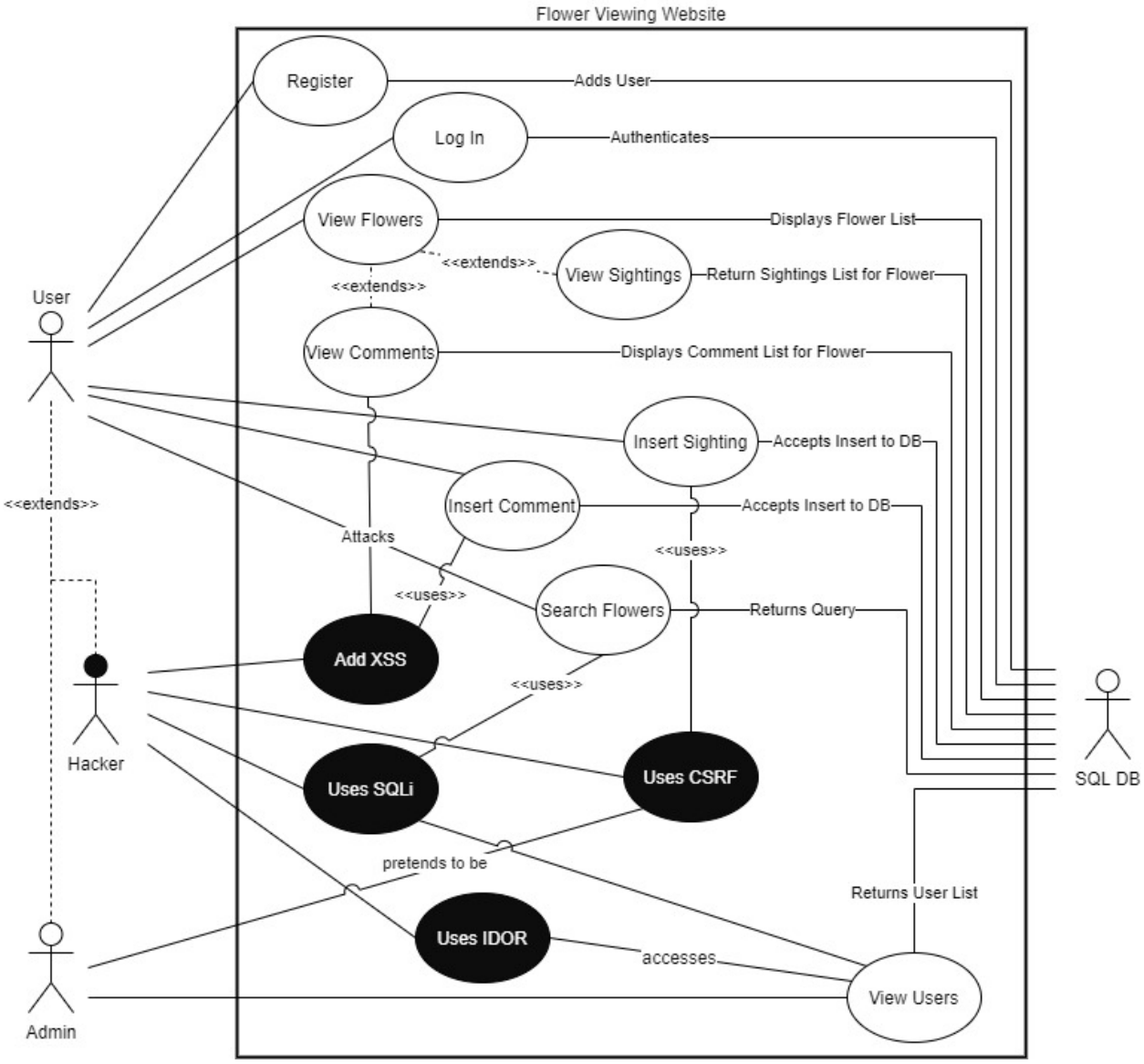
Kevin Thomas, Mustafa Mohamed, Nicholas Tateo, Pedro Sicilia

Mini-Project 1 – Web Exploits Use/Abuse Cases & Report

Use/Abuse Cases:



