Table of Contents

Executive Summary ............................................. 3
Context and Background ............................... 4
Key Findings ..................................................... 6
Additional Findings ....................................... 12
Conclusion ...................................................... 13
Recommendations .......................................... 14
Methodology .................................................. 15
Sources ......................................................... 17
Executive Summary

Java’s vulnerabilities and prevalence combine to make it perhaps the single most important security problem facing today’s enterprises.

Java was originally released with the slogan “write once, run anywhere,” which was intended to underscore its cross-platform capabilities. Over time, Java has become ubiquitous on endpoints, so “run anywhere” can be interpreted as referring to its ubiquity. Even as fewer websites and Web applications require Java in order to operate properly, the technology is pervasive on virtually every end-user system. For a variety of reasons, Java also has become a platform that is highly vulnerable to attack.

Java’s ubiquity and vulnerabilities have made it the technology most frequently exploited by cyber attackers. So it is timely to closely explore the breadth and state of its deployment among enterprises. Bit9 undertook an examination of these questions, leveraging endpoint data across many organizations. The results are surprising and concerning:

- Java has become the most targeted endpoint technology.
- Most endpoints have multiple versions of Java installed, in part because the Java installation and update process often does not remove old versions.
- Attackers often target old, vulnerable versions installed on the endpoint.
- Fewer than 1 percent of enterprises run the latest version of Java.
Context and Background

Java has been a trending security concern for several years. In the past year, however, there has been a significant rise in reported vulnerabilities, security-related patches, and Java-related attacks. In 2012, Java surpassed Adobe Reader as the most exploited endpoint software in real-world attacks\(^1\), and the first half of 2013 has seen several major exploits and patches to Java 7.

Use of Java in the Enterprise

Many in the security community urge the widespread and complete removal of Java from all endpoints. This option is often difficult to implement in practice, however, because tools for managing Java in the enterprise are lacking, and it is often difficult for organizations to fully assess the impact of removing Java in their environment.

We hope that by shedding light on the reasons Java is so widely targeted, enterprises will have a deeper understanding of the issues involved and be better equipped to make decisions and take actions to remediate this important threat vector.

It is important to note that while Java has many vulnerabilities, their exposure is primarily a problem when Java is used as a client-side Web technology. Many products contain their own embedded versions of Java, and these instances are generally not made available to the browser, and so do not expose the same threat surface as a typical standalone Java installation. Internally developed non-Web applications also can use Java outside the browser. However, since Java poses such a significant risk when used as a client-side Web technology, caution must be exercised to ensure that Java instances that are made available for such applications are not also made available to the browser.

Java-related Attacks

There are many examples of attacks involving Java that a simple Web search will show. The following are some attacks that have unique and interesting aspects.

In early 2012, Trojan malware was discovered infecting Macs and creating a botnet of Mac-based endpoints. This malware was notable for being the largest scale threat to the Mac platform to that date\(^2\). The relative ease of creating malware in Java, and the ability to construct it with only a
modest amount of platform-specific code, may have aided the attackers.

Similarly, a cross-platform Trojan called "Minecraft Hack Kit," a password stealer, affected both Mac and Windows users. Another cross-platform Trojan, Mal/JavaJar-B, even attacked Linux systems as well as Mac and Windows.

Another example of threats related to Java in the wild was identified in May 2013. This attack targeted a 0day vulnerability in Internet Explorer. However, the first action taken by the malicious code, according to AlienVault Labs' analysis, was to enumerate all of the Java versions on the affected endpoint. Quite likely this reconnaissance step was intended to ensure the ability to compromise the host again in the future if, for example, the 0day vulnerability was patched via the Windows update mechanism. It also is interesting to note that the attackers apparently used code directly taken from the Java Deployment Toolkit as you can see by comparing this screenshot with Oracle's code [http://www.java.com/js/deployJava.txt].

