Review of Group 4 by Group 1

Cyrill Krähenbühl Silvan Egli Lukas Bischofberger

December 14, 2016

Page limit: 18 pages.

Contents

1 Background

Developers of the external system: *Badoux Nicolas, Chibotaru Victor, Ilunga Marc*

Date of the review: 6th December 2016

2 Report and Design Review

2.1 System Architecture and Security Concepts

2.1.1 Architecture

Positive aspects

- There is a firewall on a separate machine where all incoming and outgoing traffic passes through. All communication between the system and the outside is based on either SSL/TLS or SSH which provides integrity and secrecy. Only system relevant ports are open and redirected to the corresponding machine in the system which reduces the attack surface of the system.
- The database server is on a separate machine. Even if part of the system (e.g. the webserver) are compromised the user data may still be uncompromised.
- The log files from different machines, all application data and all configurations are backed up to a separate machine which allows faster restoring of the system after failures. Making the logs append-only additionally helps detecting intrusions or modifications of the system since existing log entries can not be changed and therefore traceability is guaranteed.
- The root CA, which signs the intermediate CA, is not used in the running system. This is a good practice, because in a case where the intermediate CA private key gets compromised, the root CA can still revoke the intermediate CA's certificate. After that the root CA can create and sign a new intermediate CA.

Negative aspects

- CA core and web server are not separated. Separating them would increase the resilience against partial compromise according to the principle of compartmentalization. For example, even if an attacker had full access to the file system through a vulnerability or misconfiguration in the web server he would not be able to access the user certificates.

2.1.2 Security Design

Positive aspects

- They apply the minimum exposure principle whenever possible. For example only sending user data between the webserver and the database server, only accessing the backup server directly from the firewall and sending logs and backups to the backup server but not sending anything in the other direction. All these decisions reduce the attack surface of the system and expose as little information as possible.
- The sessions are managed using a Secure Cookie[1] that is not alterable from the client because it adds a checksum the server checks for. All web traffic uses HTTPS and there are measures against Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF). Active sessions expire after being inactive in the system for a certain amount of time or after a longer fixed amount of time to prevent someone from hijacking the session and keeping it alive.
- Keys for the certificates are generated using OpenSSL and up-to-date algorithms and are only sent to the clients over secure channels such as HTTPS. OpenSSL is a widely used tool for creating keys and certificates and it is not likely to contain any major bugs and such bugs would be quickly fixed. The keys are backed up to be able to restore the CA authority in case of a failure. The private keys of the clients are not encrypted in the CA authority, which is reasonable since it would not actually increase the security because the CA authority needs them in an unencrypted form.
- The data at rest is secured using full disk encryption provided by the Debian system. The full disk is a good countermeasure against physical theft of the hardware and is a nice application of the Complete Mediation security principle.

Negative aspects

- The database is not backed up to another machine and thus there is no resilience against failure in this component. If an attacker could e.g. overwrite or delete all entries in the database or if he could damage the harddisk of the database server, one could not recover a working system.

2.2 Risk Analysis

In this section we will discuss the evaluated assets, the according threats and countermeasures. We point out the validity of the threats and countermeasures and if their likelihood and impacts are defined appropriately. For this we list the assets:

[1] http://werkzeug.pocoo.org/docs/0.11/contrib/securecookie/

Web Server In general the threats and their countermeasure seem appropriate, but there are a few threats missing. The analysis focuses mostly on rendering the service unavailable, but does not talk about integrity of the systems. For example a possible threat that a hacker or malware could get access to the system and install sniffing software or similar is not considered. Also it does not mention the possibility of changing the service in a non-destructive manner. Furthermore we think that the impact of a DDoS is rated too high, as it would probably only cause a temporary service outage which is nowhere as bad as a compromise of e.g the CA root key. Additionally the risk acceptance discusses the possible solution for DDoS but ignores the possibility for simple counter measures (e.g additional rules for iptables).

Database / Backup and Log Server The listed threats and countermeasures seem fine, only for threat #3 it is not clearly described how this should be a threat. It is not clear if the term "employee" is used differently than described in section 2.1, because it is odd that a client of the system could somehow delete the database, logs or the backups.

Firewall Here the possible misconfiguration of the firewall is missing, also the threat of bypassing the firewall (e.g. its malfunctioning or not serving its purpose) should be listed. This would probably introduce a higher risk than the ones listed currently.

