**Dynamic Code Evaluation: Code Injection Development Mitigation SOP**

JavaScript has functions that will allow code to be executed from an imported source. The benefit of these functions gives developers the capacity to perform dynamic instructions based on input received. Code Injection is the byproduct of allowing dynamic interpretation of source instruction from an untrusted source that has malicious intentions. Typically, the developer assumes the code from a user will be supplied directly from the user to perform an operation without any validation.

**Defense Against Dynamic Code Evaluation: Code Injection**

The best way to avoid being vulnerable is to avoid using dynamic code all together. Having a method like eval() in your code can be a major threat if discovered by hackers. If dynamic code execution must be used, the user input needs to be properly validated before execution.

**Example**

j = eval('(' + text + ')');

**Explanation**

The example above allows the immediate dynamic instructions from user input to be performed.

All data pulled from sources trusted or untrusted need to be validated before applied. Apply the c:out around the code. It is good practice to not log sensitive information, such as PII.

**Example**

In this classic code injection example, the application implements a basic calculator that allows the user to specify commands for execution.  
  
...  
ScriptEngineManager scriptEngineManager = new ScriptEngineManager();  
ScriptEngine scriptEngine = scriptEngineManager.getEngineByExtension("js");  
userOps = request.getParameter("operation");  
Object result = scriptEngine.eval(userOps);  
...  
  
**Explanation**  
The program behaves correctly when the operation parameter is a benign value, such as "8 + 7 \* 2", in which case the result variable is assigned a value of 22. However, if an attacker specifies languages operations that are both valid and malicious, those operations would be executed with the full privilege of the parent process. Such attacks are even more dangerous when the underlying language provides access to system resources or allows execution of system commands. For example, JavaScript allows invocation of Java objects; if an attacker were to specify " java.lang.Runtime.getRuntime().exec("shutdown -h now")" as the value of operation, a shutdown command would be executed on the host system.

**Recommendation**

This will prevent any script from executing.

Remember to refer to the tag library whenever it is used, even if the parent page has already called the library. Since they can be compiled separately, this is a good practice to prevent cross-site scripting.

**References**

1. [OWASP – Injection Flaws](https://www.owasp.org/index.php/Injection_Flaws)
2. [HP - Dynamic Code Evaluation: Code Injection](http://www.hpenterprisesecurity.com/vulncat/en/vulncat/javascript/dynamic_code_evaluation_code_injection.html)