WHERE'S THE FIRE?

Java Security Fears Are More "Smoke and Mirrors" Than Five-Alarm Fire

In fact, the government agency tasked with monitoring technological security vulnerabilities ranks Java as more secure than dozens of other technologies and products. This paper will explore the disconnect between inflammatory rhetoric and data-backed reality.

Java security is a disaster, if the headlines are to be believed. No other technology poses such an egregious risk, they tell us. The only solution, many pundits and proponents of other technologies insist, is the nuclear option: completely uninstall Java from your computers, servers and networks.

Is Java truly such an abysmal security failure, as portrayed by the press and non-Java software companies? No, but the point of this paper is not to prove that it's totally secure. Yes, Java has security vulnerabilities. So does *everything* else. *No* technology platform or product is totally secure; and the surprising reality is that, in the spectrum of security risk, Java does not fall anywhere close to where you think it does.

In fact, as we'll explore shortly, out of the top 30 most vulnerable tech products, Java falls around #25.

"But everyone says it's insecure!" you protest. Welcome to FUD: fear, uncertainty and doubt. FUD is not a realistic portrayal of the state of Java security; it's a propagandistic tactic meant to influence public perception with the overwhelming presentation of negative, dubious or outright inaccurate information. FUD is also known as "fear-mongering," and you may recognize it from other areas of life, like politics.

But fear not: in this paper, we are going to ask and answer three very specific, pertinent questions whose answers are then drawn from reputable, independent sources.

1. Does Java face *more* security vulnerabilities than other technologies?
2. Are Java security vulnerabilities more severe?
3. Does Java face more high-severity vulnerabilities *specifically* than other technologies?

The answer to all three questions is no, but it's no better to take that answer at face value than to accept the FUD without question. Instead, let's do a deep dive into government-sourced and backed data analysis for an answer that cuts through the smoke.

Java Is Not (Necessarily) What You Think

To start, let's point out an important caveat: not everyone means the same thing when they say "Java." As a class of technologies, Java has been hugely successful – Java Virtual Machine, for example, runs on an estimated 89% of computers, 3 billion mobile phones and 125 million TV devices.[i]

In fact, part of Java's reputation for insecurity is due to its ubiquity; if it has had higher numbers than other technologies, it also has more developers and end-users, making it a more attractive target for malicious agents and bad actors, a fact that has no bearing on its intrinsic security relative to other platforms. As the SANS Institute for computer security training says:

> *"The risks arise because [Java and .NET] are the [languages] commonly used to build big, feature-rich, business-critical applications with a lot of valuable code, especially legacy code written by developers who didn't understand secure development – code that is exposed to attack."*[ii]

It's analogous to the security of Mac OS X or Linux versus Windows: the latter gets targeted much more frequently due to its massively disproportionate market penetration. For Java, developers embraced its "write once, run anywhere" principle, and the result is a similarly enormous market penetration. According to a Stack Overflow survey, JavaScript is the most commonly used programming language in 2015 (54.5%), with Java in the third spot at 37.4%.[iii]

But that statement illustrates our point that not everyone means the same thing by "Java." JavaScript and Java are *not* the same thing; and Java itself has multiple versions, iterations, forms, and platforms. And not all of them bear the same security flaws.

And if you're running an outdated version of Java (i.e., Sun rather than Oracle), you *are* risking your security. But does modern Java, for those following best practices, pose the ponderous risks claimed?

Be prepared to be surprised.

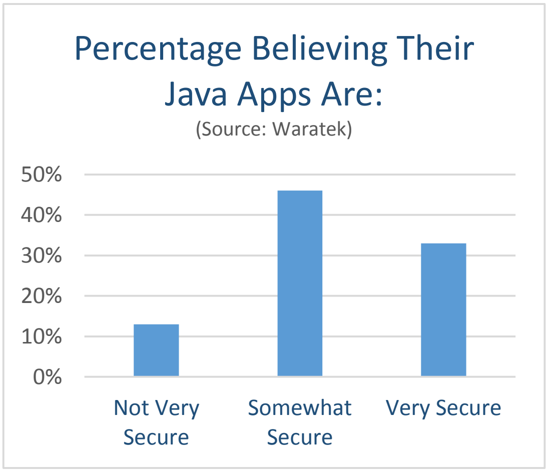
Java's Security Improvements Are Outpacing Public Perception

Additionally, even if Java did deserve its reputation in full, improvements to its security have been fast outpacing public perception.

