Reset Password

Aaron Sierra

Verdict:

Revisto.live needs to have a required CAPTCHA following the “forgot password” button. Once the following information has successfully been filled out, have a URL token link sent via email for a password reset that follows standard security practices *as seen below.* Revisto then sends an email with a password reset link that expires in <30 mins. *(preferably 10 mins).* The user should only be able to reset their password after clicking on the link in the email. The email link should be generated using a cryto safe algo. Once the user inputs a new password *twice* the site should then log them out and terminate any session involving that specific user. Revisto then needs to send a confirmation email, confirming the password has been changed.

💻 My reasons for the verdict are below:

Source for all information: OWASP

https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html#url-tokens

Who are they and why are they reliable?

OWASP - Open web application security project

They are:

- nonprofit
- founded in 2001
- goal of helping website owners and security experts
- goal of preventing cyber attacks
- I have heard of them before googling their info

What does the OWASP recommend?

1. Return a consistent message for both existent and non-existent accounts

   The reasoning behind this: if a hacker does not have a specific target, we should not be the ones to give him one.

   For example: Suppose a hacker tries to reset the password for an account called “bob”. If the hacker receives a message “bob does an exist” he is going to move on. then if hacker repeats for “jim” and receives “a password reset has been sent”. The hacker now has a place to start and has more information that they had before.

   Do not give hackers more information then they need.

2. Ensure the time taken for user response message is uniform

   Granting longer than a 10 min window can open your system and the user to potentially dangerous vulnerabilities such as a brute force attack

3. URL tokens are the fastest and most simply method

   I now believe we should do the URL thing in the email

4. Do not make a change to the account until a valid token is received

   The hacker (or even just some troll) can target a user and give them a not fun experience

   For example: Steve does not like user “elonmusk12” and would like to troll them. Steve goes and submits a “I forgot my password” or even “lock my account” for a user by the name of “elonmusk12”. elonmusk12 is now the victim of trolling and is displeased by the fact that they have to re-enable their account for a request they did not submit

   Ensure the proper user is making the proper requests

The Request

The URL request should return in a consistent amount of time to prevent attackers enumerating *(discovering)* which accounts exits.

You can secure your Request page by using:

- **asynchronous calls**
- **randomly generated tokens**
    - For this use: Cryptographically Safe Algorithms
    - information about ^
        - These are designed to produce higher qualitiy of randomness making them safe
        - Note: they are slower and more cpu intensive and can possibly end up blocking when large amounts of random data are requested. (side note: its very unlikely that a mass amount of users on the platform are going to request a password reset all within the same 10 min time frame)
    - How to use crypto safe algos? super simple
        - Do not know which the website uses, so I threw three guesses in here.
        - Java use following code:

    ```
    java.security.secureRandom
    ```

        - Python:

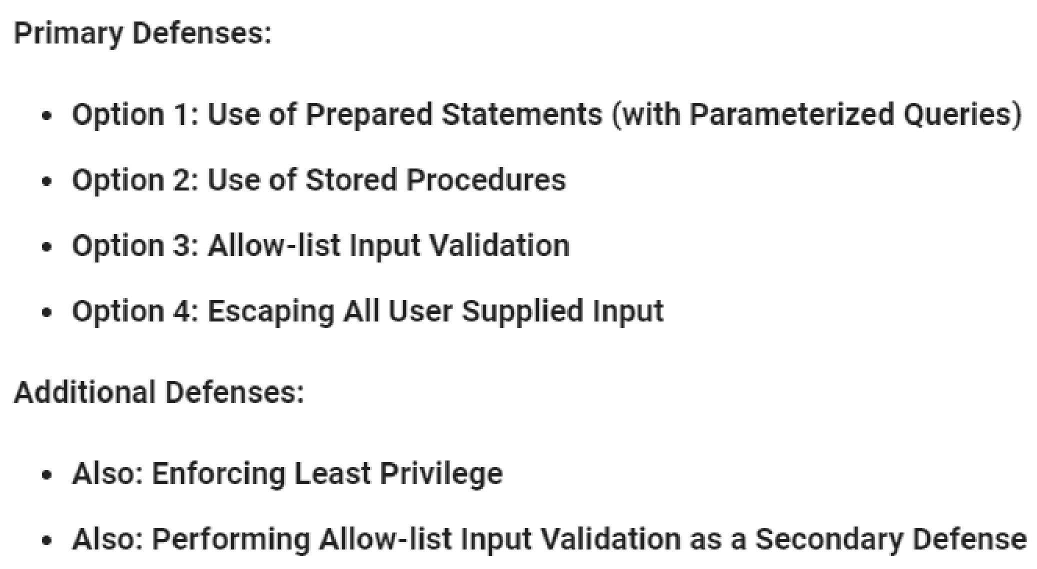
    ```
    secrets()
    ```

        - Go:

    ```
    crypto.rand package
    ```

- **CAPTCHAS**
    - you know what this is
- **sql injection prevention methods**
    - SQL injection is where an attack inserts malicious code into a login or anything they can type into

- SQL Injection flaws are introduced when software developers create dynamic database queries that include user supplied input. To avoid SQL injection flaws is simple. Developers need to either: a) stop writing dynamic queries; and/or b) prevent user supplied input which contains malicious SQL from affecting the logic of the executed query.
- DEFENSE



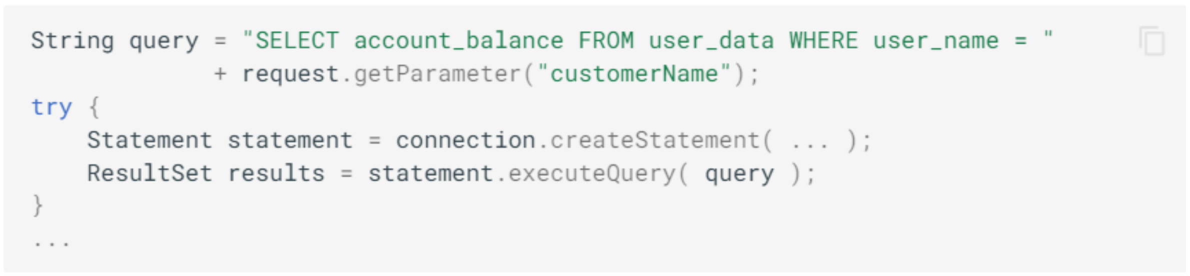
Primary Defenses:

- **Option 1: Use of Prepared Statements (with Parameterized Queries)**
- **Option 2: Use of Stored Procedures**
- **Option 3: Allow-list Input Validation**
- **Option 4: Escaping All User Supplied Input**

Additional Defenses:

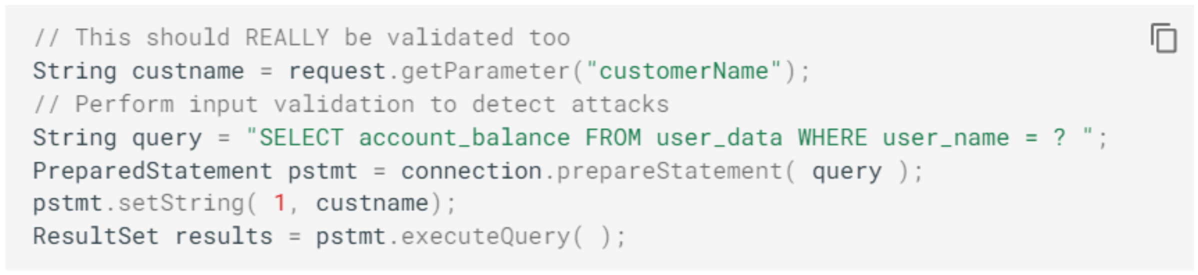
- **Also: Enforcing Least Privilege**
- **Also: Performing Allow-list Input Validation as a Secondary Defense**

