

# 7월Dreamhack CTF Writeup

## Dreamhack Season 6 Round #1

32기 안현진

이번 CTF에서 출제된 분야는 포너블, 리버싱 웹, 그리고 크립토(암호학)이다. 리버싱은 깊은 수준의 공부가 뒷받침되어야 접근할 수 있는 분야라서, 그리고 암호학은 한번도 공부해 본 적이 없어서 문제를 풀지 못했다. 아직 나는 CTF를 풀 수준이 아니기 때문에, 문제를 푸는 목적이 아닌, 'CTF문제가 어떻게 출제되는지 파악하기'목적으로 현재 공부하고 있는 웹 분야에 도전했다.

×

C

Simple Login

100 pts

deayzl, GORiyA, 군인김도현, wlswothmd, xavi 외 3명이 해결했습니다.

web

Description

드림이는 JSON Web Token (JWT)을 공부하기 위해 로그인 페이지를 구현해보았어요.  
웹 서비스를 보안을 수 있도록 취약점을 찾고 익스플로잇하여 플래그를 획득하세요!  
플래그 형식은 DH{...} 입니다.

Translate

문제 파일 다운로드

접속 정보

접속 정보 보기

접속 정보를 받으신 이후 서버 상태 및 환경에 따라 바로 연결이 되지 않을 수 있습니다. 이 경우 잠시 기다리신 후 연결해 주세요. 5분 이상 대기 후에도 연결이 되지 않는다면 VM 재요청을 통해 다시 신청할 수 있습니다.


FLAG를 입력하세요.

Flag(정답)을 입력해 주세요.

제출하기

웹 문제의 메인화면이다. 설명을 보니, 로그인 페이지의 취약점을 찾아 공격하면 플래그를 얻을 수 있는 것 같다. 일단 문제 파일을 다운받아보자.

문제 파일을 다운로드 받고, 압축을 풀면 먼저 하나의 YAML파일과 폴더가 나온다. 먼저 파일 내용을 살펴보았다. (YML파일은 사람이 읽기 쉽도록 설계된 데이터 직렬화 형식의 파일이다. JSON이나 XML과 비슷한 역할을 하지만, 더 간단하고 가독성이 뛰어나다)

 6c0ba3c5-ab6e-4660-af52-414039291e2a	2024-07-06 오후 4:44	압축(ZIP) 파일	15KB
 docker-compose	2024-07-06 오후 4:45	YML 파일	1KB
 deploy	2024-07-06 오후 4:45	파일 폴더	

```
<<docker-compose>>
version: "3.9"

services:
  app:
    build:
      context: deploy/
    networks:
      - internal
    ports:
      - "7000:7000"

networks:
  internal:
```

이 파일이 무슨 역할을 하는지 몰라, 파일의 이름인 'docker-compose'에 대해 검색해보았다. Docker Compose 파일은, 여러개의 Docker 컨테이너를 정의하고 관리하기 위한 도구인 Docker Compose에서 사용하는 설정 파일이다. 이 파일은 YAML 형식으로 작성되며, 애플리케이션의 여러 서비스를 정의하고 각 서비스의 설명을 명시한다. 그러니까, 이 파일은 드림이가 구현한 로그인 페이지의 서비스를 정의하고 각 서비스의 설명을 명시하는..안내서같은 역할을 하는건가?

해당 Docker Compose 파일을 이해하기 위해 기본적인 Docker Compose 파일의 구조를 조금만 살펴보고 넘어가자.

```
<<Docker Compose 파일의 구조-yaml 파일 형식>>
version: '3.9' # Docker Compose 파일의 버전(버전은 사용할 도커 컴포즈의 기능 결정)
services: # 서비스를 정의하는 블록. 각 서비스는 독립된 컨테이너로 실행된다.
  web: # 웹 서비스 정의
    image: nginx # 사용할 Docker 이미지
    ports: # 포트 매핑 설정
      - "80:80"
  db: # 데이터베이스 서비스 정의
    image: mysql # 사용할 Docker 이미지
    environment: # 환경 변수 설정
      MYSQL_ROOT_PASSWORD: example
networks: # 네트워크 정의
  default:
volumes: # 볼륨 정의
  db-data:
```

