
# The Unfortunate *Reality* of Insecure Libraries

**Contrast Security**
9175 Guilford Road, Suite 300
Columbia, MD 21046
www.contrastsecurity.com

## Unfortunate Reality RELOADED

Just over two years ago, Aspect Security wrote a paper describing a serious but widely ignored problem in the world of application security – using components with known vulnerabilities. Since then, the use of open source software has more than doubled from 6 billion to 13 billion component downloads per year. And there is now clear evidence that hackers are starting to focus on the software supply chain, including third party components.

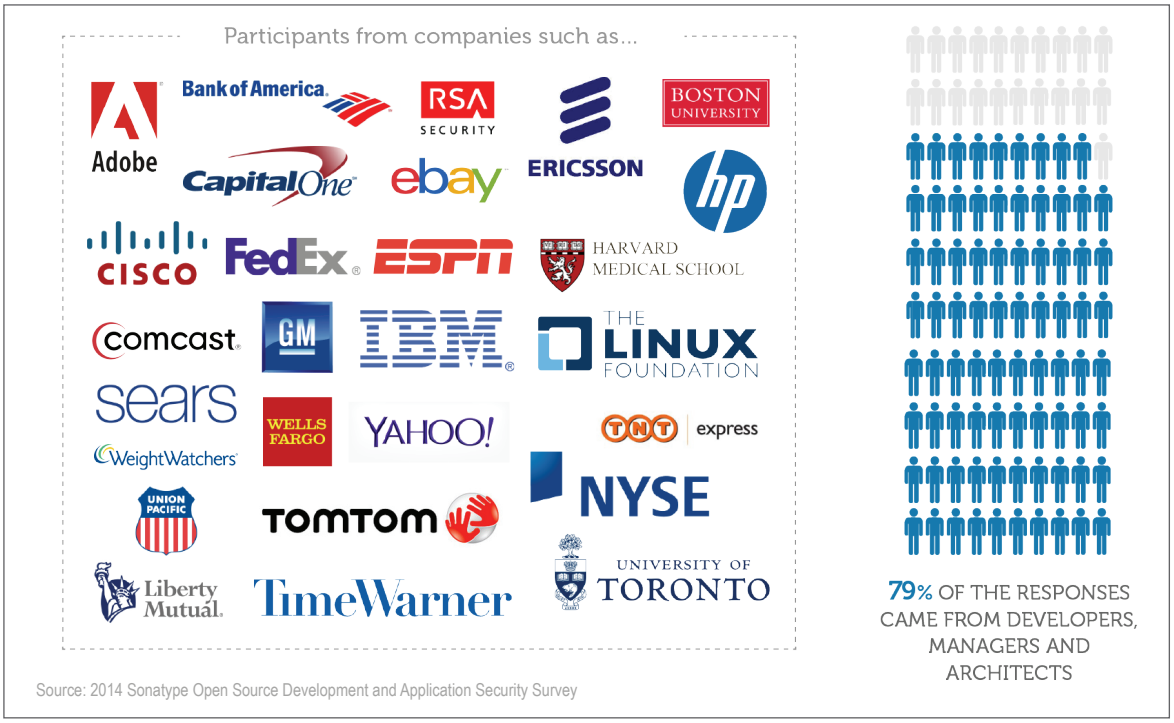
After two years, many organizations have taken notice of this important problem:

- OWASP added A9 "Use of Components with Known Vulnerabilities" to the OWASP Top Ten.
- PCI 3.0 now requires the use of secure components.
- FS-ISAC issued a new standard governing open source component use.
- MITRE CWE-937 was added to focus on components with known vulnerabilities.

There are now several commercial and open source tools to help manage the challenge:

- Contrast Security – real time continuous monitoring for known and unknown vulnerabilities
- Sonatype – management for development organization policy enforcement
- OWASP Dependency Check – open source tool to analyze an application's components

However, there is still a long way to go. The 2014 Fourth Annual "Open Source Development and Application Security Survey" sponsored by Contrast Security and Sonatype assessed the state of the art. Over 3,300 respondents from a wide range of major companies were surveyed.




The study showed that:

- A quarter of organizations continue to run without an open source policy
- Of those organizations with an open source policy, only 68% actually follow it
- Only 22% of organizations have actually banned an open source component found to have vulnerabilities
- 75% don't have meaningful controls over what components are in their applications
- Less than 1% of security budgets are spent on application security
- Yet application security is the leading cause of breaches

Click to view infographic [1]

Contrast Security is committed to helping organizations find and fix the vulnerabilities in their application portfolio that really matter. The recent HeartBleed incident drives home the importance of ensuring the security of the components that you use to run your business.

Contrast is a new approach to application security that doesn't require scanning. Contrast's agent installs on your application servers across the development lifecycle. As soon as you include a library in your project, Contrast checks to be sure there are no known vulnerabilities. Then Contrast goes on to analyze exactly how your application uses those libraries, identifying the unknown vulnerabilities that no other tool can see. You can be up and running with Contrast in a matter of minutes without changing anything about your build, test, deployment, and operation of your applications. Contrast is designed to easily scale across your entire application portfolio.

Your applications are the lifeblood of your business, and you deserve continuous protection. Contrast is the only application security solution that runs continuously as you build, test, and operate your application. So when the next big component vulnerability is identified, Contrast will instantly let you know exactly which applications are affected and what you need to do to fix the problem.

[1] Infographic courtesy of Sonatype, Inc.

Eighty percent of the code in today's applications come from libraries and frameworks, but the risk of vulnerabilities in these components is widely ignored and under appreciated. A vulnerable library can allow an attacker to exploit the full privilege of the application, including accessing any data, executing transactions, stealing files, and communicating with the Internet. Organizations literally trust their business to the libraries they use.

## Abstract

In partnership with Sonatype, researchers from Aspect Security analyzed 113 million downloads from the Central Repository ("Central") of the 31 most popular Java frameworks and security libraries and made some conclusions about this important aspect of application security. Central is the industry's most widely used repository of open-source components, and currently contains more than 300,000 libraries. The study analyzed more than 113 million downloads of these libraries from more than 60,000 commercial, government, and non-profit organizations.

The analysis revealed several interesting findings, including:

- 29.8 million (26%) of library downloads have known vulnerabilities
- The most downloaded vulnerable libraries were GWT, Xerces, Spring MVC, and Struts 1.x
- Security libraries are slightly more likely to have a vulnerability than frameworks
- Based on typical vulnerability rates, the vast majority of library flaws remain undiscovered
- Neither presence nor absence of historical vulnerabilities is a useful security indicator
- Typical Java applications are likely to include at least one vulnerable library

The data show that most organizations do not appear to have a strong process in place for ensuring that the libraries they rely upon are up-to-date and free from known vulnerabilities. We conclude that there are no shortcuts to a secure application infrastructure and that the only useful indicator of library security is a broad and rigorous review that finds minimal vulnerability.

This paper is not a critique of open source libraries, and we caution against interpreting this analysis as such. The authors are strong advocates of free and open software and created two of the security libraries studied. Instead, we recommend that organizations recognize that libraries are a critical part of their software infrastructure and ensure they have the level of awareness and the necessary tooling within their organization to generate appropriate assurance.

## Study Design

In partnership with Sonatype, Aspect Security researchers analyzed more than 113 million downloads of the 31 most popular Java frameworks and security libraries from the Central Repository. The 31 libraries were selected by examining over 500 enterprise applications submitted to Aspect Security for code review and security testing in the past 12 months.