For all the physical assets we are missing natural threats, e.g. component failure, bad components, compromised hardware or impact of nature. There are various threats that could not only impact the availability but also the integrity of the system. The use of FDE is mentioned, but devaluated in the same sentence. It is not quite clear how much it really helps. In general a lot of attention is given to availability, but the integrity of the systems is mostly disregarded. On all the systems introduction of backdoors, sniffers or loggers are not discussed. Another asset we were missing in the evaluation was the system configurations and application code which could either be misconfigured, stolen or modified all resulting in a higher risk for an attack.

Private Keys The impact for threat #2 seems too high as certificates can be revoked and new keys would then be generated. On the other hand the likelihood should be increased, as stealing private keys of the user is probably the easiest entry point to the system.

User Data The threats and countermeasures seem appropriate but the likelihoods again should be increased, as user data is probably the most interesting and most targeted asset.

System Logs The evaluation doesn't mention the possible tampering/deletion of logs through insiders. Also it ignores the explicit threat of malware or

hackers deleting, modifying or injecting false logs.

Internet connection First, the connection aspect could/should be separated in internal and external connectivity. For both assets the availability aspect is missing. For the internal network / connectivity threats to hardware should probably be discussed.

Trust The possible threats to trust are potentially much bigger and therefore more threats should be considered. The leakage of data of some form by anyone could be taken into consideration

Persons The evaluation of the persons asset should be split into several evaluations as very different threats apply to system administrators and employees. Thus also very different countermeasures can be taken.

In general we would like to point out, that a spell correction would have helped the analysis formally and it seems that a lot of sections were copy pasted. This was shown by the fact that the system names were not updated after copying. Also in our opinion referring to the already proposed countermeasure would have been better than copy pasting the same all over (verbosity not helpful).

In conclusion we would like to mention that we liked most of the countermeasures described but for some we are not sure how they can be realized in practice and how effective their implementation would be. Further we were surprised by the specified impacts and likelihoods, in our opinion it is not realistic if the risk always evaluates to "low".

3 Implementation Review

3.1 Compliance with Requirements

In the following we will analyze the implementation of the functional and security requirements given in the project assignment.

3.1.1 Functional Requirements

1. **Certificate Issuing:** The web application provides all requested functionality needed for the Certificate Issuing Process. This includes login with credentials or user certificate, changing the user's information, requesting a new certificate, displaying the the new certificate in a list along with all other valid (not yet revoked) certificates and the possibility to download all valid certificates. Client certificates are issued by an intermediate CA which uses a certificate signed by the root CA.
2. **Login with client certificate:** For the login with client certificates, the web server (nginx) verifies the certificate using the intermediate and

root CA's certificates. The result is then forwarded together with the user certificate to the backend (Flask application) which checks if the certificate is not revoked and whether the user id (uid) provided in the certificate's subject field exists. If so the user is logged in.

3. **Certificate Revocation:** In the list of valid certificates there is a revoke link allowing the user to revoke the corresponding certificate. Before revoking, the backend checks if the certificate was not already revoked. The revocation process includes the update of the CRL. Login with revoked certificates is prevented by the backend as described above. The newly generated CRL is published on the web server. The CRL however, is only accessible for logged in users for which we don't see any specific reason. Moreover, the CRL contains no signature which bars the user from verifying the CRL's authenticity.

4. **CA Administrator Interface:** CA administrators can inspect the CA's current state (number of issued/revoked certificates and current serial number) through a dedicated web interface. They must provide a valid user certificate in order to be authenticated. CA administrator authentication is implemented in the same way as client authentication. As a last step, the backend checks if the user in the certificate is listed in the ca_admins.txt file which contains the name of all CA administrators. This prevents a normal user from accessing the admin interface.

5. **Key Backup:** The backup for client as well as CA keys and certificates is located on the logbox machine. Newly issued client certificates and keys are transferred after creation over an SSH connection from the webbox to the backup. However, if the logbox is not running (e.g maintenance reasons) or the connection fails for other reasons, the backup will not be performed. This is a problem as there is no regular key backup (mentioned in section 1.3 of the report) meaning, that if the backup of client keys and certificates at creation time fails, they will never be backed up.

