Remediation Advice

Even during bug bounty hunting, it is essential to share remediation advice. Knowing how to remediate a vulnerability does help not only the client but also ourselves since we may quickly suspect the existence of a vulnerability by noticing the lack of a related security measure.

So dedicate some time to studying how the vulnerabilities you are aware of can be remediated, as this can make you even more competitive.

Remediating Session Hijacking

It is pretty challenging to counter session hijacking since a valid session identifier grants access to an application by design. User session monitoring/anomaly detection solutions can detect session hijacking. It is a safer bet to counter session hijacking by trying to eliminate all vulnerabilities covered in this module.

Remediating Session Fixation

Ideally, session fixation can be remediated by generating a new session identifier upon an authenticated operation. Simply invalidating any pre-login session identifier and generating a new one post-login should be enough.

As already mentioned, the established programming technologies contain built-in functions and utilize libraries for session management purposes. There is no need for custom implementations to remediate session fixation. Find some examples below:

PHP

Code: php

```
session_regenerate_id(bool $delete_old_session = false): bool
```

The above updates the current session identifier with a newly generated one. The current session information is kept. Please refer to the following resource for more in-depth details: session_regenerate_id

Java

Code: java

```
...
session.invalidate();
session = request.getSession(true);
...
```

The above invalidates the current session and gets a new session from the request object.

Please refer to the following resource for more in-depth details: Using Sessions

.NET

Code: asp

```
...
Session.Abandon();
...
```

For session invalidation purposes, the .NET framework utilizes *Session.Abandon()*, but there is a caveat. Session.Abandon() is not sufficient for this task. Microsoft states that "When you abandon a session, the session ID cookie is not removed from the browser of the user. Therefore, as soon as the session has been abandoned, any new requests to the same application will use the same session ID but will have a new session state

instance." So, to address session fixation holistically, one needs to utilize *Session.Abandon()* and overwrite the cookie header or implement more complex cookie-based session management by enriching the information held within and cookie and performing server-side checks.

Remediating XSS

Ideally, XSS can be remediated by following the below secure coding practices:

Validation of user input

The application should validate every input received immediately upon receiving it. Input validation should be performed on the server-side, using a positive approach (limit the permitted input characters to characters that appear in a whitelist), instead of a negative approach (preventing the usage of characters that appear in a blacklist), since the positive approach helps the programmer avoid potential flaws that result from mishandling potentially malicious characters. Input validation implementation must include the following validation principles in the following order:

- Verify the existence of actual input, do not accept null or empty values when the input is not optional.
- Enforce input size restrictions. Make sure the input's length is within the expected range.
- Validate the input type, make sure the data type received is, in fact, the type expected, and store the input in a variable of the designated type (strongly defined variable).
- Restrict the input range of values. The input's value should be within the acceptable range of values for the input's role in the application.
- Sanitize special characters, unless there is a unique functional need, the input character set should be limited to [a-z], [A-Z] and [0-9]
- Ensure logical input compliance.

HTML encoding to user-controlled output

The application should encode user-controlled input in the following cases:

- Prior to embedding user-controlled input within browser targeted output.
- Prior to documenting user-controlled input into log files (to prevent malicious scripts from affecting administrative users who view the logs through a web interface)

The following inputs match the user-controlled criteria:

- Dynamic values that originate directly from user input (for example, GET, POST, COOKIE, HEADER, FILE UPLOAD, and QUERYSTRING values)
- User-controlled data repository values (database, log, files, etc.)
- Session values originated directly from user input or user-controlled data repository values.
- Values received from external entities (machines or human-controlled)
- Any other value which could have been affected by the user.
- The encoding process should verify that input matching the given criteria will be processed through a data sanitization component, which will replace non-alphanumerical characters in their HTML representation before including these values in the output sent to the user or the log file. This operation ensures that every script will be presented to the user rather than executed in the user's browser.

Additional instructions:

- Do not embed user input into client-side scripts. Values deriving from user input should not be directly embedded as part of an HTML tag, script tag (JS/VBS), HTML event, or HTML property.
- Complimentary instructions for protecting the application against cross-site scripting can be found at the following URL: Cross Site Scripting Prevention Cheat Sheet
- A list of HTML encoded character representations can be found at the following URL: Special Characters in HTML

Please also note that Content-Security-Policy (CSP) headers significantly reduce the risk and impact of XSS attacks in modern browsers by specifying a whitelist in the HTTP response headers, which dictate the HTTP response body can do. Please refer to the following resource for more in-depth details around CSP: Content Security Policy

Before we continue, let us also remind you that cookies should be marked as HTTPOnly for XSS attacks to not be able to capture them. Bypasses exist, but they are out of this module's scope.

Remediating CSRF

It is recommended that whenever a request is made to access each function, a check should be done to ensure the user is authorized to perform that action.

The preferred way to reduce the risk of a Cross-Site Request Forgery (CSRF) vulnerability is to modify session management mechanisms and implement additional, randomly generated, and non-predictable security tokens (a.k.a Synchronizer Token Pattern) or responses to each HTTP request related to sensitive operations.

Other mechanisms that can impede the ease of exploitation include: Referrer header checking. Performing verification on the order in which pages are called. Forcing sensitive functions to confirm information received (two-step operation) – although none of these are effective as a defense in isolation and should be used in conjunction with the random token mentioned above.

In addition to the above, explicitly stating cookie usage with the SameSite attribute can also prove an effective anti-CSRF mechanism.

Please refer to the following resource for more in-depth details around SameSite cookies: SameSite cookies explained

Remediating Open Redirect

The safe use of redirects and forwards can be done in several ways:

- Do not use user-supplied URLs (or partial URL elements) and have methods to strictly validate the URL.
- If user input cannot be avoided, ensure that the supplied value is valid, appropriate for the application, and is authorized for the user.
- It is recommended that any destination input be mapped to a value rather than the actual URL or portion of the URL and that server-side code translates this value to the target URL.
- Sanitize input by creating a list of trusted URLs (lists of hosts or a regex).
- Force all redirects to first go through a page notifying users that they are being redirected from your site and require them to click a link to confirm (a.k.a Safe Redirect).