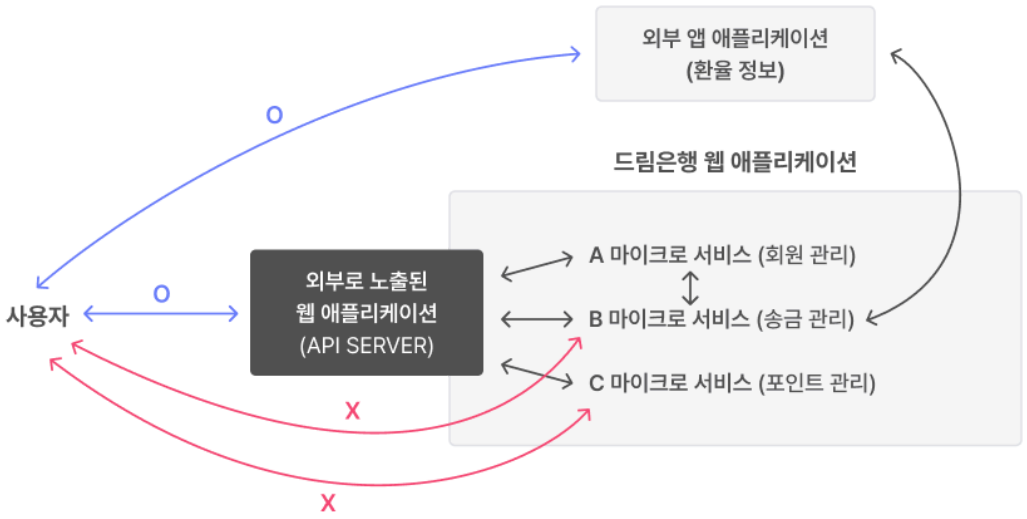
SSRF (Server Side Request Forgery)

- 웹 개발 언어는 HTTP 요청을 전송하는 라이브러리를 제공한다.
  - 각 언어별 HTTP 라이브러리는 PHP의 php-curl, NodeJS는 http, 파이썬은 urllib, requests를 예로 들 수 있다.
    - 이러한 라이브러리는 HTTP 요청을 보낼 뿐만 아니라 서버와 서버간 통신을 위해 사용되기도 한다.
      - 일반적으로 다른 웹 어플리케이션에 존재하는 리소스를 사용하기 위한 목적으로 통신한다.
        - 예를 들어, 마이크로서비스 간 통신, 외부 API 호출, 외부 웹 리소스 다운로드 등이 있다.

- 기존의 웹 서비스는 단일 서비스로 구현될 수 있었지만, 최근의 웹 서비스는 지원하는 기능이 증가함에 따라 구성요소가 증가했다.
  - 이에 따라 관리 및 코드의 복잡도를 낮추기 위해 마이크로 서비스들로 웹 서비스를 구현하는 추세이다.
    - 이때 각 마이크로서비스는 주로 HTTP, GRPC 등을 사용해 API 통신을 한다.



- 이처럼 서비스 간 HTTP 통신이 이뤄질 때 요청 내에 이용자의 입력 값이 포함될 수 있다.
    - 이용자의 입력값으로 포함되면, 개발자가 의도하지 않은 요청이 전송될 수 있다.
        - Server Side Request Forgery (SSRF)는 웹 서비스의 요청을 변조하는 취약점으로, 브라우저가 변조된 요청을 보내는 CSRF와는 다르게 웹 서비스의 권한으로 변조된 요청을 보낼 수 있다.

- 최근의 대다수 서비스들은 마이크로서비스로 구조를 많이 바꾸고, 새롭게 개발하는 추세이기 때문에 SSRF 취약점의 파급력이 더욱 높아지고 있다.

> 💡 마이크로서비스란?
>
> - 마이크로 서비스는 소프트웨어가 잘 정의된 API를 통해 통신하는 소규모의 독립적인 서비스로 구성되어 있는 경우의 소프트웨어 개발을 위한 아키텍처 및 조직의 접근 방식이다.
>   - 이러한 서비스는 독립적인 소규모 팀에서 보유한다.

