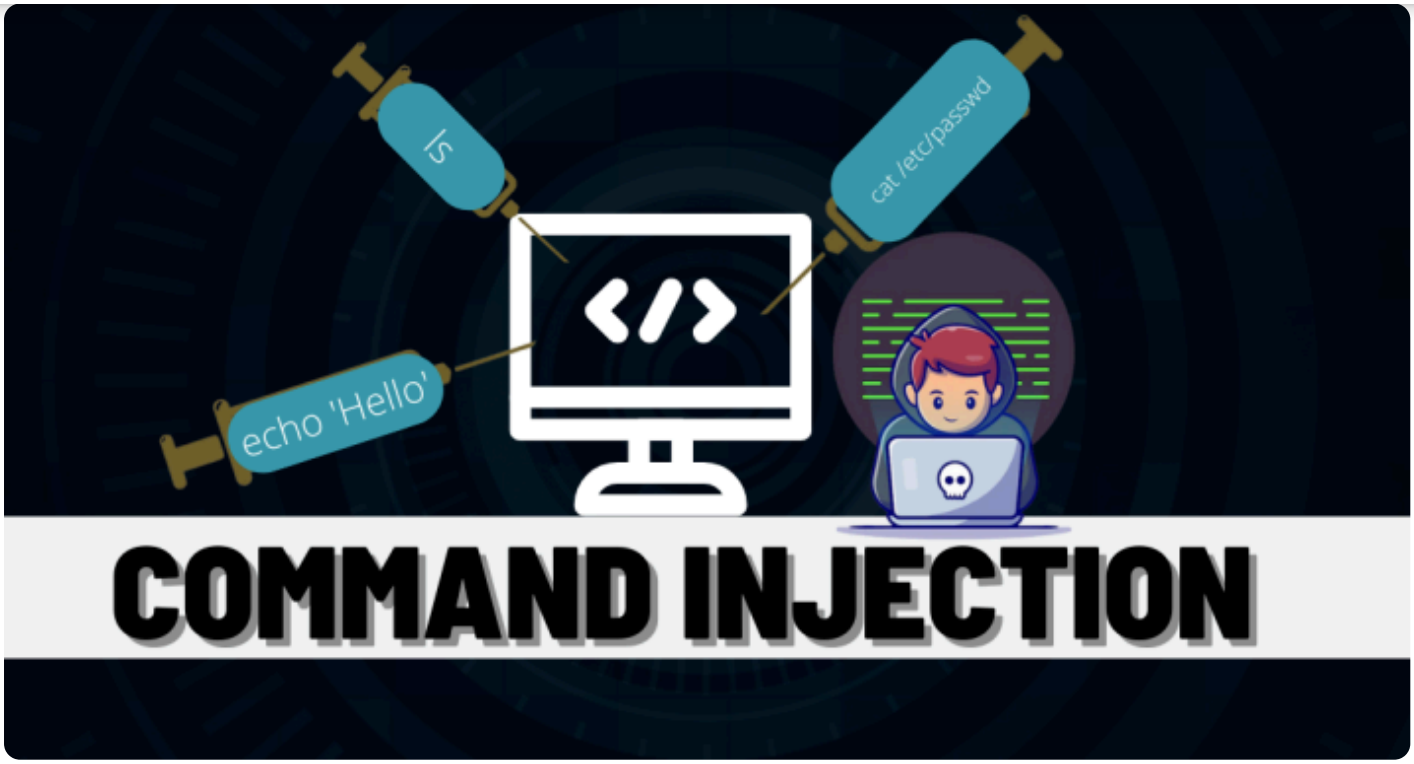


Command Injection (DVWA Series)

Nguyen N.

Cyber Security Analyst I @ Ideal Integrations ||...

Published Dec 9, 2022

Today, we will be covering Command injection. Our goal for today is

- Learn the methodology behind Command Injection
- How to carry out Command Injection Attack
- How do you know if the attack was successful?
- How to detect a Command Injection using Snort
- How to block a Command Injection using Snort

Methodology:

it to pass commands to the system shell without injecting malicious code. In many cases, command injection gives the attacker greater control over the target system.

TLDR: Due to the lack of input validation, the computer fully trusts the user input and will run anything inputted in the form item (user input).

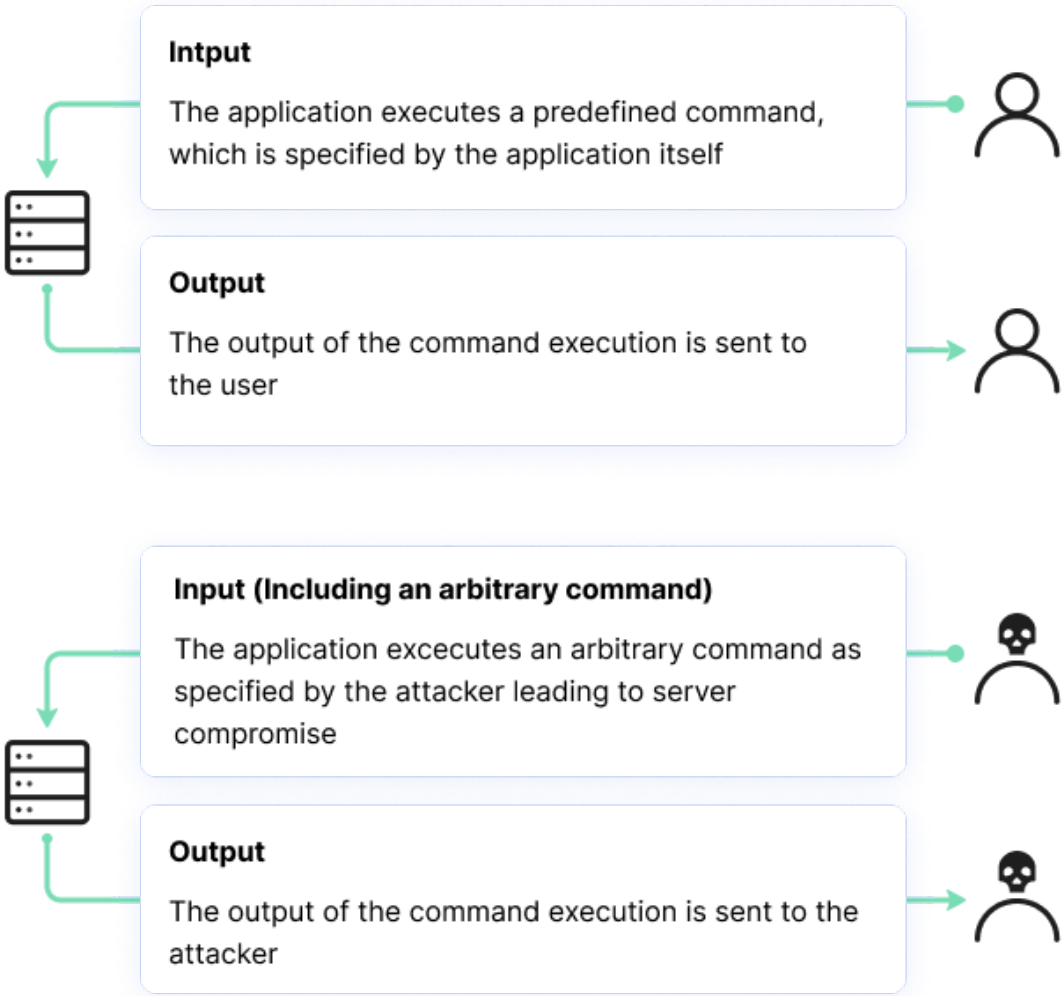
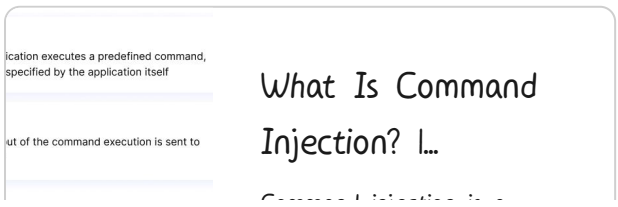


