Bypassing CSRF Protection

Protection exists != not exploitable.

Vickie Li Follow
Aug 22 · 5 min read ★

No matter the obstacles, we're gonna CSRF.

CSRFs are common and very easy to exploit. Upon first glance, most sites seem to be doing it right: when you examine requests on sensitive actions, they more often than not will have some sort of CSRF protection implemented. Sometimes, it's a CSRF token tagged onto the request body, sometimes, it could be a referer check, or sometimes, it's a special header or cookie.

But the existence of protection does not mean that it's comprehensive, well implemented and cannot be bypassed. Today, let's talk about some of the techniques that I have used to bypass CSRF protection in the wild.

(The same goes for SSRF protection. Just because a site has defense mechanisms implemented, does not mean that it's *well protected*. Read more about bypassing SSRF protection here.)

All CSRFs

No matter the type of CSRF protection deployed, you can always try two things first: clickjacking and changing the request method.

Clickjacking

(If you aren't familiar with clickjacking attacks, more information can be found here.)

Exploiting clickjacking on the same endpoint bypasses all CSRF protection. Because technically, the request is indeed originating from the legitimate site. If the page where the vulnerable endpoint is located on is vulnerable to clickjacking, all CSRF protection will be rendered irrelevant and you will be able to achieve the same results as a CSRF attack on the endpoint, albeit with a bit more effort.

Change the request method

Another thing worth trying is changing the request method of the request. If the sensitive request that you would like to forge is sent via the POST method, try converting the request to a GET request. And if the action is done via a GET, try converting it into a POST. The application might still execute the action and the same protection mechanism is often not in place.

For example, this request:

```
POST /change_password

POST body:

new_password=qwerty
```

Can be rewritten as:

```
GET /change_password?new_password=qwerty
```

CSRF Protection via Tokens

Just because a site is using CSRF tokens does not mean that it is validating them properly. Here are a few things that you can try to bypass CSRF protection via tokens.

Delete the token param or send a blank token

Not sending a token works fairly often because of this common application logic mistake: applications sometimes only check the validity of the token if the token exists, or if the token parameter is not blank. In this case, sending a request without the token, or a blank value as the token may be all you need to bypass the protection.

For example, if a legitimate request looks like this:

```
POST /change_password

POST body:

new_password=qwerty &csrf_tok=871caef0757a4ac9691aceb9aad8b65b
```

Try this:

```
POST /change_password

POST body:
```

```
new_password=qwerty
```

Or, this:

```
POST /change_password

POST body:

new_password=qwerty &csrf_tok=
```

Use another session's CSRF token

The application might only be checking if the token is *valid* or not, and not checking if it *belongs to the current user*. If that's the case, you can simply hard code your own CSRF token into the payload!

Let's say the victim's token is *871caef0757a4ac9691aceb9aad8b65b*, and yours is YOUR_TOKEN. You can obtain your own CSRF token easily but not the victim's token. Try to bypass the CSRF protection by providing your own token in the place of the legitimate token.

In other words, instead of sending this:

```
POST /change_password

POST body:

new_password=qwerty &csrf_tok=871caef0757a4ac9691aceb9aad8b65b
```

Send this:

```
POST /change_password

POST body:

new_password=qwerty &csrf_tok=YOUR_TOKEN
```

Session fixation

Sometimes, sites use something called a double-submit cookie as a defense against CSRF. This means that the request sent 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 seen as legitimate. This defense is quite commonly seen in the wild.

If the double-submit cookie is used as the defense mechanism, the application is probably not keeping the valid token server-side. It has no way of knowing if any token it receives is actually legitimate, and is merely checking that the token in the cookie and the token in the request body is the same. This means that if you can send along a fake cookie as well, you'll still be able to execute the CSRF.

The attack will then consist of two steps: first, you use a session fixation technique to make the victim's browser store whatever value you choose as the CSRF token cookie. Then, execute the CSRF with the same CSRF token that you chose as the cookie.

1. Session fixation. This is an attack that will allow you to control a victim's cookie store. More about it here.
2. Execute CSRF with the following request:

```
POST /change_password

Cookie: CSRF_TOK=FAKE_TOKEN;

POST body:

new_password=qwerty &csrf_tok=FAKE_TOKEN
```

CSRF Protection via Referer

Let's say that *attacker.com* is a domain that you own. And *bank.com* is the site that you are attacking. The site is not using CSRF tokens but is checking the referer header instead. What can you do now?