Server-Side Request Forgery (SSRF)

- 웹 서비스는 외부에서 접근할 수 없는 내부망의 기능을 사용할 때가 있다.
    - 내부망의 기능은 백오피스 서비스를 예로 들 수 있다.
        - 백오피스 서비스는 관리자 페이지라고도 불리며, 이용자의 행위가 의심스러울 때 해당 계정을 정지시키거나 삭제하는 등 관리자만의 수행할 수 있는 모든 기능을 구현한 서비스이다.
            - 이러한 서비스는 관리자만 이용할 수 있어야 하기 때문에 외부에서 접근할 수 없는 내부망에 위치한다.

- 웹 서비스는 의심스러운 행위를 탐지하고 실시간으로 대응하기 위해 백오피스의 기능을 실행할 수 있다.
    - 다시 말해, 웹 서비스는 외부에서 직접 접근할 수 없는 내부망 서비스와 통신할 수 있다.

        - 만약 공격자가 SSRF 취약점을 통해 웹 서비스의 권한으로 요청을 보낼 수 있다면, 공격자는 외부에서 간접적으로 내부망 서비스를 이용할 수 있고, 이는 곧 기업에 막대한 피해를 입힐 수 있다.
    - 웹 서비스가 보내는 요청을 변조하기 위해서는 요청 내에 이용자의 입력 값이 포함되어야 한다.
        - 입력값이 포함되는 예시로는 웹 서비스가 이용자가 입력한 URL에 요청을 보내거나 요청을 보낼 URL에 이용자 번호와 같은 내용이 사용되는 경우, 그리고 이용자가 입력한 값이 HTTP Body에 포함되는 경우로 나눠볼 수 있다.

이용자가 입력한 URL에 요청을 보내는 경우

분석

```
# pip3 install flask requests # 파이썬 flask, requests 라이브러리를 설치하는 명령입
# python3 main.py # 파이썬 코드를 실행하는 명령입니다.

from flask import Flask, request
import requests

app = Flask(__name__)



@app.route("/image_downloader")
def image_downloader():
    # 이용자가 입력한 URL에 HTTP 요청을 보내고 응답을 반환하는 페이지 입니다.
    image_url = request.args.get("image_url", "") # URL 파라미터에서 image_url 값
    response = requests.get(image_url) # requests 라이브러리를 사용해서 image_
    return ( # 아래의 3가지 정보를 반환합니다.
        response.content, # HTTP 응답으로 온 데이터
        200, # HTTP 응답 코드
        {"Content-Type": response.headers.get("Content-Type", "")}, # HTTP 응답
    )



@app.route("/request_info")
def request_info():
    # 접속한 브라우저(User-Agent)의 정보를 출력하는 페이지 입니다.
```

```
    return request.user_agent.string


app.run(host="127.0.0.1", port=8000)
```

- 위의 코드는 이용자가 전달한 URL에 요청을 보내는 예제 코드이다.
  - 코드를 살펴보면, 두 가지의 엔드포인트가 존재한다.
    - 아래는 각 엔드포인트에 대한 설명이다.

- image_downloader
  - 이용자가 입력한 image_url을 requests.get( ) 함수를 사용해 GET 메소드로 HTTP 요청을 보내고, 응답을 반환한다.
    - 브라우저에서 다음과 같은 URL을 입력하면, 드림핵 페이지에 요청을 보내고 응답을 반환한다.

      ```
      http://127.0.0.1:8000/image_downloader?image_url=https://dreamh
      ```

- request_info
  - 웹 페이지에 접속한 브라우저의 정보 (User-Agent)를 반환한다.
    - 브라우저를 통해 해당 엔드포인트에 접속하면 접속하는데에 사용된 브라우저의 정보가 출력된다.

      ```
      Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.3
      ```

- 문제점 확인
  - 아래와 같이 image_downloader 엔드포인트의 image_url에 request_info 엔드포인트 경로를 입력해본다.