A user can register an account on the website and login using their credentials. This allows them to view all the flowers, add a new flower sighting, or search for a flower. Viewing all the flowers also allows a user to see the sightings of the respective flower as well as add and view the comments made under that flower. The vulnerable version of the site allows a hacker to add XSS to the comment field which can be executed when clicked on the links within the viewed comments. This exposes both users and admins to the risk of running malicious code from the comments. An admin user can navigate to the `/admin` page in order to view all the users. There is the risk that a hacker can utilize an IDOR exploit to navigate to this page, violating the policy that only admins can see user information. The search function allows users

to search for flowers in the database, but poses the risk that a hacker can use a SQL injection to view all the users from the database. This violates the same security policy as before, allowing user information to leak to non-admins. Users can insert sightings of new flowers, allowing the sighting information to be viewed from the flower, but this form exposes the risk of a CSRF attack allowing a hacker to pretend to be another user or even an admin on the post of the information. This carries the risk of allowing users to feign their actions as those of another user or even those of an admin and in coordination with other exploits could allow the insertion of malicious content to appear as though it came from an admin.

Task 4: 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.
Task 6: Report (max 1.5 pages): Discuss how to initiate exploits, type of vulnerabilities, how the mitigations were applied, difference between password hashing & encrypting password and what salting and salt + peppering passwords enable (compare / contrast).

Report:

There are four types of vulnerabilities in the insecure version of the site. How t initiate each exploit and how they were mitigated is as follows:

XSS

To initiate this exploit, a payload such as `<a onclick=alert(1)>xxs link</a>` is inserted in the comment field or `javascript:alert(1)` in the link field when adding a comment to a flower. This allows the abuser to run javascript code, like an alert when regular users click the comment or link. The exploit in the comment field is mitigated by not using innerHTML, which allows React's built in XSS protection to sanitize the input. The link field is checked to make sure it is a valid link and the input is sanitized, ensuring both fields of a comment don't allow XSS.

IDOR

This exploit allows a regular user to access the full list of the site's usernames which would otherwise only be accessible by the site's admin user. To initiate this exploit from the dashboard, replace `/dashboard` in the url to `/admin` (ex: http://minip1-vulnerable-272103.appspot.com/admin). This attack was mitigated by adding a check in the admin dashboard page to see if the user is actually an admin.

CSRF

This exploit allows a user to falsify their session and pretend to be another user. This allows them to add sightings and comments as a username they do not have access to. To initiate this exploit, change the locally saved token to the desired username. This attack

was mitigated by creating a JSON web token that has a header, payload/body, and signature section. The payload has user information and expiration date. The signature section is a HMACSHA256 hash of the header, payload and a secret key. This makes it extremely hard for a regular user to replicate the token without the secret key.

SQL Injection

This exploit allows for a regular user searching for a flower to inject SQL to display the full list of users. To initiate this exploit, inject the SQL code to the search bar in Flower Search (ex: `Lovage" UNION SELECT password FROM Users; --`). This attack was mitigated by using prepared statements for SQL queries to the database which sanitizes all user inputs as a string, preventing malicious SQL execution.

Hashing functions are slow, one-way cryptographic functions; they are not reversible, making it hard for an attacker to recover plaintext from the ciphertext. Hashing functions are many-to-one, meaning that collisions are often possible and there may be multiple possible inputs that correspond to the same hash output. Because hashes are slow to compute, they increase the security of passwords by making it more difficult to carry out a brute-force attack by guessing inputs.

Encryption functions are two-way and usually rely on a key/keystream or some sequence of cryptographic primitives to recover the plaintext from the ciphertext or vice versa. They mitigate against the possibility of an attacker recovering sensitive data in transmission or storage as long as the attacker cannot access the key. Encryption functions are one-to-one mappings, and this provides the important benefit to the user of recovering the plaintext, whereas hashing functions are more suitable when only output comparison is necessary.

Salts are constants which are stored in the database alongside hashed passwords and are randomly generated per new or changed password; peppers are constants stored in the application. Each is applied to the password to increase the strength of the hash by increasing the input space. In applications where the application code is separate from the database, use of both a salt and pepper is preferred because an attacker that compromises the application code will still not know the salt and an attacker that compromises the database would still not know the pepper.