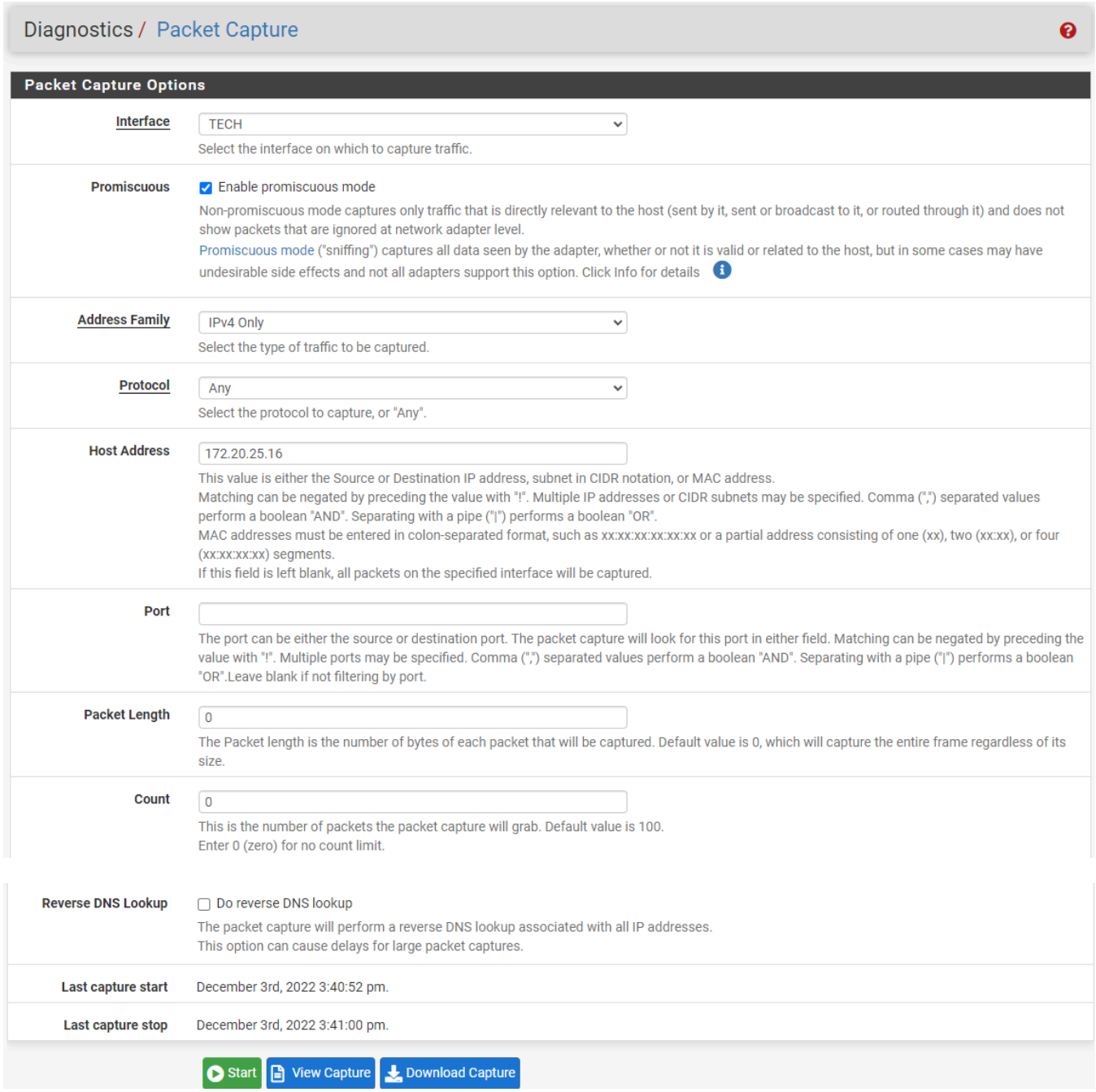
Image Source & Reference:



How to carry out a Command Injection

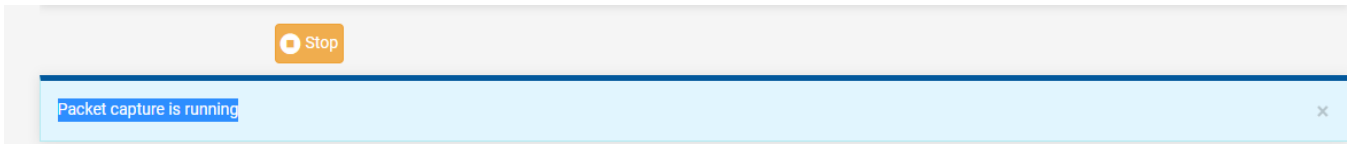
Before we start our attack, let's begin our packet capture in pfsense

172.20.25.16 = our DVWA machine



Click start to start capturing packets.

After hitting start, your screen should look like this:



Let's begin our attack. We will start from the lowest defense and work our way up.'

This is what was given. If we input an IP, it will be sent as an ICMP packet to the IP. Let's give it a try. We will be pinging our DVWA server (172.20.25.16)

Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Vulnerability: Command Injection

Ping a device

Enter an IP address: 172.20.25.16 Submit

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

```
PING 172.20.25.16 (172.20.25.16) 56(84) bytes of data.
64 bytes from 172.20.25.16: icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from 172.20.25.16: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 172.20.25.16: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 172.20.25.16: icmp_seq=4 ttl=64 time=0.041 ms

--- 172.20.25.16 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3104ms
rtt min/avg/max/mdev = 0.013/0.037/0.052/0.014 ms
```

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

Even if we have zero knowledge of PHP, we can see that $target holds the user input. Then, take that input and execute it with "ping -c 4," and then the "echo "<pre>{$cmd}</pre>" will display the value of variable $cmd.

The problem with this is that an Attacker can execute another command within this user prompt by simply using the:

- &&
- ;
- ||
- |

When this command is first executed, apt-get update and then apt-get upgrade.

