VOLUME I

State of Software Security Report

The Intractable Problem of Insecure Software

March 1, 2010

VERACODE

Software Security Simplified

ON ALMOST A DAILY BASIS, news of data breaches and cyber threats remind us that the security of our critical financial and public services software infrastructure is inadequate. Software is the very fabric of how the world communicates and conducts business. The question we face today is whether the resiliency and security quality of software will ultimately determine the rate at which it continues to be the creative force behind human progress.

In this first-ever report on the State of Software Security, Veracode offers actionable information designed to protect organizations from the very real and costly impact of insecure software. This semi-annual report comprises security intelligence gleaned from billions of lines of code analyzed by the world's first and only cloud-based application risk management services platform. It is our hope that this report assists executives, policy makers, purchasing groups, security professionals, and software development professionals involved in the global software supply chain to make and buy more secure software.

As you examine the information presented here I welcome your questions and ideas about why software remains so vulnerable and what we can do to collectively improve its security quality. Please visit my blog to continue this important conversation.

Best Regards,

Matthew Moynahan
Chief Executive Officer, Veracode

veracode.com/ceo-blog

Table of Contents

Introduction

Real information about the state of software security is needed to understand why software remains so insecure. Until now most of the information available has come from "perimeter defense" companies that provide network, gateway or endpoint protection technologies such as firewalls and anti-virus. While valuable, these approaches are insufficient because they focus on known vulnerabilities and not the all-important unknown or Zero Day vulnerabilities that are hidden in the final application binary and subject to attack by sophisticated hackers. Other reports, such as those from website security testing companies, typically represent only one type of testing ("black box" or "dynamic") performed against a single type of application (web applications).

According to a SANS Report issued in September 2009 on Top Cyber Security Risks, "the number of vulnerabilities being discovered in applications is far greater than the number of vulnerabilities discovered in operating systems. As a result, more exploitation attempts are recorded on application programs."[1] The impact of these attacks has been widespread. In a recent Forrester Research study[2] 62% of respondents claimed to have been the victim of a breach exploiting software vulnerabilities.

Missing until now has been security intelligence derived from multiple testing methodologies (static, dynamic, and manual) on the full spectrum of application types (components, shared libraries, web and non-web applications) and programming languages (including Java, C/C++, and .NET) from every part of the software supply chain (Internally Developed, Open Source, Outsource, Commercial) that organizations rely on. By filling this void the Veracode State of Software Security Report aims to bring clarity and a broader perspective into the security quality produced by the complex global software supply chain.

This semi-annual report is the most comprehensive of its kind as it draws on continuously updated information resident in Veracode's unique cloud-based application risk management services platform. The data represents intelligence gleaned from the analysis of billions of lines of code and thousands of applications. It is growing every minute as more organizations come to Veracode for independent verification of the security quality of their software. As the only cloud-based application risk management services platform in the world to perform and aggregate results from multiple testing techniques, application types, and participants across the global software supply chain, Veracode is in the unique position of being able to provide the software community with the broadest and deepest repository of code level application security intelligence in the world.

For those executives, security and development professionals who want to better understand the vulnerabilities that threaten the integrity and performance of software in the software supply chain, this series of reports is essential reading. Veracode welcomes any questions or comments from readers and will continually strive to improve and enrich the quality and detail of our analysis. Additionally, we invite all members of the software supply chain to participate in constructive dialogue on the topic of software security at veracode.com/ceo-blog.

[1] www.sans.org/top-cyber-security-risks

[2] www.computerworld.com/s/article/9132506/Survey_Software_flaws_account_for_breaches_at_62_of_companies

Executive Summary

As the only cloud-based application risk management services platform in the world to perform and aggregate results from multiple security testing techniques and application types across all participants in the global software supply chain, Veracode is in the unique position of being able to offer the broadest and deepest repository of code level application security intelligence in the world. In this first-ever report of its kind Veracode found strong code-level evidence to support the following observations:

1. Most software is indeed very insecure.
2. Third-party software is a significant percentage of the enterprise software infrastructure, and third-party components are a significant percentage of most applications.
3. Open source projects have comparable security, faster remediation times, and fewer Potential Backdoors than Commercial or Outsourced software.
4. A significant amount of Commercial and Open Source software is written in C/C++ making it disproportionately susceptible to vulnerabilities that allow attackers to gain control of systems.
5. The pervasiveness of easily remedied vulnerabilities indicates a lack of developer education on secure coding.
6. Software of all types from Finance and Government sectors was relatively more secure on first submission to Veracode for testing.
7. Outsourced software is assessed the least, suggesting the absence of contractual security acceptance criteria.

Key Findings

1. Most software is indeed very insecure.

Regardless of software origin, 58% of all applications submitted for verification did not achieve an acceptable security score for its assurance level upon first submission to Veracode for testing when assessed using Veracode's risk adjusted verification methodology.[3] When evaluating against OWASP Top 10 (2007) and CWE/SANS Top 25 Most Dangerous Programming Errors (2009) standards, neither of which adjust for risk, Internally Developed applications fared the poorest, with failure rates as high as 88%. Extrapolating from the application sample set, more than half of the software deployed in enterprises today is potentially susceptible to an application layer attack similar to that used in the recent Heartland[4] or Google[5] security breaches.

Recommendation(s): Implement a comprehensive, risk-based application security program. Design your secure software initiative for breath and depth. Going deep on a handful of applications and ignoring the rest will not lower your organization's overall application risk. Each development team should be implementing a minimum process for application security as part of the development lifecycle. Establishing a security verification step for about-to-be deployed applications is the best place to start.

[3] Refer to Methodology section in Addendum for a description of Veracode's risk adjusted verification methodology.

[4] www.computerworld.com/s/article/9126379/Heartland_data_breach_could_be_bigger_than_TJX_s, www.businessweek.com/technology/content/jul2009/tc2009076_891369_page_2.htm

[5] www.forbes.com/2010/01/14/google-china-mcafee-technology-cio-network-hackers_print.html, www.wired.com/threatlevel/2010/01/operation-aurora

2. Third-party software is a significant percentage of the enterprise software infrastructure, and third-party components are a significant percentage of most applications.

Of applications in the sample set, 60% were designated as Internally Developed, 30% were designated as Commercial, and 10% were designated as Open Source or Outsourced. Regardless of the designation, Veracode observed that between 30% and 70% of all code comprising Internally Developed applications was identifiably from third-parties. Furthermore, there was a "nesting effect" as third-party components themselves often contained other third-party components.

> Recommendation(s): Implement security acceptance criteria and policies for an approved list of third-party suppliers, and conduct security testing on third-party components prior to integrating into final application. Do not develop a false sense of security and control when developing applications with internal teams given the abundance of third-party code integrated into all software.

3. Open Source projects have comparable security, faster remediation times, and fewer Potential Backdoors than Commercial or Outsourced software.

As noted above, no software supplier excelled at delivering secure software upon first submission. Only 39% of submitted Open Source applications and 38% of Commercial applications were acceptable on first submission when evaluated against the CWE/SANS Top 25 Most Dangerous Programming Errors. Open Source applications fared somewhat better than Internally Developed applications, which had an acceptable rate of only 31% against the same industry benchmark.

Open Source project teams remediated security vulnerabilities faster than all other users of Veracode's application risk management services platform. Open Source applications took only 36 days from first submission to reach an acceptable security score, compared to 48 days for Internally Developed applications and 82 days for Commercial applications. This is not surprising given the numerous political and organizational complexities of enterprise development efforts and the formal, customer-centric release plans of Commercial software vendors.

Finally, Open Source contained the fewest Potential Backdoors of any software supplier; substantially less than 1% of vulnerabilities detected across all Open Source applications fell into this category. The relative absence of Potential Backdoors is apparent testimony to the positive effect of transparency in the Open Source community.

> Recommendation(s): Do not fall victim to the fear, uncertainty, and doubt surrounding the use of Open Source software in critical business infrastructure. However, given the risks associated with using code of unknown security, test Outsourced, Commercial, and Open Source suppliers as rigorously as you would test Internally Developed code for security quality and backdoors.

4. A significant amount of Commercial and Open Source software is written in C/C++ making it disproportionately susceptible to vulnerabilities that allow attackers to gain control of systems.

