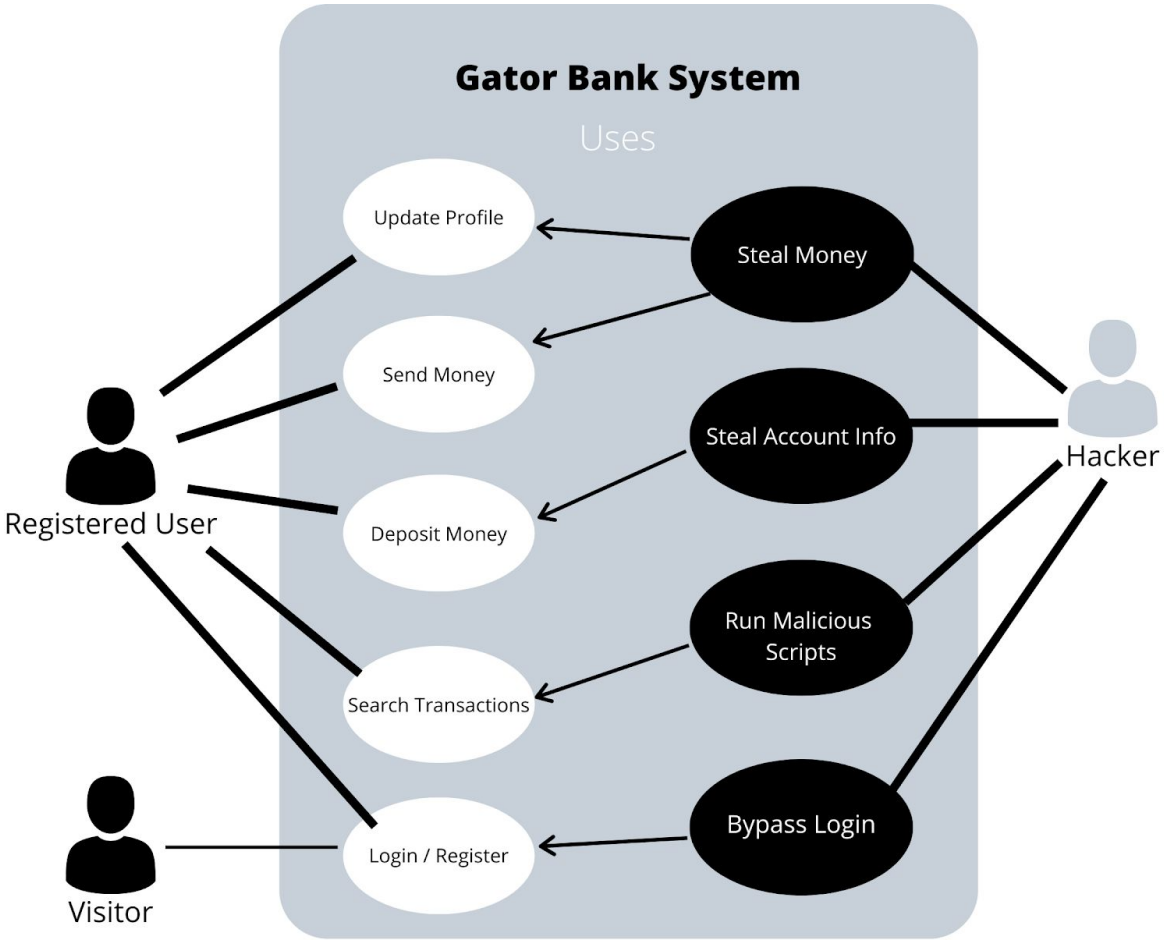
Use / Abuse Cases



Create Use / Abuse Case diagram showing functionality and potential threats. Provide a description of the usage scenarios, potential exploits, risks, and potential for security policy violations with the abuse cases. Your objective is to
focus on security policy that targets to the use/abuse cases specified and not the overall domain of the application if the context is not needed to understand the impact of the exploit.

