Report

Cyber risk report 2013

Table of contents

Introduction

Welcome to the HP Security Research Cyber Risk Report 2014. In this report we provide a broad view of the vulnerability and threat landscape, ranging from industry-wide data down to a focused look at different technologies, including open source, web, and mobile. The goal of this report is to provide security information that can be used to understand the vulnerability landscape and best deploy resources to minimize security risk.

With the cost of cyber crime continuing to rise at a worrying rate[1] and several organizations calling attention to the observed increase in the use of exploits in the wild,[2] the report this year focuses on specific areas of the attack surface; the technologies that define it, and vulnerabilities and actors that drive how it is abused.

The attack surface is a useful term to describe the sum of areas that are vulnerable to compromise by attackers, including technologies, people, and processes. While classically used to describe software, in this report we refer to the attack surface in a more holistic manner. What factors comprised the attack surface of an organization in 2013? The answer to that question is integral to understanding the threat landscape, prioritizing the risks that face an organization, and deciding on the appropriate security investments with which to respond.

To this end, the report takes a close look at Java's contribution to the attack surface, summarizing the results of studies into the security concerns plaguing mobile and open source software. We use the cyber attacks against South Korea in 2013 as a case study to illuminate security best practices in the current threat landscape. And, most importantly, we analyze the implications of these findings on organizations and provide recommendations for how to be more secure.

It's important to remember that security isn't a box that can be checked—it's an ongoing process of gathering and sharing intelligence, responding to changing technology and conditions in the wild, and balancing security measures against functionality. It is also simply not possible to reduce the attack surface to zero without sacrificing functionality necessary to operate the organization. However, with the right information and advice, organizations can respond appropriately, mitigate risks, and reduce their attack surface significantly.

About HP Security Research

Actionable security intelligence is critical to protecting organizations from the rising tide of security threats. HP Security Research (HPSR) leverages innovative research in multiple focus areas to deliver actionable security intelligence across the portfolio of HP security products, including HP ArcSight, HP Fortify, and HP TippingPoint.

Security research publications and regular threat briefings complement the intelligence delivered through HP products and provide insight into the present and future security threats facing organizations. HPSR brings together data and research to produce a detailed picture of both sides of the security coin—the state of the vulnerabilities comprising the attack surface, and, on the flip side, the ways adversaries exploit those vulnerabilities to compromise victims.

Our data

To provide a broad perspective on vulnerabilities and the nature of the attack surface, the report draws on the following sources:

- HP Zero Day Initiative (ZDI)
- HP Fortify on Demand security assessments (static and dynamic)
- HP Fortify Software Security Research
- ReversingLabs
- National Vulnerability Database (NVD)

[1] hpenterprisesecurity.com/ponemon-2013-cost-of-cyber-crime-study-reports

[2] f-secure.com/static/doc/labs_global/Research/Threat_Report_H1_2013.pdf

Key findings

Based on our analysis of the data examined, we offer these key findings:

Research gains attention, but vulnerability disclosures stabilize and decrease in severity[3]
While vulnerability research continued to gain attention, the total number of publicly disclosed vulnerabilities in 2013 was stable and the number of high-severity vulnerabilities decreased for the fourth consecutive year. The number of vulnerabilities classified as "high severity" as reported by NVD has declined since 2010. This finding seems to be at odds with the amount of interest we've seen in the area of vulnerability research. Is this a good indication of the improving awareness of security in software development or does this indicate a more nefarious trend—the increased price of vulnerabilities on the black market for APTs resulting in less public disclosures? We discuss the implications in Vulnerabilities—the state of play.

80% of applications contain vulnerabilities exposed by incorrect configuration
While we often hear about vulnerabilities that arise due to bugs in an application's code, assessments performed by HP Fortify on Demand of 2200 applications using both static and dynamic analysis found that many vulnerabilities lie outside the application's source code. Many vulnerabilities were related to server misconfiguration, improper file settings, sample content, outdated software versions, and other items related to insecure deployment. Eliminating bugs and the resultant vulnerabilities from code won't fix this—even perfectly coded software can be dangerously vulnerable when misconfigured. Don't overlook this security gap. Dedicate resources to auditing software for misconfiguration as well as for more expected forms of vulnerability. See our Software security section for the full details.

Differing definitions of "malware" make measuring mobile malware risk extremely difficult
Our examination of over 500,000 apps for the Android platform turned up some surprising results, including major discrepancies between how Google™ and different antivirus companies judge the behavior and intent of mobile apps. Limiting the number of apps available within an organization, monitoring approved apps, and thoroughly vetting EULAs are the absolute baseline for responsible defense. We look at the top threats uncovered by our analysis, discuss the discrepancies identified, and make recommendations in Mobile—vulnerabilities, exploits, and malware.

The attack surface allows for multiple avenues for compromise
As we discovered in the course of analysis on the South Korea targeted attacks, even though the malware involved was not that sophisticated, it was good enough to compromise the networks of several organizations and deliver a damaging payload that caused malicious damage and significant interruptions to normal function. Organizations should understand that there isn't a single path to take to protect vital business assets from threats. We explore this finding in South Korean case study—a glimpse into the future and lessons on the nature of targeted attacks.

[3] darkreading.com/vulnerability/lessons-learned-from-a-decade-of-vulnera/240148896

46% of mobile iOS and Android applications use encryption improperly
Improper use of encryption was one of the top client-side issues uncovered in Fortify on Demand assessments of over 180 iOS and Android applications. This finding represents a convergence of two significant issues in modern computing—the near-ubiquity of mobile technology and the increasing importance of protecting sensitive data in light of persistent attackers. As the lines are blurred between mobile technology and traditional form factors, and mobile devices are often used to manipulate confidential data for both personal and business use, encryption of targeted data is increasingly important as a key defense. See the results of this study in **Security pitfalls of hybrid mobile development platforms**.

Internet Explorer was the software most targeted by Zero Day Initiative (ZDI) researchers
Many more vulnerabilities were discovered with more than a 100% increase recorded by the ZDI over 2012 numbers. This is not a gauge of the security of Internet Explorer®, but rather, results from the market forces (both legitimate and illegitimate) that govern the price of vulnerabilities in software with massive market penetration. See **Vulnerabilities—the state of play** for more information on what determines the price of vulnerability research, the players involved, and more ZDI trends from 2013.

Sandbox bypass vulnerabilities are the #1 issue for Java
Sandbox bypass vulnerabilities caused by unsafe reflection are the most prolific issue in the Java framework and sandbox bypass due to type confusion is the most exploited. Attackers are significantly escalating their exploitation of Java by simultaneously targeting multiple CVEs and using Java more often to successfully compromise victims' computers. Organizations should seriously consider reducing their attack surface by eliminating Java from environments where it is not required. We dig into this finding and take a close look at the impact of Java vulnerabilities on the threat landscape in **Java every days**.

SCADA systems are increasingly targeted
Supervisory control and data acquisition (SCADA) systems have become increasingly tempting as a target as represented by the ZDI submissions in 2013. These control systems manage widespread or niche-based automated industrial processes such as those used for manufacturing processes, power generation, mining, water treatment, and possibly general quality control and monitoring processes, which have historically operated over separate networks and with proprietary protocols.

Further observations/ Security headlines—security by numbers

SCADA vulnerabilities continue to rise

- Manufacturing processes
- Power generation
- Mining
- Water treatment

Since Stuxnet in 2010, research in these vulnerabilities has dramatically increased.

Software security

56%	Weaknesses revealing information about application, implementation or user
31.5%	Prone to leak system information through poor error handling

Mobile apps security

52%	Security issues were a result of insecure client-side operation
48%	Insecure server-side application code or code quality issues—unstable application behavior
74%	Unnecessary permissions

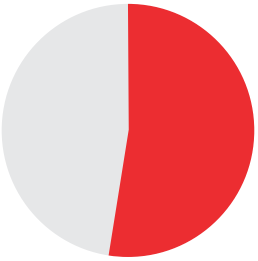
Cross-frame scripting security

33.5%	Pervasive in sites across the internet and remains significantly present within current test data

#1 Java might be big news, but Microsoft is still the no. 1 subject for vulnerability research

ZDI team 2013

Microsoft's Internet Explorer had the most vulnerabilities submitted against a single product.



Mobile—malware, exploits, and vulnerabilities

With the ubiquity of mobile technology, the increasing acceptance of personal devices in the workplace, and the tight integration of mobile technology into both business and personal functions, we wonder how much longer the identifier "mobile" as if it's something different from everything else will last. Regardless, while the label continues to hold some meaning, we're looking at mobile technology from multiple angles this year. Along with our study into the security pitfalls of hybrid mobile development platforms, and the results of this year's Mobile Pwn2Own competition,[4] we worked with ReversingLabs to investigate malware on the Android platform.

Mobile malware—or is it?

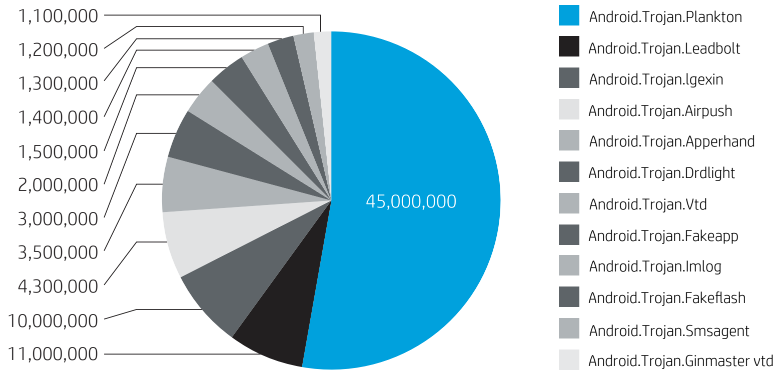
There has been significant discussion this year about the prevalence of malware targeting the Android platform, with some antivirus vendors claiming concerning levels in the wild,[5, 6] while others (including Google) have stated the opposite[7]—that the numbers were insignificant. Is Android malware a problem or isn't it?

In order to answer this question and get an idea regarding the prevalence of malware and potentially unwanted software on the platform, we took over 500,000 apps available on Google Play, and cross-referenced them against a large collection of over 2 million known Android malware and adware samples. We also noted how many times the apps had been downloaded from Google Play.