More than 30% of applications were identified as Commercial and supplied to the enterprise by Independent Software Vendors. Commercial suppliers are more likely to use C/C++ than any other language, which can increase risk. Of the C/C++ applications Veracode analyzed, 42% contained vulnerabilities that, if exploited, could result in remote code execution. The vulnerability in Internet Explorer 6 that enabled the Aurora attacks is an example of a remote code execution vulnerability. These classes of defects, including buffer overflows, integer overflows, use after free, and others, are well-known coding errors that have been difficult to eradicate from C/C++ based programs. To exacerbate the problem for enterprises, much of the software written in C/C++ is purchased from software vendors, not built internally. As Gartner[6] and others are recommending, organizations should have purchased software reviewed for security to mitigate this risk.

Recommendation(s): Critical business systems often comprise multiple tiers and development languages. Many contain a hybrid of managed and native code originating from a heterogeneous software supply chain. C/C++, with its idiosyncratic vulnerabilities, is pervasive in the supply chain and no verification process that ignores it will be successful.

Despite the higher likelihood of remote execution vulnerabilities in C/C++, do not be complacent about the risks presented by software written in other languages. Our data reinforces the fundamental notion that serious coding vulnerabilities exist across all languages.

5. The pervasiveness of easily remedied vulnerabilities indicates a lack of developer education on secure coding.

Cross-site Scripting (XSS), the most prevalent vulnerability category by overall frequency and the third most prevalent by number of affected applications, is a stark illustration of the challenges of writing secure code. Despite nearly a decade of focus on cross-site scripting as a serious security threat, its continued prevalence reflects both the pervasive nature of the problem and the evolving threat landscape (i.e. increasing use of dynamic web content). Cross-site Scripting remains as rampant as ever, undeterred by the wide availability of libraries intended to eliminate the risk via proper output encoding. Better education of web developers on this vulnerability and others such as SQL Injection is essential.

Recommendation(s): Implement specific developer training initiatives as part of your overall security program. Follow the lead of corporations such as Microsoft in addressing as many coding mistakes as possible during the education phase of the Secure Development Lifecycle.[7] Educating developers is a cost effective way of preventing security vulnerabilities from being introduced into critical applications. Remember that developer education only helps significantly on new code. Use static analysis on legacy code to eliminate the XSS vulnerabilities already in your code base.

[6] blogs.gartner.com/neil_macdonald/2010/01/14/more-application-security-goodness-from-owasp

[7] blogs.msdn.com/sdl/archive/2009/01/27/sdl-and-the-cwe-sans-top-25.aspx

6. Software of all types from Finance and Government sectors was relatively more secure on first submission to Veracode for testing.

More than half of applications in the Financial Related Industries and Government sectors were deemed acceptable at first submission, using Veracode's risk adjusted verification methodology. This placed them at the top of the more than 15 industries represented in the data set. To a certain extent, this revelation is unsurprising, given that Financial Related Industries have historically been among the first to invest in comprehensive application security programs. Additionally, both sectors have suffered some of the most prominent public breaches in the past, which may have encouraged them to bolster their software security initiatives. The performance of these sectors should be encouraging for all of us who rely on the services they provide; however, room for improvement certainly exists.

> Recommendation(s): Look to organizations with high risk profiles and learn what they have done to implement operating controls in complex environments. It is instructional for lower performing sectors to realize that improvement is possible.

7. Outsourced software is assessed the least, suggesting the absence of contractual security acceptance criteria.

Software identified as Outsourced by submitters accounted for only 2% of the applications in the data set, which was surprising when considering that many enterprises are increasingly relying on offshore development shops as a cost saving measure. With the primary motivation being cost reduction, it is likely that these Outsourcing contracts neglect to define specific security acceptance requirements. This could be one reason why Outsourced software was underrepresented in our data. However, as noted earlier, most applications labeled as Internally Developed actually contained a significant percentage of third-party code, including Outsourced components that were not identified separately.

> Recommendation(s): Do not overlook security requirements when contracting for Outsourced development. When drafting procurement contracts, insist upon the authority to perform independent security testing and set minimum acceptance criteria. This will ensure that you are not charged for rework due to security defects. See the *OWASP Secure Software Contract Annex*[8] and *SANS Application Security Procurement Language*[9] for sample contract language.

[8] www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex

[9] www.sans.org/appseccontract

Software Supply Chain

While people tend to think that software is written from scratch, modern economics and productivity imperatives have long since changed the reality. Today software is truly a composition of code originating from multiple sources across the world.

There are many types of suppliers in the global supply chain including Internal Development teams, Outsourced development partners, Open Source projects, and Commercial Independent Software Vendors. Even individual developers are participating in the enterprise software supply chain by creating popular Apple iPhone or BlackBerry mobile applications or by participating in the fast-growing "crowdsourcing" development model. Cybersecurity professionals in government sometimes refer to this complex and rarely understood supply chain as SOUP, or Software of Unknown Pedigree. To illustrate the point, Veracode observed that between 30% and 70% of all code comprising Internally Developed applications was identifiably from third-parties, most often in the form of Open Source components and Commercial or Outsourced shared libraries and components. Furthermore, there was a "nesting effect" as third-party components themselves often contained other third-party components. For executives, the evidence points to an increasing percentage of software infrastructure and associated liability coming from unknown and unmanaged third-parties.

In this section we examine the security quality of software produced by the software supply chain most often found in organizations. Historically, enterprises and government agencies have focused more on functionality or "fit for purpose" of software acquired from third-parties. This state of affairs is clearly no longer acceptable given today's regulatory and compliance environment and sophisticated hacker community. Only by understanding the various degrees of software security quality produced by supply chain participants can we begin to understand the requirements to change policies and processes, properly manage application risk in organizations, and protect critical software infrastructure.

The evidence points to an increasing percentage of software infrastructure and associated liability coming from unknown and unmanaged third-parties.

Distribution of Application Development by Supplier Type

An analysis of software suppliers enables organizations to identify the application security risk contributed by third-parties versus that created by internal development efforts. This insight is a prerequisite to implementing a proper application risk management strategy given the different corrective actions required to mitigate such risk. In the chart below, of all applications submitted, greater than one third of all applications were developed by third-parties, clearly indicating a heavy reliance on the extended supply chain for business critical applications. It is surprising to note that out of billions of lines of code analyzed by Veracode to date only 2% was attributed to outsourced development partners. Customer feedback suggests this statistic is largely a result of the time required for large enterprises to cycle through existing Master Services Agreements (MSAs) that do not include proper acceptance criteria associated with security quality of their deliverables. We expect this to be a temporary phenomenon as large enterprises begin to heed the advice of research analysts at Gartner[10] and implement security acceptance criteria, terms and conditions in all new outsourced development contracts.

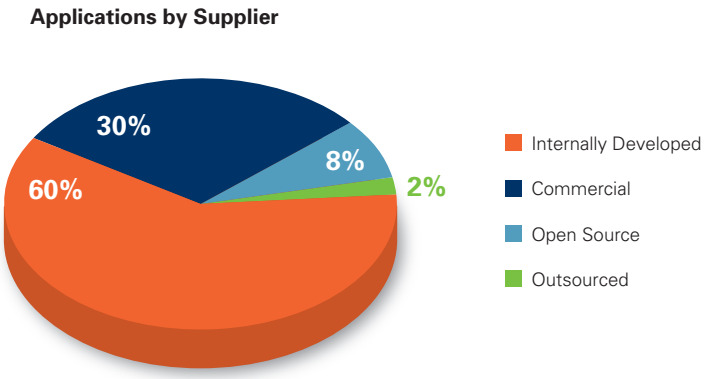

Figure 1: Applications by Supplier

Distribution of Application Business Criticality by Supplier Type

An analysis of the Application Business Criticality by Supplier Type allows organizations to assess the relative importance of application security risk and the degree to which it is controllable. For example, the decision to outsource only non-web facing applications of low business criticality is likely to be a more controllable and manageable risk than outsourcing highly business critical web-facing applications given the lack of security acceptance criteria in existing outsourcing contracts and the variability of secure coding skills across outsourcing partners.

