1인 가구 및 취약 계층 위험 예방을 위한

스마트 플러그 패킷 분석

네트워크 프로그래밍 자유주제 프로젝트

8조 - 20193121 허동윤/ 20213044 이정민/ 20211806 이소현

[목차]

1. 프로젝트 개요

1.1. 프로젝트 배경

스마트 플러그의 사용은 최근 몇 년 동안 점점 보편화되어 다양한 가전 제품에 편리함과 제어 기능을 제공한다. 그러나 즉각적인 도움 없이 사고나 비상사태의 위험에 처할 수 있는 1인 가구와 취약 계층의 안전과 복지에 대한 우려는 여전히 존재한다.

이러한 우려를 해소하기 위해 본 프로젝트는 스마트 플러그의 패킷 분석을 활용하여 잠재적인 위험을 예방하고 1인 가구의 안전을 확보하는 것을 목표로 한다.

네트워크 트래픽 패턴 및 빈도, 프로토콜 헤더 및 페이로드 콘텐츠를 분석하여 비정상적인 활동과 잠재적인 위험 식별할 수 있다. 먼저, 일정 기간 동안 스마트 플러그에서 트래픽이 없는 경우에는 사람과 스마트 플러그 사이의 상호 작용이 끊긴 것으로, 사용자가 응급 상황과 같은 위험한 상황에 처해 있다고 간주한다. 이때 시스템은 사용자에게 알람을 보내 상황에 대한 정보를 요청한다. 응답이 없을 경우 위험 상황으로 판단하여 관련 센터에 알린다.

또한 정상적인 범위를 벗어난 모든 트래픽은 차단되고 해당 트래픽이 신뢰할 수 있는 사용자로부터 온 것인지 확인한다. 확인이 된다면 정상적인 트래픽으로 분류하고, 그렇지 않은 경우 시스템은 위험을 감지하고 그에 따라 사용자에게 알린다.

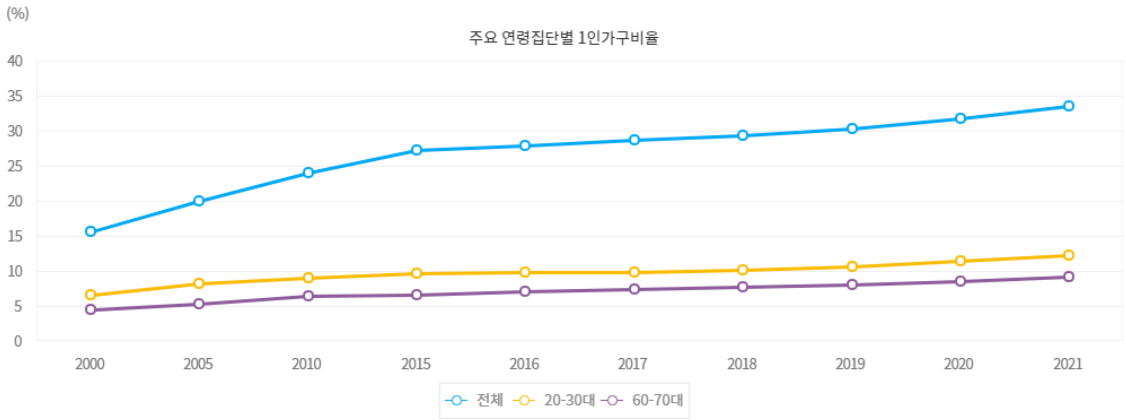
1.2. 프로젝트 목표

본 프로젝트에서는 스마트 플러그에서 발생하는 패킷을 분석하여 1인 가구의 위험을 방지한다. 사용자로부터 주 7일 동안의 플러그 사용 패턴을 입력받고, 이를 기반으로 정상 범위에서 벗어난 패턴이 감지되면, 사용자에게 알림을 전송하거나 자동으로 전원을 차단하는 등의 조치를 취한다. 알림을 통해 사용자는 위험 상황을 신속하게 인지하고 적절한 대응을 할 수 있다. 또한, 자동으로 전원을 차단함으로써 잠재적인 위험으로부터 사용자를 보호한다. 이를 위해 네트워크 패킷을 실시간으로 분석하는

기능을 구현한다. 이렇게 자동화된 위험 감지 시스템을 통해 1인가구의 안전을 보장하고, 일상 생활에서의 번거로움을 최소화할 수 있다.

1.3. 프로젝트 필요성

빠른 고령화, 늦어지는 결혼과 출산율 감소, 이혼율 증가 등으로 1인 가구가 급속하게 늘어나고 인구의 고령화도 급속도로 진행되고 있다. 이러한 사회적 변화는 사망에 이를 수 있는 갑작스러운 사고나 사건의 위험에 노출된 취약 계층의 증가로 이어진다.



1인 가구는 자신 외에는 집을 돌볼 사람이 없다는 단점이 존재한다. 갑자기 쓰러졌을 때에는 제때 치료를 받지 못해 고독사의 위험이 있고, 화기 제품과 같은 위험하고 전력 소비량이 많은 기기를 끄지 않고 외출했을 때에는 타인의 도움 없이는 해결하지 못한다.

이처럼 1인 가구와 취약계층의 안전과 복지를 위해 지속적인 모니터링과 지원이 필요한 상황이 늘고 있는 상황에서, 최근 1인 가구와 취약계층에 대한 위험을 예방하는 수단으로 가전 제품 제어와 전력 사용량 모니터링이 가능한 스마트 플러그가 주목받고 있다.

스마트 플러그를 사용하면 생성된 패킷을 분석해 에너지 소비 및 사용 패턴에 대한 실시간 데이터인 기상, 취침, 외출, 가전제품 사용 등 거주자의 일상 활동에 대한 정보를 얻을 수 있다. 이때 비정상적인 패턴을 식별함으로써 시스템은 간병인이나 응급 서비스에 데이터를 보내 위험 사고, 부상 또는 사망을 예방할 수 있는 것이다.

따라서, 본 프로젝트는 1인 가구 및 취약계층에 대한 위험 사고 문제를 스마트 플러그 패킷 분석을 통한 자동화 프로그램으로 해결하고자 한다.

