IPTIME DBG

문서번호 Ver. 1.0

개 정 이 력

버전	작성일	변경사유[1]	변경내용[2]	작성자	승인자
1.0	2022.01.12	최초 작성		정형수	

[1] 변경 사유: 변경 내용이 이전 문서에 대해 최초작성/승인/추가/수정/삭제 중 선택 기입

[2] 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장.절과 변경 내용을 기술한다.)

목 차

1. 개요

1.1 본 문서의 목적

본 문서는 IOT 진단을 수행할 경우, 신규 취약점에 대한 정보 및 점검 절차를 기술하기 위한 문서로, 신규 입사자와 IOT 를 처음 수행하는 자에게 쉽게 따라 할 수 있도록 하는 것이 그 목적이다.

[분석 도구]

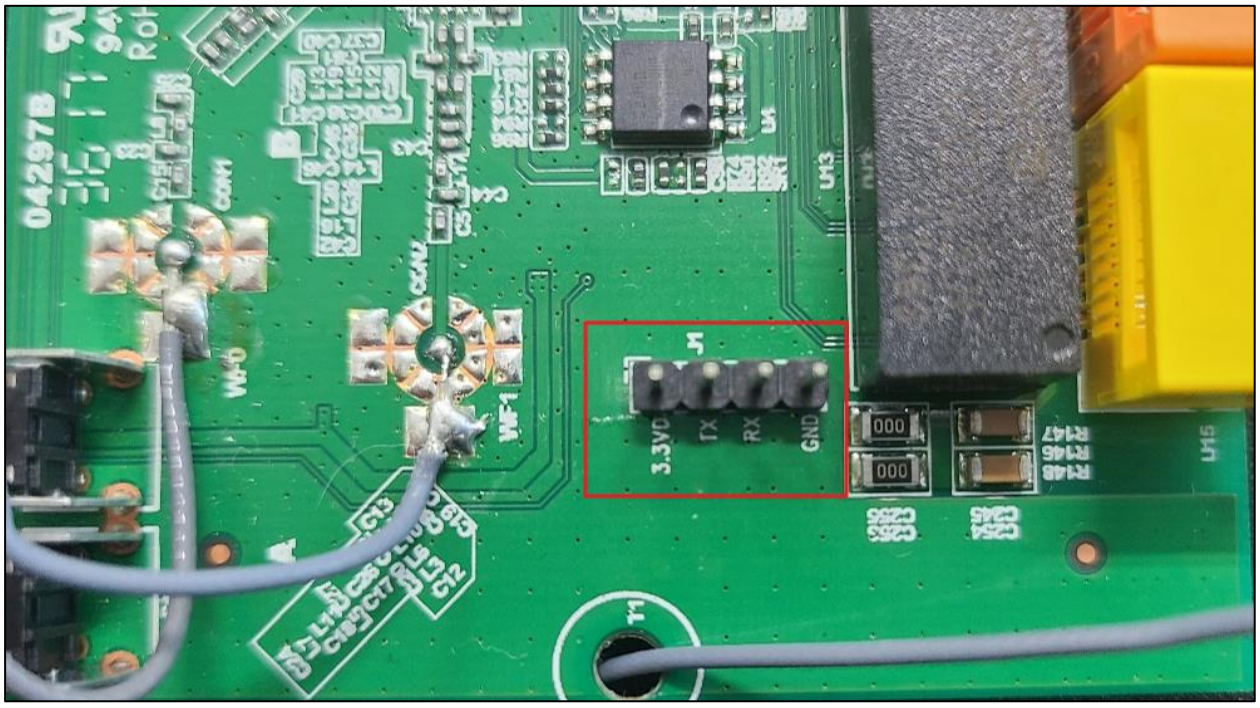
분석도구	버전	기능
IPTIME	ipTIME N604 black 9.98.2	분석대상
Kali linux	v2021-4a	분석 환경 OS
binwalk	v2.2.1	펌웨어 이미지 내의 파일과 코드를 추출
FirmAE	-	펌웨어 에뮬레이션 구동 자동화 프레임워크
gef	-	리버스 엔지니어링 도구
gdb-multiarch	v3	크로스컴파일러 아키텍처 분석 도구
AFL fuzzer	2.52b	Fuzzer 도구

2. UART

UART 는 병렬 데이터의 형태를 직렬 방식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종이다. UART 핀을 찾아 얻을 수 있는 데이터는 커널, OS 메시지, 디버그 메시지, 오류 메시지, 숨겨진 메뉴, Command Shell 등이 있다. UART 핀을 찾는 방법은 여러가지 방법이 존재하지만 본 문서에서는 ipTIME 을 기준으로 설명한다.

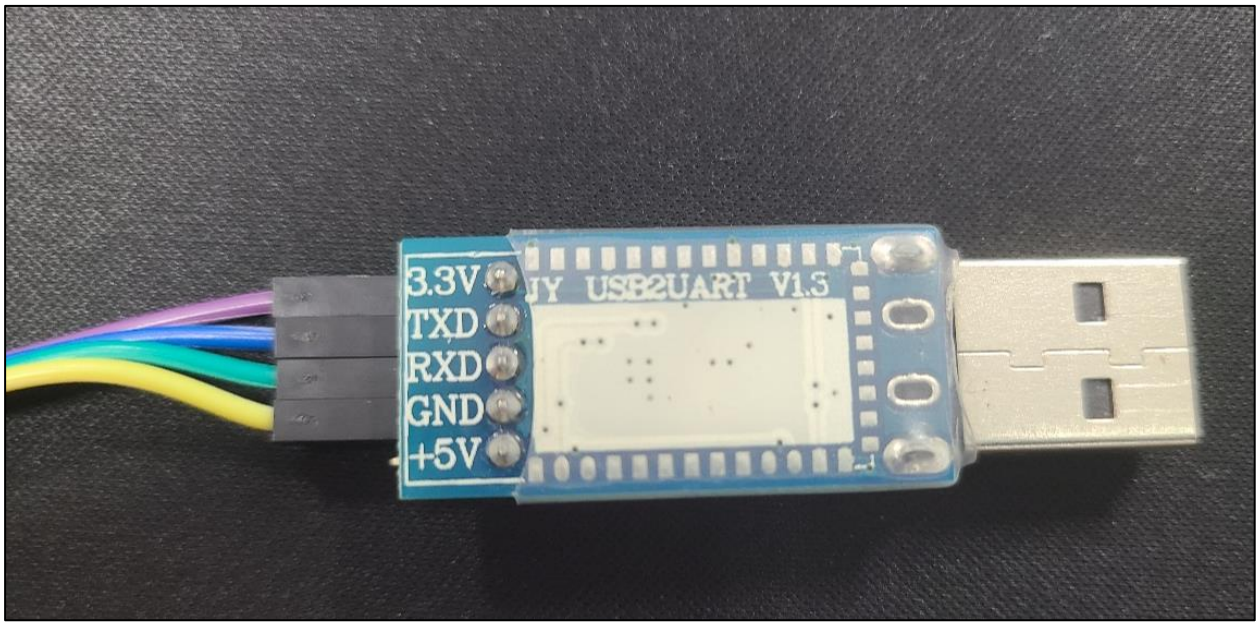
2.1 USB to UART

진단 시 분석대상이 될 ipTIME 과 USB to UART 가 필요하다. USB to UART 와 점퍼는 ipTIME 의 UART 포트에 연결하여 PC 와 연결하기 위해서 반드시 구비가 되어 있어야 한다.

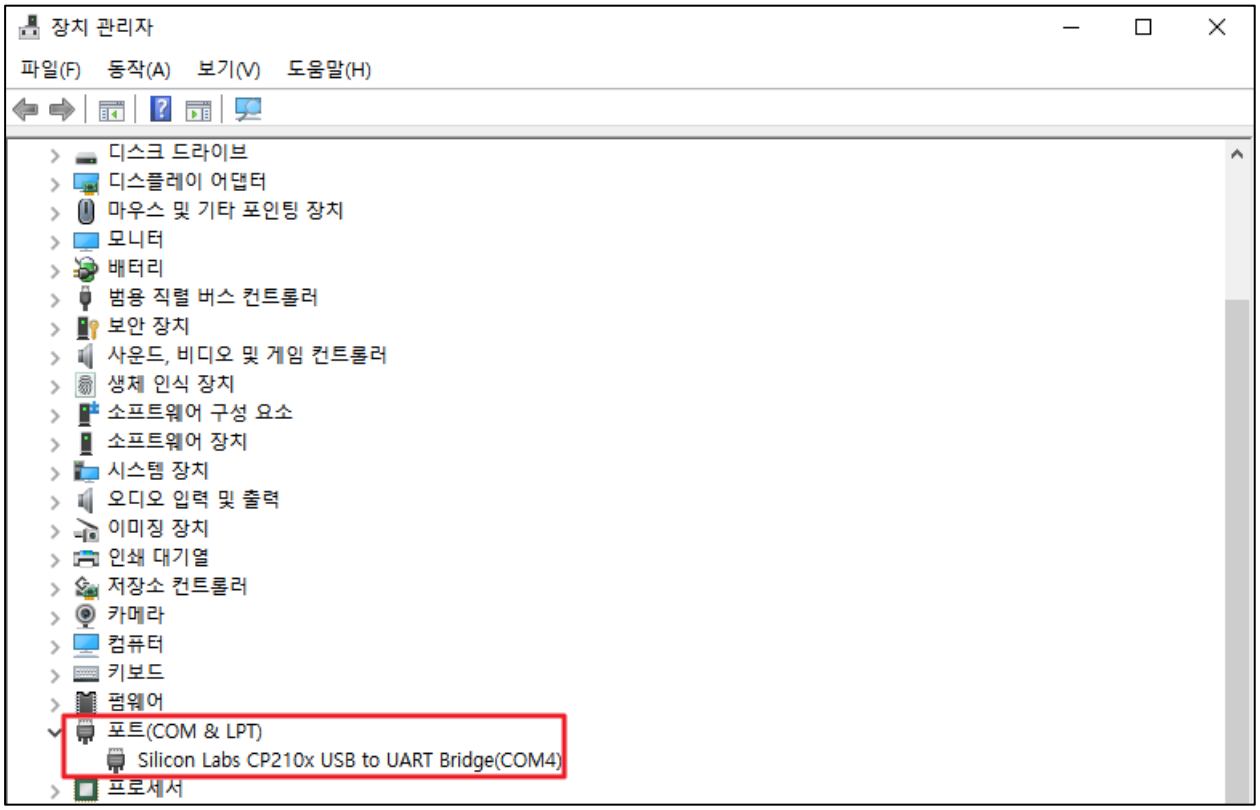


ipTIME 의 경우 UART 핀이 제거되지 않아 분해 시 쉽게 찾아볼 수 있다. UART 핀은 3.3VD, TX, RX, GND 로 구성되어 있으며 각 핀의 역할은 다음과 같다.

