Command Injection | Tryhackme Walkthrough

Rahul Kumar · Follow
9 min read · Jul 27, 2023

Listen Share More

> *Learn about a vulnerability allowing you to execute commands through a vulnerable app, and its remediations.*

Introduction:

In this room, we're going to be covering the web vulnerability that is command injection. Once we understand what this vulnerability is, we will then showcase its impact and the risk it imposes on an application.

Then, you're going to be able to put this knowledge into practice, namely:

- *How to discover the command injection vulnerability*
- *How to test and exploit this vulnerability using payloads designed for different operating systems*
- *How to prevent this vulnerability in an application*
- *Lastly, you'll get to apply theory into practice learning in a practical at the end of the room.*

To begin with, let's first understand what command injection is. Command injection is the abuse of an application's behavior to execute commands on the operating system, using the same privileges that the application on a device is running with. For example, achieving command injection on a web server running as a user named `joe` *will execute commands under this* `joe` *user - and therefore obtain any permissions that* `joe` *has.*

A command injection vulnerability is also known as a "Remote Code Execution" (RCE) because an attacker can trick the application into executing a series of payloads that they provide, without direct access to the machine itself (i.e. an interactive shell). The web server will process this code and execute it under the privileges and access controls of the user who is running that application.

Command injection is also often known as "Remote Code Execution" (RCE) because of the ability to remotely execute code within an application. These vulnerabilities are often the most lucrative to an attacker because it means that the attacker can directly interact with the vulnerable system. For example, an attacker may read system or user files, data, and things of that nature.

For example, being able to abuse an application to perform the command `whoami` *to list what user account the application is running will be an example of command injection.*

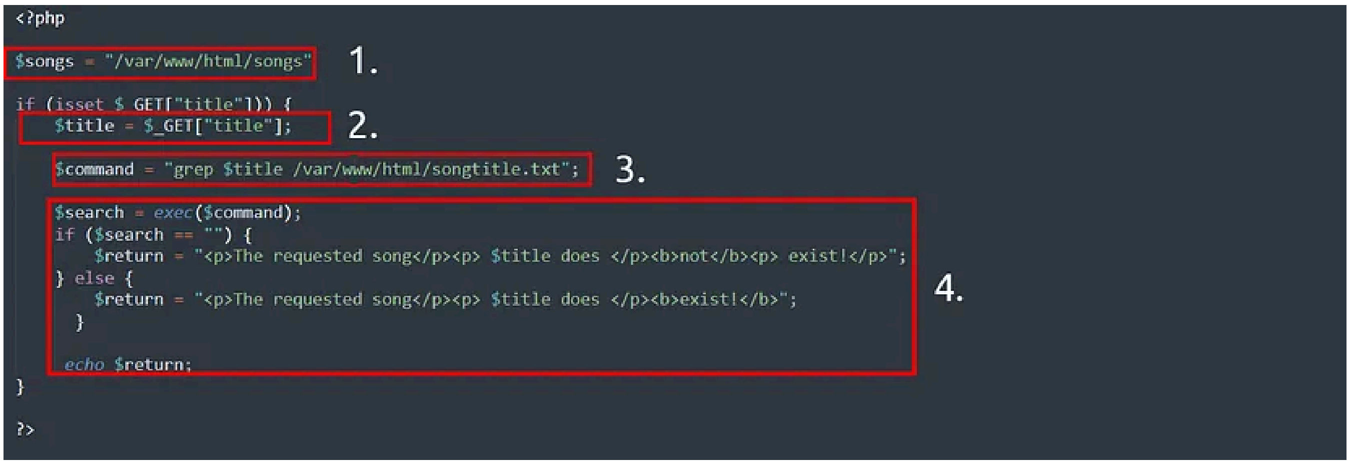
Command injection was one of the top ten vulnerabilities reported by Contrast Security's AppSec intelligence report in 2019. (Contrast Security AppSec., 2019). Moreover, the

OWASP framework constantly proposes vulnerabilities of this nature as one of the top ten vulnerabilities of a web application (OWASP framework).

Discovering Command Injection:

This vulnerability exists because applications often use functions in programming languages such as PHP, Python and NodeJS to pass data to and to make system calls on the machine's operating system. For example, taking input from a field and searching for an entry into a file. Take this code snippet below as an example:

In this code snippet, the application takes data that a user enters in an input field named `$title` *to search a directory for a song title. Let's break this down into a few simple steps.*



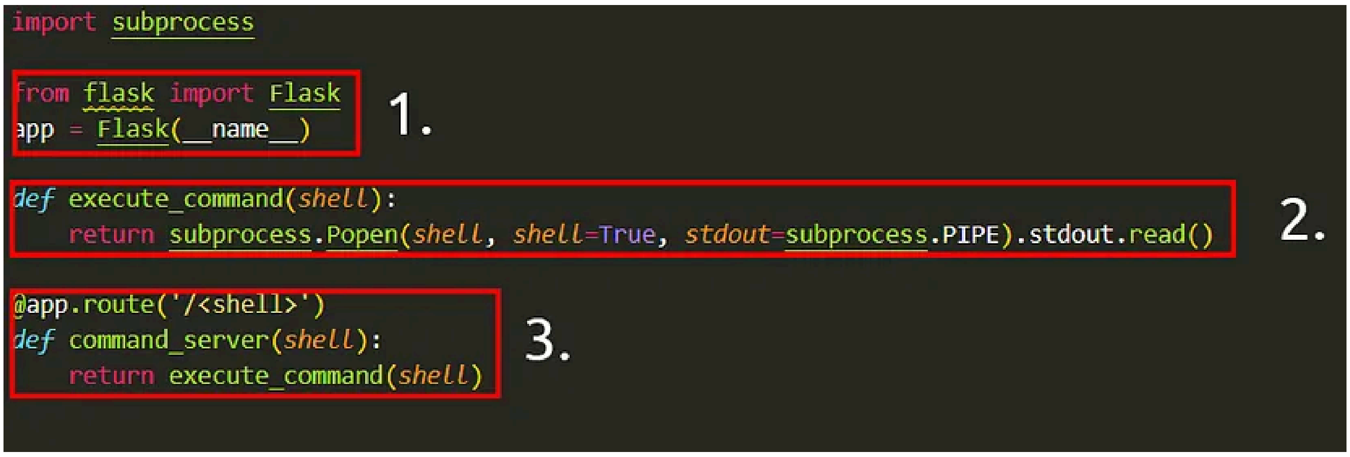
```
<?php
$songs = "/var/www/html/songs"

if (isset $_GET["title"])) {
    $title = $_GET["title"];

    $command = "grep $title /var/www/html/songtitle.txt";

    $search = exec($command);
    if ($search == "") {
        $return = "<p>The requested song</p><p> $title does </p><b>not</b><p> exist!</p>";
    } else {
        $return = "<p>The requested song</p><p> $title does </p><b>exist!</b>";
     }

     echo $return;
}

?>
```

1. The application stores MP3 files in a directory contained on the operating system.
2. The user inputs the song title they wish to search for. The application stores this input into the `$title` *variable.*
3. The data within this `$title` *variable is passed to the command* `grep` *to search a text file named songtitle.txt for the entry of whatever the user wishes to search for.*
4. The output of this search of songtitle.txt will determine whether the application informs the user that the song exists or not.