[10] blogs.gartner.com/neil_macdonald/2010/01/14/more-application-security-goodness-from-owasp

The distribution of Application Business Criticality by Supplier Type is depicted below. The distribution clearly illustrates that high business criticality is not a major determinant in keeping development projects "in-house." More than 30% of applications rated as being of Very High or High business criticality were sourced from Commercial software vendors. It can be inferred that features, functionality and time-to-market are likely the primary drivers in the decision to develop applications internally versus procure them from third-parties. This in turn should increase the importance of applying uniform application verification practices across Internally Developed and third-party applications that are deemed Very High or High Business Criticality. Business Criticality was specified by the organization operating the software.

More than 30% of applications rated as being of Very High or High business criticality were sourced from Commercial software vendors.

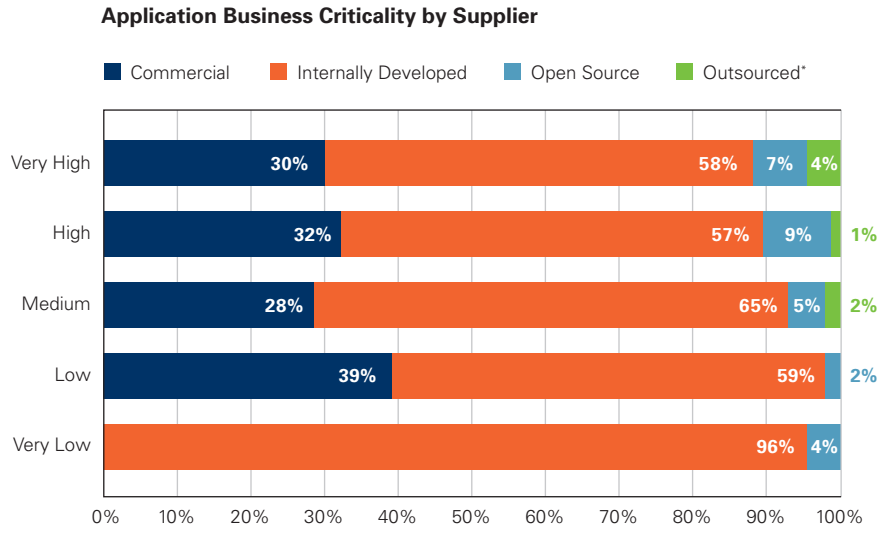

Figure 2: Application Business Criticality by Supplier

Distribution of Application Type by Supplier Type

An analysis of the Application Type by Supplier Type provides a more detailed understanding of security risk contributed by supplier and their relative capabilities to produce secure code for a particular language and environment. This information should be an important determinant in the decision-making process on where to source business critical applications if the decision is to purchase Commercial software or outsource development to a third-party.

Except for very few Open Source .NET projects the distribution of C/C++, Java and .NET languages was significant. The distribution of web and non-web applications was also material for each type of supplier, with Internally Developed code skewing most heavily towards web applications. Both the distribution of languages and web and non-web applications across suppliers reinforces the importance of consistent testing practices across the application portfolio.

The distribution of languages and web and non-web applications across suppliers reinforces the importance of consistent testing practices across the application portfolio.

Supplier Application Profiles

	C/C++	Java	.NET	Web	Non-Web
Internally Developed	22%	53%	25%	73%	27%
Commercial	44%	35%	21%	44%	56%
Open Source	45%	54%	1%	30%	70%
Outsourced* (Low sample size)	0%	67%	33%	60%	40%

Table 1: Supplier Application Profiles

Distribution of Security Quality and Remediation Efforts by Supplier Type

An analysis of the Security Quality and Remediation Efforts by Supplier Type provides important insights into the secure coding skills of suppliers and their relative effectiveness in remediating security vulnerabilities. This information can prove to be valuable in many ways including enabling more accurate total cost calculations of a particular software development project, improved ability to hit ship or deployment dates, more accurate scoping accuracy of security testing phase, or better cost containment of outsourcing contracts by controlling "rework charges" associated with code changes related to fixing security vulnerabilities.

58% of all applications were deemed to have "unacceptable" security quality.

The illustration below depicts Supplier Performance on First Submission as measured by the Veracode risk adjusted verification methodology. When calculated as a percentage of total applications submitted 58% of all applications were deemed to have "unacceptable" security quality upon first submission. Commercial suppliers achieved lower scores than both Internally Developed and Open Source applications. It is not surprising given that most organizations do not have developers trained in application security and secure coding principles or have not implemented a secure software development lifecycle.

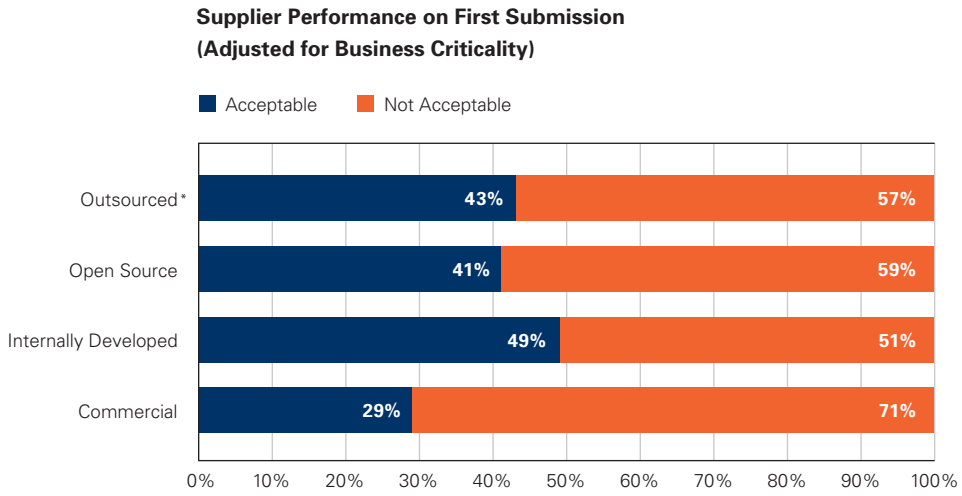

Figure 3: Supplier Performance on First Submission (Adjusted for Business Criticality)

In the following chart it is interesting to note that Open Source applications had an equivalent percentage of Very High severity vulnerabilities (Buffer Overflows, Numeric Errors) but a higher percentage of High severity vulnerabilities (SQL Injection). This effect is not reflected in the Veracode rating above due to the adjustment for business criticality. A larger percentage (25% vs. 16%) of Commercial software was submitted at the highest business criticality, when compared to Open Source.

Commercial remediation cycles must often fit within more formal software and patch release cycles. The fastest at security remediation were Open Source project teams.

Vulnerability Severity by Supplier

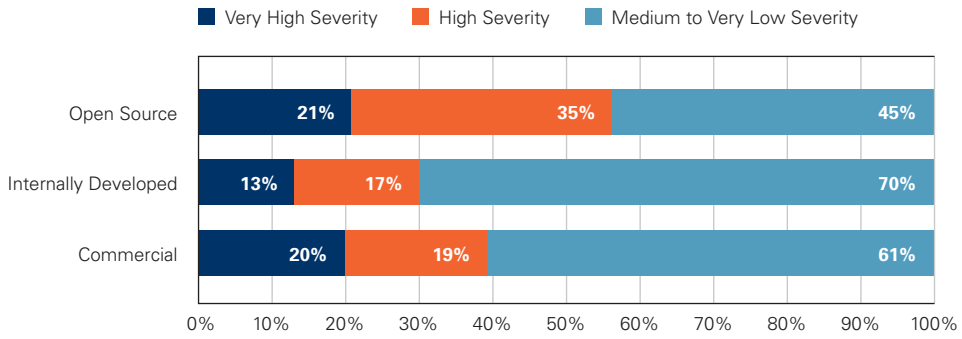

FIgure 4: Vulnerability Severity by Supplier
(Outsourced omitted due to small sample size.)

