VOLUME 5

State of Software Security Report

The Intractable Problem of Insecure Software

APRIL 2013

Read Our Predictions for 2013 and Beyond

VERACODE

Dear SoSS Report Reader,

As some of you may know I have spent most of my 25 year career in the IT Security industry, more specifically, I've been focused on application security as the use of web and mobile applications has flourished. For the past five years I have been an active participant in the preparation of the report before you today—our annual State of Software Security Report, or as we fondly refer to it at Veracode, the SoSS Report.

Throughout my career I have been evangelizing the need for more secure application development practices, and with the release of each new SoSS report I find myself of two minds. The optimist in me is proud of the vast improvement in general awareness of the importance of securing the application layer. But the pessimist remains very concerned that we are not seeing the dramatic decreases in exploitable coding flaws that I expect to see with each passing year. It's as if for each customer, development team, or application that has become more secure, there are an equal number or more that do not.

While the benefits of web applications are clear to organizations, the risks to their brands, infrastructure, and their data are seemingly not as clear, despite being more apparent than ever. It's at this point of my letter that I could mention that a cyber-Vesuvius is about to bubble over and create a cyber-Pompeii as there are so many breaches reported; but I'll resist that temptation. Instead, here are a few links to recently released reports that do a shockingly good job of telling the scary story:

- 2013 Trustwave Global Security Report[1]
- 2012 Verizon Data Breach Investigations Report[2]

I only cite these examples because the reports illustrate the *"after"* scenario, evaluating what has happened when vulnerable systems are exposed to the threat space. We at Veracode see the SoSS report as different, using data to shine light on what is to come by understanding the latent vulnerabilities in software organizations are deploying. The *"before"* scenario means our SoSS reports have become great predictors about future data breaches. For example, this report shows 32% of applications analyzed by Veracode contain SQL injection flaws. Knowing that, you should not be surprised that Trustwave reported that SQL injection was the attack method for 26% of all reported breaches in 2012. I can tell you with confidence that malicious actors target the flaws that are easy to find and exploit—like SQL injection—therefore the instances of SQL injection attacks will surely increase in 2013. Put more bluntly, we must figure out a way to code more securely simply to keep up with attacks from the most basic attacker.

As you read this report I urge you to consider your organization's application portfolio and how you currently make decisions about the risks your organization is willing to take. The amount of risk an organization takes should be a strategic business decision—not the aftermath of a particular development project. If you're learning about risks after a breach—be it yours or an industry counterpart—then the time to act is now. Use this SoSS report to estimate your current application risk landscape—particularly on applications that you have never tested or only tested manually. Consider how you can act now to improve the security posture of your organization, by addressing the applications that you currently have in development and/or in production. Hopefully by the time we release SoSS V6 in 2014, we'll see that dramatic improvement I've been waiting for!

I hope you enjoy the report.

Chris Wysopal
Co-Founder, CISO and CTO, Veracode

[1] www2.trustwave.com/rs/trustwave/images/2013-Global-Security-Report.pdf

[2] www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf

Table of Contents

Introduction

For the past five years, the Veracode State of Software Security (SoSS) report has examined trends associated with vulnerabilities in applications. Our initial goal was to provide key insights to those charged with managing enterprise application security risk, to give them a series of benchmarks from which they could measure their own application security posture. After five years and five versions of SoSS our goal now is to highlight the slow progress in securing the application layer. Since insecure applications are a leading cause of security breaches and data loss for organizations of all types and sizes, we can't continue to whistle past the graveyard. We want the readers of this report to leverage the data to build a business case for an application security program at their organization.

As with past SoSS reports, this analysis draws on continuously updated information in Veracode's cloud-based application security platform. Unlike a survey, the data comes from actual application security assessments conducted to identify vulnerabilities and validate remediations. SoSS Volume 5 examines data collected over an 18 month period from January 2011 through June 2012 from 22,430 application builds uploaded and assessed by our platform (compared to 9,910 application builds analyzed in Volume 4, which was published in December 2011).

This report examines application security quality, remediation, and policy compliance statistics and trends. The data analyzed represents multiple security testing methodologies (including static binary, dynamic and manual) on a wide range of application types (web, mobile and non-web) and programming languages (including Java, C/C++, .NET, PHP and ColdFusion). We also expanded our analysis of the mobile vulnerability landscape with sections on Android, iOS and Blackberry applications. The resulting intelligence is unique in the breadth and depth it offers.

Readers of Volume 5 will notice an increased focus on vulnerability distribution trends for each language. Also, new to this Volume is an analysis of the percentage improvement in the vulnerability distribution between the first and second application builds. This metric should provide some perspective on which vulnerabilities organizations chose to fix upon receiving the results from their first submission. These new visualizations and analysis are in response to customer questions about demonstrating the impact of security programs on enterprise risk profiles.

Veracode's data analytics team is always looking for new perspectives on security metrics. We welcome reader questions, comments and ideas so that we can continually improve and enrich the coverage, quality and detail of our analysis.

Executive Summary

The following are some significant findings in the Veracode State of Software Security Report Volume 5. Each finding is accompanied by a prediction for the next 12 to 18 months, where we sketch out the possible futures if the status quo continues. We also provide recommendations for altering our predicted trajectory, because we can change the future.

Key Findings

70% of applications failed to comply with enterprise security policies on first submission.
This represents a significant increase in the failure rate of 60% reported in Volume 4. While the applications may eventually become compliant, the high initial failure rate validates the concerns CISOs have regarding application security risks since insecure applications are a leading cause of security breaches and data loss for organizations of all types and sizes.

Prediction: Average CISO Tenure Continues to Decline. The average tenure of a CISO is 18 months, and more CISO jobs will be at risk given the current state of software security. The expansive threat profile associated with software means the likelihood of CISOs being negatively affected by a high-impact security event has never been greater.

Recommendation: Driving up compliance with enterprise application security policies lowers the risk of high-impact events. To accomplish this, CISOs and security professionals must work closely with their counterparts in Development and Procurement to set security policies and enable internal and external developers to consistently comply with those policies.

SQL injection prevalence has plateaued, affecting approximately 32% of web applications.
The downward trend in SQL injection that we reported in Volumes 3 and 4 has flattened. For six consecutive quarters, from the first quarter of 2011 to the second quarter of 2012, the percentage of applications affected by SQL injection has hovered around 32%. This should be a concern, as three of the biggest SQL injection attacks in 2012 resulted in millions of email addresses, user names, and passwords being exposed and damaged the respective brands.

Prediction: The Rise of the Everyday Hacker. Once the sole domain of technical experts, now a simple search for "SQL Injection Tutorial" enables anyone to exploit a serious vulnerability and wreak havoc. The data shows that everyday hackers are on the rise, as Trustwave reported SQL injection to be the attack method for 26% of all reported breaches in 2012. We predict that number to exceed 30% in 2013. SQL injection vulnerabilities are just too easy to find and exploit.

Recommendation: Organizations should institute zero-tolerance policies for SQL injection vulnerabilities and employ routine monitoring to detect vulnerabilities as new applications are deployed.

Eradicating SQL injection in web applications remains a challenge as organizations make tradeoffs around what to remediate first.

The percentage of applications affected by SQL injection has hovered around 32% and cross-site scripting around 67% for the last six quarters. For the first time we are reporting improvement percentages by language to illustrate which flaws organizations are choosing to fix after receiving results from their first submission. Java, representing 56% of web applications, showed 16% improvement in SQL injection and 14% improvement in cross-site scripting between the first and second submission. .NET, representing 28% of web applications, showed a 25% improvement in SQL injection and 15% improvement in cross-site scripting.

Prediction: Decreased Job Satisfaction/Higher Turn-over for Security Professionals. The challenge is daunting. Companies face a seemingly ever-expanding threat profile brought on by new applications and application updates containing easy to exploit flaws such as SQL injection (26% of all 2012 reported breaches according to Trustwave). This can create a very frustrating work environment for security pros. The desire to find roles where their efforts will bear more fruit and where success is apparent will drive increased turnover among security pros. There is some good news, however. According to the Bureau of Labor Statistics,[3] the employment segment that includes information security analysts is projected to grow 22% between 2010 and 2020, faster than the average for all occupations.

Recommendation: Making a difference as a security professional often means building relationships with development executives. Instead of taking a "scan and scold" approach, the program goal should be improving overall developer productivity by efficiently integrating security remediation into existing development methodologies. Getting development executives focused on process integration, knowledge transfer, remediation support and incentives for secure code creation as key success criteria would represent a significant breakthrough in the relationship.

Cryptographic issues affect a sizeable portion of Android (64%) and iOS (58%) applications.

Using cryptographic mechanisms incorrectly can make it easier for attackers to compromise the application. For example, cryptographic keys can be used to protect transmitted or stored data. However, practices such as hard-coding a cryptographic key directly into a mobile application can be problematic. Should these keys be compromised, any security mechanisms that depend on the privacy of the keys are rendered ineffective.

Prediction: Default Encryption, Not "Opt-in," Will Become the Norm. Eavesdropping on mobile communications can make it easier for attackers to design successful social engineering attacks against key employees. There is a staggering amount of transmitted data at risk, considering the growth of open (i.e. easy to eavesdrop) Wi-Fi networks in combination with the number of social network users (Facebook 1.2B; Twitter 190M tweets/day) and the number of mobile devices (Cisco[4] predicts that by the end of 2013, the number of mobile devices will exceed the number of people on earth—7.1B). These concerns have prompted companies like Twitter and Facebook to encrypt all traffic by default, despite the additional computing power required to encrypt every connection. As more business is conducted through applications resident on personal mobile devices, we expect enterprises to insist on mobile applications that force encryption to protect data in motion.

Recommendation: Developers and security professionals should expect data encryption to be involved in all aspects of designing the business user's experience with mobile applications. From a data in motion perspective, this would include understanding the performance impact and incremental infrastructure costs of encrypting traffic between the mobile application and the server side application. From a data at rest perspective, additional attention should be paid to the cryptographic techniques used to protect the application itself from unintended data disclosure.

[3] www.bls.gov/ooh/computer-and-information-technology/information-security-analysts-web-developers-and-computer-network-architects.htm

[4] www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

Security of Applications

The evidence linking organizational intrusions and data breach events to application security issues continues to grow. Web-based intrusions and hacking in general account for 52% of the breaches in 2011 and 2012 tracked by Open Security Foundation's DataLossDB (Figure 1).

While these categories are extremely broad, hacking and web-based intrusions often involve exploiting software vulnerabilities. Reports published by companies that conduct actual breach investigations provide additional insight. The Verizon Data Breach Report, released in March 2012, indicated that 81% of attacks utilized some sort of hacking. This section explores application compliance with standard policies, remediation submission rates and security quality scores to shed some light on why this connection exists.

Data Loss Breaches Categorized by Root Cause

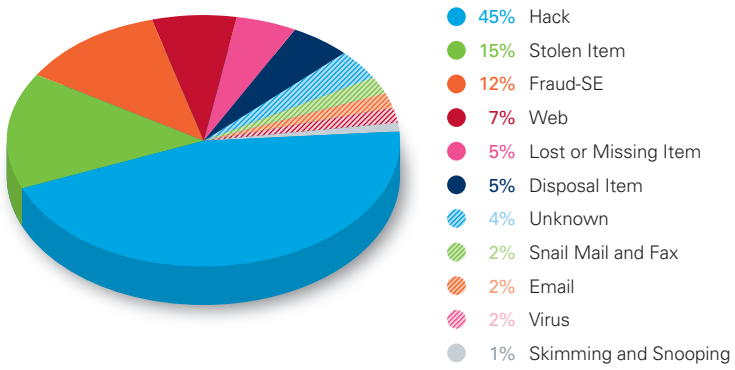

Figure 1: Data Loss Breaches Categorized by Root Cause (Source: DataLossDB)

Compliance with Standard Policies Upon First Submission

Figure 2 illustrates the compliance upon initial application submission against two standard policies.[5] Web applications are assessed against the OWASP Top 10 and only 13% complied on first submission. Non-web applications are assessed against the CWE/SANS Top 25 and 31% complied on first submission. Only 30% of applications complied with enterprise defined policies. Compliance with policies upon first submission of an application can be a good indicator of the success or failure of "building-in" security as part of the software development lifecycle (SDLC).

[5] More details about how the Veracode platform determines policy compliance can be found in the Appendix.

Because security flaws that are eliminated before deployment, or never created in the first place, are much less expensive to remediate, thus building remediation into the SDLC at an early stage is often a key goal for most organizations. Yet, with more than two thirds of the applications failing to comply, our results show that secure software development practices are still not as widespread as they should be. While applications may eventually become compliant, the high initial failure rate validates the concerns CISOs have regarding the business risks related to application security.

Compliance with Policies Upon First Submission

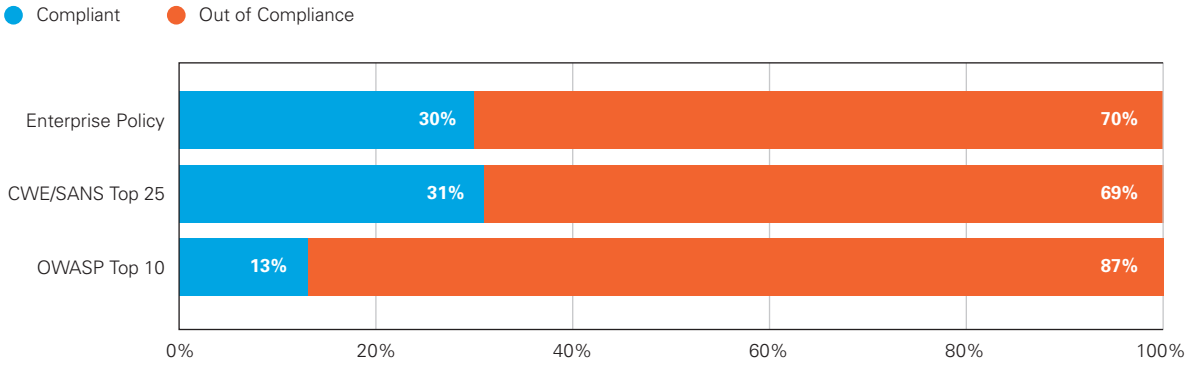

Figure 2: Compliance with Policies Upon First Submission

The OWASP Top 10 compliance rate did not change significantly from Volume 4 (14%). In contrast, the percentage of applications passing enterprise policies declined significantly from Volume 4 (40%). Similarly, the percentages of non-web applications that complied with SANS/CWE policy were respectively 42% and 31% in Volumes 4 and 5, which is highly statistically significant decrease. We decided to investigate whether language or supplier types where potential drivers of the decrease in SANS/CWE policy compliance. Our analysis[6] suggests that the major contributors to this Vol4 to Vol5 decrease in compliance rate were as follows:

- There is evidence that Language (99.9% confidence level) influences CWE/SANS compliance with C/C++ being the most significant factor. This means that C/C++ applications, which represent 29% of non-web applications in our dataset, had a significant impact on driving down the CWE/SANS compliance rate from Volume 4.
- There is no compelling evidence that software supplier types (internally developed, commercial, outsourced, and open source) influence CWE/SANS compliance.