Another example:

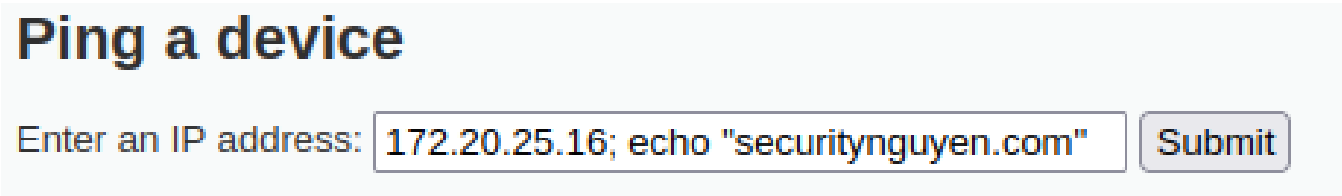
cat ABC.txt | sort

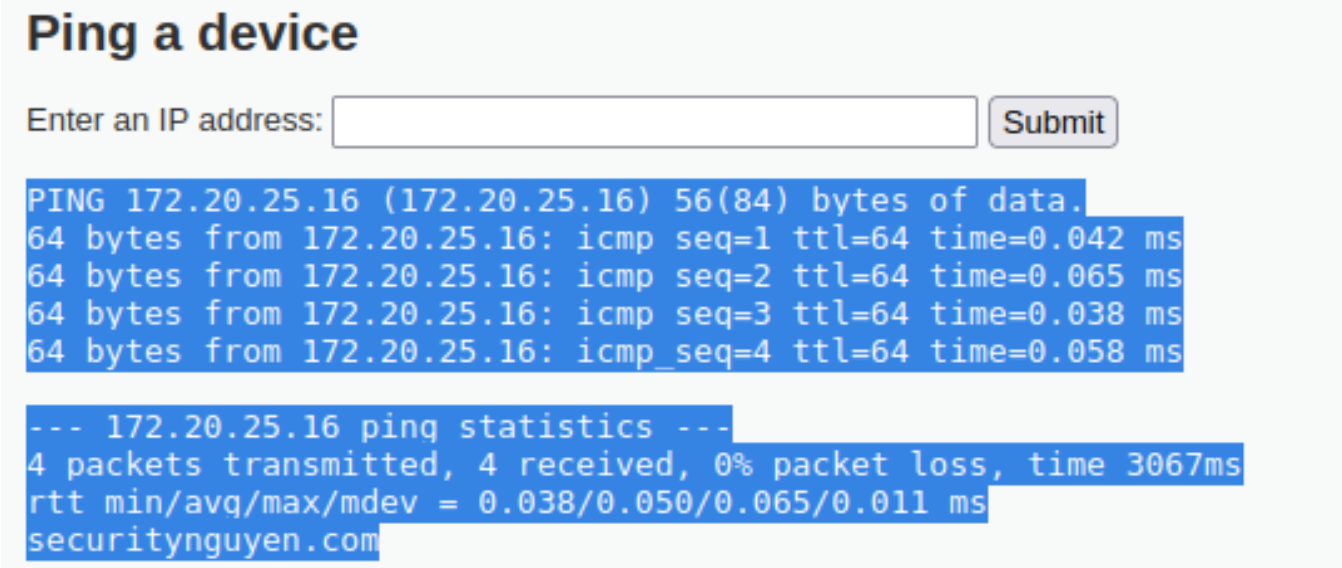
Output:

A
B
C
D

This is simply the same methodology.

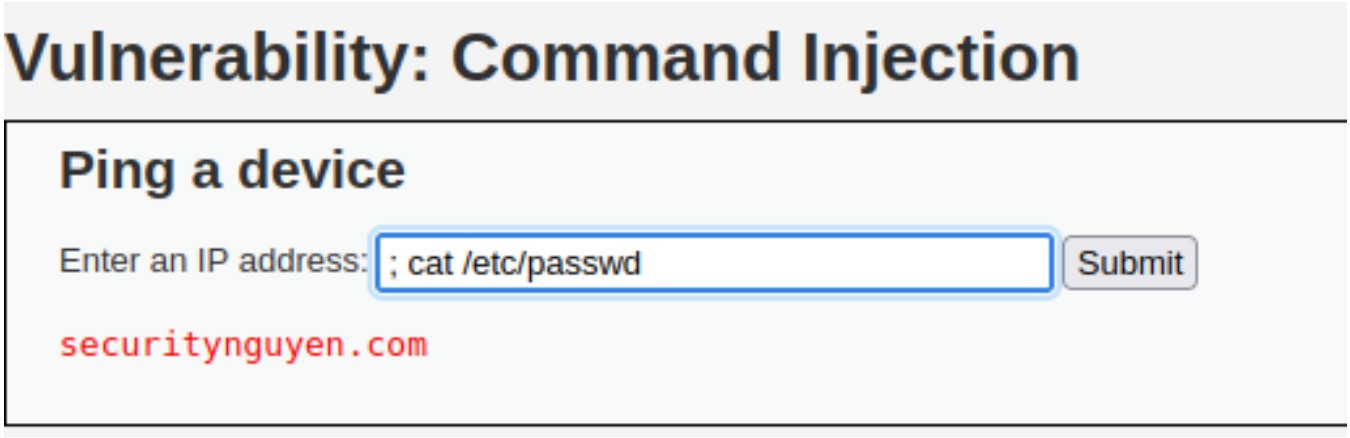
Let me show you an example using the "Ping a device":