    ```
    http://127.0.0.1:8000/image_downloader?image_url=http://127.0.0.1:800
    ```

        - 해당 경로에 접속하면 image_downloader에서는 http://127.0.0.1:8000/request_info URL에 HTTP 요청을 보내고 응답을 반환한다.
            - 반환한 값을 확인해보면 브라우저로 **request_info** 엔드포인트에 접속했을 때와 다르게 브라우저 정보가 `python-requests/<LIBRARY_VERSION>`인 것을 확인할 수 있다.
            - 접속한 브라우저 정보로 `python-requests`가 출력된 이유는 웹 서비스에서 HTTP 요청을 보냈기 때문이다.
                - 이처럼 이용자가 웹 서비스에서 사용하는 마이크로서비스의 API 주소를 알아내고, `image_url`에 주소를 전달하면 외부에서 직접 접근할 수 없는 마이크로서비스의 기능을 임의로 사용할 수 있다.

웹 서비스의 요청 URL에 이용자의 입력 값이 포함되는 경우

분석

```
INTERNAL_API = "http://api.internal/"
# INTERNAL_API = "http://172.17.0.3/"


@app.route("/v1/api/user/information")
def user_info():
    user_idx = request.args.get("user_idx", "")
    response = requests.get(f"{INTERNAL_API}/user/{user_idx}")


@app.route("/v1/api/user/search")
def user_search():
    user_name = request.args.get("user_name", "")
    user_type = "public"
    response = requests.get(f"{INTERNAL_API}/user/search?user_name={user_
```

- 위의 코드는 이용자의 입력 값이 포함된 URL에 요청을 보내는 예제 코드이다.
    - 코드를 살펴보면 두 가지 엔드포인트가 존재한다.
        - 아래는 각 엔드포인트에 대한 설명이다.

user_info

- 이용자가 전달한 user_idx값을 내부 API의 URL 경로로 사용한다

  ```
  http://x.x.x.x/v1/api/user/information?user_idx=1
  ```

  - 이용자가 위와 같이 user_idx를 1로 설정하고 요청을 보내면 웹 서비스는 다음과 같은 주소에 요청을 보낸다.

    ```
    http://api.internal/user/1
    ```

user_search

- 이용자가 전달한 user_name 값을 내부 API의 쿼리로 사용한다.

  ```
  http://x.x.x.x/v1/api/user/search?user_name=hello
  ```

  - 이용자가 위와 같이 user_name을 “hello”로 설정하고 요청을 보내면, 웹 서비스는 다음과 같은 주소에 요청을 보낸다.

    ```
    http://api.internal/user/search?user_name=hello&user_type=public
    ```

문제점 확인

- 웹 서비스가 요청하는 URL에 이용자의 입력 값이 포함되면, 요청을 변조할 수 있다.
  - 이용자의 입력 값 중 URL의 구성 요소 문자를 삽입하면 API 경로를 조작할 수 있다.
    - 예를 들어, 예시 코드의 user_info( ) 함수에서 user_idx에 ../search를 입력할 경우 웹 서비스는 다음과 같은 URL에 요청을 보낸다.

      ```
      http://api.internal/search
      ```

      - ..는 상위 경로로 이동하기 위한 구분자로, 해당 문자로 요청을 보내는 경로를 조작할 수 있다.

        - 해당 취약점은 경로를 변조한다는 의미에서 Path Traversal이라고 불린다.
            - 이 외에도, # 문자를 입력해 경로를 조작할수 있다.
                - 예를 들어, user_search 함수에서 user_name에 secret&user_type=private#을 입력할경우 웹 서비스는 다음과 같은 URL에 요청을 보낸다.

```
http://api.internal/search?user_name=secret&user_
```

        - #문자는 Fragment Identifier 구분자로, 뒤에 붙는 문자열은 API 경로에서 생략된다.
            - 따라서 해당 URL은 실제로 아래와 같은 URL을 나타낸다.

```
http://api.internal/search?user_name=secret&user_t
```