The 20 frameworks and 11 security libraries selected are a small but frequently downloaded subset of the more than 36,000 libraries (and 303,000 total versions) in the Central Repository ("Central"). The dataset analyzed the downloads of these libraries from Central in a year's time:

| Dataset | Value |
|---|---|
| **Libraries** | 31 |
| **Library Versions** | 1,261 |
| **Companies** | 61,807 |
| **Downloads** | 113,939,358 |

These numbers represent a partial view into the true picture of library use since many organizations, particularly larger enterprise class organizations, are likely to download libraries to their own local repositories for internal reuse. Downloads from these repositories and direct downloads of libraries from project web sites are not included in the study. These factors notwithstanding, we believe that the large size of the dataset provides strong support for the conclusions of the study.

The study focuses only on open-source Java libraries, but there is no reason to believe that the data for other languages and platforms would be significantly different. Similarly, our experience in evaluating the security of hundreds of custom applications indicates that the findings are likely to apply to closed-source and commercial libraries as well.

## Library Security Risks

While many organizations are awakening to the idea that their custom code needs security attention, very few have yet realized the risk from libraries.

Libraries are software modules designed to perform commonly required functions. Central contains over 300,000 components with an average of 8 versions each. There are tens of thousands of open-source projects on the Internet to develop these libraries and many commercial closed-source libraries as well. Many organizations have "open-source initiatives" to expand the use of these low-cost resources. The common misperception of open source libraries is that they are of consistently high quality and

secure, due to their widespread usage and the ability of any developer to review the source code to identify and resolve any problems in the code.

Applications leverage these libraries throughout their execution. They provide support for business functions, data access, resource management, communications, and user interface creation. Today's applications commonly use 30 or more libraries, which can comprise up to 80% of the code in an application.

Libraries run with the full privilege of the application, enabling them to access any data, write to any file, and send data to the Internet, literally anything the application could do. Therefore, a vulnerability in these libraries can completely undermine the security of the entire application.

Libraries run with the full privilege of the application, enabling them to access any data, write to any file, and send data to the Internet, literally anything the application could do. Therefore, a vulnerability in these libraries can completely undermine the security of the entire application.

In this study, we focus on two types of libraries that are particularly security critical. The first type is the "framework" – a library that provides the common code necessary to generate a web application. The other type is the "security library" that provides security controls such as encryption, input validation, logging, access control, and other critical security functions.

## Inadvertent Vulnerabilities in Libraries

Inadvertent coding mistakes in libraries can allow attackers to cause serious harm. These mistakes are quite common and are far easier for developers to make than most people realize. In fact, MITRE has cataloged almost 1,000 different classes of these mistakes in their Common Weakness Enumeration (CWE) database. All of these flaws are tricky, and they are neither taught in school nor easy for a coder to identify on their own.

Inadvertent coding mistakes in libraries can allow attackers to cause serious harm. These mistakes are quite common and are far easier for developers to make than most people realize.

Security researchers periodically discover vulnerabilities in libraries and make them available through a disclosure process of their own choosing. Some of these disclosures are coordinated, others simply write blog posts or emails to mailing lists. Our dataset maps the vulnerabilities listed in the MITRE Common Vulnerabilities and Exposures (CVE) and the Open Source Vulnerability Database (OSVDB) to the appropriate library. This mapping was used to analyze the dataset for patterns indicating the causes of vulnerable libraries.

Not all inadvertent vulnerabilities are created equal. While some vulnerabilities allow the complete takeover of the host using them, others might result in data loss or corruption, and still others might provide a bit of useful information to attackers. In most cases, the impact of a vulnerability depends greatly on how the library is used by the application. A flawed library might result in a devastating exposure in one application and no exposure at all in another.

There are a variety of factors that may prevent an inadvertent vulnerability from being exploitable in a particular application. The most common is that the vulnerability is in a part of the library code that is not used by the application. Another reason is that the vulnerability is shielded by input validation, access control, transformation, or other security controls in the application that prevents exploit.

Based on this discussion, one might (wrongly) conclude that the use of vulnerable libraries is an acceptable risk. It might be argued that developers can work around the problems in components. Or developers might rely on automated tools to catch vulnerabilities in applications. Or developers might believe that their penetration testing process will uncover any exploitable problems in the final application.

Unfortunately, in order to work around a problem, developers have to know that it is there. Currently, developers have no way to know that the library versions they are using have known vulnerabilities. They would have to monitor dozens of mailing lists, blogs, and forums in order to stay abreast of this information. Further, development teams are unlikely to find their own vulnerabilities, as it requires extensive security experience and automated tools are largely ineffective at analyzing libraries.

Establishing a process to manage vulnerable library versions in an organization is a relatively easy way to eliminate a significant amount of risk from an application portfolio. By focusing efforts on a small set of approved libraries, one organization was able to eliminate thousands of old vulnerable components and versions and save millions of dollars maintaining their software portfolio.

## Three Recent Vulnerabilities in Very Popular Libraries

**Vulnerability: Struts2 Remote Code Execution**

Struts2 is a fairly popular remake of the highly successful Struts framework. Unfortunately, Struts2 has suffered a series of remote code execution flaws that have affected all known versions. In the last year, Struts2 was downloaded more than 1 million times by over 18,000 organizations. In 2010, a researcher from Google's Security team discovered a unique class of weakness in the library that allowed attackers to execute arbitrary code on any Struts2 web application.

Essentially, their use of the Object Graph Navigation Language (OGNL) including data provided by users, allows attackers to access data objects and invoke arbitrary methods, such as Runtime.exec(). The implication is that a successful exploit could completely compromise an application and the host on which it runs.

CVE-2010-1870 exploit to run arbitrary OS command:

http://example.org/struts2app/myaction?foo=%28%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3D+[...],%20%23_memberAccess[%22allowStaticMethodAccess%22]%3d+[...],%20@java.lang.Runtime@getRuntime%28%29.exec%28%27mkdir%20/tmp/PWND%27%29[...]27meh%27%29]=true

Spring is the most popular application development framework for Java. It was downloaded over 18 million times by over 43,000 organizations in one year, during which time many vulnerable versions were created and downloaded.

CVE-2011-2730 exploit to steal data out of user's session:

http://example.org/springapp/search?query=${requestScope}

**Result:** Your search for: "javax.servlet.forward.request_uri=/ELInjection/eval.htm,javax.servlet.forward.servlet_path=/eval.htm,user.roles=[ADMIN,USER,ANONYMOUS] display name [WebApplicationContext for namespace 'cashflowServlet']; startup by [uid=root];org.springframework.web.servlet.view.InternalResourceView.DISPATCHED_PATH=/var/opt/test/eval.jsp,..." returned zero results.

These flaws resulted in numerous vulnerable downloads within a year. These flaws don't require authentication or special skills to exploit. Since then, Google and others have regularly unearthed similar critical flaws.

**Vulnerability: Spring Expression Language Injection**

Spring is the most popular application development framework for Java. It was downloaded over 18 million times by over 43,000 organizations in one year, during which time many vulnerable versions were created and downloaded.

Aspect Security and Minded Security collaborated on a white paper (https://www aspectsecurity.com/expression-language-injection) in 2011 that discusses a new class of vulnerabilities in Spring's use of Expression Language (EL). This vulnerability allows attackers to submit HTTP parameters that gets interpreted as EL and executed. An exploit can leak data out of the server, including sensitive information such as system data, application data, and user cookies.

This attack doesn't require a lot of skill or authentication. When processing this request, Spring takes the user's parameter and evaluates it. Spring sends the result of this evaluation back to the user, accidentally leaking important session data.