Another possible factor in the decrease of SANS/CWE policy compliance could be the increase in the number of first submissions in our sample set. The Volume 5 data had 75% more first builds than Volume 4. This increase in first builds suggests more broad use of the service by a wider variety of companies, perhaps with higher variation in secure software development practices.

[6] The analysis that we performed used a generalized linear model to perform logistic regression on a proportional response variable (SANS Compliance) with categorical explanatory variables (Volume, Flaw Category, Industry, Supplier, and Language). See Appendix for additional detail.

Remediation Analysis

We frequently get questions from customers and analysts on whether discovered vulnerabilities are actually remediated and whether those remediations are validated through additional testing. To shed some light on this issue, we start by examining how frequently organizations resubmit applications following the initial analysis. These resubmitted applications typically contain a combination of security remediations for previously reported vulnerabilities. Resubmitted applications may also contain new or altered code components to address non-security issues and new code components representing new functionality.

One might expect that more companies would resubmit higher percentages of their very high criticality applications than they would their medium criticality applications. If this expectation were true then one would anticipate the distribution pattern for medium and very high criticality applications to look very different (possibly a classic bell curve for the medium criticality applications and an exponential curve for the very high criticality applications). At the very least, one might expect variability in resubmission rate to decrease as application criticality increases. The data does not support those expectations. Figure 3 shows statistically insignificant differences in the distribution patterns. Roughly 45% of organizations resubmit 91-100% of their applications regardless of the business criticality. In Volume 4 we reported that the very high group was slightly different from the high and medium groups, since over 50% of companies resubmitting 91-100% of their very high criticality applications, however that slight difference has disappeared in Volume 5.

Percentage of Applications Resubmitted by Business Criticality

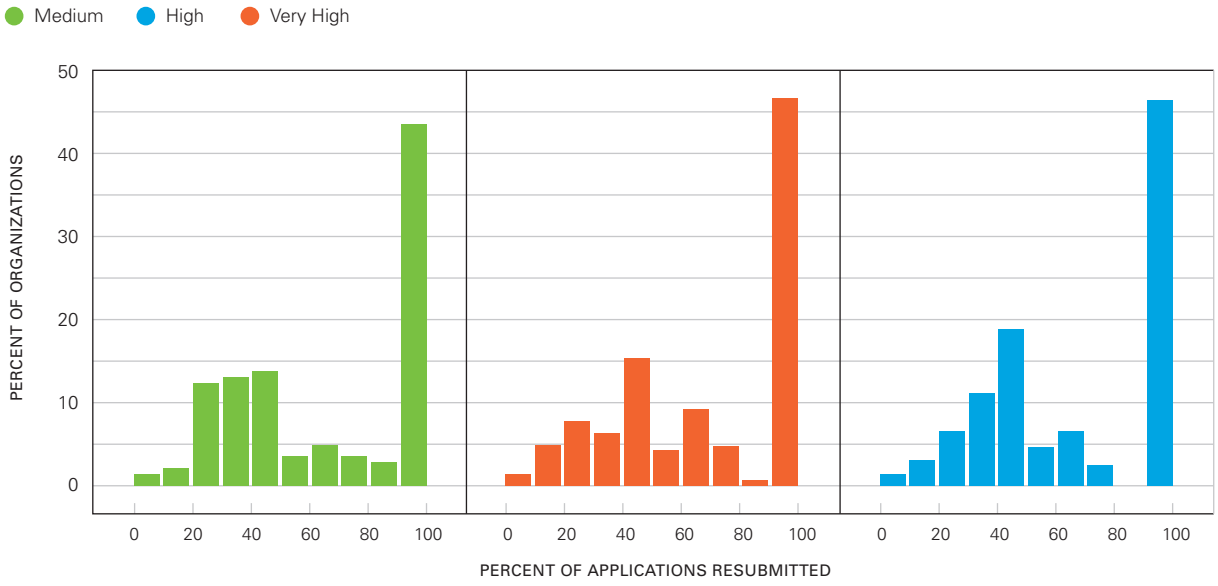

Figure 3: Percentage of Applications Resubmitted by Business Criticality

Security Quality Analysis

We continue to track the quarterly mean Veracode Security Quality Score (SQS) as a means of determining when security quality becomes a standard part of developing software. We expect that when most organizations have built security into their SDLC we will begin to see an upward trend in SQS developing. An upward trend would indicate that applications from new Veracode customers and newly developed applications from existing customers are a less significant force in dragging down the mean with very low scores. Figure 4 shows we still have a lot of work to do in building in security. The best fit line across our analysis timeline has a p-value[7] of 0.37 indicating that the trend is flat. This flat trend is consistent with the trends reported in Volumes 3 and 4—there has been no increase or decrease in the quarterly mean SQS since the fourth quarter of 2009.

Veracode Security Quality Score Trend

p-value = 0.37

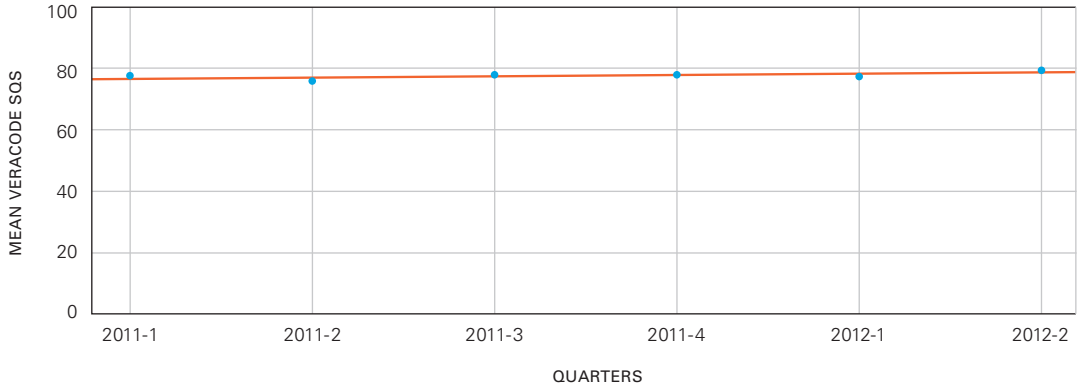

Figure 4: Veracode Security Quality Score Trend

Next we examine the progress an application makes build-over-build as the developers respond to findings and attempt to remediate flaws using the median value of the Veracode Security Quality Score (SQS) as a progress indicator. The distribution of the Veracode Security Quality Score by application build is shown as a whisker plot[8] in Figure 5. The data shows statistically significant build-over-build improvement from the first to third builds. Builds four through six remain statistically flat, followed by a marked improvement in builds seven and eight. The median score decreased in build nine, however, it is still above the plateau of builds four through six. This pattern suggests the security quality in applications with nine or more builds has been permanently improved even as functionality in the form of new code is being added in the later builds.

[7] See Appendix for definition.

[8] See Appendix for definition.

Veracode Security Quality Score by Build

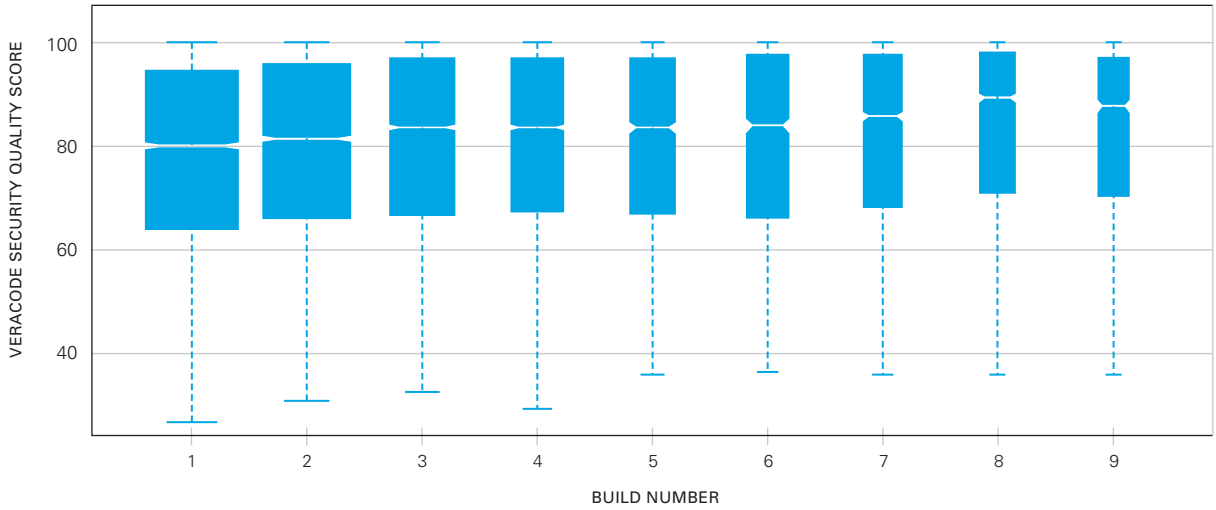

Figure 5: Veracode Security Quality Score by Build

The pattern of statistically significant improvement in security quality scores for builds one, two and three seen in Figure 5 is consistent with the figures reported in Volumes 3. However, there are significant differences between Volumes 4 and 5 in the patterns reported for later builds. In Volume 4 we saw an oscillating behavior with peaks occurring at builds four, seven and nine. The Volume 4 pattern suggested that new functionality was introduced in the build after each peak, which resulted in a new set of security flaws found and the consequently lower score. It is not immediately clear what has caused this shift in pattern between Volumes 4 and 5. It could be that our dataset for later application builds is richer in Volume 5 and therefore more representative of the actual improvement pattern. It is also possible that developers are starting to introduce new code that does not suffer from the vulnerabilities in the old code. The developers have learned from the mistakes and do not repeat them.

Language Analysis

In this section we dive deeper into each language. For each language we look at the distribution of each vulnerability category. We measure vulnerability distribution in terms of share of vulnerabilities found in each language group.

We calculate this by first filtering our data by language. For each language we determine the total number of vulnerabilities found and the number vulnerabilities that belong to a specific category. These values allow us to calculate the percentage share for each vulnerability category for that language. These vulnerability distribution calculations allow us to make statements such as, 3% of vulnerabilities found in Java applications are SQL injection vulnerabilities (Figure 6). The vulnerability distribution metrics also give us a historical perspective, since we have been reporting them since Volume 3.

Another metric we explore is the vulnerability prevalence in terms of the percentage of applications affected by each vulnerability category. To calculate this metric, we also begin by filtering our data by language. Then we identify how many applications contain one or more vulnerabilities from each category, which allows us to calculate the percentage affected. These calculations enable us to make statements such as: SQL injection vulnerabilities affect 31% of Java applications (Figure 7).

Vulnerability distribution and prevalence information can be useful for planning purposes, particularly when internal and/or industry-specific benchmarks[9] are not readily available. Organizations can estimate the resource impact of implementing or changing application security policies. Consider the situation of a security team writing a policy aimed at eliminating SQL injection flaws and a development team writing their application in Java. The percentage affected data tells the teams there is a 31% chance that their application will have SQL injection flaw. The vulnerability prevalence data means that if the application does have SQL injection, it is likely that only 3% of the vulnerabilities found will be SQL injection.

Finally, we investigate the percentage improvement in vulnerability distribution between an application's first and second build. This metric should provide some perspective on which vulnerabilities organizations chose to fix upon receiving the results from their first submission. For each language, we looked at the subset of applications with their first and second builds occurring within the analysis timeframe for this report. This means we excluded applications with their first build occurring before January 2011 and applications with their second build occurring after June 2012. We also excluded applications with components written in more than one language. Then we calculated the change in vulnerability distribution from the first build to the second build. The percentage change will be affected by the volume of flaws. For example, consider the case of a development team that has fixed 10 flaws between the first and second build. If there were 20 flaws in the first build then the calculation would show a 50% improvement. However, if the first build contained 100 flaws, then the calculation would show a 10% improvement. To acknowledge this impact we indicate the top vulnerability categories in the percentage improvement charts. The percentage change may also be affected by improvements to the Veracode platform, and we'll discuss those improvements where applicable.

[9] The Veracode Analytics capabilities enable organizations to benchmark their internal application security metrics with industry benchmarks.

Java

Figure 6 shows that vulnerability distribution in Java applications has not significantly changed since Volume 3. The cross-site scripting category consistently represents more than half of all vulnerabilities discovered in Java applications. In the Volume 5 dataset, SQL injection makes its first appearance in the top 5 list at fifth place, replacing cryptographic issues. Figure 7 shows code quality, CRLF injection and information leakage affecting the most applications with 82%, 68% and 58% respectively.