Remove the referer header

Similar to sending a blank token, sometimes all you need to do to bypass a referer check is to simply not send a referer. To do this, you can add the following meta tag to the page hosting your payload:

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

The application might only be validating the referer if one is sent, in that case, you've successfully bypassed its CSRF protection!

Bypass the regex

If the referer check is based on a whitelist, you can try bypassing the regex used to validate the URL. For example, you can try placing victim domain name in referer URL as a subdomain or as a directory.

If the site is looking for "*bank.com*" in the referer URL, maybe "*bank.com.attacker.com*" or "*attacker.com/bank.com*" will work.

. . .

Hi there, thanks for reading. Please help make this a better resource for new hackers: feel free to point out any mistakes or let me know if there is anything I should add!

. . .

Disclaimer: Trying this on systems where you don't have permission to test is illegal. If you've found a vulnerability, please disclose it responsibly to the vendor. Help make our Internet a safer place :)

Cybersecurity | Bug Bounty | Hacking | Web Security | Pentesting

360 claps

WRITTEN BY

Vickie Li

Follow

Basically a nerd. Studies web security. Stalks great hackers. Creates god awful infographics. https://twitter.com/vickieli7

The Startup

Follow

Medium's largest active publication, followed by +511K people. Follow to join our community.

See responses (1)

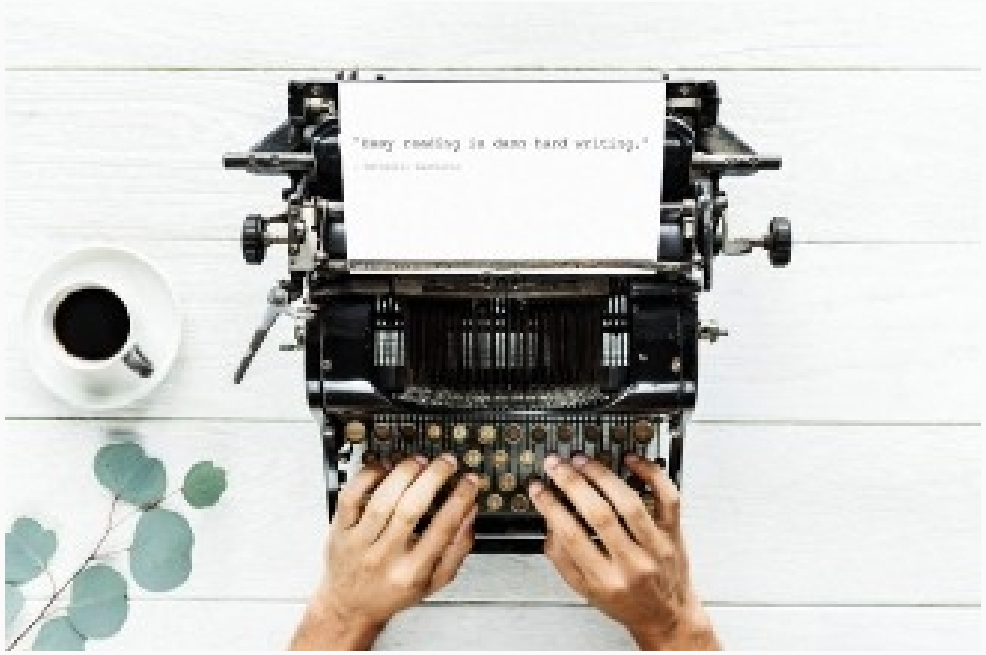
More From Medium

More from The Startup

Write to Express, Not to Impress

Ali Mese in The Startup
Sep 23 · 6 min read ★

20K



More from The Startup

Active listening Lessons From FBI Negotiators That Will Get You What You Want

Thomas Oppong in The Startup
Oct 5 · 4 min read ★

1.91K

More from The Startup

How to Be an Insanely Effective Tech Lead

Ravi Shankar Rajan in The Startup
Sep 30 · 6 min read ★

1.5K