**Vulnerability: CXF – Authentication Bypass**

Apache CXF is a framework for developing Web Services, from small JSON utilities for web applications to a full scale enterprise service bus (ESB). CXF was downloaded 4.2 million times by more than 16,000 organizations in the past 12 months. Since 2010, CXF has had two major vulnerabilities (CVE-2010-2076 and CVE 2012-0803) that allowed attackers to trick any service using CXF to download arbitrary system files and entirely bypass authentication.

The authentication bypass was rated "Critical." A user could access any protected resource by simply not passing in the expected WS-Security "UsernameToken" field. This exposed all business Web Services written In CXF. To many businesses, this is the worst possible vulnerability.

## Most Vulnerabilities Go Undiscovered

In this study, a set of vulnerabilities were examined that were discovered inadvertently by a diverse set of dedicated and talented researchers. The libraries in the study are the most popular components and they have received considerably more scrutiny than the rest of the 300,000 libraries in Central. While the other libraries have fewer "known" vulnerabilities, the expectation is that they actually contain vulnerabilities that are yet to be discovered.

Libraries typically vary in size from 10,000 to 200,000 lines of code. Aspect Security's code review practice has evaluated thousands of applications, and found that custom Java applications contain from 5 to 10 security vulnerabilities per 10,000 lines of code.

Libraries typically vary in size from 10,000 to 200,000 lines of code. Aspect Security's code review practice has evaluated thousands of applications, and found that custom Java applications contain from 5 to 10 security vulnerabilities per 10,000 lines of code. Given these rates, it is extremely unlikely that a library has never had a vulnerability introduced. It is far more likely that libraries have simply not been examined for vulnerabilities. Therefore, libraries with no known vulnerabilities should not automatically be considered safe.

By the same token, libraries with a history of many vulnerabilities should not necessarily be considered unsafe. A responsible application security process will naturally uncover vulnerabilities and report them so that users can protect themselves. Because the discovery of a vulnerability generally affects all previous versions, we should expect most older versions to have known vulnerabilities. The presence of known vulnerabilities is evidence of this responsible process.

Paradoxically, neither the presence nor absence of known vulnerabilities reveals much information about the security of a library. Instead, we suggest that the best indicator of a library's future security is a culture that places value on security and clear evidence of broad and rigorous security analysis.

The dataset included 70 vulnerabilities that affected the 31 libraries in the study. That is roughly one-quarter of what would be expected based on typical vulnerability rates. Are there another 210 vulnerabilities waiting to be discovered in these 31 libraries? Are there 3 million undiscovered vulnerabilities in all of Central? Even if the vulnerability rate were simply extrapolated for the libraries in the sample, we should expect almost 680,000 more vulnerabilities in Central.

Let's assume that malicious attackers are capable of discovering new library vulnerabilities before they become public knowledge and that they are motivated to do so. There is a statistical certainty that numerous undiscovered vulnerabilities exist in the Central repository, and these libraries deserve considerably more security scrutiny than they currently receive.

## Dependency Management and Security

Dependency management is the process used by development teams to identify which libraries their project directly depends on, and recursively determining all of the further dependencies that those libraries require. By making it significantly easier for developers to pull libraries into their projects, the use of dependency management has resulted in the rapid growth of the number of libraries used in applications. This has caused the total size of applications to grow rapidly as more and more libraries are pulled in.

Dependency management has the potential to yield several security benefits. In theory, this process could enable organizations to keep libraries more up-to-date, gain awareness of security vulnerabilities in libraries more quickly, and ensure that libraries have appropriate licenses.

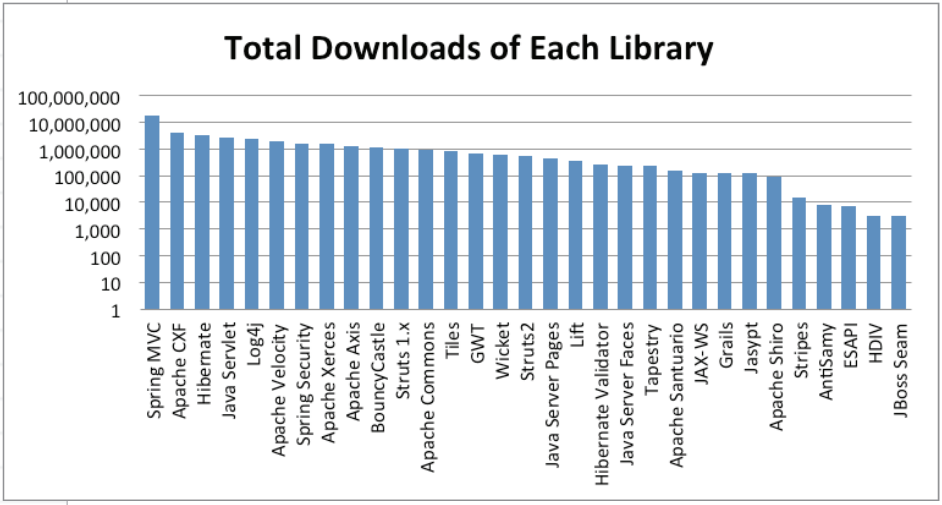
However, these potential security benefits have yet to materialize. Even though there have been ample demonstrations of the cost of not controlling supply chains in other industries, little has been done to establish this control in the software industry. While organizations typically have strong patch management processes for software products, open source libraries are typically not part of these processes. In virtually all development organizations, updates to libraries are handled on an ad-hoc basis, by development teams.

## Java Library Usage Statistics

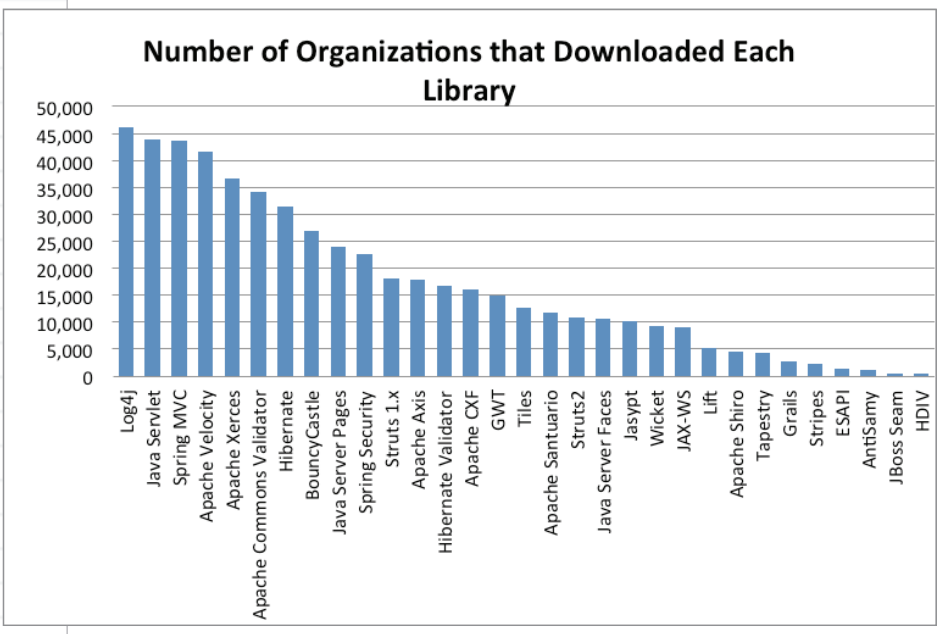
The data reveals extremely heavy use of the 31 libraries by a wide range of organizations, including almost half the Global 500.