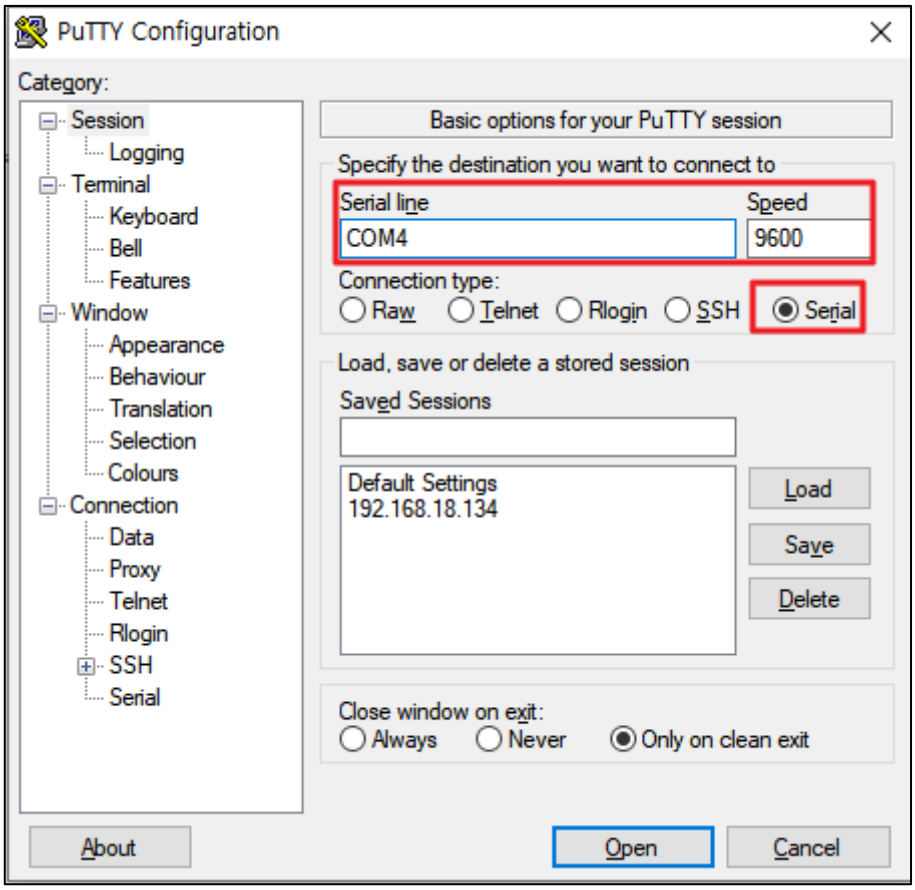
3.3VD : 3.3VCC 전압, **TX** : 데이터 송신 핀, **RX** : 데이터 수신 핀, **GND** : 그라운드 핀



USB to UART 변환기에도 UART 각 핀에 대한 설명이 있으며 점퍼를 이용하여 ipTIME 과 연결하면 된다. 연결 시 에는 ipTIME 과 PC 간의 통신이 이루어 지도록 TX 와 RX 핀을 교차하여 연결해 준다.
ex) TX → RX 연결, RX → TX 연결



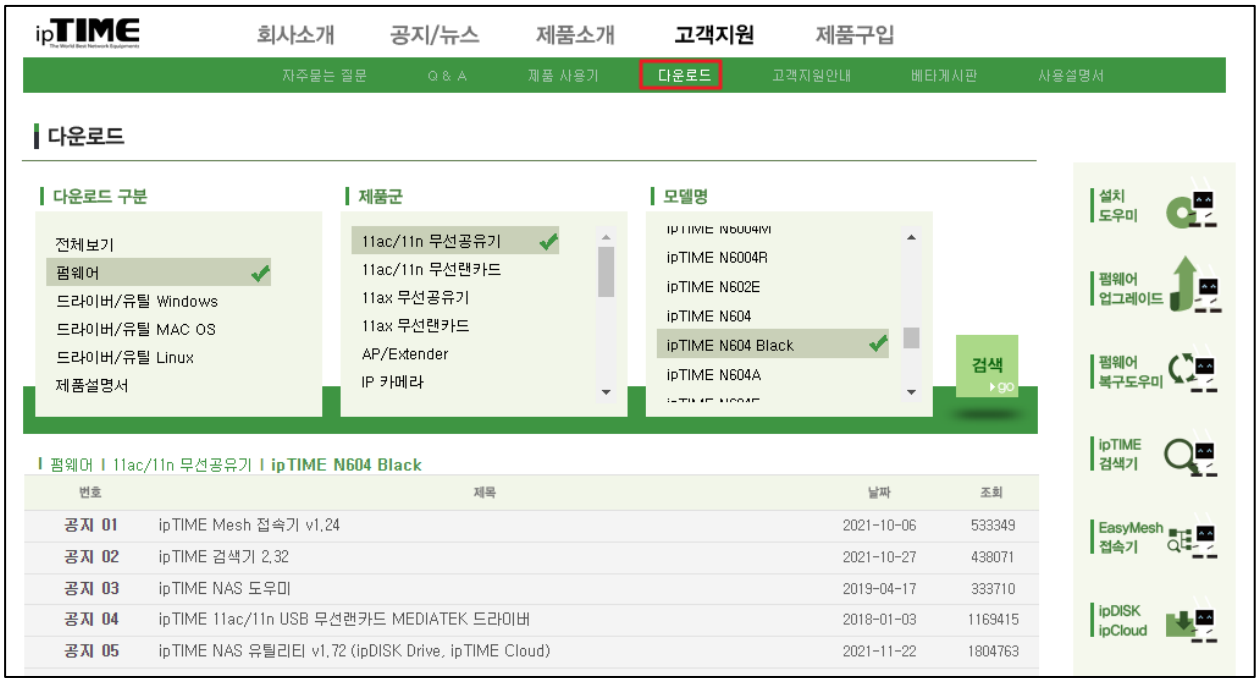
케이블을 연결한 이후에는 장치관리자에서 USB 포트가 인식이 되는지 확인하고 인식이 되지 않는 경우 CP2102, PL2303, FTDI 등과 같은 USB 드라이버를 설치해야 한다.



연결 이후 원격접속 프로그램을 이용하여 시리얼 포트로 접속을 시도한다. 접속할때 사용되는 Speed 는 Baudrate, Serial Port 라고도 불리며 해당 값은 무작위대입을 통해 구해야 한다.
시리얼 포트는 115200, 57600, 38400, 19200, 9600 을 사용한다.

```
COM4 - PuTTY
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00002000
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 8128
Kernel command line: console=ttyS1,57600n8 root=/dev/mtdblock5 console=
PID hash table entries: 128 (order: -3, 512 bytes)
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Primary instruction cache 64kB, VIPT, , 4-waylinesize 32 bytes.
Primary data cache 32kB, 4-way, PIPT, no aliases, linesize 32 bytes
Writing ErrCtl register=00028cf8
Readback ErrCtl register=00028cf8
Memory: 28152k/32768k available (3163k kernel code, 4564k reserved, 712k data, 144k init, 0k highmem)
NR_IRQS:128
console [ttyS1] enabled
Calibrating delay loop... 381.95 BogoMIPS (lpj=763904)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
Switching to clocksource Ralink Systick timer
NET: Registered protocol family 2
-->  ipconflict proc OK !!!!!!!!!!
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 1, 8192 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
```

해당 ipTIME 은 57600 포트로 연결되었으며 연결 시 커널, OS 메시지가 무한루프되어 커멘드 쉘을 획득할수 없는 상태이다. 하드웨어는 중지되지 않는 무한루프의 특징을 가진다.



