Cybersecurity Project

Ryan Byloos (182167) and Thomas Lavend'Homme (181386)

May 31, 2022

1 Security Implementation Choices

1.1 General Design

SSChat (Software Security Chat, or Super Secure Chat) is designed as a web app, including a user-side frontend and a server-side backend. SSChat uses very common web technologies, such as the Django framework and React. Is is using common technologies on-purpose. These projects have well-detailed security pages and tutorials. The choice of web technologies was made because modern web browsers provide a lot of build-in protections, such as HTTPS SSL encryption, sandboxing from the OS, etc.

1.2 Django API and Database

1.2.1 Authentication

Account authentication is handled via the Django authentication system. Django has a built-in account system, using the standard username-email-password scheme. Django stores password using PBKDF2 by default. PBKDF2 slows down brute-force attacks by applying multiple times a pseudo-random function of the password, a salt and the iteration number. As seen during the course, this makes rainbow attacks much more compute-intensive. PBKDF2 is not yet broken when configured correctly, but a sysadmin with a very restrictive threat model could easily use the Argon2 algorithm instead.

Accounts must be protected from brute force login attempts. django-axes is used to ban the IP of someone who failed to login three times in a row. The current policy to ban the IP users until a system admin unbans it. The logs of a ban, resulting in a `403 Forbidden` response with the body `Account locked: too many login attempts. Contact an admin to unlock your account.` is shown in Figure 1. This policy is quite extreme, and probably too strict. A sounder policy would be to add a captacha and/or ban the IP after a few login attempts to prevent automated login attempts by bots. If failed logins continue to arrive, then the account should be temporarily locked, an email should be sent to the user to warn them of the attempt to log in to their account (eventually with the IPs of the attackers). The email could also ask to reset or strengthen the password.

An additional security measure for the account would be to add two-factor authentication to the authentication procedure. However, this has not been implemented.

1.2.2 Session management

Django includes the concept of sessions. When the client makes a request to the API, a cookie containing a random number, the session id, is sent to and stored by the client.

```
1 "GET /msg/sign_in HTTP/1.1" 500 145
2 AXES: New login failure by {username: "BannedUser", ip_address: "127.0.0.1",
      user_agent: "vscode-restclient", path_info: "/msg/sign_in"}. Created new
      record in the database.
3 "GET /msg/sign_in HTTP/1.1" 200 42
4 AXES: Repeated login failure by {username: "BannedUser", ip_address: "127.0.0.1",
      user_agent: "vscode-restclient", path_info: "/msg/sign_in"}. Updated existing
      record in the database.
5 "GET /msg/sign_in HTTP/1.1" 200 42
6 AXES: Repeated login failure by {username: "BannedUser", ip_address: "127.0.0.1",
      user_agent: "vscode-restclient", path_info: "/msg/sign_in"}. Updated existing
      record in the database.
7 AXES: Locking out {username: "BannedUser", ip_address: "127.0.0.1", user_agent:
      "vscode-restclient", path_info: "/msg/sign_in"} after repeated login failures.
8 Forbidden: /msg/sign_in
```

Figure 1: Logs of the API after banning the user called "BannedUser" due to 3 failed logins in a row.

In the following requests, the cookie is sent back with the request. If the client makes a successful login, Django stores the fact that the session id is authenticated to the specific user used for the login. The user does not have to login until the session id expires, which happens after 2 weeks by default in Django. The user can also log out to delete all information liked to his session id.

Cookie-based sessions are vulnerable to Cross Site Request Forgery. As suggested on Django's website, all GET requests in SSChat are safe. Unsafe requests use a `csrf_token`, which is a random value that is unknown by other websites and that is necessary to add to the request headers to make unsafe requests.

1.2.3 Database and SQL Queries

SSChat stores user data in a SQLite3 database. In a real deployment, a database management system such as PostgreSQL or MySQL to store data would be preferable. PostgreSQL also has the advantage to support encryption which can provide additional protection against database theft.

SQL injection attacks appear when the developers use raw SQL queries without using prepared statements. SSChat does not use raw SQL queries but restricts its usage to the Django Model API. According to Django's Security Page, all parameters are espaced using the Model API. This should make SSChat resistant to SQL attacks.