The data reveals extremely heavy use of the 31 libraries by a wide range of organizations, including almost half the Global 500. The chart below illustrates that Spring MVC is, by far, the most widely downloaded library studied with more than 10 million downloads in the past twelve months. Spring MVC is a framework library that allows software developers to focus on the business logic and "look and feel" of their application, without having to create and maintain all the "plumbing" to make the application work.

The next two most frequently downloaded libraries are Apache CXF and Hibernate. Apache CXF is a services framework that helps developers build services using a variety of protocols and transports. Hibernate is a persistence library that helps programmers map their data structures into a database.



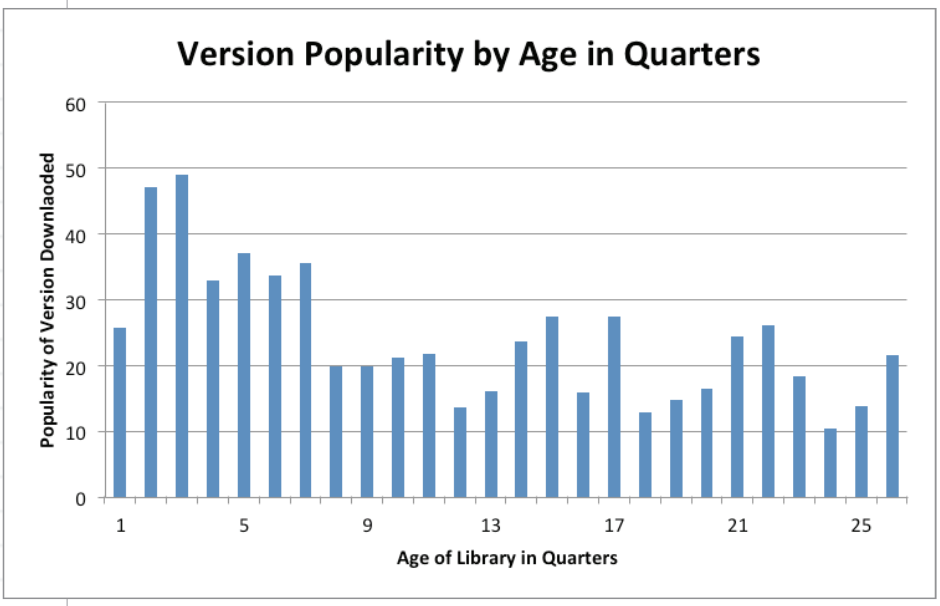

While Spring MVC was the most frequently downloaded library in the study, Log4j reached more organizations. Log4j was downloaded by 45,000 organizations in the last twelve months. One possible explanation is that organizations of all sizes need a strong logging library. Another is that many of the open source frameworks include Log4j as a dependency, and it gets automatically downloaded with the framework.




## Do Organizations Download Old Versions?

Organizations download many old versions of libraries. In fact, several of the libraries that have been around for a longer period of time, such as BouncyCastle (an encryption library), Hibernate (a persistence library), and Struts 1.x (a web framework) all have extremely popular versions that are over 5 years old.

The data is quite complex, as every library has a different release cycle and version scheme. However, the graph below shows that the most popular versions of libraries are within the six months, but after that, the popularity seems to level off, suggesting that libraries are continuing to be downloaded for a very long time. Sonatype ranks version popularity on a scale of 0 to 100, with the most popular artifact receiving a score of 100 and others being scaled down appropriately.



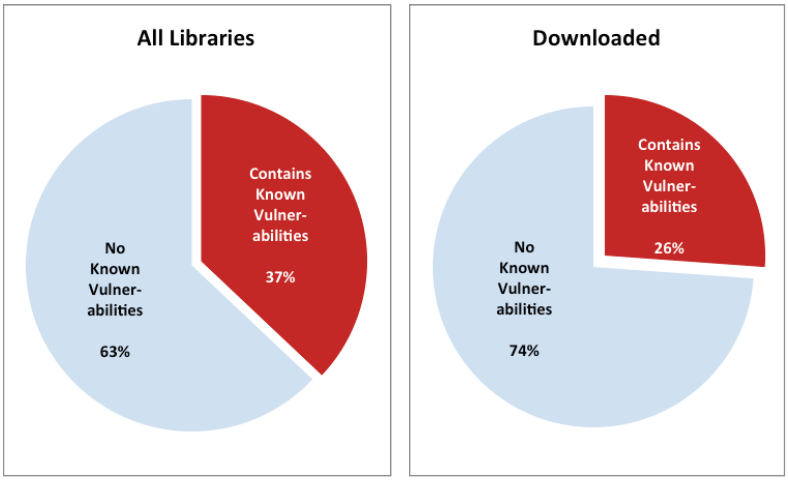

In sum, over a year's time there were 29.8 million vulnerable jar files downloaded, out of a total of 113 million downloads. That magnitude of downloads means there are quite a lot of vulnerable applications in use. This data shows that security is not a major consideration in the determination of which library to download.

If people were updating their libraries, it would be expected that the popularity of older libraries would drop to zero within the first two years. However, the data clearly show popularity extending back over six years. One possible explanation is that some projects, perhaps new development efforts, tend to use the latest version of a library, accounting for the spike in popularity for the libraries in the first year. The continuing popularity of libraries for extended months suggests that incremental releases of legacy applications are not being updated to use the latest versions of libraries but are continuing to use older versions.

## Do Organizations Download Vulnerable Versions?

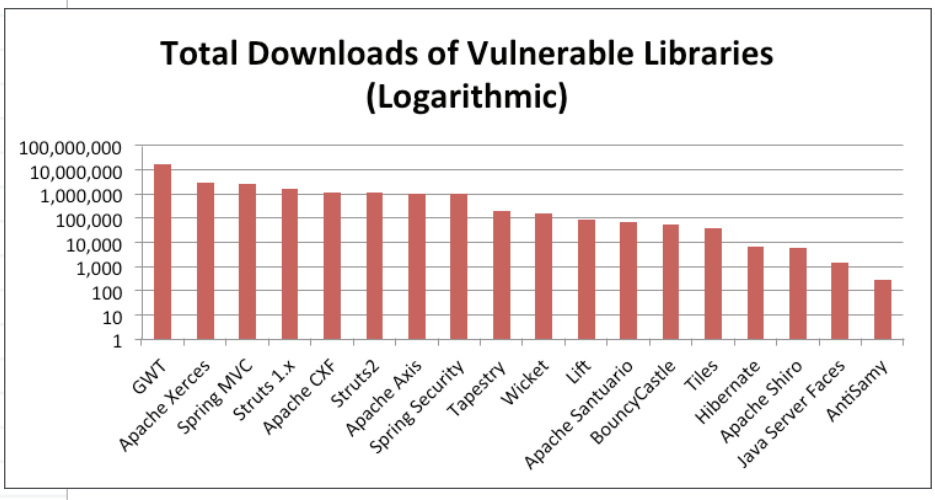
If organizations download old versions, it stands to reason that they also download vulnerable versions. In fact, 37% of the 1,261 versions of the 31 libraries studied contain a CVE or OSVDB vulnerability associated with their use. Further, 26% of the downloads of these libraries contained known CVE or OSVDB vulnerabilities.

In sum, over a year's time there were 29.8 million vulnerable jar files downloaded, out of a total of 113 million downloads. That magnitude of downloads means there are quite a lot of vulnerable applications in use. This data shows that security is not a major consideration in the determination of which library to download.