1.4. 기존 기술 대비 프로젝트 특장점

1.4.1. 1인 가구 고독사 예방을 위한 더 효율적인 설계

1인 가구 고독사 예방을 위해 기업이나 민간 모두 다양한 노력을 하고 있다. 대표적으로 매일우유의 '우유 안부' 캠페인은 배달원이 매일 우유를 배달하며 미개봉 우유가 남아있는지 확인하고, 두개 이상 남아 있다면 위험으로 간주해 관공서에 신고하는 방식이다. 이 외에는 주기적으로 1인 취약계층의 집에 찾아가 안부를 묻는 봉사 등이 있다.

이러한 방법은 사용자가 많을수록 노동력이 추가적으로 필요하다는 단점이 존재하고, 꾸준한 물리적/금전적 지원을 요구한다.

하지만 스마트 플러그 패킷 분석 기반 1인 가구 위험 예방 프로그램을 이용해 스마트 플러그를 설치하고 네트워크에 연동한다면 그 뒤로 별 다른 유지보수가 필요하지 않다는 장점이 존재한다.

1.4.2. 자동 위험 차단 서비스

원격으로 특정 기기의 전원을 차단하는 것은 일반 스마트 플러그에서도 가능하지만, 사용자가 먼저 알아차리고 APP/WEB을 통해 직접 차단해야 한다는 번거로움이 존재한다.

해당 프로젝트에서는 사용자가 스마트 플러그에 연결한 기기를 주로 언제 사용하는지, 어느정도 사용하는지 지정할 수 있어 그 외의 범위에서 트래픽이 발생하는 것을 감지한다. 신뢰하는 사용자로부터 발생된 트래픽인지 확인하고, 알람을 보내는 방식으로 처리한다. 응답이 없거나 비정상적인 트래픽이 계속 발생하면 자동으로 전원을 차단해준다.

1.4.3. 네트워크 오류로 인한 스마트 플러그 오작동 방지

네트워크 오류로 스마트 플러그 기기와 연동이 빈번하게 끊어진다면 이를 감지해 사용자에게 더 안전한 환경을 만들도록 요청한다.

패킷에 오류가 존재하거나 패킷이 아예 발생하지 않는다면 사용자의 생활 패턴과 환경을 알기 어렵기 때문이다. 시스템 자체의 결함으로 인한 네트워크 오작동도 이러한 방식으로 확인할 수 있다. 패킷 분석을 통해 사용자의 안전뿐만 아니라 사용자의 안전에 관련된 외부적인 요소도 탐지할 수 있다.

1.5. 수업 연관성

1.5.1. Wireshark

WireShark를 통해 패킷 헤더 정보를 읽는 방법과 각 프로토콜이 어떻게 동작하는지 학습하였다. 이러한 실습을 기반으로 각 패킷이 신뢰성 있게 데이터를 전송하고 있는지 알 수 있고, 사용자로부터 오는 패킷이 정상적인 접근인지 확인할 수 있다.

1.5.2. client - server socket 통신

네트워크 프로그래밍에서 가장 주되게 배운 client - server socket 통신을 구현하였다. server 차원에서는 client를 더 신뢰성 있게 관리할 수 있으며, 복잡한 모듈과 숨겨 client에게 더 편리한 경험을 제공한다.

1.5.3. select()

client 다중 통신을 구현하기 위해 사용하였다.

1.5.4. Multi Thread

하나의 프로세스 내에서 공유자원을 공유하며 Thread별로 독립적으로 수행한다는 장점을 이용하여 server를 구현한다. 크게 패킷 캡쳐, 사용자 패턴 일치 여부 확인하는 Thread로 나눌 수 있으며 둘 사이에는 packet.time을 전송할 buffer라는 공유자원이 존재한다.

패킷 발생시 패턴 일치 여부를 판단할 때, 패킷 캡쳐를 멈추지 않기 위해 이러한 방식을 사용하였다. 패킷 발생시마다 Thread를 생성한다면 시스템 오버헤드가 커질 것으로 예상해 한번에 두개의 Thread를 동시 생성하였다.

1.5.5. 네트워크 프로토콜

HTTP, TCP, TLS 등 프로토콜 종류에 따른 쓰임새와 특징에 대해 학습하였다. 본 제품의 통신은 TLS, TCP 기반이기 때문에 네트워크 패킷 헤더를 해석하여 암호화 기법, 알고리즘 종류, 암호화 형식 등에 관해 알 수 있었다.

2. 프로젝트 개발 환경 및 기술

2.1. 개발 언어 및 도구

2.1.1. 운영체제 및 개발 도구

Microsoft Windows 11 (버전 22H2) 환경에서 Visual Studio Code를 이용해 개발하였다.

2.1.2. Python 3.9.31

Python는 네트워크 패킷 캡처와 분석을 위한 다양한 라이브러리가 있으며, 쉬운 문법과 쉬운 사용성을 제공한다. 다양한 분야에서 활용되고 있는 간편한 언어이지만, 특히 데이터 분석과 처리를 위해 많이 사용되고 있다. 패킷 분석 server를 만들고 사용자 입력을 받을 때 이용한다.

2.1.3. TP-Link mini Smart WiFi Plug P100

WiFi를 이용하는 스마트 플러그를 통해 네트워크에 발생하는 패킷을 캡쳐해 분석한다. 사용자의 접근성을 고려하여, 온라인 쇼핑몰 쿠팡에서 랭킹 1등에 위치한 제품을 이용하였다.

2.1.4. WIFI

무선 인터넷에 연결된 장치끼리 데이터를 주고받는 데 사용되며, 개발환경의 스마트 플러그 또한 WiFi 환경을 이용하기 때문에 선택하였다.

2.2. 사용된 라이브러리 및 오픈 소스 소개

2.2.1. datetime

datetime 라이브러리는 날짜와 시간을 표현하기 위한 다양한 클래스와 형식을 제공한다. 패킷 발생 시간, 사용자가 입력한 시간을 datetime 라이브러리를 이용해 형식화한다.

2.2.2. Scapy ver. 2.5.0

파이썬 패킷 캡쳐 라이브러리이다. 다양한 필터 기능과 계층별 프로토콜, 송/수신 주소 등을 제공한다.

