<BEYOND XSS>

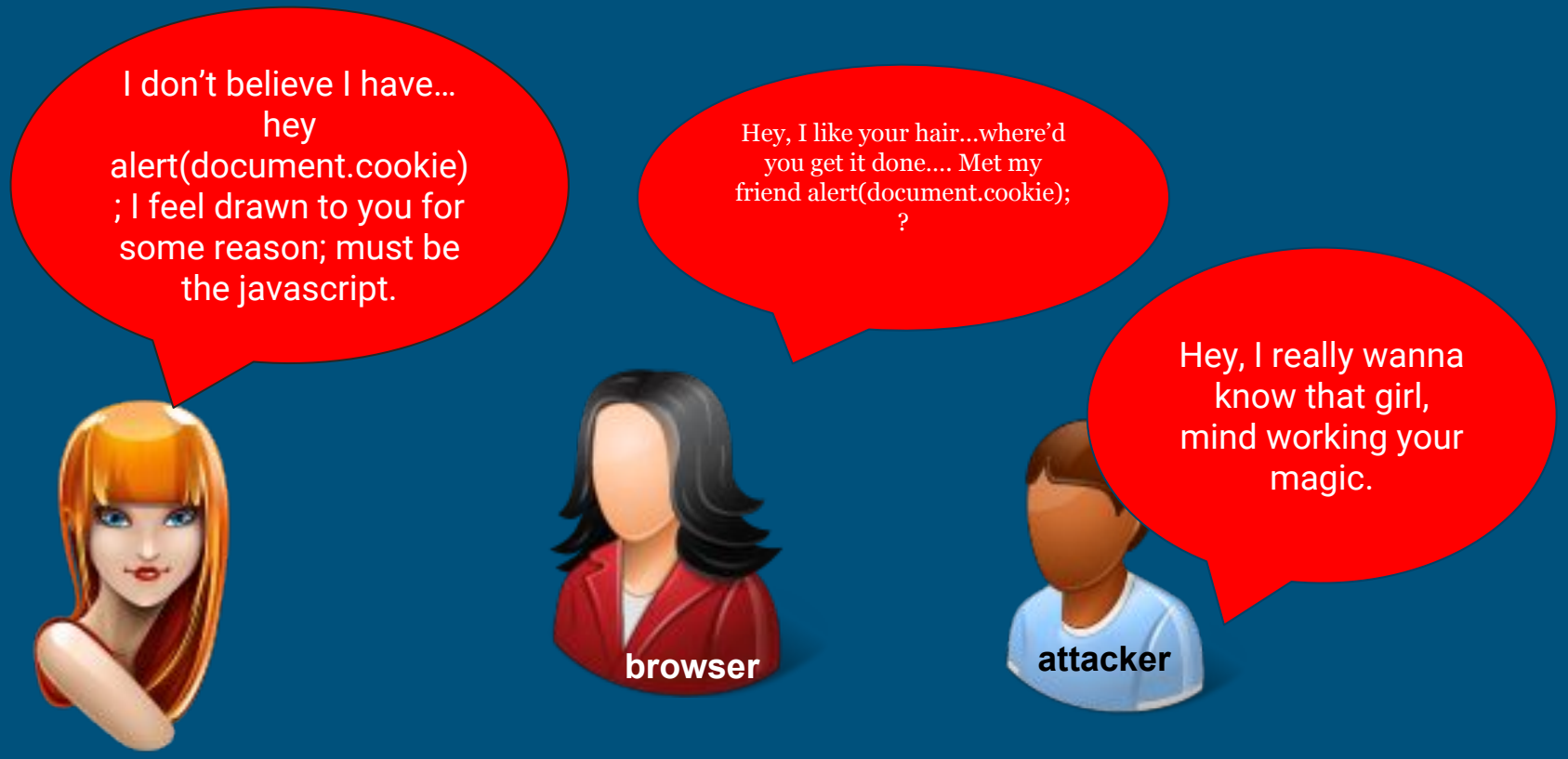
Project by:
<Judy> & <Munir>

>what to cover

> What is xss
> Types of xss
> Just because i have an xss in my web app, does it make my web app risky to use?
> Oooh i have an xss in my web app how can i exploit it?
> Demos
> protections
> Q& A

<Title> XSS</Title>

What is this xss and what does it entail?



