener's low exponent attack[2] to get δ.

- 3. Once δ is known, get a multiple of $\phi(n)$, that is: $\epsilon.\delta - 1$.
- 4. Factor n by known algorithms[1].

We added a secret endpoint to the web app called /b3ckd00r that will mention a NSA agent called Wiener and the AES key we used. This endpoint is listed in the robots.txt file. Still, the other team should manually inspect the CA certificate to find the secret and the key we appended at the end as a comment, and have knowledge of Wiener's low exponent attack and the factoring problem. Note that we add to change the previous version of the backdoor for compatibility issues given by the huge public exponent in the certificate.

2 Risk Analysis and Security Measures

2.1 Assets

2.1.1 Logical assets

- **Software running on machines** This includes operating systems and applications running on every machine. They could contain vulnerabilities that are not patched, or not be up to date. All machines runs Ubuntu 20.04 LTS version.
- **Firewall** The firewall server connects the DMZ with the company network. It blocks any connection not issued by the web server, apart from SSH connections.
- **Web Server** The Web Server is the entrypoint of the system. It proxies the requests from the users to the CAServer. It offers a first layer of authentication on the users' using certificates.
- **CA Server** The CAServer holds the main logic of the CA system. It handles verification of the certificates, retrieval or update of information from the Database, and issuance of certificates and private keys. It is one of the most sensitive parts of the system.
- **Database** The Database holds all users' personal information, including hashed digest of the passwords. Also it is used by the CAServer to keep state about the CA information. The database is kept in unencrypted for legacy reason.
- **Backup Server** CA Server and Database use the Backup Server to backup users' private keys (encrypted) and database dumps (unencrypted). Additionally all the machines send system logs to this machine.
  Moreover, the Backup Server stores an encrypted zip archive containing all the TLS private keys of the machines, including the Intermediate CA and Root CA private keys.

- **Information** This includes all data of value to the stakeholders, such as users' personal information, issued certificates, users' private keys. This data shouldn't leak and its security must be preserved.
- **Keys** This includes:
  - Root CA private key.
  - Intermediate CA private key.
  - Server TLS private keys.
  - SFTP private keys.
  - Backup private key.

  Protecting the keys with access control is essential for the security of the infrastructure.

2.1.2 Physical assets

- **Physical integrity of hardware, services infrastructure:** the entire infrastructure will be self-hosted by iMovies. Some employees may have access to the physical servers, which could lead to some security risks. Integrity of the server room has to be preserved from unauthorized access, and data shouldn't leak from physical access. Also we have to prevent single points of failures.
- **Internet connectivity:** internet connectivity represents a huge asset for IMovies since the IT infrastructure relies on it. IMovies has a subscription with a well know ISP which provides high speed internet.

2.1.3 Persons

- **Sysadmins**: the company has 5 Sysadmins that administrate the entire infrastructure and have access to the most critical parts of the system. There is one Sysadmin per service, which can access the machines and gain admin privileges.
- **Engineers**: they develop new functionalities and fix bugs. They have access to the codebase of the software running on the machines, but not to the machines theirselves or the sensitive information they store.
- **CEO**: The CEO of IMovies holds the secret passphrase that encrypts the Root CA private key. He does not hold any other credentials for accessing the machines.

2.2 Threat Sources

- **Skilled hacker**: Has strong knowledge on attacks and systems. Can be dangerous, his motivations are money, fame or challenge

- **Script kiddies**: Have basic knowledge and only use well-known exploits and vulnerabilities. Not highly motivated and attack just for fun or money.
- **Hacker organisation**: Powerful group of skilled hackers, having unlimited resources. Their motivation would generally be ideological.
- **iMovies Employee**: Anyone working at iMovie, they could attack the system from inside. They would generally act out of vengeance (they were fired), bribery, or to damage the company.
- **State agencies**: Highly powerful and motivated, they generally have access to unknown 0 days and unlimited capabilities. They generally would act to hide compromising movies, ...
- **Environment**: Environmental factors such as natural disasters should be taken into account. They have no specific motivation but have an high impact.

2.3 Risks Definitions

Define likelihood, impact and risk level using the following three tables.

Likelihood	
Likelihood	Description
High	Threat has high motivation and capabilities to find and exploit vulnerabilities. The defences in place are either not adequate or the attack is simply too potent to defend from.
Medium	Threat has motivation and capabilities to find and exploit vulnerabilities. There are defence mechanisms in place that might reduce the success likelihood of the attacker.
Low	Threat lacks either motivation or capabilities to attack, or defences in place are sufficiently strong to impede the attack.

Impact	
Impact	Description
High	The attack results in major state change or loss of one or more assets (e.g complete unavailability of service, leak or loss of data, loss of customer trust...).It is either impossible or very difficult to recover from the attack.
Medium	The attack results in major state change or loss of one or more assets. Recovery from the attack is difficult.
Low	The attack might or might not result in a slight state change of one or more assets. There are protocols in place to react quickly and efficiently to the attack.