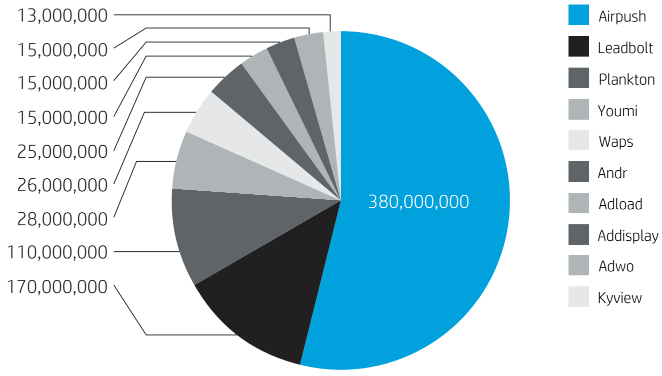
The results

The initial results appeared to confirm the opinions of some in the antivirus industry that malware and potentially unwanted software are rampant on the platform. We found adware that had been downloaded hundreds of millions of times and malware that had been downloaded tens of millions of times from Google Play by Android users. A breakdown of the top 10 malware and adware in the charts below:

Android malware—downloads



Android adware—downloads



The takeaway

Although our research concludes that malware does not pose a huge threat to mobile platforms (at least relative to the impact on desktop environments), the antimalware industry needs to work in tandem with platform providers on meaningful and consistent definitions of what constitutes malicious or deceptive behavior. The differences between mobile and desktop operating systems interrupt some threats from monetizing with more traditional payloads, but there's no room for complacency. If there's money to be made by taking advantage of people, processes, or technology then the threats will undoubtedly appear.

[4] pwn2own.com/

[5] mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2013.pdf

[6] sophos.com/en-us/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf

[7] http://h30499.www3.hp.com/t5/HP-Security-Research-Blog/Trick-or-treat-Who-s-afraid-of-mobile-malware/ba-p/6255831

These numbers looked excessive to us. While Google doesn't appear to like to use the term "malware,"[8] according to Google Play's app developer content policy this shouldn't be occurring:

> "We don't allow content that harms, interferes with the operation of, or accesses in an unauthorized manner, networks, servers, or other infrastructure.
>
> - *Don't transmit viruses, worms, defects, Trojan horses, malware, or any other items that may introduce security vulnerabilities to or harm user devices, apps, or personal data.*
> - *Apps that collect information (such as the user's location or behavior) without the user's knowledge (spyware) are prohibited.*
> - *Malicious scripts and password phishing scams are also prohibited on Google Play, as are apps that cause users to unknowingly download or install apps from sources outside of Google Play.*
> - *An app downloaded from Google Play may not modify, replace or update its own APK binary code using any method other than Google Play's update mechanism.[9]"*

Discussion

The number of detections for our sample of Android apps was extremely large. However, those that appeared the most prevalent were detected as adware—a type of potentially unwanted software (PUS). These apps are still available from Google Play, and have been downloaded and installed by millions of users deliberately with their consent. When looking at both combined, the PUS numbers dwarf the malware numbers for downloads.

After a little more digging, it looks like most of these are detected for the following reasons:

- Pop-up notifications. Annoying and potentially unwanted for some, but others might appreciate this functionality. Unlike PC versions of adware (which can be notoriously difficult to remove), Android apps can be removed more easily due to the restrictions of the sandbox.
- Personal information capture. The permissions for collecting information are shown during the app's install—although it might be the case that most users accept them without reading them.

Based on our analysis, it appears that most of the apps that are detected as malware/adware are not primarily malicious. Their detection seems to be associated with the inclusion of an ad-serving library. As the app economy has shifted from paid apps to free apps, it has become normal to include ads in free versions of apps. The developers of these apps could choose any ad provider, and some of the highest-priced ad providers could supply library code that targets user info aggressively.

However, several adware libraries, such as Plankton, have been reported to contain backdoor functionality that has led some in the AV industry to detect these programs as Trojans rather than adware.[10] Even though false detections (or false positives in this case) are possible, data provided by ReversingLabs lists many software applications available on Google Play that are detected as malicious. They together amount to over 80M application installs. One would expect that while these applications are still in Google Play, it would be the responsibility of Google to determine which of these applications are truly harmful to users and if determined to be malicious, promptly removed from Google Play.

However, as is often the case when we start discussing less than straightforward malicious intent, it's not just Google that might be at odds here with what it defines as malicious and there is great variability between AV companies and their determinations of Android malware/adware. We recorded scan results for over 7000 of the most downloaded adware. The table below lists our results and shows that there is massive variability between the determinations made by different AV companies. It seems one person's adware might be another's benign app.

[8] http://developer.android.com/guide/faq/security.html#malware

[9] http://play.google.com/about/developer-content-policy.html

[10] http://nakedsecurity.sophos.com/2011/06/14/plankton-malware-drifts-into-android-market/

Antivirus scanner	Number of detections	Antivirus scanner	Number of detections	Antivirus scanner	Number of detections
ESET-NOD32	5121	McAfee	1698	Kaspersky	164
Fortinet	4799	McAfeeGW	1588	Rising	103
DrWeb	4689	Gdata	1336	Kingsoft	51
Sophos	4507	Emsisoft	1314	Microsoft	39
AntiVir	3666	BitDefender	1235	Avast	37
F-Prot	3661	Microworld	174	ClamAV	18
TrendMicro-HouseCall	3574	Bkav	1120	Antiy-AVL	10
Comodo	3253	Commtouch	904	QuickHeal	7
Ikarus	3210	NANO	830	Symantec	6
AVG	3052	Baidu	655	nProtect	1
Fsecure	2795	TrendMicro	168	Norman	1
VIPRE	2538	Jiangmin	166	VBA32	1

The industry has not yet come to consensus. ESET, Sophos, and Fprot are on the high side (>50%), whereas Symantec, Microsoft, Kaspersky are on the low side (<5%). These numbers are probably also affected by the fact that some vendors have an AV solution for the Android platform while others do not. It is also possible that the Windows®-based command line scanners used for this test do not contain the full set of Android signatures (as would be available in a scanner specifically for Android devices).

Android vs iOS

Compared to the high detection numbers for Android apps reported by particular AV companies, things look very different for iOS, with very few reports of malware for this platform. A major difference between the Android and iOS app platforms is the screening process of the app store. The Apple iOS store performs a detailed screening process that can take weeks, and will reject apps for a number of non-technical reasons, including:

- 2.9 Apps that are “beta,” “demo,” “trial,” or “test” versions
- 2.11 Apps that duplicate apps already in the App Store may be rejected, particularly if there are many of them
- 2.12 Apps that are not very useful or do not provide any lasting entertainment value may be rejected
- 2.13 Apps that are primarily marketing materials or advertisements will be rejected[11]

Google Play store publishing happens within hours, and allows apps to be published without much restriction. One way of looking at it could be that Google is friendlier to developers, and that it does not reject apps for aesthetic reasons. Another way of looking at it could be that Google’s revenue is more ad-related than Apple’s hardware-centric revenue, and therefore Apple can enforce a more consumer-friendly app store policy.

[11] https://developer.apple.com/appstore/guidelines.html

Conclusions

Modern operating systems have security baked in and mobile platforms are doing better than their desktop counterparts. A centralized app store is a big win to filter out malware, and compartmentalized permission requests are also a big win for security, if implemented correctly. App makers should be following the security best practice of least privilege.

The common-sense line for the mobile apps is yet to be decided in the Android market, as shown by different stances on mobile malware between Google and antivirus vendors. The industry needs to work together to come up with consistent definitions of what constitutes malicious or unwanted behaviors and a sensible app store policy and guidelines, accommodating to app users, app makers, and third-party ad providers while preventing abuses.

For what we might refer to as "real" (or, more objectively, malicious) malware incidents, the reported numbers for mobile malware still appear to be low relative to the number of adware detected. However, absence of evidence is not an evidence of absence. The security industry should remain vigilant for the new attacks that are sure to come.

Recommendations

With the types of behaviors observed in the malware and adware samples examined in this study, we offer the following, practical advice for limiting exposure to malicious or potentially unwanted apps on mobile platforms:

1. Only download apps from reputable sources.
2. Pay close attention to permissions when installing apps—read the EULA. If the permissions seem excessive, out of context for the app, or unjustifiable—don't install the app. (A location-sharing flashlight app that recently made headlines[12] is a good example of an app deceptively and unnecessarily gathering location and device information from users of the app.)
3. Consider speaking to your service provider to block premium-rate services (sending messages to premium rate services is a popular payload for some mobile malware—block these services and remove the risk).
4. Use a reputable antivirus scanner appropriate to the device's platform. Much modern malware is complex, cloud-based, and multi-component—and the same holds true on mobile platforms. In many cases determining the actual behavior and intent of a mobile application is a difficult job for even experienced researchers (the difficult nature of this research may even contribute to the lack of consistent determinations across the industry as noted in our report).

[12] techrepublic.com/blog/it-security/why-does-an-android-flashlight-app-need-gps-permission/

Mobile vulnerabilities and exploits

The takeaway

Our experience at this year's Mobile Pwn2Own competition placed the perceived security of mobile devices right in the spotlight and raised some interesting questions regarding users' perception of mobile security.

The competition showed clearly that vulnerabilities do exist and can be exploited to see the same kinds of resulting compromises and payloads as we see on more traditional platforms. Same methods of attack, same avenues of compromise, same targeted information and resources, same payloads. The real difference is users don't expect these attacks on mobile platforms and hence aren't modifying their behavior accordingly considering the level of risk.

There is an implicit level of trust that users bring to their use of mobile devices that may be somewhat misplaced. While the exploit of mobile devices isn't exactly child's play (or even remotely close) our observations indicate an endgame where the personal, sensitive, confidential, valuable data that is stored on a mobile is just as vulnerable to compromise directly by attackers, or indirectly by malware, as the data stored anywhere else.

Mobile Pwn2Own

Nothing beats a good proof of concept to turn ideas about the security of particular platforms on their head—and Mobile Pwn2Own provided some impressive examples. Mobile Pwn2Own is an annual contest that rewards security researchers for highlighting security concerns on mobile platforms. The contest focuses on hardening the mobile attack surface through cutting-edge research and responsible disclosure. For more information on the HPSR Pwn2Own competitions, see pwn2own.com/.