6. **System Administration and Maintenance:** The remote system administration all happens over SSH. Every component runs an ssh daemon listening for incoming SSH connections on port 22. The firewall forwards incoming TCP connections on the ports 12346, 12347, 12348 to port 22 on the webbox, sqlbox and logbox respectively. The automated backup solution is partly implemented. For the logs they use rsyslog which forwards the system and application logs to the logbox. The configuration files are not backed up automatically and neither is the user database.

3.1.2 Security Requirements

1. **Access Control with regard to CA Functionality and Data** The Flask backend restricts access to the CA Functionality over https to logged in users only. It also checks whether a user is authorized to download or

revoke the certificate in question. The webserver verifies the signature of provided user certificates in order to prevent access with a faked certificate. As the application server (uwsgi) is started as service with user caweb and group www-data and the issued user keys and pk12 files are owned by a different user, the webserver adheres to the principles of least privilege.

2. **Secrecy and Integrity for Private Keys in Backup** The harddrive of the logbox machine is encrypted which protects against physical theft. The machines are all in the internal network which makes eavesdropping on the private keys difficult for an attacker outside this network. The backed up keys are owned by a special backup user.

3. **Secrecy and Integrity for User Data** The harddrive of the sqlbox machine is encrypted which protects against physical theft. The user data which is sent between the client and the system uses HTTPS, which protects against an eavesdropping attacker. A user can only change his own user information after authenticating to the system which guarantees the integrity of the user data. The system reveals user data only to the user that authenticated himself.

4. **Access Control on all Components** Access control should be defined according to the principles of least privilege, minimum exposure and compartmentalization while still remaining usable and as simple as possible. For this section, good practice means according to these principles.

   (a) **Firewall**: There are two user accounts on this machine: *admin* and *syslog*. The user *admin* has its home folder `/home/admin` and `/var/mail/admin`. The user *syslog* also has its own home folder `/home/syslog` and logs the iptables in `/var/log/iptables`. It is good practice to have a separate user that stores and owns the log files.

   (b) **Webbox**: There are three user accounts: *admin*, *caweb* and *syslog*. The users *admin* and *syslog* are similar as in the firewall except that *syslog* stores logs of different applications. The user *caweb* stores all CA scripts and application functionality in its own home folder `/home/caweb`. Having a user *caweb* which perfoms all CA application related operations while having no admin rights is a good practice since compromising this user does not automatically grant full access to the system. He can for example not modify any log files (apart from those of the application) since they are owned by *syslog*.

   (c) **Sqlbox**: There are two user accounts: *admin* and *syslog*. Both are similar as in the firewall except that *syslog* stores logs of different applications. The `mysql` database has three users: *root*, *rsyslog* and *caweb*. Only *caweb* can access the user data from the machine webbox. *Rsyslog* can only access log data from the sqlbox itself while *root* can access any database from sqlbox itself. Restricting access

to the user data or log data to one user and one machine is a good practice.

(d) **Logbox**: There are three user accounts: *admin*, *syslog* and *backups*. The user *admin* behaves similar as in the firewall. The user *syslog* has logs from all machines (including itself) stored under `/var/syslog/` using the following naming convention: `<machine>/<app name>.log` with the ext file attribute `a` making them append-only. Since the files are append-only, an attacker that does not have the same rights as the syslog user or root rights cannot remove his traces by modifying the log files. The user *backups* owns backups of different webserver and CA authority applications and settings stored in his home folder `/home/backups`. These users are separated because they serve a different purpose, logging versus backup.

3.2 System Security Testing

We examined the system using white and black box testing, for this we roughly followed the OWASP testing guide[2], from where we picked several test cases.

3.2.1 Black box testing

For black box testing the web application we used ZAP[3]. Additionally we used different tools to test the HTTPS connection[4]. A very broad vulnerability scanning was also performed with Nessus[5]. No weak ciphers have been detected for HTTPS, also no XSS attacks or SQL injections have been found. We only discovered, that UDP port 12345 was open inexplicably. TCP ports 12346-12348 were open and an SSH daemon was listening, as described in the system architecture. Furthermore, we examined the system's functionality and security design by clicking through the application. We looked at the used CSRF tokens and cookies which also seemed secure. By inspecting the fields of the user certificates we found the command injection vulnerability. Also access control regarding to the CA functionality (e.g. downloading the certificate of another user, login with a forged certificate) was tested by hand. In order to scan and connect to open ports we used, amongst others, the system commands `nmap`, `telnet` and `netcat`.