Use of Java in Exploit Kits

Another piece of evidence showing Java's favored status among attackers is how frequently it is integrated into various exploit kits. These kits represent the evolution of publicly available scripted attacks into relatively mature, supported products that often are sold on the underground market. Exploit kits are typically used to implement browser-based attacks by deploying them on compromised or attacker-staged Web servers, and include a variety of exploits for use against client systems. Because vulnerable versions of Java are so pervasive and severe, the inclusion of Java exploits in these kits provides a high probability of successful compromise.
Key Findings

Most enterprises have multiple versions of Java

*Nearly half of all endpoints have more than two versions of Java*

Installing a new version of Java will not always remove older versions of the software. The Java updater, however, will attempt to remove the latest version in favor of the newer version. Using the installation process instead of the updating process allows for redundant instances of Java on a given endpoint. For example, running the Java update process when version 6 Update 13 is installed will cause the update process to attempt to remove version 6 Update 13 and install the latest version (i.e., 7 Update 25), but it will not remove version 5 update 22 if that version also had been installed previously.

Our analysis found that each endpoint had an average of 1.6 versions of Java installed. Considering that many deployments comprised mostly of server, point-of-sale, or other fixed-function endpoints never have Java installed, the average number of versions is quite likely much higher when looking strictly at typical end-user desktop environments. Approximately 42 percent of endpoints have more than two versions of Java installed at the same time. And approximately 20 percent of endpoints observed had more than three versions of Java on each system.

*The average organization has more than 50 distinct versions of Java*

Five percent of organizations analyzed had 100 or more distinct Java versions installed in their environment. Observed organizations on average hosted 51 distinct Java versions. This is likely due in large part to the fact that Java’s installers do not uninstall all older versions. Only Java 7 version installers and some versions of Java 6 installers remove minor versions within that major version (version 6 installers do not uninstall version 5 installations, for example). It seems likely that older endpoints carry more and older versions of Java. More redundant legacy versions...
of Java result in greater security risk across each organization. Typically, organizations that have fewer total versions of Java within their environment are organizations with more fixed-function devices, which usually do not have any version of Java installed.

The most prevalent versions of Java are highly vulnerable

**Java 6 is the most vulnerable major version of Java**

In determining the most vulnerable versions of Java, we summed the Common Vulnerability Scoring System (CVSS) scores associated with each version. Scores range from one to 10. To determine severity, we filtered by scores greater than seven, though the results did not vary substantially from those obtained without filtering. The chart below shows the most vulnerable Java versions—the top 20 all being version 6.

It is no surprise that Java 1.6.0 (version 6) is the most vulnerable version since it is very prevalent (present on 82 percent of endpoints) and attackers and researchers alike clearly have incentive to find flaws in the most prevalent versions of software. Because of the low adoption rate of newer versions and the difficulty in remediating older resident versions, attackers can target the most vulnerable versions of Java available—even when newer versions of Java are installed.
Java 6 Update 20 has 96 “perfect 10” vulnerabilities

The single most widely seen Java version, 6 update 20, has a total of 215 vulnerabilities, 96 of which score 10 out of 10 in the CVSS scale. This version of Java appears in a little more than 9 percent of the sampled endpoints. There are virtually no modern versions of Java without any known severe vulnerabilities, and vulnerabilities found in one version of Java often exist in a large number of older versions as well. Only a minority of Java versions that were found installed on endpoints do not have a large number of severe vulnerabilities, and these are restricted to the very old and unpopular 1.4- and 1.3-based versions.
More than 90 percent of organizations are running a version of Java at least 5-years old

Ninety-three percent of organizations are running a version of Java at least five-years old and 51 percent have a version that is between 5 and 10 years old. Only 7 percent of organizations do not have Java versions 5 years or older installed in their environment. The fact that a majority of observed environments apparently use significantly out-of-date versions of Java points to potential issues in how well the average organization manages its software as well as the large attack surface area presented by Java in the majority of organizations.

Attackers can often target old vulnerable versions installed on the endpoint

It is perhaps not well known outside the security research community that malicious Java code can target outdated instances of Java even after the most recent version of Java has been installed on an endpoint. Sometime during the Java 6 family of updates, the Java updaters began removing some older versions of Java. However, an installer for a major version of Java does not remove versions of Java from older major versions. For example, installing a version 6 of the Java runtime will not remove any Java 5 or Java 4 (aka Java 1.4) versions of the runtime. In addition, in our testing, installing Java 6 Update 30 when Java 6 Update 13 had been installed previously resulted in contradictory information about the installed version.

This fact has led to products, such as the open source JavaRa tool, whose sole purpose is to help users deal with the problem of identifying and removing old versions of Java.