Now, this sort of information would typically be stored in a database; however, this is just an example of where an application takes input from a user to interact with the application's operating system.

An attacker could abuse this application by injecting their own commands for the application to execute. Rather than using `grep` *to search for an entry in* `songtitle.txt`*, they could ask the application to read data from a more sensitive file.*

Abusing applications in this way can be possible no matter the programming language the application uses. As long as the application processes and executes it, it can result in command injection. For example, this code snippet below is an application written in Python.



```
import subprocess

from flask import Flask
app = Flask(__name__)
```
1.

```
def execute_command(shell):
    return subprocess.Popen(shell, shell=True, stdout=subprocess.PIPE).stdout.read()
```
2.

```
@app.route('/<shell>')
def command_server(shell):
    return execute_command(shell)
```
3.

Note, you are not expected to understand the syntax behind these applications. However, for the sake of reason, I have outlined the steps of how this Python application works as well.

1. *The "flask" package is used to set up a web server*
2. *A function that uses the "subprocess" package to execute a command on the device*
3. *We use a route in the webserver that will execute whatever is provided. For example, to execute* `whoami`*, we'd need to visit http://flaskapp.thm/whoami*

Ques 1: What variable stores the user's input in the PHP code snippet in this task?
Ans 1: $title

Ques 2: What HTTP method is used to retrieve data submitted by a user in the PHP code snippet?
Ans 2: get

Ques 3: If I wanted to execute the `id` command in the Python code snippet, what route would I need to visit?
Ans 3: /id

Exploiting Command Injection:

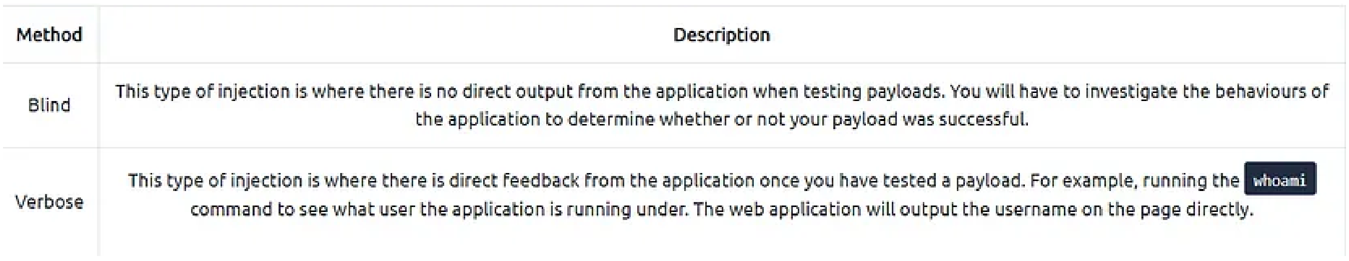
You can often determine whether or not command injection may occur by the behaviours of an application, as you will come to see in the practical session of this room.

Applications that use user input to populate system commands with data can often be combined in unintended behaviour. ***For example, the shell operators `;`, `&` and `&&` will combine two (or more) system commands and execute them both****. If you are unfamiliar with this concept, it is worth checking out the Linux fundamentals module to learn more about this.*

Command Injection can be detected in mostly one of two ways:

1. *Blind command injection*
2. *Verbose command injection*

I have defined these two methods in the table below, where the two sections underneath will explain these in greater detail.



Method	Description
Blind	This type of injection is where there is no direct output from the application when testing payloads. You will have to investigate the behaviours of the application to determine whether or not your payload was successful.
Verbose	This type of injection is where there is direct feedback from the application once you have tested a payload. For example, running the `whoami` command to see what user the application is running under. The web application will output the username on the page directly.

Detecting Blind Command Injection

Blind command injection is when command injection occurs; however, there is no output visible, so it is not immediately noticeable. For example, a command is executed, but the web application outputs no message.

For this type of command injection, we will need to use payloads that will cause some time delay. For example, the `ping` *and* `sleep` *commands are significant payloads to test with. Using* `ping` *as an example, the application will hang for x seconds in relation to how many pings you have specified.*

Another method of detecting blind command injection is by forcing some output. This can be done by using redirection operators such as `>` *. If you are unfamiliar with this, I recommend checking out the Linux fundamentals module. For example, we can tell the web application to execute commands such as* `whoami` *and redirect that to a file. We can then use a command such as* `cat` *to read this newly created file's contents.*

Testing command injection this way is often complicated and requires quite a bit of experimentation, significantly as the syntax for commands varies between Linux and Windows.

The `curl` *command is a great way to test for command injection. This is because you are able to use* `curl` *to deliver data to and from an application in your payload. Take this code snippet below as an example, a simple curl payload to an application is possible for command injection.*

```
curl http://vulnerable.app/process.php%3Fsearch%3DThe%20Beatles%3B%20whoami
```

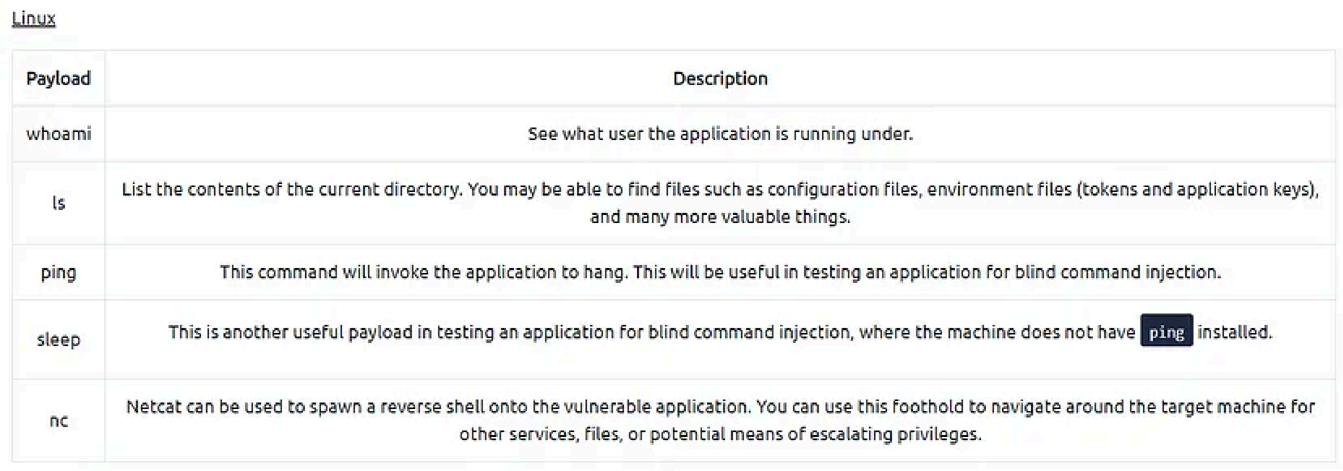
Detecting Verbose Command Injection

Detecting command injection this way is arguably the easiest method of the two. Verbose command injection is when the application gives you feedback or output as to what is happening or being executed.