Vulnerability Distribution Trends for Java Applications (Share of Total Vulnerabilities Found)

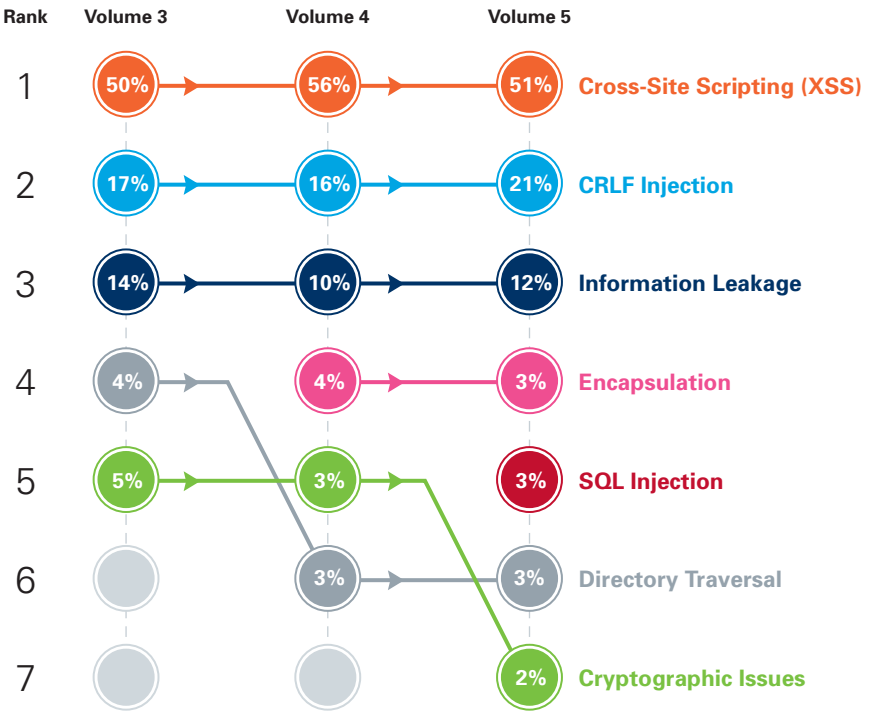

Figure 6: Vulnerability Distribution Trends for Java Applications (Share of Total Vulnerabilities Found)

Vulnerability Prevalence in Java Applications (Percentage of Applications Affected)

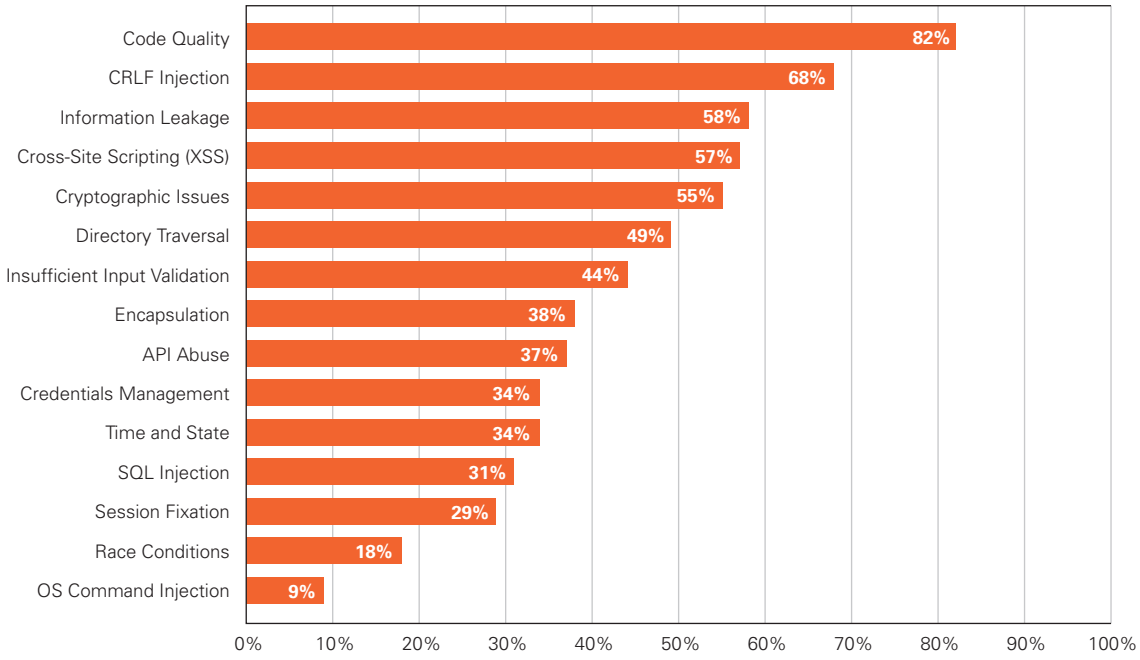

Figure 7: Vulnerability Prevalence in Java Applications (Percentage of Applications Affected)

Figure 8 indicates the untrusted search path category had the highest improvement percentage from first to second application build. Although this vulnerability category does not occur very often (it is absent from Figure 6 and Figure 7) it contains some very high severity flaws. For example, CWE-114 is defined as executing commands or loading libraries from an untrusted source, or in an untrusted environment, can cause an application to execute malicious commands (and payloads) on behalf of an attacker.[10] Figure 8 also shows an improvement percentage of 45% for CRLF injection, which holds the second place in both Java vulnerability distribution (21%) and prevalence (68%).

CRLF injection, which holds the second place in both Java vulnerability distribution (21%) and prevalence (68%), showed an improvement percentage of 45% from first to second submission.

[10] For the complete description see cwe.mitre.org/data/definitions/114.html

Percent Improvement in Java Vulnerability Distribution from First to Second Submission

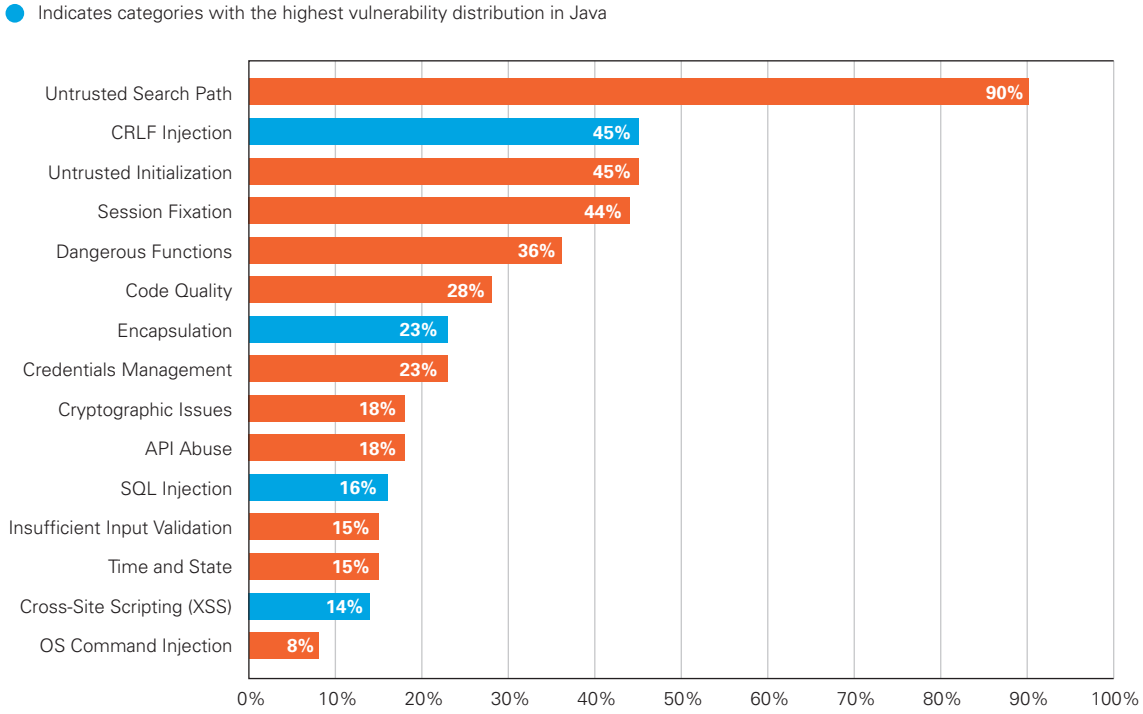

Figure 8: Percent Improvement in Java Vulnerability Distribution from First to Second Submission

.NET

The vulnerability distribution for .NET applications has not changed significantly over the last three Volumes (Figure 9). Cross-site scripting (XSS) retains the highest share of vulnerabilities at 49%. However, the percentages have been changing over time. Cross-site scripting and directory traversal categories have been slowly increasing while information leakage and cryptographic issues have been slowly decreasing.

Cross-site scripting and SQL injection showed improvement from first to second build in terms of share of vulnerabilities discovered, but still affect 60% and 30% of all .NET applications respectively.

In addition, 61% of .NET applications contain one or more XSS vulnerabilities (Figure 10). The high percentages in both metrics indicate that cross-site scripting is a pervasive vulnerability, i.e. it occurs many times in many applications. Figure 11 appears to indicate a fairly low percentage improvement (15%) between the first and second build for XSS. When taken together, these three data points demonstrate the enormity of the task of removing cross-site scripting from existing applications, because there are so many vulnerabilities to remediate.

Significantly, the top five categories that showed the most improvement are comprised of less than 10% of all discovered flaws and affect at most 50% of all .NET applications (Figure 11). If you leave out SQL injection, the top four categories that showed improvement comprise at most 20% of all .NET applications. Cross-site scripting and SQL injection showed improvement in terms of share of vulnerabilities discovered, but still affect 60% and 30% of all .NET applications respectively.

Vulnerability Distribution Trends for .NET Applications (Share of Total Vulnerabilities Found)

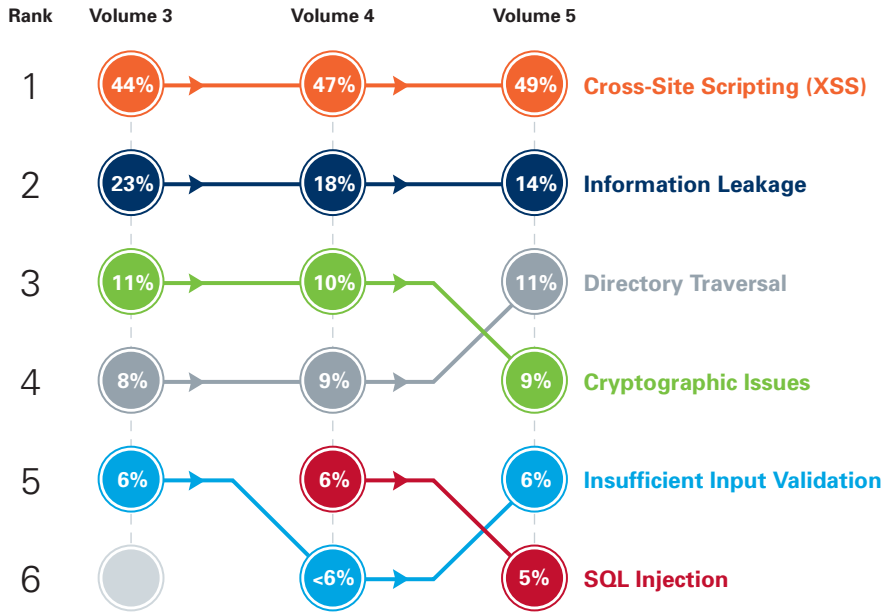

Figure 9: Vulnerability Distribution Trends for .NET Applications (Share of Total Vulnerabilities Found)

Vulnerability Prevalence in .NET Applications (Percentage of Applications Affected)

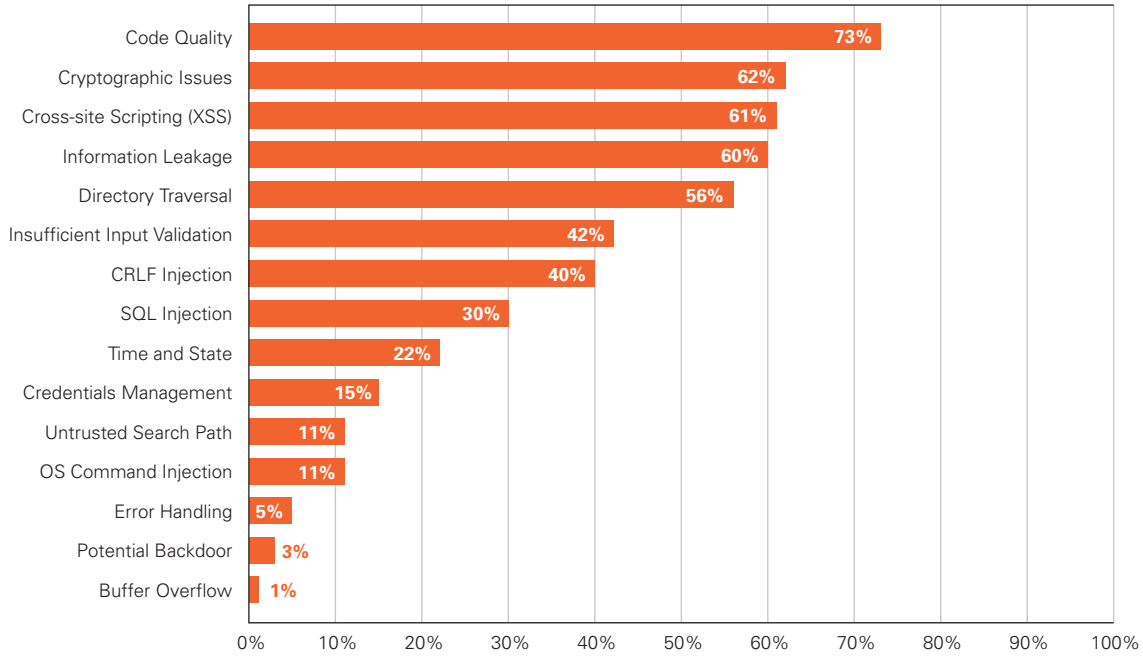

Figure 10: Vulnerability Prevalence in .NET Applications (Percentage of Applications Affected)

Percent Improvement in .NET Vulnerability Distribution from First to Second Submission

● Indicates categories with the highest vulnerability distribution in .NET

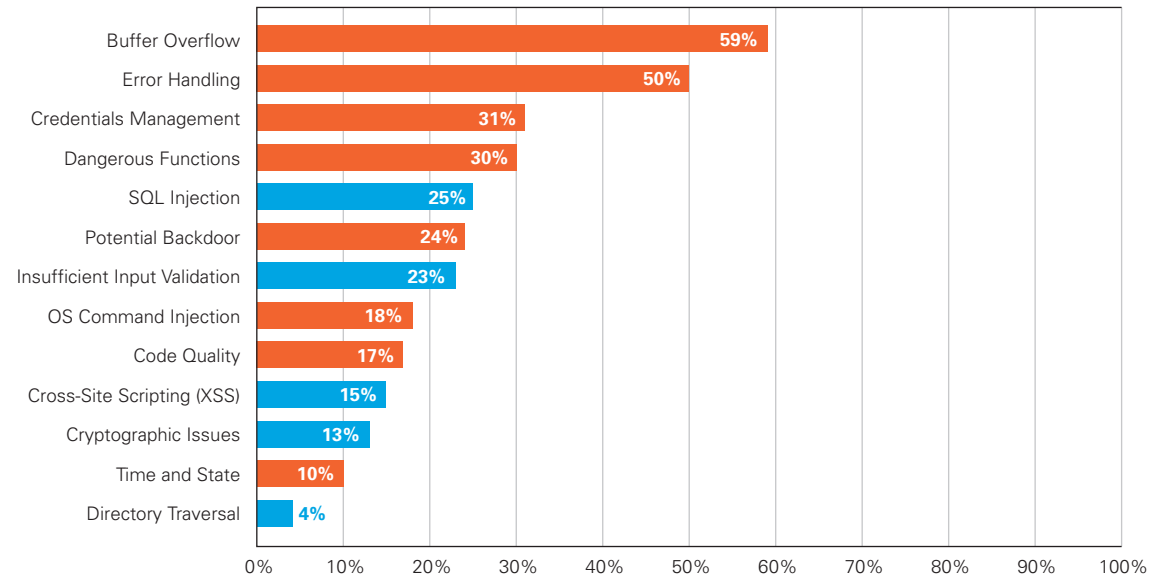

Figure 11: Percent Improvement in .NET Vulnerability Distribution from First to Second Submission

C/C++

Vulnerability distribution for C/C++ applications has changed significantly over the past three Volumes (Figure 12). The buffer management errors category has risen from fifth place in Volume 3 to second place in Volume 5. Buffer overflow issues have fallen to third place and cryptographic issues appear for the first time in fifth place. While error handling issues retain the top spot from Volume 4, the percentage share has increased from 23% to 38%.

Vulnerability distribution for C/C++ applications has changed significantly over the past three Volumes, with error handling and buffer management errors rising to the top of both the vulnerability distribution and prevalence lists.