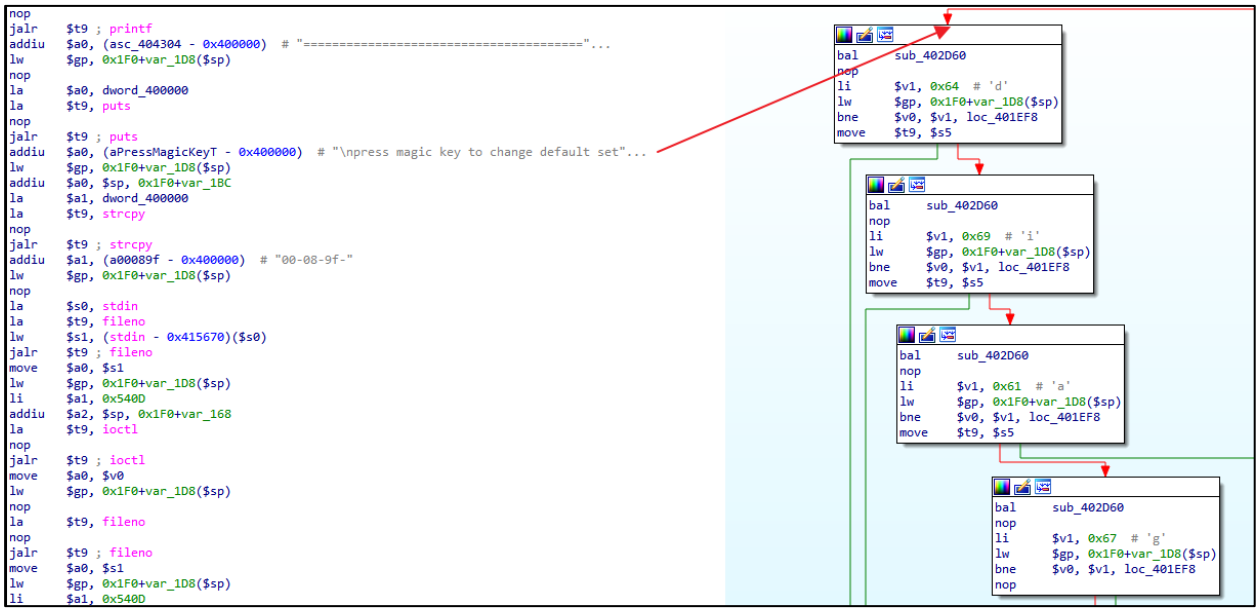
ipTIME은 무한루프에서 빠져나오기 위해 magic key값이 존재한다. 해당 key값을 확인하기 위해 공식 홈페이지에서 OS 메시지에서 확인된 해당 버전의 동일 펌웨어를 다운받는다.

```
code@code-virtual-machine:~/FirmAE$ ls -al | grep n604bl
-rw-rw-r--  1 code code  3441916  1월 10 15:04 n604bl_kr_9_982.bin
code@code-virtual-machine:~/FirmAE$ binwalk -e ./n604bl_kr_9_982.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0xC4EC2D86, created: 2016-12-13 09:20:56, image size: 3441
852 bytes, Data Address: 0x80000000, Entry Point: 0x8000C120, data CRC: 0x8DE743AC, OS: Linux, CPU: MIPS, image type: OS Kernel Image, com
pression type: lzma, image name: "n604bl"
64            0x40            LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 4119628 bytes

WARNING: Extractor.execute failed to run external extractor 'sasquatch -p 1 -le -d 'squashfs-root-0' '%e'': [Errno 2] No such file or dire
ctory: 'sasquatch', 'sasquatch -p 1 -le -d 'squashfs-root-0' '%e'' might not be installed correctly

WARNING: Extractor.execute failed to run external extractor 'sasquatch -p 1 -be -d 'squashfs-root-0' '%e'': [Errno 2] No such file or dire
ctory: 'sasquatch', 'sasquatch -p 1 -be -d 'squashfs-root-0' '%e'' might not be installed correctly
1344764       0x1484FC        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 2094554 bytes, 1261 inodes, blocksize
: 131072 bytes, created: 2016-12-13 09:20:53

code@code-virtual-machine:~/FirmAE$
code@code-virtual-machine:~/FirmAE$ ls -al | grep n604bl
-rw-rw-r--  1 code code  3441916  1월 10 15:04 n604bl_kr_9_982.bin
drwxrwxr-x  4 code code     4096  1월 13 09:20 _n604bl_kr_9_982.bin.extracted
```

다운르도한 펌웨어는 binwalk를 이용하여 코드와 파일을 추출한다.

```
code@code-virtual-machine:~/FirmAE/_n604bl_kr_9_982.bin.extracted/squashfs-root$ pwd
/home/code/FirmAE/_n604bl_kr_9_982.bin.extracted/squashfs-root
code@code-virtual-machine:~/FirmAE/_n604bl_kr_9_982.bin.extracted/squashfs-root$ ls
bin  cgibin  default  dev  etc  etc_ro  home  lib  linuxrc  ndbin  plugin  proc  save  sbin  sys  tmp  upgrade-bin  usr  var
code@code-virtual-machine:~/FirmAE/_n604bl_kr_9_982.bin.extracted/squashfs-root$
```

추출하게 되면 "/_n604bl_kr_9_982.bin.extracted/squashfs-root" 위치에 파일이 추출된 것을 확인할 수 있다.



IDA 를 이용하여 /sbin/inittime 파일 분석 시 magic key 값을 "diag"로 받는 것을 확인할 수 있다.

3. AFL Fuzzer

AFL 은 American Fuzzy Lop 의 약자로, instrumentation-guied genetic algorithm 과 결합된 fuzzer 이다. 브루트포스로 입력을 받으며 거기서 끝나는 것이 아니라, 커버리지를 넓혀가며 프로그램 제어 프름에 대한 변경사항을 기록하고, 이를 로깅하여 유니크한 크래시를 발견해 낼 수 있다. 또한 AFL Fuzzer 는 커버리지 기반 fuzzer 이기 때문에 매우 효율적인 퍼징이 가능하며, 코드 커버리지 측정을위한 코드를 컴파일 타임에 삽입한다. QEMU 를 이용하여 컴파일 타임이 아닌, 런타임시에 코드 삽입도 가능하다.(코드 삽입을 Instrumentation 이라고 한다.)

3.1 AFL Logic

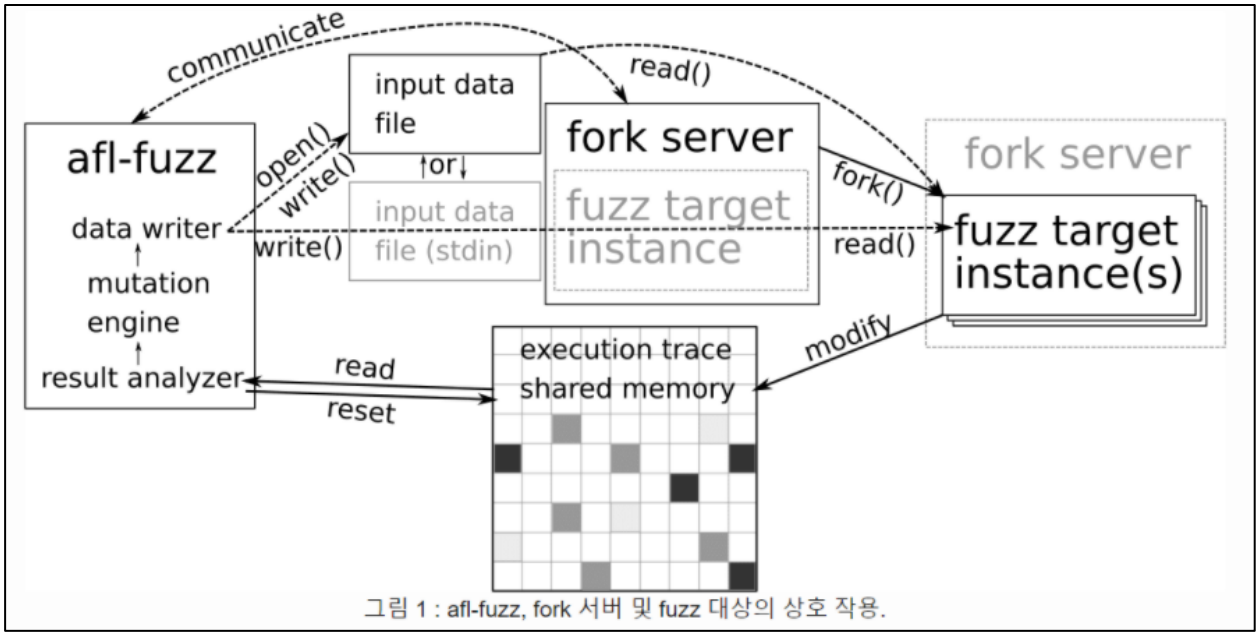


그림 1 : afl-fuzz, fork 서버 및 fuzz 대상의 상호 작용.

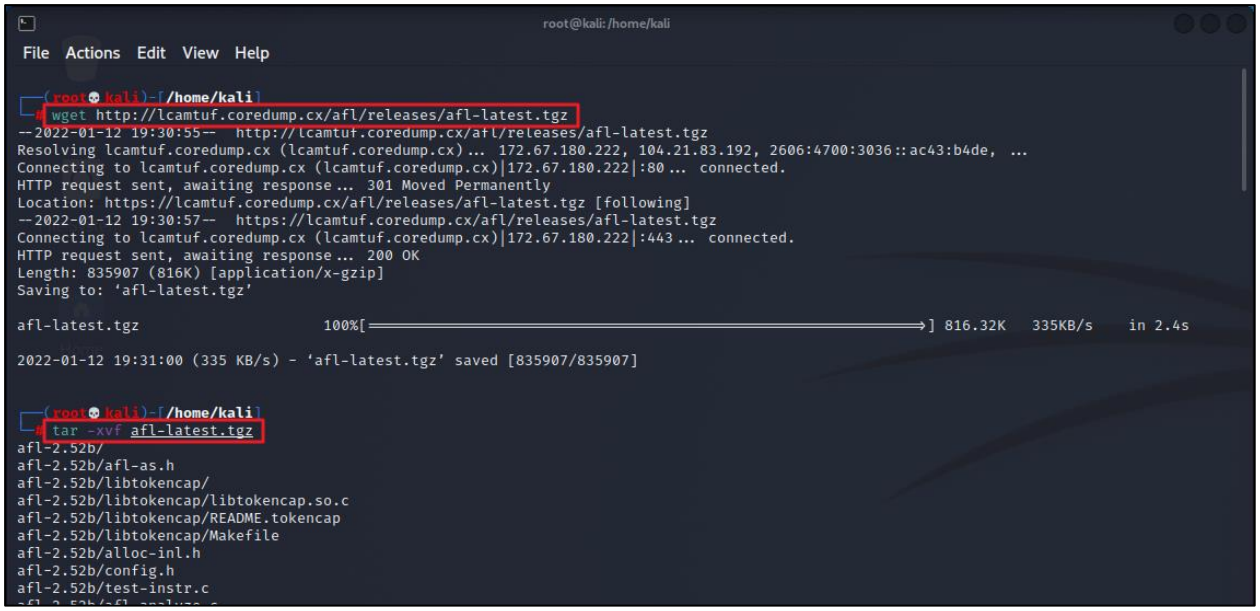
위 그림은 퍼징 로직을 대략적으로 보여준다.

1. afl-fuzz 에서 퍼징하려고 한 대상 프로그램을 실행한다.
2. 해당 프로그램이 처음으로 실행되면서 afl-fuzz 와 pipe 로 통신을 하면서 fork server 를 만들고, 새로운 타켓 인스턴스를 fork call()로 실행한다.
3. 표준 입력 or file 로 들어온 입력이 fuzz 대상 프로그램으로 전달된다.
4. 실행된 결과를 공유 메모리에 기록하여 실행을 완료한다.
5. afl-fuzz 는 공유 메모리에서 fuzzer 대상이 남긴 기록을 읽고 이전 항목을 변경하여 새로운 입력을 만든다.
6. 새롭게 만든 입력은 다시 프로그램에 들어가서 실행된다.