The following illustration depicts Remediation Performance by Supplier Type. It is interesting to note that the average time to remediate was 59 days. Commercial took the longest time to remediate at 82 days. Commercial remediation cycles must often fit within more formal software and patch release cycles. Open Source project teams were the fastest at security remediation. The quality of the remediation efforts was also superior with Open Source teams, taking only 1.1 resubmissions on average to confirm the security fix was properly implemented. Internally Developed projects had the second shortest time to remediate but took the most resubmissions to properly implement the security fixes and achieve compliance with desired security quality. One of the most encouraging aspect of this information is that developers are generally very capable of quickly fixing security vulnerabilities when they are accurately pointed out in their code base. It appears far more difficult to write secure code and find security vulnerabilities than it does to remediate security vulnerabilities once discovered.

The very encouraging aspect of this information overall is that developers are very capable of quickly fixing security vulnerabilities when they are accurately pointed out in their code base.

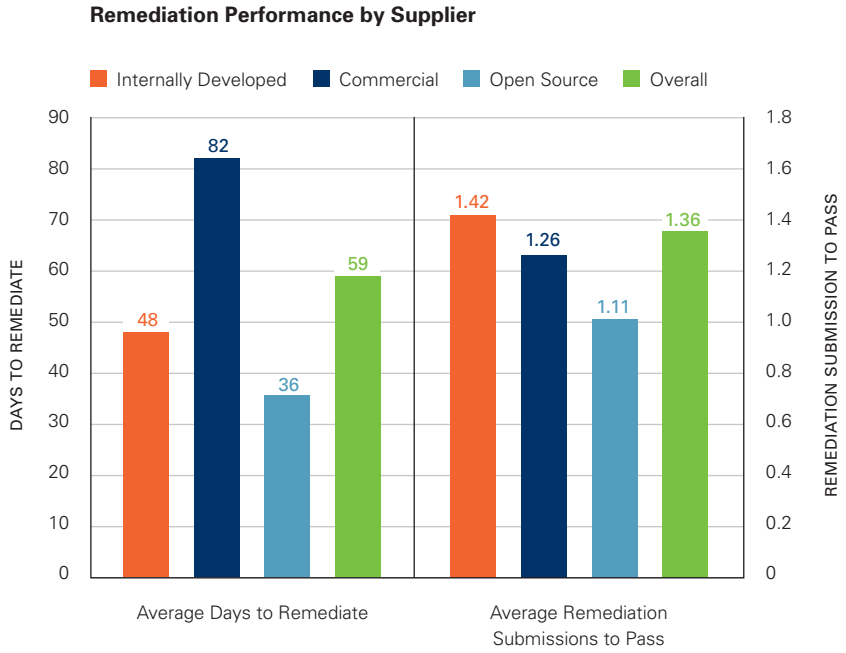

Figure 5: Remediation Performance by Supplier
(Outsourced omitted due to small sample size.)

Distribution by Ability to Meet Security Compliance Policy by Supplier

An analysis of a supplier's ability to meet a specific security compliance policy is useful information when implementing security acceptance criteria during the software acquisition process. For both commercial and outsourcing agreements this information may be used to structure appropriate terms and conditions and to ensure security quality of the deliverable while simultaneously managing contract costs.

The following chart examines suppliers ability to deliver applications as measured by compliance against industry standard lists of dangerous vulnerability types, including the OWASP Top 10 (2007) and the CWE/SANS Top 25 (2009). An application was labeled Not Acceptable if it contained any vulnerabilities defined in the standard lists.

Open source and Commercial software fared much better than Internally Developed code against OWASP and SANS standards. The cause for this may be that Commercial code is subjected to a greater level of scrutiny by purchasing enterprises or may be subject to regulations such as PCI. Open Source code benefits from the community network effect where issues are identified and fixed faster. Open Source code has also benefited from Commercial source code scanning technologies being made available to them for free.

Open Source and Commercial software fared much better than Internally Developed code against OWASP and CWE/SANS standards.

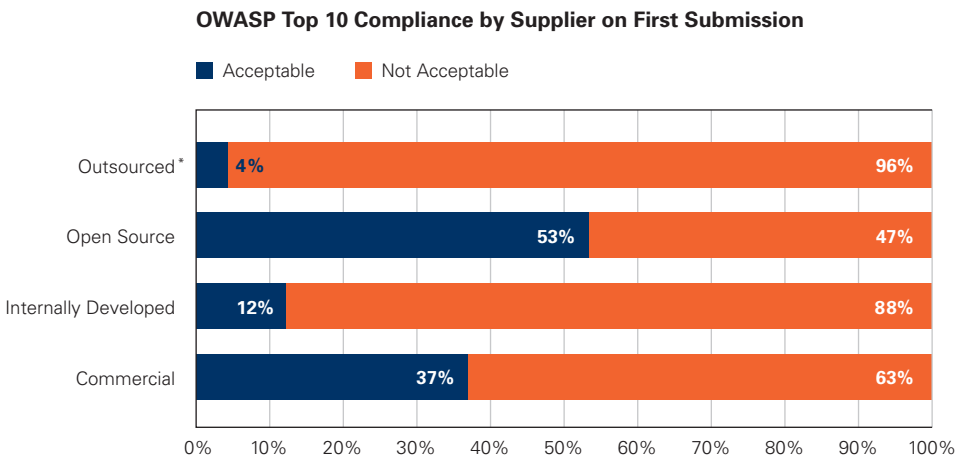

Figure 6: OWASP Top 10 Compliance by Supplier on First Submission

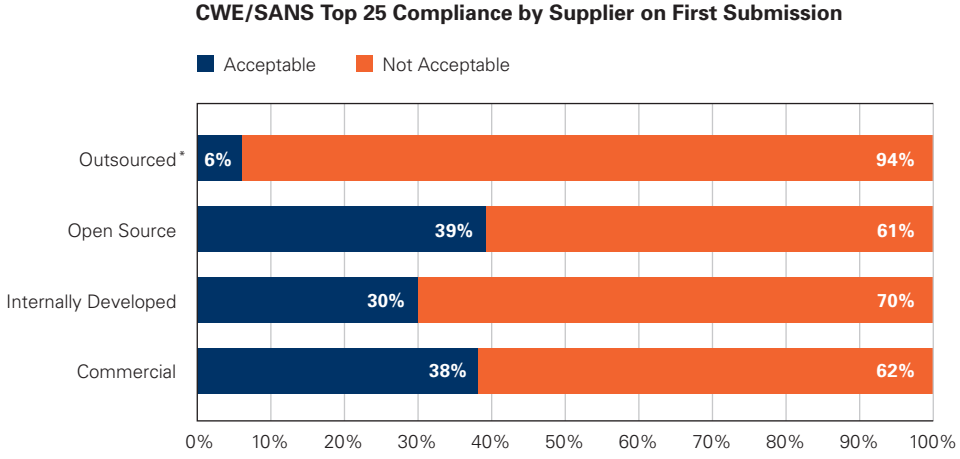

Figure 7: CWE/SANS Top 25 Compliance by Supplier on First Submission

Distribution of Most Common Security Vulnerabilities by Supplier

An analysis of the Most Common Security Vulnerabilities by supplier is extremely helpful in determining the specific secure coding strengths and weaknesses. It can influence targeted education or other corrective actions or improve accountability with a particular supplier. The following chart identifies the most commonly occurring vulnerabilities across suppliers. The following table summarizes the top 15 vulnerabilities in order of prevalence, where the percentage indicates the frequency of occurrence for that type of Supplier.