1.2.4 Checks

Django has a built-in check utility. This utility looks at the configuration of Django and displays warnings when some settings are configured in a way that is not secure for deployment. The results, visible in Figure 2, show that the only reported issues are linked to TLS (HTTPS) encryption. The API is indeed not configured to use TLS encryption, because it will only be accessed through a nginx reverse-proxy, which will add the TLS encryption.

1.3 React Client

A big threat concerning the development of web clients are injection attacks. Fortunately, React JSX protects against most types of injections by escaping any value embedded in the JSX. SSChat only handles user data in memory and outputs through the JSX format.

```
System check identified some issues:

WARNINGS:
?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS setting.
    If your entire site is served only over SSL, you may want to consider setting
    a value and enabling HTTP Strict Transport Security. Be sure to read the
    documentation first; enabling HSTS carelessly can cause serious, irreversible
    problems.
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless
    your site should be available over both SSL and non-SSL connections, you may
    want to either set this setting True or configure a load balancer or
    reverse-proxy server to redirect all connections to HTTPS.
```

Figure 2: The output of Django's `python manage.py check --deploy` command as configured in SSChat.

In order to encrypt messages, the OpenPGP.js library was used. This library is used and maintained by Proton Mail. Messages are stored in PGP-armoured ciphertext in the IndexedDB of the browser. Although both the local storage and the IndexedDB are accessible, retrieval of messages is not possible without the password of the user while metadata are still accessible.

1.4 End-to-end message encryption

Each user generates locally a PGP key pair when the account is created. The key pair is stored encrypted in the local storage of the browser. The public key is then sent to the server using an HTTPS POST request.

End-to-end message encryption is ensured by these PGP keys. When a user fetch its friends list, it also fetches its friends' public PGP keys, and also stores them in the local storage of the browser, allowing it to send them encrypted messages.

1.5 Security Code

The validity of the public key sent to the server at the account creation is paramount to the confidentiality of messages.

Users should not trust the central server, in case it is compromised. Users should be able to verify their public keys directly. While the entire armoured public key could have been shown, checking every single character would be tedious. Many users would not look at the keys they uses. By taking inspiration from Signal's Safety Numbers, safely codes have been introduced. The safety code of a user is the six first characters of the SHA256 hash of the public key, encoded in hexadecimal. The length of this hash should be adjusted according to the security threat model.

The safety number of a user is shown at all times besides his account name. When a user talks to someone else, the safety number of the correspondent is shown besides their name, on top of the conversation. Every user should verify the safety number of every contact using a out-of-band method of communication. When the public key of a contact changes, the user is notified and should re-verify the safety number. This change can be legitimate if the contact changes his public key willingly, for example if the previous key was lost. A change of key without modification by the contact can indicate that a server admin is trying to temper with the communication.

1.6 Deployment

The deployment procedure is as important for security as the development of the application itself. SSChat is made up of a React frontend and a Django + SQLite3 backend.

The React frontend is first compiled into a static website. These static files are served using the nginx web server. nginx has been configured to use TLS encryption using self-signed certificate. In a real deployment, a certificate signed by an authority such as Let's Encrypt is necessary. For the sake of simplicity, a self-signed certificate has been used.

The general recommendation for the deployment to production of a framework like Django (or Flask, etc.) is to launch the framework on a WSGI HTTP Server (SSChat uses gunicorn) which is designed to scale better than the build-in development server of Django (`python manage.py runserver`).

Then, nginx is configured to serve as a reverse-proxy on the `/api/` path. This allows to implement security features for both the static website and the API. For example, TLS encryption and the `X-Frame-Options SAMEORIGIN` headers are added to all requests. Logs can also be configured in nginx.

Configuring nginx for serving the static website, reverse-proxying the API and changing headers and its logging policy takes a few commands. SSChat is distributed with OCI Container build files to ease this process. It means that Docker, Podman or any OCI-compatible runtime can be used to build and run these containers. SSChat recommends using Podman over Docker as Podman allows to run rootless containers out of the box. It is also recommended to run these containers with a specific user account to prevent the containers to access data stored by the user. When building the OCI images, Podman will automatically use the lastest versions of nginx, Django and npm modules that are compatibles with the versions specified in the source code.