2.2.3. pickle

파이썬에서 객체 직렬화를 위한 내장 모듈이다. 소켓 통신으로 class의 객체를 전송하기 위해 바이트 스트림으로 변환할 때 사용하였다.

3. 프로젝트 실행 환경 및 API 소개

3.1. 실행 환경 구성

3.1.1. 노트북 핫스팟

핫스팟 Wi-Fi에 스마트 플러그를 연결하고, client와 server를 구동시킨다.

3.1.2. Tapo Application

본 프로젝트에서는 Tapo-Link의 P100 스마트 플러그를 이용했기에 Tapo 사에서 지원하는 어플리케이션을 이용해 플러그를 제어한다.

3.1.3. Tapo-Link P100 SmartPlug

펌웨어 1.1.5 Build 230324 Rel.143956

3.2. 주요 API 소개 및 사용법

3.2.1. PyP100

TP-Link 제품을 조작할 수 있는 API이다. P100, P105, L530 등의 제품을 지원하며 보고서에서는 P100 제품을 이용하였다. 플러그와 원격으로 연결하고 간단한 조작 및 상태 불러오기가 가능하다.

- P100()

P100 플러그의 IP, TP-Link 이메일, 패스워드을 입력하여 플러그 객체를 생성한다.

- handshake()

객체의 여러 메소드를 호출하기 위해 Access Token을 생성한다.

- login()

플러그 인증 전송 후 AES를 사용한 key와 iV를 생성한다.

- turnOn(), turnOff()

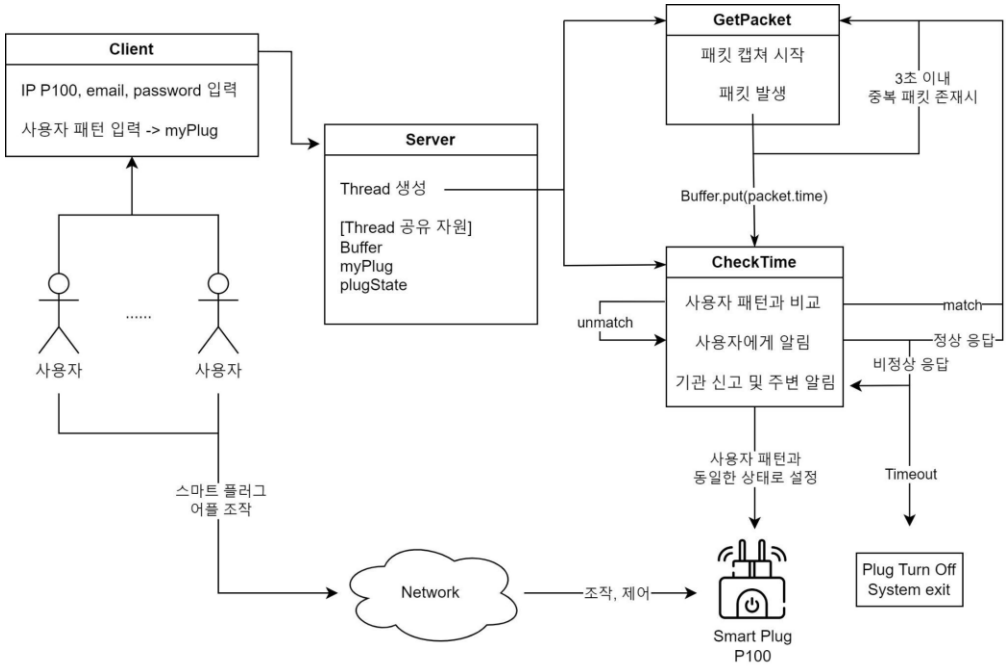
플러그의 상태를 제어하는 메소드이다.

- getDeviceInfo()

연결되어있는 플러그의 정보를 json 형태로 제공한다. IP, mac, 플러그의 현재 상태 등이 존재한다.

4. 프로젝트 설계

4.1. 프로젝트 아키텍처



4.2. 구성 요소 및 모듈 설명

1. client 연결
    a. 스마트 플러그 P100의 IP, TP-Link의 email, password를 입력받는다.
2. client -> server 접속

a. server는 패킷 캡쳐하는 Thread 1, 패킷 발생 시간과 사용자의 패턴을 비교하는 Thread 2를 생성한다.

b. Thread 1에서 패킷 발생시 버퍼를 통해 Thread 2에게 패킷 발생시간을 공유한다.

3. Thread 2에서 패킷 발생 시간과 사용자의 패턴과 비교 / 분석한다.

a. 이상 패킷 발생시 사용자에게 알린다.

i. 사용자의 응답이 "Yes"이면 정상적으로 처리한다.

ii. 사용자의 응답이 "Yes"가 아닐 경우 플러그를 임의로 Turn off한다.

iii. 사용자의 응답이 오지 않으면 시스템 자체를 종료한다.

5. 성능 평가

5.1. 기존 기술 대비 성능 비교

기존의 1인 가구 고독사 예방을 위한 접근 방식은 매일 물리적인 방문이나 주기적인 전화 등을 통해 안부를 확인하는 방법이 주로 사용되었다. 이러한 방법은 사용자 수가 많을수록 더 많은 인력과 시간이 필요하며, 꾸준한 물리적/금전적인 지원을 요구하는 단점이 있다. 그러나 스마트 플러그 패킷 분석 기반 1인 가구 위험 예방 프로그램은 기존의 방식을 보완하여 기술적으로 발전한 성능을 제공한다.

첫째, 본 프로그램을 설치하고 네트워크에 연동하면 추가적인 유지보수가 필요하지 않다. 이는 사용자가 스마트 플러그에 연결한 기기의 사용 패턴을 기반으로 자동으로 위험을 감지하고 처리하기 때문이다.

둘째, 본 프로그램은 자동 위험 차단 서비스를 제공한다. 일반적인 스마트 플러그에서도 전원 차단은 가능하지만, 사용자가 직접 알아차리고 애플리케이션이나

웹을 통해 차단해야 하는 번거로움이 있다. 하지만 본 프로그램은 사용자가 미리 설정한 사용 패턴을 기반으로 사용자를 신뢰하고 정상적인 트래픽인지 확인한 후, 비정상적인 트래픽이 감지되면 자동으로 전원을 차단하여 위험을 예방한다.