3.2.2 White box testing

For white box testing we used code review, privilege escalation techniques/scripts and information gathering approaches[6] on all components. We checked file permissions and attributes by using system commands such as `ls`, `namei`, `lsattr`

[2] https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents
[3] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[4] http://wiki.cacert.org/SSLScanner
[5] https://www.tenable.com/products/nessus-vulnerability-scanner
[6] https://netsec.ws/?p=309

and identified running network services and processes with the use of `netstat`, `ps` and `lsof`.

Webbox: On the webbox machine we analyzed different configuration/settings files including those of nginx, uwsgi (application server), openssl (used for CA functionalities) and Flask but could not find any suspicious or unusual configuration. Looking at the application's python files and the shell scripts (containing the openssl commands) confirmed the existence of the command injection vulnerability together with a Git repository file we found in the Flask application folder. This allowed us to inspect their Flask implementation history including some interesting commits named "adding second exploit".

Firewall: For the firewall we used system tools like `iptables` in order to investigate the defined firewall rules. The logs and all running services were also closely analyzed using linux internal commands.

Sqlbox: In order to check the existing database users and their privileges we used `mysql`. We especially checked which user have access to a certain part of the database and from which machine they can connect to the database.

Logbox: We investigated the backup of log files and application files. We especially looked at the files owned by *syslog* and what kind of log files from which machine are stored. For the backed up application and configuration files owned by *backups*, we checked which parts were updated and how the updates were performed.

3.2.3 Countermeasures

In this section we briefly discuss the implementation of the proposed countermeasures in the risk analysis. The number after each countermeasure corresponds to the number in the risk analysis.

Web Server

(a) **FDE [7]:** Implemented using Debian's encryption capabilities

(b) **DDOS (2):** The mentioned proposals in the risk acceptance seem legitimate. However we want to mention that all types of ACK and SYN-ACK DDoS attacks as well as DDoS attacks that use bogus TCP flags could already be prevented by adding some additional iptables rules that DROP those packages [8]. One could also argue to perform the DDoS prevention on the firewall might be more appropriate as it then happens at an earlier stage and can be implemented on top of the already existing iptables rules.

[7] `https://en.wikipedia.org/wiki/Disk_encryption`
[8] `https://javapipe.com/iptables-ddos-protection`

(c) **Monitor Intrusion and Attacks (4):** The intrusion and attacks are logged in the nginx, uwsgi and Debian's system logs. However, the system administrator needs to scan them manually in order to detect any strange behavior. Furthermore, once an attacker gains root access, these files can be modified too. More sophisticated solutions could aim for Host Based Intrusion Detection, automated Log File Monitoring, automated alerting system or File Integrity Checks using tools like Tripwire[9] for example.

(d) **Monitoring Logs (5)**: Likewise here, a system administrator manually needs to observe the logs of the application server (uwsgi) in order to detect harming employees. Probably a more realistic approach would be to include a verification step into the process of issuing user certificates.

(e) **Access to CA private key with two user credentials(5):** Even if we don't understand how this helps against harmful employees, this is not implemented.

Database Server

(a) **Monitor Intrusion and Attacks (2):** The same applies as for the Web Server.

(b) **Backup (3)**: There is no backup of the MySQL database.

(c) **Protect DB against user misbehavior (3)**: There is no protection against the misbehavior of a single employee in the form of requiring two user credentials or something similar.

Backup and Log Server

(a) **Monitor Intrusion and Attacks (2):** The same applies as for the Web Server.

(b) **Require two credentials to delete (3)**: The same applies as for the Database Server.

Firewall

(a) **Alert system for status(2)**: There is no alert system implemented.

Private Keys

(a) **Access Control (1)**: As mentioned in 3.1.2 the private keys and the pk12 files of the users are accessible to the web server. This is not needed as the files are passed through the backend for download anyways and should therefore be changed.

[9] `http://www.tripwire.org/`

(b) **HTTPS (2)**: HTTPS is properly implemented on the webserver and weak cyphers are disallowed.

(c) **Notification Channel for Irregularities (3):** Could not be found.

User Data

(a) **Access Control (1):** There are three users for the mysql database on the sqlbox machine: root@localhost, caweb@webbox and rsyslog@localhost. root can access any database. caweb can only access the iMovies database and rsyslog can only access the Syslog database. If an attacker would gain access to the mysql database but not as user caweb or root he would not be able to extract user information.