The vulnerability categories with the highest distribution also affect the highest percentage of applications (Figure 13), from error handling affecting 77%, to numeric errors affecting 44%. One implication of this data for organizations scanning C/C++ applications for the first time is that it is likely several of these vulnerabilities will be listed in the assessment report.

Although Figure 13 shows API abuse affecting relatively few applications (7%), Figure 14 indicates a significant percentage improvement from first to second submission in API abuse (85%). This is an example of how targeting a few flaws (such as CWE 243: Creation of chroot Jail Without Changing Working Directory, which may allow unauthorized access to data files) can significantly improve an application's security posture.

Vulnerability Distribution Trends for C/C++ Applications (Share of Total Vulnerabilities Found)

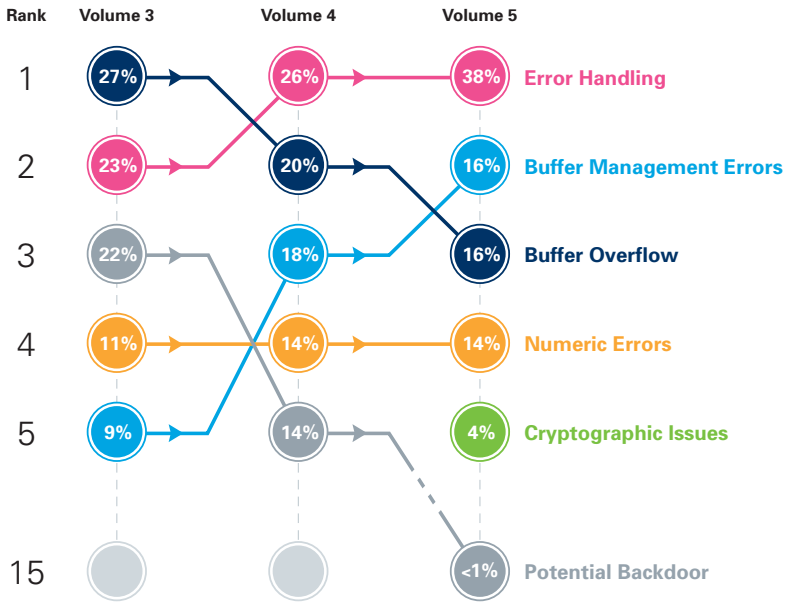

Figure 12: Vulnerability Distribution Trends for C/C++ Applications (Share of Total Vulnerabilities Found)

Vulnerability Prevalence in C/C++ Applications (Percentage of Applications Affected)

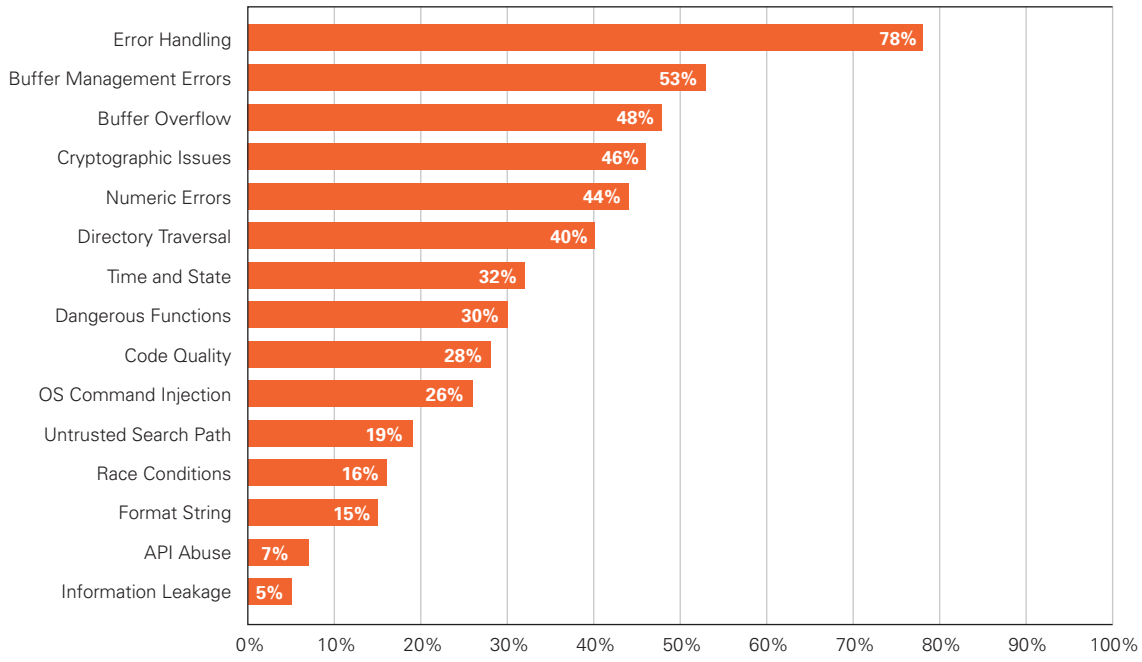

Figure 13: Vulnerability Prevalence in C/C++ Applications (Percentage of Applications Affected)

Percent Improvement in C/C++ Vulnerability Distribution from First to Second Submission

● Indicates categories with the highest vulnerability distribution in C/C++

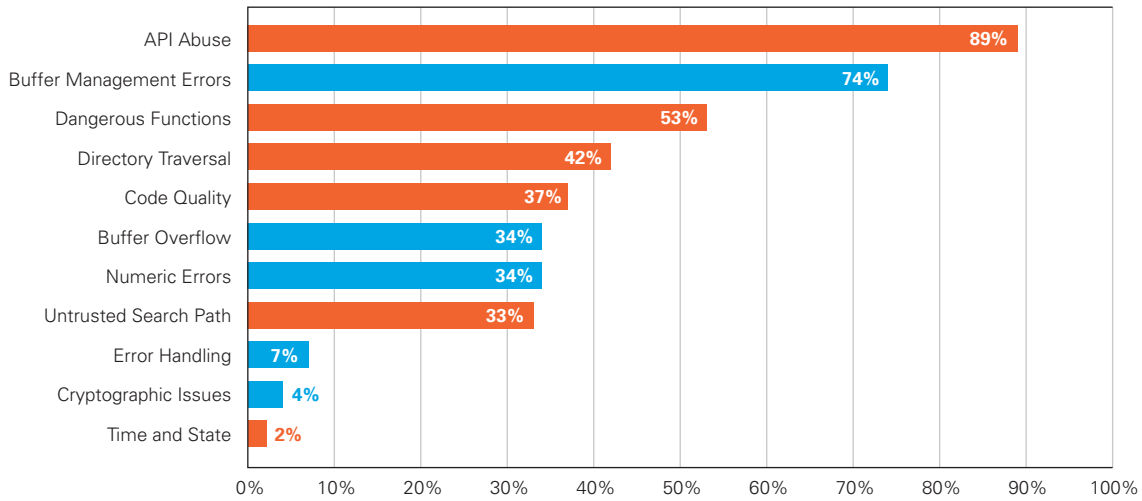

Figure 14: Percent Improvement in C/C++ Vulnerability Distribution from First to Second Submission

PHP

Cross-site scripting has the largest share of total vulnerabilities found in PHP applications (Figure 15). The good news is that the percentage continues to drop from 80% in Volume 3, to 75% in Volume 4, and to 60% in Volume 5. Cryptographic issues debut at second place with 12% share. While, the share of SQL injection vulnerabilities remains steady at 7% it drops to fifth place. 27% of PHP applications have SQL injection issues (Figure 16), with just over half of those vulnerabilities remediated by the second submission (Figure 17).

Interestingly, with PHP we see a similar disconnect between the rankings of vulnerability categories by percentage of affected applications and percent improvement. While XSS holds first place in both Figure 15 and Figure 16, its improvement from build one to build two is in third place (Figure 17). Code injection is ranked second in Improvement but sixth in percent affected applications. Two interpretations are possible. First, the two rankings are not exactly apples to apples. In the case of percent affected applications, all flaws in a given vulnerability category much be fixed to register a change in the metric; while fixing just one flaw in the Improvement metric will (other things being equal) affect improvement. The second interpretation is that developers are not necessarily prioritizing eradication of all flaws of a given category.

Vulnerability Distribution Trends for PHP Applications (Share of Total Vulnerabilities Found)

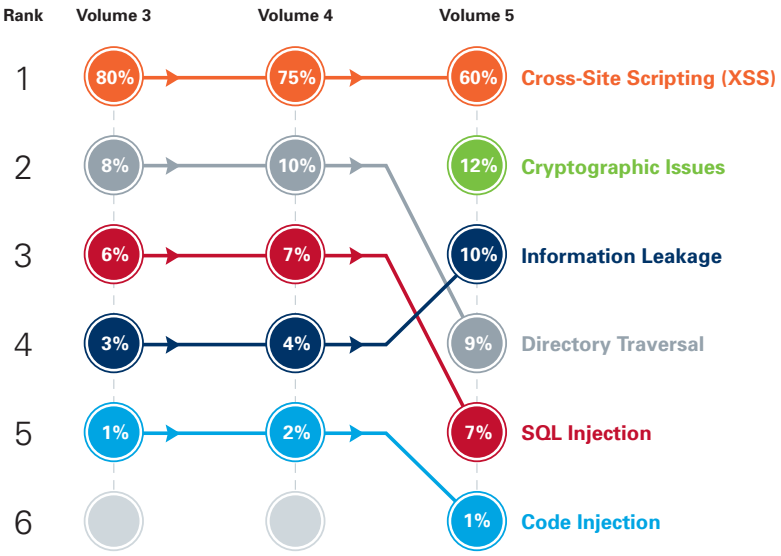

Figure 15: Vulnerability Distribution Trends for PHP Applications (Share of Total Vulnerabilities Found)

Vulnerability Prevalence in PHP Applications (Percentage of Applications Affected)

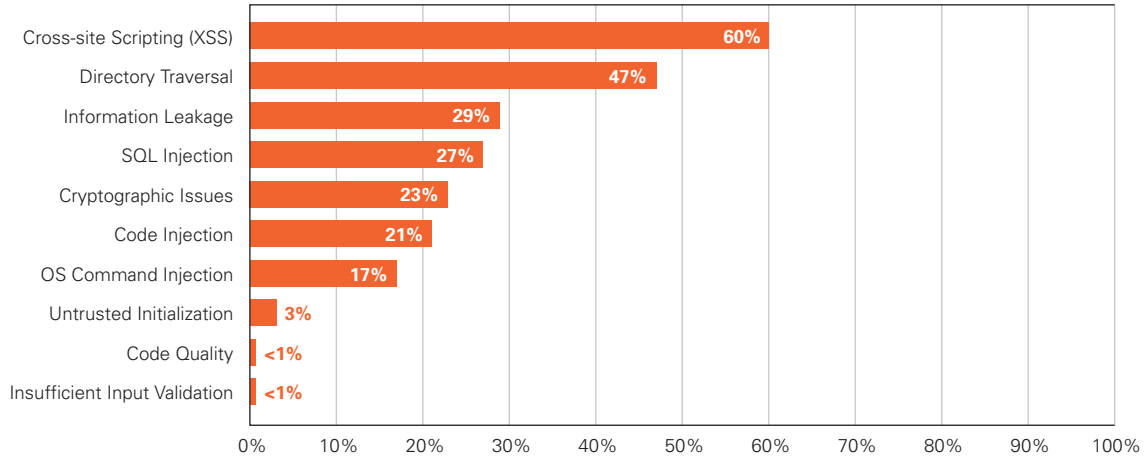

Figure 16: Vulnerability Prevalence in PHP Applications (Percentage of Applications Affected)

Percent Improvement in PHP Vulnerability Distribution from First to Second Submission

● Indicates categories with the highest vulnerability distribution in PHP

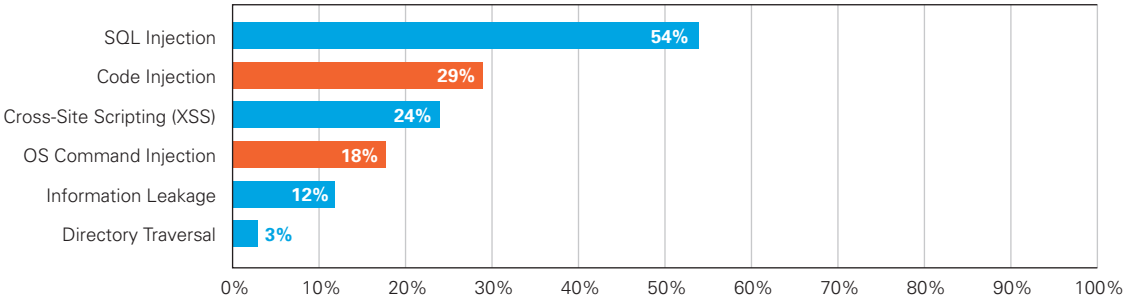

Figure 17: Percent Improvement in PHP Vulnerability Distribution from First to Second Submission

ColdFusion

Cross-site scripting (XSS) has remained the dominant vulnerability category in ColdFusion applications assessed by the Veracode platform. It accounts for 81% of all vulnerabilities found (Figure 18) and affects 94% of applications (Figure 19). Fortunately, recent versions of the ColdFusion platform have built in several fixes and new features to address the high occurrence of XSS. We expect the number of XSS vulnerabilities to decline as we update our scans to reflect platform changes and as customers deploy newer platform versions.

We also found a much lower percentage of ColdFusion applications have been resubmitted for testing within our reporting timeframe, therefore we are not reporting the percentage improvement data for ColdFusion. There are no obvious language-specific reasons for low resubmission rates. However, there may be some organizational issues at work, for example:

- The organizations may be focused on baselining and reporting their current security posture for compliance purposes and have yet to allocate resources for remediation.
- The organizations are relying on network mitigations, such as creating WAF rules based on Veracode's findings, until either the development organization or software supply chain can be influenced to remediate the identified flaws.

Neither of these situations is ideal, as the ultimate goal of an application security program is to improve the organization's security posture over time. Achieving that goal requires vulnerabilities found during the testing process to be remediated, preferably on a prioritized[11] remediation schedule so that developers can focus on the most impactful remediations first.