셋째로, 우리의 시스템은 네트워크 오류로 인한 스마트 플러그 오작동을 방지한다. 네트워크 오류로 인해 스마트 플러그와의 연결이 끊어질 경우, 시스템은 이를 감지하고 사용자에게 안전한 환경을 유지할 것을 요청한다. 또한, 패킷 분석을 통해 사용자의 생활 패턴과 환경을 파악할 수 있으며, 시스템 자체의 결함으로 인한 네트워크 오작동도 식별할 수 있다.

위의 특장점을 바탕으로 본 프로젝트는 기존의 고독사 예방 기술과 비교하여 더 나은 성능과 효율성을 보여주었다. 사용자는 편리하게 스마트 플러그를 설치하고 네트워크에 연동함으로써 실시간으로 위험을 감지하고 자동 차단되는 서비스를 이용할 수 있다. 이를 통해 1인 가구 고독사 예방에 큰 도움이 될 것으로 기대된다.

6. 소스 코드 분석

6.1. 주요 모듈 및 클래스 설명

6.1.1. client.py

client가 접속하는 코드이다. client로부터 IP_P100, TP-Link의 이메일, 비밀번호를 입력받아 server에게 socket 통신으로 전송한다.

사용자의 패턴을 입력하는 부분이며, 여기서 선언된 myPlug 객체 또한 server에게 전송된다.

사용자의 패턴과 일치하지 않는 패킷 발생시 server로부터 알람을 받을 수 있다. 미응답시 종료, 정상 응답시 상태태 지속, 비정상적인 응답시 plug를

turn off한다. Threading.Timer Thread를 사용했으며 timeout 기준은 60초로 설정하였다.

```
if(response == "You are using at the wrong time. Are you sure you are?"):
    #Timeout 되면 실행하는 함수
    def sendans():
        #answer=""이기 때문에 turnoff된다
        client_socket.send(answer.encode())
        t.cancel()

        #사용자의 입력 없이 timeout 발생시 프로세스 종료
        client_socket.close()
        os._exit(0)

    #60초 동안 input 없으면 timeout
    timeout = 60
    t = threading.Timer(timeout, sendans)

    t.start()
    answer = input(f"You have {timeout} seconds to answer: ")
    t.cancel()
    t.join()

    #입력이 있었다면 입력에 따라 plug 상태 변경
    client_socket.send(answer.encode())
```

6.1.2. server.py

6.1.2.1. start_server()

server에 접속시 가장 먼저 실행되는 함수이다. select를 이용하여 다중통신이 가능하도록 구현하였다.

6.1.2.2. handle_client()

새 client의 연결 요청이 들어왔을 때 처리해주는 함수이다.

smartPlug.main()에 IP_P100, email, password를 파라미터로 넘겨 server와 사용자의 스마트 플러그를 연결한다.

6.1.2.2.1. Multi Thread 구조

smartPlug.py 의 흐름은 동시에 다중 작업 수행을 계속해야 하기 때문에 멀티Thread 실행으로 진행된다.

```
# 가장 마지막에 받은 패킷을 사용하기 위해 LIFO 이용
buffer = queue.LifoQueue()

# 멀티스레드 사용
# th0은 패킷을 캡쳐하는 스레드
# th1은 발생한 패킷을 사용자 패턴과 매치하는 스레드
th0 = threading.Thread(target=smartPlug.getPacket, args=(buffer, addr))
th0.start()
th1 = threading.Thread(target=smartPlug.checkTime, args=(buffer, conn))
th1.start()
th0.join()
th1.join()
break
```

위의 그림과 같이 먼저 main을 실행하여 준비 과정을 실행한 후 getPacket을 수행하는 Thread와 checkTime을 수행하는 Thread를 생성하여 동시에 진행할 수 있게 한다. 이때 inter Thread communication 을 수행하기 위해 queue 라이브러리의 LifoQueue를 사용한다. 가장 마지막에 받은 패킷을 처리하기 위해 스택 구조인 LIFO를 택했다.

6.1.3. pattern.py

6.1.3.1. Class plug

잠재적으로 여러 플러그를 관리 할 가능성이 있기 때문에 클래스로 관리하여 각 플러그 객체 마다 사용 패턴을 저장할 수 있게 한다.

- getPattern()

사용자의 플러그 사용 시간 입력 받는다. 사용자의 입력 편의성을 위해 요일 별 사용 시간 패턴으로 입력받는다. 이때 사용시간은 사용시작시간, 사용 종료시간을 시간(hour)단위로 입력받는다. 입력

받은 문자열을 int형으로 변환 하여 필드드에 저장함 으로써 이후 matchPattern 메소드에서 연산을 가능하게 한다.

- getPattern_file()

기능은 위의 getPattern과 유사하다. 패턴을 사용자로부터 터미널에서 직접 입력받지 않고 패턴을 저장해놓은 파일로 부터 가져오는 메소드이다. 상황에 따라 사용자가 직접 터미널을 사용하지 않고 텍스트 파일만 전송하여 서비스를 이용할 수 있게 하기 위해 구현해놓았다.

- printPattern()

해당 객체 필드에 저장되어 있는 사용 패턴을 출력해주는 메소드이다. 프로그램 실행중 필요에 따라 입력받은은 패턴을 확인해야할때 사용하기 위해 구현했다.

- matchPattern()

필드에 저장되어 있는 사용 패턴과 매개변수로 들어온 시간 정보를 비교하여 해당 시각이 사용 패턴 시간 사이의 시각이 아니라면 true를 리턴하고, 그렇지 않고 시각이 사용패턴에 부합한다면 false를 리턴한다.