-> 이 예시에서 웹 서비스는 nginx 웹 서버를 실행하고, db서비스는 mysql 서비스를 실행

이제 CTF문제의 도커파일을 해석해보자.

```
<<docker-compose>>
version: "3.9" #docker compose의 파일 버전은 3.9이다.

services: #생성될 컨테이너들을 묶어놓은 단위
  app:
    build:
      context: deploy/ #context 설정은 도커 컴포즈 파일에서 이미지를 빌드할 때 사용되는 디렉토리를 지정한다. deploy/ 디렉토리를 빌드 항목으로 사용한다는 의미이다.
    networks:
      - internal #app 서비스가 internal 네트워크를 통해 다른 컨테이너와 통신한다. 사용자가 정의한 internal 네트워크를 app 서비스가 연결되도록 설정했다. 단일 네트워크이다.
    ports:
      - "7000:7000" #클라이언트의 7000번 포트 -> 서버의 7000번 포트 사용

networks:
  internal:
```

파일을 살펴보고, 파일의 역할을 알게 되었지만 아직 이것이 어떻게 사용되는지는 잘 모르겠다. 이제 폴더로 넘어가자.

폴더 이름이 deploy이다. 아까 도커 컴포즈 파일에서 본 적 있는 것 같다. context 부분에 정의되어있는 것이 deploy디렉토리였다. 앱을 구성하는 주요 내용이 여기에 있는걸까?

deploy 폴더를 열면 이렇게 2개의 파일과 1개의 폴더가 보인다. Dockerfile먼저 열어보자.

 app	2024-07-06 오후 4:45	파일 폴더	
 Dockerfile	2024-07-06 오후 4:45	파일	1KB
 flag	2024-07-06 오후 4:45	파일	1KB

```
<<Dockerfile>>
FROM node:21.5-slim@sha256:c88704924204ee6d4e9da02275a8fd27355502f7177e4175cd0d3a4375a9c9c8

ARG DEBIAN_FRONTEND=noninteractive
ENV NODE_ENV production
ENV PORT 7000
ENV USER app
```

```
# Install packages
RUN apt update && \ #RUN은 이런 설치, 빌드 작업에 사용된다
    apt install -y \
    tzdata \
    openssl
RUN ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime
RUN dpkg-reconfigure tzdata

# Add user
RUN adduser --disabled-password --gecos "" --uid 1001 $USER

# Set working directory
RUN mkdir /app
WORKDIR /app

# Add files
COPY --chown=root:root app/server.js /app/
COPY --chown=root:root app/views /app/views
COPY --chown=root:root flag /
COPY --chown=root:root app/package.json /app/
COPY --chown=root:root app/package-lock.json /app/

# Install dependencies
RUN npm install

# Newly generate keys
RUN openssl genrsa -out /priv.pem 2048
RUN openssl rsa -in /priv.pem -pubout -out /pub.crt

RUN chmod 644 /flag
RUN chmod 644 /priv.pem
RUN chmod 644 /pub.crt

USER $USER
EXPOSE $PORT
CMD while :; do sleep 0.5; node server.js; done
```

와 정말 길다. 그리고 이런 코드 형식도 처음본다. 파일 이름이 도커 파일이니, Dockerfile 형식을 찾아보면 풀이 단서를 얻을 수 있지 않을까? 아까와 마찬가지로 먼저 Dockerfile에 대해 간단히 알아보자.

Dockerfile은 Docker 이미지를 빌드하기 위한 명세서를 정의하는 파일이다. Dockerfile은 Docker 이미지를 구성하는 단계별 지침을 포함하고 있어 이를 통해 반복 가능한 방식으로 이미지를 생성할 수 있다고 한다. 아래는 도커파일의 기본 구조이다.