Vulnerability Distribution Trends for ColdFusion Applications (Share of Total Vulnerabilities Found)

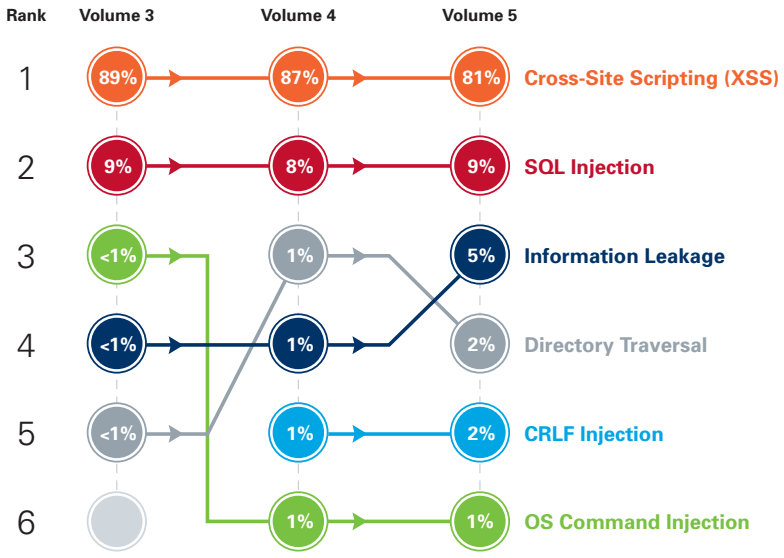

Figure 18: Vulnerability Distribution Trends for ColdFusion Applications (Share of Total Vulnerabilities Found)

[11] The Veracode platform's "Fix First" analysis automatically prioritizes an application's vulnerability results based on severity and effort to fix.

Vulnerability Prevalence in ColdFusion Applications (Percentage of Applications Affected)

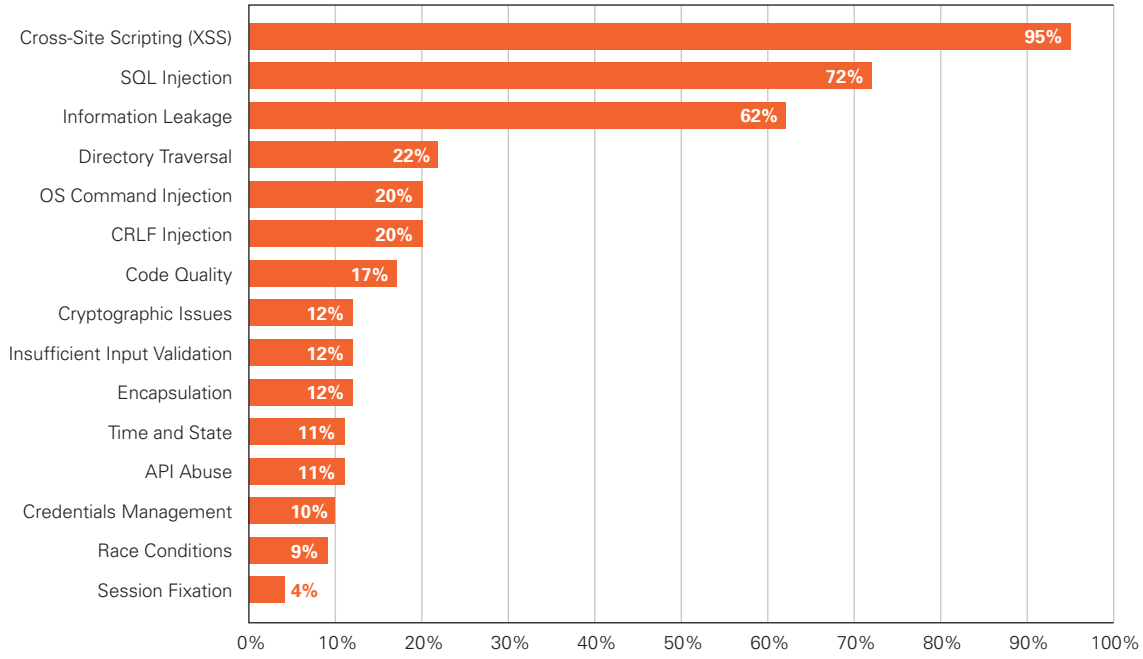

Figure 19: Vulnerability Prevalence in ColdFusion Applications (Percentage of Applications Affected)

Application Types

In this section we take a closer look at three types of applications: mobile, web and non-web. For each application type we start with a brief discussion of the threat landscape and then explore the available vulnerability data.

Mobile Threat Landscape

The mobile threat landscape has three distinct areas: mobile malware, application behaviors, and code vulnerabilities. Each of these areas is a risk to the enterprise that is specific to the mobile space and each manifests itself in a slightly different attack model. Since the risk and attack models are slightly different, enterprises should consider a multi-pronged approach to applying preventative security controls.

The traditional, signature-based malware detection methodology is broken in that modern mobile malware is designed to easily evade signature-based detection. Device vendors have adopted different approaches in attempt to minimize the malware proliferation. Apple adopted the walled approach, in that it has strict controls over which applications are approved for listing in its App Store. However, it is not clear what security measures are being checked during this approval process. Android is more open with more distribution channels and third party app stores. Google Bouncer[12] has some static testing but they are trying to balance turnaround time and security—there is research that shows that this project has a low detection rate of 15.32% (www.cs.ncsu.edu/faculty/jiang/appverify). Both iOS and Android platforms have built in security features such as kill switches to remove identified malware applications from mobile phones.

The behavioral aspects of the threat landscape are particularly important. It is not possible to determine whether data exfiltration is part of the intended user experience. For example, the application FourSquare's transmission of a phone's GPS location is a core component of the functionality delivered to the user. On the other hand if the application is a single-user solitaire game, then the transmission of the phone's GPS location may signify malicious behavior. Having a control point for determining the behavioral aspects of mobile apps may be one reason that enterprises are interested in creating their own internal app stores. An enterprise operated app store could potentially give the enterprise a control point to implement a zero trust model for adopting mobile applications. In a zero trust model, every new application is treated as if it is malware and put into the hands of a malware analyst. The analyst would initially conduct a behavioral analysis, (with static testing tools to understand code and data flows and dynamic testing tools to understand the runtime behavior) and then add human intelligence to determine what the application does, how it maps to malicious intent and what the risk is to their enterprise.

In terms of code vulnerabilities, the Cloud Security Alliance Mobile Working Group released findings[13] in October 2012 that data loss from missing mobile devices ranks as the top threat related to mobile devices for enterprises. The number two and three threats were mobile malware and data leakage due to poorly written third party applications. In the next section we examine the vulnerability distribution and prevalence in mobile applications scanned by the Veracode platform.[14]

[12] *Android and Security,* by Hiroshi Lockheimer, VP of Engineering, Android, February 2012. See googlemobile.blogspot.com/2012/02/android-and-security.html

[13] cloudsecurityalliance.org/csa-news/data-loss-mobile-ranks-top-threat-enterprises

[14] Note that the analysis in this section does not include vulnerability analysis from our recent Marvin Mobile acquisition.

State of Mobile Application Security

First we examine the vulnerability distribution in terms of share of total vulnerabilities discovered across all application builds associated with each mobile platform. In Volume 5, we have enough data on all three platforms to provide a statistically sound basis for comparison.

Figure 20 shows all three mobile platforms that we analyzed share cryptographic issues and information leakage in the Top 5 list of vulnerabilities as measured by percent of total vulnerabilities found. As jailbreaking becomes more common practice and new features such as surviving reboots are supported, cryptographic issues significantly weaken data protection. Attackers with physical control of a mobile device for a small amount of time can jailbreak it and install a backdoor with keyloggers or other malware and/or copy the content. Both cryptographic issues and information leakage vulnerabilities increase the attack surface for mobile applications, and are two of Cloud Security Alliance's top five identified threats to mobile devices.

Vulnerability Distribution for Mobile Platforms (Share of Total Vulnerabilities Found)

Android		iOS		Java ME	
CRLF Injection	37%	Information Leakage	62%	Cryptographic Issues	47%
Cryptographic Issues	33%	Error Handling	20%	Information Leakage	47%
Information Leakage	10%	Cryptographic Issues	7%	Directory Traversal	3%
SQL Injection	9%	Directory Traversal	6%	Insufficient Input Validation	2%
Time and State	4%	Buffer Management Errors	3%	Credentials Management	<1%

Figure 20: Vulnerability Distribution for Mobile Platforms (Share of Total Vulnerabilities Found)

In Volume 4 we published our first analysis of Android applications and focused on cryptographic issues and data exfiltration (i.e. transmission of potentially sensitive information off the device). At the time we reported that 61% of Android applications contained at least one insufficient entropy issue and 39% had potential data exfiltration issues. Figure 21 shows that these categories are still cause for concern. Cryptographic issues affect 64% of Android applications, while information leakage issues occur in 26% of applications.

Cryptographic issues also affect 58% of iOS applications (Figure 22). These are common coding mistakes that can be readily fixed. A hard-coded key is much simpler to extract from a mobile application than from a J2EE web application since the mobile application can simply be copied from the mobile device. Once these keys are compromised any security mechanisms dependent on the secrecy of the keys are rendered ineffective.

Cryptographic issues affect a sizeable portion of Android (64%) and iOS (58%) applications.

Android Vulnerability Prevalence (Percentage of Applications Affected)

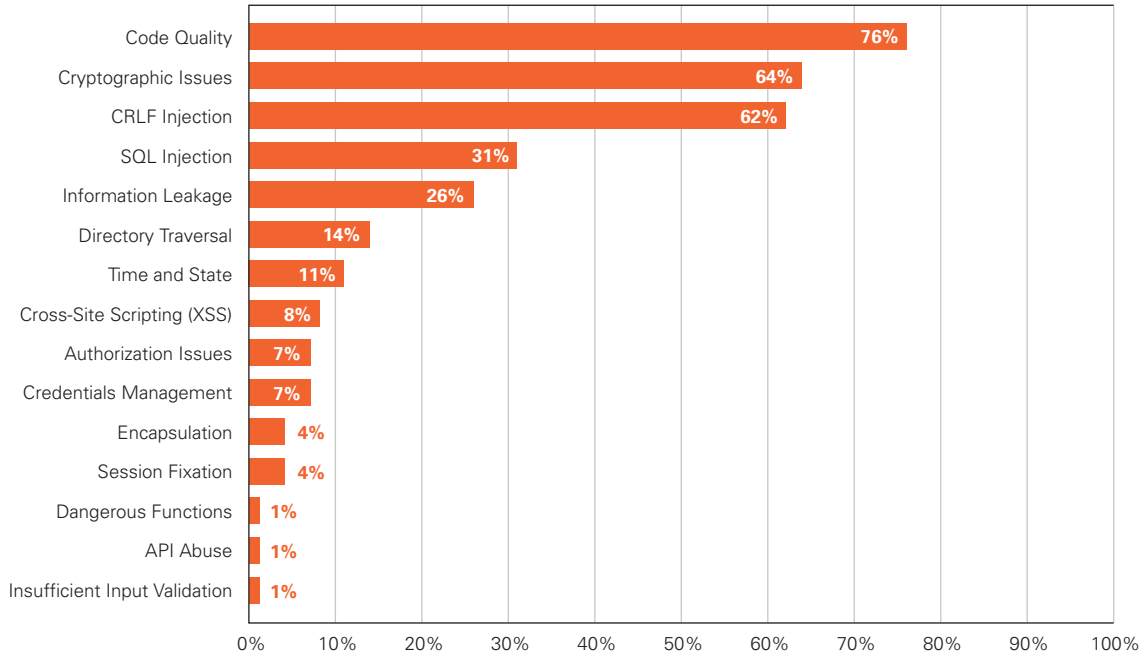

Figure 21: Android Vulnerability Prevalence (Percentage of Applications Affected)

The vulnerability landscape for iOS is somewhat different than Android in part because the differences in the languages used (Objective C in the case of iOS and Java in the case of Android). For example, Figure 22 shows iOS applications are more susceptible to error handling and credentials management, than Android applications. Similarly, Android developers must pay more attention to SQL injection and code quality issues than iOS developers.

iOS (ObjectiveC) Vulnerability Prevalence (Percentage of Applications Affected)

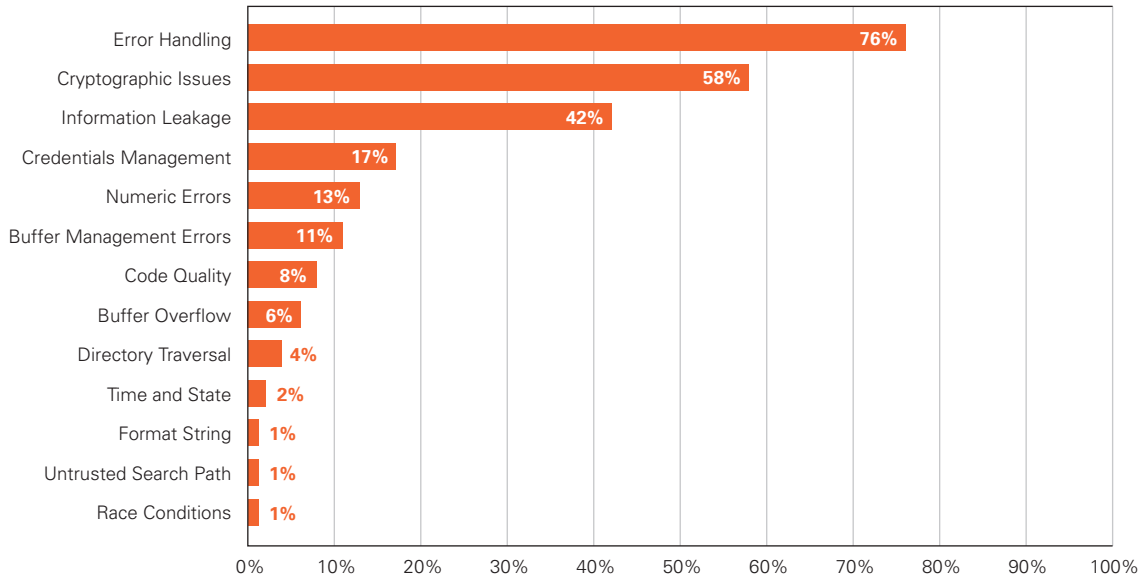

Figure 22: iOS (ObjectiveC) Vulnerability Prevalence (Percentage of Applications Affected)

While the number of iOS and Android applications submitted for testing has grown dramatically over our reporting periods, the same has not been true for BlackBerry applications. The data we collected from BlackBerry applications is relatively small, however, we want to give readers some indication of the vulnerability categories being discovered (Figure 23). Due to the limited amount of data, we expect the statistics to experience some variability and we will assess the value of reporting BlackBerry statistics in future reporting periods.

