Additional CSRF Protection Bypasses

Even though diving deeper into CSRF protection bypasses is out of this module's scope, find below some approaches that may prove helpful during engagements or bug bounty hunting.

Null Value

You can try making the CSRF token a null value (empty), for example:

`CSRF-Token:`

This may work because sometimes, the check is only looking for the header, and it does not validate the token value. In such cases, we can craft our cross-site requests using a null CSRF token, as long as the header is provided in the request.

Random CSRF Token

Setting the CSRF token value to the same length as the original CSRF token but with a different/random value may also bypass some anti-CSRF protection that validates if the token has a value and the length of that value. For example, if the CSRF-Token were 32-bytes long, we would re-create a 32-byte token.

Real:

`CSRF-Token: 9cfffd9e8e78bd68975e295d1b3d3331`

Fake:

`CSRF-Token: 9cfffl3dj3837dfkj3j387fjcxmfjfd3`

Use Another Session's CSRF Token

Another anti-CSRF protection bypass is using the same CSRF token across accounts. This may work in applications that do not validate if the CSRF token is tied to a specific account or not and only check if the token is algorithmically correct.

Create two accounts and log into the first account. Generate a request and capture the CSRF token. Copy the token's value, for example, `CSRF-Token=9cfffd9e8e78bd68975e295d1b3d3331`.

Log into the second account and change the value of *CSRF-Token* to `9cfffd9e8e78bd68975e295d1b3d3331` while issuing the same (or a different) request. If the request is issued successfully, we can successfully execute CSRF attacks using a token generated through our account that is considered valid across multiple accounts.

Request Method Tampering

To bypass anti-CSRF protections, we can try changing the request method. From *POST* to *GET* and vice versa.

For example, if the application is using POST, try changing it to GET:

Code: http

```http
POST /change_password
POST body:
new_password=pwned&confirm_new=pwned
```

Code: http

```http
GET /change_password?new_password=pwned&confirm_new=pwned
```

Unexpected requests may be served without the need for a CSRF token.

Delete the CSRF token parameter or send a blank token

Not sending a token works fairly often because of the following common application logic mistake. Applications

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

sometimes only check the token's validity if the token exists or if the token parameter is not blank.

Real Request:

Code: http

```http
POST /change_password
POST body:
new_password=qwerty&csrf_token=9cfffd9e8e78bd68975e295d1b3d3331
```

Try:

Code: http

```http
POST /change_password
POST body:
new_password=qwerty
```

Or:

Code: http

```http
POST /change_password
POST body:
new_password=qwerty&csrf_token=
```

Session Fixation > CSRF

Sometimes, sites use something called a double-submit cookie as a defense against CSRF. This means that the sent request will contain the same random token both as a cookie and as a request parameter, and the server checks if the two values are equal. If the values are equal, the request is considered legitimate.

If the double-submit cookie is used as the defense mechanism, the application is probably not keeping the valid token on the server-side. It has no way of knowing if any token it receives is legitimate and merely checks that the token in the cookie and the token in the request body are the same.

If this is the case and a session fixation vulnerability exists, an attacker could perform a successful CSRF attack as follows:

Steps:

```
1. Session fixation
2. Execute CSRF with the following request:
```

Code: http

```http
POST /change_password
Cookie: CSRF-Token=fixed_token;
POST body:
new_password=pwned&CSRF-Token=fixed_token
```

Anti-CSRF Protection via the Referer Header

If an application is using the referer header as an anti-CSRF mechanism, you can try removing the referer header. Add the following meta tag to your page hosting your CSRF script.

`<meta name="referrer" content="no-referrer">`

Bypass the Regex

Sometimes the Referer has a whitelist regex or a regex that allows one specific domain.

Let us suppose that the Referer Header is checking for *google.com*. We could try something like `www.google.com.pwned.m3`, which may bypass the regex! If it uses its own domain (`target.com`) as a whitelist, try using the target domain as follows `www.target.com.pwned.m3`.

You can try some of the following as well:

`www.pwned.m3?www.target.com` or `www.pwned.m3/www.target.com`

In the next section, we will cover Open Redirect vulnerabilities focusing on attacking a user's session.