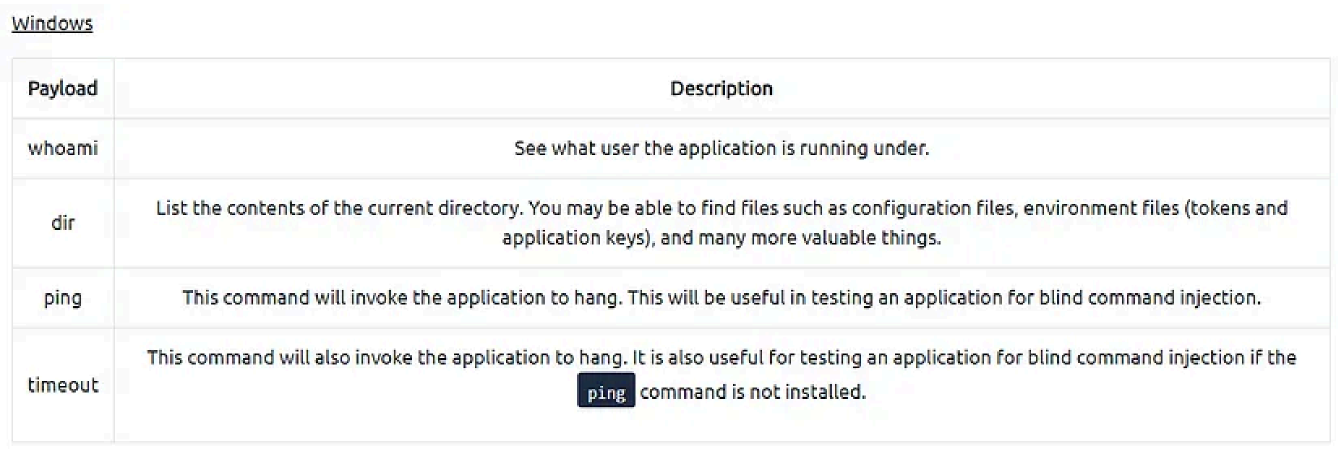
For example, the output of commands such as `ping` *or* `whoami` *is directly displayed on the web application.*

Useful payloads

I have compiled some valuable payloads for both Linux & Windows into the tables below.


Linux

Payload	Description
whoami	See what user the application is running under.
ls	List the contents of the current directory. You may be able to find files such as configuration files, environment files (tokens and application keys), and many more valuable things.
ping	This command will invoke the application to hang. This will be useful in testing an application for blind command injection.
sleep	This is another useful payload in testing an application for blind command injection, where the machine does not have `ping` installed.
nc	Netcat can be used to spawn a reverse shell onto the vulnerable application. You can use this foothold to navigate around the target machine for other services, files, or potential means of escalating privileges.


Windows

Payload	Description
whoami	See what user the application is running under.
dir	List the contents of the current directory. You may be able to find files such as configuration files, environment files (tokens and application keys), and many more valuable things.
ping	This command will invoke the application to hang. This will be useful in testing an application for blind command injection.
timeout	This command will also invoke the application to hang. It is also useful for testing an application for blind command injection if the `ping` command is not installed.

Ques 4: What payload would I use if I wanted to determine what user the application is running as?
Ans 4: whoami

Ques 5: What popular network tool would I use to test for blind command injection on a **Linux** machine?
Ans 5: ping

Ques 6: What payload would I use to test a **Windows** machine for blind command injection?
Ans 6: timeout

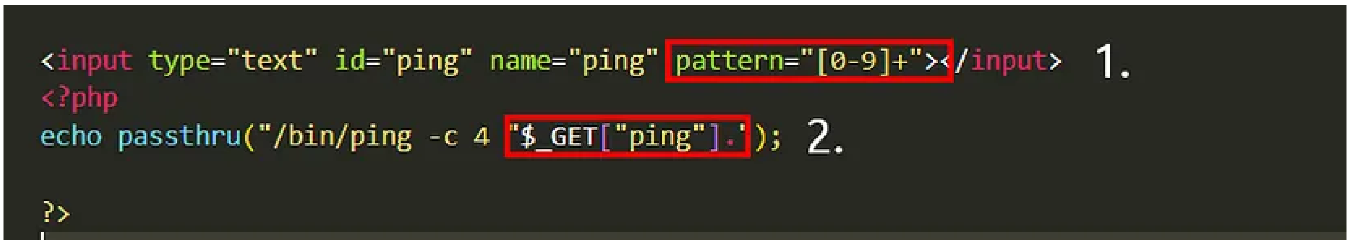
emediating Command Injection:

R Command injection can be prevented in a variety of ways. Everything from minimal use of potentially dangerous functions or libraries in a programming language to filtering input without relying on a user's input. I have detailed these a bit further below. The examples below are of the PHP programming language; however, the same principles can be extended to many other languages.

Vulnerable Functions
In PHP, many functions interact with the operating system to execute commands via shell; these include:

- *Exec*
- *Passthru*
- *System*

Take this snippet below as an example. Here, the application will only accept and process numbers that are inputted into the form. This means that any commands such as `whoami` *will not be processed.*



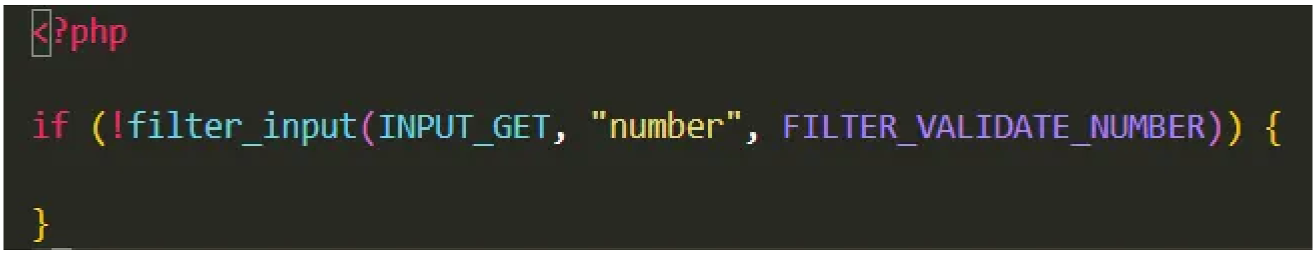
```
<input type="text" id="ping" name="ping" pattern="[0-9]+"></input>
<?php
echo passthru("/bin/ping -c 4 "$_GET["ping"].');

?>
```

1. *The application will only accept a specific pattern of characters (the digits 0–9)*
2. *The application will then only proceed to execute this data which is all numerical.*

These functions take input such as a string or user data and will execute whatever is provided on the system. Any application that uses these functions without proper checks will be vulnerable to command injection.