Blackberry Vulnerability Prevalence (Percentage of Applications Affected)

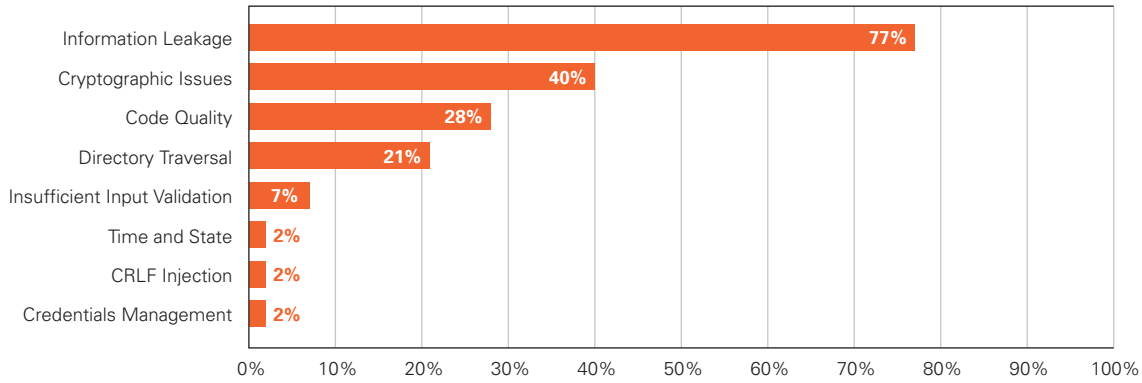

Figure 23: Blackberry Vulnerability Prevalence (Percentage of Applications Affected)

Web Application Threat Landscape

Exploitation of web application vulnerabilities continues to be a significant threat for organizations. The Web Application Security Consortium continually tracks media reported security incidents that can be associated with web application security vulnerabilities in the Web Hacking Incident Database (WHID).[15] Figure 24 illustrates the trends in the top attack techniques, which enables organizations to take a risk-based approach to application security testing and subsequent remediation.

According to WHID, 10% of the incidents occurring in 2012 are categorized as SQL injection attacks, yet millions of records are leaked through SQL injection attacks during 2012.

Trends in Attack Methods from 2010 to 2012

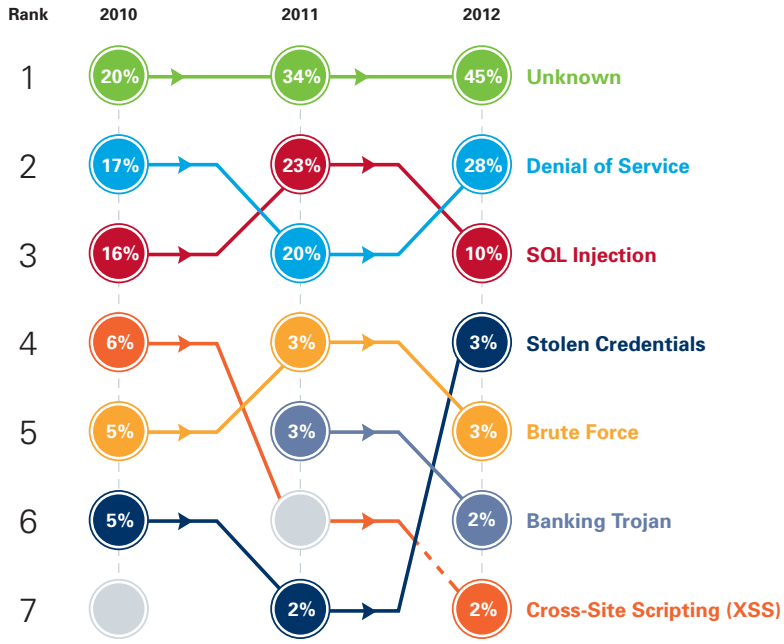

Figure 24: Trends in Attack Methods from 2010 to 2012 (Source: WHID)

45% of the incidents occurring in 2012 are currently categorized as unknown, meaning that there is not yet enough information available to conclusively determine the attack method. The consortium updates the attack method classification as those breaches are investigated. Therefore, it is likely that many of those unknown attacks used one of the other popular techniques listed. The listing shows there is a limited set of vulnerability categories that are exploited at scale; therefore an application security program that covers the top vulnerability categories can reduce the attack surface in a cost-effective manner. This is where most organizations should start when designing an initial application security policy to address the current threat space.

[15] projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database

The rise in Denial of Service attacks appears to be evidence of the rising activity of hacktivist groups such as Anonymous. At the time of writing, WHID contained 82 records associated with Anonymous attacks in 2012. 10% of the incidents occurring in 2012 are categorized as SQL injection attacks, down from 23% in 2011, however it remains in third place in spite of the percentage drop. Figure 25 illustrates how devastating SQL injection attacks can be to organizations.

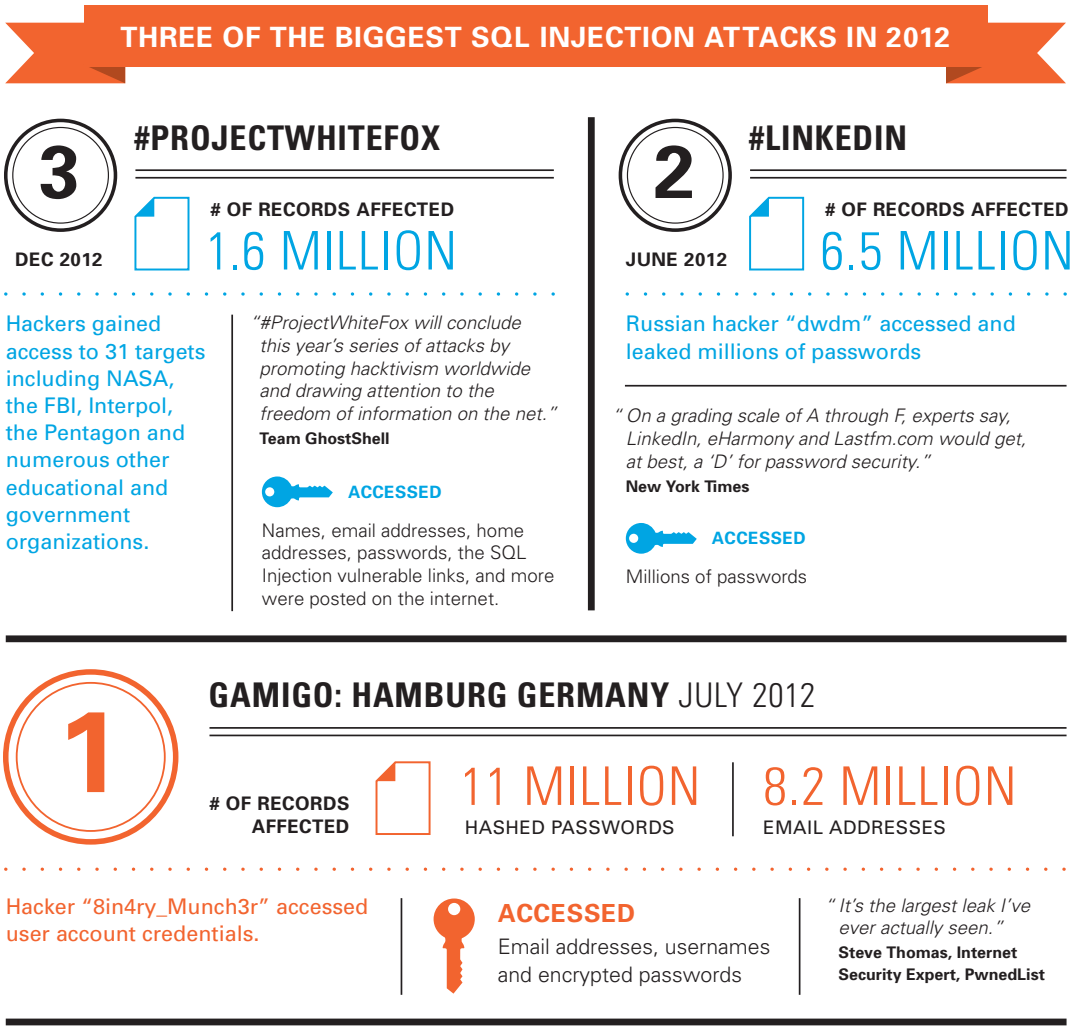

Figure 25: Three of the Biggest SQL Injection Attacks in 2012

State of Web Application Security

Figure 26 shows how the top ten vulnerability categories for web applications have varied over the last three SoSS volumes. Not much has changed. The top five categories remain the same as Volume 4. Cross-site scripting and information leakage are at the top with 67% and 65% respectively. Volume 5 reporting includes two additional CWE categories associated with the insufficient input validation category which vaulted the category to sixth place. API abuse dropped just out of the top ten.

Top Vulnerability Categories (Percentage of Affected Web Application Builds)

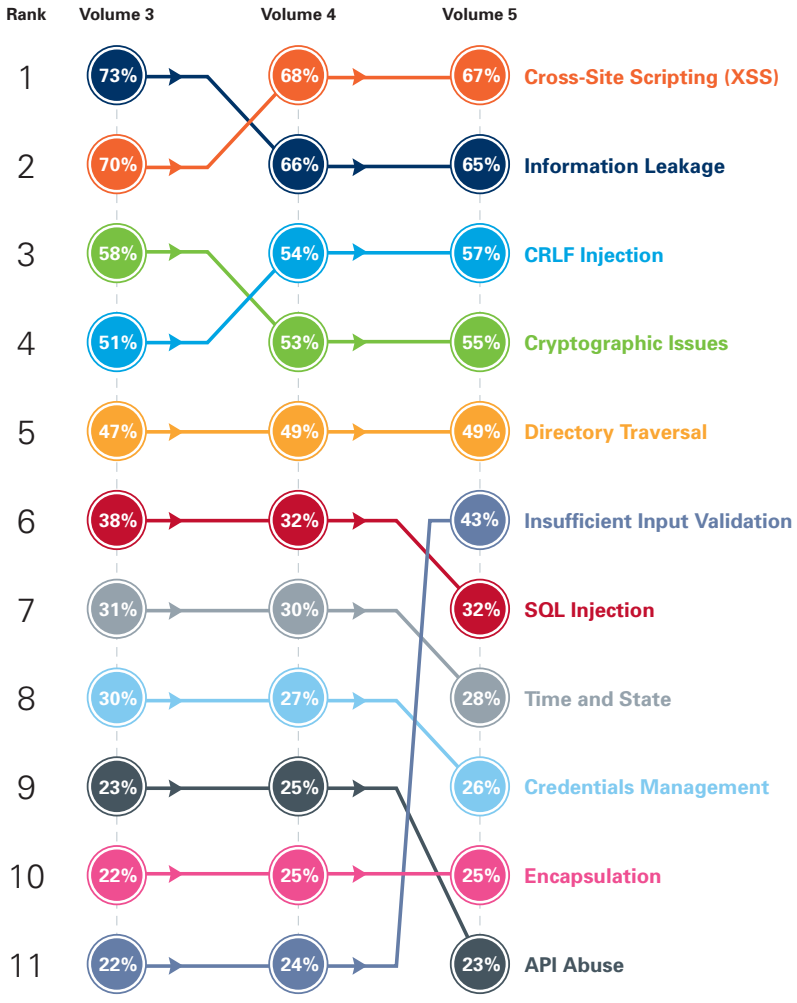

Figure 26: Top Vulnerability Categories (Percentage of Affected Web Application Builds)

The quarterly trend for the percentage of applications affected by cross-site scripting (XSS) remains statistically flat with a p-value of 0.441 (Figure 27), as it has been in both Volume 3 and 4. Although this vulnerability is fairly easy to fix, it is often given a lower remediation priority to other vulnerabilities because attackers are not leveraging them as much in profit-driven attack scenarios.[16] However there are indications that attacks on XSS vulnerabilities are on the rise. According to the Microsoft Security Intelligence Report Vol 13[17] there has been a significant increase in reported XSS cases over the past two years. Both Microsoft and Google have added filters to their browsers to block some types of XSS attacks. However, these mitigations should not be viewed as permanent defenses because not all users will implement the filters and filtering capabilities can be bypassed. The best protection is still eliminating the flaws and using secure coding techniques to prevent the creation of new flaws.

Quarterly Trend for Cross-Site Scripting (XSS) Prevalence (Percentage of Affected Web Applications)
p-value = 0.441

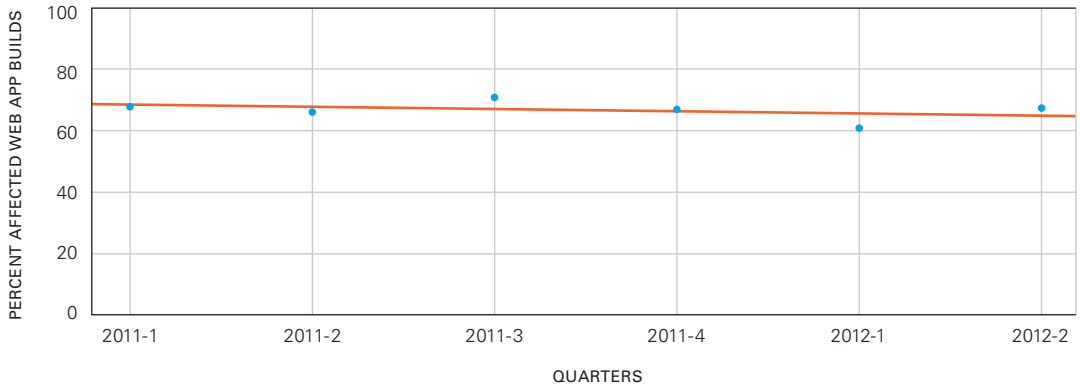

Figure 27: Quarterly Trend for Cross-Site Scripting (XSS) Prevalence (Percentage of Affected Web Applications)

Figure 28 shows the downward trend for SQL injection seen in past Volumes has flattened. For six consecutive quarters, from the first quarter of 2011 to the second quarter of 2012, the percentage of applications affected by SQL injection has hovered around 32% (with a p-value of 0.868). In Volume 3 we reported SQL injection decreasing at a rate of 2.4% per quarter. In Volume 4 the downward trend was still visible, going from 38% of applications affected in the fourth quarter of 2009 to 32% in the third quarter of 2011.

[16] Trustwave 2012 Global Security Report

[17] www.microsoft.com/security/sir/default.aspx

The trend data reported here reveals that eradicating XSS and SQL injection continues to be a significant challenge. Even with increased attention and media coverage linking high profile data breaches to XSS and SQL injection vulnerabilities, there has not been a dramatic reduction in the occurrence of these critical vulnerabilities.

Eradicating SQL injection and cross-site scripting remains a challenge as the vulnerability prevalence trends remain flat for the last 6 quarters. SQL injection has hovered around 32% and cross-site scripting around 67%.

There may be a few factors keeping these trends relatively constant over time, for example:

- An influx of web applications. It is likely that a significant portion of our dataset includes web applications that have never been tested before.
- Expansion of web assessment focus. Security teams are beginning to expand their focus from analyzing a few critical web applications to analyzing the entire website portfolio. This observation implies that we are seeing assessments of many more web applications that were built before security policies were implemented. As these web applications are tested for the first time, we are logging many more instances of XSS and SQL injection.
- Frequent web application changes. Agile development techniques drive more new and updated web applications online with increasing frequency. An enterprise web application portfolio can change from week to week. This means most enterprise portfolios have a built-in influx of new vulnerabilities. Until secure coding techniques go from being recommended best practices to standard web application development practices, CISOs will struggle with maintaining constant vigilance over the website portfolios having a constant stream of new vulnerabilities.

