Unsafe Reflection Defect Best Practices

When an application uses user-provided values for model reflection, this is called unsafe reflection. If an attacker can supply values that the application then uses to determine which class to instantiate or which method to invoke, the potential exists for the attacker to create control flow paths through the application that were not intended by the application developers. This attack vector may allow the attacker to bypass authentication or access control checks or otherwise cause the application to behave in an unexpected manner. Even the ability to control the arguments passed to a given method or constructor may give a wily attacker the edge necessary to mount a successful attack.

This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application's classpath or add new entries to the application's classpath. Under either of these conditions, the attacker can use reflection to introduce new, presumably malicious, behavior into the application.

# Preventing Unsafe Reflection

The best way to prevent unsafe reflection is with a level of indirection: create a list of legitimate names that users are allowed to specify, and only allow users to select from the list. With this approach, input provided by users is never used directly to specify a name that is passed to the reflection API.

Reflection can also be used to create a custom data-driven architecture, whereby a configuration file determines the types and combinations of objects that are used by the application. This style of programming introduces the following security concerns:

- The configuration file that controls the program is an essential part of the program's source code and must be protected and reviewed accordingly.

- Because the configuration file is unique to the application, unique work must be performed to evaluate the security of the design.

- Because the semantics of the application are now governed by a configuration file with a custom format, custom rules are required for obtaining optimal static analysis results.

For these reasons, avoid using this style of design unless your team can devote a large amount of effort to security evaluation.

Beware of attempts to validate user input before using it as part of a call to a reflection method. Because the application is likely to evolve faster than the operating system, the file system, or other system components, the work required to validate user input must evolve much more rapidly than the input validation required for sending user data to other system components. Even if the validation is currently correct, it might not be correct in the future.

# Example

String ctl = request.getParameter("ctl");

Class cmdClass = Class.forName(ctl + "Command");

Worker ao = (Worker) cmdClass.newInstance();

ao.doAction(request);

This code allows an attacker to instantiate any object which implements the Worker interface.

# References

<http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/unsafe_reflection.html>

<https://www.owasp.org/index.php/Unsafe_Reflection>