<<Dockerfile 기본 구조>>

# 기본 이미지 지정

FROM python:3.9

# 작업 디렉토리 설정. 이후 명령어들은 이 디렉토리 내에서 실행된다

WORKDIR /app

# 필요한 파일들을 컨테이너로 복사

COPY . /app

# 컨테이너 내부에서 명령어를 실행. 주로 패키지 설치나 빌드 작업에 사용된다.

RUN pip install --no-cache-dir -r requirements.txt

# 컨테이너 시작 시 실행할 명령어

CMD ["python", "app.py"]

- +) EXPOSE: 컨테이너가 수신할 포트를 지정한다. 이 포트는 외부에서 접근할 수 있도록 한다.
- +) ENV: 환경 변수를 설정한다.
- +) ENTRYPOINT: 컨테이너가 시작될 때 실행할 명령어를 지정한다. CMD와 유사하지만, ENTRYPOINT가 우선순위가 더 높다.
- +) VOLUME: 컨테이너와 호스트 간의 데이터 볼륨을 설정한다.
- +) ARG: 빌드 타임에 사용할 변수를 정의한다.

이제 다시 CTF 도커 파일로 넘어가자. 이제 뭔가 조금씩 보이기 시작한다. 위에서 살펴본 도커파일에서 나타난 명령어들의 순서가 조금씩 뒤바뀐 부분이 있긴 하지만, 전체적으로는 비슷하다. CTF문제 도커 파일의 #Install을 보니, 직접 도커 컴포즈를 설치하고 도커 파일을 실행해야 할 것 같다.

도커파일 실행 전, 살펴봐야 할 부분이 있다. 위의 코드에서 초록색 글씨 부분을 보자.

# Add files 부분

COPY --chown=root:root app/server.js /app/

COPY --chown=root:root app/views /app/views #에는 다른 것과 달리 디렉토리 형태

COPY --chown=root:root flag /

COPY --chown=root:root app/package.json /app/

COPY --chown=root:root app/package-lock.json /app/

이 부분을 보면 COPY 뒤에 app/ 디렉토리 내에 파일이 있는 형태의 코드가 보인다. app 디렉토리 안에 server.js 파일, views 디렉토리, package.json 파일, package-lock.json 파일이 보인다.

그런데 아까 다운받은 문제 파일을 살펴보면 app/ 디렉토리에 있어야 하는 파일과 디렉토리가 모두 존재한다.

views	2024-07-06 오후 4:45	파일 폴더	
package	2024-07-06 오후 4:45	JSON 파일	1KB
package-lock	2024-07-06 오후 4:45	JSON 파일	36KB
server.js	2024-07-06 오후 4:45	JSFile	4KB

앞에 chown은 파일이나 디렉토리의 소유권을 변경하는데 사용하는 명령어라고 배운 적 있다. root라고 써있는 것을 보니, root가 이 권한을 가지는 것 같다.

그리고 views 폴더 안에는 ESJ형식의 파일 두 개가 있다. EJS는 Embedded JavaScript의 약어로, 자바스크립트 코드와 HTML을 결합하여 동적 웹페이지를 생성하는 템플릿 엔진이다. EJS는 .ejs 확장자를 가지며, 서버측에서 렌더링되어 클라이언트에게 HTML으로 전송된다. 파일을 열어보면 html 형식으로 코드가 작성되어있다.

index.ejs	2024-07-06 오후 4:45	EJS 파일	3KB
login.ejs	2024-07-06 오후 4:45	EJS 파일	2KB

나는 칼리 리눅스에서 도커를 실행해보았다. 먼저 Dockerfile에 적혀있는 그대로, 칼리 리눅스 환경에서 app디렉토리 내에 파일과 디렉토리를 추가한다. 추가한 파일과 디렉토리에, 다운받은 문제 파일 내에 있는 해당 파일과 디렉토리 안에 적혀야할 코드들을 붙여넣기 해준다.