We had three entrants bring their exploits to the arena to share their research and claim their prize (in total US$117,500).

- Our first-ever winning Chinese team presented two different exploits against Safari. The first exploit demonstrated by the Keen Team resulted in the compromise and capture of Facebook credentials on iOS 7.0.3 while the second exploit against iOS 6.1.4 resulted in the theft of photos from the affected device.
- The MBSD team's exploit of multiple apps on the Samsung Galaxy S4 was described as "elegant" by the researchers observing in the room. They were greeted by a surprised and respectful round of applause as malware was silently installed on the device and the data exfiltration payload was executed.
- On day two, within minutes of the attempt, we witnessed a successful exploit on two different devices and paid US$50,000 for the privilege. Pinkie Pie compromised Chrome on both a Nexus 4 and a Samsung Galaxy S4 just for good measure.

Security pitfalls of hybrid mobile development platforms

The takeaway

This study sheds light on some of the areas to consider when selecting hybrid development platforms and the associated security risks faced by mobile applications developed using this approach. Missing or weak encryption was identified as the top issue in native mobile applications and based on the API analysis, which has a potential to be equally damaging to the hybrid mobile applications. In other cases, certain features of the hybrid platform such as the whitelist-based cross-domain access restrictions, could prove beneficial for developers looking to rewrap their existing web applications, which have most of the sensitive work and storage done on server side. Hybrid development technologies are evolving rapidly and developers planning to use them must carefully review their security requirements against the support available within the framework being considered.

With more of today's enterprises looking for a cost-effective means to enter the mobile application market, the adoption of hybrid development strategies for mobile platforms is growing. Leveraging existing software development skills such as HTML, CSS, and JavaScript and applying that knowledge to the mobile application paradigm offers optimal flexibility in budget and time to market. However, what price does an enterprise have to pay with regard to security?

Approach

In this study, HP Security Research set out to conduct a comprehensive evaluation of how well the built-in feature sets of various hybrid platforms encourage and ease the secure development of cross-platform mobile applications. Essential to the usefulness and completeness of the study was the proper selection of the hybrid development platforms and the categories of security vulnerabilities to evaluate the chosen platforms against. Selection of the hybrid development platforms was done based on the popularity of the platforms amongst developers as well as the coverage of the architectural principles employed by the platforms.

The vulnerability categories used in the evaluation were chosen based on real-world data collected by HP Fortify on Demand. Analysis of the issues reported by these data highlighted the criticality of four major components: encryption, storage, permissions, and the WebView to the security of a mobile application. In addition to these four categories, our research also identified extensibility architecture as a prominent contributor to security health, especially, of a hybrid mobile application given its importance to closing the gap between the native functionality support and the features exposed by the hybrid platform.

Overview of hybrid platforms

The main goal of nearly all hybrid mobile technologies is to support cross-platform development. The similarities, however, end there. Beyond this common purpose, the popular hybrid development platforms that are being increasingly adopted today utilize fairly distinct approaches to expose the native device functionality to applications using popular web technologies or programming languages. In this study, we focused on three hybrid development technologies, namely, PhoneGap 2.9.0, Xamarin Studio 4.0.13, and Appcelerator Titanium 2.x. While all of the three platforms aspire to provide the benefit of cross-platform development, they differ noticeably in four key areas.

Programming languages

In addition to supporting cross-platform development, hybrid technologies also offer developers the convenience of using widely adopted web technologies or programming languages and minimize the need for mastering device-specific technologies. PhoneGap, for instance, supports popular web development technologies like HTML, Cascading Style Sheets (CSS), and JavaScript.[13] The Xamarin platform, on the other hand, generates native code from C# code thus allowing developers to use the power of the .NET framework for cross-platform mobile application development.[14] Titanium, similar to PhoneGap, allows developers to use JavaScript code to create cross-platform mobile applications and uses V8 or Rhino on Android and JavascriptCore on iOS for JavaScript execution.[15]

User interface (UI) rendering

PhoneGap relies on an embedded WebView container for rendering UI and executing JavaScript. Since Xamarin compiles the C# code into native binaries, the rendering is performed by the corresponding device OS. It forgoes the need for an intermediate rendering layer like the WebView container in PhoneGap. Titanium also generates native application binaries from the JavaScript code and also relies on the native OS for UI rendering instead of the WebView.

Access to native functionality

PhoneGap makes use of the WebView container as a bridge that allows the exchange of messages between the JavaScript code and the native code thus providing access to native device features such as camera and accelerometer. Xamarin builds against the .NET Base Class Libraries with specially crafted Xamarin Mobile Profile packaged in MonoTouch and Mono.Android dll to provide a one-to-one mapping with the device feature APIs.[16] Titanium's API library consists of modules, which when referenced create the JavaScript to native API bindings.[17]

Extensibility

PhoneGap has developed a plugin API to enable developers to create their own extensions or use third-party plugins for exposing device features missing from official PhoneGap standard API.[18] Xamarin supports C# bindings to third-party Objective-C libraries[19] for iOS applications and also allows developers to bundle Android native libraries with their C# projects.[20] In Titanium extensibility is supported in the form of a module API that allows the creation and use of third-party modules for extending the core platform functionality.[16]

Top security issues plaguing mobile applications

In an attempt to focus the hybrid platforms evaluation on relevant and significant security issues plaguing the mobile applications, we evaluated security findings from assessments performed by HP Fortify on Demand. The data set included findings from audits of over 180 native mobile applications across both the iOS and Android platforms. The 216 unique vulnerability categories detected during the audits were distributed almost evenly between two major buckets. Nearly 52% of the issues were a result of insecure client-side operation whereas about 48% were related to either insecure server-side application code or code quality issues that could result in unstable application behavior. The server-side vulnerabilities and code quality issues were excluded from this study because they are fairly agnostic to the application development approach and must be fixed regardless of the use of native or the hybrid approach.

Deeper analysis of the client-side issues helped us identify four major categories of security vulnerabilities that commonly put the integrity of the mobile applications and the user's data at risk. Figure 1 shows the distribution of these categories across the entire data set. A breakdown of each category shown in Figure 2 offers further insight into the security mistakes that developers commonly fall victim to when developing mobile applications.

[13] http://docs.phonegap.com/en/edge/guide_overview_index.md.html#Overview

[14] http://docs.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/

[15] appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/

[16] http://docs.appcelerator.com/titanium/2.0/#!/guide/Creating_a_New_Titanium_Module

[17] http://docs.phonegap.com/en/2.9.0/

[18] http://docs.phonegap.com/en/edge/guide_hybrid_plugins_index.md.html#Plugin%20Development%20Guide

[19] http://docs.xamarin.com/guides/ios/advanced_topics/binding_objective-c/

[20] http://docs.xamarin.com/guides/android/advanced_topics/java_integration_overview/

Figure 1
Distribution of native mobile application vulnerabilities across the top four categories

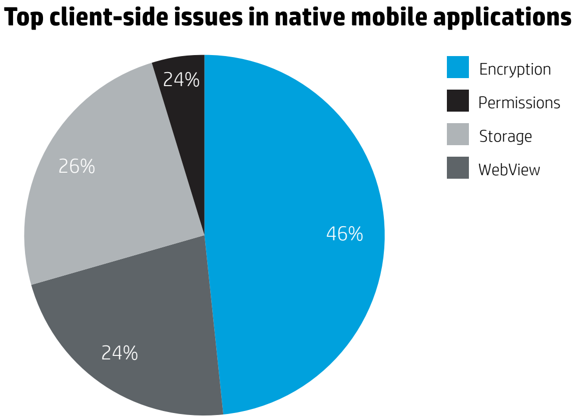
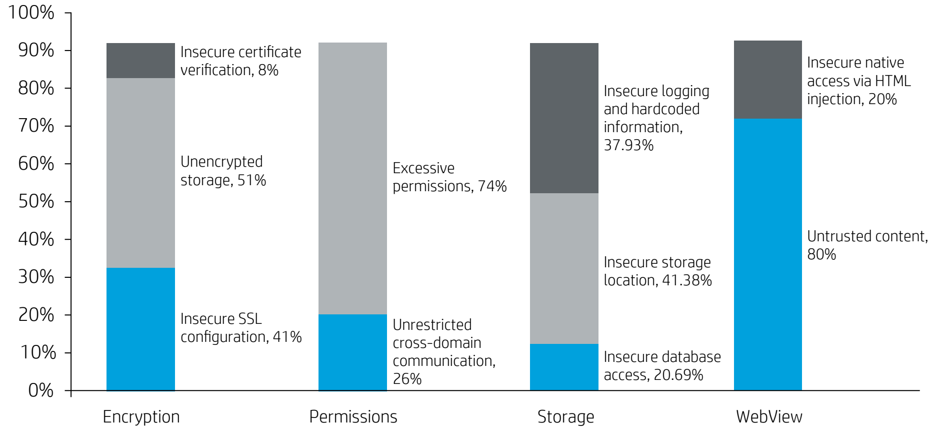


Figure 2
Further breakdown of the security issues constituting the top four categories



Encryption

Missing or misused cryptographic APIs made for a common occurrence in our analysis of encryption-related vulnerabilities. The statistics indicate that the developers either completely miss encryption before storing sensitive information on device or often rely on weak algorithms. Also 41% of the encryption-related issues resulted from unencrypted transfer of sensitive information. Although a small fraction, it was interesting to see that developers often released applications with SSL certification validation disabled.

The statistics indicate that the developers either completely miss encryption before storing sensitive information on device or often rely on weak algorithms.

Storage

Our analysis of native vulnerabilities found that the insecure use of storage APIs is a prominent root cause of security issues. Unsafe storage of information on publicly accessible external SD cards is seen to be a common practice amongst mobile application developers and was found to be responsible for nearly 42% of all storage-related issues. SQL injection vulnerabilities, which could similarly expose contents of the device database constituted approximately 21% of the issues. Additionally, almost 38% of the issues were related to insecure logging practices as well as hardcoding of sensitive information, which runs counter to age-old security best practices.

Permissions

Modern mobile phones offer powerful device features natively and also allow applications to expose custom features for reusability. The permissions model is essential for preventing the misuse of these features, examples of which include permission to use the camera, external storage, Internet, and others as well as permissions to share the custom components between apps. Unfortunately, 74% of the issues were caused by Android applications requesting more permissions than were necessary for their operation thus putting the user's data at risk in case of a compromise.