According to Cisco's 2015 Annual Security Report, Java exploits declined 34% in 2014, with 54 urgent Java vulnerabilities in 2013 versus only 19 in 2014, as a result of security improvements.[iv]

These improvements are partially because Oracle (which has owned Java since 2010) has dramatically improved its patching response speed. "Java Virtual Machine (JVM) is continually being updated," according to Martin Roesch, chief architect of the security business group at Cisco. "I suspect that many Java attacks are against the JVM, and the JVM is just getting better with better security."[v]

Those findings track with the experience of end-users. Eighty percent of respondents to a 2014 survey of IT professionals believe Java apps to be "very" or at least "somewhat" secure, with fully one-third (33%) deeming it "very" secure (see chart).[vi]



In fact, the Cisco report actually called out Adobe Flash, PDF Reader and web browsers like Microsoft's Internet Explorer as having the most application vulnerabilities and exploits. That has serious implications for services delivered via web browsers, particularly in sensitive environments dealing in private information, like health care, financial and government offices.

That brings us to our first major question: does Java face *more* security vulnerabilities than other technologies?

Question #1: How Many Total Vulnerabilities Affect Java?

For our data, we're going to turn toward the MITRE Corporation, a not-for-profit organization that's partially funded by the National Cyber Security Division of the U.S. Department of Homeland Security and established back in 1985. MITRE manages the federally-funded research and development centers that support the DHS, DOD, FAA, IRS, VA and others. Interestingly, mitre.org was the first ".org" domain name ever registered.

MITRE created a system – the Common Vulnerabilities and Exposures (CVE) system – to track and categorize publicly known information-security vulnerabilities and exposures. MITRE reports, validates, documents, and then makes these issues public, publishing the CVE List Master Copy on a monthly basis at cve.mitre.org.

Why MITRE? They are a fact-driven, dispassionate source of information. "MITRE is chartered to work in the public interest. We have no commercial interests."[vii] They cut through the FUD to pull out the truth which means, for example, they don't cherry-pick particular months or data sets to prove their case.

Why the CVE? It's clear, factual, and easy-to-use. It takes the top 50 technology products and platforms and ranks them by number of certified vulnerabilities, with the highest number of vulnerabilities ranked at #1 (in other words, higher is worse).

When we look at a recent data set (see Figure 1[viii], next page), we observe that no form of Java – not a single iteration – falls in the Top 10. Web browsers and operating systems, even Mac OS X and iOS, have more certified vulnerabilities than Java.

Again, that has security implications for zero footprint clients and non-Java software that passes data through the web browser: since platforms like Chrome, Internet Explorer and Firefox all lead the pack in vulnerabilities, any browser-based software solution risks those additional vectors of attack, on top of any inherent vulnerabilities.

Other than outdated versions of Java, Java doesn't make its first appearance until spot 25, where we finally find Java Runtime Environment (JRE, installed on end-user devices in order to *run* Java apps) and Java Development Kit (JDK, which developers use to *write* Java applications but is *not* installed on end-user devices). However, those two iterations use the same code base, so they share the same vulnerabilities; that's why they appear right next to each other.

So that explains our point that Java faces notably fewer vulnerabilities than many universal platforms, browsers and OSes. But there's still a fair question to ask: even if Java has fewer vulnerabilities than many other technologies, are its vulnerabilities more severe?

Figure 1. Total Number of Vulnerabilities

Rank	Product	Vendor	Vulnerabilities
1	Linux Kernel	Linux	1289
2	Firefox	Mozilla	1187
3	Chrome	Google	1095
4	Mac Os X	Apple	1081
5	Windows Xp	Microsoft	728
6	Seamonkey	Mozilla	695
7	Thunderbird	Mozilla	676
8	IE	Microsoft	632
9	Mac Os X Server	Apple	626
10	Safari	Apple	565
11	Internet Explorer	Microsoft	548
12	Windows Server 2008	Microsoft	543
13	Windows Vista	Microsoft	538
14	Solaris	SUN	533
15	Windows 2000	Microsoft	508
16	Iphone Os	Apple	495
17	Flash Player	Adobe	459
18	JRE	SUN	435
19	Windows 2003 Server	Microsoft	429
20	Windows 7	Microsoft	416
21	Windows Server 2003	Microsoft	408
22	JDK	SUN	405
23	PHP	PHP	399
24	Database Server	Oracle	390
25	JRE	Oracle	389
26	Acrobat Reader	Adobe	389
27	JDK	Oracle	378
28	Acrobat	Adobe	373
29	IOS	Cisco	372
30	AIX	IBM	321