Vulnerability Distribution by Supplier

Internally Developed		Commercial		Open Source		Outsourced*	
Cross-site Scripting (XSS)	40%	Cross-site Scripting (XSS)	22%	Cross-site Scripting (XSS)	41%	CRLF Injection	36%
Information Leakage	24%	Information Leakage	19%	CRLF Injection	21%	Cross-site Scripting (XSS)	16%
CRLF Injection	8%	Numeric Errors	12%	Information Leakage	13%	Information Leakage	14%
Cryptographic Issues	6%	Buffer Overflow	9%	Error Handling	5%	Directory Traversal	11%
Buffer Overflow	4%	Cryptographic Issues	7%	SQL Injection	5%	Cryptographic Issues	9%
Directory Traversal	4%	Error Handling	7%	Directory Traversal	3%	Time and State	6%
SQL Injection	4%	Directory Traversal	6%	Cryptographic Issues	2%	Credentials Mgmt	3%
Time and State	2%	CRLF Injection	5%	Encapsulation	2%	API Abuse	3%
Error Handling	2%	SQL Injection	3%	Numeric Errors	1%	Encapsulation	1%
Numeric Errors	1%	Buffer Mgmt Errors	3%	Session Fixation	1%	SQL Injection	<1%
Potential Backdoor	1%	Time and State	2%	Time and State	1%	Insufficient Input Validation	<1%
Encapsulation	1%	Potential Backdoor	1%	Buffer Overflow	1%	Error Handling	<1%
Credentials Mgmt	1%	Credentials Mgmt	1%	API Abuse	1%	Numeric Errors	<1%
API Abuse	1%	Dangerous Functions	<1%	Dangerous Functions	<1%	OS Command Injection	<1%
Buffer Mgmt Errors	1%	API Abuse	<1%	Credentials Mgmt	<1%	Session Fixation	<1%

Table 2: Vulnerability Distribution by Supplier

Internally Developed software showed comparable vulnerability prevalence rankings to the sample overall, probably because Internally Developed software accounted for the majority of the applications assessed (60%).

The relatively high prevalence of numeric errors, buffer overflows, and buffer management errors in Commercial Software can be attributed to the higher concentration of C and C++ based applications.

The relatively high prevalence of numeric errors, buffer overflows, and buffer management errors in the Commercial Software applications can be attributed to the higher concentration of C and C++ based applications in this group, as noted above. That these vulnerability categories were so prevalent among Commercial software, and that C and C++ are so widely used, suggest that any third-party risk management program should include coverage of Commercial C and C++ software as well as bytecode (Java and .NET).

Another interesting point was the relatively higher prevalence of CRLF (Carriage Return/Line Feed) Injection vulnerabilities among Open Source applications, compared to Information Leakage. A leading type of CRLF Injection vulnerabilities was Log Injection, and the leading type of Information Leakage was an Error Message Information Leak; this suggests that Open Source applications may be more likely to direct error messages to a system log than to the user. This practice mitigates information leakages but increases log-related security risks.

Finally in this section, the relative prevalence of Potential Backdoors was low, but their existence at all is noteworthy. For the purposes of this report, Potential Backdoors include hardcoded passwords, logic and time bombs, the presence of anti-debugging code, rootkit-like behaviors, and call hiding (the practice of masking a call to a potentially malicious routine via indirection). While there may be legitimate uses for some of these techniques, malicious use mandates close inspection if they are found in the course of an assessment. Some Potential Backdoors were debugging and support features that the developer intended but put the customer at risk.

The presence of Potential Backdoors was low but their existence at all is noteworthy.

For security professionals and development teams it is helpful to understand what types of vulnerabilities are most common by software supplier. There are a few differences that stand out in the table above, however the main point is that vulnerabilities are more related to the type of application and language than the supplier.

Security of Applications

The previous section presented information from the Software Supplier perspective in an attempt to help enterprises change policies and processes, properly manage application risk in organizations, and protect critical software infrastructure. It also proposed mechanisms to identify and enforce that acceptable levels of security quality are met by internal and external development teams.

It is important to note that no software will ever be perfectly secure. Understanding the nature of software vulnerabilities themselves and how they contribute to enterprise security risk once they reside in an application regardless of how they got there helps us on the path of maturing application security as a discipline. Building in high levels of "manufacturing level" security quality is eminently doable, and buyer and supplier expectations on acceptable security quality should be no different than that for other complex products such as automobiles that have material consequences to security failures. This section of the report is designed to provide vulnerability level information to assist with implementing mitigating controls and proper application risk management strategies.

As background, software vulnerabilities are the attack points in applications used by hackers to compromise a system. Different types of applications have different attack points that can be used by a hacker in different ways. For example, web applications have different attack surfaces than non-web applications such as desktop software or databases. Additionally, vulnerabilities can vary significantly by programming language and platforms such as the Windows versus BlackBerry operating systems. It is also possible for applications in different industries to have different vulnerabilities based on the secure coding skills of the engineering population serving those industries (e.g. Financial Services versus Retail) and the sophistication of their software development practices or central security teams. In this section we will explore which types of applications are being analyzed by Veracode's cloud-based services platform, what vulnerabilities are being discovered, and the subsequent implications for application risk management programs.

Distribution of Application by Type

All applications analyzed by Veracode are inventoried and classified according to a profile which includes key characteristics such as whether the application is web-facing, its language and platform, and the industry of the organization submitting it. All applications were deemed important enough to warrant a security review given the security risk for their business and the customer, partner, employee data, and/or financial transactions they supported. Not surprisingly, the majority of applications analyzed were web (60%) versus non-web (40%). This reflects the justified concerns of submitters that web application risks are high and attack vectors well known and constantly being exploited (e.g. SQL Injection, Cross-site Scripting).

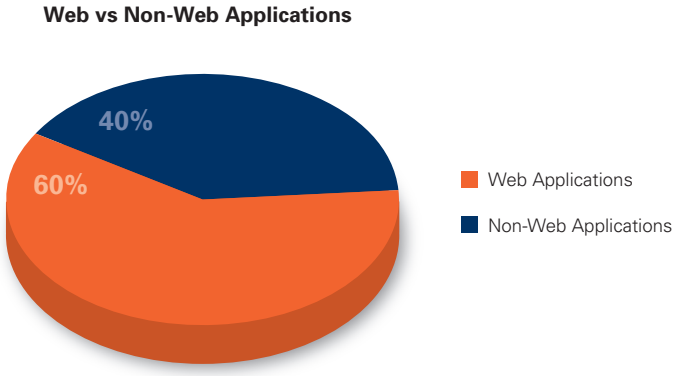

Figure 8: Web vs Non-Web Applications

Distribution of Applications by Language

An analysis of the Distribution of Applications by Language is a useful indicator of the enterprise market's preference for developing applications of a certain type and platform. For instance, it is useful for peer benchmarking purposes to understand similarities or differences in enterprise development for highly critical web-facing applications. Do they prefer Java or .NET? If so, what is driving the preference and what are the implications for developers from a training perspective? Understanding the distribution of applications by language is also important in that it is a leading indicator and reasonable proxy for the ever-changing attack surface of the world's software infrastructure. As we have seen, attackers tend to focus on more broadly deployed platforms and increase their activity as platform gains market share. For example, Apple's recent success in topping 10% market share in 2009 and growing iPhone market share from 11.2% in the fourth quarter of 2008 to 16% in the fourth quarter of 2009, has seen rapid rise in attacks on both the Mac and iPhone platforms.[11] The following graph shows the distribution of applications submitted for analysis. These languages represent the most popular programming languages used for web, network, server, PC, device, and mobile software applications.

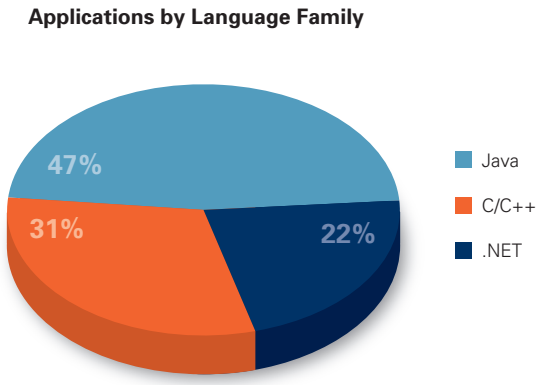

Figure 9: Applications by Language Family

[11] www.infosecurity-us.com/view/6841/security-and-malware-threats-to-mac-and-apple-products-are-on-the-rise-/

Distribution of Applications by Vulnerability Type

Once an understanding has been gained regarding the programming languages used to deploy business critical infrastructure, it is important to understand the types and prevalence of the vulnerabilities across Application Type and Languages. This is an important next step in determining the overall threat exposure, the ease of exploit of vulnerabilities and the ultimate implementation of corrective actions to mitigate risk. The charts below depict top vulnerability categories by Vulnerability Prevalence across web and non-web applications and rank of the Top 15 vulnerabilities. The rows highlighted in red are vulnerability categories that also appear in the CWE/SANS Top 25 or OWASP Top 10 standards. There is considerable overlap between Veracode findings and these industry standards further confirming the relevance of these vulnerability categories as top areas of security weakness to focus on for enterprises.