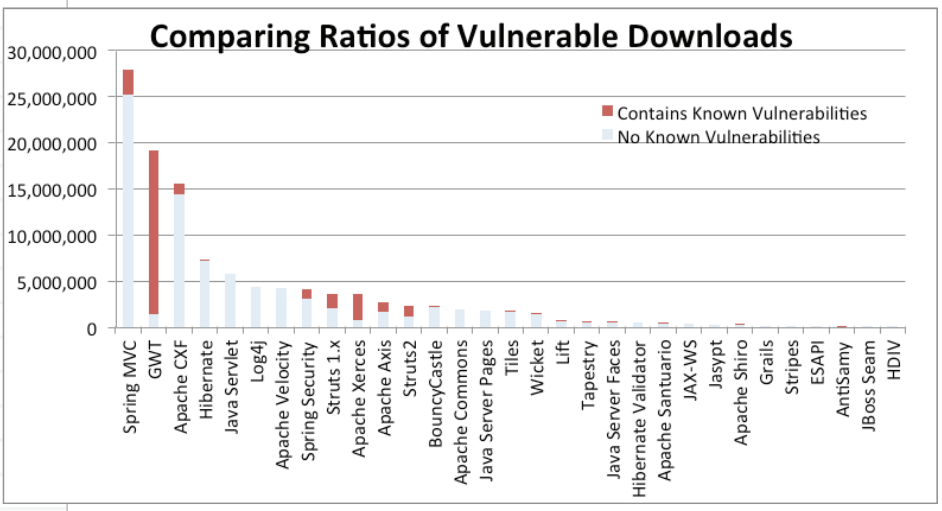



This number may be significantly underestimated because many developers get their libraries from local repositories (e.g. Sonatype's Nexus Repository Manager). These local repositories store, often in perpetuity, libraries within an organization and can mask an organization's ongoing use of these cached files from the dataset.

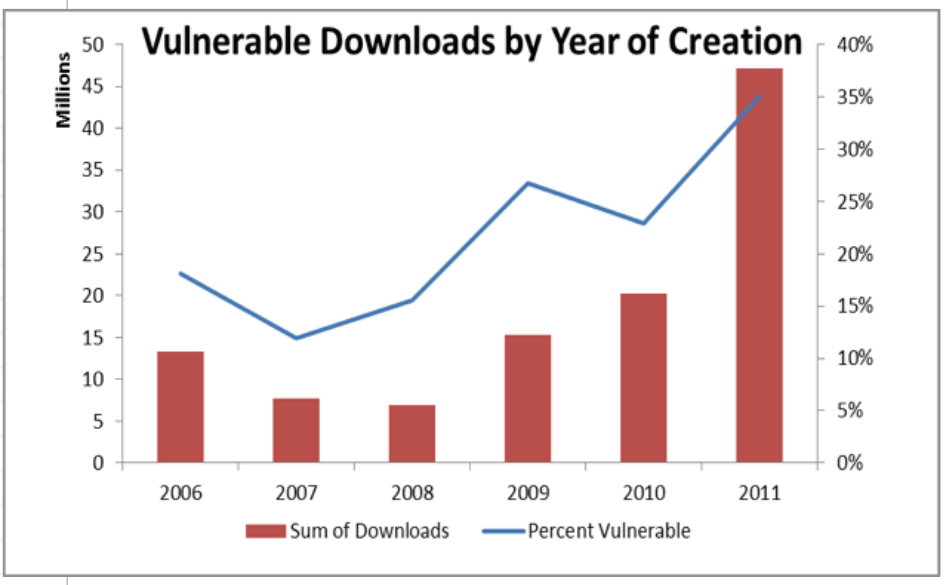
Examining the specific libraries showed that Google Web Toolkit (GWT) had more vulnerable downloads by an order of magnitude than any other library. GWT is a user-interface framework that allows Java web applications to create dynamic user interfaces that leverage JavaScript in the browser. GWT had a huge number of downloads and numerous vulnerable versions created in 2011.



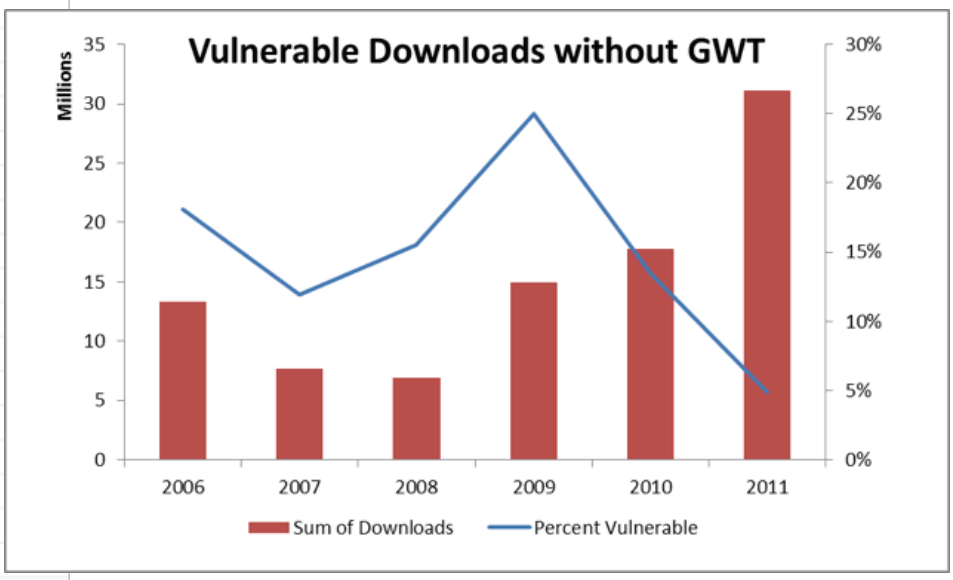

When shown along with downloads of versions without known vulnerabilities, the overall picture of the vulnerabilities becomes more clear. The overwhelming number of vulnerable GWT downloads is obvious.




The analysis of vulnerable downloads by the year the version was created shows a disturbing trend. Older libraries were expected to be likely more vulnerable downloads. But we found the opposite – newer libraries were considerably more likely to be a vulnerable download. In fact, from our dataset of the most popular libraries, 35% of the downloads of versions created in 2011 were vulnerable, compared to an average of 15% for libraries created in 2007.



The primary explanation for the trend is GWT. GWT represents about one-third of the downloads of libraries created in 2011 and 97% of the vulnerable ones. When GWT is removed from the data, the trend returns to what would have been expected. However, because there were so many downloads of vulnerable versions of GWT in 2011, it still means that there are a large number of vulnerable applications that use those versions.