Figure 2. Average Severity of Vulnerabilities

Rank	Product	Vendor	Weighted Avg.
1	Internet Explorer	Microsoft	9.4
2	Acrobat	Adobe	9.4
3	Office	Microsoft	9.4
4	Flash Player	Adobe	9.3
5	Acrobat Reader	Adobe	9.2
6	Quicktime	Apple	8.6
7	Firefox Esr	Mozilla	8.4
8	Itunes	Apple	8.4
9	Windows Server 2003	Microsoft	8.2
10	Thunderbird	Mozilla	8.1
11	Windows Vista	Microsoft	8.1
12	Webkit	Apple	8.1
13	Seamonkey	Mozilla	8
14	Windows Server 2008	Microsoft	8
15	Windows Xp	Microsoft	7.9
16	Windows 2003 Server	Microsoft	7.9
17	Windows 7	Microsoft	7.9
18	JRE	SUN	7.8
19	JDK	SUN	7.8
20	Firefox	Mozilla	7.6
21	Windows 2000	Microsoft	7.6
22	JRE	Oracle	7.6
23	JDK	Oracle	7.6
24	Chrome	Google	7.5
25	Safari	Apple	7.5
26	IOS	Cisco	7.3
27	AIX	IBM	7.3
28	IE	Microsoft	7.2
29	Database Server	Oracle	7.1
30	Hp-ux	HP	6.9

Question #2: How Severe Are The Vulnerabilities?

To answer this question, we're going to turn to a different information source. First, we need to classify the severity of a vulnerability. To do that, we need to use a standardized system: CVSS, the Common Vulnerability Scoring System. The Department of Homeland Security National Cyber Security Division maintains a National Vulnerability Database and uses CVSS as the primary method of quantifying the severity of vulnerabilities on a scale of 0 to 10 (see Figure 3).

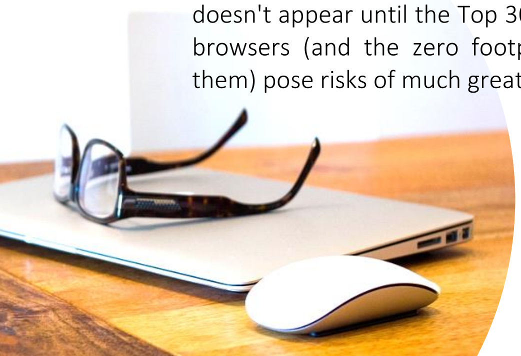
Oracle Java Runtime has a weighted average of 7.7, so it does fall in the high category. But the question we're investigating: are Java vulnerabilities more severe than others, on average? No: once again, Java doesn't appear until the Top 30 (see Figure 2), while browsers (and the zero footprint clients that use them) pose risks of much greater severity.

Figure 3. CVSS Weighted Average for Vulnerabilities

Score	Severity
0.0 – 3.9	Low
4.0 – 6.9	Medium
7.0 – 10.0	High

So not only does Java have fewer *total* vulnerabilities (the first question), neither are those vulnerabilities as severe as the others'. But could it be that Java faces more high-severity vulnerabilities specifically?

In other words, if we look solely at the high-severity problems and compare just those, does Java suddenly deserves its reputation? We'll look at that question next.



Question #3: How Many *Severe* Vulnerabilities Affect Java?

Figure 4. Summary of CVE and CVSS Appraisal of Java Security Vulnerabilities

Analysis	Rank
Total # of Vulnerabilities	25th
Average Severity	22nd
Total High Severity	27th
Total Medium *and* High	27th

To answer this question, we're going to examine the total number of vulnerabilities with a "high" severity score (between 7.0 and 10.0 according to the CVSS). We'll be brief: once again, JRE and JDK don't appear until the Top 30 (with 177 such high-severity vulnerabilities), well behind browsers and operating systems that pose greater security risks than Java.

In fact, the #1 spot is occupied – surprisingly – by Chrome, with three times the number of high-severity vulnerabilities as Java. Funny how no articles cry, "Chrome is the biggest vulnerability for U.S. computers."[1]

Why So Much Smoke, If So Little Fire?

Yes, we should absolutely be concerned with Java vulnerabilities. We should be concerned with vulnerabilities on *all* platforms. But we also need to understand that not all Java is equal, and inflammatory rhetoric around its security flaws is unhelpful – it's just FUD – without realistically understanding Java's various risk profiles.

Any given system or network will be vulnerable to many different vectors of attack. Java is simply one of many, and it is neither the worst nor the weakest.

Proper risk management that uses industry standards and information published by federal authorities to cut through the smoke will be able to accurately, effectively prioritize the most pressing vulnerabilities.

Those organizations will then be able to create policies and make software choices that effectively mitigate the real problems, rather than the trumped up ones.

