

Cross Site Request Forgery

The attack we explore this week is Cross Site Request Forgery. This attack requires a victim, a trusted website and a malicious website. Basically the user will believe they're connected to a trusted website however it's just a trusted session while they connect to the malicious website that'll inject HTTP requests to cause damage to the user. In this instance we attack the social network application, Elgg.

```
[10/26/19]seed@VM:~$ cd /var/www/CSRF
[10/26/19]seed@VM:.../CSRF$ ls
Attacker  Elgg
[10/26/19]seed@VM:.../CSRF$
```

Above is the location for the Elgg and attacker URLs on the VM.

```
<VirtualHost *:80>
    ServerName http://www.csrflabelgg.com
    DocumentRoot /var/www/CSRF/Elgg
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.csrfabattacker.com
    DocumentRoot /var/www/CSRF/Attacker
</VirtualHost>
```

Above is the virtualhosts enabled on the apache server

3.1)

| St... | Met... | Fi... | Di... | Ca... | T... | Transf... | Size | 0 ms | 2.56 s | 5.12 s | 7.4 |
|-------|--------|---------|-------|-----------|------|-----------|----------|----------|--------|--------|-----|
| 200 | GET | boby... | ... | docum... | html | 2.89 KB | 9.97 KB | → 107 ms | | | |
| 304 | GET | fon... | ... | styleh... | css | cached | 28.38 KB | → 12 ms | | | |
| 304 | GET | elg... | ... | styleh... | css | cached | 58.10 KB | → 7 ms | | | |
| 304 | GET | col... | ... | styleh... | css | cached | 3.80 KB | → 5 ms | | | |
| 304 | GET | jqu... | ... | script | js | cached | 0 B | → 4 ms | | | |
| 304 | GET | jqu... | ... | script | js | cached | 0 B | → 10 ms | | | |
| 304 | GET | req... | ... | script | js | cached | 800 B | → 4 ms | | | |
| 304 | GET | req... | ... | script | js | cached | 0 B | → 6 ms | | | |
| 304 | GET | elg... | ... | script | js | cached | 0 B | → 5 ms | | | |
| 304 | GET | 42t... | ... | img | jpeg | cached | 863 B | → 13 ms | | | |
| 304 | GET | 43l... | ... | img | jpeg | cached | 9.06 KB | → 73 ms | | | |
| 304 | GET | enjs | ... | script | js | cached | 0 B | → 8 ms | | | |
| 304 | GET | initjs | ... | script | js | cached | 619 B | → 8 ms | | | |
| 304 | GET | rea... | ... | script | js | cached | 271 B | → 5 ms | | | |
| 304 | GET | Plu... | ... | script | js | cached | 630 B | → 10 n | | | |
| 200 | POST | add... | ... | xhr | json | 629 B | 309 B | | | | |

| Headers | Cookies | Params | Response |
|-------------------------------------|---------|--------|----------|
| ▼ Filter request parameters | | | |
| ▼ Query string | | | |
| _elgg_token: N1amTbgXdHDtMbUYnXsW8Q | | | |
| _elgg_ts: 1572674340 | | | |
| friend: 43 | | | |
| ▼ Form data | | | |
| _elgg_token: N1amTbgXdHDtMbUYnXsW8Q | | | |
| _elgg_ts: 1572674340 | | | |

Using Elgg I captured a HTTP GET and HTTP POST. The HTTP GET didn't have any parameters but the HTTP POST parameters are `_elgg_token`, and `_elgg_ts`.

3.2 & 3.3)

To perform this GET attack, the URL created when adding boby as a friend has to be inserted into the attacking website. This URL carries the token and timestamp parameters above to authenticate the friend request.

http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1572674340&__elgg_token=N1amTbgXdHDtMbUYnXsw8Q

Insert above url in a tag on the attacker websites html code.

I couldn't perform the POST attack properly but what should have occurred was that in the html file, the function created to forge a HTTP POST should have posted a blog post on Alices page stating Bobby is the hero.

Question 1: One way Bobby could have got Alice's account password would have been to sql inject the password attribute to pull a table of Alice's information.

Question 2: No, random users going to the attack website won't be affected by the attack because the GET URL has a timestamp and token that's unique per user.