WebView
Lack of proper input validation was found to be a prevalent problem as well, with 80% of the issues resulting in cross-site scripting vulnerabilities and 20% of the issues allowing unsafe access to the native device APIs.

Top concerns of hybrid development

The preceding analysis of the native mobile application vulnerabilities formed the basis for selection of the security requirements used during the evaluation of the built-in security support offered by the three hybrid development platforms. In addition to the encryption, permissions, storage, and WebView dimensions chosen based on the analytical data referenced above, we also included extensibility as the fifth and final dimension. As seen in the overview of the hybrid platforms, extensibility is a critical component of any hybrid development environment.

These five dimensions were further broken down into sub-categories for deeper analysis. During the evaluation, we reviewed each framework to determine if:

- They make available APIs that would encourage developers to implement sensitive features in an insecure fashion
- They are missing critical APIs thus forcing the developers into implementing features insecurely

Below, we describe the key findings from the evaluation of the three frameworks against the five category selections:

Cryptography / Secure Sockets Layer (SSL)
Neither PhoneGap[20] nor Titanium[21] provide a built-in encryption API, which could result in the proliferation of information disclosure issues arising from unencrypted storage and the need to rely on native plugins for their encryption needs. They also lack the API support for applications to selectively enable SSL ciphers, which can result in insecure SSL configuration and the reliance on weak ciphers for protection of information in transit. Furthermore, they allow insecure certificate verification practices by offering a trustAllHosts setting in the FileTransfer[22] API thus allowing developers to completely disable SSL hostname verification.[23] While hostname verification can be disabled in Xamarin as well,[24] it does not support a convenient API or a setting like the other two frameworks that will let developers do so easily.

Storage
Failure to strictly prohibit non-parameterized queries in the SQLite APIs of both Titanium[25] and Xamarin libraries[26] can lead to insecure development practices and expose the hybrid applications to insecure database access or SQL injection issues. In PhoneGap, on the other hand, the API lacks the option to allow developers control over whether a file is written to the external or internal storage as well as whether it is visible to world for read/write or none, thus potentially exposing PhoneGap applications to unintentional use of insecure storage location.[27]

Permissions
Both the Titanium and PhoneGap frameworks automatically generate the native configuration file (AndroidManifest) with insecure defaults[28, 29] increasing the potential for excessive permissions to be granted to an application on the Android platform without the developer's knowledge. Because any application with an INTERNET permission is capable of reading sensitive information and transmitting it to any domain on the Internet, restricting an application's domain access is essential. Our study, however, shows that PhoneGap offers a provision to restrict such access through the Access Origin setting[30] whereas Titanium follows the best practice by enforcing the same origin restriction by default.[31] Xamarin, however, lacks support for domain whitelisting, which could potentially cause the applications to enable **unrestricted cross-domain communication**.

[20] http://docs.phonegap.com/en/2.9.0/

[21] http://docs.appcelerator.com/titanium/2.1/#!/api

[22] http://docs.phonegap.com/en/2.9.0/cordova_file_file.md.html#FileTransfer

[23] http://docs.appcelerator.com/titanium/2.0/#!/api/Titanium.Network.HTTPClient

[24] http://social.msdn.microsoft.com/Forums/vstudio/en-US/acb2fde0-32a5-403c-805d-43bdc8e37c7f/how-to-disable-certificate-hostname-validation-between-net-c-client-and-secure-web-service

[25] http://docs.appcelerator.com/titanium/3.0/#!/api/Titanium.Database.DB-method-execute

[26] http://msdn.microsoft.com/en-us/library/sebfsz50(v=vs.110).aspx

[27] http://docs.phonegap.com/en/2.9.0/cordova_file_file.md.html#requestFileSystem

[28] http://developer.appcelerator.com/question/118483/android-uses-permissions

[29] http://stackoverflow.com/questions/16787934/do-i-need-an-androidmanifest-xml-when-working-with-phonegap

[30] http://docs.phonegap.com/en/2.9.0/guide_whitelist_index.md.html#Domain%20Whitelist%20Guide

[31] http://docs.appcelerator.com/titanium/3.0/#!/guide/Mobile_Web_Debugging_and_Testing_Tools

WebView

The PhoneGap platform solely relies on the WebView container for rendering and executing applications and is necessary to expose the native device features to the HTML and JavaScript code.[32] The ability for a PhoneGap-based hybrid application to access native device features from HTML/JavaScript code using the native WebView or UIWebView can lead to insecure native access via HTML injection.

Extensibility

A hybrid application could suffer a compromise from insecure third-party components because most hybrid platforms don't support all the native functionality and require developers to rely on plugins by providing the custom plugin interface. However, in the absence of a vetting process, there is a high propensity for the plugins to contain serious vulnerabilities or have a malicious intent.

The table below depicts the security concerns associated with individual hybrid development platforms based on the API characteristics described above.

Categories		**Platforms**					
		PhoneGap		**Titanium**		**Xamarin**	
		Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Crypto/SSL	Unencrypted storage	X	X	X	X		
	Insecure SSL configuration	X	X	X	X		
	Insecure certificate verification	X	X	X	X	X	X
Storage	Insecure database access			X	X	X	X
	Insecure storage location	X					
Permissions	Excessive permissions	X		X			
	Unrestricted cross-domain communication					X	X
WebView	Insecure native access via HTML injection	X	X				
Extensibility	Insecure third-party components	X	X	X	X	X	X

Secure development of hybrid mobile applications

The results from the analysis of the APIs currently supported by the chosen hybrid platforms pinpoint some of the hybrid development features, which if hardened with basic security enhancements, could help developers avoid the same pitfalls observed in insecurely built native applications.

One such example is the FileStorage API in PhoneGap. Enhancing this API to let developers configure the destination storage location (application sandbox vs. external SD card) and the visibility of the generated files (public vs. private) will help reduce the risk of information theft in PhoneGap applications. Similarly, implementing the SQL query execution API in Xamarin and Titanium in a manner that encourages developers to use parameterized queries by default can curtail SQL injection vulnerabilities.

On the encryption front, the ability to configure the desired SSL ciphers could enable developers to work around the device's default cipher list. Increasing the reliance on the native data protection API for data storage whenever possible and built-in support for strong encryption algorithms would ensure the reliable protection of sensitive information handled by hybrid mobile applications.

Additionally, a careful review of the default permissions granted in the configuration artifacts generated automatically by the frameworks is paramount and will help mitigate issues resulting from excessive permissions. Prevention of insecure cross-domain data exchange via malicious input injection can similarly be facilitated in the Xamarin framework by offering domain whitelisting capabilities as done by PhoneGap and Titanium. The hybrid platforms could also take steps to mitigate the risks introduced by the inclusion of third-party components. Enhancing the built-in support for native features and offering a central repository backed by a robust verification process for crowd sourced components could help reduce the risks from malicious plugins.

Recommendations

Developers must stay cognizant of such security pitfalls in the hybrid frameworks and must also bear some of the burden by ensuring that they follow the security guidelines to protect the integrity of their applications and the privacy of their users. The following best practices have been formulated based on the findings from this study:

- Avoid disabling host name validation from production applications to ensure proper verification of all the services involved in any interaction.
- Review all the configuration artifacts and the permissions requested by the application.
- Properly validate all external content and input consumed by the application when executing in the WebView container.
- Carefully configure the domain whitelists to minimize impact from input injection attacks.

Vulnerabilities—the state of play

Vulnerabilities, proofs of concept, and exploits

This section of the report focuses specifically on vulnerability research, the factors that drive its direction, and the tangible outcomes that arise and impact everyone as a result of that research. However, before we dig into the numbers and the details, we need to talk a little bit about the greater context for vulnerability research, and how it fits into and shapes the greater threat landscape. These considerations are key to understanding the relevance of the vulnerability statistics to follow.

In this report, we discuss **vulnerabilities**, **proofs of concept**, **exploits**, and **the vulnerability market**. For the uninitiated:

- **Vulnerabilities** are defects or bugs that allow for external influence on the availability, reliability, confidentiality, or integrity of software or hardware. Vulnerabilities can be exploited to subvert the original function of the targeted technology.
- **Proofs of concept (POC)** are provided by vulnerability researchers when they report the discovery of a vulnerability. These POCs demonstrate the vulnerability to the affected vendor or buyer, (but generally exclude a hostile payload delivery).
- **Exploits** are code written expressly to take advantage of the security gap created by a particular vulnerability in order to deliver a malicious payload. They may be targeted at specific organizations or used en masse in order to compromise as many hosts as possible. Delivery mechanisms utilize many different technologies and vehicles and often contain a social engineering element (effectively an exploit against vulnerabilities in human nature in order to make the victim take a particular action of the attacker's choosing).
- **The vulnerability market** is the informal, unregulated marketplace for the buying and selling of vulnerability research. The legitimacy of the actors in the marketplace is highly variable and ranges from white hat bug bounty programs (such as the ZDI), to ethical researchers motivated by improving security for everyone, to mercenary vulnerability researchers who sell their discoveries and creations to the highest bidder, to organized criminals or even nation-states.

The takeaway

Regardless of the influences of both the market and the nature of our measurements of vulnerabilities, it is important to prioritize your patch management efforts based on what is actively being targeted by attackers today. Introduce risk calculations into metrics surrounding vulnerability exposure for a specific business, or industry (e.g. the number of unpatched machines running Java vs. Notepad). If targeted technology exists in your environment and it is not critical for your business then remove it. Reduce your exposure; reduce the attack surface.

A marketplace of ideas

The vulnerability market is an economic sub-ecosystem within the security industry. Operating as a worldwide marketplace with buyers, sellers, supply and demand, its complexity has grown over the past decade. In the earliest days, the market was a place for hackers to trade and sell exploits amongst themselves for eminence, disruption of traditional IT and software development pipelines, and sometimes for ill-gotten profit. Over the last three decades, the market for vulnerabilities and coveted exploits has changed into a more legitimate commercial space rather than strictly being a black-market.