아래 사진과 같은 순서로 파일과 디렉토리를 생성했다.

```
(kali@kali)~[/docker]
$ ls
app Dockerfile
(kali@kali)~[/docker]
$ cd app
(kali@kali)~[/docker/app]
$ ls
package.json package-lock.json server.js views
(kali@kali)~[/docker/app]
$ cd views
(kali@kali)~[/docker/app/views]
$ ls
index.ejs login.ejs
```

views 디렉토리 내에 있는 파일을 작성하면서 nano라는 새로운 편집기를 사용해보았다. 기존

에는 vi편집기와 vim편집기만 사용해봤는데 사용 방법은 크게 다르지 않았다.

nano 편집기를 사용하여 html 파일을 저장하는 과정에서 계속 오류가 났다. 왜 오류가 나는지 확인해보니 편집기 사용 권한이 없어서 그렇다고 한다. 그래서 sudo nano [파일명] 형식으로 작성했고, 이렇게 작성하니 에러 메시지가 뜨지 않았다.

docker 디렉토리로 이동한 다음 #docker build -t my-node-app . 명령어를 통해 docker 디렉토리에서 Docker 이미지를 빌드했다. 그런데 마지막 부분에 에러 메시지가 뜬다.flag 파일을 찾을 수 없다고 한다.

```
(root@kali) [/home/kali/docker]
# docker build -t my-node-app .
[+] Building 5.4s (14/22)
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 1.04kB
-> [internal] load metadata for docker.io/library/node:21.5-slim@sha256:c88704924204ee6d4e9da
-> [internal] load .dockerignore
-> => transferring context: 2B
-> CANCELED [ 1/18] FROM docker.io/library/node:21.5-slim@sha256:c88704924204ee6d4e9da
-> => resolve docker.io/library/node:21.5-slim@sha256:c88704924204ee6d4e9da
-> => sha256:c88704924204ee6d4e9da02275a8fd27355b2f7177e41 1.21kB / 1.21kB
-> => sha256:ed48b7d3cf246498c77c9c9d8a6c70115a35bcb9054154 1.37kB / 1.37kB
-> => sha256:27779f858751f2a52b385016be970045d0d82d9727363e 7.62kB / 7.62kB
-> [internal] load build context
-> => transferring context: 43.99kB
-> CACHED [ 2/18] RUN apt update 66 apt install -y tzdata opens
-> CACHED [ 3/18] RUN ln -sf /usr/share/zoneinfo/Asia/Seoul/etc/localtime
-> CACHED [ 4/18] RUN dpkg-reconfigure tzdata
-> CACHED [ 5/18] RUN adduser --disabled-password --gecos "" --uid 1001 app
-> CACHED [ 6/18] RUN mkdir /app
-> CACHED [ 7/18] WORKDIR /app
-> CACHED [ 8/18] COPY --chown=root:root app/server.js /app/
-> CACHED [ 9/18] COPY --chown=root:root app/views /app/views
-> ERROR [10/18] COPY --chown=root:root flag /
> [10/18] COPY --chown=root:root flag /:
Dockerfile:26
24 | COPY --chown=root:root app/server.js /app/
25 | COPY --chown=root:root app/views /app/views
26 | >>> COPY --chown=root:root flag /
27 | COPY --chown=root:root app/package.json /app/
28 | COPY --chown=root:root app/package-lock.json /app/
ERROR: failed to solve: failed to compute cache key: failed to calculate checksum of ref ee305018-e8a4-4f67-ac25-2d0d4798152e::ksa7w4bma42b6mttlbd9ytaje: "/flag": not found
```

# Add files 부분

COPY --chown=root:root app/server.js /app/

COPY --chown=root:root app/views /app/views #에는 다른 것과 달리 디렉토리 형태

COPY --chown=root:root flag / <-여기에는 flag 파일이 존재한다.

COPY --chown=root:root app/package.json /app/

COPY --chown=root:root app/package-lock.json /app/

그래서 Dockerfile의 코드를 찾아봤는데, 여기에는 flag가 적혀있다. 근데 이 flag는 app 디렉토리에 있는것 같지는 않다. 뭐지? 일단 Dockerfile 내에 적혀있는 저 문장을 지워보기로 했다.

```
# Add files
COPY --chown=root:root app/server.js /app/
COPY --chown=root:root app/views /app/views
COPY --chown=root:root app/package.json /app/
COPY --chown=root:root app/package-lock.json /app/
```