6.1.4. smartPlug.py

6.1.4.1. getPacket()

스마트 플러그 p100에서 발생한 패킷을 캡쳐하는 기능을 한다. 패킷 캡쳐를 위해 파이썬 패킷 캡쳐 라이브러리 scapy를 이용한다.

scapy의 sniff()를 호출 하는데 이때 sniff의 매개변수로 다음을 할당한다.

```
#공유자원에서 P100의 ip 불러오기
sniff(iface='로컬 영역 연결* 2', prn=lambda pkt: packet_callback(pkt, buffer,addr),
      filter=f'''ip host {p100.getDeviceInfo()['result']['ip']}
      and not ip host {os.getenv('IP_local')}''')
```

iface: 패킷을 캡쳐할 네트워크 인터페이스

packet_callback: 패킷이 캡쳐되면 실행될 패킷캡쳐 콜백함수

filter: ip host를 통해 IP_P100에서 송/수신되는 패킷만 캡쳐하고, not ip host로 IP_local 를 할당하여 plugstate를 확인하는 등의 프로그램에서 스마트 플러그에 접근하는 패킷은 캡쳐하지 않도록 제외한다.

6.1.4.2. packet_callback()

getPacket함수에서 sniff를 실행하여 패킷을 캡쳐 했을때 호출 될 패킷 캡쳐 콜백 함수이다.

해당 제품은 TLS v1.2 프로토콜을 사용하여 암호화 통신을 하기 때문에 데이터를 한 번 주고받는 과정에서 10개가량의 패킷이 발생하게 된다. 발생하는 패킷마다 처리하게 된다면 시스템 오버헤드가 커질 것으로 예상되었다. 따라서 lastpacket을 사용해 가장 마지막에 처리된 패킷의 시간을 기록하고, time_scope를 2초로 설정하여 lastpacket을 받고 2초 내에 오는 패킷은 처리하지 않는 로직을 구현하였다.

```python
# 패킷 캡쳐 콜백 함수
def packet_callback(packet, buffer,addr):
    global lastpacket  # 전역 변수로 선언
    time_scope = timedelta(seconds=2)

    # 패킷이 처음 발생하고 2초가 지나지 않으면 무시
    # 한 동작 당 한꺼번에 여러개의 패킷이 발생하기 때문에...
    if datetime.fromtimestamp(packet.time) < lastpacket+time_scope:
        return
    else:
        raw_packet_time = datetime.fromtimestamp(packet.time)
        lastpacket = raw_packet_time
        packet_time = raw_packet_time.strftime("%Y-%m-%d %H:%M")

        print("Packet time:", packet_time)
        print("Client ",addr,": ",packet.summary())  # 캡처한 패킷 보여주기
        buffer.put(packet_time)
```

6.1.4.3. checkTime()

checkTime 은 패킷을 전달받아 다음과 같은 기능을 한다.

첫번째로, pattern 과 비교해 올바른 사용 시각인지 판단하여 만약 패턴과 알맞지 않다면 정상적인 사용시간이 아님을 알리는 기능이다.

두번째로, 패킷과 패킷사이의 시간 측정으로 설정한 최대 사용 시간(limit_time)을 초과했을 때 위험을 알리는 기능이다.

checkTime의 실행 흐름은 다음과 같다. 위에 설명한 두가지 기능을 계속 유지해야 하기 때문에 while 문으로 연속적으로 상태를 검사한다. 반복마다 경과한 시간을 계산하여 만약 경과한 시간이 설정한 limit_time을 초과 하였을 경우는 사용시간을 넘어 위험을 감지했다는 알림을 발생한다.

```
#사용불가시간에 사용함, 클라이언트에게 알림
if(myPlug.matchPattern(getDay, getTime)):
    status = "You are using at the wrong time. Are you sure you are?"
    clnt.send(status.encode())

    answer  = None

    while True:
        if(clnt):
            answer = clnt.recv(1024).decode()
            if(answer == "Yes"):
                break
            else :
                status = "Turn off the plug"
                clnt.send(status.encode())
                p100.turnOff()
                plugState=0
                break
```

또한 반복마다 buffer에 값이 있는지 확인하여 만약 값이 있다면 버퍼안의 값을 가져오고 그 값을 요일, 시간 정보로 가공한다. 그리고 getDeviceInfo 메서드를 통해 현재 패킷으로 부터 변화된 플러그 상태가 on인지 off인지지를 판단하여 plugstate를 관리하고 만약 플러그가 켜진 상황이라면 matchPattern 메소드를 호출하여 pattern과 비교하여 올바른 사용시간인지 검사한다. 만약 matchPattern이 true를 리턴한다면 잘못된 사용시간으로 위험 알림을 발생시키고 p100 스마트 플러그를 종료시킨다.

6.1.4.4. main()

smartPlug.py 의 main 함수는 앞서 설명한 기능을 하기 전 먼저 실행하는 준비과정들이다. p100 플러그를 제어하고 발생하는 패킷을 가져올 수 있게 handshake와 login 과정을 수행한다. 이는 앞에서 소개한 오픈소스 p100 라이브러리를 사용하여 구현하였다. getDeviceInfo를 통해 초기 p100 플러그의 on/off 상태를 plugstate에 저장한다.

```python
def main(IP_P100, email, pwd, myPluglocal):
    global p100
    p100 = PyP100.P100(IP_P100, email, pwd)

    p100.handshake()
    p100.login()

    global plugState
    global myPlug
    myPlug = myPluglocal #파라미터로 받은 값 global선언

    info = p100.getDeviceInfo() #연결된 플러그에 대한 정보를 info에 저장
    if info['result']['device_on']: #info (json형태)
        print("State: Turn On")
        plugState = 1
    else:
        print("State: Turn Off")
        plugState = 0

    pluginfo = p100.getDeviceName() #Returns the name of the connected plug
    print(pluginfo)
```

7. 프로젝트 구현

7.1. server를 열고, client가 접속하여 플러그의 IP, email, password, 사용자가 플러그를 사용하는 시간대를 입력한다.

```
C:\Users\leeso\NP_Project\NP-smart_plug_packet>
python server.py
Connected with client: ('127.0.0.1', 4822)
```

```
C:\Users\leeso\NP_Project\NP-smart_plug_packet>
python client.py
Server connect!!
Please enter yout data.
IP P100: 192.168.137.183
email: leeso3076@naver.com
password:
```

```
IP P100: 192.168.137.183
email: leeso3076@naver.com
password:[redacted]
-------------사용자가 사용하는 시간 입력----------------
월요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
화요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
수요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
목요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
금요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
토요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
일요일의 사용하는 시작 시간, 끝시간을 입력하세요. 10 17
월요일 start time: 10, end time: 17
화요일 start time: 10, end time: 17
수요일 start time: 10, end time: 17
목요일 start time: 10, end time: 17
금요일 start time: 10, end time: 17
토요일 start time: 10, end time: 17
일요일 start time: 10, end time: 17
```