HP's Zero Day Initiative (ZDI) pioneered the vulnerability "white market" and its mission is to disrupt the operation of the black market section of the marketplace by legitimately purchasing vulnerability research that can then be disclosed to affected vendors. In this way, these vulnerabilities are effectively taken off the market for possible abusers and the affected vendors are able to create patches to address these holes before the information is made public. Or put another way, the ZDI team's focus is on securing the ecosystem using incentivized coordinated disclosure to affected vendors to prevent blind-sided attacks on corporate environments.

The price that is determined by the operation of the market for the discovery of particular types of vulnerabilities is a major determiner of the types of vulnerability research that is carried out. The higher the price, the more interest there will be in particular types of technology, OS, or application. This is an important distinction—just because there are a lot of vulnerabilities reported for a particular technology or application, it doesn't necessarily mean that that technology is inherently unsafe (or less safe than other, similar technology). All software and hardware is prone to vulnerabilities. Those that are discovered are often discovered because there is greater motivation to research that technology due to the higher perceived rewards for the researchers. (The interest in Microsoft Internet Explorer and the correspondingly high number of vulnerabilities discovered is a good example of this effect.)

There are a number of different factors that drive the price of a particular vulnerability, but it is generally determined by a combination of the following characteristics:

Exclusivity	Time sensitivity	Valuation	Reputation	Vulnerability scope
• Restricting purchase to a single buyer demands a higher price • Multiple buyers reduces vulnerability or exploit value	• Value diminishes proportionally to the number of people aware of a vulnerability • A race takes place against vendors, other researchers, and the market components (white, grey, and black)	• Vulnerability rates are considered intellectual property and not publicly available • Many difficult-to-measure factors affect final value	• Researchers develop their own finesse and management of their brand equity (can include contest wins), resulting in higher payouts	• Vulnerabilities in widely used applications or operating systems are more sought after resulting in higher valuation, particularly if a vulnerability applies across multiple/current versions

Over the past year, the debate on pricing has morphed into a debate on regulating the marketplace. Advocacy arguments publicly occur on both sides: Regulation would bring transparency to a secretive market, versus regulation impinging on researchers' rights, which needlessly restricts a free market and creates unnecessary additional risk to corporate endeavors such as IT operations or software development.

A final word on collecting accurate vulnerability data
Vulnerability information is available from a number of different outlets, including public repository programs such as NVD or OSVDB, private bug bounties such as HP's ZDI, private security consultants, vendor bug bounty programs, and the underground black market.

While the public repositories provide a glance into the vulnerability landscape—*it is limited to reporting those that are publicly disclosed or directly submitted to the organization*. This leaves a silo-driven gap for any one organization's ability to speak to the security and vulnerability landscape as a whole.

The very nature of software and its propensity toward vulnerabilities suggests strongly that there are zero days being exploited in the wild that we do not yet know about. So, we know what we know, we don't know what we don't know, but we also know that there is some information that we don't know—yet. And you need to make decisions and take appropriate action understanding these limitations.

The numbers

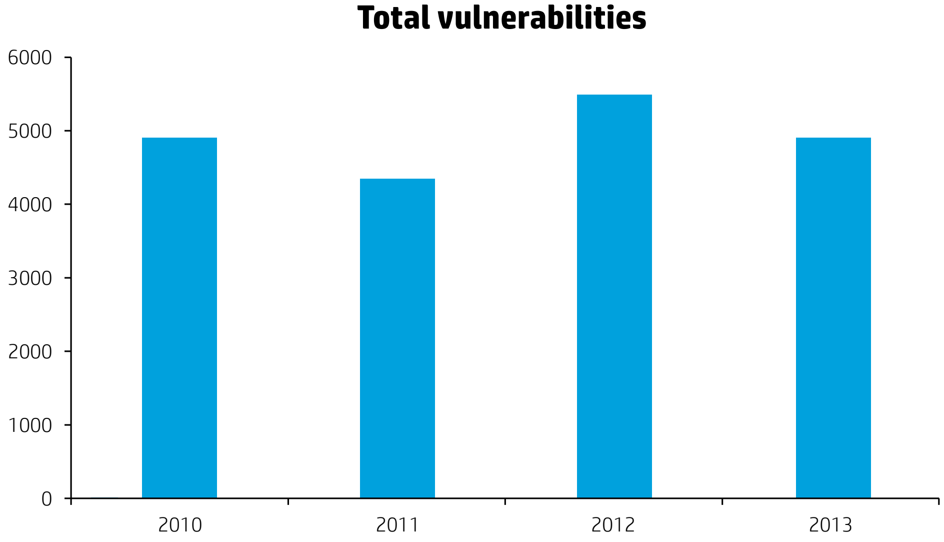
This section of the report uses data from the National Vulnerability Database (NVD) and the HP Zero Day Initiative (ZDI). Examining this data has led us to make the following observations:

- Finding 1: Total number of software vulnerabilities reported holds steady over time.[33]
- Finding 2: The number of critical severity vulnerabilities is declining.[34]
- Finding 3: SCADA systems are being increasingly targeted by vulnerability researchers and that interest is having an impact. The ZDI received more than double the number of SCADA vulnerability submissions in 2013 than it did in 2012.

Finding 1: Total number of software vulnerabilities reported holds steady

The total number of new vulnerabilities reported through November 2013 (4704) decreased by roughly 6% from those disclosed in the same period for 2012 (5012). However, total numbers reported increased over 2010 and 2011 statistics.

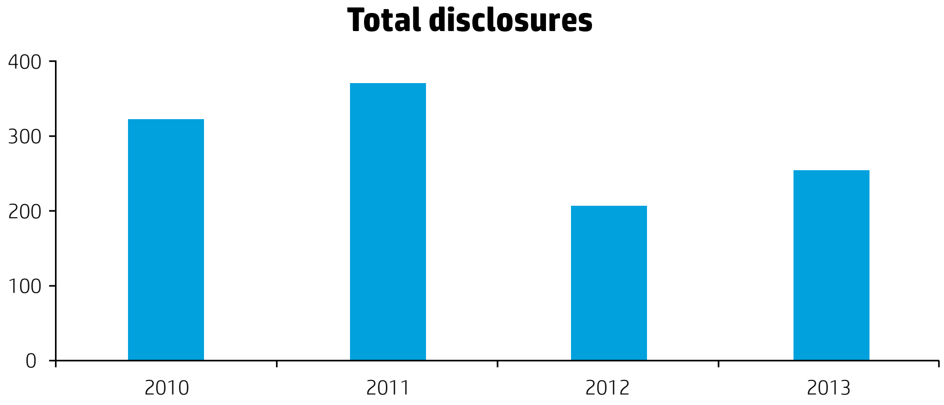
Figure 1
Disclosed vulnerabilities measured by NVD, 2010–2013[35]



The total number of disclosed vulnerabilities as reported by HP's ZDI demonstrates a fairly consistent trend although for program reasons, 2012's statistics are significantly reduced and do not reflect occurrences in the broader landscape.

[33] http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml
[34] http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml
[35] http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml

Figure 2
Vulnerabilities disclosed by HP's Zero Day Initiative, 2010–2013



The lack of a significant decrease in the number of reported vulnerabilities demonstrates the continued struggle to secure the ecosystem. Vulnerabilities still occur, largely due to software developers continuing to make assumptions regarding the environment in which their application will run and that frequently do not hold during the execution of the program. Researchers continue to discover them, and vendors must continue to patch.

What this does not speak to is the volume of vulnerabilities that are not publicly disclosed when found—those being delivered to the black market for private and/or nefarious consumption.

Finding 2: High-severity vulnerabilities are decreasing

Vulnerability severity is based on the CVSS scoring system. This system is designed to provide an open and standardized method for rating IT vulnerabilities.[36]

The number of vulnerabilities classified as "high severity" as reported by NVD has slowly declined since 2010.[37] Vendors have begun incorporating better security technologies—such as sandboxes—into their software, which makes it more difficult to obtain system access. To do so would require chaining multiple vulnerabilities together.

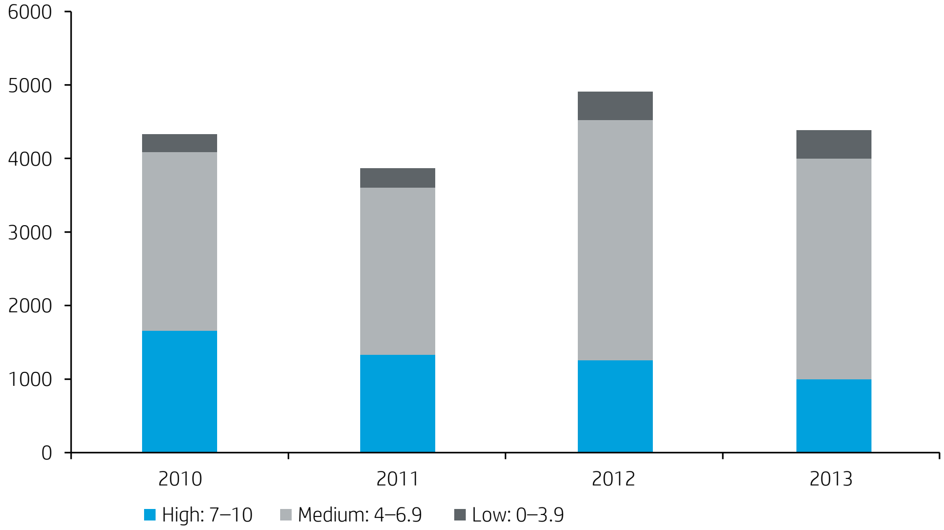
The score is calculated against six base metrics:[38]

- Access vector
- Access complexity
- Authentication
- Confidentiality impact
- Integrity impact
- Availability impact

From this, the vulnerability is assigned a numeric score on a scale of 0 to 10. Not all vulnerabilities have equal impact. Those vulnerabilities of the highest severity are scored in the range of 7 to 10; medium at 4 to 6.9; and the lowest severity are 0 to 3.9. A CVSS base score greater than 7.5 would indicate a severe compromise of the operating system of the host target where the product executes.

