PeerSet Security Assessment

Phoenix Boisnier – December 9, 2021

Table of Contents:

Introduction

This document serves as a brief rundown of the security analysis performed for the PeerSet Application, developed by the SUNY Oswego Software Development Team of Fall 2021. This analysis is not fully comprehensive, and should not be intended to serve as an exhaustive list. What this

document should be used for is to bring attention to the administrator of the software on potential issues that exist upon deployment, and contain a suggested course of action for fixing the listed issues.

Methodology

- *Asset Identification*

The vulnerability assessment seeks to identify key assets of the PeerSet software that are critical to its ability to deliver the service without error and perform as expected under daily use circumstances. The assets presented in this security assessment are not necessarily exhaustive, but present a list of key points the development team has identified as critical infrastructure.

- *Threat Identification and Risk Assessment*

Identifying potential threats to the PeerSet service helps the development team and end users find and react to potential future issues with the software in the event that a malicious actor attempts to prevent access to, steal data from, or modify data within the system. Once again, this list is not exhaustive, and further testing may reveal further potential issues. Additionally, as the software PeerSet uses to operate is updated, completely new threats may emerge.

Upon identifying a list of potential threats, each threat was analyzed for its potential impact on PeerSet, as well as the likelihood of said threat becoming an issue. As time was limited, this assessment is primarily theoretical in nature, and a full test may be warranted if security is a concern for the end user.

- *Risk Mitigation*

Having identified threats, certain steps can be taken to lower the likelihood of those threats becoming problematic for PeerSet. This may include steps at any level of operation, from hardware choices to policy decisions for end users.

- *Existing Security*

Lastly, a brief overview of existing security features will be included, along with an explanation of why they are included / what these features do.

Assessment

- *Assets*

PeerSet uses the Model View Control (MVC) design paradigm, and relies on the following components to operate as expected: the front-end Graphical User Interface (GUI) service (The View), four back-end services that provide Application Programming Interface (API) endpoints for the GUI to interact with to access data and perform functions on the data (The Control), one utility folder containing supporting Java classes and objects, a MySQL database service to store and retrieve data (The Model), as well as one of the SUNY Oswego Computer Science Department servers.

The GUI as well as the four back-end services are all written using the micro-service architecture and deployed on Open Liberty software. Each of these services may independently fail at any given point, which helps prevent full service failure.

The Model is available to any authorized user on the SUNY Oswego CS server with appropriate credentials, and can be accessed remotely. The MySQL server is administered by a member of the SUNY Oswego CS department staff.

The hardware is located on the SUNY Oswego campus and is only physically accessible to approved members of the CS department. The server is accessible remotely via SSH, and all students of the CS department have login credentials to access the server via SSH.

- *Threats and Risks*

The list of potential threats is an ever changing list, but can be broadly summarized into three categories: data leakage, data corruption, and service disruption.

1: With respect to the services themselves, GUI and the back-end services, three of these categories apply. The final version of PeerSet at the time of this document uses HTTP requests to communicate between the individual micro-services. The data transmitted is in plain text, which means that if the data were intercepted by a *man-in-the-middle*$_1$, that data would be visible. Further, the data being transmitted is formatted as Java Script Object Notation (JSON) text. As JSON text is often easy to read, this data could be modified by an attacker. Though the HTTP requests rely on JSON Web Tokens (JWTs) to provide an authentication and authorization method, the tokens themselves can also be intercepted and used in a *replay attack*$_2$ scenario. The JWTs cannot be edited without being detected, however, which can prevent *privilege escalation*$_3$. More on this will be covered in the “Current Technology” section on page 4. Lastly, service interruption may occur if something occurs to crash or stop a given service.

The likelihood of a man-in-the-middle attack or a replay attack is unknown, but would not be too technically difficult to pull off. Privilege escalation from unauthenticated user to authenticated user is possible if an attacker gained access to a token, but should not allow privilege escalation from that point further.

2: With respect to the Model portion, once again, all three categories apply. The current MySQL database is accessible remotely with knowledge of the URL, associated username, and password. Additionally, the current database uses a weak password, and the account does not have authorization to modify the password. A *brute-force*$_4$ attack would likely reveal the password in a short period of time should an attacker be able to identify the appropriate URL and username. This could lead to all three of data leakage, data corruption, or service disruption. The SUNY Oswego database server has been targeted in the past, successfully, and so this issue should be taken seriously.

The likelihood of the database being compromised is unknown. Given that this has occurred in the past, it is clearly not impossible. This would require an attacker to successfully guess or gain access to not only the URL, but the username and password as well. Given that these items may be easy to guess, the risk for this occurring may be one of the greater threats to PeerSet’s functionality.