The Java Browser Plug-In: A Closer Look

The vast majority of the identified vulnerabilities relate to the Java plug-in for web browsers. When the user browses a web site that is malicious or compromised, such content is able to leverage Java vulnerabilities in the browser through the plug-in.

But Java is much more than just a browser plug-in. Full-blown Java applications (i.e., not zero footprint clients), running locally, accessible and usable *without* the browser, are much safer. It's the *applets*, as security vendor Sophos describes them, "which are delivered into your browser as you use the internet [that's] where the risk from Java presents itself." Specifically, the browser-based Java runs a deliberately restricted version of the Java environment, but those restrictions don't "always impose the limitations it should, due to software vulnerabilities in Java." [xi]

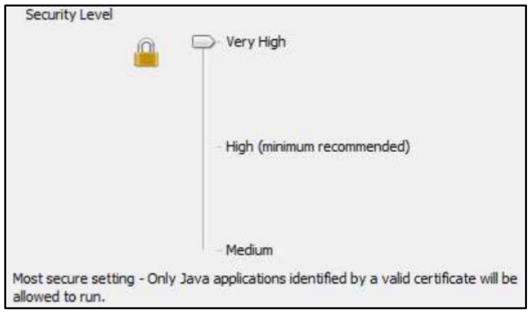
In other words, if we were to apply the same comparison test we just did for Java against other technologies, and did it just for specific iterations of Java, the browser plug-in would be at the top of the list. Full-blown Java would be at or near the bottom.

[1] *See CSO Online's "Java is the Biggest Vulnerability for U.S. Computers." The title is provocative, but clarifications inside the main text indicate that it's* outdated *instances of Java that are posing the risk. People who keep their Java up-to-date are actually in a strong position, security-wise.*

Seven Best Practices For CIOs Setting Java Security Policy

1 Put Java in perspective. More than anything else, strive for a realistic understanding of Java's strengths and weaknesses. Any single computer offers many security vulnerabilities, and therefore many vectors of attack. Proper risk management uses industry standards and information published by federal authorities to prioritize the most pressing vulnerabilities. Organizations can then create policies that effectively mitigate the real problems, rather than the inflated ones.

2 Apply security standards to all platforms. Java security should still be taken seriously, of course; and that means turning your Java security up as high you can (Figure 5). We recommend setting the slider at "very high," which allows only *trusted* applications to run (see the next best practice for an explanation of what that means).



Most secure setting - Only Java applications identified by a valid certificate will be allowed to run.

Figure 5. Java Security Settings

3 Only use trusted, signed apps from trusted sites. Trusted, signed apps reduce risk by ensuring the applications have not been altered since the official release. Also, only install updates made available through official channels.

4 Update regularly. According to StatOwl.com, at least 47% of Java users are still running Java 6; due to tracking limits, the number could actually be higher. That means half or more of Java users are using an outdated version of Java that does not provide the same security protections as the most recent.[ix] Those users who keep Java up-to-date face far, far fewer problems. "There are still old Java exploits floating around, but the Java Virtual Machine (JVM) is continually being updated," Martin Roesch, chief architect of the security business group at Cisco, told *eWEEK*.[x]

5 Provide security awareness training. Security is not just a tech problem, it's a people problem, and users need to understand how to use software properly and safely. Similarly, IT technicians need to know how to properly update and maintain software, or at least know whom to contact for assistance.

6 Do not use the Java *plug-in* if you can avoid it. Remember, Java *applications* and Java *applets* are two separate things and carry two separate risk profiles. It may not be possible to disable the plug-in entirely, for a variety of reasons, but you can minimize its usage; also choose fully-fledged Java clients over zero footprint. Power tip: disable the Java plug-in on your primary browser and use it when needed only a secondary browser, which you use to access *only* trusted websites, or use it only in a virtualized environment. That approach doesn't provide perfect security – it doesn't protect against watering-hole attacks, for example – but it minimizes the risk landscape.

7 Learn to discern good information from bad. Headlines often fail to tell the whole story. Learn to look for clarifications, like which *kinds* of Java are being referenced (browser plug-in? outdated installations? zero footprint clients versus applications?). Identify speculation, assumptions and over-generalizations. Business environments are very different from personal computing, for example; and studies that examine private use of Java are unlikely to apply to corporate or government usage.

Where OpenFox® Fits: Our Risk Profile

We've been making the point that the risk profiles of different iterations of Java vary enormously, from the higher risk of the browser plug-ins to the much better than expected security of modern, up-to-date JRE and JVM installations.

Since OpenFox® is a Java application, it logically follows to ask: where does *its* security profile fit into our discussion?

OpenFox® is a real client.