[36] first.org/cvss
[37] http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml
[38] hfirst.org/cvss/faq

Figure 3
Disclosed vulnerabilities by severity measured by NVD, 2010–2013[39]



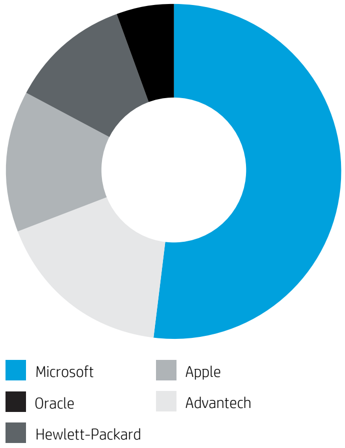
Zero Day Initiative 2013

Due to the ZDI's position as one of the premier vulnerability acquisition programs, the team's researchers have analyzed some of the most interesting vulnerabilities to have occurred over the past year. The daily traffic gives the team a unique insight into what attracts the interest of external researchers.

In 2013 the ZDI saw a number of new external researchers join the white hat market and as a result, there was a shift in vendors and products reported against. However, the shift from 2012 to 2013 was only significant enough to change the order of the top four vendors (Microsoft, Oracle, Apple, and Hewlett-Packard) and includes the addition of one new vendor (Advantech).

When simultaneously conducting research as well as performing as a participant in the vulnerability white market, the goal is to earn a sustainable living, supplement one's income, and/or gain brand equity. To do so successfully, the researcher often targets vulnerabilities in the most widely used applications or operating systems by major vendors with the largest user mass, referred to as "surface area." That is not to imply in any way that these vulnerabilities are easy discoveries. If such vulnerabilities are found, they are more likely to result in payment by the worldwide top four vendors in the chart above.

Figure 4
Top five vendors by submission to the ZDI in 2013



Finding 3: SCADA systems are increasingly targeted by vulnerability researchers and that interest is having an impact on the number of vulnerabilities identified.

Another extremely tempting target—supervisory control and data acquisition (SCADA) systems—is represented by the inclusion of Advantech in the top five vendors for ZDI submissions in 2013. These control systems manage widespread or niche-based automated industrial processes such as those used for manufacturing processes, power generation, mining, water treatment, and possibly general quality control and monitoring processes, which have historically operated over separate networks and with proprietary protocols.

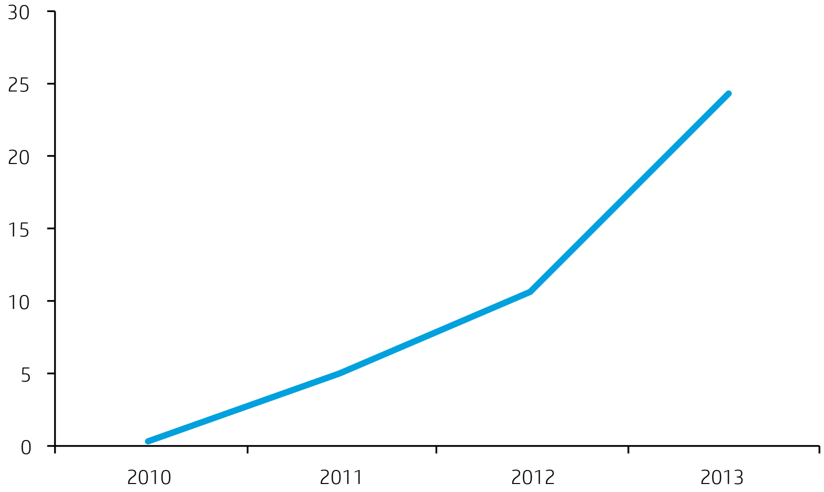
As documented in last year's report,[40] migration has begun to fold these systems into standard networks, and in some cases even via the Internet to simplify asset management, billing, and operations. As these systems continue to migrate away from their separate isolated networks, certain security problems that were once masked by a restricted surface area for attack have begun to emerge. SCADA systems first gained attention after the Stuxnet worm was discovered to have infiltrated an Iranian uranium enrichment plant in 2010 specifically targeting equipment manufactured by one company, and illustrates why ZDI's external researchers are actively interested in finding, and disclosing these vulnerabilities.

[39] http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml

[40] hpenterprisesecurity.com/collateral/whitepaper/HP2012CyberRiskReport_0213.pdf

It has been observed from the SCADA submissions into the ZDI program that most of the SCADA software applications are lacking security functionality, thus researchers find numerous vulnerabilities. Most of these vulnerabilities are of high severity and are not complicated to exploit—such as stack overflows.

Figure 5
SCADA submissions to the ZDI 2010–2013



ZDI top products

As discussed in detail in another section of this report, Java continues to place within the top five product submissions forwarded to the ZDI program by researchers. External researchers are shifting focus to client-side vulnerabilities more, thus targeting IE and Java. Looking at total surface area, it comes as no surprise that Internet Explorer is the number one targeted product for vulnerabilities in 2013. Web browser vulnerabilities overall more than doubled in 2013 with all but one attributed to Microsoft's flagship browser.

It is likely this is what prompted Microsoft to enter the white market in June of 2013. Its follow-on program included payouts for critical vulnerabilities that affected IE 11 Preview on Windows 8.1 Preview during a 30-day window.

Internet Explorer is also the number one product purchased in the ZDI program comprising just over 51% of all vulnerabilities purchased. Researchers are finding more complicated vulnerabilities in Internet Explorer—such as use-after-free—which is a trending vulnerability in the web browser's world.

Figure 6
Top five products by submission to the ZDI in 2013

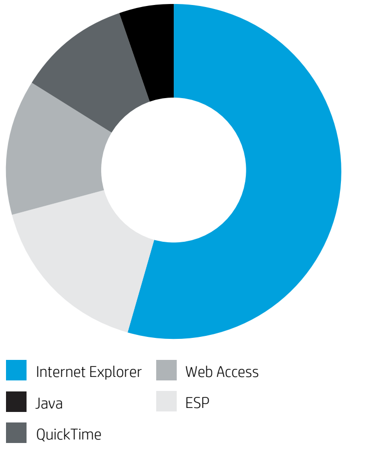
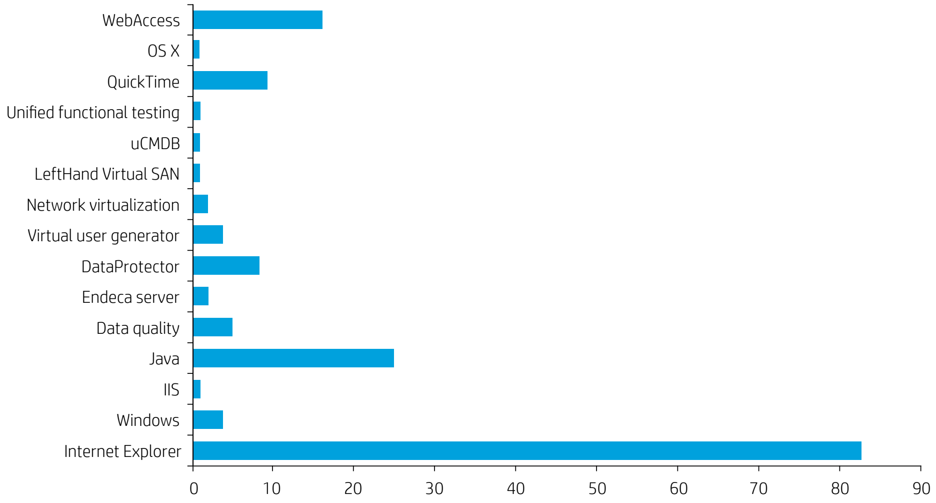


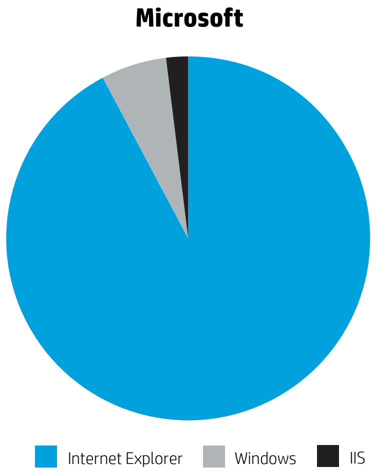
Figure 7
All products purchased by the ZDI in 2013

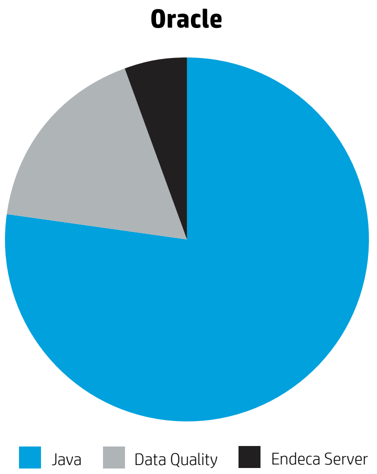


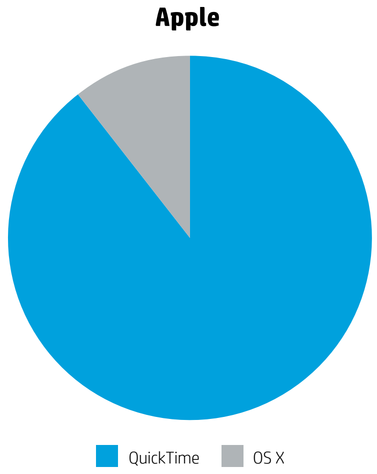
Diving into the top four vendors a bit more we see that, with the exception of HP, researchers are targeting relatively few products.

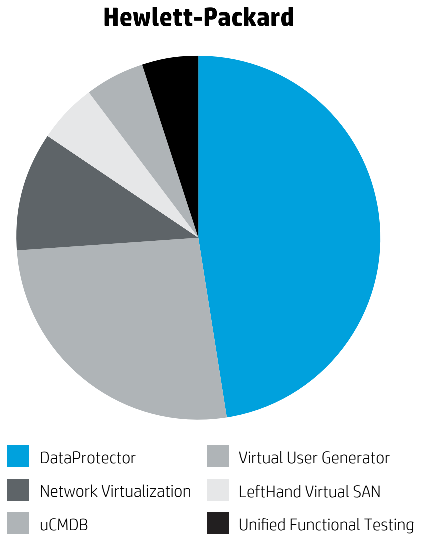
The takeaway

While there are small shifts in the landscape—less critical vulnerabilities, focus on client-side software, and increase in SCADA vulnerabilities—there are no major improvements for users. The threat landscape still exists and we continue to mitigate through patch management and identifying the chinks in the armor that may allow for targeted attacks.