3: With respect to the hardware, again, all the three issues are a concern. Data leakage due to physical access to the system is unlikely, as the server exists in a locked room in a building on the SUNY Oswego campus. The server is accessible via SSH, though, and accounts could potentially have access to the account on which the PeerSet server runs. Given that the software is open source, this is less of a concern, but it does open the avenue for modifying the configuration file, or exposing any of the clear text passwords stored in the file itself. Further, an attacker could modify the configuration file to prevent key services from functioning, or delete necessary files, as well as start and shutdown the service or add and remove authorized professor accounts.

The likelihood of the server being compromised via SSH is unknown; while many students have weak, default passwords, the chances of finding the necessary account may not be likely. The server does experience intermittent outages, but this is the case for any server. It should be noted that the server does exist in an area that is prone to severe winter weather, so power outages and downed wires may be more likely in the winter months.

- *Mitigation*

In order to account for the above risks, the following actions are recommended.

1: To prevent service outages due to faulty software, further bug testing and stress testing is recommended. To prevent data leakage while in transmission, HTTPS should be implemented, and a certificate should be signed by a major vendor. For extra security, or in lieu of HTTPS, integrating an encryption mechanism between the services could be used for the JSON payloads, or even the JWTs themselves. To prevent service disruption, man-in-the-middle attacks, or replay attacks, modification to the “authObject.java” file is recommended. This file’s purpose is to authorize owners of a particular JWT for a given endpoint. Because this file’s function is handled by PeerSet, and not Open Liberty’s built in authorization, custom programming can be performed to include extra checks for a token including checking the metadata, identifying and revoking suspicious tokens, or any additional number of functions.

2: In order to prevent unauthorized access to the database, a less easily guessed URL, in combination with a less easily guessed username and a password that uses more than 8 characters, is alpha-numeric, contains at least one capital letter, contains at least one special character, and does not only use the capital letter at the front of the password or use the numbers and special characters at the end would reasonably secure the database. Another method for reducing the likelihood of data leakage would be to encrypt all of the data in the database, though this could slow down operation time.

3: For the server, options are limited. The end user may choose to deploy PeerSet on a different server, but this option will come with a new set of threats and mitigation strategies. In order to prevent unauthorized access from within the CS server at SUNY Oswego, the administrator of the PeerSet software could secure access to their personal account using any of the operating system specific commands, or may be able to negotiate with the server administrator to develop a specific security solution.

- *Current Technology*

PeerSet currently uses multiple technologies in order to provide security for the users.

1: Open Liberty and the micro-service architecture helps prevent full system failure should one or more of the services stop functioning. A partial failure can allow a subset of users to perform a

subset of actions as opposed to no users being able to perform any actions. PeerSet also OAuth2.0 to authenticate Gmail accounts and generate JWTs for authenticated accounts. These JWTs are used for authorization on particular endpoints within the API, and are signed, which means that modification after generation will invalidate them, thus ensuring that an attacker cannot change the payload of the JWT. The JWT is checked at both the front-end and back-end. Additionally, the code that interfaces with the database makes use of prepared statements, which was briefly tested for *SQL injection*[5] and was determined to be safe.

2: The database is accessed via login credentials. As mentioned above, user input is sanitized by use of prepared statements, so SQL injection should not be possible. Personally identifiable information on students within the student table is encrypted.

3: The server used is maintained by the SUNY Oswego Computer Science department. To find out what security policies they use, contact information can be found on the SUNY Oswego web page.

Conclusions

In its current state, PeerSet offers a tolerable level of security for its users. There are several identified risks associated with the software, and while these risks do not pose a foreseeable usability risk, a determined attacker could disrupt the operation, modify, or steal the data of PeerSet. It is recommended that further encryption be used in environments where data breeches are a concern, and that data in transit be encrypted.

Glossary of Terms

1 – Man-in-the-Middle: This is a type of attack where a bad actor situates themselves in a way such that they are able to view, modify, or interrupt data in transmission from its origin to its destination.

2 – Replay Attack: This is a type of attack where a bad actor has intercepted, saved, and replayed data in transmission in order to replicate or imitate a valid request.

3 – Privilege Escalation: This is when a bad actor is able to gain more access to a system than they should be able to. In the case of PeerSet, this would include an unauthorized user being able to act as a student or professor, or a student being able to act as a professor.

4 – Brute Force: This is a type of attack where a bad actor uses the speed of a computer to systematically guess many passwords in a short period of type. Typically, this type of attack is only effective on short (8 characters or less) passwords that do not contain special characters, as a brute force attack guesses every combination of characters up to a certain length.

5 – SQL Injection: This is a type of attack where a bad actor uses a specially crafted input in order to access a database. This kind of attack is possible because the input subverts portions of the code in order to perform functionality that is normally restricted.