Input sanitisation
Sanitising any input from a user that an application uses is a great way to prevent command injection. This is a process of specifying the formats or types of data that a user can submit. For example, an input field that only accepts numerical data or removes any special characters such as `>` *,* `&` *and* `/` *.*

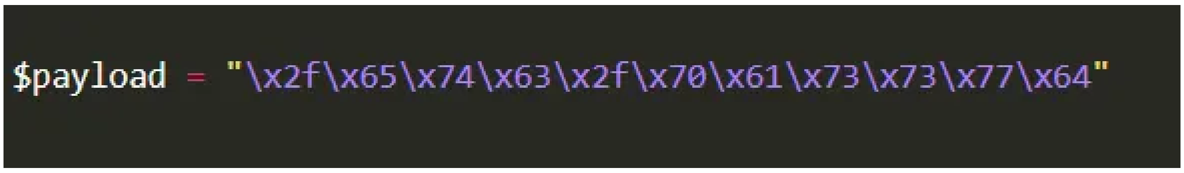
In the snippet below, the `filter_input` *PHP function is used to check whether or not any data submitted via an input form is a number or not. If it is not a number, it must be invalid input.*



```php
<?php

if (!filter_input(INPUT_GET, "number", FILTER_VALIDATE_NUMBER)) {

}
```

Bypassing Filters:
Applications will employ numerous techniques in filtering and sanitising data that is taken from a user's input. These filters will restrict you to specific payloads; however, we can abuse the logic behind an application to bypass these filters. For example, an application may strip out quotation marks; we can instead use the hexadecimal value of this to achieve the same result.

When executed, although the data given will be in a different format than what is expected, it can still be interpreted and will have the same result.

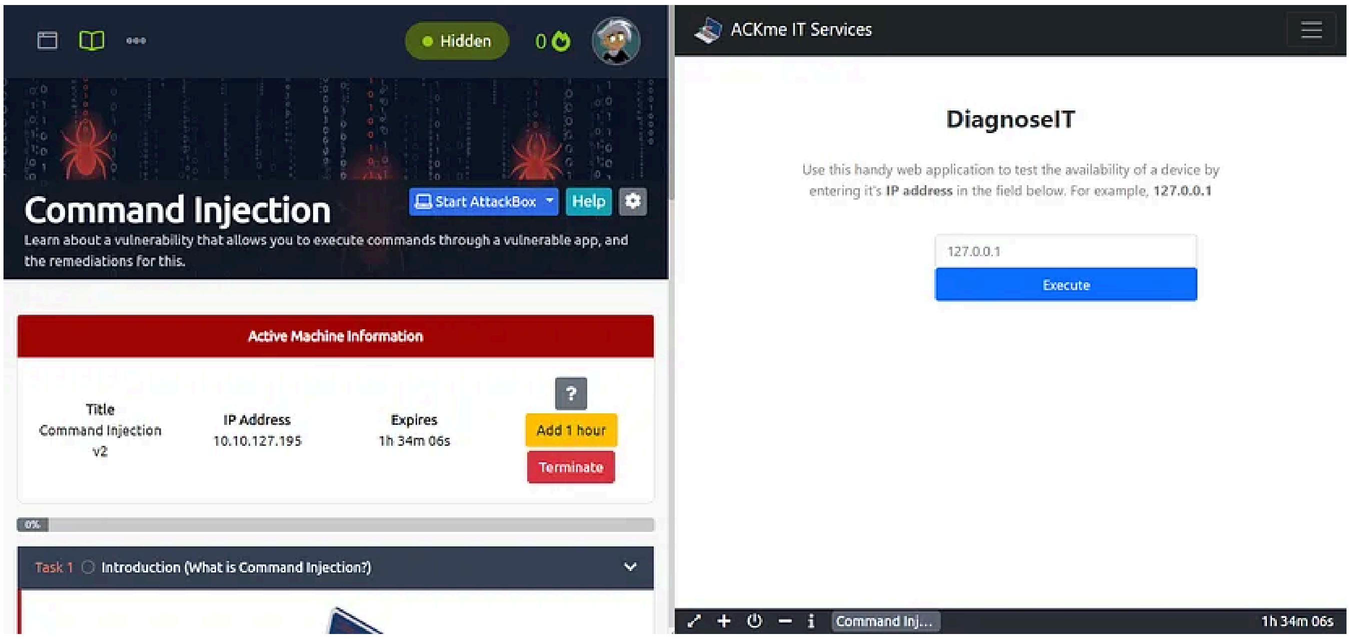


```php
$payload = "\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64"
```

Ques 7: What is the term for the process of "cleaning" user input that is provided to an application?
Ans 7: sanitisation

Practical: Command Injection (Deploy):

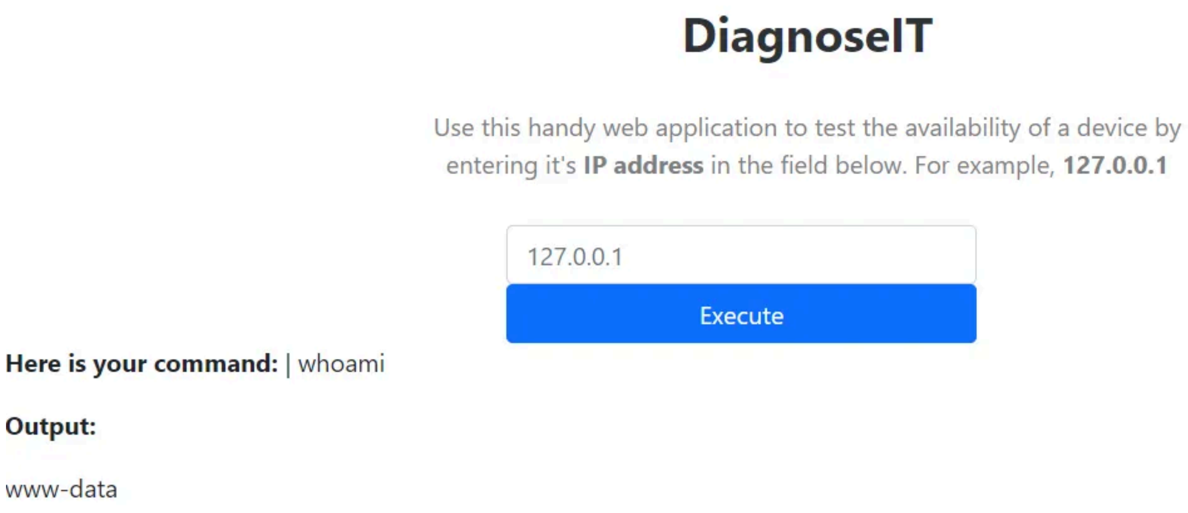
Deploy the machine attached to this task; it will be visible in the split-screen view once it is ready.