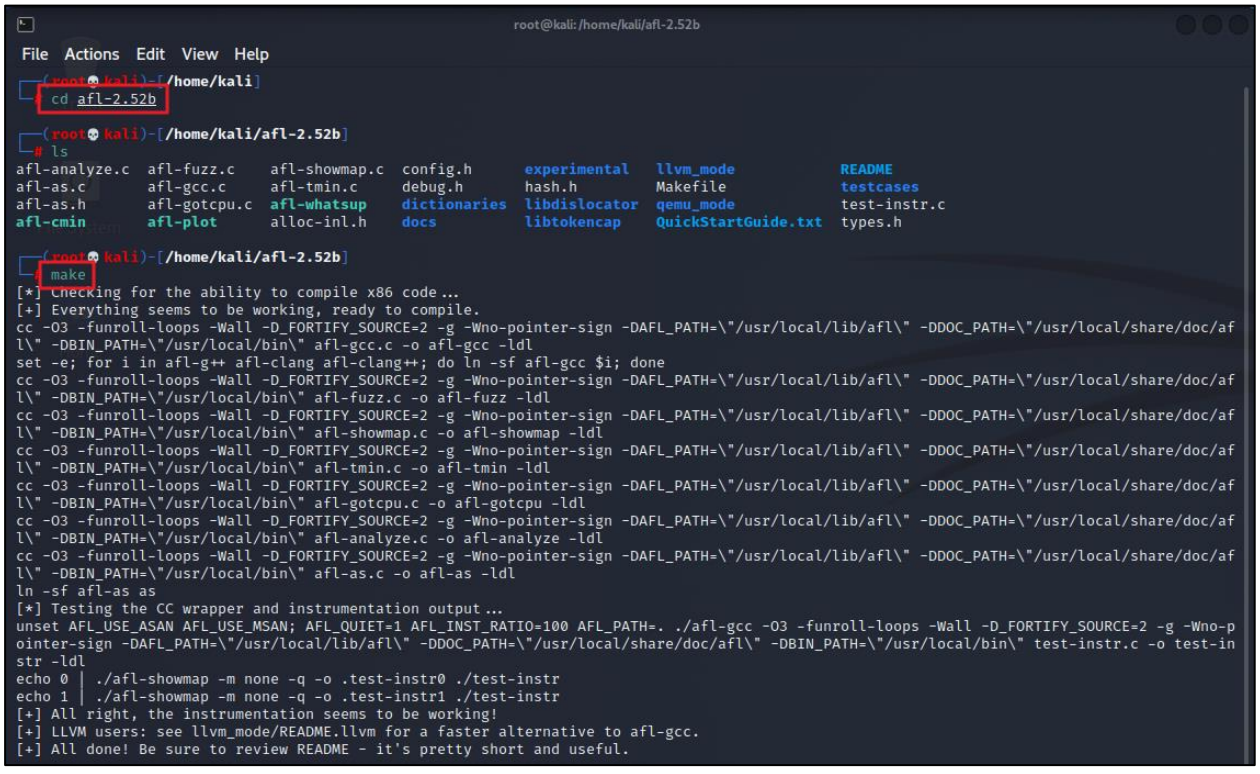
공유 메모리를 통해 새로운 입력을 만드는데 이 부분이 코드 커버리지를 높이기 위한 로직으로, 컴파일시 branch 에 다음과 같은 코드가 삽입된다.

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

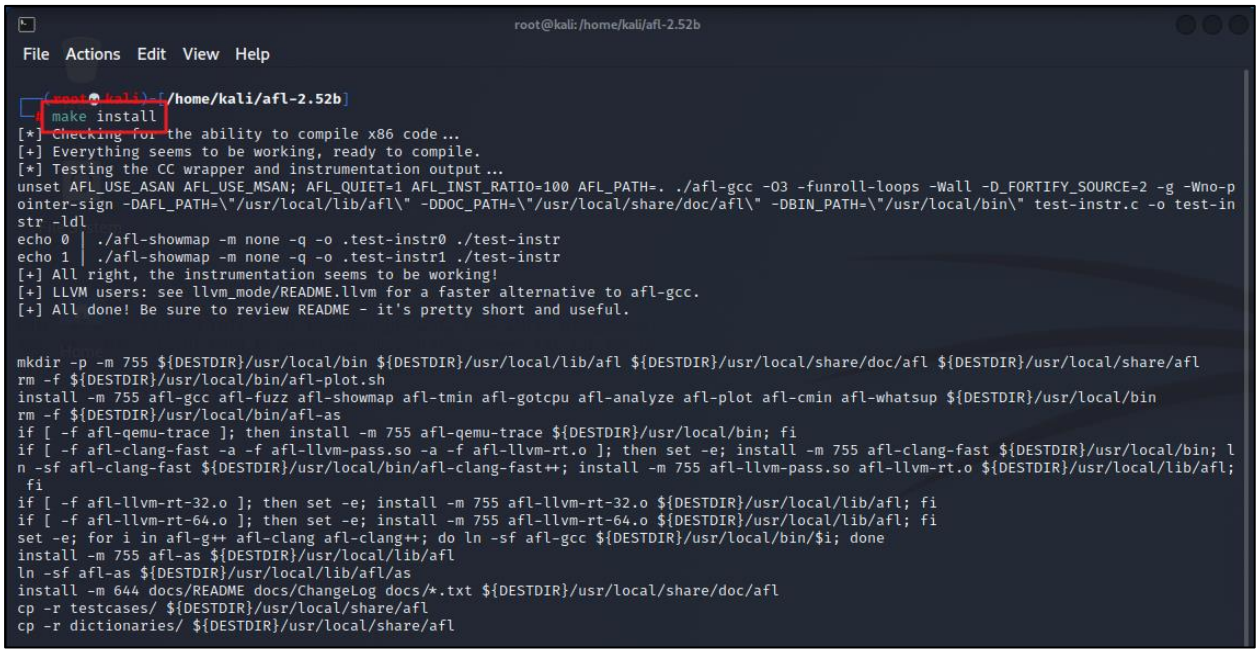
3.2 AFL Fuzzer 설치 및 사용법



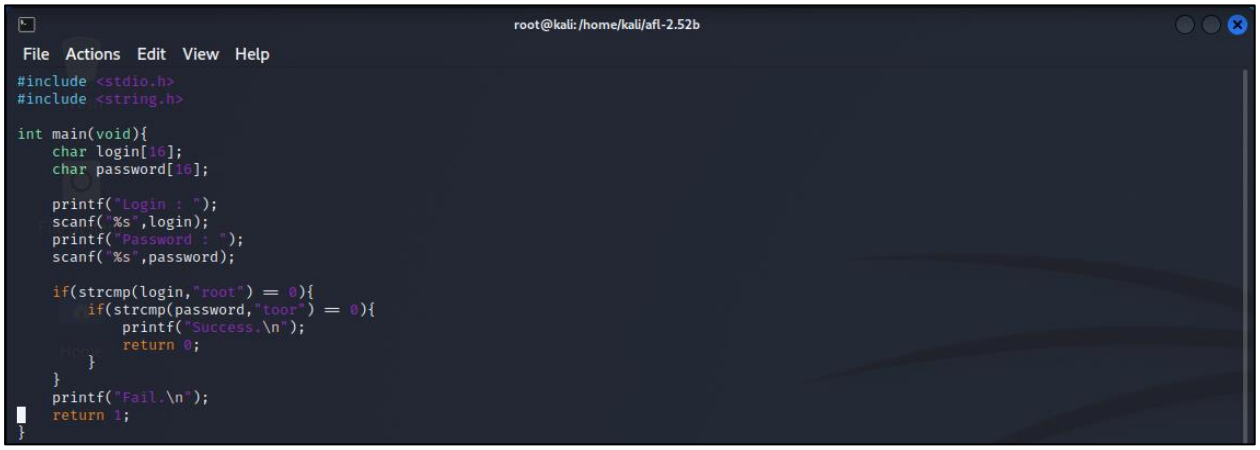
afl fuzzer 를 사용하기 위해 파일을 다운 받고 압축해제한다.



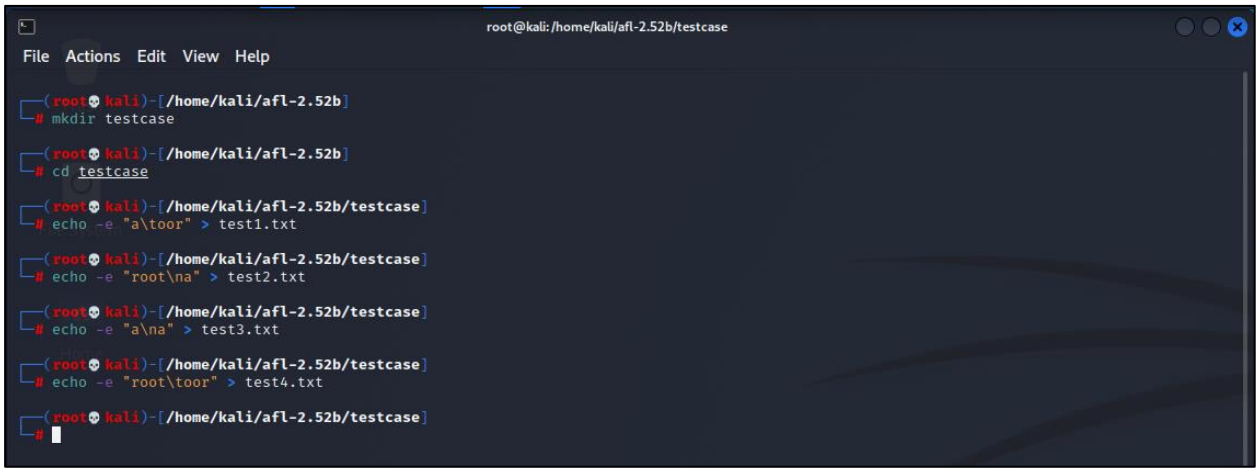
폴더로 이동하여 소스를 컴파일한다.



