Managing Java Security in the Enterprise

Introduction

Since the introduction of Java, enterprise organizations have relied on it as a powerful development platform on which to build internal line of business applications. Java's 'write once, run anywhere' promise theoretically offered a technological hedge against the challenges of building cross platform applications, with their associated complex matrix of dependencies. This promise was realized in the early days of Java for a large number of standalone Java applets running directly on client operating systems.

Over time, developers shifted their focus to web-based applications built on the Java Runtime Environment (JRE) and run in a browser. These web-based Java applications shared many similarities with standalone Java applets, but had to be packaged and delivered differently for the web. This difference resulted in a variety of compatibility and security issues caused by JRE incompatibilities from version to version and by unlimited threat vectors on the web.

Despite these issues, Java application development remains incredibly common within medium and large organizations. As a result, broad Java deployments to all end user PCs have become the norm – Java is nearly always part of an organization's 'standard' desktop image. In addition to enterprise adoption, consumer use of Java is huge. Oracle claims more than 3 billion devices running Java globally.

That large target population is very attractive to hackers, and Java has a long history of security vulnerabilities. Since 2010 there have been at least 376 reported security vulnerabilities in the various revisions of Java 7 and 8. In 2014 alone, there were 133 reported vulnerabilities, resulting in roughly 25 updates each for Java 7 and Java 8. Most of those updates were delivered in quarterly update packages, but several critical vulnerabilities required updating every few weeks to remain fully secure.

In response to ongoing Java security threats, most Information Security organizations have pressed for regular internal updates to Java versions on end user PCs. This drives two primary challenges for most organizations: change management processes and application compatibility. In reality the two are often linked, but they are distinct issues and should be analyzed separately. This paper will overview each challenge individually and provide detailed guidance for managing Java security in the enterprise using a combination of sound security practices and Browsium's browser management platform.

Challenge I: Java updates and change management

Most organizations have implemented a change management process that utilizes a set of steps to ensure documentation and validation of any changes made to internal systems. This process is critical to ensuring system stability, avoiding downtime, and minimizing operational risk. At the same time, these steps take time to complete and that creates a window of security exposure. Many companies require at least 21 business days to complete all steps in their change management process. From a Java security standpoint, that can be an excessive and unacceptable amount of time for the organization to be at risk of attack.

To address this security exposure, organizations are faced with an operational challenge and must choose between two unpalatable options: either bypass the change management process or reduce the security mandate. Neither option is a viable choice for most organizations. The risk of deploying Java updates without proper validation and testing could result in severe impacts to the business if systems fail or operate incorrectly. Accepting security exposure during the time required to complete the change management process could result in security breaches and potential financial or reputational liability impacts.

Challenge II: Java updates and application compatibility

The Java platform is designed to avoid dependency issues by offloading many environmental components and interactions to the JRE, rather than being coded directly by the developer. However, updates to the JRE are common, often resulting in feature behavior changes or even feature deprecation. These feature changes are typically caused by a modification to the JRE in response to a security vulnerability. Applications that make use of one of these impacted features will no longer work in the new version of Java, and the organization must decide if they will remain on the previous (working) version of the JRE or upgrade to the more secure JRE and rewrite the applications.

This choice leaves the organization with another oppositional challenge – continue running an older JRE and accept the security risk, or upgrade the JRE and accept the development costs and impacts of rewriting the affected applications. Making the choice to rewrite applications comes with additional complications as development can't be completed, tested, and accepted into deployment instantly. The time required to perform these tasks puts the organization at risk.

Fulfilling the security mandate by rewriting applications also introduces direct and indirect costs. Some department, most often the business unit to which the application belongs, must accept the costs for developers to rewrite the application. Most business units don't have budget for this type of expense, especially when the development work would simply redesign the application and not provide any new functionality. Every development organization has limited resources, so the organization will

struggle to deliver new projects when resources have been reallocated from new project work to these security-mandated maintenance releases.

Looking at the JRE version update cadence for Java 7 from 2011 to early 2015 (a total of 76 updates), it is reasonable to assume that a new version of the JRE will be released before the organization has completed development, testing, and acceptance of the quickly outdated JRE they were planning to deploy. This 'Java treadmill effect' could easily stagnate an organization's ability to move the business forward and deploy new technologies that enable the organization to grow.

The Answer: Secure Java Management

For most organizations, the answer to competing compatibility and security objectives and challenges is simple – find a solution that enables the organization to keep running the older, known-compatible JRE versions in a secure and isolated environment. This approach will resolve business concerns around compatibility, avoid extra expense and the 'update treadmill', and meet the security mandate to ensure older, insecure JRE versions are not exposed to external threats. Unfortunately, while this works to secure standalone Java applets, it will not work to secure web-based Java applets. An understanding of Java's history clarifies why this is the case.

Standalone Java applets

A key capability of the Java platform – the side by side installation of various JRE versions – helps to enable running older versions of Java in a secure and isolated environment. Oracle (then Sun Microsystems) originally designed Java to allow any vendor or in-house developer to build Java-based software solutions, then redistribute customized JRE configurations with their software to ensure the proper operation and settings needed for their applications. By design, the JRE is loaded from the application folder prior to looking for a general-purpose, system-installed version of Java. Therefore, the proper JRE is always paired with the application. This solution works perfectly for standalone Java applets that are hosted by the operating system.