Hidden
0
Command Injection
Learn about a vulnerability that allows you to execute commands through a vulnerable app, and the remediations for this.
Start AttackBox
Help
Active Machine Information
Title
Command Injection v2
IP Address
10.10.127.195
Expires
1h 34m 06s
Add 1 hour
Terminate
0%
Task 1 Introduction (What is Command Injection?)
ACKme IT Services
DiagnoseIT
Use this handy web application to test the availability of a device by entering it's IP address in the field below. For example, 127.0.0.1
127.0.0.1
Execute
Command Inj...
1h 34m 06s

Test some payloads on the application hosted on the website visible in split-screen view to test for command injection. Refer to this cheat sheet if you are stuck or wish to explore some more complex payloads.

*Find the contents of the flag located in **/home/tryhackme/flag.txt**. You can use a variety of payloads to achieve this — I recommend trying multiple.*

Ques 7: What user is this application running as?
Ans 7: www-data


DiagnoseIT

Use this handy web application to test the availability of a device by entering it's **IP address** in the field below. For example, **127.0.0.1**

127.0.0.1

Execute

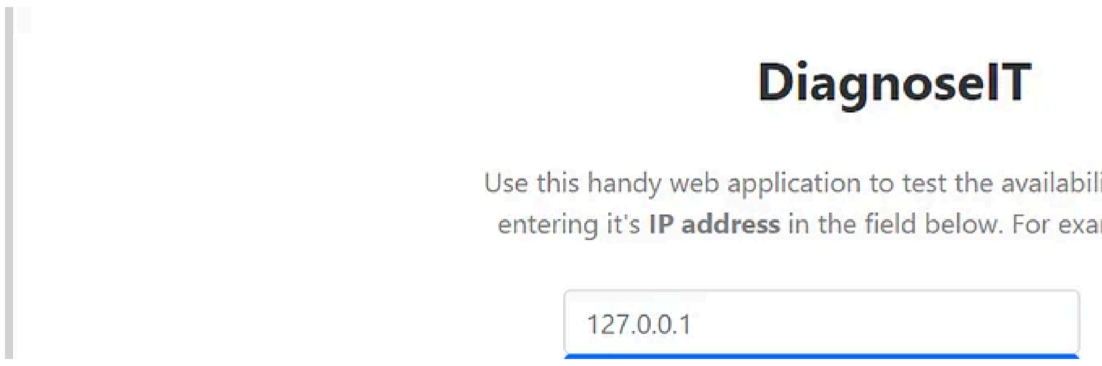
Here is your command: | whoami

Output:

```
www-data
```

Ques 8: What are the contents of the flag located in **/home/tryhackme/flag.txt**?

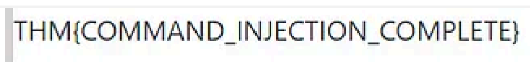
Ans 8: THM{COMMAND_INJECTION_COMPLETE}



DiagnoseIT

Use this handy web application to test the availability of a device by entering it's **IP address** in the field below. For example, **127.0.0.1**

127.0.0.1



THM{COMMAND_INJECTION_COMPLETE}

Conclusion:

Well done for making it to the end of this room. To recap, we've learned about the following elements of command injection:

- *How to discover the command injection vulnerability*
- *How to test and exploit this vulnerability using payloads designed for different operating systems*
- *How to prevent this vulnerability in an application*
- *Applying your learning by performing command injection in a practical application*

As you will have probably discovered, there are multiple payloads that can be used to achieve the same goal. I highly encourage you to go back to the practical element of this task and try some alternative methods of retrieving the flag.

References: https://tryhackme.com/room/oscommandinjection

Tryhackme Tryhackme Walkthrough Cybersecurity Command Injection Os Command Injection

Follow

Written by Rahul Kumar

150 Followers · 6 Following

Cybersecurity Enthusiast!! | COMPTIA SEC+ | CCSK | CEH | MTA S&N | Cybersecurity Analyst | Web Application Security

No responses yet

What are your thoughts?

Respond

More from Rahul Kumar

Rahul Kumar

Burp Suite : Other Module

Take a dive into some of Burp Suite’s lesser known modules

Aug 4, 2023 52 1

```
GET / HTTP/1.1
Host: 10-10-26-169.p.thmlabs.com
User-Agent: Mozilla/5.0 (Windows NT
10.0; Win64; x64; rv:91.0)
Gecko/20100101 Firefox/91.0
Accept:
text/html,application/xhtml+xml,applica
tion/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://tryhackme.com/
Dnt: 1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Sec-Gpc: 1
Cache-Control: max-age=0
Te: trailers
Connection: open
```