Software security

To gain a true measure of the current state of software security, we analyzed the results of over 2200 audits performed by HP Fortify on Demand. Both static (performing a code level review) and dynamic (attacking the running application as an attacker would) analysis methods were employed. To accurately compare the two distinct methods of testing, we used the HP Fortify taxonomy[41] to examine the results.

The takeaway

From the applications tested over the past year, the clearest message our analysis uncovers is that organizations need to be more cognizant of what their applications are revealing post implementation. While the relative impact of vulnerabilities such as cross-site scripting and SQL injection is indeed significant, improvement is needed to decrease the amount of information provided to the attacker. In a similar vein, paying more attention to the data your application handles and its implied sensitivity will help define the necessary security controls for your application and, most importantly, where to apply them.

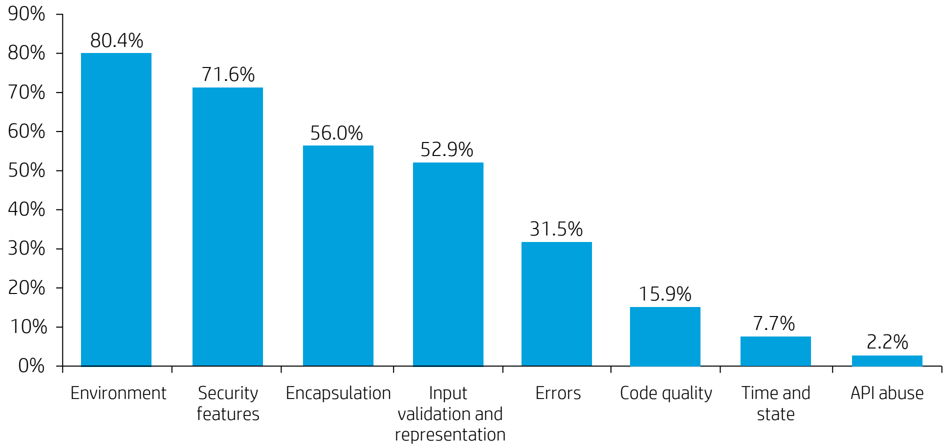
The HP Fortify taxonomy

The HP Fortify taxonomy was introduced in February 2006[42] as a means to identify a common vocabulary of software security errors. The taxonomy primarily classified vulnerabilities discovered through static analysis and, more recently, runtime analysis. New for 2014, HP Fortify will introduce dynamic analysis to the taxonomy; allowing for a single unified vocabulary across static, runtime, and dynamic analysis. For a more detailed description of the taxonomy, please refer to the online reference HP Fortify Taxonomy: Software Security Errors.[43]

Kingdom distribution

A kingdom in the HP Fortify taxonomy represents a collection of the software security errors related to a particular area of program functionality or resulting from the violation of a common principle. Each kingdom contains categories and sub-categories of software security errors. The following graph represents the distribution of kingdoms discovered in the test data.

Figure 4
Vulnerability category distribution by kingdom



At a glance, Figure 4 reveals nearly 81% of the applications we tested contained at least one vulnerability relating to the environment. Consider that most vulnerability categories within the environment kingdom reside outside of the source code; specifically, vulnerabilities related to server misconfiguration, improper file settings, sample files, outdated software versions, or other issues related to insecure application deployment. This reinforces the need for organizations to implement a layered approach to application security beyond solely static or dynamic analysis—focusing on holistic security topics that also consider liabilities related to the deployed application.

Next, 72% of the applications under test fell victim to improper implementation of key security features such as authentication, access control, confidentiality, cryptography, and privilege management. With the recent attention that's been given to breaches in confidentiality and "privacy" violations, this statistic suggests there's still room for improvement when it comes to effectively implementing sound software security principles throughout the development lifecycle.

Top five vulnerability categories

The following top-level vulnerability categories reside solely within the first four kingdoms shown in Figure 1. The percentage figure represents the proportion of applications that contained one or more of the vulnerability categories, and a further breakdown of sub-categories is provided for deeper insight into the specific problems observed.

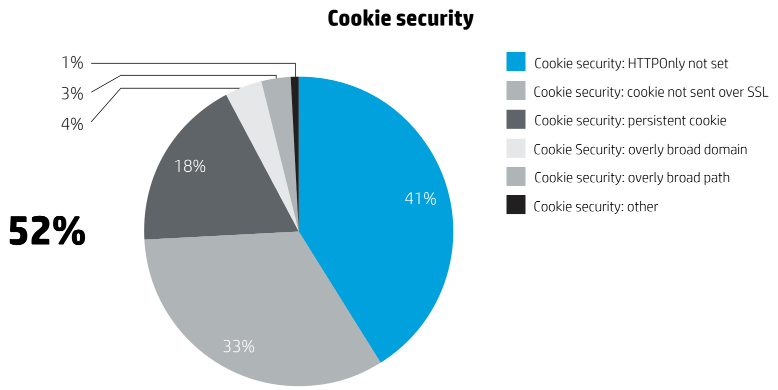
[41] hpenterprisesecurity.com/vulncat/en/vulncat/index.html
[42] Katrina Tsipenyuk, Brian Chess, Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors." In Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics, Elizabeth Fong ed., U.S. National Institute of Standards and Technology (NIST) Special Publication (SP) 500-265, February 2006.
[43] hpenterprisesecurity.com/vulncat/en/vulncat/index.html

In today's world of rising cyber crime attacks and the growing expectation of secure software implementation, it is imperative to limit any opportunity to unintentionally reveal information that may be beneficial to a malicious attacker. With 56% of the applications under test exhibiting weaknesses to revealing information about the application, its implementation, or its users, this is a disturbing trend that may very well be the low hanging fruit of choice for organizations to address first in their remediation strategy. Combined with 31.5% of the applications also prone to leak system information through poor error handling, this unfortunately only increases the propensity of giving an attacker the upper hand.

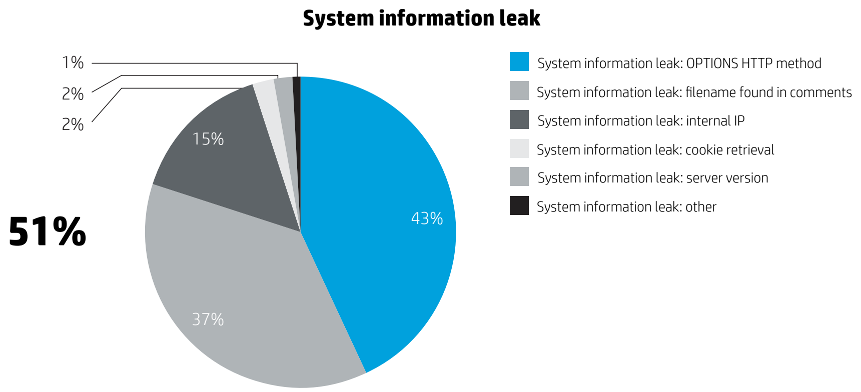
Input validation and representation, the kingdom containing popular categories such as cross-site scripting and SQL injection, affected 53% of the applications tested. Interestingly, in terms of instance count, cross-site scripting and SQL injection dominated the list, coming in first and second, respectively. However, and while cross-site scripting enters our top five list at number four, SQL injection only occurred in 9% of the applications tested compared to 40% for cross-site scripting. Many conclusions can be drawn about the under representation within our test data, but finding even a single instance of SQL injection remains one of the most impactful web application vulnerabilities. One may hopefully conclude that organizations are becoming more cognizant about SQL injection and proactively eliminating the root causes. The reality is that even after a decade and a half SQL injection still persists and thorough application testing remains a prerequisite.

Top five vulnerability categories

The following top-level vulnerability categories reside solely within the first four kingdoms shown in Figure 1. The percentage figure represents the proportion of applications that contained one or more of the vulnerability categories, and a further breakdown of sub-categories is provided for deeper insight into the specific problems observed.

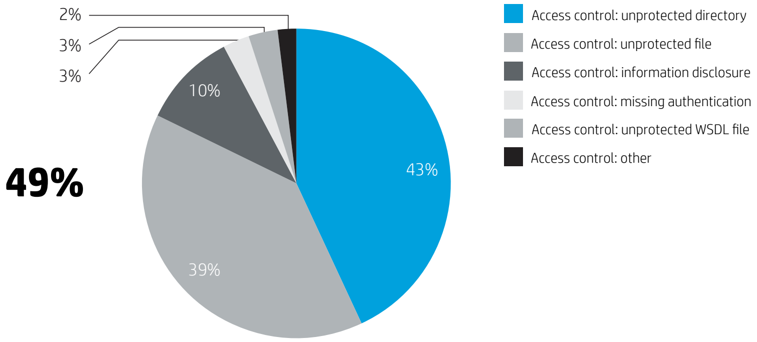


The first step in adopting a secure cookie strategy is to categorize the sensitivity of the information contained within each cookie. Is this cookie used for maintaining an authenticated state, or is it merely referenced for a benign user setting? Special care should be given to limiting just how much access client-side scripts should have to cookies, as noted by 41% of this category being attributed to not setting the HTTPOnly attribute. Related to information sensitivity, special care should be taken on how those cookies are transmitted and how long they remain on the user's system.



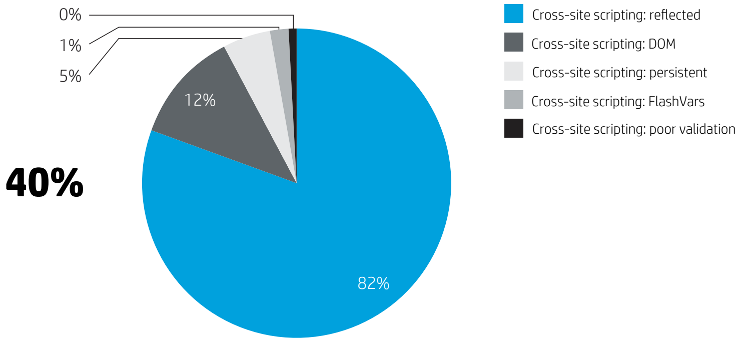
While their ethical boundaries may differ, security professionals and malicious attackers both craft attack strategies in a similar fashion. Mining information about the application that it freely reveals to the observer is a key first step in their overall methodology. Limiting the amount of readily available information the application, or it's implementation, forces the attacker to make assumptions or search for an easier entrypoint into your application, or preferably, to avoid it altogether and move on to another target. Furthermore, while a single piece of information gleaned from the application may appear insignificant alone, in combination it can become deadly. This category of vulnerabilities is really all about good housekeeping. The goal is to make it more difficult for an attacker to learn about your application.