Quarterly Trend for SQL Injection Prevalence (Percentage of Affected Web Applications)
p-value = 0.868

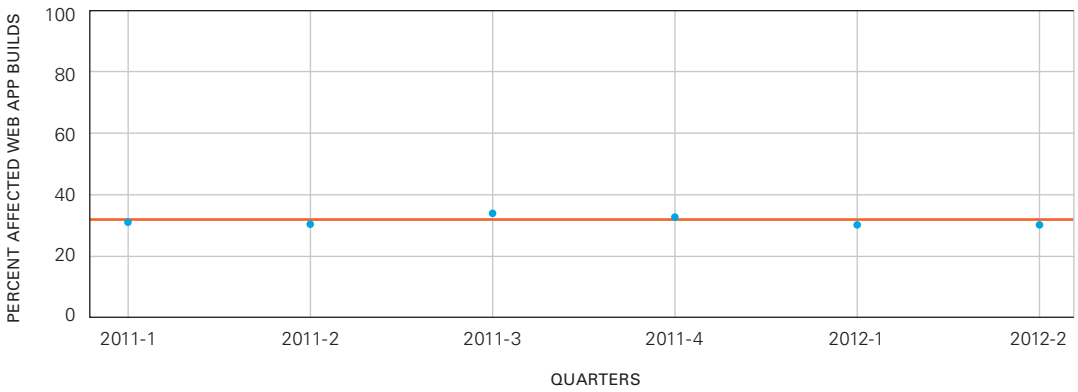

Figure 28: Quarterly Trend for SQL Injection Prevalence (Percentage of Affected Web Applications)

Non-Web Applications Threat Landscape

While most of the media attention focuses on incidents perpetrated by external attackers, organizations must also protect themselves from internal attackers. The threat landscape for applications that are internal to an organization is much more difficult to report on in a comprehensive manner. Most organizations are not willing or able to share relevant information in a public forum in spite of repeated calls for information sharing by law enforcement agencies.[18] Organizations rightfully are concerned about the brand damage of reporting data breaches, particularly those caused by insiders capable of exploiting vulnerabilities in back-office applications. Indeed the 2011 CyberSecurity Watch Survey[19] reported that 46% of respondents considered insiders a more critical threat than external hackers.

State of Non-Web Application Security

Figure 29 shows the trends in the top vulnerability categories for non-web applications over the last three Volumes. Cryptographic issues and directory traversal have remained the top two vulnerability categories for the last three Volumes, affecting 47% and 38% of all non-web applications in the current reporting period. Information leakage (26%) takes the third spot from error handling, which drops to fourth place. Buffer overflow dropped out of the top ten for the first time in this volume, and is replaced by SQL injection which is now affecting 16% of non-web applications.

Cryptography remains the top vulnerability category for non-web applications. 47% of non-web applications contain at least one vulnerability in this category.

The good news is that the percentage of applications with buffer management errors is declining, from 20% in Volume 3 to 13% in Volume 5. However, the rise in the percentage of applications containing information leakage and SQL injection vulnerabilities is disturbing since applications are the conduit through which attackers gain access to confidential or proprietary information.

It is noteworthy that the percentages reported in Figure 29 are generally lower than those reported in our software supply chain feature supplement published in November 2012. For example, cryptographic issues affected 62% of vendor supplied applications but only 47% of all applications (which include internally developed, outsourced, and open source applications in addition to vendor supplied applications). The relatively higher percentages reported in the supplement demonstrate the need for vendors to continue to work towards developing more secure software.

In Volume 5 we corrected a mistake in how the past SoSS analysis logic was counting potential backdoors. The correction accounts for the significant decline in the percentage of applications affected.

[18] www.fbi.gov/news/speeches/combating-threats-in-the-cyber-world-outsmarting-terrorists-hackers-and-spies

[19] www.cert.org/archive/pdf/CyberSecuritySurvey2011.pdf

Top Vulnerability Categories (Percentage of Affected Non-Web Application Builds)

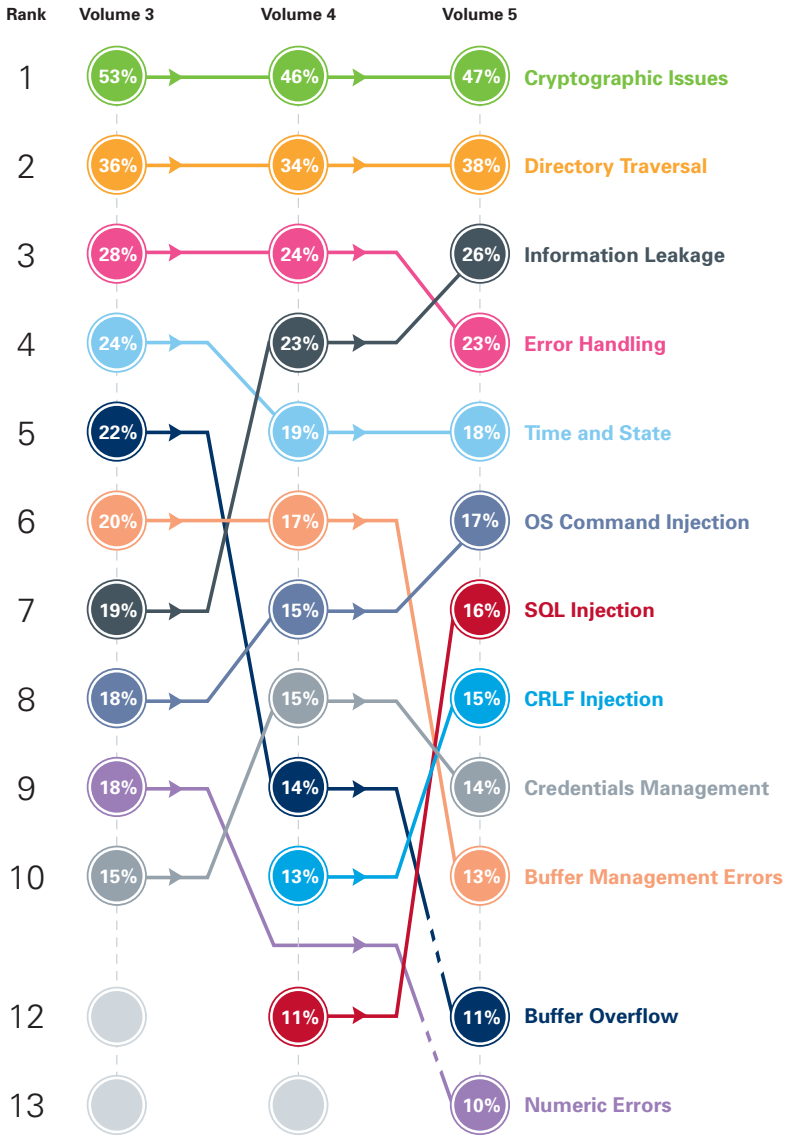

Figure 29: Top Vulnerability Categories (Percentage of Affected Non-Web Application Builds)

In Figure 2 we saw that 69% of non-web applications contain at least one flaw in a vulnerability category appearing on the 2011 CWE/SANS Top 25 list of most dangerous software errors. Figure 30 provides more details, by showing the percentage of applications affected by vulnerability categories which appear on the CWE/SANS list. Vulnerabilities that appear on the CWE/SANS list are relatively easy to find and exploit for malicious actors with access to the installed software.

Vulnerability Categories on the 2011 CWE/SANS Top 25 List (Percentage of Applications Affected)

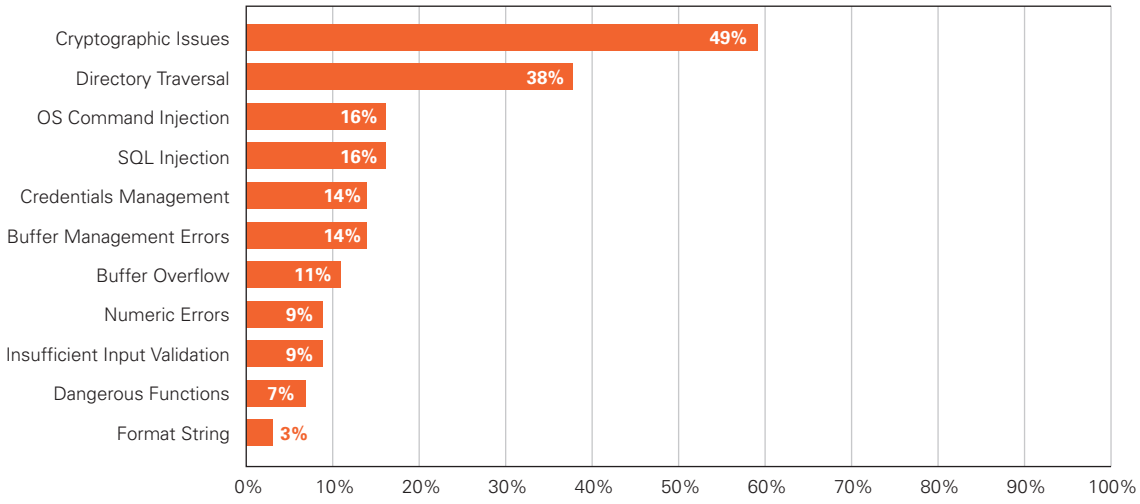

Figure 30: Vulnerability Categories on the 2011 CWE/SANS Top 25 List (Percentage of Applications Affected)

Appendix A: About the Dataset

Figure 31 illustrates how the language and platform distribution has evolved since Volume 3. Overall, only small shifts in percentages occurred between Volume 4 and 5. Java and .NET applications continue to dominate, together representing 77% of the assessments conducted. C/C++ applications are holding steady at 9%. Mobile applications have increased to 3%.

Distribution by Language Apps

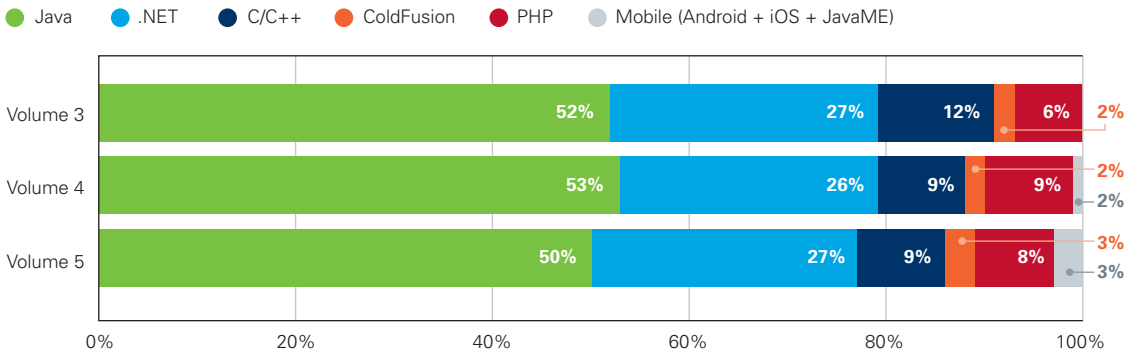

Figure 31: Distribution by Language Apps

All applications analyzed by Veracode are classified according key characteristics such as whether the application is web-facing, whether it is developed for a mobile platform (Android, iOS), the remaining applications are typically developed to operate behind the firewall (although they can include web-based user interfaces). Figure 32 shows that 73% of the applications assessed were web applications, which is slightly down from 75% in Volumes 3 and 4. Mobile application assessments are filling the gap, growing from <1% in Volume 4 to 3% in the Volume 5.

Distribution of Web, Mobile and Other Applications

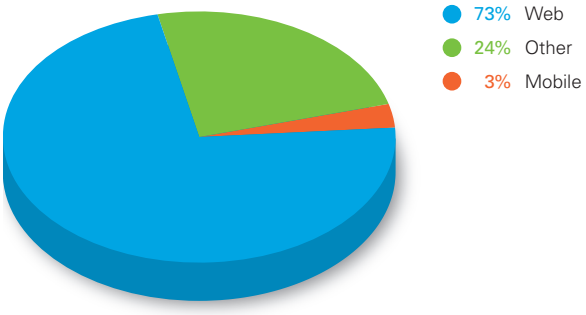

Figure 32: Distribution of Web, Mobile and Other Applications

Figure 33 and Figure 34 show that Java is the most popular language for both web (56%) and non-web applications (39%) tested by the Veracode platform. 28% of web applications and 29% of non-web applications were written in .NET. C/C++ accounted for 29% of non-web applications but only 2% of web applications.

Distribution of Web Applications by Language

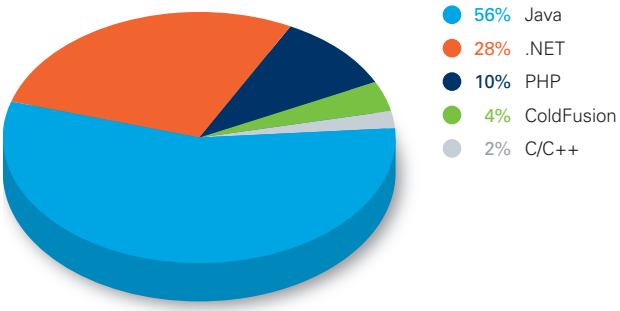

Figure 33: Distribution of Web Applications by Language

Distribution of Non-Web Applications by Language

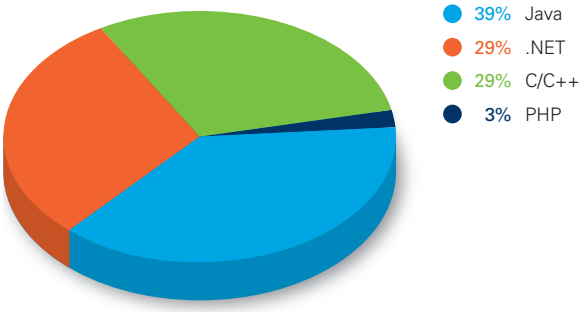

Figure 34: Distribution of Non-Web Applications by Language

Figure 35 shows the platform distribution of mobile applications in our dataset, with iOS representing the largest share (56%). At first glance, the relatively large percentage of applications developed on iOS may appear surprising, since one would expect that most enterprises and software vendors to create and test Android and iOS versions of their mobile applications. Veracode's partnership with Good Technology to test iOS applications for inclusion in enterprise app stores accounts for this imbalance.

Distribution of Mobile Applications by Platform

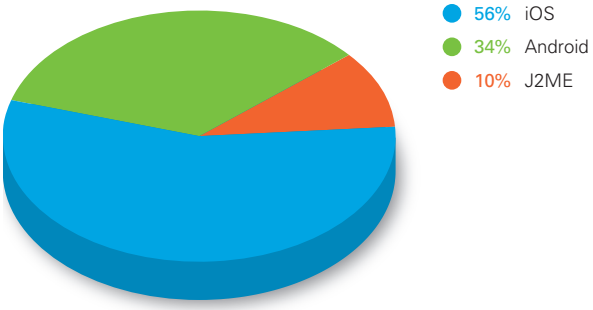

Figure 35: Distribution of Applications by Supplier

Figure 36 shows approximately 22% of the applications analyzed were identified as third-party (commercial, open source and outsourced). The percentage of outsourced applications remains low at 1%. Oftentimes the outsourced nature of applications labeled "internally developed" is only revealed during remediation when an outsourced party is assigned the task of fixing vulnerabilities. Hence we believe that the true percentage of "outsourced" code is higher than represented.

