TeamJoshButNotJosh's Guide to Abusing Abuse Cases

In order to create a ticketing system secure from malicious users, TeamJoshButNotJosh is integrating countermeasures to protect the integrity of our users' personal information and the security of our server.

Safeguarding a user's information such as their payment details is imperative to maintaining a healthy relationship with the client. To counteract malicious users attempting to obtain the aforementioned data Spring Boot, Stripe's API, and encrypted data will be utilized to thwart attacks.

Spring Boot will be used to validate a user's transactional information prior to sending the information to Stripe's API. A malicious user may attempt to intercept and alter web requests sent from a client to the server. Spring Boot will be configured to force HTTPS by redirecting HTTP traffic by using Spring Boot's requiresChannel() method. Allowing for a smooth transition between servers for the client. Stripe's API, tested and tried built-in security features will be integrated into our software to counteract any malicious user intercepting a user's payment credentials.

As mentioned, validating web requests from the client to the server is imperative to maintaining security while transitioning from server to server. Without solid security a user may be able to manipulate the price of tickets in their cart. This would be very bad for business. So, we must ensure proper payments are sent. Requiring we integrate server-side validation, Spring security, and Spring Boot.

Spring Boot's "@Valid" annotation provides server-side validation for our system by verifying any incoming request payloads. A session's security will be backed by using Spring security. At the end of a session, the session's ID will be stored in our database by using

PostgreSQL. As with the previous abuse case, configuring Spring Boot to force HTTPS by redirecting HTTP traffic will be done. Ensuring server security in general is crucial to preventing prospective attacks that we may or may not consider.

Technical Implications

1. The price manipulation abuse causes the need for more space in our database to ensure session IDs are stored appropriately. All of the countermeasures made will be put into their own methods in the ticket class for the price manipulation abuse case. By securely storing and accessing user's session IDs, we will disallow bad actors from performing acts on their behalf.

2. For the Payment Infiltration technical implications, by implementing secure, proven 3rd party APIs with built-in security, we will be able to achieve a level of payment security unattainable given our resource limitations.

Functional Implications

To counteract the payment infiltration abuse case, users will be required to fill a payment form by manually inputting their credentials. Lastly, they will have to click a "Pay with *****" button to confirm the transaction. This functional implication is minimal, adding a substantial level of security for the client, yet barely changing the client's experience

The beginning to end user experience will not be affected by the price manipulation abuse case. All of the validation will be done internally, server side. Providing the user with an effortless experience and us with the exact amount of money they owe.