In the following screen capture, a Web page using Oracle’s Java Deployment Toolkit reports that 6 Update 30 is installed, while the Java control panel reports that Java 6 Update 13 is installed.

In any case, the fact that older major versions of Java are not removed during installation of newer versions has led to continued high prevalence of very old and vulnerable versions of Java remaining on a high percentage of endpoints.

The following screen capture shows a Java applet executing under Java 5 Update 22, despite the presence of Java 6 Update 13.

Starting with Java 7 Update 21, released in early 2013, the Java launcher will warn the user when code at-
tempts to run against older versions of Java.

While this will mitigate some percentage of attacks, many users will not understand the warning and may choose to allow the code to execute under the old vulnerable version. In addition, so-called “click bypass” vulnerabilities often are discovered in Java, allowing attackers to prevent the mitigating interactive messages from ever being seen by the user. In July 2013, a new vulnerability was found that affects Java major version 7 update 21, the newest version as of our data collection, as well as earlier versions. It allows for attackers to bypass the Java click-2-play security warning dialogue box without user interaction. This means that attackers can still target an older version on an endpoint, without user notification.

The latest version, Java 7 Update 25 goes further and will not allow users to select older versions to run against, as seen here. It remains to be seen if click-bypass vulnerabilities are more difficult to uncover, however.
Fewer than 1 percent of organizations have installed the latest version of Java

As of the time of the data collection the most up-to-date version of Java, version 7 update 21, was only observed on 3 percent of all endpoints. Since that version was only observed in 0.26% of organizations it would seem that larger environments with higher endpoint totals are more likely to install the latest version of Java than smaller organizations. Overall, less than 1 percent of organizations represented in our analysis had installed the most current version of Java. It seems likely, then, that most organizations also have not installed the more recent updates of Java that have been made available since the data was gathered for this report.

It seems reasonable to conclude that most organizations are susceptible to a large number of old vulnerabilities for which fixes are available simply due to lack of updating. However, even if users were running the latest version in their environment, they would still potentially be susceptible to attack due to old versions of Java being present in their environment—especially older major versions such as version 6.
Additional Findings

In addition to the key findings relating specifically to Java’s prevalence, age and exposure to risk on the endpoint, several other items of interest were observed as a result of the effort to compile this report.

Quality is lacking in NIST data

While the NIST CVE data is a useful resource for vulnerability information, it is not without some issues. Product names are not well normalized. For example, some vulnerabilities are reported under the product “Java,” while others (most in fact) are reported under the product “JRE.” Version strings in reports sometimes contain spaces where others do not (“Update 19” versus “Update19,” for example). The CPE names reflect these inconsistencies using corresponding redundant “Update_19” and “Update19” strings. In addition, the change to the version format of Java introduced reporting inconsistencies, with some vulnerabilities being reported, for example, against “version 6 Update 10,” and some being reported against “version 1.6.0 Update 10.”

Java ranks number 16 (as Sun/JRE) in the list of “Top 50 Products By Total Number of ‘Distinct’ Vulnerabilities,” with 397 vulnerabilities. However, if the data quality issues above were addressed, this number would far exceed 500 and put Java in the top 10.

Some organizations choose to remove Java entirely

There is both anecdotal and quantitative evidence that some organizations are choosing to remove Java from their environment altogether, as many in the security industry have recommended. According to the collected data, some large organizations have successfully reduced the average number of installed Java instances per endpoint to as low as one instance per 50 endpoints (0.02 versions per endpoint).
Conclusion

Java is installed widely in enterprises, and most instances are highly vulnerable. The way updating and installing works has led to a proliferation of versions on endpoints. Java continues to be a required technology for many companies, but its ubiquity seems to be out of proportion with its current business use cases. Many enterprises appear to be choosing to remove Java from their environments, and the facts in this report underscore the rationale for doing so.

The data also seems to show the reasons behind Java’s new favored status among attackers. The data suggest that many enterprises have so far been slow to respond appropriately to this trend, despite evidence that doing so would, for many, substantially reduce their exposure to today’s most common successful attacks.