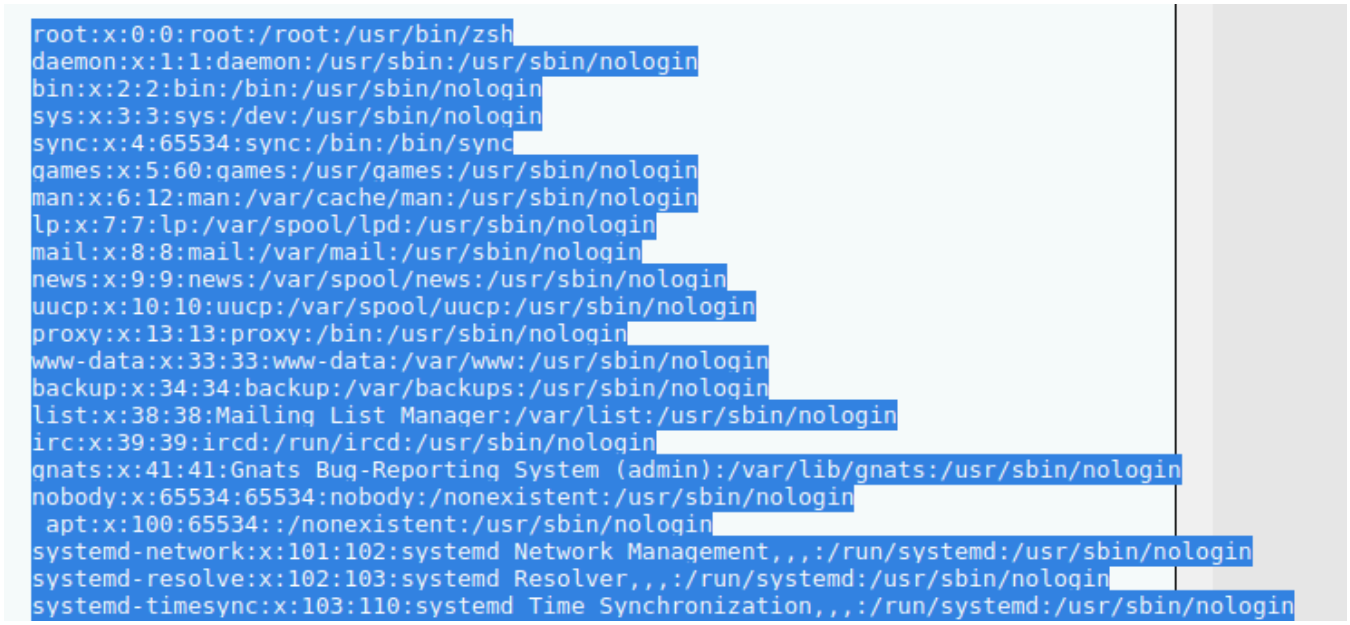




The application ping first the IP and then echo "securitynguyen.com".

- Side note: you don't have to include the IP. You can only include ; (command), and it still works.





We can stop our pcap capture and download the capture:

Make sure the data in the Packets are captured. If that section is empty, please try again.

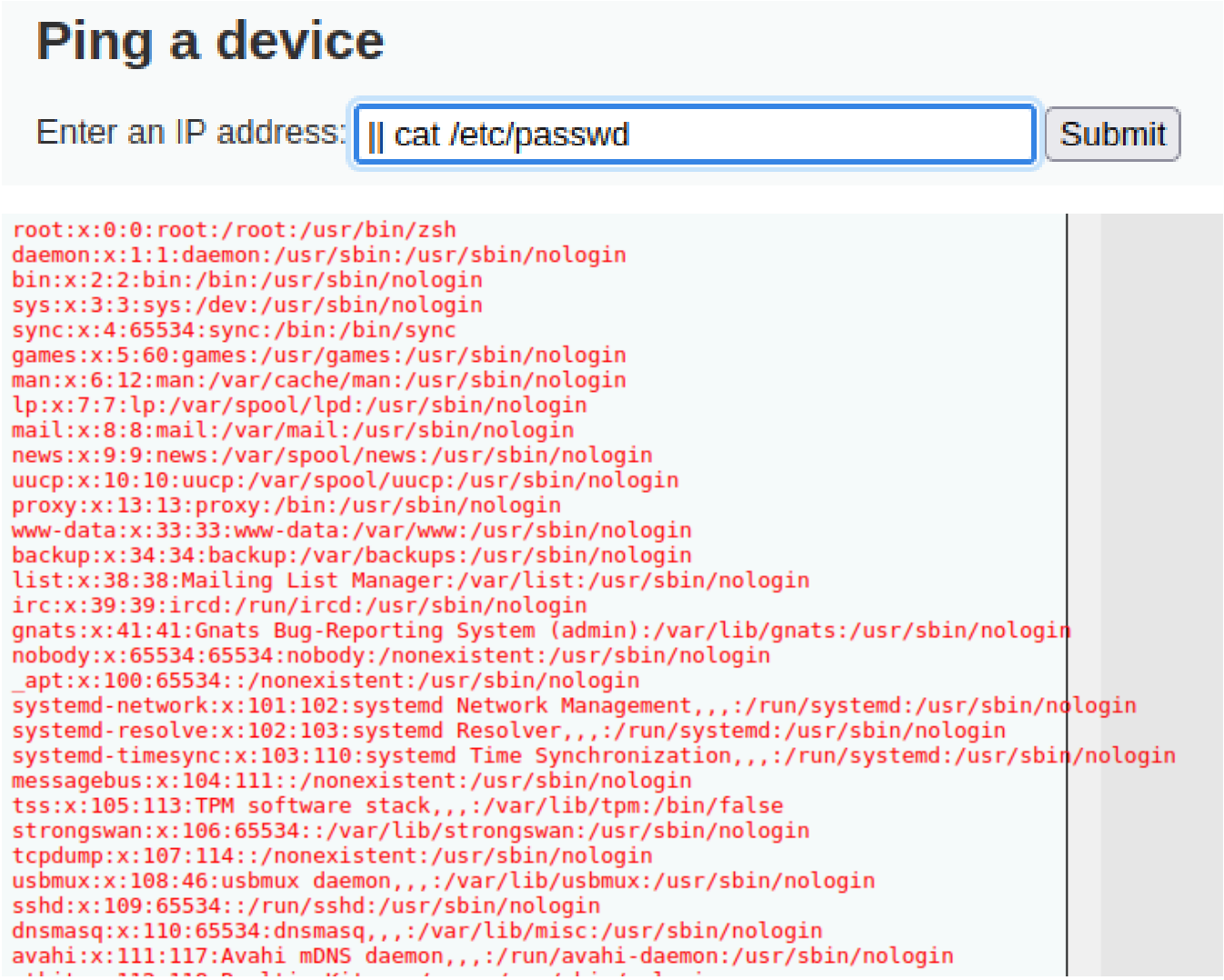
We will focus on that later

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';'  => '',
    );

    // Remove any of the charactars in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitut

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

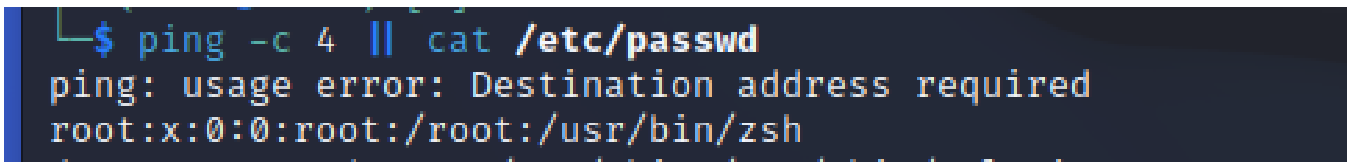
The developer added a section where he blocklisted or "filtered" some of the commands. However, this blocklist doesn't include all the commands the attacker can do. The attacker can still do the following:

- &

For example, we will be showing || cat /etc/passwd



This work because cat /etc/passwd will return a 0 (true) since it successfully executes, and || (or) only runs if either side is true



High (Hard) mode:

Page source:

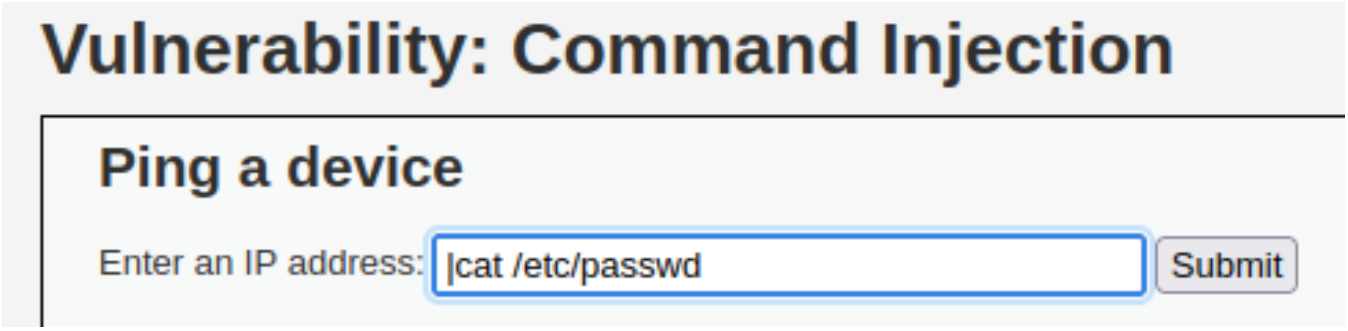
```
<?php
```

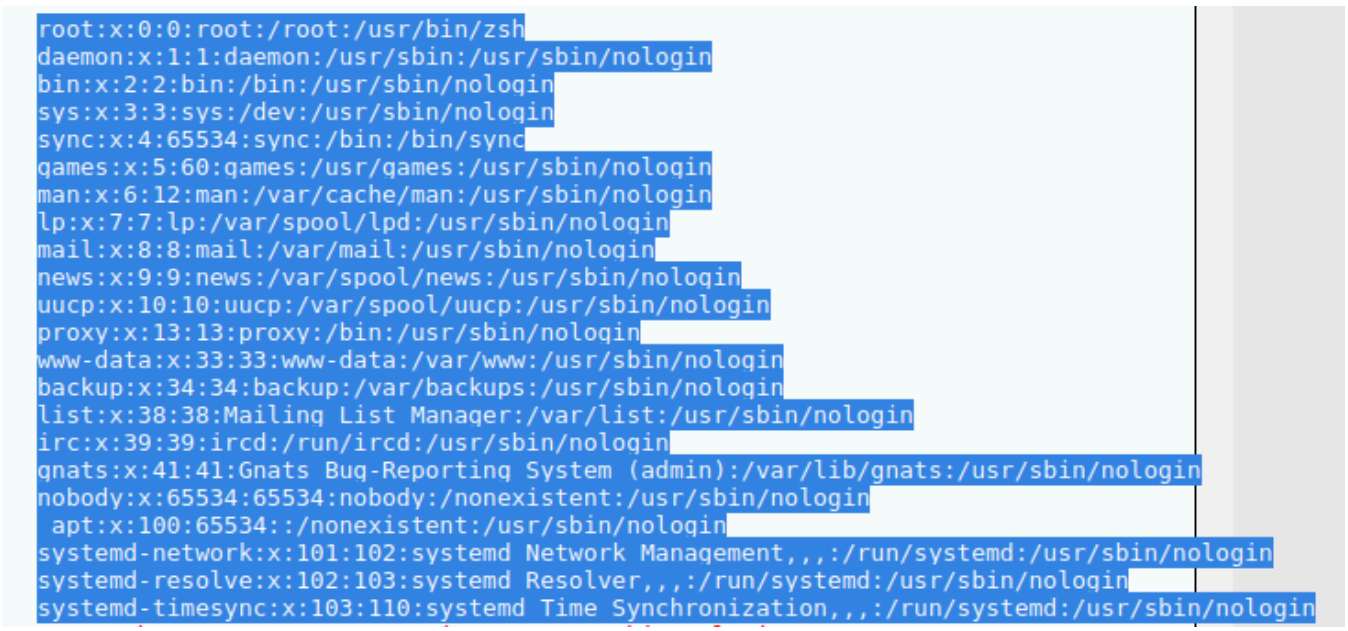
```
        // Set blacklist
        $substitutions = array(
            '&'  => '',
            ';'  => '',
            '| ' => '',
            '-'  => '',
            '$'  => '',
            '('  => '',
            ')'  => '',
            '`'  => '',
            '||' => '',
        );

        // Remove any of the characters in the array (blacklist).
        $target = str_replace( array_keys( $substitutions ), $substitut

        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
            // Windows
            $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
            // *nix
            $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        // Feedback for the end user
        echo "<pre>{$cmd}</pre>";
    }

    ?>
```

The developer added more filters. We can check the hint:

"At the high level, the developer returns to the drawing board and puts in even more patterns to match. But even this isn't enough.

The developer has either made a slight typo with the filters or believes a particular PHP





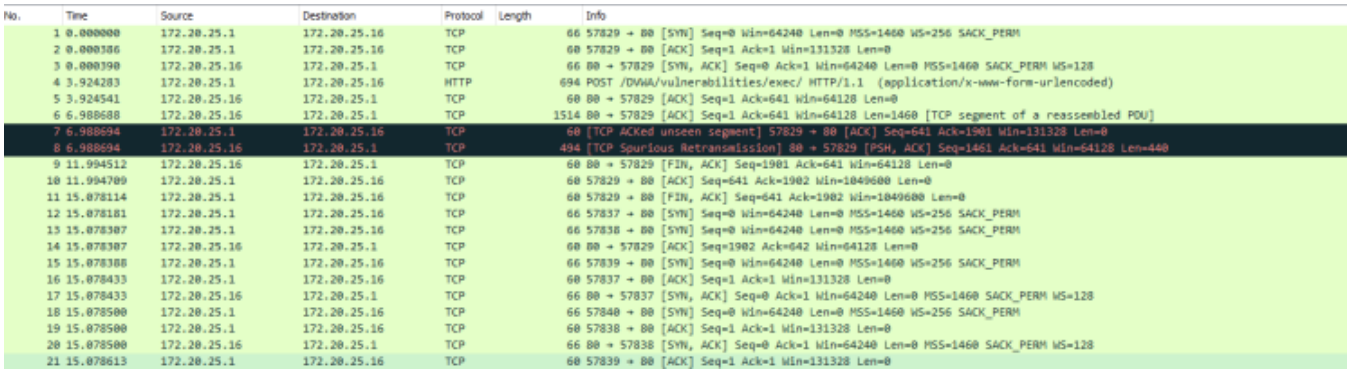
How to know if the attack was successful?:

Previously, we initiated a packet capture during our command injection attack, and we downloaded our capture to our computer. The file should be in the downloads folder, and the file should be named packetcapture.cap

We will transform it into a pcap to view it on Wireshark.