7.2. server는 플러그 패킷 모니터링을 시작하고, 패킷이 발생할때마다 플러그의의 현재 상태를 출력한다.

```
State: Turn Off
스마트 플러그
Start getPacket
패킷 캡쳐 시작:  Start checkTime
2023-05-27 20:49:54.862843
Packet time: 2023-05-27 20:50
Client  ('127.0.0.1', 4954) :  Ether / IP / TCP 192.168.137.183:52437 > 3.0.189.238:https PA / Raw
check Time:  2023-05-27 20:50:00
State: Turn Off
```

192.168.137.183: 플러그 P100의 IP

3.0.189.238: 어플을 통해 플러그를 제어하고 있는 사용자의 IP

https PA/Raw: 통신 프로토콜 (현재 TLS를 사용한다)

7.3. 사용자 플러그를 켰는데, 이전에 입력받은 사용 패턴 시간에 벗어난다면 사용자에게 알린다.

```
You are using at the wrong time. Are you sure you are?
You have 60 seconds to answer:
```

Yes를 입력하면 계속 진행, Yes 제외한 값을 입력하면 플러그 Turn off, Timeout시 시스템을 종료한다.

```
Start SmartPlug init...
Start Packet Monitoring...
You are using at the wrong time. Are you sure you are?
You have 60 seconds to answer: Yes
You are using at the wrong time. Are you sure you are?
You have 60 seconds to answer: No
Turn off the plug
You are using at the wrong time. Are you sure you are?
You have 60 seconds to answer:
C:\Users\leeso\NP_Project\NP-smart_plug_packet>
```

7.4. 이러한 과정을 반복하며 패킷을 모니터링하고, 사용자의 패턴과 비교한다.

7.5. 통신 중 다른 client가 연결하여도 동일하게 진행된다.

8. 프로젝트 설계 및 구현 난이도 평가

8.1. 설계 복잡성 평가

이 프로젝트는 다양한 구성 요소와 모듈로 구성되어 있다. 사용자로부터 플러그 사용 패턴을 입력받는 부분은 사용자 인터페이스와 관련하여 설계되어야 한다. client - server 통신과 다양한 함수로 구성되어 있기때문에 코드 흐름 설계 자체는 까다로웠으나 사용자에게는 단순한 입력과 출력으로 나타나도록 설계하였다.

사용자는 편리하게 플러그 사용 패턴을 입력하고 관리할 수 있어야 하며, 이를 효율적으로 처리하기 위한 설계 고려 사항이 필요하다. 이에 따라 사용자가 요일별로 사용 시간의 시작과 끝을 입력하도록 설계하였다.

또한, server는 패킷을 캡처하는 Thread1과 패킷 발생 시간과 사용자 패턴을 비교하는 Thread2를 생성하는데, 이는 다중 Thread 환경에서의 설계 복잡성을

동반한다. 이상 패킷 발생 시 사용자에게 알림을 보내고 그에 따라 처리하는 부분에서 사용자와의 상호작용 및 오류 처리 등을 고려해야 하는 설계 복잡성이 존재한다. 각 모듈과 구성 요소 간의 상호작용과 데이터 처리의 정확성, 안정성, 성능 등을 고려하여 설계하였다.

8.2. 구현 난이도 평가

본 프로젝트에서는 먼저 기술적인 측면에서, 다른 언어에 비해 비교적 쉬운 문법을 가진 Python으로 개발을 하였다. Python은 네트워크 패킷 캡처와 분석을 위한 다양한 라이브러리가 있고, 스마트 플러그 P100의 오픈소스가 있는 간편한 언어이기 때문에 프로젝트 접근성은 높지 않았다.

개발 환경 측면에서는 다중 Thread 환경에서 프로그램을 구현했고 사용자 인터페이스를 고려하여 설계했기 때문에 동기화 및 상호 작용 문제에 대한 이해와 처리가 필요했다. 또한 본 프로젝트는 스마트 플러그와의 상호 작용을 위해 Wi-Fi, 네트워크 통신 프로토콜, 암호화 프로토콜에 대한 이해와 client - server 구축에 대한 이해가 필요했다. 네트워크 프로그래밍 수업 시간에 배운 내용도 많은 도움이 됐지만, 과목과 다른 언어를 이용하여 개발을 하였기에 추가적인 공부가 더 필요하였다.

Scapy와 PyP100 오픈소스를 주로 활용하였기에 오픈소스에 대한 이해가 필요했다. 버전 오류나 통신 영역 등의 차이로 각자 개발한 코드를 합치는 데에 어려움을 겪었다.

9. 결론

9.1. 프로젝트 결과

9.1.1. client - server 연결 및 소켓 통신

client는 server에 접속하여 스마트 플러그에 대한 정보와 사용 패턴을 제공하고, server는 해당 정보를 기반으로 패킷 모니터링과 사용 패턴을 비교하는 Thread를 생성한다. 이 통신은 TCP 소켓을 이용해 이루어진다.

9.1.2. 패킷 모니터링, 사용 패턴 비교 Thread

스마트 플러그에서 발생하는 패킷을 모니터링하고, 사용 패턴에 벗어난다면 client에게 알린다. client는 해당 알림에 대한 응답을 보낼 수 있으며, 응답에 따라 플러그를 제어한다.

9.1.3. 다중 client 지원

server는 select 모듈을 통해 다중 client 연결을 지원한다. 여러 client가 동시에 server에 접속하고 요청을 처리할 수 있다.