Usage Scenarios:

- A *visitor* to the Gator Bank website can Register for an account by creating a username and password.
- Once registered, a *user* can then:
    - Log in using their previously set credentials.
    - View their dashboard containing their user info and other details.
    - Search through their transactions on their Gator Bank account.
    - Deposit money into their account.
    - Send money to other users.

- Update their profile picture by submitting a URL link to a photo.

Abuse Cases:

- A *hacker* can bypass both the need to register for an account and log in by using a *URL manipulation* tactic to navigate to Gator Bank home page directly.
- A *hacker* can use a *SQL injection* attack tactic to abuse the transaction search functionality presented on the website. Namely, injecting specific SQL that would allow them to view all transactions, even those that do not pertain to them.
- A *hacker* can implement a *XSS* attack by again using the search transaction functionality on the Gator Bank dashboard and inserting a JavaScript-based query. This search "term" is reiterated unto the user after the results are found and thus presents an opportunity for malicious scripts to be run.
- A *hacker* can steal money from another user by implementing a *CSRF attack* through the profile picture feature. If they set their profile picture URL to instead be a request to the site's "send money" route, going from the victim to the hacker, they can utilize the logged in status of the victim to complete that request whenever their profile is viewed.

Gator Bank: Mini-Project Report

Our group was tasked with creating a two-fold website, one part subjectable to a selection of the various security breaches we have discussed in lecture, the other fortified to the extent that it is no longer subjectable.

How to initiate exploits

- *SQL Injection*
    - Once on the Gator Bank home page, a user can view and search through their transactions. This allows for a possible SQL injection with the input given to the search functionality. A malicious user can enter a search query such as ‘ ” OR 1=1 ##’ and retrieve all transaction info for all users, not just themselves.
- *IDOR/URL Manipulation*
    - A malicious user can bypass the login and registration requirements of the Gator Bank site by navigating to the URL ‘…/home-unsafe’. This path allows unauthenticated access to the Gator Bank home page, wherein the user can perform other malicious attacks such as the aforementioned SQL Injection.
- *CSRF/Session Attacks*
    - A user can use the profile image feature of the site to set a malicious request URL as the source for their profile image. When another user views the malicious user’s profile image, it will send a request to the send money functionality of the site, utilizing the user’s logged in status to successfully complete the attack.
- *XSS*
    - Like the SQL Injection attack previously mentioned, once on the Gator Bank home page, a user can again view and search through their transactions. This allows for a possible XSS attack with the input given to the search functionality. When a term is searched for, the term is reiterated to the user upon the showing of the search results. This input can be formed in a way that allows for malicious JavaScript code to be run.

Types of vulnerabilities

The types of vulnerabilities in our site are as follows:

- SQL injections
- Missing authentication
- Cross-site scripting and forgery
- Unsanitized user input
- Malicious Image tag source URLs.

Mitigations

To address the aforementioned vulnerabilities of our system, we implemented the following mitigations.

- Input sanitization. Input given by the user is sanitized to mitigate the effects of both an SQL injection and an XSS attack. For the SQL injection, specific terms/characters are removed from user input, if present, prior to executing the query. This includes characters such as ';', '##', etc. In a similar manner for the XSS attack, and html tags were removed from the user query after submission.
- Session cookies were used to mitigate a malicious user's ability to visit restricted areas of the site while unauthenticated. If a request is made to the home page of the site and there is no cookie demonstrating previous authentication, the user is instead given an error and the option to go back to the login page.
- Request validation was used to stop a malicious user's ability to make requests on behalf of another user through the site's profile picture feature.

Lastly, briefly discuss the difference between password hashing & encryption, as well as what salting and salt + peppering passwords enable (compare/contrast)

The main difference between hashing and encryption is that hashing is a one-way function whereas encryption is two-way. Meaning that an encrypted password can then be decrypted with the right knowledge. This is not however possible with a hashed password.

A salt is a random value often stored within a database with the hashed content, and thus means that each password must be brute forced individually. With both the password and the hash stored in the database however, this means that a compromised database **loses both**. A pepper is an unchanging value used all throughout a site, not stored in a database but instead within the sites source code. This is done to avoid the database compromise issue that a salt suffers from.