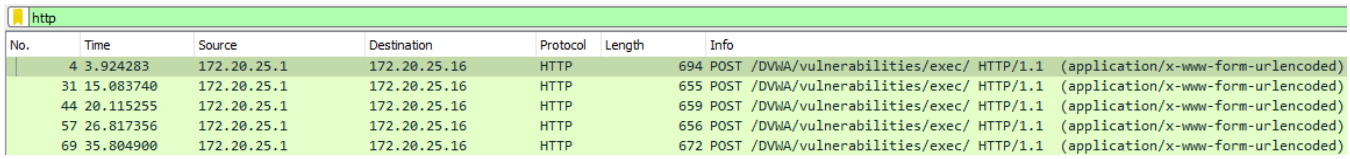
First, right-click on the packetcapture.cap -> Rename -> Command_Injection.pcap

Boom, there we go. It's in a pcap format. Let's open it in our Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.20.25.1	172.20.25.16	TCP	66	57829 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
2	0.000386	172.20.25.1	172.20.25.16	TCP	60	57829 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
3	0.000390	172.20.25.16	172.20.25.1	TCP	66	80 → 57829 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
4	3.924283	172.20.25.1	172.20.25.16	HTTP	694	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)
5	3.924541	172.20.25.16	172.20.25.1	TCP	60	80 → 57829 [ACK] Seq=1 Ack=641 Win=64128 Len=0
6	6.988688	172.20.25.16	172.20.25.1	TCP	1514	80 → 57829 [ACK] Seq=1 Ack=641 Win=64128 Len=1460 [TCP segment of a reassembled PDU]
7	6.988694	172.20.25.1	172.20.25.16	TCP	60	[TCP ACKed unseen segment] 57829 → 80 [ACK] Seq=641 Ack=1901 Win=131328 Len=0
8	6.988694	172.20.25.16	172.20.25.1	TCP	494	[TCP Spurious Retransmission] 80 → 57829 [PSH, ACK] Seq=1461 Ack=641 Win=64128 Len=440
9	11.994512	172.20.25.16	172.20.25.1	TCP	60	80 → 57829 [FIN, ACK] Seq=1901 Ack=641 Win=64128 Len=0
10	11.994709	172.20.25.1	172.20.25.16	TCP	60	57829 → 80 [ACK] Seq=641 Ack=1902 Win=1049600 Len=0
11	15.078114	172.20.25.1	172.20.25.16	TCP	60	57829 → 80 [FIN, ACK] Seq=641 Ack=1902 Win=1049600 Len=0
12	15.078181	172.20.25.1	172.20.25.16	TCP	66	57837 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
13	15.078307	172.20.25.1	172.20.25.16	TCP	66	57838 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
14	15.078307	172.20.25.16	172.20.25.1	TCP	60	80 → 57829 [ACK] Seq=1902 Ack=642 Win=64128 Len=0
15	15.078388	172.20.25.1	172.20.25.16	TCP	66	57839 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
16	15.078433	172.20.25.1	172.20.25.16	TCP	60	57837 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
17	15.078433	172.20.25.16	172.20.25.1	TCP	66	80 → 57837 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
18	15.078500	172.20.25.1	172.20.25.16	TCP	66	57840 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
19	15.078500	172.20.25.1	172.20.25.16	TCP	60	57838 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
20	15.078500	172.20.25.16	172.20.25.1	TCP	66	80 → 57838 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
21	15.078613	172.20.25.1	172.20.25.16	TCP	60	57839 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0

This is what our pcap file looked like when first opened. Let's filter it to HTTP to see what we get:



http

No.	Time	Source	Destination	Protocol	Length	Info
4	3.924283	172.20.25.1	172.20.25.16	HTTP	694	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)
31	15.083740	172.20.25.1	172.20.25.16	HTTP	655	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)
44	20.115255	172.20.25.1	172.20.25.16	HTTP	659	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)
57	26.817356	172.20.25.1	172.20.25.16	HTTP	656	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)
69	35.804900	172.20.25.1	172.20.25.16	HTTP	672	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-www-form-urlencoded)

As we can see, We are getting POST requests from the attacker (172.20.25.1) to the server.

POST sends data to a server to create/update a resource.

Down below named: "HTML Form URL Encoded: Application/x-www-form-urlencoded"

```
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▼ Form item: "ip" = "172.20.25.16; echo "securitynguyen.com""
      Key: ip
      Value: 172.20.25.16; echo "securitynguyen.com"
  ▼ Form item: "Submit" = "Submit"
      Key: Submit
      Value: Submit
```

We can see the attacker's command and what he executed. Let's follow the TCP stream to see what we get.

```
POST /DVWA/vulnerabilities/exec/ HTTP/1.1
Host: 172.20.25.16
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/
102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 62
Origin: http://172.20.25.16
Connection: keep-alive
Referer: http://172.20.25.16/DVWA/vulnerabilities/exec/
Cookie: PHPSESSID=k190f9gr2plc2htl0d6boj8qqh; security=low
Upgrade-Insecure-Requests: 1

ip=172.20.25.16%3B+echo+%22securitynguyen.com%22&Submit=SubmitHTTP/1.1 200 OK
Date: Thu, 08 Dec 2022 11:38:04 GMT
Server: Apache/2.4.54 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1548
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

Red = Client, Blue = Server.

We see that the IP variable in the POST request is holding the attacker command. If we decode this, we will see the command perfectly clearly. Let's do this

```
ip=172.20.25.16; echo "securitynguyen.com"&Submit=Submit
```

Look at that: we can see what our adversary executed on our server.

```
ip=172.20.25.16%3B+echo+%22securitynguyen.com%22&Submit=SubmitHTTP/1.1 200 OK
Date: Thu, 08 Dec 2022 11:38:04 GMT
Server: Apache/2.4.54 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1548
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

Our standard IP request:

```
ip=172.20.25.16&Submit=SubmitHTTP/1.1 200 OK
Date: Thu, 08 Dec 2022 11:59:05 GMT
Server: Apache/2.4.54 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1543
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

As we can see, a standard IP request is 1543, but a command injection is 1548. An increase in content length can indicate that an attack has been successful. But we shouldn't only use the content length as a be-all indicator but the form item to see if there was a command injection inside of it.

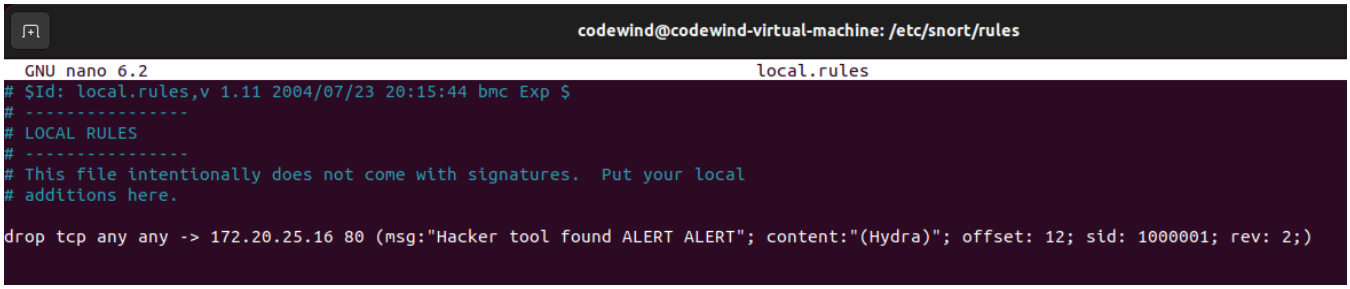
How to detect a Command Injection using snort:

- Look for keywords related to the terminal language: Check the data received from the user for keywords that are related to terminal commands such as dir, ls, cp, cat, type, etc.

- Please familiarize yourself with frequently used Command Injection payloads: When attackers detect a command injection vulnerability, they usually create a reverse shell in order to work more efficiently. Knowing frequently used Command Injection payloads will make seeing a command injection attack easier.

Let's get right to it.