You undoubtedly noted that the most popular web browsers appear above in the Top 10 lists of vulnerabilities over and over again; and we've already talked about the vulnerabilities of the Java browser plug-in. OpenFox® bypasses all of that *entirely*. CPI does *not* pass CJIS data through a browser. Instead, the web browser is used to download the application initially, but OpenFox® itself is a full-fledged, self-contained application running locally on your machine. It does not run within a browser, nor does it require your users to use the Java plug-in, or even for it to be enabled.

OpenFox® Messenger is a "Trusted Application."

This falls in line with one of our Best Practice recommendations. The program was published by CPI and has gone untouched by anyone else; no unverified third-party code is incorporated into the program. It has not been altered in any way since it was published by CPI's controlled release process. Further, the certificate is issued by industry giants Symantec/Verisign.

OpenFox® has been built with security in mind.

What we've learned over the course of this paper is that (1) Java has fewer security vulnerabilities than dozens of other commonly used technologies; (2) Java security vulnerabilities are no worse than others, and are less severe than dozens of others; and (3) Java has fewer high-severity security vulnerabilities (those scoring 7.0+ on the CVSS scale) than dozens of others. Since OpenFox® is built on the Java platform, all of those statements are true of OpenFox® as well.

Altogether, OpenFox® has been specifically engineered to minimize its already low risk profile. That's why it's designed as a full fledged application (not as a zero footprint browser client), why we don't require the use of the browser plug-in to use it, and why we take such care in certifying the software.

Proven Progress. Proven Protection.

Computer Projects of Illinois, Inc. (CPI), with its headquarters in Bolingbrook, Illinois, is a privately held corporation and an acknowledged leader in information-sharing software systems for the law enforcement and criminal justice community.

CPI's sole focus has been, and will continue to be, this sector. CPI expends all of our energies on the development, installation and maintenance of our software products. CPI systems are state-of-the-art and cost-effective; ensuring that our customers get the most for their investment.

Computer Projects of Illinois, Inc.
475 Quadrangle Drive, Suite A
Bolingbrook, IL 60440

Tel: (630) 754-8820
Fax: (630) 754-8835
Help Desk: 866-471-6305

www.openfox.com

The "OpenFox" Company

References

[i] Java. "Learn About Java Technology." Retrieved June 2015 from https://www.java.com/en/about/.
[ii] Bird, J., Johnson, E. & Kim, F. (2015 May). "2015 State of Application Security: Closing the Gap." *SANS Institute InfoSec Reading Room.* Retrieved June 2015 from https://www.sans.org/reading-room/whitepapers/analyst/2015-state-application-security-closing-gap-35942.
[iii] Avram, A. (2015, Apr 9). "Stack Overflow Survey 2015: Technologies Used, Loved, Disliked or Wanted." *InfoQ.* Retrieved June 2015 from http://www.infoq.com/news/2015/04/stack-overflow-survey-2015.
[iv] Cisco. "Cisco 2015 Annual Security Report." Retrieved June 2015 from http://www.cisco.com/web/offers/lp/2015-annual-security-report/index.html.
[v] Schwartz, M. (2013, Mar 8). "9 Must-Know Java Security Facts." *InformationWeek.* Retrieved June 2015 from http://www.informationweek.com/security/application-security/9-must-know-java-security-facts/240150346.
[vi] Barker, I. (2014, Oct). "Professionals believe their Java apps are secure despite relying on third-party code." *BetaNews.* Retrieved June 2015 from http://betanews.com/2014/10/08/professionals-believe-their-java-apps-are-secure-despite-relying-on-third-party-code/.
[vii] MITRE. "Corporate Overview." Retrieved June 2015 from http://www.mitre.org/about/corporate-overview.
[viii] MITRE. "Common Vulnerabilities and Exposures." Retrieved May 2015 from https://cve.mitre.org/.
[ix] Schwartz, M. (2013, Mar 8). "9 Must-Know Java Security Facts." *InformationWeek.* Retrieved June 2015 from http://www.informationweek.com/security/application-security/9-must-know-java-security-facts/240150346
[x] Kerner, S. (2015, Jan 20). "Java no longer such a big risk, Cisco security report finds." *eWeek.* Retrieved June 2015 from http://www.eweek.com/security/java-no-longer-such-a-big-risk-cisco-security-report-finds.html.
[xi] Ducklin, P. (2014, Apr 16). "No Hearbleed holes in Java, but here comes a sea of patches anyway." *Sophos Naked Security.* Retrieved June 2015 from https://nakedsecurity.sophos.com/2014/04/16/no-heartbleed-holes-in-java-but-here-comes-a-sea-of-patches-anyway/