<Theory>Types</theory>

Reflected

- where the malicious string originates from the victim's request

Stored

- where the malicious string originates from the website's database.

DOM

- where the vulnerability is in the client-side code rather than the server-side code.

>Simple Demo

- Simple xss

Reflected :

Input the infamous payload

<script>alert(“hey am a reflected xss”);</script>

to confirm XSS.

http://142.93.112.220/admin/home.php/"onmouseover="alert(document.cookie);"

Stored:

Another example of XSS, but this time it ll be Stored XSS, where the XSS payload is saved and executed when the saved payload is executed in victim’s browser.

</Beyond XSS>

What can we do with a site that's vulnerable to XSS?

> Cookie grabbing
> Phishing attacks
> Defacement (scary and kuwl)

What will we do <Demo Do>

- Allows you to sign in with just a cookie
- Sends the sensitive cookies insecurely
- Allows user-defined text to be parsed without sanitizing
- Allows sending data out to unknown domains

>Lab set up

victims machine

http:142.93.112.220/admin/

http:142.93.112.220/user/

attack machine

http://159.65.201.122/0x676f6861636b696e67/login

Requirements:

Chrome (or/and) firefox with edit this cookie plugin ☺

>Lets grab some cookies

A lot of sites will let you sign in with just a cookie. which means you can take a cookie from one machine, move it to another and sign in without knowing the users username or password.

The number of developers who ensure a cookie only worked on the computer it was created on by browser fingerprinting are 0.1 out of 10

> Cookie Monster

- In this demo, I show how a user impersonates an administrator by Stealing their session using stored XSS. Say attacker logins into a vulnerable web app He crafts a XSS payload that returns victim's cookies to an attacker controlled server. When the administrator who is a victim, logs in to his account and views the vulnerable page the stored XSS payload gets triggered in admin's browser that send his cookies to attacker.

>..contd

- Now, when admin logs in
- Now, modifying attacker session using the obtain admin cookies gives admin user.
- (notes) check cookies on source code before and after escalation

>Defacement

Since the web page has no element identifiers or names we have to count them on the page , in the payload above if you look at the content you will see a section of the title with a class name of content-header the 0 after it means we are taking the first instance of this content-header as part of our defacement , any other is ignored . without it it will try action on all; the getElementsByTagName helps us tell the application that within the content header there is a heading type 1 i.e. h1 and we also just want the first instance of this heading otherwise it will try loop through all of them. the innerHTML tells the script the content we want to appear in the element we have drilled down to and replace what is there :-) ..... all this just means i defaced you :-D changed your content unwillingly.

>payload to use

```
<script>document.getElementsByClassName("content-header")[0].getElementsByTagName("h1")[0].innerHTML="Hacked by Munir";</script>
```

<explain the payload>

>phishing

Well guess what it’s not just cookies , your credentials too if a payload is injected onto a page will redirect all traffic once loaded to a fake login page; once u enter any details on it it will take you back to your original home in admin after which it will have logged the actual username and password in a text file. we have intentionally named it fake however in a real life scenario it wouldn't be as such this is just to help identify the difference. to view captured password visit the link below:
http://142.93.112.220/fake/shehacks.txt

>payload

```
<script>window.location.href =
"http://142.93.112.220/fake/";</script>
```

</Cure>

So how do u prevent this as:
> user : Nothing really you are doomed
> Developer: set a cookie and define it as http only
>Encode output
>use x-xss protection header
this means a hacker cant access the cookie using document.cookie

injected input

```php
<?php
$str = "<script>alert(document.cookie)</script>";


echo htmlentities($str);

echo htmlentities($str, ENT_QUOTES);
?>
```

Encoded output

```
&lt;script&gt;alert(document.cookie)&lt;/script&gt;&lt;script&gt;alert(document.cookie)&lt;/s
cript&gt;
```

/>

Thank You

You may now ask questions