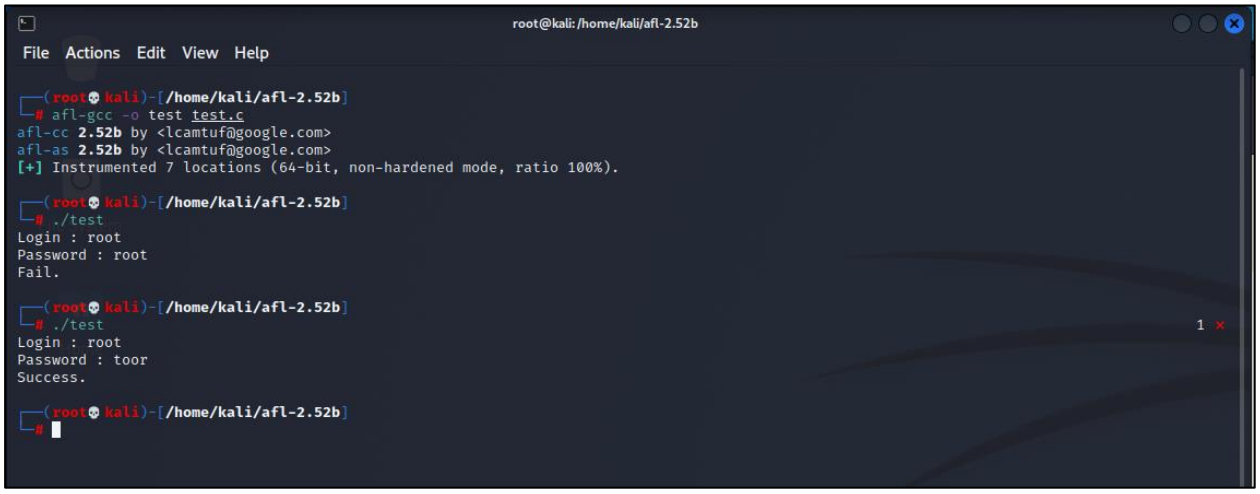
make 를 통해 만들어진 설치파일을 설치한다.



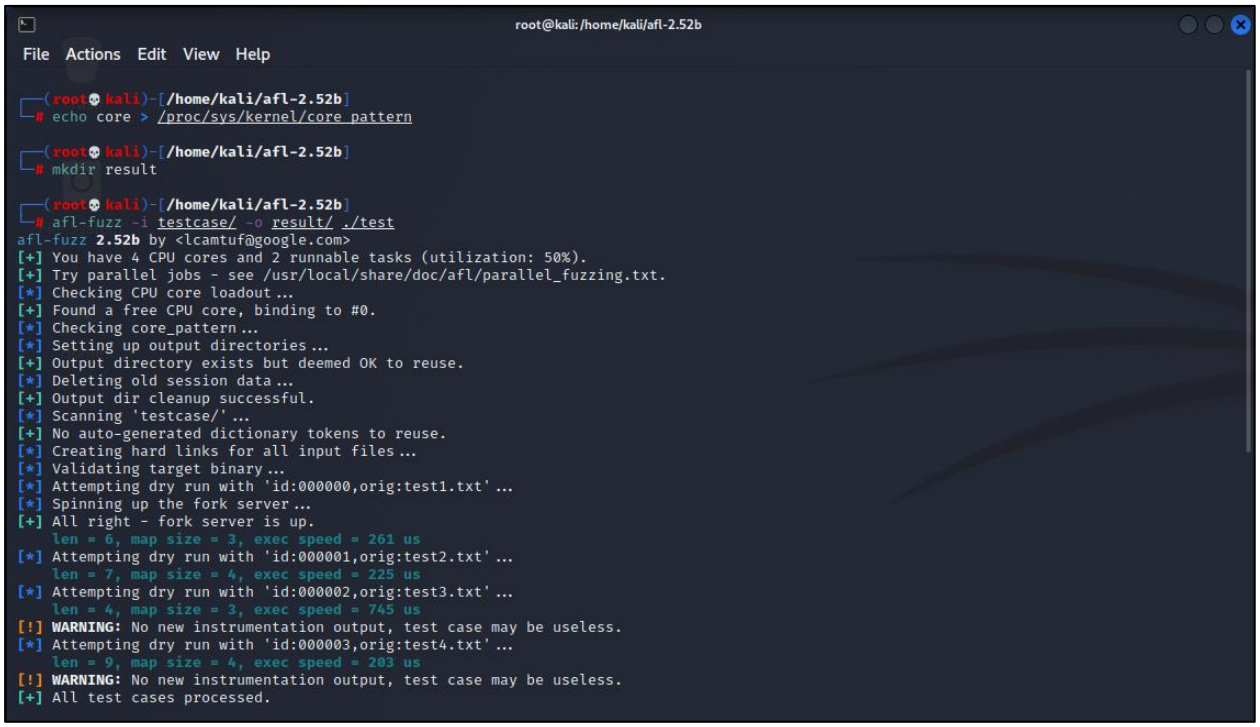
사용자로부터 id, password 를 입력받아 프로그램이 요구하는 값과 일치하면 "Success"를 출력하고, 그렇지 않으면 "Fail"을 출력하는 테스트 파일을 생성한다.



퍼징에서 사용하기 위해 테스트 케이스(ID 가 틀린 경우, Password 가 틀린 경우, ID, Password 모두 틀린 경우, ID, Password 정확한 경우)를 생성한다.

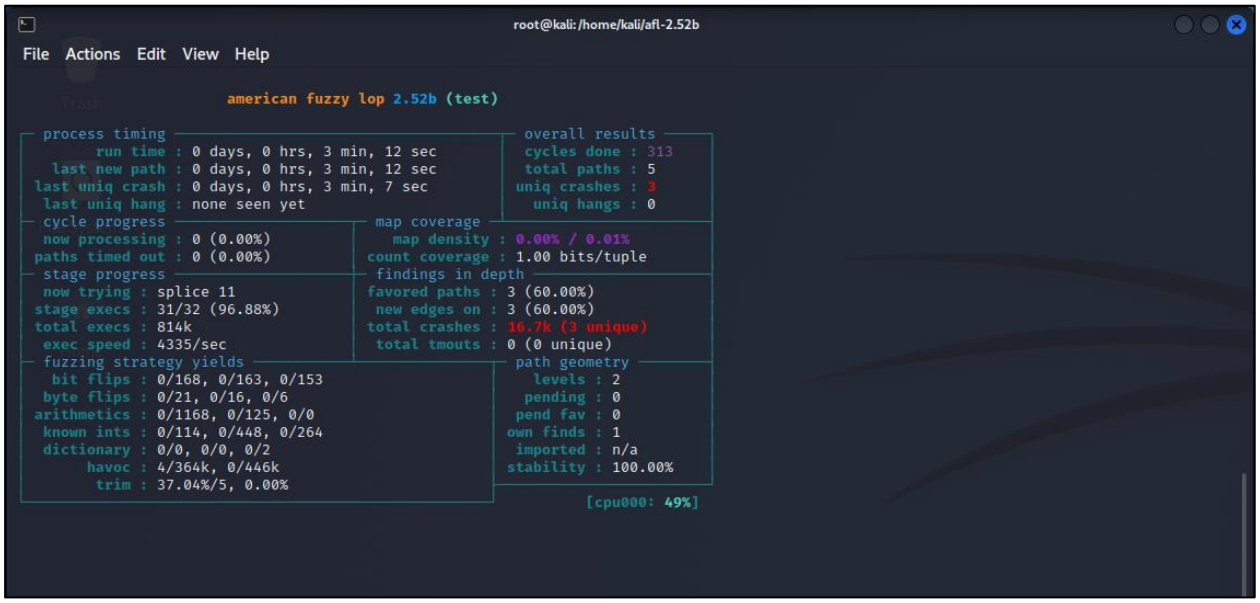


afl 에서 제공하는 컴파일러를 이용하여 빌드하며, 빌드된 파일이 정상적으로 동작하는 것을 확인한다.



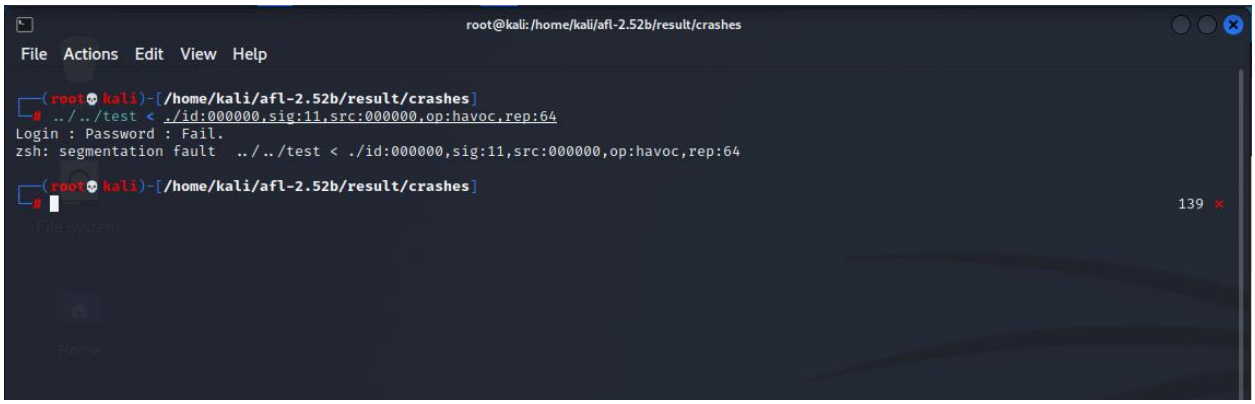
[afl-fuzz -i testcase/ -o result/ ./test]

afl-fuzz 를 사용하여 테스트 케이스가 저장된 디렉토리 위치를 지정하고 결과가 저장될 디렉토리 위치를 지정한 후 실행할 파일을 지정한다.



퍼징이 실행되고 있는 상태를 보여주며, findings in depth 를 보며 total crashes 를 확인 가능하며, 그 중 unique crashes 가 3 개인 것을 확인 가능하다.

result/crashes 폴더로 이동하면 크래시가 터진 변조된 파일들을 확인 할 수 있다.