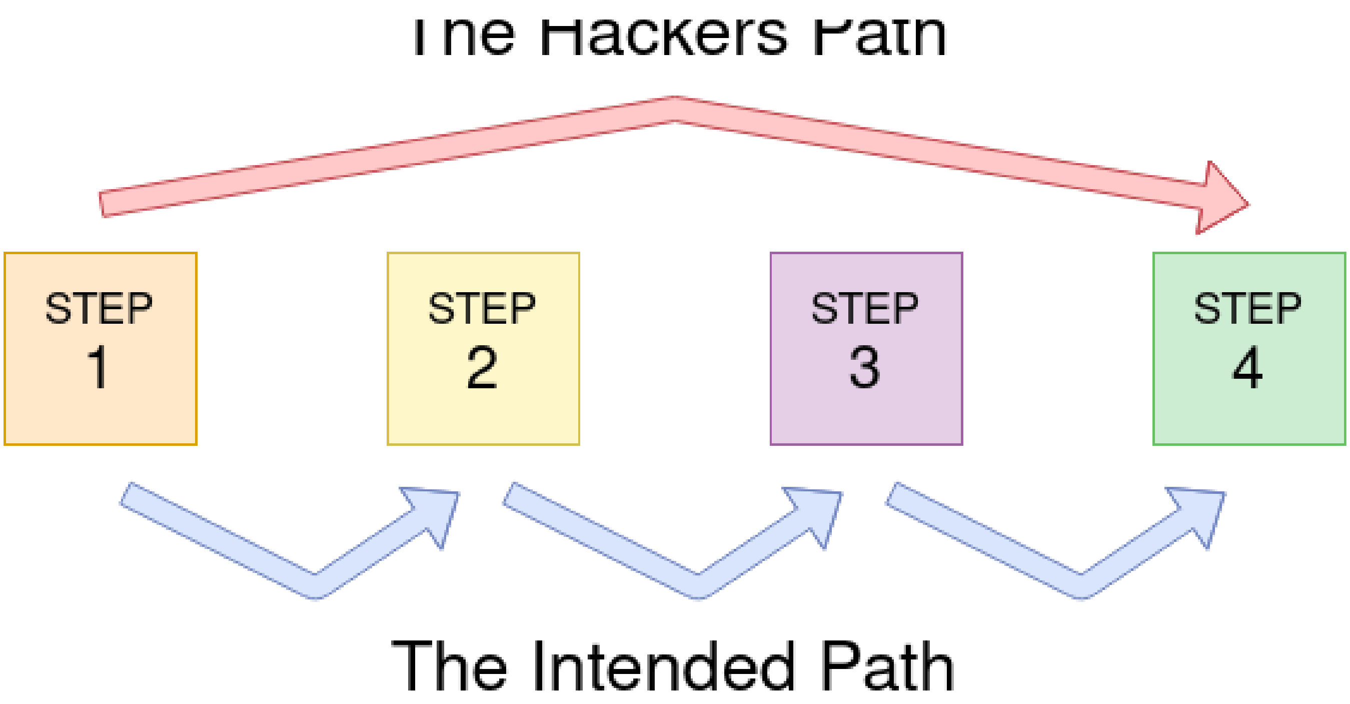
```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Sat, 04 Sep 2021 22:51:00 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Front-End-Https: on
Content-Length: 6613

<!DOCTYPE html>
<html lang=en>
  <head>
    <title>
      Bastion Hosting
    </title>
    <meta charset=utf-8>
    <meta name=viewport content="widtl
    <link rel="icon" type="image/x-ico
    <link href="/assets/css/bootstrap-
    <link href="/assets/css/styles.css
    <link href=/assets/css/home.css re
  </head>
  <body class="d-flex flex-column h-10
    <main class="flex-shrink-0">
```

Rahul Kumar

Burp Suite: Repeater | Tryhackme Walkthrough

Learn how to use Repeater to duplicate requests in Burp Suite

Jul 28, 2023 7



Rahul Kumar

Authentication Bypass | Tryhackme Walkthrough

Learn how to defeat logins and other authentication mechanisms to allow you access to unpermitted areas.

Jul 23, 2023 3

In System Weakness by Rahul Kumar

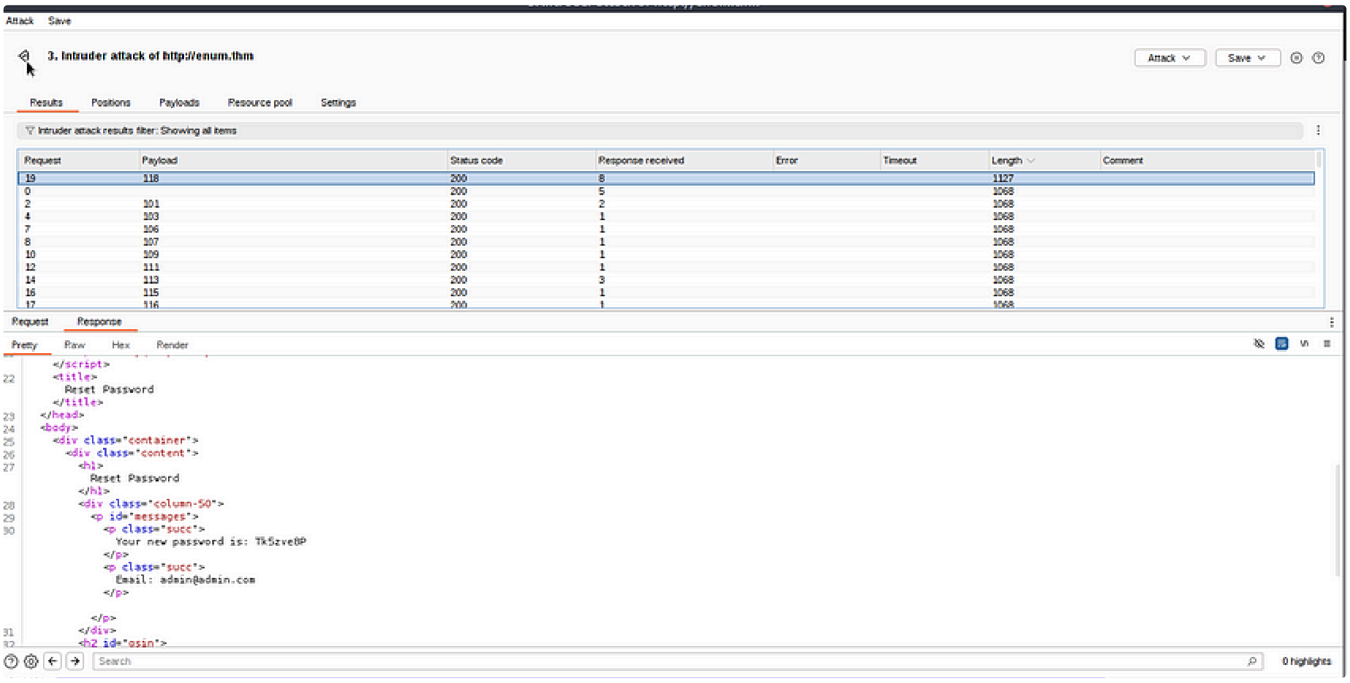
Metasploit: Exploitation | Tryhackme Walkthrough

Using Metasploit for scanning, vulnerability assessment and exploitation.

Feb 2, 2024 84 1

See all from Rahul Kumar

Recommended from Medium



Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
19	118	200	8			1127	
0		200	5			1068	
2	101	200	2			1068	
4	103	200	1			1068	
7	106	200	1			1068	
8	107	200	1			1068	
10	109	200	1			1068	
12	111	200	1			1068	
14	113	200	3			1068	
16	115	200	1			1068	
17	116	200	1			1068	