nano 편집기를 사용해서 flag부분을 지웠다. 그리고 다시 #docker build -t my-node-app . 명령어를 통해 docker 디렉토리에서 Docker 이미지를 빌드했다.

```
=> [13/17] RUN openssl genrsa -out /priv.pem 2048 1.2s
=> [14/17] RUN openssl rsa -in /priv.pem -pubout -out /pub.crt 0.7s
=> ERROR [15/17] RUN chmod 644 /flag 0.6s

> [15/17] RUN chmod 644 /flag:
0.488 chmod: cannot access '/flag': No such file or directory

Dockerfile:36
34 |     RUN openssl rsa -in /priv.pem -pubout -out /pub.crt
35 |
36 | >>> RUN chmod 644 /flag
37 |     RUN chmod 644 /priv.pem
38 |     RUN chmod 644 /pub.crt

ERROR: failed to solve: process "/bin/sh -c chmod 644 /flag" did not complete successfully: exit code: 1
```

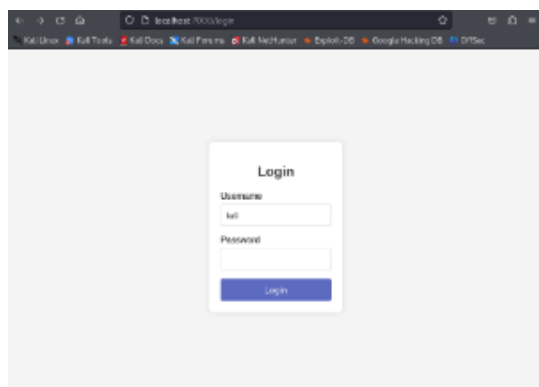
아까보다 실행되는 시간이 길어 성공하는 줄 알았는데, 에러가 또 뜬다. 또 flag관련 에러다. 분명 flag는 지웠는데? 지워서 생기는 오류인가?

그래서 Dockerfile에 flag 문장을 다시 추가하고, docker디렉토리 아래에 flag 파일을 만든 뒤, 다시 #docker build -t my-node-app . 명령어를 사용했다.

```
File Actions Edit View Help
GNU nano 8.0
DH{**flag**}
```

```
# Add files
COPY --chown=root:root app/server.js /app/
COPY --chown=root:root app/views /app/views
COPY --chown=root:root app/package.json /app/
COPY --chown=root:root app/package-lock.json /app/
COPY --chown=root:root flag /
```

이번에는 에러 없이 성공적으로 코드가 작동한다! 그러면 이제 #docker run -p 7000:7000 my-node-app 명령어를 사용하여 앱을 실행해보자. 브라우저를 열고, http://localhost:7000 주소로 이동하면 아래와 같은 화면이 뜬다.





이제야 CTF 설명에 있었던 로그인 사이트에 들어왔다... 내가 문제를 잘못 이해해서 이렇게 돌아온건가? 근데 다운받은 파일에 브라우저로 바로 연결되는 파일은 없었는데...

어쨌든 이제 해당 로그인 페이지의 취약점만 찾으면 된다. Username과 Password에 kail를 쳤더니 리눅스 셸에 이런 메시지가 떴다.

```
(root@kali)-[/home/kali/docker]
# docker run -p 7000:7000 my-node-app
^C
TypeError: Cannot read properties of undefined (reading 'password')
    at /app/server.js:116:86
    at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
    at next (/app/node_modules/express/lib/router/route.js:144:13)
    at requireNoAuthentication (/app/server.js:93:5)
    at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
    at next (/app/node_modules/express/lib/router/route.js:144:13)
    at Route.dispatch (/app/node_modules/express/lib/router/route.js:114:3)
    at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
    at /app/node_modules/express/lib/router/index.js:284:15
    at Function.process_params (/app/node_modules/express/lib/router/index.js:346:12)
    2)
```

TypeError라고 한다. 비밀번호가 틀렸다는 의미인 것 같다. 일단 비밀번호에 대한 정보가 있었는지 찾아보자.

다운받은 문제 파일의 server 파일에 비밀번호와 관련된 내용이 있는데, 이것 사용하면 될까? 아니면 로그인 화면 url 주소가 http 프로토콜을 사용하던데.. 이게 취약점인가?. 로그인 서버에서 취약점을 발견하고, 이 부분을 익스플로잇 해서 flag를 얻어야하는데, 이것 어떻게 해야 할지 생각이 나지 않는다.