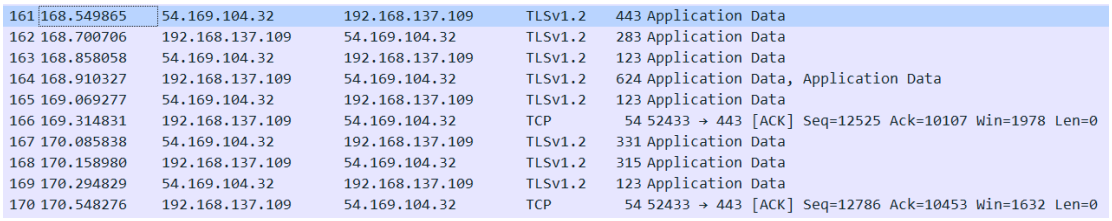
9.2. 개발 과정의 어려움

프로젝트 주제가 자유 주제였기 때문에 어떤 프로그램을 개발할 지 정확히 결정하는데 어려움을 겪었고 주제의 범위와 구현해야 할 수준을 정하는 과정에서 시간이 조금 걸렸다. 특히, IoT기기를 활용한 프로젝트였기 때문에 해당 IoT 기기와 관련된 개념, 통신 프로토콜, 데이터 포맷 등에 대한 이해가 필요했다. 공식 문서를 읽어도 패킷 전송 방식, 자세한 통신 과정에 대한 설명이 없었기에 Wireshark를 통해 스마트플러그에서 발생하는 패킷을 읽는 것부터 시작하였다.

9.2.1. WIreshark 패킷 읽기

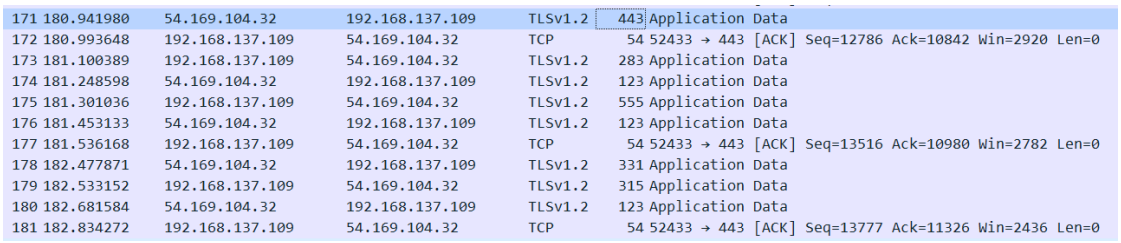
프로젝트를 진행하기 위해서는 플러그가 켜져있는지, 꺼져있는지 구분해야 하기 때문에 플러그의 전원을 켰을 때의 패킷과 껐을 때의 패킷의 특징을 찾고자 했다.

- 어플에서 껐을 때 발생하는 패킷



161	168.549865	54.169.104.32	192.168.137.109	TLSv1.2	443	Application Data
162	168.700706	192.168.137.109	54.169.104.32	TLSv1.2	283	Application Data
163	168.858058	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
164	168.910327	192.168.137.109	54.169.104.32	TLSv1.2	624	Application Data, Application Data
165	169.069277	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
166	169.314831	192.168.137.109	54.169.104.32	TCP	54	52433 → 443 [ACK] Seq=12525 Ack=10107 Win=1978 Len=0
167	170.085838	54.169.104.32	192.168.137.109	TLSv1.2	331	Application Data
168	170.158980	192.168.137.109	54.169.104.32	TLSv1.2	315	Application Data
169	170.294829	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
170	170.548276	192.168.137.109	54.169.104.32	TCP	54	52433 → 443 [ACK] Seq=12786 Ack=10453 Win=1632 Len=0

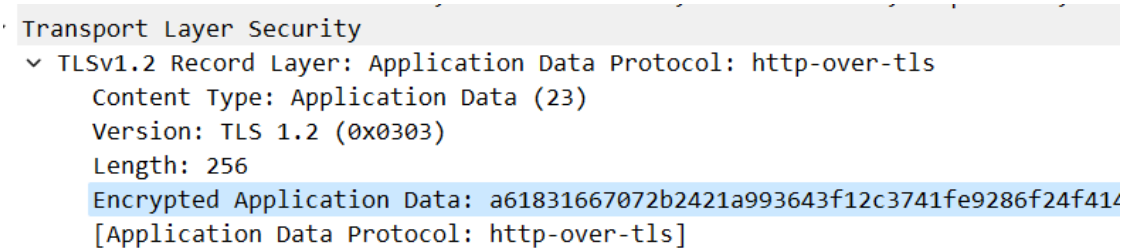
- 어플에서 켰을 때 발생하는 패킷



171	180.941980	54.169.104.32	192.168.137.109	TLSv1.2	443	Application Data
172	180.993648	192.168.137.109	54.169.104.32	TCP	54	52433 → 443 [ACK] Seq=12786 Ack=10842 Win=2920 Len=0
173	181.100389	192.168.137.109	54.169.104.32	TLSv1.2	283	Application Data
174	181.248598	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
175	181.301036	192.168.137.109	54.169.104.32	TLSv1.2	555	Application Data
176	181.453133	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
177	181.536168	192.168.137.109	54.169.104.32	TCP	54	52433 → 443 [ACK] Seq=13516 Ack=10980 Win=2782 Len=0
178	182.477871	54.169.104.32	192.168.137.109	TLSv1.2	331	Application Data
179	182.533152	192.168.137.109	54.169.104.32	TLSv1.2	315	Application Data
180	182.681584	54.169.104.32	192.168.137.109	TLSv1.2	123	Application Data
181	182.834272	192.168.137.109	54.169.104.32	TCP	54	52433 → 443 [ACK] Seq=13777 Ack=11326 Win=2436 Len=0

이를 통해 해당 플러그는 TLS v1.2로 암호화되어 통신한다는 것을 알 수 있었다. 단순히 패킷으로만 봤을 때는 큰 차이가 없어 보였다.

아래는 위에서 발생한 패킷 중 한 패킷에 대한 상세 정보이다.



```
Transport Layer Security
  TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 256
      Encrypted Application Data: a61831667072b2421a993643f12c3741fe9286f24f414
      [Application Data Protocol: http-over-tls]
```

TLS 프로토콜은 Application Layer와 Transport Layer 사이에 위치하여 Application 에서 발생한 데이터를 암호화 하는데 사용된다. Turn on/off에 대한 정보가 담겨있을 Application data 부분이 전부 암호화되어 보이지 않았다. TLS를 복호화 하는 것은 불가능에 가깝기 때문에 패킷 길이, 헤더 등에 담겨있는 정보를 비교하고자 하였지만 유의미하게 차이나지는 않아 보였다.