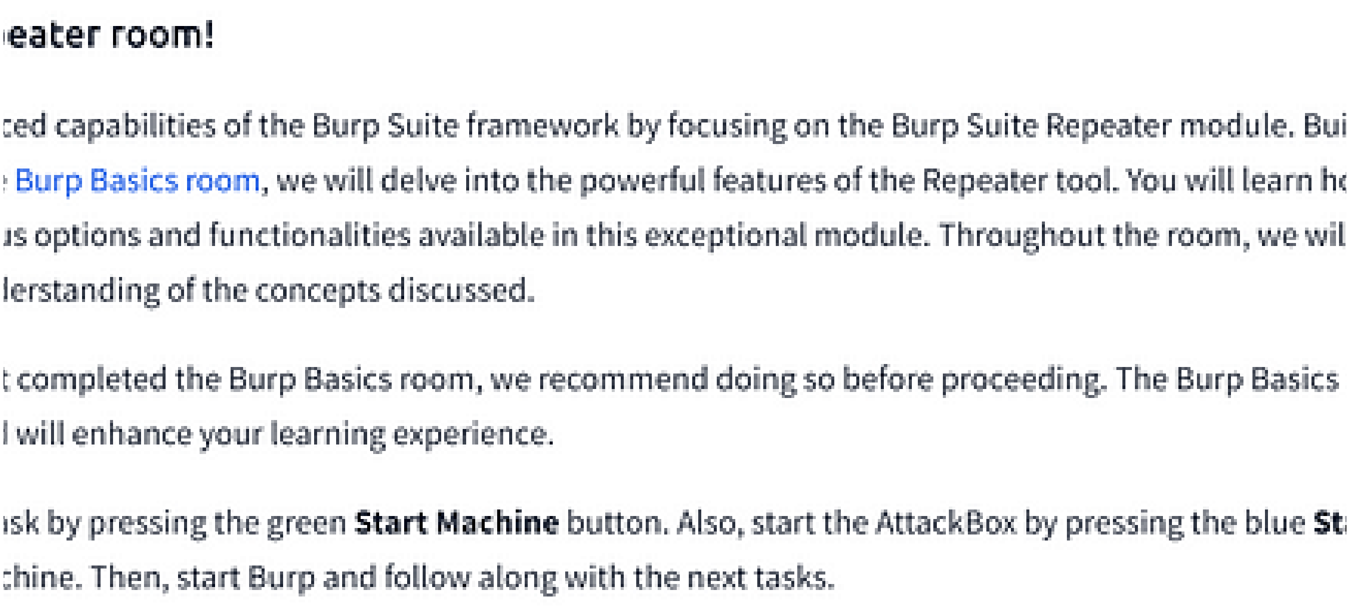
```
    </script>
    <title>
      Reset Password
    </title>
  </head>
  <body>
    <div class="container">
      <div class="content">
        <h1>
          Reset Password
        </h1>
        <div class="column-50">
          <p id="messages">
            <p class="succ">
              Your new password is: TkSzve8P
            </p>
            <p class="succ">
              Email: admin@admin.com
            </p>

          </p>
        </div>
        <h2 id="osin">
```

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



eater room!

ced capabilities of the Burp Suite framework by focusing on the Burp Suite Repeater module. Bui
Burp Basics room, we will delve into the powerful features of the Repeater tool. You will learn h
is options and functionalities available in this exceptional module. Throughout the room, we wil
lerstanding of the concepts discussed.

t completed the Burp Basics room, we recommend doing so before proceeding. The Burp Basics
l will enhance your learning experience.

isk by pressing the green **Start Machine** button. Also, start the AttackBox by pressing the blue **St**
chine. Then, start Burp and follow along with the next tasks.

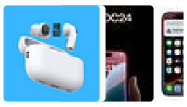
Daniel Schwarzentraub

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

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

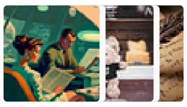
Aug 27, 2024

Lists



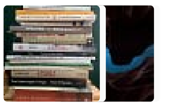
Tech & Tools

22 stories · 377 saves



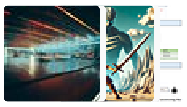
Medium's Huge List of Publications Accepting Submissions