Two types of vulnerability prevalence are presented below. The first is Vulnerability Frequency, which illustrates the share of the total vulnerabilities discovered. The second is Affected Application, which shows the percentage of applications containing one or more of the vulnerabilities in each category.

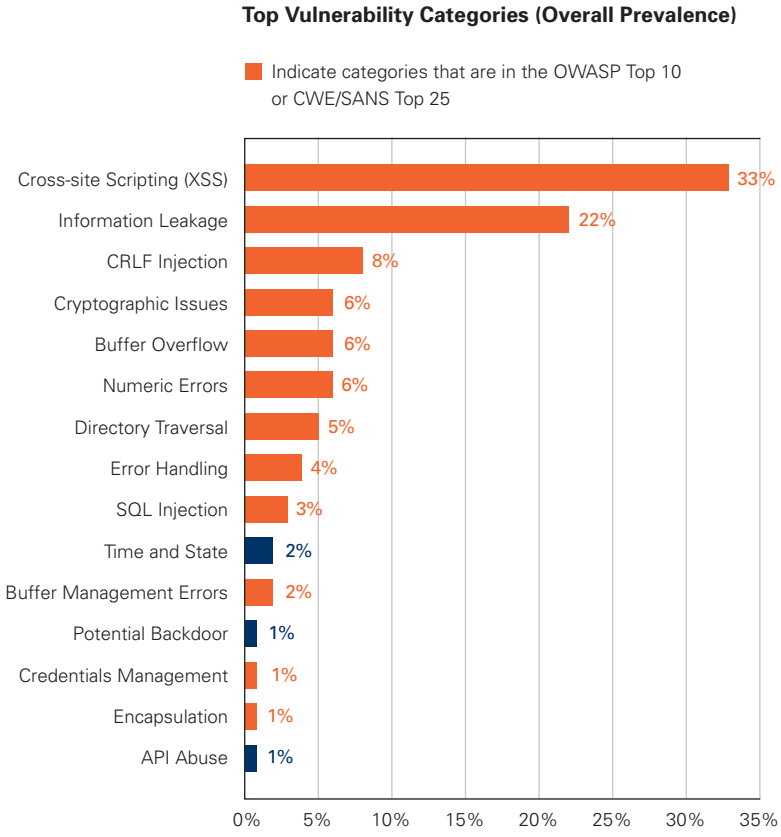

Figure 10: Top Vulnerability Categories (Overall Prevalence)

Cross-site Scripting prevalence remains high despite the wide availability of libraries intended to eliminate the risk via output encoding.

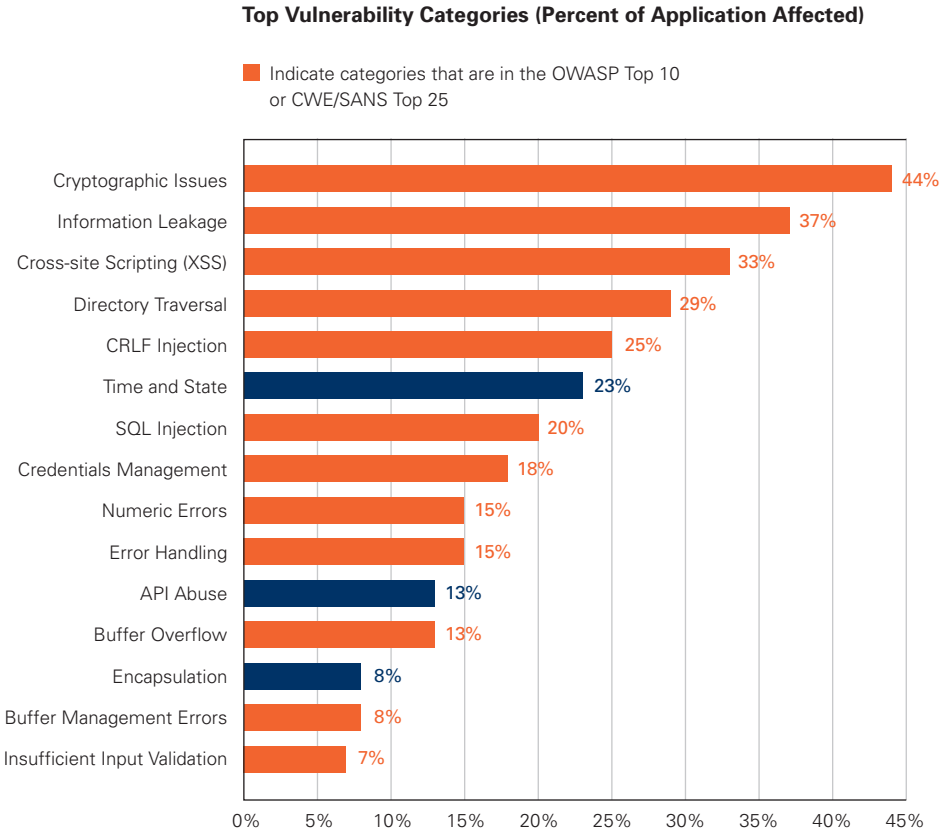

Figure 11: Top Vulnerability Categories (Percent of Application Affected)

Cross-site Scripting, which was the most prevalent vulnerability category by Frequency and the third most prevalent by Affected Applications, is an illustration of the challenges of writing secure code. After years of focus on Cross-site Scripting it remains a serious security threat. The continued high prevalence of the vulnerability is a factor both of the pervasive nature of the problem and the increasing threat landscape (e.g. higher use of dynamic content generation). The prevalence remains high despite the wide availability of libraries intended to eliminate the risk via output encoding. It may suggest that developers are focusing more on writing strong functional code and hitting ship or deployment dates. It may also mean that development practices are relatively immature as they relate to security testing and that proper threat modeling and security testing processes have either not been implemented or are failing to detect common vulnerabilities for other reasons.

The most prevalent vulnerability category by Affected Application was cryptographic issues. These included insufficient entropy, plain text storage of sensitive data, use of hardcoded cryptographic keys, and use of algorithms with inadequate encryption strength. Cryptographic issues are often not well understood by developers, so developer education is an important component of mitigating this risk category. Clearly this is an important and negative trend given the focus of hackers in accessing confidential data.

Cryptographic issues are often not well understood. Developer education is an important component of mitigating this risk category.

Both prevalence rankings should be considered when evaluating the seriousness of a risk category. For instance, SQL Injection has a relatively small share of the overall vulnerability population (3%), but is found in 20% of all applications assessed. Some less severe vulnerability categories may occur hundreds of times within an application, while other more severe vulnerabilities only appear a few times.

Vulnerabilities by Language Distribution

The table below presents the most prevalent categories (by share of total vulnerabilities discovered) based on language family. One interesting aspect was the unusually high frequency of Cross-site Scripting vulnerabilities in .NET applications. This can be attributed to developer use of .NET controls that do not automatically encode output.

Vulnerability Distribution by Language

Java		C/C++		.NET	
Cross-site Scripting (XSS)	34%	Buffer Overflow	32%	Cross-site Scripting (XSS)	64%
Information Leakage	31%	Numeric Errors	28%	Cryptographic Issues	13%
CRLF Injection	12%	Error Handling	18%	Directory Traversal	8%
Cryptographic Issues	6%	Buffer Mgmt Errors	8%	CRLF Injection	5%
Directory Traversal	5%	Potential Backdoor	6%	Information Leakage	5%
SQL Injection	4%	Cryptographic Issues	3%	SQL Injection	1%
Time and State	3%	Directory Traversal	1%	Insufficient Input Validation	1%
Credentials Mgmt	1%	Dangerous Functions	1%	Credentials Mgmt	1%
Encapsulation	1%	Time and State	<1%	Numeric Errors	1%
API Abuse	1%	Untrusted Search Path	<1%	Error Handling	1%

Table 3: Vulnerability Distribution by Language

Distribution of Vulnerability Type by Application Input Vector

An analysis of Vulnerabilities by Input Vector is important as one way to determine the ease of exploit by a potential attacker. For example, web-based vectors are the highest risk because they are directed over the network and potentially from anonymous actors. File and database vectors, in comparison, require an attacker to first get attack data onto the local file system or into the database. These types of attacks typically require exploiting a second vulnerability to be successful and require higher degrees of attacker sophistication.