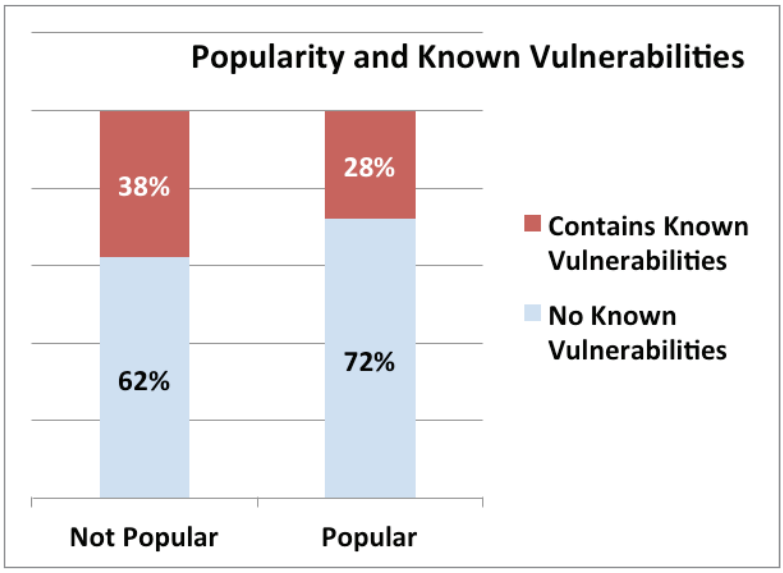



## Does Version Popularity Indicate Security?

In Central, the popularity of each version was measured against the others. For a given library, the versions were ranked on a scale of 0 to 100, with the most popular artifact receiving a score of 100 and others were scaled down appropriately.

The popular versions were 10% less likely to contain a known vulnerability. This is probably due to the fact that popular releases have typically been through several release cycles, allowing bugs to be identified and fixed.

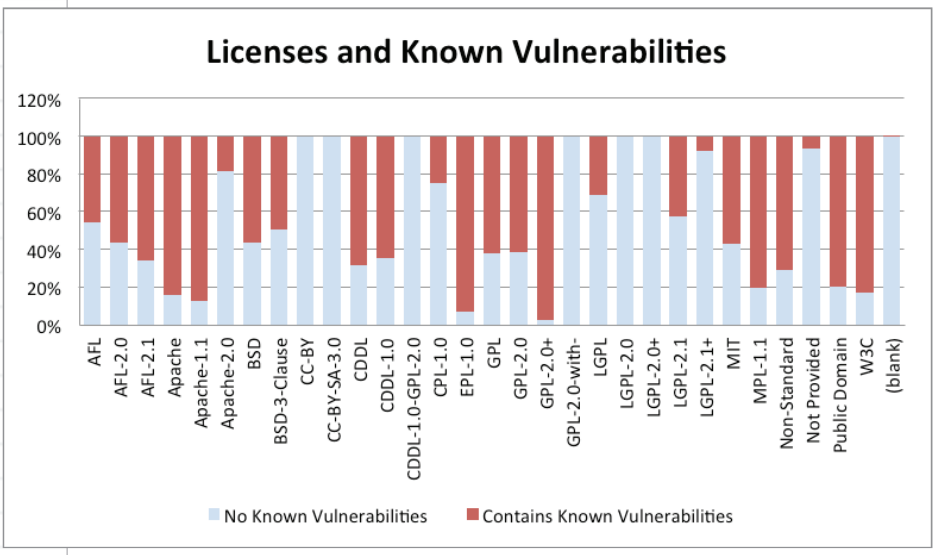
Applications typically include many different libraries, which significantly increases the likelihood that an application will contain a vulnerable library.




Applications typically include many different libraries, which significantly increases the likelihood that an application will contain a vulnerable library. Developers are playing a strange computer version of Russian roulette when including libraries in their application. Imagine an application that uses 10 libraries from the dataset of the most popular frameworks and security libraries. Even if developers select popular releases of all the libraries they need, there's still a 28% chance that each of them is vulnerable. Because our example application uses ten libraries, the chance that the application includes at least one vulnerable library is more than 95%.

## Does a Library's Open Source License Indicate Security?

Arguments have been made for decades about whether open-source code should be considered more or less secure than closed source. Although there are clearly differences in the rates of vulnerability for the different licenses, it would be incorrect to assume that the choice of license somehow correlates to the level of security consciousness of the project's developers. Specific terms in the licenses could provide greater encouragement for security researchers to analyze the code for vulnerabilities. However, it is more likely that security researchers simply focus on the most successful libraries and frameworks regardless of the license.




Organizations are recommended to make their commitment to security clear. A strong security culture in some organizations, such as BSD, may reduce the likelihood of vulnerabilities in their codebase. It's hard to read BSD's security page and not be encouraged about their approach. Their page starts with the quote below:

*"OpenBSD believes in strong security. Our aspiration is to be NUMBER ONE in the industry for security (if we are not already there). Our open software development model permits us to take a more uncompromising view towards increased security than Sun, SGI, IBM, HP, or other vendors are able to. We can make changes the vendors would not make."*

–http://www.openbsd.org/security.html

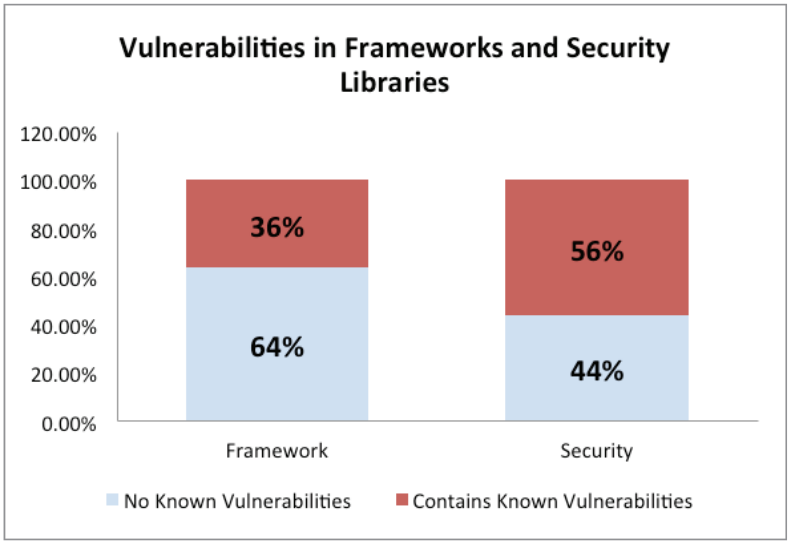
However, though their culture may encourage security, not all BSD licensed libraries come from OpenBSD. It is possible that some aspect of the license selected by project leaders does affect the likelihood of vulnerabilities being discovered. This will make an interesting research project for the future.

## Are Frameworks and Security Libraries Equally Vulnerable?

Vulnerabilities in security libraries are critical, as even the slightest vulnerability can cause significant exposure. The cryptographic community has established a thriving community of both "builders" and "breakers" that regularly advances the state of the art. Their ecosystem has a long record of creating strong security controls and improving them over time. All security controls should evolve in this type of ecosystem, but they don't.

To help fill this void, the OWASP ESAPI security library was created and leads this free and open project. The library is used by over 1,400 organizations and is one of the security libraries in this study. The ESAPI project invented new defenses for web applications like context-sensitive output encoders, canonicalization for web encodings, access reference maps, and realtime application intrusion detection. ESAPI has also been scrutinized by security researchers, received line-by-line code reviews from several very large organizations, and underwent a full review by the U.S. National Security Agency. This proactive approach has produced positive results for the OWASP ESAPI project. Creating a culture of "builders" and "breakers" in your own organization is highly recommended.

It is expected that security libraries would have fewer vulnerabilities on average than web frameworks because they are so critical and should have been written carefully with security in mind. However, security libraries are roughly 20% more likely to have a reported security vulnerability than a web framework. These libraries provide critical security functions and must be held to a higher level of assurance.



Security libraries may have more reported vulnerabilities because they naturally attract a greater degree of scrutiny by security researchers and attackers. This may explain the greater incidence of security vulnerability reports for these libraries. As mentioned, the dataset does not include the level of scrutiny targeting each library. It is also possible that because security libraries are specifically focused on security-critical code, virtually any bug is likely to be a reportable security vulnerability.