Please note that the current implementation has issues with logging: the IP addresses in logs contains a local IP address created by podman, and not the external IP address of the user.

Using OCI images is not mandatory. However, no installation instructions have been provided for a local installation.

2 Security Checkup

2.1 Do I properly ensure confidentiality?

Messages are stored in PGP-armoured ciphertext both on the server and in the client database (IndexedDB). The messages are only decrypted in-memory at the client side. The content of the messages are thus protected from server admins. Messages sent to a user are also deleted from the server immediately after being retrieved from the client of the user.

However, very little has been done to protect metadata. A server admin can see who messages whom, and when the message has been sent. This metadata is also available in the logs.

2.2 Do I properly ensure integrity of stored data?

Data integrity is provided at different levels.

All traffic between the server and the client is exchanged via the HTTPS protocol. If the key is not compromised, HTTPS guarantees (given that there are no issues in the

implementation and that the key is strong enough) that data was not corrupted, lost or altered by an attacker between the client and the server.

PGP-armoured messages and public keys could be manipulated by the server admin. However, the client interface will warn a user when the public key of a correspondent changes. The security code of the new public key is shown and the user must acknowledge the change to the new key. The user is expected to verify the validity of the security code through a out-of-band method of communication.

If a message was tampered with by an admin, two cases are possible:

- If the public key of the sender was not modified, the signature will not match (except if the rogue admin has the private key of the user), and the message will not be decrypted.
- If the public key was modified, the user will be notified and can verify it with the sender and will see that the public key is compromised.

2.3 Do I properly ensure non-repudiation?

Non repudiation is guaranteed using PGP public key cryptography. When a Alice sends a message to Bob, the message is signed with the private key of Alice, a key that only Alice has access to. If Bob has verified the security code of Alice, when he receives her signed message, he can be sure that Alice sent the message and no one else. Alice cannot pretend not to have sent the message, except if someone had access to her password and private key.

2.4 Do my security features rely on secrecy, beyond cryptographic keys and access codes?

Confidentiality, integrity and non-repudiation are guaranteed by PGP public key cryptography. The implementation used is free software, its source code is available on GitHub.

Authentication in SSChat uses the authentication system of Django. Django's source code is also available publicly on GitHub.

These security features do not rely on secrecy. However, the verification of the security code must be performed using an out-of-band communication.

2.5 Am I vulnerable to data remanence attacks?

Yes.

The Django API is written in Python3. Sensible information such as password are stored as Python `str`, which are immutable and thus vulnerable to remanence attacks. Django's `set_password` function accepts `str` or `bytes` objects. Both are immutable, so it is not possible to prevent remanence attacks with Django.

The only option to prevent remanence attacks on the API process is to re-write it using a framework which does not use immutable types for sensitive information. Two suggestions (but it have not been verified) would be to use a C++ framework called Drogon or a Rust framework called Actix. However, the authors of this report prioritised using a framework in a programming language they are used to (to reduce the risk of programming mistakes) and with a safe, built-in authentication system as there are more attacks on web servers on authentication systems rather than remanence attacks, which require access to the server.

Furthermore, on the client, there is also a possibility of remanence attack. Sensitive information such as the password of the user is stored in a JavaScript variable, which means that it is stored in memory and thus that a malicious actor could access it by dumping the heap of the browser JavaScript engine. Browsers such as Chrome allow users to dump the heap using its developer tools. Although it is technically doable, there are easier attacks.

Since it is a web application running in the browser, there is the possibility of using a malicious browser extension in order to obtain sensitive data. For example, an extension that is a keylogger, an extension that attempts to get the keys from the local storage, etc.

In order to mitigate the downsides of using a web application, while not completely avoiding data remanence issues, using software such as Electron (like Signal's desktop application) or Tauri to distribute the client would limit the user's ability to install extensions and would allow the distribution of a signed client.