Web-based Input Vectors are the highest risk. File and database vectors, in comparison, typically require exploiting a second vulnerability to be successful and require a higher degree of attacker sophistication.

Veracode's static binary analysis identifies code paths that are vulnerable to data that may be tainted by an attacker; some of these types of attacks include cross-site scripting, path manipulation, and SQL injection. The analysis draws a distinction between code vulnerabilities that can be triggered via data from a web request and those triggered with data coming from other vectors such as data stored in a file or persisted in the database.

Flaw Type by Input

Flaw Type	Web-Based Input Vector	Non-Web-Based Input Vector
Cross-site Scripting (XSS)	60.4%	39.5%
Path Manipulation	33.0%	67.0%
SQL Injection	54.2%	45.8%

Table 4: Flaw Type by Input

An understanding of application vulnerabilities by Industry Group is useful given that skill sets and often language and platform usage vary by industry and can contribute to overall security risk exposure. The following table shows vulnerability distributions by Industry Group. As you can see, software developed by the Software industry reflects the prevalence of an unmanaged programming language such as C/C++. Unmanaged code is vulnerable to additional categories of vulnerabilities such as buffer overflow and numeric errors. These categories have higher prevalence in the software-related industry group.

Vulnerability Distribution by Industry

Finance-related		Software-related		Government	
Cross-site Scripting (XSS)	35%	Cross-site Scripting (XSS)	19%	Cross-site Scripting (XSS)	53%
Information Leakage	21%	Information Leakage	17%	Information Leakage	12%
CRLF Injection	5%	Numeric Errors	11%	CRLF Injection	6%
Cryptographic Issues	5%	Buffer Overflow	8%	Buffer Mgmt Errors	4%
Directory Traversal	3%	Cryptographic Issues	6%	SQL Injection	3%
SQL Injection	2%	Error Handling	6%	Cryptographic Issues	2%
Buffer Overflow	2%	Directory Traversal	5%	Numeric Errors	2%
Time and State	2%	CRLF Injection	4%	Encapsulation	1%
Encapsulation	1%	SQL Injection	3%	Directory Traversal	1%
Potential Backdoor	1%	Buffer Mgmt Errors	3%	Credentials Mgmt	1%
Credentials Mgmt	1%	Time and State	2%	OS Command Injection	1%
Error Handling	1%	Potential Backdoor	1%	Time and State	1%
Numeric Errors	<1%	Credentials Mgmt	1%	Buffer Overflow	1%
Insufficient Input Validation	<1%	API Abuse	<1%	API Abuse	<1%
API Abuse	<1%	Dangerous Functions	<1%	Insufficient Input Validation	<1%

Table 5: Vulnerability Distribution by Industry

Industry group definitions

The Finance-related industries group combines applications from the Financial Service, Insurance, and Banking industries (self identified); the Computer-related industries category combines applications from the Computer Software, Computer Services, and Security Products and Services industries (self identified); Government is unclassified US federal, state, and local government agencies (self-identified).

Cross-site Scripting and Information Leakage ranked as the top two vulnerability categories across software from all industries.

Distribution of Application Security Performance by Business Criticality

Many submitters use Veracode's risk adjusted benchmark to compare Internally Developed versus externally sourced applications. Grounded in industry standards (CWE, CVSS and NIST) it has a sliding scale, requiring applications of higher business criticality (assurance levels) to have a higher degree of security quality. This pragmatic approach allows organizations to optimize their remediation effort and expense intelligently across their portfolio rather than spend excessively to bring less business critical applications up to the same standard as highly critical applications.

Only 24% of applications designated "Very High" critically were found to have acceptable security on first submission.

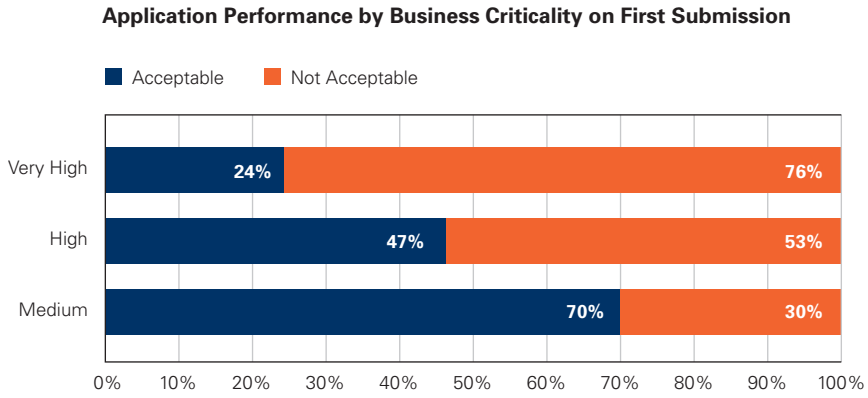

Figure 12: Application Performance by Business Criticality on First Submission

Given the higher security required for the most business critical applications, it is not surprising that only 24% of applications designated "Very High" were found to have acceptable security on first submission. Similarly, we would expect to see a significantly higher percentage of Medium criticality to have adequate security. This proved to be true with 70% of all such applications passing upon first submission.

We explored the impact of industry on the application ratings above, given the potential that some industry verticals could have more security-aware development teams and more formal development practices. Our analysis proved to be surprising as Software-related Industries fared the worst with only 34% being deemed acceptable, while Financial and Government applications were at the top with more than half acceptable upon first submission. The information below further highlights the need for enterprises to address third-party security risk should they choose to purchase or outsource significant portions of their application development efforts.

Software-related Industries fared the worst and Financial-related and Government Industries were at the top in terms of acceptable security on first submission.

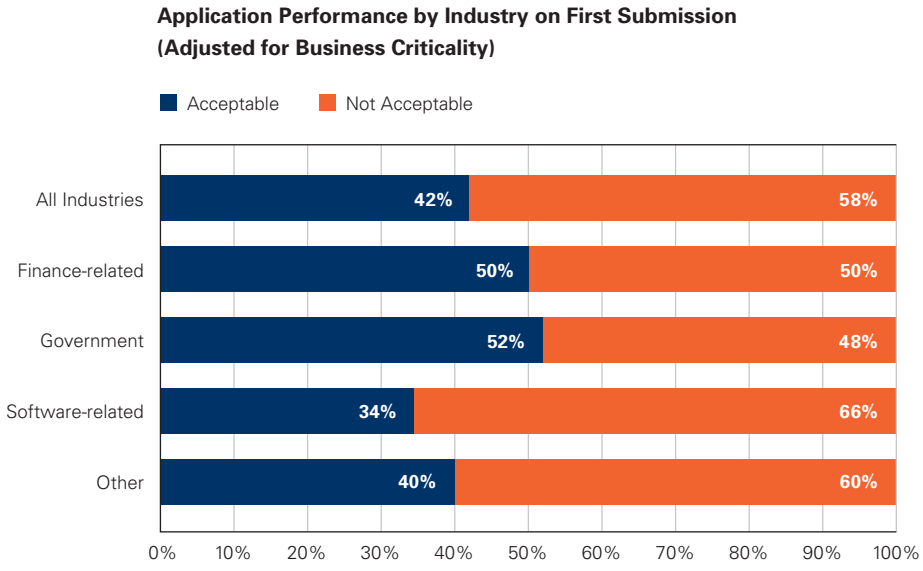

Figure 13: Application Performance by Industry on First Submission (Adjusted for Business Criticality)

Application Threat Space Trends

Without context, data and statistics have limited meaning. Context is especially important for application risk management. Throughout this report we have added the context of application origin, platform, language, and industry to our vulnerability data. Now let's add the most important context for software security: the Application Threat Space, translated as "connecting the code base to the threat space."

Not easily picked out in the data above is evidence of new attack trends that leverage vulnerabilities in software to breach corporate data. Software vulnerabilities like those found by Veracode are being used in new and creative ways by attackers for targeted attacks. This trend will continue. However, in addition to being used to compromise systems and access data, software vulnerabilities are being used to breach perimeters.

In addition to being used to compromise systems and access data, software vulnerabilities are being used to breach perimeters.