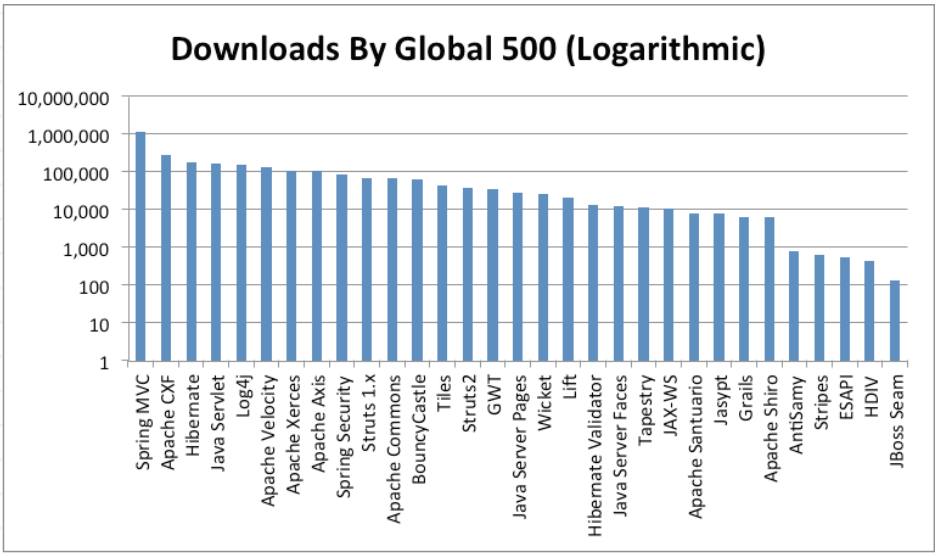
Despite the higher rate of vulnerabilities in security libraries, writing your own security controls isn't recommended. There are a huge number of subtle mistakes that can introduce vulnerabilities into controls written by the best developers. The best route to a secure set of controls for developers is to use proven and tested components and carefully have your implementation verified by security experts. You can improve your odds by externalizing and standardizing your controls.

The data reveals that almost half of the Global 500 use these libraries. Spring MVC was the overwhelming download leader with over 100 downloads per hour in a 12 month period. Of the security libraries, Log4j was the leader with over 152,000 downloads.

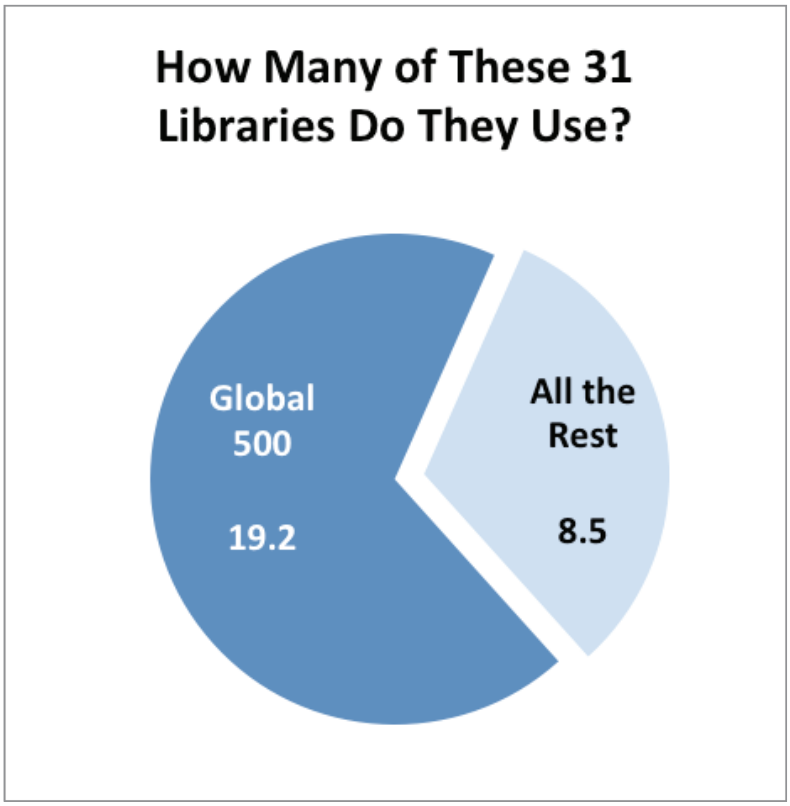
## Do the Global 500 Use Libraries Differently?

The data reveals that almost half of the Global 500 use these libraries. Spring MVC was the overwhelming download leader with over 100 downloads per hour in a 12 month period. Of the security libraries, Log4j was the leader with over 152,000 downloads.

These numbers are probably significantly underreported, as it is difficult to correctly associate all of the downloads with an organization (due to repository managing caching). Many downloads were attributed to an Internet provider, and slightly over 40% of the downloads could not be attributed to any specific organization. Many developers are downloading from home or other networks not associated with an organization. Many organizations not represented in the data are using Central and many have designed their networks to camouflage their identity when employees access the Internet.

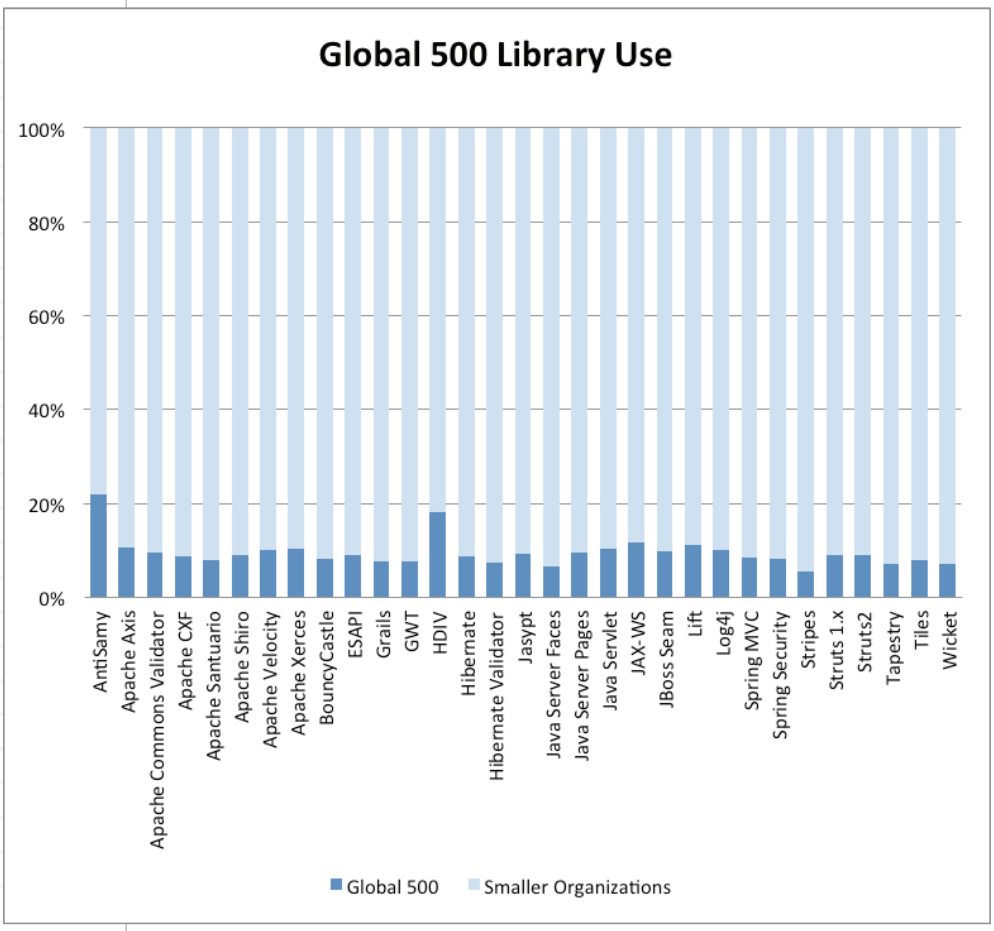


There does seem to be a significant difference in how many of the 31 libraries are used by the organizations studied. On average, the Global 500 downloaded 19.2 of the 31 libraries in the study. Smaller organizations downloaded an average of only 8.5 of the libraries studied. Since the larger organizations have considerably larger application portfolios, it is not surprising that they use more of the libraries. However, this is also a concern because there is considerable overlap in the libraries selected. That means that the larger organizations have not standardized on a small set of framework and security libraries. More libraries means more code. And more code increases the chance of a devastating vulnerability.