- UNSAFE EXAMPLE BELOW



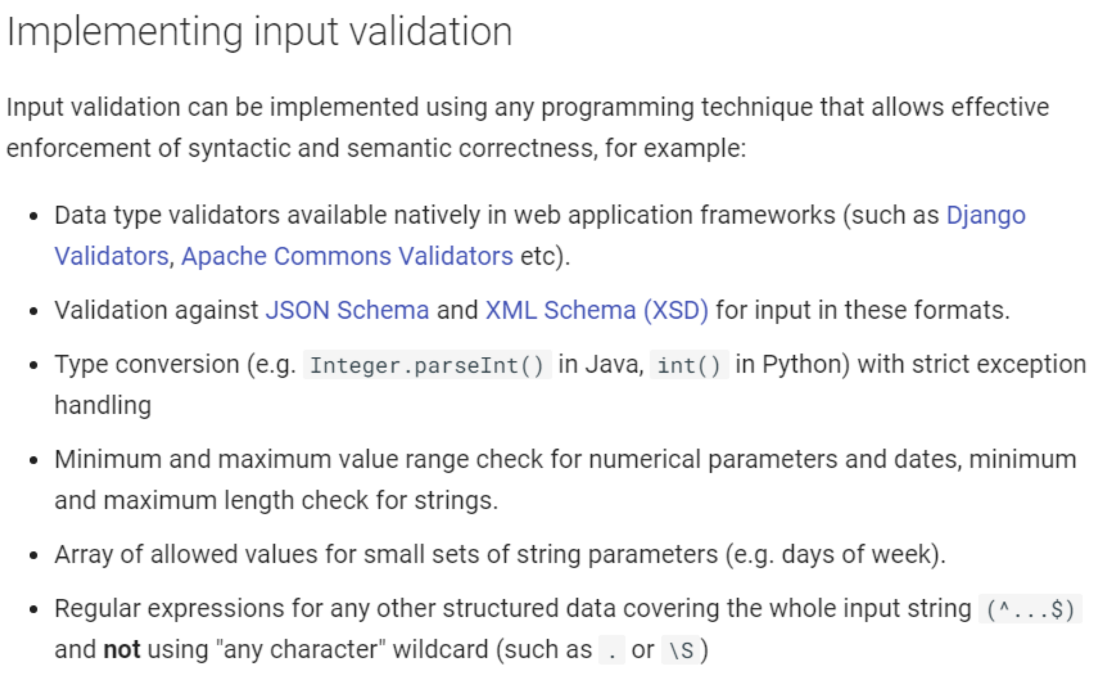
```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
             + request.getParameter("customerName");
try {
    Statement statement = connection.createStatement( ... );
    ResultSet results = statement.executeQuery( query );
}
...
```

- DECENTLY SAFE EXAMPLE BELOW



```java
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

- **input validation**
  - Using input validation can stop attackers before they are able to take a first step toward malicious behavior
    - Input validation should be applied on both **syntactical** and **Semantic** level.
    - **Syntactic** validation should enforce correct syntax of structured fields (e.g. SSN, date, currency symbol).
    - **Semantic** validation should enforce correctness of their *values* in the specific business context (e.g. start date is before end date, price is within expected range).
  - How to implement it:



Implementing input validation

Input validation can be implemented using any programming technique that allows effective enforcement of syntactic and semantic correctness, for example:

- Data type validators available natively in web application frameworks (such as Django Validators, Apache Commons Validators etc).
- Validation against JSON Schema and XML Schema (XSD) for input in these formats.
- Type conversion (e.g. `Integer.parseInt()` in Java, `int()` in Python) with strict exception handling
- Minimum and maximum value range check for numerical parameters and dates, minimum and maximum length check for strings.
- Array of allowed values for small sets of string parameters (e.g. days of week).
- Regular expressions for any other structured data covering the whole input string `(^...$)` and **not** using "any character" wildcard (such as `.` or `\S`)

- **Referrer policy tag with the -noreferrer value**
  - this avoids referrer leakage
  - **info on referrer leakage**: The HTTP Referrer header is used to store the URL of the page from which the user is coming from. Confidential information about the user may be leaked if it is stored in query parameters used by the application.

Successful Authentication

Once the user proves they are who they say they are, they should reset their password to a new secure one. In order to secure this you need:

- confirm password by writing it twice
  - so they do not mistype
- password policy (must be x length and contain x characters)
  - if they make a bad password, we should not have let them reset it in the first place imo
- update and store password following secure practices
  - make sure you HASH the passwords *(hashes are irreversible)* and do not encrypt them.
  - To see if a password matches the other password using irreversible hash:
    1. store their password using hash *pretend this is the hash* (F^i#n32xx%)
    2. ask the user what the password is: *they enter* bob123
    3. hash bob123 *pretend that bob123 hashed is* (B&$3j&f)
    4. compare the hashes
    5. if no match, no entry
- send user an email informing them their password has been reset (DO NOT send password in the email)
  - send passwd in email is just bad practice,
  - introduces vulnerabilities

    - possible Wireshark target
    - etc.
- Do NOT log them in.
    - Logging them in adds complexity to the authentication session handling code and increases likelihood of introducing vulnerabilities
- invalidate other sessions
    - kill their other sessions, this is good practice.