웹 서비스의 요청 Body에 이용자의 입력값이 포함되는 경우

분석

```
# pip3 install flask
# python main.py

from flask import Flask, request, session
import requests
from os import urandom


app = Flask(__name__)
app.secret_key = urandom(32)
INTERNAL_API = "http://127.0.0.1:8000/"
header = {"Content-Type": "application/x-www-form-urlencoded"}


@app.route("/v1/api/board/write", methods=["POST"])
```

```
def board_write():
    session["idx"] = "guest" # session idx를 guest로 설정합니다.
    title = request.form.get("title", "") # title 값을 form 데이터에서 가져옵니다.
    body = request.form.get("body", "") # body 값을 form 데이터에서 가져옵니다.
    data = f"title={title}&body={body}&user={session['idx']}" # 전송할 데이터를 구
    response = requests.post(f"{INTERNAL_API}/board/write", headers=header
    return response.content # INTERNAL API 의 응답 결과를 반환합니다.


@app.route("/board/write", methods=["POST"])
def internal_board_write():
    # form 데이터로 입력받은 값을 JSON 형식으로 반환합니다.
    title = request.form.get("title", "")
    body = request.form.get("body", "")
    user = request.form.get("user", "")
    info = {
        "title": title,
        "body": body,
        "user": user,
    }
    return info


@app.route("/")
def index():
    # board_write 기능을 호출하기 위한 페이지입니다.
    return """
        <form action="/v1/api/board/write" method="POST">
            <input type="text" placeholder="title" name="title"/><br/>
            <input type="text" placeholder="body" name="body"/><br/>
            <input type="submit"/>
        </form>
    """


app.run(host="127.0.0.1", port=8000, debug=True)
```

- 위의 코드는 이용자의 입력 값이 요청의 Body에 포함되는 예제 코드이다.

- 코드를 살펴보면 세 가지의 엔드포인트가 존재한다.
  - 아래는 각 엔드포인트에 대한 설명이다.

board_write

- 이용자의 입력값을 HTTP Body에 포함하고 내부 API로 요청을 보내다.
  - 전송할 데이터를 구성할 때 세션 정보를 “guest”로 설정한다.

internal_board_write

- board_write 함수에서 요청하는 내부 API를 구현한 기능이다.
  - 전달된 title, body, 그리고 계정 이름을 JSON 형식으로 변환하고 반환한다.

index

- board_write 기능을 호출하기 위한 인덱스 페이지이다.

문제점 확인

- 위 코드를 실행하고 다음 URL에 접속하면, titlte과 body를 입력하는 페이지가 표시된다.

```
http://127.0.0.1:8000
```

  - 입력창에 값을 입력하고 제출 버튼을 누르면 다음과 같은 응답을 확인할 수 있다.

```
{ "body": "body", "title": "title", "user": "guest" }
```

    - 요청을 전송할 때 세션 정보를 “guest”로 설정했기 때문에 user가 “guest”인 것을 확인할 수 있다.
      - 예시 코드를 살펴보면, 내부 API로 요청을 보내기 전에 다음과 같이 데이터를 구성하는 것을 확인할 수 있다.

```
data = f"title={title}&body={body}&user={session['idx']}
```

    - 데이터를 구성할 때 이용자의 입력값인 title, body, 그리고 user의 값을 파라미터 형식으로 설정한다.
        - 이로 인해 이용자가 URL에서 파라미터를 구분하기 위해 사용하는 구분 문자인 &를 포함하면 설정되는 data의 값을 변조할 수 있다. title에서 title&user=admin을 삽입하면 다음과 같이 data가 구성된다.

```
title=title&user=admin&body=body&user=guest
```

- 이용자가 & 구분자를 포함해 user 파라미터를 추가했다.
    - 내부 API에서는 전달받은 값을 파싱할 때 앞에 존재하는 파라미터의 값을 가져와 사용하기 때문에 user의 값을 변조할 수 있다.
        - title&user=admin을 삽입했을 때에 실행 결과를 확인해보면 user가 “admin”으로 변조된 것을 확인할 수 있다.

```
{ "body": "body", "title": "title", "user": "admin" }
```