2.6 Am I vulnerable to injection?

- **URL**: SSChat only uses three sub-pages: `/`, `/sign_up` and `/chat`. These URLs do not use parameters.
- **SQL**: Protection against SQL injection attacks is provided by Django. SSChat does not use raw SQL queries but restrict its usage to the Django Model API. According to Django's Security Page, all parameters are espaced using the Model API. This should make SSChat resistant to SQL attacks.
- **Cross-site scripting** (XSS): Protection against XSS is provided by React JSX. All user input is shown using JSX. Sending a message such as `<script>alert('xss');</script>` does **not** launch an alert prompt. The same applies for the username. Even though SSChat does make use of JSON bodies inside of POST requests, the header `Content-Type : application/json` is always set and thus there is no JavaScript execution.
- **Clickjacking**: The NGINX configuration adds the `X-Frame-Options SAMEORIGIN` header to all HTTP responses. Most browsers will prevent the page from being displayed in an iFrame in another web page. A browser could choose to ignore the header, but most users of the app will be protected from clickjacking attacks.
- **JSON injection**: The deserializer that we use is Python's default `json` module, which doesn't allow arbitrary code to be executed.

2.7 Am I vulnerable to fraudulent request forgery?

As explained before, all GET requests in SSChat are safe and unsafe requests such as POST requests are using a `csrf_token`. This protects the application against cross site request forgery (CSRF).

Moreover, SSChat is using TLS encryption and is as such protected against replay attacks.

2.8 Am I monitoring enough user activity so that I can detect malicious intents, or analyse an attack a posteriori?

Attacks on account login are protected by banning the IP of a user who failed to login three times in a row. There is no active monitoring of any other data.

nginx is configured to logs all requests. If nginx is run via Podman `podman`, then `podman logs name_of_container` can be used to access the logs. However, the logs are deleted when the container is shut down.

The issue with IP addresses in the nginx container is a big issue concerning monitoring user activity.

2.9 Am I using components with know vulnerabilities?

The implementation of PGP used by SSChat, called OpenPGP.js, was audited by Cure53 in 2014. Since then, all critical, high and medium issues found during the audit have been fixed. A lookup for CVEs for OpenPGP.js found that there are no CVEs at the moment.

CVEs for Django.

The SSChat frontend is a React app with many dependencies. These dependencies are bound to introduce security vulnerabilities sooner or later.

Output of **npm audit** on May 31, 2022:

```
# npm audit report

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check-
    https://github.com/advisories/GHSA-rp65-9cf3-cjxr
fix available via `npm audit fix --force`
Will install react-scripts@2.1.3, which is a breaking change
node_modules/svgo/node_modules/nth-check
  css-select  <=3.1.0
  Depends on vulnerable versions of nth-check
  node_modules/svgo/node_modules/css-select
    svgo  1.0.0 - 1.3.2
    Depends on vulnerable versions of css-select
    node_modules/svgo
      @svgr/plugin-svgo  <=5.5.0
      Depends on vulnerable versions of svgo
      node_modules/@svgr/plugin-svgo
        @svgr/webpack  4.0.0 - 5.5.0
        Depends on vulnerable versions of @svgr/plugin-svgo
        node_modules/@svgr/webpack
          react-scripts  >=2.1.4
          Depends on vulnerable versions of @svgr/webpack
          node_modules/react-scripts

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force
```

2.10 Is my system updated?

The system-update frequency will depend on the admins of the host of the web server. The suggested deployment uses OCI containers. OCI images are immutable, system updates cannot be applied to them. System admins will need to regularly re-build images in order to use up-to-date versions of the software. The images are based on `python:alpine` and `nginx:alpine` which are regularly updated. When building new images, `npm install` will download an up-to-date compatible versions of libraries. A monitoring solution such as GitHub dependabot should be installed on the repository to monitor updates and security issues on libraries used for this project.

As an alternative to using containers, a system admin can also install SSChat by setting up a reverse-proxy, building the React website using `npm install && npm build` and launching the Django api with `gunicorn`. The system admin then has the responsibility to patch the web server, npm packages, python and Django.

2.11 Is my access control broken (cf. OWASP 10)?

All API requests that send back user information actively check that the user are in a authenticated session. The user never specifies who he is in a request, it is always reduces from the session. This prevents authenticated user to impersonate someone else. Rate limiting is implemented for the API at the reverse-proxy level: only 10 requests are allowed per second per IP, this reduces the impact of automated attacks.