Risk Level			
Likelihood	**Impact**		
	Low	Medium	High
High	Low	Medium	High
Medium	Low	Medium	Medium
Low	Low	Low	Low

2.4 Risk Evaluation

In the following, risks associated with the asset "Keys" will be discussed in the scope of the machine storing the keys.
Also we will refer to engineers and sysadmins as employees. When we will discuss about employees with special permissions, we will refer to them as sysadmins explicitly.

Note: in the table headers, "L" stands for "Likelihood" and "I" stands for "Impact". In the table rows, "L" stands for "Low", "M" stands for "Medium", and "H" stands for "High"

2.4.1 *Logical Assets: Firewall*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Employees unintentionally misconfigure firewall rules, leading to unexpected access to the inner DMZ	Enforce proper training and regular code audits	*L*	*M*	*L*
2	Skilled Hacker/ Hacker organization/ State Agency exploits a 0 day vulnerability that can enable them to change firewall configuration and bypass it	Harden the firewall system as much as possible to prevent privilege escalation	*L*	*H*	*L*

2.4.2 *Logical Assets: Software running on machines inside company network*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Employees unintentionally forget to update software or patch vulnerabilities, resulting in a wider attack surface	Well-trained employees aware of cybersecurity matters, as well as running regular vulnerability scans on the infrastructure	*M*	*M*	*M*
2	Malicious employee leaks data or private keys used by the CA, compromising the integrity of the system	Only let trusted employees access the secrets (*Complete Mediation*), make secrets access as limited as possible, ensure proper key rollovers (*Fail-Safe Default*)	*L*	*H*	*L*
3	Skilled hacker/ Hacker organization/ State Agency exploits a 0 day vulnerability, leading to root access to some/all machines	0 days are impossible to predict, use system hardening as much as possible to reduce the impact	*L*	*H*	*L*

2.4.3 *Logical Assets: Web Server*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Skilled hacker/ Hacker organization/ State Agency performs a DDoS attack on the web server, leading to a denial of service	Deploy DDos mitigations solutions with ISP	*M*	*H*	*M*
2	Skilled hacker exploits a frontend client vulnerability to exfiltrate other users' data (XSS, CSRF, ...) or interfere with the certificate issuance process	Deploy Web Application Firewall, use known libraries with template renderers, command a web security audit	*L*	*M*	*L*
3	Skilled hacker / Hacker Organization / State Agency breach the Web Server, compromising the certificate verification mechanism	Harden the Web Server, deploy redundant security mechanisms	*L*	*H*	*L*
4	Script kiddies try to run automatic exploits on the web server	Same as no.2	*H*	*L*	*L*

2.4.4 *Logial Assets: CA server*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Skilled Hacker/ Hacker organization/ Statal Agency or malicious employee leaks Intermediate CA private key	Ensure both physical and logical access control to server machine and key file (Complete Mediation, Least Privilege).	*M*	*H*	*M*

2.4.5 *Logical Assets: Database server*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Skilled Hacker/ Hacker organization/ Statal Agency or malicious employee leaks Database file	Ensure both physical and logical access control to server machine. Consider upgrade the legacy database to add Transparent-Data-Encryption or encryption at-rest (Complete Mediation, Least Privilege)	*L*	*H*	*L*

2.4.6 *Logical Assets: Backup server*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Skilled Hacker/ Hacker organization/ Statal Agency or malicious employee exfiltrates data and logs from backup	Harden backup machine as much as possible. Employ encryption at-rest whenever possible. Ensure Access Control and Complete Mediation to sensitive data and decryption keys. Ensure not to log sensitive data	*L*	*H*	*L*
2	Skilled Hacker/ Hacker organization/ Statal Agency or malicious employee exfiltrates private keys (server TLS private keys, CAs private keys, Backup decryption key)	Employ HSM or off-site 3rd party solution to safely store keys	*L*	*H*	*L*

2.4.7 *Persons: SysAdmin & Engineers*

No.	Threat	Countermeasure(s)	L	I	Risk
1	Unavailability, due to sickness, employees leaving the company, or even death.	IMovies should follow the principle of *No Single Point of Failure* by replicating roles among different employees (even if this means distributing trust).	*M*	*H*	*M*
2	Sysadmin or CEO lose credentials	Deploy key and credential rollover procedures, store credentials backup with off-site 3rd party solutions	*H*	*H*	*H*