377 stories · 4314 saves



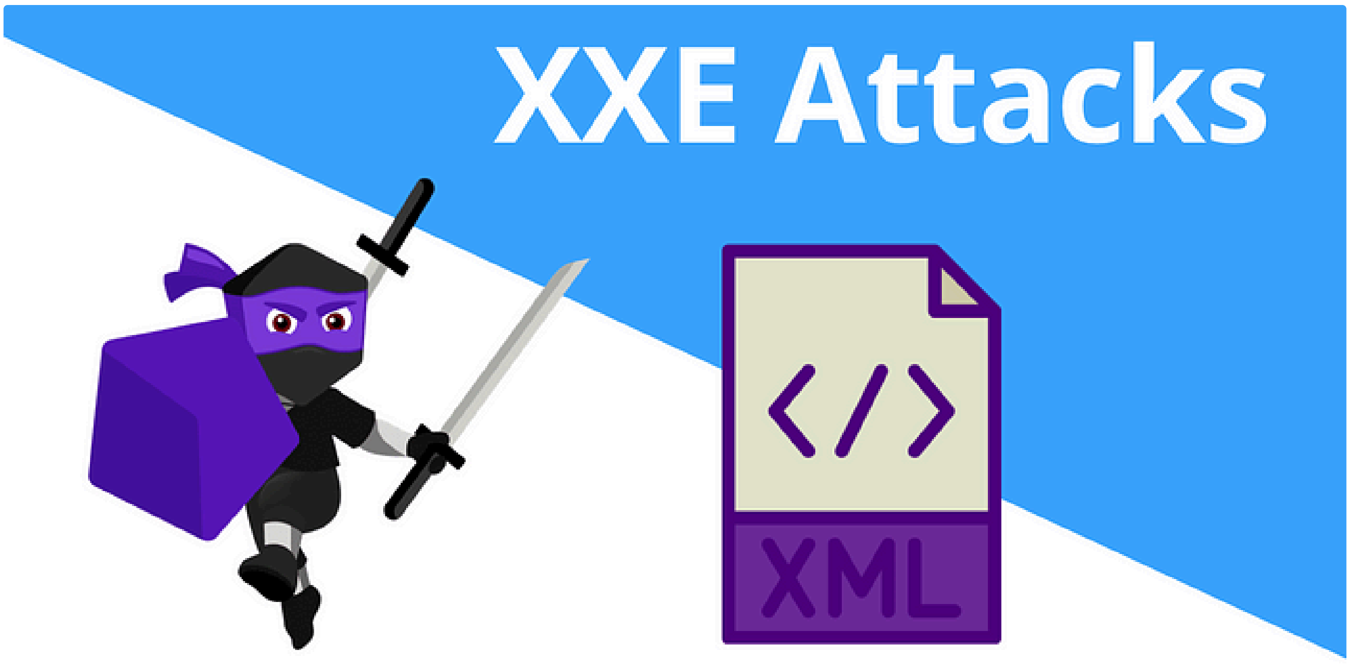
Staff picks

793 stories · 1546 saves



Natural Language Processing

1882 stories · 1519 saves

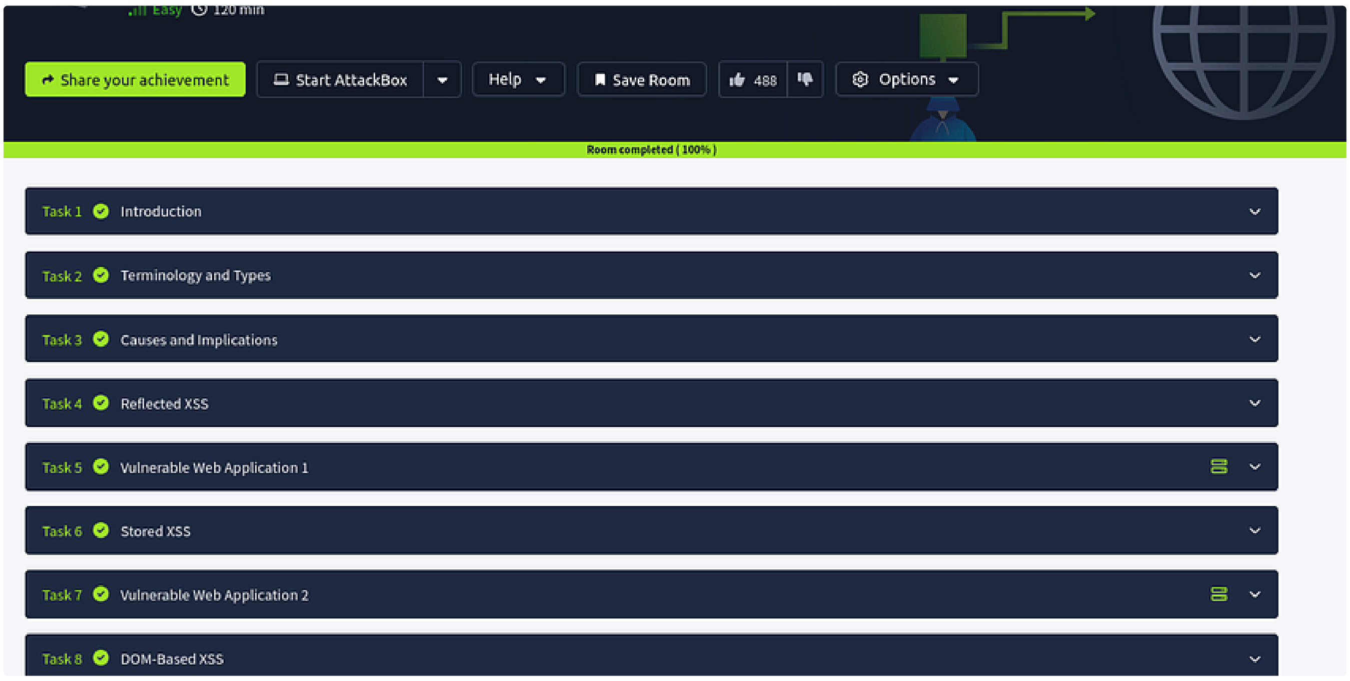


Abhijeet kumawat

Day 13 of 30 Days — 30 Vulnerabilities | XML External Entity (XXE)

Day 13: Mastering XML External Entity (XXE) Vulnerability — Essential Tricks & Techniques Based on Personal Experience and Valuable POCs

Aug 17, 2024 62



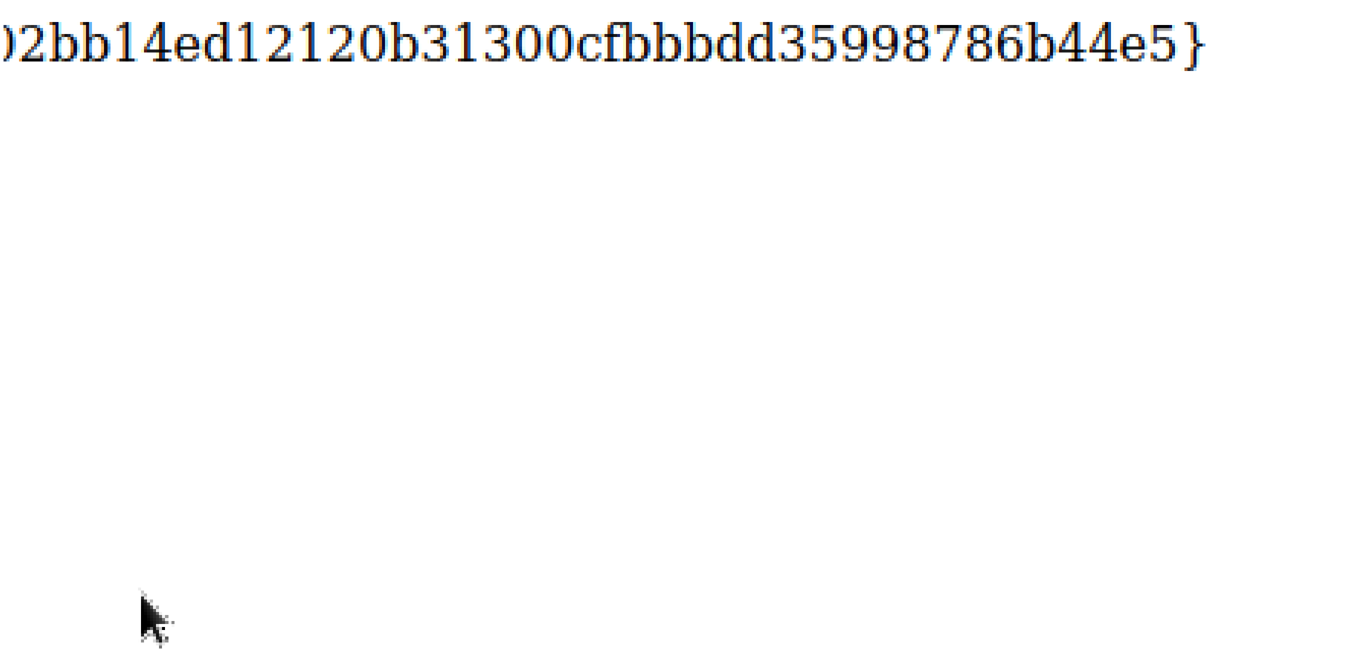
Task 1 Introduction
Task 2 Terminology and Types
Task 3 Causes and Implications
Task 4 Reflected XSS
Task 5 Vulnerable Web Application 1
Task 6 Stored XSS
Task 7 Vulnerable Web Application 2
Task 8 DOM-Based XSS

Jawstar

XSS Tryhackme Walkthrough Write up

Overview:

Nov 20, 2024 4


)2bb14ed12120b31300cfbbbdd35998786b44e5}

embossdotar

TryHackMe — Shells Overview — Writeup

Key points: Shells | Reverse Shell | Bind Shell | Web Shell | Shell Listeners | Payloads. Shells Overview by awesome TryHackMe! 🎉

Oct 24, 2024 101

In T3CH by Axoloth

TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

Nov 5, 2024 60

See more recommendations