However, there is an issue with CORS configuration. The deployment targets localhost:8080 and not a domain name. The CORS configuation must also accept the localhost "domain". This is a vulnerability because anyone can create a website with an address in localhost. The fix would be to deploy in a server with a public IP address and use a domain name, with a TLS certificate.

2.12 Is my authentication broken (cf. OWASP 10)?

Django does flush the session id after a successful login or logout and does not add the session id to the URL. Brute force logins are slowed down because IPs are banned after 3 failed logins in a row. Passwords are stored salted and hashed in the data using the PBKDF2 algorithm and transmitted over HTTPS.

However, the authentication procedure has several flaws:

- There is no two-factor authentication.
- A user cannot change his password.
- Weak password are allowed, a check for common password should should be implemented.
- There is no lost-password procedure.

2.13 Are my general security features misconfigured (cf. OWASP 10)?

Tests of Django and nginx configurations should be intgrated into a CI/CD pipeline such as GitLab CI/CD.

At the time of shipping this software, as can be seen in Figure 2, the misconfiguration reported auto-analysis tool included in Django are linked to the lack of TLS encryption. This makes sense because the API is behind a reverse proxy. The `SECRET_KEY` has to be generated and stored in the environment variables, and no admin account exists by default because the build process flushes the database before creating the image. The Django `DEBUG` flag is set to False, this limits the amount of information available to user when issues arise.

Using OCI containers allows to expose only the necessary ports to the outside world. In the current configuration, only the nginx server is accessible to the outside world and only through the port 8080.

2.14 Tests

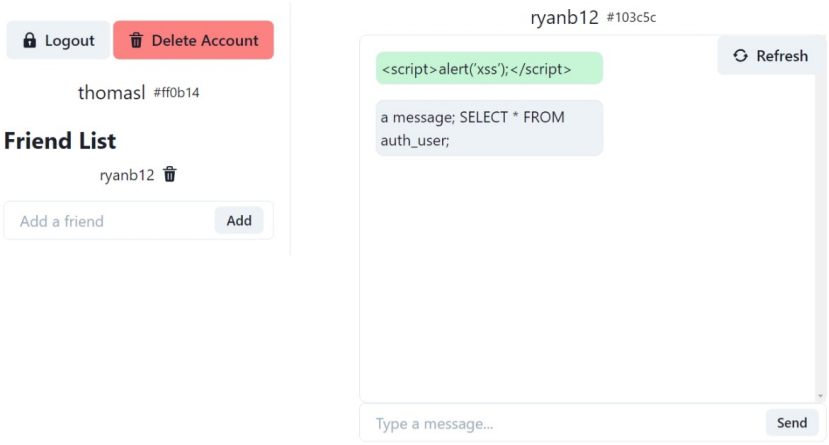

Figure 3: SQL injection and XSS test.

Some tests were done in order to see if SSChat was vulnerable to XSS and to SQL injection. The tests were done through the chat input as well as by modifying the recipient's username on the POST requests. See figure 3.

A Licence notice

SSChat makes use of libraries with the following licences:

A.1 Deployment Tools

- **podman**: Apache License 2.0
- **podman-compose**: GNU General Public License v2.0

A.2 Backend

- **python3**: PSF LICENSE AGREEMENT
- **Django**: BSD 3-Clause "New" or "Revised" License
- **django-cors-headers**: MIT
- **django-axes**: MIT
- **gunicorn**: MIT

A.3 Frontend

- **React**: MIT
- **npm CLI**: The Artistic License 2.0
- **Node.js**: Multiple Licences, including MIT.
- **OpenPGP.js**: GNU Lesser General Public License v3.0

Output of **npx license-checker –summary**:

```
MIT: 1045
ISC: 57
CC0-1.0: 38
BSD-2-Clause: 36
Apache-2.0: 27
BSD-3-Clause: 27
(MIT OR CC0-1.0): 3
Unlicense: 2
0BSD: 2
Python-2.0: 1
MPL-2.0: 1
CC-BY-4.0: 1
(Apache-2.0 OR MPL-1.1): 1
(AFL-2.1 OR BSD-3-Clause): 1
ODC-By-1.0: 1
(BSD-3-Clause OR GPL-2.0): 1
LGPL-3.0+: 1
Custom: https://github.com/facebook/create-react-app: 1
```

PS: The licence of create-react-app is MIT.