2.4.8 *Physical Assets: Services infrastructure*

No.	Threat	Countermeasure(s)	L	I	Risk
1	A malicious employee with access to the database steals hard drives to exfiltrate or destroy data	Encrypt all data at rest, reduce access to secrets to minimal set of trusted employees (Least Privilege, Minimize trust), enforce strong physical access control policies to the server room, like biometrics, access cards, security personnel... (Complete Mediation)	*L*	*H*	*L*
2	State agencies manage to break in the data center to destroy or leak data. They might be motivated by a sabotaging a movie on a n ongoing investigation on the government.	Same as no.1	*L*	*H*	*L*
3	Environmental disaster like flooding, fire destroys the infrastructure	Make a disaster recovery plan, if possible store the servers in a non-floodable area, store the backup server separated from the rest of the infrastructure. Employ redundant Backup strategies like "3-2-1" rule (3 copies of data, on 2 different medias, 1 off-site) (No Single Point of Failure)	*L*	*H*	*L*

2.4.9 *Detailed Description of Selected Countermeasures*

DDoS Mitigation: In the context of mitigating the impact of a DDoS attack and reducing its impact, IMovies should rely both on on-premises solutions (e.g deployment of a reverse proxy, traffic monitoring to tackle the attack from its start, implementation of protocol for augmenting resilience and availability like replication) and out-of-premises solutions (e.g rely on a CDN, stipulate a plan with ISP, rely on more ISPs) (i.e take a Defence-in-Depth approach by relying on several, diverse, defence mechanisms).

Regular security audits: IMovies infrastructure is indeed a complex system: for this reason, regular code audits are essential to individuate and fix eventual security flaws. The company should rely on external professionals offering security consulting, periodic training sessions to the internal staff, and security reviews of the system like penetration tests. IMovies internal staff should also be involved in the process by conducting internal reviews as well as periodic maintenance aimed to keep the deployed software up-to-date.

Infrastructure management: In this risk analysis, we mentioned that physical access to the infrastructure should be constantly monitored and restricted where possible. We therefore suggest implementing physical access control to the machines hosting company services(i.e access cards, security personnel). The solution should follow the principles of Least Privilege and Complete Mediation by granting access only to the few, needed employees.

Backup keys, credentials and personnel management: One of the most sensitive assets to the system is indeed the Backup Server IMovies uses to store the employees credentials, as well certificates and private keys. Proper management of this asset is clearly essential. In particular, we highlighted the following critical points:

- Key rollover: whenever a key is compromised, the IMovies should start a Key Rollover procedure.
- Credentials: Since gaining the credentials to access the machines will grant access to sensitive datas, including private keys, it is crucial to implement proper access control to all the machines. Complete mediation, Traceability and Least Privilege principles must be applied in this case. Also IMovies should consider relying on 3rd parties for outsorcing the access credentials.
- Personnel: since having access to the machines is essential to IMovies for guaranteeing the management of the system, the company must avoid having a Single Point of Failure by having one Sysadmin holding the credentials. We suggest applying replication and having at least two persons in charge of each machine, at cost of Distributing Trust.

2.5 Risk Acceptance

2.5.1 *Intranet Software*

No. of threat	Proposed additional countermeasure including expected impact
1. Out dated software	IMovies could additionally build a SOC and an incident response team to tackle eventual attacks which exploited the mentioned vulnerabilities. The Impact in this case might decrease from *High* to *Medium*.
2. Insider leaks data	IMovies should consider investing in a commercial version of the DBMS in order to exploit more secure key management policies like key vaults on cloud or HSMs.

2.5.2 *Firewall*

No. of threat	Proposed additional countermeasure including expected impact
1. Misconfiguration leads to vulnerabilities	Same countermeasure as threat no.1 for Software

2.5.3 *Web Server*

No. of threat	Proposed additional countermeasure including expected impact
1.DDoS	As described in the countermeasure section, consider implementing DDoS mitigation also on premises and investing on cloud based solutions like CDN. The impact of such an attack could be mitigated by such countermeasures.
2.Front-end vulnerabilities	IMovies could additionally deploy a IDS or a IPS to mitigate the effect of a successful attack by an hacker. The impact can be mitigated from *Medium to Low* .

2.5.4 *CA Server*

No. of threat	Proposed additional countermeasure including expected impact
1. Leak of Intermediate CA private key	Apart from storing the key under proper access control, IMovies might consider using a Hardware trusted module to store the key. Leaking the Intermediate CA key still holds a considerable risk.

2.5.5 *Backup Server*

No. of threat	Proposed additional countermeasure including expected impact
1. Leak of private key	Apart from storing the key under proper access control, IMovies might consider using HSM or cloud based solution to store the keys. Leaking the private keys locally under access control and encryption still holds a considerable risk.

References

[1] Factoring n using phi or lambda. URL: `https://groups.google.com/g/sci.crypt/c/wDV49EsAZQ0`.

[2] Wiener. URL: `https://en.wikipedia.org/wiki/Wiener%5C%27s_attack`.