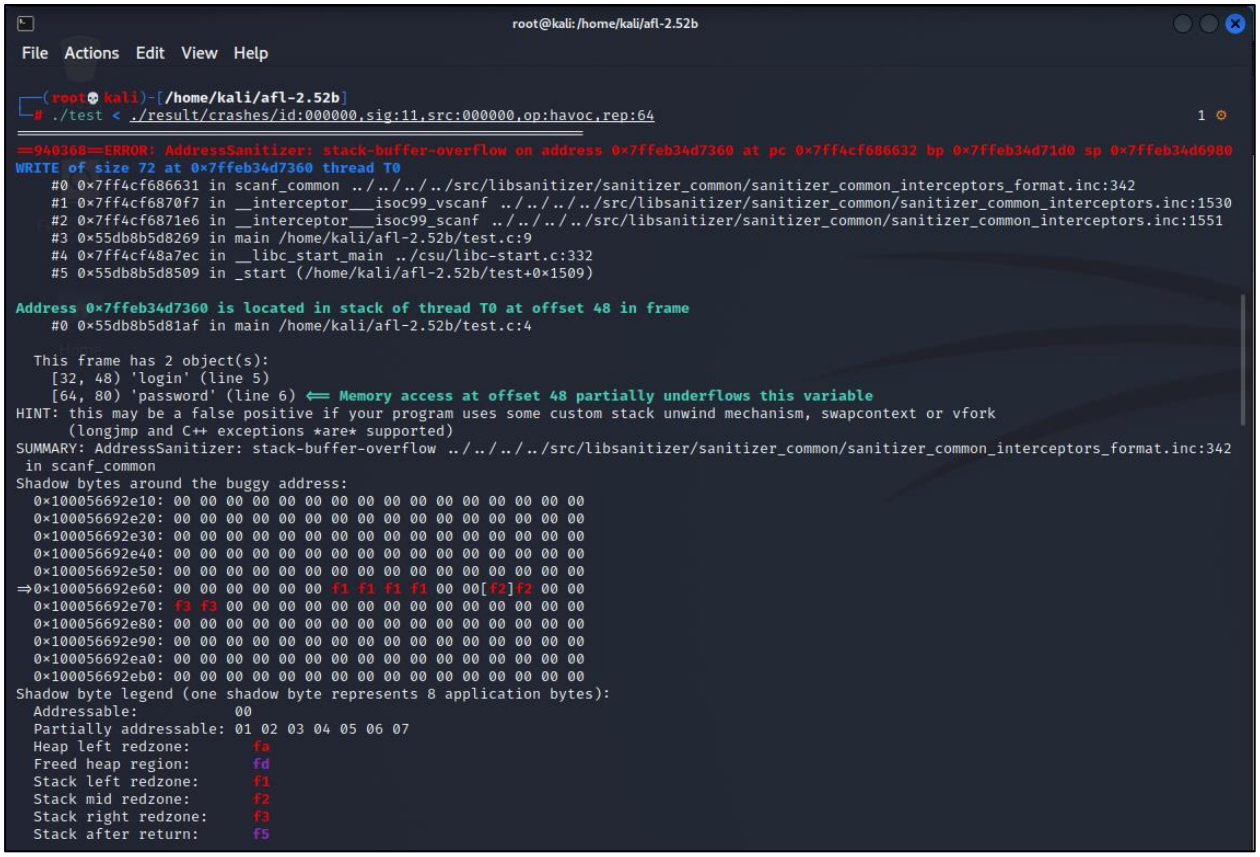


저장된 파일을 이용하여 해당 crash 를 재현할 수 있으며, 에러 내용 확인이 가능하다.

3.3 ASAN 설치 및 사용법

ASAN 은 AddressSanitizer 의 약자로 네이티브 코드의 메모리 버그 감지를 위한 빠른 컴파일러 기반 도구이다. 스택 및 힙 버퍼 오버플로우/언더플로우, 프리 후 힙 사용, 범위를 벗어난 스택 사용, 더블 프리/와일드 프리 등을 감지한다. 또한 ASAN 을 사용하면 크래시에 대한 정보가 매우 자세하게 나온다.

afl-gcc 를 사용하여 다시 빌드하며, 뒤에 ASAN 사용을 위해 -fsanitize=address 을 추가한다.



ASAN 을 사용하여 빌드한 파일을 이용하여 크래시 재현 시 보다 자세한 내용 확인이 가능하며, 어떤 오류가 발생하였고 발생한 위치 등 확인 가능하다.

4. FirmAE

FirmAE 는 에뮬레이션 및 취약점 분석을 수행하는 완전 자동화된 프레임워크이다. Firmadyne 보다 에뮬레이션 성공률을 크게 높였으며, 또한 대상 펌웨어의 파일 시스템 및 커널 로그를 기반으로 웹 서비스 정보를 유추하는 0-day 검색을 위한 동적 분석 도구를 개발하였다.

4.1 설치 및 사용법

```
$ git clone --recursive https://github.com/pr0v3rbs/FirmAE
```

Github 에 접근해 폴더를 복사한다.

```
$ ./download.sh
```

download.sh 스크립트를 실행한다.

```
$ ./install.sh
```

install.sh 스크립트를 실행한다.

```
$ ./init.sh
```

설치가 끝나면 init.sh 스크립트를 실행한다.

```
$ wget http://download.iptime.co.kr/online_upgrade/n604bl_kr_9_982.bin
```

펌웨어를 다운받는다.

```
$ sudo ./run.sh -c iptime n604bl_kr_9_982.bin
```

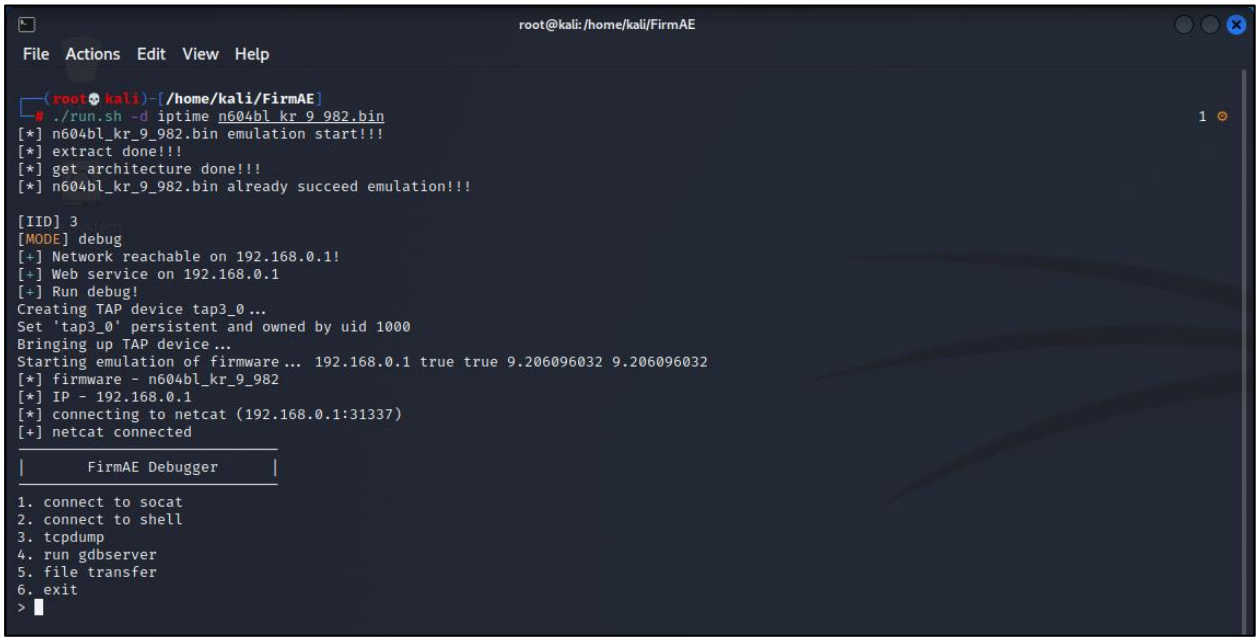
./run.sh -c <브랜드> <펌웨어> 로 명령어가 구성되어있으며, 사용할 펌웨어의 브랜드와 펌웨어 압축파일명을 작성한다.

```
$ sudo ./run.sh -d iptime n604bl_kr_9_982.bin
```

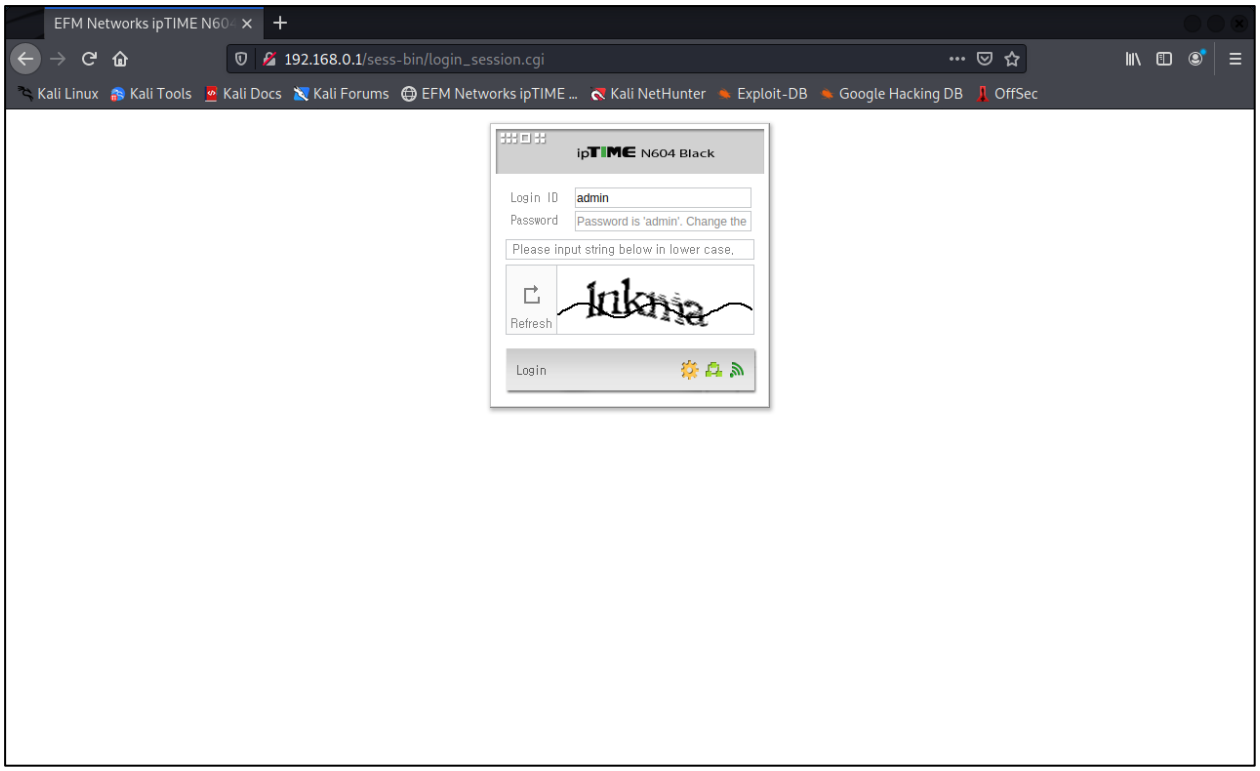
사용자 수준의 기본 디버깅 유틸리티로 에뮬레이트된 펌웨어가 네트워크를 사용할 경우 유용하다.