There does not seem to be a difference in the libraries used by the Global 500 and the smaller organizations in the study. Almost all of the libraries received between 7 and 10 percent of their downloads from the Global 500. The study had expected to find disproportionately high adoption of security libraries by the Global 500 as compared to all the other companies, but that hypothesis was not supported by the data.

Only two libraries, AntiSamy and HDIV, are disproportionately represented in the Global 500 compared to other libraries. AntiSamy, created by one of the authors of this article, received almost 22% of downloads from the Global 500. AntiSamy removes attacks from third-party HTML content to make it safe to use in webpages. HDIV adds integrity checks to HTTP input fields and received 17% of its downloads from the Global 500. Perhaps this evidence demonstrates that enterprise frameworks do not provide these niche security functions.




## Recommendations

Given the presence of vulnerabilities in commonly used versions of popular libraries in Central, it is strongly recommend that you take steps to minimize the risks to your organization. You should consider the risk from all libraries, including those with known vulnerabilities, unknown vulnerabilities, and malicious code. You trust your business to the libraries that you use.

**INVENTORY: Gather information about your current library situation**

Getting some real data about your organization's library use is a good way to get started. While broad studies like this are useful indicators, building momentum in an organization typically requires specific findings about your organization. Putting metrics around what libraries and frameworks are in use are recommended, showing how far out-of-date and out-of-version they are, the use of viral and unapproved licenses, and whether they have known vulnerabilities.

**ANALYZE: Check the project and the source for yourself**

Exercise a degree of restraint in the libraries that are used. Before trusting your enterprise to code of unknown provenance, we recommend a vetting process that gathers information about the team building the library and the process they followed to develop it. Minimally, open source projects should have an approved license, a process by which contributions are reviewed for security, and the ability to respond to security vulnerability reports. Consider the use of some recently available software tools to provide this capability.

The only way to deal with the risk of unknown vulnerabilities in libraries is to have someone who understands security analyze the source code. Static analysis of libraries is best thought of as providing hints where security vulnerabilities might be located in the code, not a replacement for experts. The lack of context with libraries makes it virtually impossible for tools to conclusively identify vulnerabilities. Manual code review can be used at various levels of rigor from the common flaws level, like the OWASP Top Ten, all the way up to searching for malicious code and rootkits. Be sure to discuss the level of rigor you require with reviewers.

**CONTROL: Restrict the use of unapproved libraries**

One way to gain control over ad-hoc library use is to establish a local repository that only contains approved libraries. A strict version of this policy could block direct access to Central by developers, although we have anecdotal evidence that this frequently results in "workarounds" where developers download jar files directly or access other open repositories. A better approach is to create a governance process around library use and help development groups take advantage of it.

Consider enabling the Java SecurityManager, sometimes known as the "sandbox." Java was originally designed with security features to allow remotely controlled code known as "applets" to run within the browser without compromising security. The Java SecurityManager was designed to keep this potentially malicious code in a "sandbox" where it could do no harm. When you consider that libraries are also remotely controlled code, the solution seems clear. The SecurityManager can prevent libraries from making dangerous calls and make many rootkits impossible.

Also consider creating a "Secure Use" guideline that details how frameworks and libraries are allowed to be used in your organization. The guideline should specifically detail the patterns for using a library securely and point out any known patterns that could lead to an insecure application.

**MONITOR: Keep libraries up-to-date**

Development teams should plan for and allocate resources to keep libraries up-to-date. In most cases, updates can be made compatible with an existing application without significant rework. Note that major releases may require significant rework. This is the cost that you incur as result of using the library in the first place, rather than investing in writing your own code. Establish systems and processes for monitoring the libraries that you are using. This will help you to identify and respond to security vulnerabilities and updates.

## Conclusions

The use of libraries has become a pervasive, almost overwhelming aspect, of modern software development. In the past few years, the use of dependency management tools has caused a significant increase in the number of libraries involved in a typical application. In this paper, we examined some of the security implications of this change and conclude that there are significant risks associated with the use of libraries. We recommend that any organization building critical software applications protect itself against these risks by taking steps to inventory, analyze, control, and monitor the use of libraries across the organization.

## About Contrast Security

Founded in 2012, Contrast Security's products empower security professionals to protect any organization from application security risks. Contrast enables applications to continuously test themselves in real-time for security vulnerabilities without experts, scans or attacks. Contrast installs and delivers actionable results in minutes, eliminating the time and money spent planning, scheduling, executing and analyzing legacy security tests and supercharging application delivery to DevOps speed and portfolio scale. Patented Behavioral Analysis Technology expertly pinpoints risky behaviors in running applications under normal use and provides remediation direction before or after deployment. For more information, please visit www.contrastsecurity.com.

## About the Authors

**Jeff Williams** brings more than 20 years of security leadership experience as Co-Founder and Chief Technology Officer of Contrast Security. In 2002, Jeff co-founded and became CEO of Aspect Security, a successful and innovative consulting company focused on application security. Jeff is also a founder and major contributor to OWASP, where he served as the Chair of the OWASP Board for 8 years and created the OWASP Top 10, OWASP Enterprise Security API, OWASP Application Security Verification Standard, XSS Prevention Cheat Sheet, and many other widely adopted free and open projects. Jeff has a BA from Virginia, an MA from George Mason, and a JD from Georgetown. You can contact him at jeff.williams@contrastsecurity.com.

**Arshan Dabirsiaghi** is Co-Founder and Chief Scientist at Contrast Security. He is an accomplished security researcher with 10+ years of experience advising large organizations about application security. Arshan has released popular application security tools, including AntiSamy and JavaSnoop. As the Director of Research at Aspect Security, Arshan specialized in advanced Web security including Web 2.0 security, rich input validation, and instrumentation. Arshan is a frequent speaker at major application security conferences including BlackHat and OWASP. Arshan is also the author of the OWASP AntiSamy, OWASP ESAPI WAF, and JavaSnoop security projects. You can reach him at arshan.dabirsiaghi@contrastsecurity.com.

## Summary

Contrast Security provides simple, scalable and accurate applications security–no experts or testing required.


## Get Started Today!

You can sign up and get started with Contrast in less than 5 minutes. Just visit: www.contrastsecurity.com

For more information, please contact one of our experts today by calling 1.855.365.APPS or email us at: sales@contrastsecurity.com


**Contrast Security**
9175 Guilford Road, Suite 300
Columbia, MD 21046
www.contrastsecurity.com