For example, it is telling that one third of all web applications analyzed had SQL Injection vulnerabilities. This easy-to-find and easy-to-prevent vulnerability is still a significant problem. SQL Injection has always been a favorite of attackers because of the ease of exploit, the proliferation of web applications, and the vulnerability's locality to valuable data. Indeed, many reports say it is vulnerability #1 for data breaches.[12] A new trend that was revealed in the Heartland Payment Systems credit card data breach is the widespread use of SQL Injection in a high profile attack as a way to bypass perimeter controls. This is a new risk to organizations that requires all internet facing web applications be free from SQL Injection vulnerabilities, not just the web applications that access sensitive data.

The Google attack of December 2009 is another example. Attackers leveraged a zero day vulnerability in Internet Explorer to breach the firewall at Google. Commercial software on an employee workstation with outbound access to the Internet was used as a bridge to the internal network. Attackers needn't exploit a web browser vulnerability. They could have taken advantage of any vulnerable software installed on the workstation that received data from the Internet. Document editors and viewers, browser plugins, media players, chat programs, games, even Microsoft Paint,[13] could all have been the culprit.

The overarching trend here is organizations cannot only worry about their critical server applications—the applications processing the valuable transactions and holding valuable data. Attackers are reaching their goal by using vulnerabilities in any software that they can send data to: web apps, desktop apps and soon mobile apps.

Organizations cannot only worry about their critical server applications; the ones processing the valuable transactions and holding valuable data. Attackers are using vulnerabilities in any software which they can send data to in order to reach their goal: web apps, desktop apps and soon mobile apps.

[12] UK Security Breach Investigations Report, Trustwave's Global Security Report for 2010, and Verizon's 2009 Anatomy of a Data Breach Report

[13] Vulnerability in Microsoft Paint Could Allow Remote Code Execution

At Veracode we are now starting to collect data on mobile application vulnerabilities, both coding errors and intentional malicious code. Given the trends we have seen in the last few years of attackers utilizing end user desktop software as a bridge to enable data theft it is likely that mobile apps will increasingly serve this purpose also. The supply chain for mobile apps is seemingly more controlled. Many mobile applications are delivered from platform supplier app stores. But unfortunately these app stores are not performing security testing so this is likely a false sense of low risk. In addition most mobile apps are installed by end users with little management by central IT teams. This is a recipe for the type of malware and spyware that infested the desktop over the past 10 years.

Another Threat Watch area is social networking applications. This is a rich frontier for phishing and spearphishing attacks leveraging the inherent trust users give to information, web links and other data they receive through these platforms. Most social network platforms are extensible and allow developers to build small applications known as plug-ins that tightly integrate into the social networking user interface and user data. Vulnerabilities in these applications and intentional malicious code could prove harmful to individuals and organizations.

The threat environment coupled with the state of application security detailed in our report should be a call to action for any organization that builds its own applications or purchases software. It is your entire application inventory that is putting your organization at risk. Veracode recommends instituting internal controls that require software security be measured before an application is deployed or purchased. A quality bar can then be put in place and failing software can be sent back to the developers for remediation.

Without a change in the way organizations are protecting themselves from the exploitation of software vulnerabilities we don't see progress being made. Patching quicker and updating antivirus and IDS/IPS signatures faster is not stemming the tide. The threat space moves too quickly.

Without a change in the way organizations are protecting themselves from the exploitation of software vulnerabilities, progress won't be made. Patching quicker and updating antivirus and IDS/IPS signatures faster is not stemming the tide. The Application Threat Space moves extremely quickly. Veracode recommends keeping the layered defenses but shifting some resources to fixing the root cause of data compromise which are without doubt the software vulnerabilities themselves.

Addendum

Methodology

About Veracode's Assessment and Rating Methodologies

The Veracode SecurityReview uses static and dynamic analysis (for web applications) to inspect executables and identify security vulnerabilities in applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security vulnerabilities. The static binary analysis engine creates a model of the data and control flow of the binary executable; the model is then verified for security vulnerabilities using a set of automated security scans. Dynamic analysis uses an automated web scanning technique to detect security vulnerabilities in a web application at runtime. Once the automated process is complete, a security analyst verifies the output to ensure the lowest false positive rates in the industry. The end result is an accurate list of security vulnerabilities for the classes of automated scans applied to the application.

About Software Assurance Levels

The foundation of the Veracode rating system is the concept that higher assurance applications require higher security quality scores to be acceptable risks. Lower assurance applications can tolerate lower security quality. The assurance level is dictated by the typical deployed environment and the value of data used by the application. Factors that determine assurance level include reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations.

About the Data Set

The data represents 1,591 applications submitted for analysis by large and small companies, commercial software providers, open source projects, and software outsourcers. An application was counted only once even if it was submitted multiple times as vulnerabilities were remediated and new versions uploaded. The report contains findings about applications that were subjected to static, dynamic, or manual analysis through the Veracode SecurityReview® Platform. The report considers data that was provided by Veracode's customers (application portfolio information such as assurance level, industry, application origin) and information that was calculated or derived in the course of Veracode's analysis (application size, application compiler and platform, types of vulnerabilities, Veracode rating).

In any study of this size there is a risk that sampling issues will arise because of the nature of the way the data was collected. For instance, it should be kept in mind that all the applications in this study came from organizations that were motivated enough about application security to engage Veracode for an independent assessment of software risk. Care has been taken to only present comparisons where a statistically significant sample size was present

About the Findings

Unless otherwise stated, all comparisons are made on the basis of the count of unique application builds submitted and rated.

Assurance Level Definitions

Veracode's Business Criticality designations are based on the Assurance Level standard developed by NIST, as detailed below:

Business Criticality	Description
Very High	Mission critical for business/safety of life and limb on the line
High	Exploitation causes serious brand damage and financial loss with long term business impact
Medium	Applications connected to the Internet that process financial or private customer information
Low	Typically internal applications with non-critical business impact
Very Low	Applications with no material business impact

Table 6: Business Criticality Descriptions
Source: U.S. Government. OMB Memorandum M-04-04; NIST FIPS Pub. 199

Very High (AL5)

This is typically an application where the safety of life or limb is dependent on the system; it is mission critical the application maintain 100% availability for the long term viability of the project or business. Examples are control software for industrial, transportation or medical equipment or critical business systems such as financial trading systems.

High (AL4)

This is typically an important multi-user business application reachable from the Internet and is critical that the application maintain high availability to accomplish its mission. Exploitation of high assurance applications cause serious brand damage and business/financial loss and could lead to long term business impact. Exploitation is a result of a breach in any two impact categories of confidentiality, integrity and availability of the application.

Medium (AL3)

This is typically a multi-user application connected to the Internet or any system that processes financial or private customer information. Exploitation of medium assurance applications typically result in material business impact resulting in some financial loss, brand damage or business liability. Exploitation is a result of a breach in confidentiality, integrity or availability of the application. An example is a financial services company's internal 401K management system.

Low (AL2)

This is typically an internal only application that requires low levels of application security such as authentication to protect access to non-critical business information and prevent IT disruptions. Exploitation of low assurance applications may lead to minor levels of inconvenience, distress or IT disruption. An example internal system is a conference room reservation or business card order system.

Very Low (AL1)

Applications that have no material business impact should its confidentiality, data integrity and availability be affected. Code security analysis is not required for this assurance level and security spending should be directed to other higher level assurance applications.

VERACODE

Software Security Simplified

Veracode, Inc.
4 Van de Graaff Drive
Burlington, MA 01803

Tel +1.781.425.6040
Fax +1.781.425.6039

www.veracode.com

SSSR/0310

ABOUT VERACODE

Veracode is the world's leader in cloud-based application risk management. Veracode SecurityReview is the industry's first solution to use patented binary code analysis, dynamic web assessments, and partner or Veracode delivered manual penetration testing, combined with developer e-learning and access to open source security ratings to independently assess and manage application risk across internally developed applications and third-party software without exposing a company's source code. Delivered as a cloud-based service, Veracode provides the simplest, most complete, and most accurate way to implement security best practices, reduce operational cost and comply with internal security policies or external standards such as OWASP Top 10, CWE/SANS Top 25 and PCI.