9.2.2. PyP100 API를 활용한 분석

앞서 소개한 바와 같이 PyP100 API는 본 프로젝트에서 이용하고 있는 스마트 플러그 P100에 원격으로 연결하고, 제어할 수 있게 해준다. API 코드를 읽어보니 RSA 키를 직접 생성하고, AES와 SHA1 등을 이용해 플러그와 통신하는 것을 알 수 있었다. 처음에는 API 코드 자체에서 생성되는 RSA 키를 받아와 Wireshark에서 접근하려고 했지만, 로컬에서만 접근 가능할 뿐더러 정석적인 방법이 아니라고 생각했다.

따라서 Wireshark에서 직접 접근하지 않고 scapy 라이브러리를 활용해 패킷을 직접 스니핑하고, API를 활용해 플러그 제어와 상태값을 불러오는 방식으로 변경하였다.

9.3. 향후 개선 및 확장 방향

9.3.1. 향후 개선할 점

시스템의 정확성과 신뢰성을 더욱 향상시키기 위해 패킷 분석 알고리즘의 성능 개선이 필요하다. 더 정확하고 신속한 위험 감지를 위해 머신 러닝이나 딥 러닝 기법을 도입하는 것을 고려해볼 수 있다. 사용자와의 상호 작용에 대한 세밀한 분석과 행동 패턴 모델링을 통해 신뢰할 수 있는 사용자 행동과 비정상적인 활동을 정확하게 구별할 수 있는 강력한 시스템 개발이 필요하다. 보안 취약점에 대한 강화가 필요하다. 시스템이 해킹이나 악의적인 공격으로부터 안전하게 보호되도록 추가적인 보안 기능이 필요하다.

9.3.2. 확장 방향

추가적인 위험 감지 및 예방 기능을 개발하여 사용자의 안전을 더욱 향상시킬 수 있다. 예를 들어, 화재, 가스 누출, 침입 등과 관련된 센서를 통합하여 스마트 플러그 시스템이 다양한 위험을 감지하고 적절한 조치를 취하도록 할 수 있다. 더 나아가 IoT를 활용한 스마트홈까지 영역을

확장한다면 종합적으로 안전과 위험 차단 서비스를 제공하는 서비스가 될 수 있을 것이다.

또한, 해당 프로젝트는 API 형태로 제공될 수도 있다. 기존의 PyP100 API와는 다르게 사용자의 패턴에 따른 원격 플러그 상태 제어의 목표를 갖고있다. 따라서 더 안전하게 스마트홈을 자동화시키거나, 불필요한 에너지 사용을 관리하는 등 여러 부분에 사용할 수 있다.

복지 기관 등과 협력하여 사용자가 사용 패턴대로 사용하지 않는다면 바로 알리고, 조취를 취할 수 있도록 하여 더 실제적인 서비스가 될 수 있다.

10. 프로젝트 정의서 수정 부분

10.1. Multi Process -> Multi Thread

buffer, plugState, myPlug 등의 공유자원을 효율적으로 접근하기 위해 Multi Thread를 활용하였다. Multi Thread를 사용함으로써 병렬적인 작업 수행이 가능해졌다. 각 Thread는 독립적으로 실행되며, 따라서 서로 다른 작업을 동시에 처리할 수 있다. 따라서 Multi Thread를 적용한 이 프로그램은 공유 자원에 대한 효율적인 접근과 동시성 작업 처리를 가능하게 함으로써 성능 향상과 안정성을 동시에 추구할 수 있다.

10.2. Android Application -> User Terminal Input

기존에 Android 애플리케이션으로 제작하였을 경우, 사용자는 Tapo 앱과 해당 프로젝트 앱 두 가지를 설치해야 하는데, 이는 사용자의 디바이스 공간을 차지하고, 복잡한 설치 과정을 거쳐야 하는 단점을 가지고 있다. 이러한 문제를 해결하기 위해 중앙server 형식으로 변경하였다. 이제 사용자는 별도의 애플리케이션 설치 과정을 거치지 않고도, 간편하게 유저 터미널에서 입력만을 수행하면 되는 것이다. 이러한

중앙서버 형식은 하나의 서버를 통해 모든 작업과 관리가 이루어지므로 효율적인 운영과 모니터링이 가능해졌다.

10.3. client - server 통신 추가

프로젝트 정의서에서는 사용자가 smartPlug 코드로 바로 접근할 수 있도록 설계하였지만, 본 프로젝트의 웹/앱 확장성을 열어두기 위해 client와 server가 통신하는 형태로 변경하였다. 또한 다중통신을 추가하여 코드를 더 효율적으로 사용할 수 있도록 하였다.

11. 참고 문헌 및 출처

1) 조효진, "네트워크 프로그래밍", 숭실대학교, 2023-1

2) 이세율, 신지영, 김지용, 강수용, 한혁. (2021). 스마트 IoT 기기의 동작 검증을 위한 패킷 분석 프로그램. 한국디지털콘텐츠학회 논문지, 22(5), 889-896.

3) 가브리엘, 김아론, 홍신. (2020). 스마트 홈 보안을 위한 패킷-스니핑 기반 네트워크 사용 모니터링 서비스의 설계와 구현. 한국정보과학회 학술발표논문집, (), 1393-1395.

4) 문중기, 이양돈, 전경진. 스마트 플러그에 연결된 전자 기기를 검출하기 위한 방법 및 장치. WO2017034371A1. (2017.03.02)

5) seoulsolution. "'외로운 죽음 막는다' 1인가구 고독사 예방사업 강화". 복지정책실. (2021.05.13)

6) 박현철. (2018). Python 기반의 Scapy를 이용하여 웹 기반 IoT 패킷분석 시스템 개발" 세종대학교 대학원.

7) Python PyP100 API, https://github.com/fishbigger/TapoP100

8) TP-Link Tapo P100, https://www.tp-link.com/kr/home-networking/smart-plug/tapo-p100

9) 국가 지표 체계, 1인가구 비율(2000-2021), https://www.index.go.kr/unify/idx-info.do?idxCd=5065