Distribution of Applications by Supplier

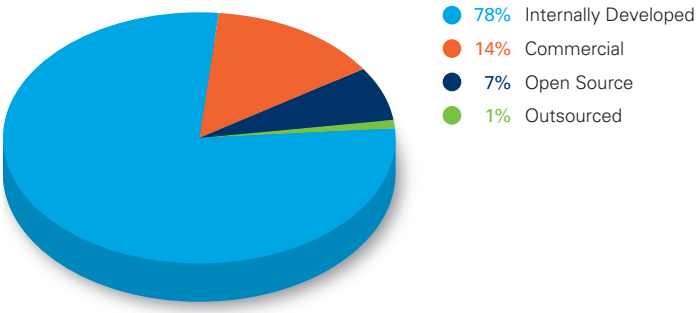

Figure 36: Distribution of Applications by Supplier

Figure 37 represents the distribution of applications by industry segment. Other and Finance represent the largest segments at 38% and 27% respectively.

Distribution of Applications by Industry

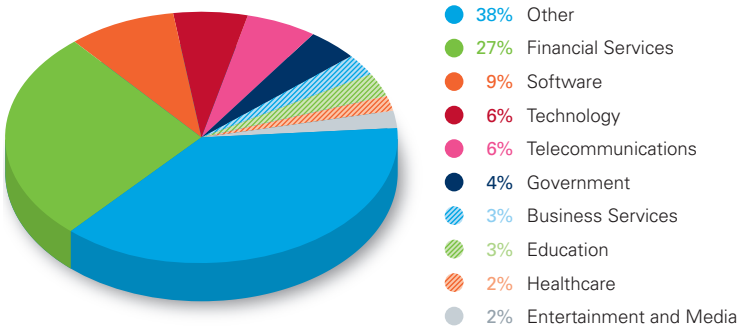

Figure 37: Distribution of Applications by Industry

Appendix B: Understanding How the Veracode Platform Determines Policy Compliance

The Veracode Platform automatically determines whether an application is compliant with OWASP Top 10, CWE/SANS Top 25, the assigned Veracode Level and the assigned enterprise policy. Veracode looks for specific flaws enumerated by the CWE list and uses standardized mappings to determine whether a flaw belongs in the most recently published OWASP Top 10 and CWE/SANS Top 25 lists. Flaws discovered in an application are compared to the respective standards. If even a single flaw belonging to the standard is discovered, then the application is deemed out of compliance with the standard. For our report, web applications are assessed against the OWASP Top 10 while non-web applications are assessed against the CWE/SANS Top 25.

Veracode Levels are Veracode's independent standard for evaluating an application's software quality. Veracode Levels are assigned based on the business criticality of application. Each Veracode Level provides a predefined security policy that aligns with different levels of risk the organization is willing to accept for applications of varying business criticality. Figure 38 shows the five Veracode Levels aligned with the five business criticality levels as defined by the National Institute of Standards and Technology (NIST). When an application is assigned a business criticality, the Veracode platform automatically assigns the appropriate Veracode Level as a predefined policy and determines whether the application is compliant with the Veracode Level. Each Veracode Level policy contains a combination of the severity of the flaws found in the application, the types of tests being performed on the application and the application's overall security quality score. An application is deemed compliant when all three aspects of the Veracode Level policy are met.

Predefined Policy Requirements for Veracode Levels

		Predefined Policy Requirements		
Business Criticality	**Veracode Levels**	**Flaw Severities Not Allowed in This Level**	**Required Test Methodologies**	**Minimum Security Quality Score**
Very High	VL5	Very High, High, Medium	Static and Manual	90
High	VL4	Very High, High, Medium	Static	80
Medium	VL3	Very High, High	Static	70
Low	VL2	Very High	Static or Dynamic or Manual	60
Very Low	VL1	Not Applicable	Static or Dynamic or Manual	Not Applicable

Figure 38: Predefined Policy Requirements for Veracode Levels

Veracode provides enterprises several options for defining custom policies, including:

- Disallowing specific flaw severities
- Disallowing specific types of flaws (specified by CWE number)
- Attaining a minimum security quality score
- Using predefined policies such as OWASP Top 10, CWE/SANS Top 25 or industry standards such as PCI
- Specifying application test methodologies and frequencies (e.g. monthly, quarterly, annually)
- Meeting timelines for remediating specified flaw severities

A custom enterprise policy may include any or all of these options. An application is deemed compliant when all aspects of the enterprise's custom policy are met.

Whisker Plot Definition

A whisker plot (aka box and whisker plot) is a graphic used for exploratory data analysis. Created by the great data analyst John W. Tukey, a whisker plot can show the distribution of a dataset in one quick glance. As an example, let us look at the whisker plot in Figure 5. The figure shows nine distribution plots of Veracode Security Quality Score (SQS), one for the first application build scanned, one for the second application build scanned, and so on. Each application build is indicated by the numbers on the x-axis. Looking at the first application build, we see one whisker (the dotted vertical line) and one box (the colored shape) occurring along each whisker.

Veracode Security Quality Score by Build

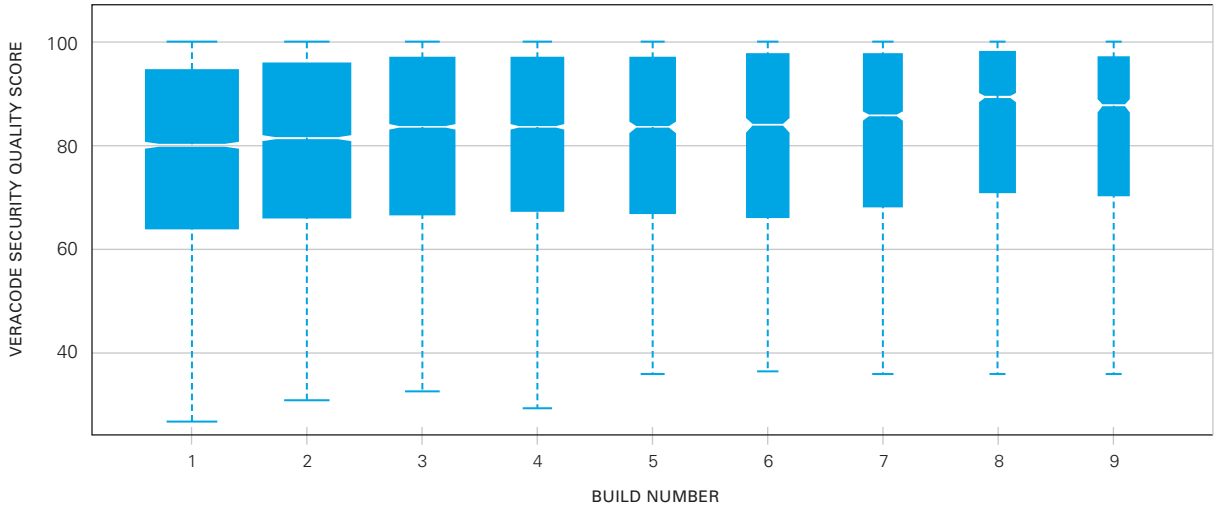

Figure 5: Veracode Security Quality Score by Build

The two horizontal lines at the bottom and top of the whisker reflect the minimum and maximum SQS observed for the first build. For the first build, the maximum score is 100 and the minimum score is 27. Looking across all nine builds, we see that the maximum for each build is a perfect score of 100. The minimum SQS values creep up from 27, to 31, to 33 for builds 1, 2, and 3. One might expect this as build 2 represents a resubmission of build 1 with several flaws remediated and similarly for builds 2 and 3.

The box indicates values of the first quadrant, median, and third quadrant for all SQS values for the associated build as we move from bottom to top of the figure. The narrow area in the middle of the box is called the "notch" which is defined by values called notchLow, notchHi and the median. For example, the first build has a notchLow value of 80 and a notchHi value of 81. The notch visually depicts values above and below the median that mark roughly a 95% confidence interval. The way to interpret this is to look across multiple whiskers for non-overlapping notches. When two whiskers have non-overlapping notches one can say with 95% confidence that the two sets of observations have a different median. From the above figure one can conclude that improvement in the SQS values for application builds actually does improve from build one to two to three, with 95% confidence.

Whisker plots also can visually reflect skewness in a distribution. For example, if the box area below the notch is much greater than the box area above the notch, then the distribution is skewed low. Build eight is skewed low—there is a wider range of SQS values in build eight below the median than above the median. There appears to be little skew for builds one and two. Note also that the width of the box area is proportional to the number of application builds observed. As expected, there are many more SQS observations for build 1 than for later builds.

P-Value Definition

For all of the quarterly trend graphs that are presented in this report, we also provide a p-value. In this context, the p-value can be viewed as a metric that is derived via a linear regression model. The p-value is the probability of observing a result at least as extreme as what we observed, given an assumption that the trend is flat.

For example, in Figure 27, we observed what appears to be a decreasing trend in percent of web application builds with cross-site scripting flaws for six quarters. A p-value of 0.441 indicates that the probably of seeing this in our data is over 44% when, in fact, the trend is flat—neither increasing nor decreasing. This means that we can continue to be hopeful that the trend is down and we can continue to monitor progress, but, speaking statistically, the data does not support any conclusion other than that the trend is flat.

Quarterly Trend for Cross-Site Scripting (XSS) Prevalence (Percentage of Affected Web Applications)
p-value = 0.441

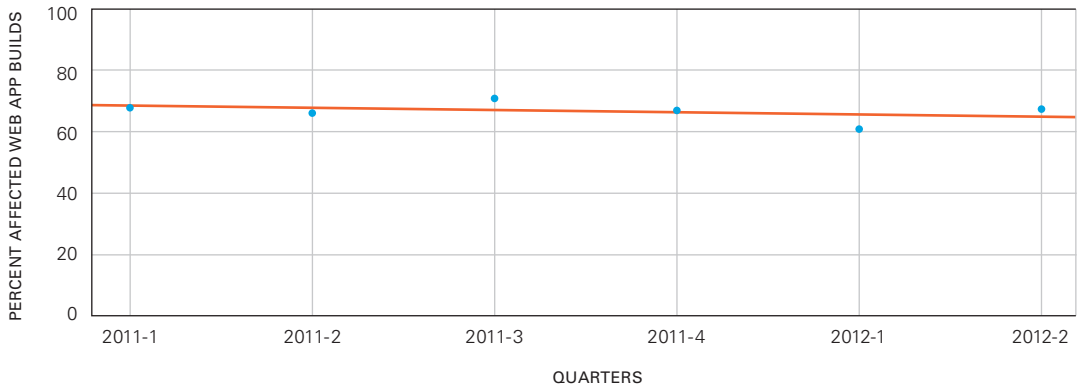

Figure 27: Quarterly Trend for Cross-Site Scripting (XSS) Prevalence (Percentage of Affected Web Applications)

Generalized Linear Model

We used a generalized linear model (GLM) to evaluate the interactions between Volume, language, industry, and supplier on the proportion of first builds that passed the CWE/SANS Top 25 compliance policy. This model leverages the following properties of our data:

- The response variable is strictly bounded,
- The variance is non-constant, and
- The errors are non-normal.

In addition, due to the large number of industry categories (see Figure 37: Distribution of Applications by Industry) with resulting low sample sizes in the model, we consolidated all categories other than Financial, Software, and Government into one category named Other. In particular, the new expanded Other category includes: Media & Entertainment, Health, Education, Business Services, Telecommunications, Technology, and Other.

The data is proportional but the GLM for interactions using the binomial distribution for errors has dispersion factors over 1.8 for all explanatory variables so we used the quasibinomial distribution for errors. Additionally, as is customary, we used the logit "link" function, namely the log-odds (log (p/q)) function as the transformation for raw proportional observations. The results of the GLM for analyzing the interaction between the Volume and Supplier factors on SANS Compliance Failure Rate for first builds (without any model simplification) are as follows:

```
Call:
glm (formula = y - Volume * Supplier, family = quasibinomial)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-9.6387771   0.2108186   0.3907429   0.7335999   4.5222136

Coefficients:
                                          Estimate     Std.Error  t value               Pr(>|t|)
(Intercept)                             1.98176746    0.19030085 10.41387 < 0.000000000000000222 ***
Volume SOSSv5                           1.48953329    0.37642676  3.95703             0.00008713 ***
SupllierInternally Developed           -0.04391537    0.23371916 -0.18790              0.8510345
SupplierOpen Source                     1.67665279    0.57449701  2.91847              0.0036796 **
SupplierOutsourced                     13.58430079 1005.61138499  0.01351              0.9892276
VolumeSOSSv5:SupplierInternally Developed  -0.04479984    0.43773916 -0.10234              0.9185258
VolumeSOSSv5:SupplierOpen Source        0.44861486    1.36860963  0.32779              0.7432117
VolumeSOSSv5:SupplierOutsourced       -14.22238819 1005.61294547 -0.01414              0.9887217
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasibinomial family taken to be 2.864496721

    Null deviance: 1133.05488  on 496  degree of freedom
Residual deviance:  885.25567  on 489  degree of freedom
AIC:NA

Number of Fisher Scoring iterations: 14
```

Performing model simplification, we conclude that there is no compelling evidence that different Supplier types influence CWE/SANS Compliance Failure Rate. For details on both the implementation of the GLM calculations as well as interpretation of the above output, we refer you to the Handbook of Statistical Analyses Using R (cran.r-project.org/web/packages/HSAUR/vignettes/Ch_logistic_regression_glm.pdf).

VERACODE

Veracode, Inc.
65 Network Drive
Burlington, MA 01803

Tel +1.339.674.2500
Fax +1.339.674.2502

www.veracode.com

SSSR/VOL5/US/0313

ABOUT VERACODE

Veracode is the only independent provider of cloud-based application intelligence and security verification services. The Veracode platform provides the fastest, most comprehensive solution to improve the security of internally developed, purchased or outsourced software applications and third-party components. By combining patented static, dynamic and manual testing, extensive eLearning capabilities, and advanced application analytics, Veracode enables scalable, policy-driven application risk management programs that help identify and eradicate numerous vulnerabilities by leveraging best-in-class technologies from vulnerability scanning to penetration testing and static code analysis. Veracode delivers unbiased proof of application security to stakeholders across the software supply chain while supporting independent audit and compliance requirements for all applications no matter how they are deployed, via the web, mobile or in the cloud. Veracode works with customers in more than 80 countries worldwide representing Global 2000 brands. For more information, visit www.veracode.com, follow on Twitter: @Veracode or read the Veracode Blog.