While Oracle appears to be making efforts to mitigate some of the issues that have brought us to where we are today, those efforts will have little impact on remediating the current situation. Enterprises can benefit from better characterizing and understanding the applications running on the endpoints in their environment, so they can better understand the risks to those endpoints and more effectively prioritize remediation efforts. In addition, it is clear that while enterprises have made great strides in rolling out Windows patches, patching for third-party applications still remains a challenge.
Recommendations

• Evaluate whether Java is necessary for your organization.

• If removal is chosen, develop a plan:
  • Consider tools that can block execution of software based on name or hash on the end-point as a quick first step toward the eventual goal of removal.
  • Use software management tools to remove instances of Java.
  • Close the loop—audit the software in your enterprise to confirm Java’s removal.
  • Use network security solutions such as those with layer 7 visibility to look for evidence of browser-based Java.

• Monitor for unexpected installation and use of Java in your environment.

• There is no need to go overboard with controlling or removing Java. Many products contain a version of Java embedded that is not made available to the browser, and thus poses little risk of Web-based infection.

• When using NIST data to find vulnerability information, keep in mind that software and vulnerabilities may not always be recorded in a consistent manner. Explore different searches that may yield additional results.
Methodology

We leveraged Bit9’s data about endpoints across many deployments to collect data about Java usage, such as prevalence and age. The data represents approximately one million endpoints and several hundred deployments. The deployments were selected based on recency and reliability of the data reported by the servers. The data also was audited previously and spot-checked for quality, and where applicable, compared with other data sources for validation.

The search for installations was performed by looking for the presence of hashes associated with “java.exe.” While this search also uncovered a small amount of malware and other non-Java software, these were generally of low prevalence and were filtered out for most analyses in this report.

It is also important to note that while there are many legitimate versions of Java that are not associated with Sun and Oracle, this report focuses on the Sun/Oracle instances that are most likely installed for use with the browser and therefore comprise the vast majority of Java-related remotely exploitable vulnerabilities. This report also restricts itself to Microsoft Windows endpoints, and does not contain data collected about Java on other platforms such as Mac and Linux.

In some cases, particularly for what appear to be Java versions 5 or more years old, data such as the Portable Executable (PE) header metadata that contains version information often was unavailable. This appeared to be the case with Java versions prior to Java 5. In those cases, we often had compile time metadata, which we used to validate that the date these versions were created was consistent with versions older than Java 5. Accordingly, we grouped these older versions together and treated them as a set for the purposes of our analysis.

The National Institute of Standards and Technology (NIST) publicly searchable database of Common Vulnerabilities and Exposures (CVE) data, which details reported software vulnerabilities, was used for determining the severity of vulnerabilities associated with Java versions found in deployments.

It is helpful to note that the popular way to refer to Java versions changed sometime during the major version 1.5 series. Prior to this change, versions were referred to with standard software dotted version notations 1.3, 1.4.2, etc. After the change, the series was referred to by the second digit, the minor version was essentially dropped, and the minor releases were referred to as “up-
dates.” So, for example, Java 5 update 20 corresponds to the Java version string “1.5.0_20”, where the third digit is always ‘0’, and the digit following the underscore, if present, refers to the update or minor release number. Java versions earlier than 5 were uniformly referred to using the dotted notation, e.g., 1.4, etc.
Sources

1. Oracle Java surpasses Adobe Reader as the most frequently exploited software, Kaspersky, http://www.kaspersky.com/about/news/virus/2012/Oracle_Java_surpasses_Adobe_Reader_as_the_most_frequently_exploited_software
5. Copyright (c) 2006, 2011, Oracle and/or its affiliates, Oracle, http://www.java.com/js/deployJava.txt
About Bit9

The Bit9 Security Platform is the only next-generation endpoint and server security solution that continuously monitors and records all activity on endpoints and servers and stops cyber threats that evade traditional security defenses. Bit9's real-time sensor and recorder, cloud-based services and real-time enforcement engine give organizations immediate visibility to everything running on their endpoints and servers; real-time signature-less detection of and protection against advanced threats; and a recorded history of all endpoint and server activity for deep forensics. Bit9 has stopped the most advanced attacks, including Flame, Gauss and the malware responsible for the RSA breach. 1,000 organizations worldwide – from 25 Fortune 100 companies to small businesses – use Bit9 to increase security, reduce operational costs and improve compliance.