Expression Language Injection

NVD Categorization

> CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection'): The software constructs all or part of an expression language (EL) statement in a Java Server Page (JSP) using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended EL statement before it is executed.

Description

Expression Language (EL) Injection happens when attacker controlled data enters an EL interpreter.

With EL implementations prior to 2.2, attacker can recover sensitive server side information available through implicit objects. This includes model objects, beans, session scope, application scope, etc. The EL 2.2 spec allows method invocation, which permits an attacker to execute arbitrary code within context of the application. This can manipulate application functionality, expose sensitive data, and branch out into system code access– posing a risk of server compromise.

A specific pattern exists in certain version of the Spring Framework, where Spring JSP tags will double resolve EL. In versions prior to 3.0.6, it is not possible to disable this functionality, and the pattern must be avoided.

Risk Factors

The likelihood of this issue is **Medium**, for the following reasons:

- Certain attack scenarios are not overly sophisticated, although require some skill.
- Automated tools may begin to pick up on the pattern, increasing the likelihood of discovery.
- Attackers are highly motivated to discover code execution vulnerabilities.

The overall impact of this issue is **High**, for the following reasons:

- An attacker could modify and invoke functionality on the application server.
- Unauthorized access to data and functionality, as well as account hijacking and remote code execution.
- Confidentiality, and Integrity concerns from a successful attack.

Examples

Spring Message Tag

The Spring Message tag will double resolve Expression Language.

A common pattern of passing URL parameters to the message tag is:

Controller.java

```
@RequestMapping(value="/")
String index() {
  if ( hasErrors() ) {
    return "redirect:/error?msg=error.generic";
  } else {
    return "index";`
  }
}
```

error.jsp

```
<spring:message code="${param.msg}" />
```

A URL request to the above code of the form:

```
?msg=${param.test}&test=INJECTION
```

Will result in the string literal “INJECTION” being passed to the message tag. The application should respond with an exception like:

```
No message found under code 'INJECTION' for locale 'en_US'
```

Accordingly, the attacker could submit methods within the EL like:

```
?msg=${pageContext.request.getSession().setAttribute("admin",true)}
```

If the container provided EL interpreter does not support static class methods (`java.lang.Runtime.getCurrentRuntime().exec()`), an attacker can use a URLClassLoader to load remote code.

Spring Eval Tag

Spring Framework provides a JSP tag that interprets Spring Expression Language (SpEL).

The following code would be vulnerable:

```
<spring:eval expression="${param.vulnerable}" />
```

A URL of the following form would provide system code access:

```
?vulnerable=T(java.lang.Runtime).getRuntime().exec(“cmd.exe”)
```

Related Attacks

Related Vulnerabilities

- Injection problem

Related Controls

Avoid putting user data into an expression interpreter if possible. Otherwise, validate and/or encode the data to ensure it is not evaluated as expression language.

In the case of Spring Framework, disable the double resolution functionality in versions 3.0.6 and above by placing the following configuration in the application web.xml.

```
<context-param>
  <description>Spring Expression Language Support</description>
  <param-name>springJspExpressionSupport</param-name>
  <param-value>false</param-value>
</context-param>
```

References

- CWE 917.
- Spring Framework: CVE-2011-2730
- EL Injection: Information Disclosure
- EL Injection: Remote Code Execution
- JSR245: EL 2.2 Spec

Edit on GitHub

Spotlight: Backslash

Backslash is the first Cloud-Native Application Security solution for enterprise AppSec teams to provide unified security and business context to cloud-native code risk, coupled with automated threat modeling, code risk prioritization, and simplified remediation across applications and teams. With Backslash, AppSec teams can see and easily act upon the critical toxic code flows in their cloud-native applications; quickly prioritize code risks based on the relevant cloud context; and significantly cut MTTR (mean time to recovery) by enabling developers with the evidence they need to take ownership of the process.

Corporate Supporters

Become a corporate supporter

PRIVACY SITEMAP CONTACT