Securing web-based Java applets

The strategy to secure and isolate standalone Java applets fails to meet the needs of web-based Java applets. To understand why, it is important to know how web-based Java applets are loaded by a web browser. When a webpage contains a Java applet tag, which can be specified as either CODEBASE or APPLET, the browser is triggered to load the JRE add-on. By design, the browser will load the latest installed version of the JRE add-on by default. This ensures the JRE that is loaded is the most current and secure version available on that system, but it causes issues for Java applications that require a

specific JRE version to function properly. Without hard-coding Java version strings (a bad choice as we will see shortly), the web application is subject to failure when the JRE version is updated. Even for applications that include hard-coded versioning, Oracle has recently taken security efforts in newer JRE versions to prevent older JRE versions from loading.

This approach was well intentioned, but an organization can't rely on using hard-coded Java versioning as it leaves an extensive security hole open to attack. By permitting trusted internal or external websites to specify and load older, known-insecure versions of Java, the organization also enables untrusted, unknown external websites (e.g., public Internet sites) to specify an older JRE version and launch attacks against it. In addition, because Oracle has continued to block loading of out-of-date JRE versions, forcing users to run the most current version, they have taken a more aggressive warning and end user notification approach to block loading older JRE versions.

All of this leaves many organizations with an identified Java security management approach, but a lack of tools to manage Java in the way needed to meet business and security requirements. Because Java's design and architecture prevent addressing security and management concerns 'out of the box', organizations must look for third party solutions.

The Solution: Browser management with Browsium Ion

Browsium Ion was built to address all of these Java security issues via its browser management platform. The central design principle of Ion is to provide the web application environment needed by opening a browser instance with the specific configuration required for that application. This approach enables Ion to open a process-isolated instance of Internet Explorer, complete with the required JRE version, so line of business application will function as needed. This client-side approach requires no changes to the application, and legacy JRE versions are prohibited from being exposed to unapproved websites. In addition, Ion eliminates the end user confusion caused by JRE warning messages typically required to bypass legacy version blocking.

Opt-in rules model

The core of the Ion solution is an 'opt-in' rules model, whereby the defined configurations are loaded only when a specified criteria is matched. The Ion rules model examines network requests made by users and responds to navigation events triggered by their actions. When a condition, or set of conditions, are matched, the Ion Broker launches an Internet Explorer process with the specified configuration. By the nature of the application instantiation process, Ion ensures the Internet Explorer and associated JRE instances are process-isolated from other Internet Explorer processes, preventing any cross process communication attacks. The Ion rules engine further prevents attack by ensuring that Internet Explorer instances can only be launched or accessed by the Ion Broker process invoking them, eliminating the ability for unauthorized websites to invoke any legacy components.

Facilitates change management

Ion resolves the security/process conflict inherent in change management by enabling an organization to make business decisions rather than technology decisions. Ion offers enterprise IT the ability to determine the risk associated with a Java update and make the choice in how to proceed. Without Ion in place, an organization would need to make a choice between updating and possibly breaking applications, or staying on the legacy version and assuming the security risk until development and testing are done. Ion changes the equation, enabling the organization to specify a specific JRE version for a given web application (or set of applications) and disable Java for unknown or unauthorized websites. Alternatively, the organization can standardize on a known-compatible version of Java (that has already been tested against all internal LOB applications) and always update to the most current, secure version as soon as it is released by Oracle for use on Internet sites. Both approaches ensures external threats are mitigated, while internal business-critical tasks remain unaffected. The change management process can continue normally and IT can take the time needed to ensure systems are not impacted, all while remaining fully secure.

Addresses multi-version JRE add-on loading

Ion addresses the multi-version JRE add-on loading issue by taking advantage of the broker instantiation process architecture. Where the browser is normally only able to load the most current JRE version installed, Ion has the ability to bypass that behavior and specify which JRE version is loaded into the specific Internet Explorer instance. Ion does this by securely hosting and replacing the required JRE components at runtime. Using this model, Ion ensures each web application has access to the required JRE version, regardless of the default version installed on the system. At the same time, this approach enables an organization to have numerous legacy JRE versions securely installed on end user PCs, without worrying about insecure versions or their components being exposed to unauthorized websites. Working in conjunction with the JRE legacy version blocking solution, non-Ion managed websites always load the default system JRE version, which is configured to prevent the loading of any legacy components if the remote website attempts to force that behavior. This defensive, in-depth strategy helps protect systems from threats at every attack angle.

Easy to manage Java security

Lastly, Ion makes it easy to manage Java security by integrating and encompassing Internet Explorer and Java functionality. Taking this approach helps organizations reduce the amount of customization or duplication of configurations needed in deployments. Since Ion settings are a superset of Internet Explorer and Java settings, Ion-managed instances reflect all of the same organizational default settings and only need to be configured for exceptions or overrides that may be needed beyond those defaults. In addition, this design offers an enhanced defense-in-depth approach to Java security by

incorporating all of the enhancements Oracle makes in the JRE, instead of recreating them and potentially introducing new vulnerabilities.

Summary: Browsium Ion delivers web-based Java security and compatibility

Traditional approaches to security management have left enterprise IT facing the dilemma of choosing between security and compatibility. Browsium Ion eliminates the need to make that choice – organizations can now have both. Updates can be deployed quickly to keep IT infrastructure secure while retaining the required legacy versions of Java for compatibility. Enterprises can upgrade business applications when the business requires it, not when Java version changes force it to happen.

To learn more about Browsium Ion and its Java management and web application remediation capabilities, visit www.browsium.com/ion/ where you can download the Browsium Ion Evaluation Kit.

Browsium, Inc.
8201 164th Ave. NE, Suite 200
Redmond, WA 98052
+1.425.285.4424
sales@browsium.com