```
cd /etc/snort/rules
sudo nano local.rules
```

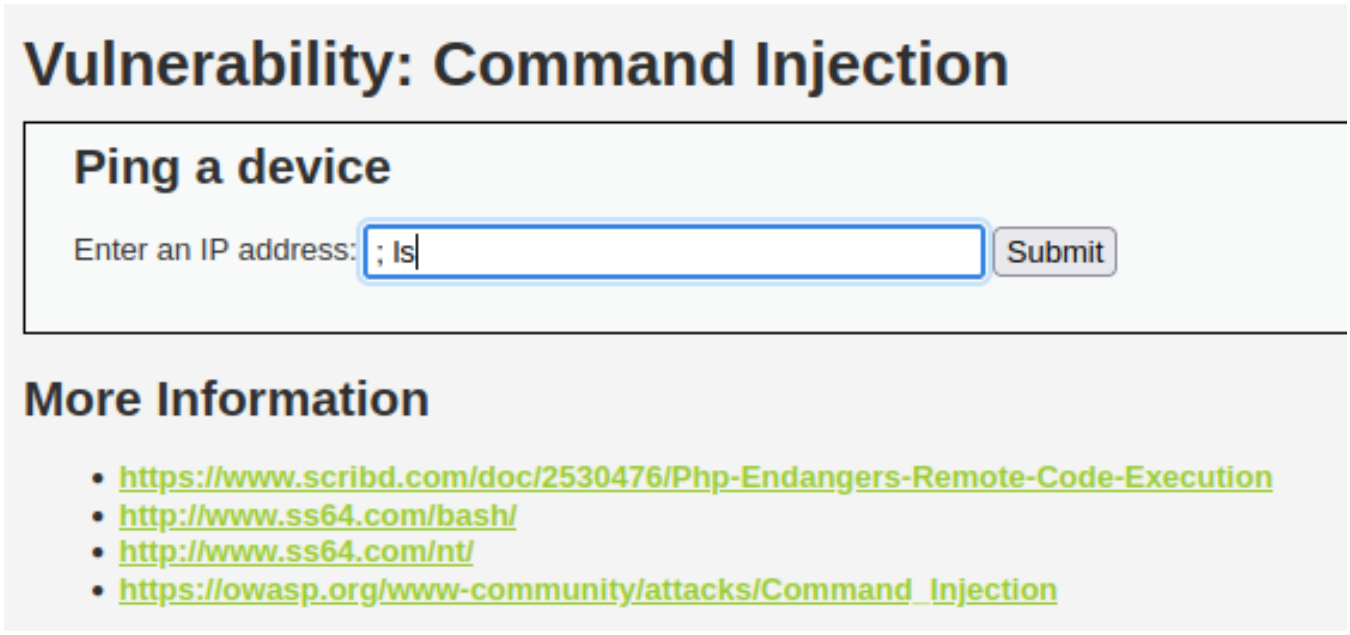


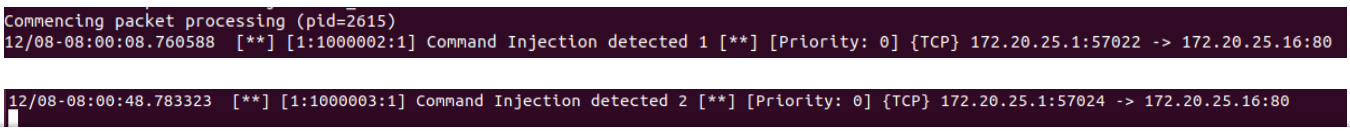
Here are the new rules I created:

```
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detect
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detect
alert tcp any any -> 172.20.25.16 80 (msg:"Command injection detect
```

Let's test them out to see if they work:

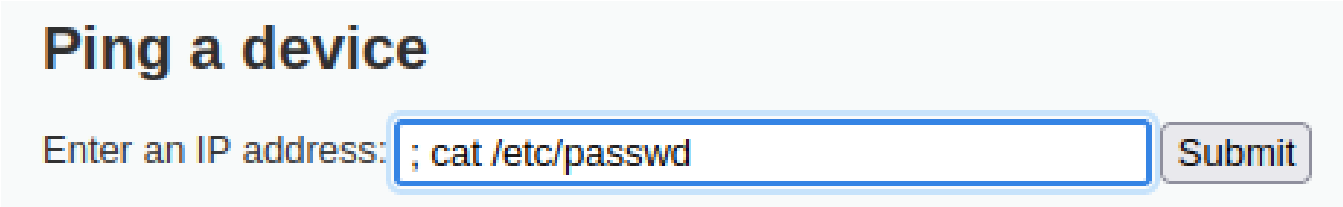
```
sudo snort -c /etc/snort/snort.conf -A console
```

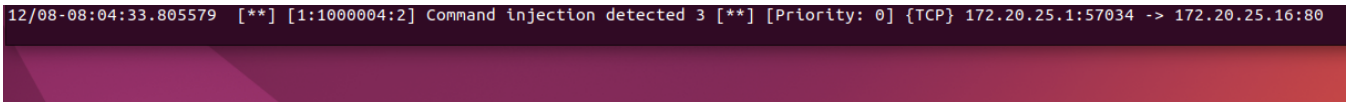




For some reason, it wouldn't detect cat /etc/passwd. Let's change the rule to just cat. Remember to edit the rev whenever you edit one of your rules.