4.2 동적 디버깅

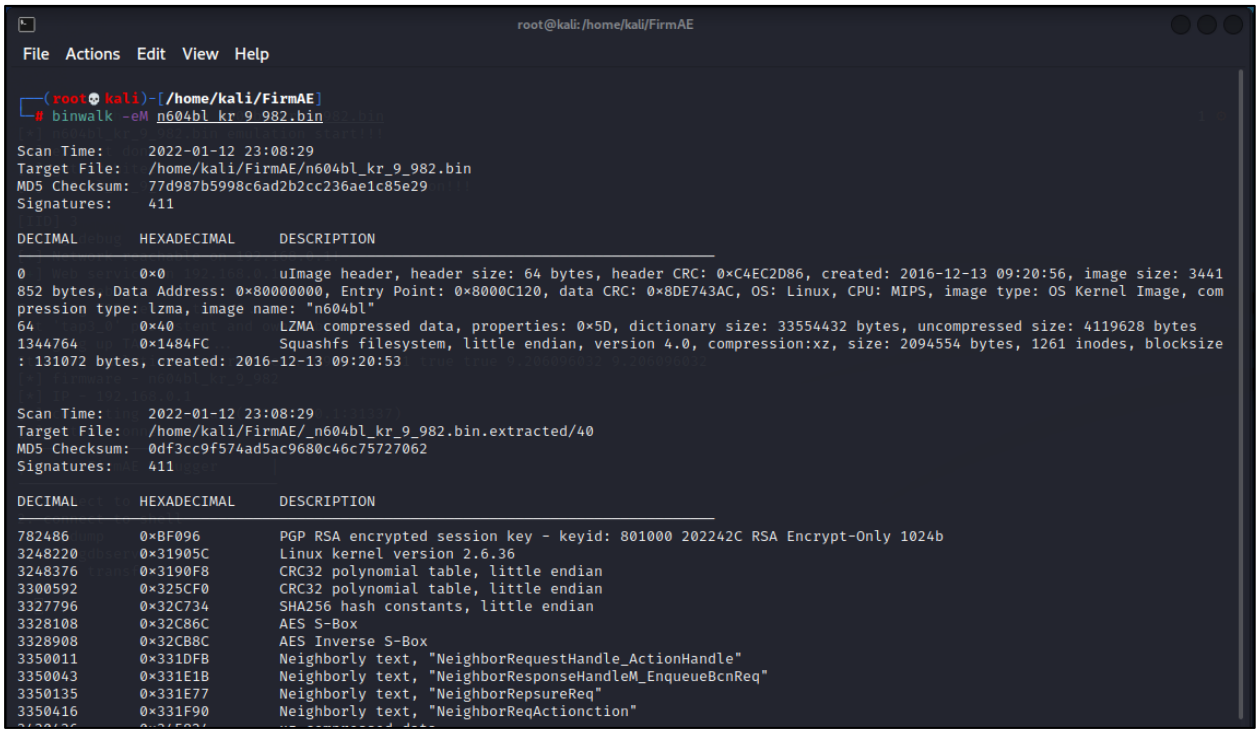
동적 디버깅이란 파일을 직접 실행시켜서 분석하는 방법으로 디버깅을 통하여 코드 흐름과 메모리 상태등을 살펴보는 방법이다. 또한 디버거를 이용하여 프로그램 내부 구조와 동작원리를 분석할 수 있다.



[./run.sh -d iptime n604bl_kr_9_982.bin] 명령어를 실행하여 debug 모드로 실행한다.

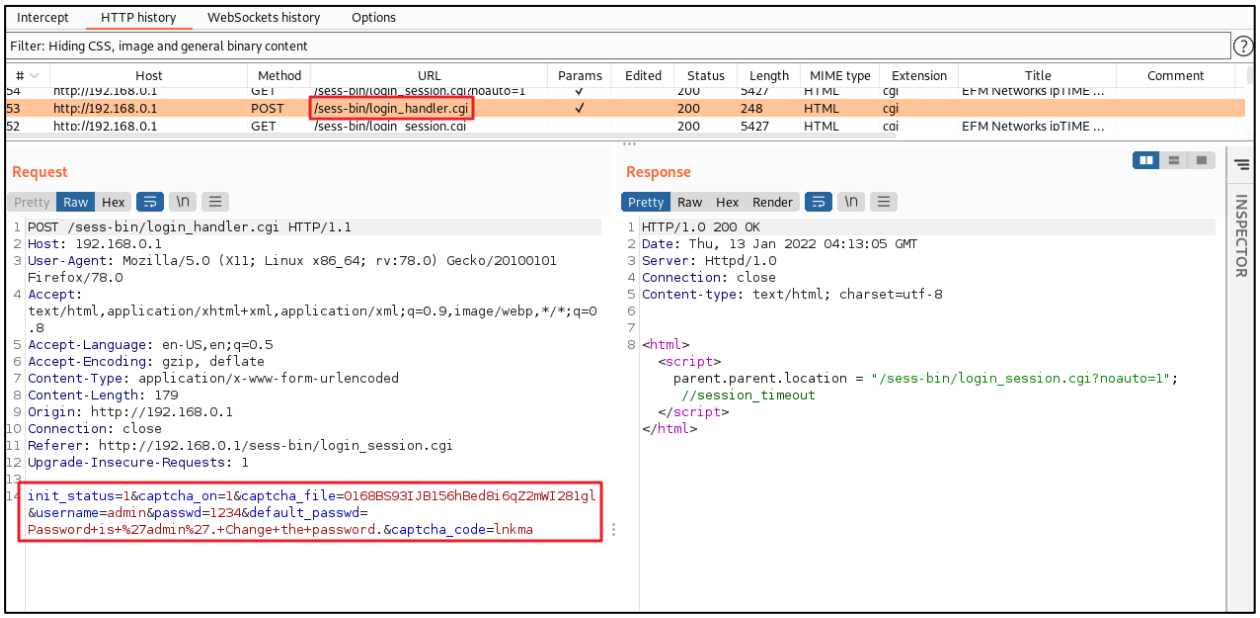


192.168.0.1 로 접근 시 iptime 관리자 페이지에 접근이 가능하다.

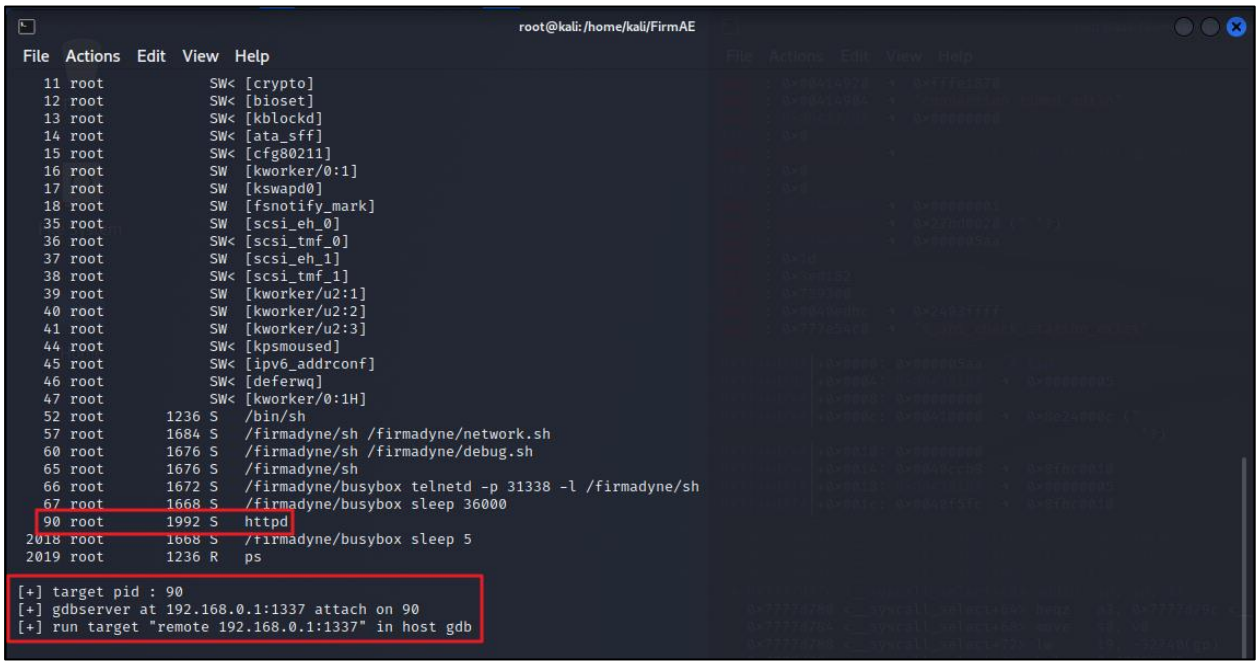


binwalk 툴을 사용하여 대상 펌웨어 파일시스템을 추출한다.