(b) **HTTPS (2):** As for the Private Keys, HTTPS is implemented and no weak ciphers are allowed.

(c) **Notification Channel for Irregularities (3):** No notification channels have been found.

System Logs

(a) **SSH (1):** The administrators connect to the system using SSH and thus the communication is confidential and no logging information should be leaked. No security issue has been found in the SSH usage.

(b) **Log encryption (2.1):** The harddisk where the logs are stored is encrypted. No further (file level) encryption scheme has been found.

(c) **Integrity control (2.2):** Using the ext filesystem attribute `a` which only allows appending to files, the integrity of the logs is ensured after each backup (unless the attacker gains syslog or root privileges on the logbox machine).

(d) **Notification channel (2.4):** No notification channels have been found.

Internet connection

(a) **SSH and HTTPS:** As already stated, the used SSH and HTTPS implementations are secure.

For assets as trust and people no technical implementations were proposed, thus they were not evaluated in the review.

3.3 Backdoors

We were able to find the following two backdoors using a combination of both, black- and whitebox testing.

3.3.1 Command Injection

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application [10].
When issuing a new user certificate, the Flask application takes the information on the user from the database and creates an openssl system command in the form of a string. This command is then executed by making use of Python's subprocess module [11] with the privileges of the Flask application (caweb user). We found the vulnerability by looking at the certificate's subject field containing an organizationalUnitName (OU) attribute which maps to the user's first name. Furthermore the result of the executed system command is returned as an HTML comment in the index.html file.

Vulnerability: The vulnerability arises from two inattentions:

1. **Insufficient user input validation:** The only restriction on the user's first name is the length of 64 characters. Appart from that any string is allowed.
2. **shell=True:** Setting shell=True in Python's subprocess module makes the command beeing executed through the shell, allowing any useage of shell supported features such as command substitution.

Impact: As a result of the vulnerability the user can execute any shell command which has the privileges of the Flask application (caweb user). This includes access to the application's settings (such as secret key) and also the possibility to act as the CA which means having all the functionality of the CA (e.g. issuing/revoking certificates) as well as access to the users' and in the intermediate CA's private keys.

Exploit: The space of possible commands beeing executed by any user is only limited by the privileges of the caweb user. As an example: by setting the first name to the following string

```
$(grep -rw secret_key app/|sed 's/\ /*/g;s/\//*/g'\)
```

and then issuing a certificate, we were able to compromise the application's secret key by retrieving it from the OU attribute in the certificate's subject field. We then could have created a cookie for an arbitrary user and sign it with the secret key.

Mitigation: The vulnerability could be mitigated by the following two countermeasures.

[10] https://www.owasp.org/index.php/Command_Injection
[11] https://docs.python.org/2/library/subprocess.html

1. Properly sanitize the user's input by allowing only alphanumerical characters for example. Note that this should be done on any field not only first name as they are vulnerable too.

2. Setting shell=False in the subprocess module and therefore disable all shell based features. In case the application relies on any shell based features it should use the wrappers provided by python. (e.g. os.walk())

3.3.2 UDP Bash

We discovered, that the port 12345 for UDP packets is open on the firewall. Upon further inspection we noticed, that communication on this port is nated to the webserver where a hidden process (executed by root, located in /home-/admin/.hidden/a.out) is listening on the port. Using some reverse engineering we discovered the password (6p9mkXhw) which needs to be sent when connecting to the mentioned port using e.g.

```
$nc 192.168.56.10 12345 -u
```

Then we got access to the webserver bash as a root user, which allows to execute whatever command we like.

4 Comparison

In this section we would like to compare our system to the one we reviewed.

There are two aspects the reviewed system does we like but did not do ourselves. The first being FDE, the second encryption of the client keys. For both we think it is good to have, but it hinders the usability of the system and is not particularly useful (e.g. if all keys use the same password). We also like the simplicity of the reviewed system, whereas our system is built a little more complex. And nevertheless they check certificate revocation status in their backend, whereas we rely on nginx. We like their solution because it eases error handling.

On the other hand there are some things we like better in our system. We have some basic DDoS protection which is missing in the reviewed system. Their system is missing periodical backup of configuration and keys and also the database backup is missing. Further, the reviewed system pushes to the backup, whereas in our system the backup pulls the data to backup.

A last thing which is different and which we think is not that nice, is that the webserver and the CA scripts are on the same machine whereas we separated the components onto two servers.