**Build Misconfiguration: External Maven Dependency Repository.**

Using the Maven build script relies on external sources, which could allow an attacker to insert malicious code into the final product or to take control of the build machine. Several tools exist within the Java development world to aid in dependency management: both Apache Ant and Apache Maven build systems include functionality specifically designed to help manage dependencies and Apache Ivy is developed explicitly as a dependency manager. Although there are differences in their behavior, these tools share the common functionality that they automatically download external dependencies specified in the build process at build time.

**Classic Example**

<dependencies>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.1</version>

</dependency>

<dependency>

<groupId>javax.jms</groupId>

<artifactId>jms</artifactId>

<version>1.1</version>

</dependency>

...

</dependencies>

Developers just store dependency information in the build file, which means that each developer and build engineer has a consistent way to obtain dependencies, compile the code, and deploy without the dependency management hassles involved in manual dependency management. The following examples illustrate how Ivy, Ant and Maven can be used to manage external dependencies as part of a build process.

Under Maven, instead of listing explicit URLs from which to retrieve the dependencies, developers specify the dependency names and versions and Maven relies on its underlying configuration to identify the server(s) from which to retrieve the dependencies. For commonly used components this saves the developer from having to research dependency locations.

**Example**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>company.system</groupId>

<artifactId>system-nwq</artifactId>

<version>15.0-SNAPSHOT</version>

<relativePath>../system-nwq/pom.xml</relativePath>

</parent>

<artifactId>system-nwq-dcp</artifactId>

<packaging>pom</packaging>

<name>System Name</name>

<properties>

<jacoco.coverage.line.threshold>0.930</jacoco.coverage.line.threshold>

<jacoco.coverage.branch.threshold>0.910</jacoco.coverage.branch.threshold>

</properties>

<modules>

<module>../system-nwq-dcp-api</module>

<module>../system-nwq-dcp-impl</module>

<module>../system-nwq-dcp-app</module>

</modules>

</project>

**Defense Against *Specific Vulnerability***

The simplest solution is to refrain from adopting automated dependency management systems altogether. Managing dependencies manually eliminates the potential for unexpected behavior caused by the build system. Obviously, the attacker could still mount one of the attacks described above to coincide with the manual retrieval of a dependency but limiting the frequency with which the dependency must be retrieved significantly reduces the window of opportunity for an attacker. A system based on manual dependency management is often more difficult to use and maintain, and may be unacceptable in some software development environments.

The second solution is a hybrid of the traditional manual dependency management approach and the fully automated solution that is popular today. The biggest advantage of the manual build process is the decreased window of attack, which can be achieved in a semi-automated system by replicating external dependency servers internally. This solution reduces the attack opportunities and allows the organization to leverage existing internal network security infrastructure.

**Defense Example *(fixing Classic example above)***

The following Maven pom.xml demonstrates the use of an explicit internal IP address (the entries can also be used in settings.xml):

<project>

...

<repositories>

<repository>

<releases>

<enabled>true</enabled>

<updatePolicy>always</updatePolicy>

<checksumPolicy>warn</checksumPolicy>

</releases>

<snapshots>

<enabled>true</enabled>

<updatePolicy>never</updatePolicy>

<checksumPolicy>fail</checksumPolicy>

</snapshots>

<id>central</id>

<name>Internal Repository</name>

<url>http://172.16.1.13/maven2</url>

<layout>default</layout>

</repository>

</repositories>

<pluginRepositories>

...

</pluginRepositories>

...

</project>

**References**

1. Standards Mapping - NIST Special Publication 800-53 Revision 4, SC-18 Mobile Code (P2)
2. Standards Mapping - OWASP Mobile Top 10 Risks 2014, M7 Client Side Injection
3. Standards Mapping - OWASP Top 10 2017, A6 Security Misconfiguration
4. Standards Mapping - Security Technical Implementation Guide Version 4.1, APSC-DV-001390 CAT II, APSC-DV-001430 CAT II, APSC-DV-001440 CAT II, APSC-DV-003300 CAT II