```
#Command Injection Alert
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 1"; content:"ls"; sid: 1000002; rev: 1;)
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 2"; content:"whoami"; sid: 1000003; rev: 1;)
alert tcp any any -> 172.20.25.16 80 (msg:"Command injection detected 3"; content:"cat"; sid: 1000004; rev: 2;)
```





How to block a Command Injection using Snort

We will use the previously created rules and just add the keyword drop to them.

Let's get started:

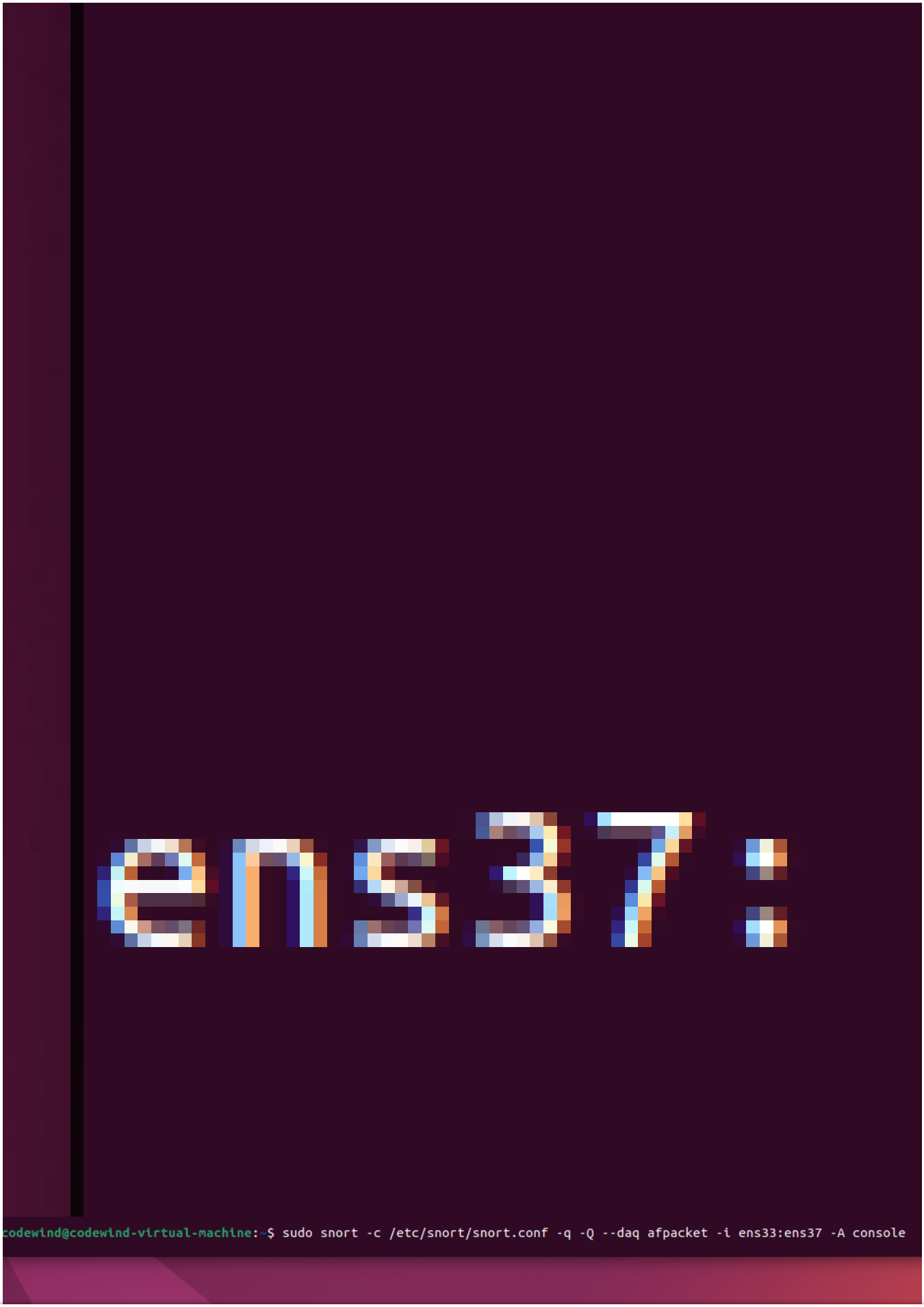
sudo nano local.rules

```
drop tcp any any -> 172.20.25.16 80 (msg:"Command Injection detecte
drop tcp any any -> 172.20.25.16 80 (msg:"Command Injection detecte
drop tcp any any -> 172.20.25.16 80 (msg:"Command injection detecte
```

```
Command Injection Alert
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 1"; content:"ls"; sid: 1000002; rev: 1;)
alert tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 2"; content:"whoami"; sid: 1000003; rev: 1;)
alert tcp any any -> 172.20.25.16 80 (msg:"Command injection detected 3"; content:"cat"; sid: 1000004; rev: 2;)
rop tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 1"; content:"ls"; sid: 1000002; rev: 1;)
rop tcp any any -> 172.20.25.16 80 (msg:"Command Injection detected 2"; content:"whoami"; sid: 1000003; rev: 1;)
rop tcp any any -> 172.20.25.16 80 (msg:"Command injection detected 3"; content:"cat"; sid: 1000004; rev: 2;)
```

Activate snort in IPS mode:

```
sudo snort -c /etc/snort/snort.conf -q -Q --daq afpacket -i ens33:e
```

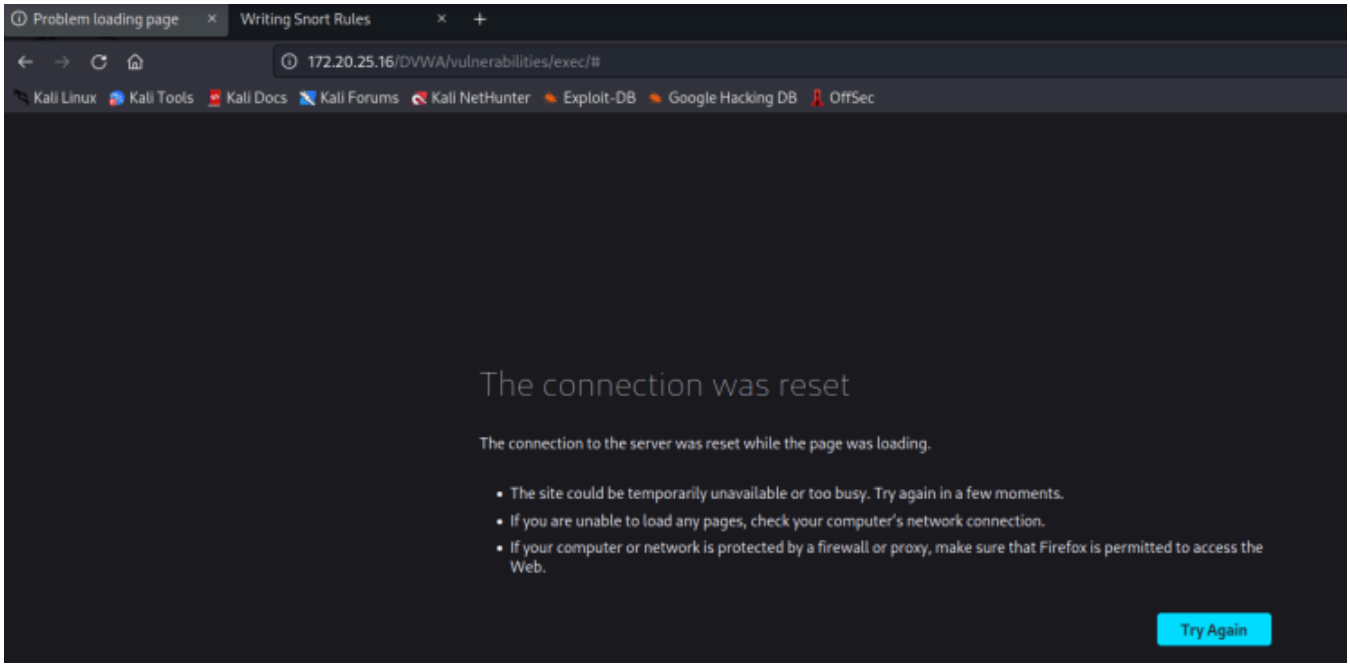


4 · 1 Comment

Like Comment

Let's launch our attack.

As you can see, the packet is dropped.



Divyanshu Maurya

Vice captain @DND || Cybersecurity enthusiast || Member of Udaan Aeromodelling Club || HIS Core team

1m

Very helpful

Like · Reply

To view or add a comment, sign in

More articles by Nguyen N.

Feb 5, 2023

Personal philosophy on life

3 · 1 Comment

Feb 5, 2023

Personal Careers Goals

4 · 1 Comment

Like Comment