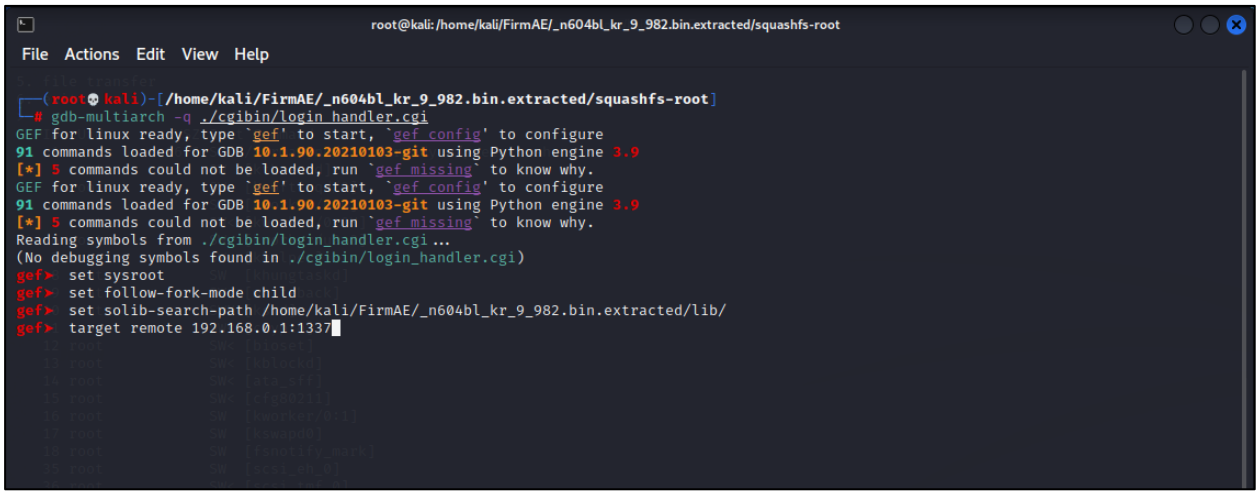
추출한 파일시스템 디렉토리로 이동하여 동적 디버깅할 대상을 확인한다.



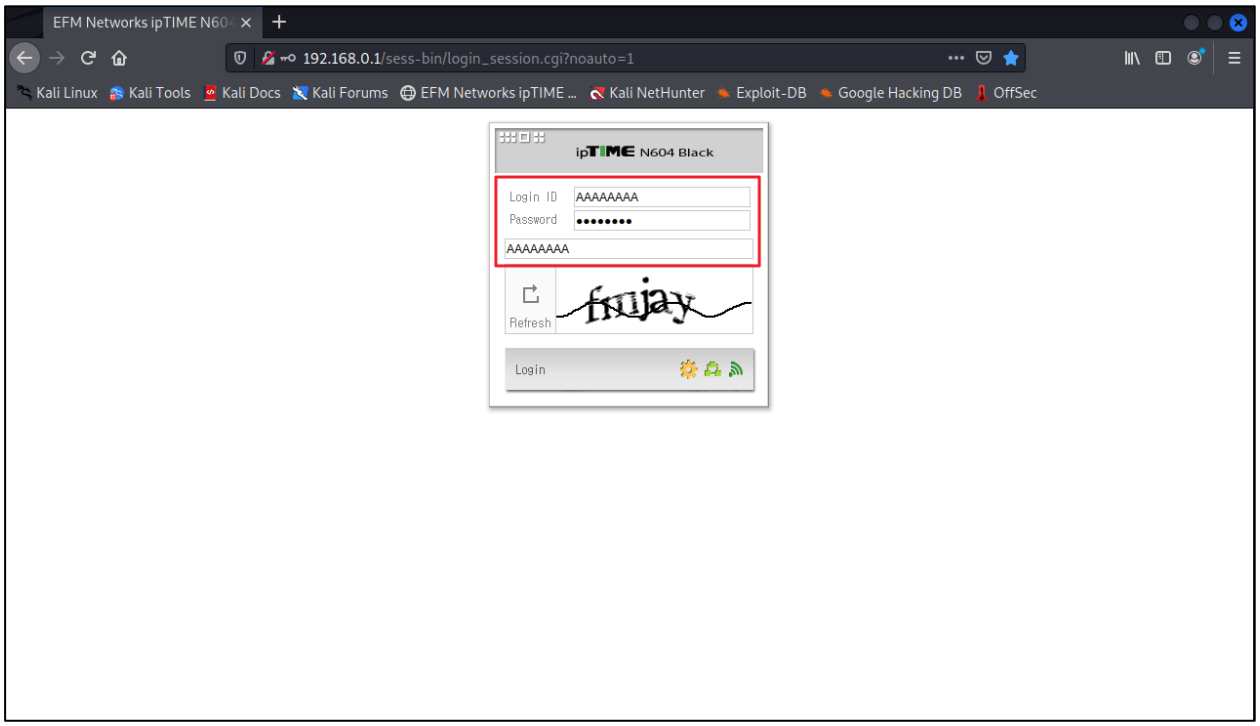
로그인 시 입력하는 값들에 대해 오버플로우가 발생하는지 확인하기 위해 로그인 시 인자 값을 전송하는 login_handler.cgi 를 공격 대상으로 정하였다.



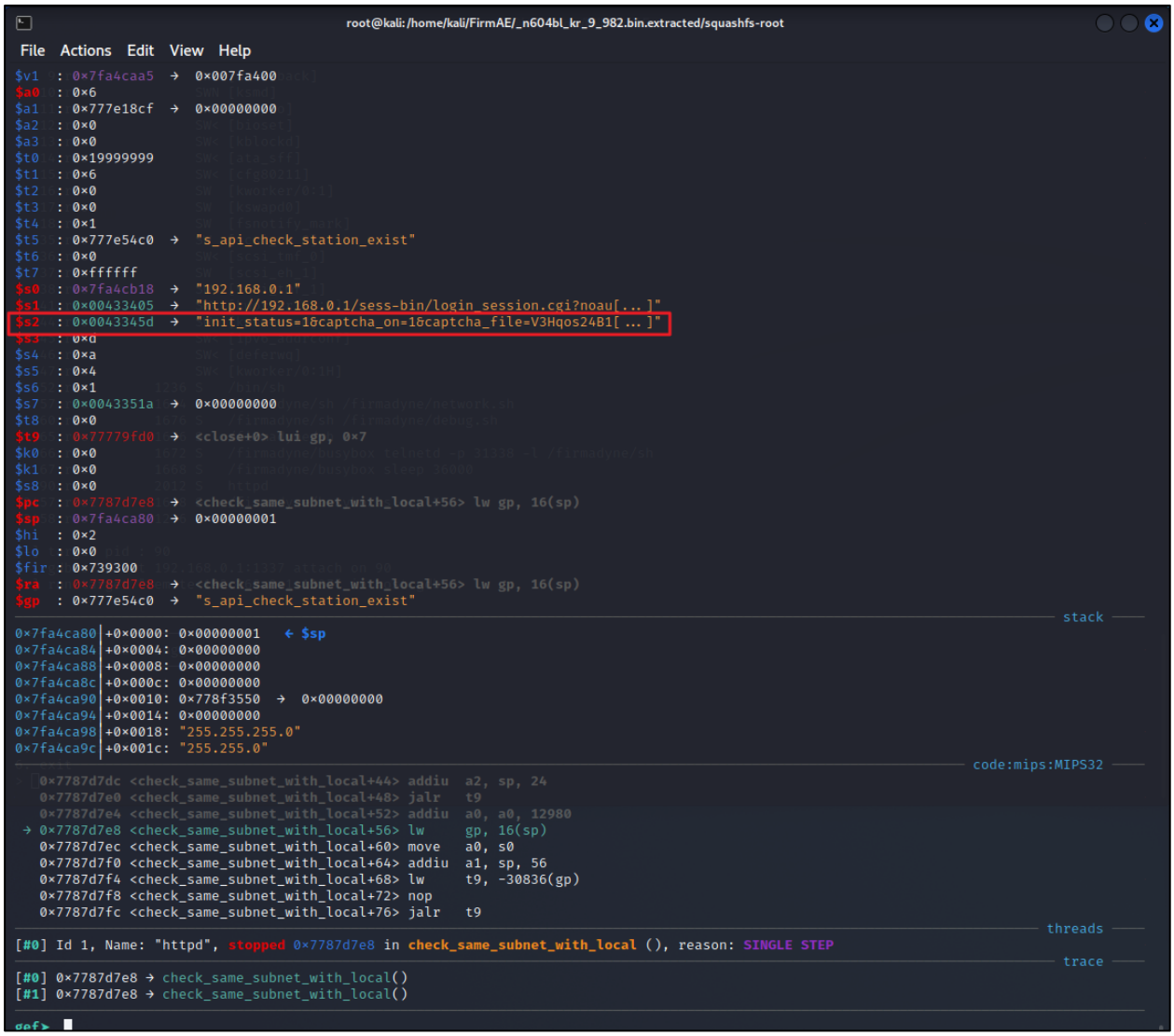
run gdbserver 를 선택하여 타켓 서비스 pid 를 입력한 후 gdb server 연결 주소를 확인한다.



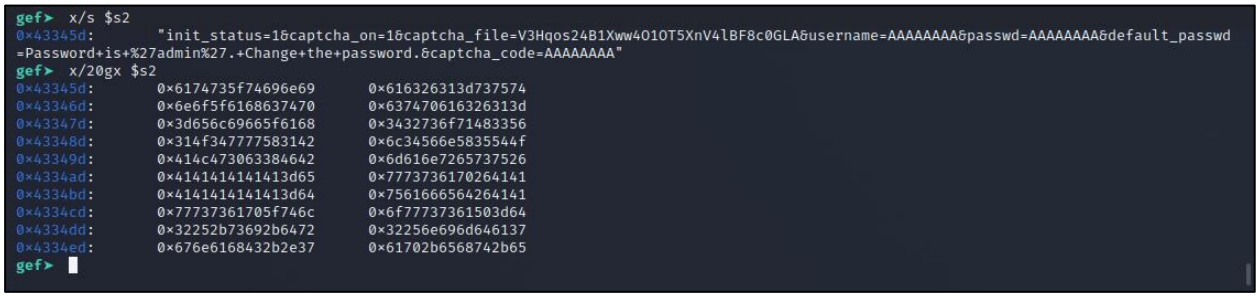
gdb 를 이용하여 대상 cgi 파일을 실행 및 디버깅 설정을 한 후 gdb 서버와 연결한다.



적절한 장소에 breakpoint 를 설정한 후 입력 값을 작성하여 로그인 시도를 한다.



레지스터에 입력한 값들이 저장된 것을 확인한다.



입력한 값들이 정상적으로 레지스터에 존재하며, 동적 디버깅이 가능하다.