Access control



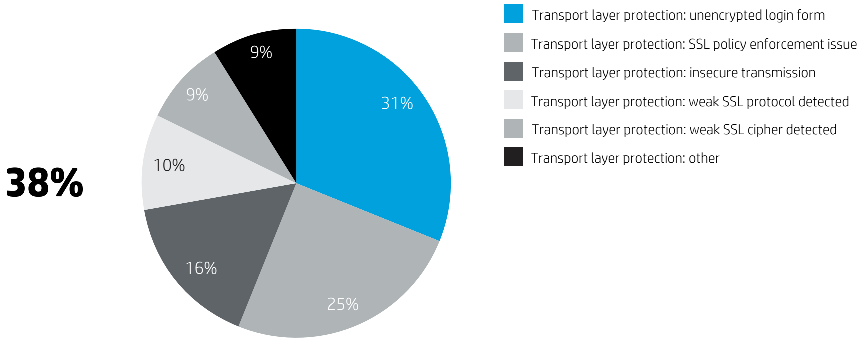
As with cookie security, care must be given to classify the sensitivity of any discovered unprotected files and directories within your application. With a combined 82%, unprotected files and directories dominated this category; showcasing how commonsense approaches to protecting information certainly aren't being followed. While concrete requirements that define sensitive areas of the application certainly should be present, ultimately it's up to the developer to ensure that the proper access controls are in place to protect access to both files and directories that are sensitive in nature.

Cross-site scripting



Arguably one of the most prolific vulnerabilities over the past decade, cross-site scripting stands at the top regarding the frequency in which it appears in the affected applications. While 82% of the affected applications demonstrated weaknesses to type one or "reflected" cross-site scripting, the category with the highest impact comprises a mere 5% of the applications: type two or "persistent" cross-site scripting. That's not to say reflected cross-site scripting isn't dangerous, but at least the highest impact vulnerability isn't in the majority.

Transport layer protection



Rounding out the top five vulnerability categories is another example of threats to confidentiality through poorly implemented or missing encryption. It's shocking to see 31% of this category attributed to unencrypted login forms; an obvious protection point for sensitive authentication information. Of course, just because you have SSL enabled doesn't mean it's done right, as is evidenced by errors in protocol or cipher selection in a combined 19% of the affected applications.

Although absent from the top five vulnerabilities, cross-frame scripting remains an important enabling attack vector that was present in 33.5% of the applications tested. As a case study analysis in last year's HP Top Cyber Security Risks Report,[44] cross-frame scripting was shown to be pervasive in sites across the Internet and remains significantly present within our current test data.

Open source case studies

Open source is an essential part of the enterprise environment, both as stand-alone applications and as components of commercial software. The same issues seen in software analyzed by HP Fortify on Demand are also found in open source applications.

A recent survey we performed of some popular open source applications discovered the following significant vulnerabilities predominant in the Fortify on Demand Data. These vulnerabilities are characterized by three kingdoms from the Fortify taxonomy.

Security features—cookie security: cookies not sent over SSL

```
public static void addCookie(HttpServletResponse
httpServletResponse, int cookieExpiration, String cookieKey,
String cookieValue) {

        Cookie cookie = new Cookie(cookieKey, cookieValue);

        cookie.setMaxAge(cookieExpiration);

        httpServletResponse.addCookie(cookie);

    }
```

A failure to use the setSecure method of the created cookie allows it to be transported in plain text. In this application, cookies were used to remember logged-in users. This is a simple error that allows attackers to easily impersonate users.

Solution: Configuring cookies to be sent over SSL would make such attacks more difficult.

Encapsulation—system information leak

```
<div align=”left”><font face=”Verdana, Arial, Helvetica, sans-
serif” size=”2”><%=UtilFormatOut.replaceString(errorMsg, “\n”,
“<br/>”)%></font></div>
```

[44] hpenterprisesecurity.com/collateral/whitepaper/HP2012CyberRiskReport_0213.pdf

In another open source web application, we find a default error.jsp page that writes full error messages to the page. The same application wrote sensitive information to exceptions such as stack traces, the servlet context path, and information about failed attempts to load resources. This is a classic example of how sensitive information is accidentally exposed.

Solution: Developers should be careful to provide the minimum amount of error information needed to investigate an issue in a web response.

Input validation and representation—path manipulation

```
} else if (DELETE_UPLOAD_FILES.equals(action)) {

String[] filesToDelete = httpServletRequest.
getParameterValues(FILE_TO_DELETE);

…

}
```

In this application, an unvalidated request parameter specifies a file to be deleted by the server. This does require the user to be logged in as an administrator, but the intended purpose of the code is to allow the user to delete a specified set of files, not arbitrary files.

Solution: All user-supplied parameters should be validated before being used to specify a file name or path.

For more information about the risks of using open source and a process to use for mitigating these risks, see our Threat Briefing No. 9.

Java every-days

In January 2013, the U.S. Department of Homeland Security urged users to uninstall Oracle's Java as vulnerabilities in the software were being actively and broadly targeted by attackers in order to compromise computers and install malware in the wild.[45] The ZDI team found itself at the coal-face of Java vulnerability research and reflects on its experience by providing the following detailed analysis of the attack surface created by the Java architecture, the root causes of the vulnerabilities involved, and information on how these vulnerabilities are being exploited in the wild.

The takeaway

Attackers are significantly escalating their exploitation of Java by simultaneously targeting multiple CVEs and being increasingly successful at compromising victim's computers. The real risk posed to organizations and individuals by attackers exploiting Java vulnerabilities to make compromises should not be understated. Make applying updates to address vulnerabilities in Java a priority for your organization and ensure that they are applied as soon as they become available. Consider reducing your organization's attack surface by removing Java from user's computers within your organization that do not require this technology for their role.

Introduction

HP's Zero Day Initiative (ZDI), the world's largest vendor-agnostic bug bounty program, experienced a surge in submissions for Oracle's Java platform in late 2012 and early 2013. It became a fairly regular occurrence for several new 0-day Java vulnerabilities to show up in the queue over a seven-day span. One of the more interesting trends revealed that ZDI researchers were not going after a single vulnerability class. At the time, the industry focused on sandbox bypasses and cases were arriving into the ZDI that took advantage of that weakness, but submissions identifying memory corruption vulnerabilities were still just as common. This prompted the following questions:

1. What is the most common vulnerability type in Java?
2. What part of the architecture has had the most vulnerabilities reported against it?
3. What part of the architecture produces the most severe vulnerabilities?
4. How do the vulnerabilities being used in the threat landscape map to the ZDI submissions?
5. How is Oracle responding to this increased pressure?

These questions continued to be discussed internally when exploit kit authors began including several new Java vulnerabilities during the first months of 2013. The targeted attacks against large software vendors and multiple 0-day vulnerabilities demonstrated at Pwn2Own were the final straw. We narrowed the focus for this paper to modern-day vulnerabilities and limited the scope to issues patched between 2011 and 2013. In total, we performed a root cause analysis on over 120 unique Java vulnerabilities including the entire ZDI dataset; major penetration testing tools; and exploit kits on the market today. We also included six 0-day vulnerabilities that have not yet been patched by Oracle but are part of the ZDI dataset. We reviewed and derived metrics about the threat landscape from a dataset that included 52,000 unique Java malware samples.

The ultimate goal of this analysis was to expose the actual attack surface that Oracle's Java brings to the table by taking an in-depth look at the most common vulnerability types, and examining the specific parts of the attack surface being taken advantage of by attackers.

[45] reuters.com/article/2013/01/11/us-java-security-idUSBRE90A0S32013011

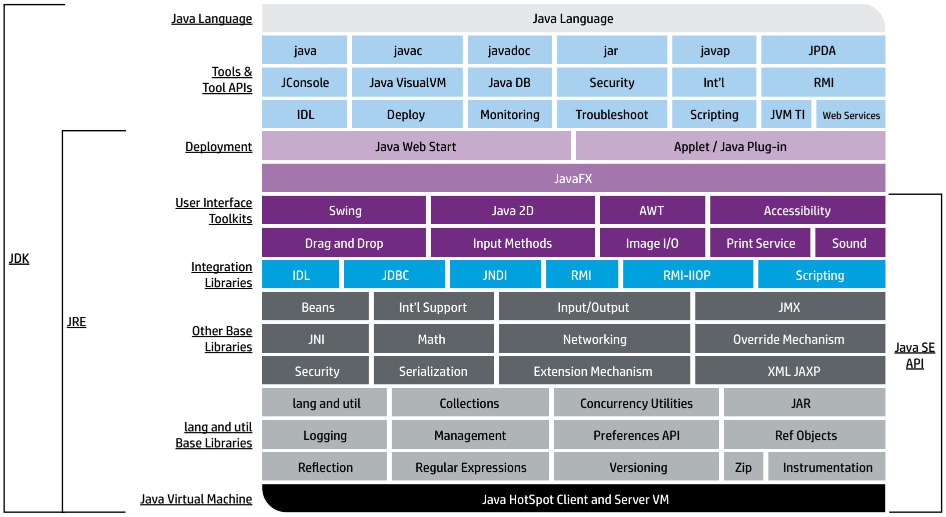
Oracle Java's footprint and software architecture

Oracle, quite famously, highlights the install base of Java via a splash screen during the installation of the product. For the software development community, a 3 billion device install base is a huge milestone. Alternatively for the security community, this constitutes a big red bull's eye.

Pair this with the statistics released from WebSense[46] that 93% of the Java install base is not running the latest patch a month after its release, or sometimes even a year after its release, these numbers become even more concerning. With such a broad install base and users running outdated software, the potential return on investment for attackers weaponizing Java vulnerabilities is considerable. Based on the numbers from Contagio,[47] exploit kit authors are required to include an average of 2+ Java exploits just to stay competitive with the other kits available on the market.

From the development perspective, the Java framework is powerful. It includes a large set of built-in capabilities to aid in the more complicated development tasks. As you can see in the conceptual diagram[48] below, the framework is made up of over 50 sub-components that bring different functionality to the table for developers. This includes capabilities to render a user interface, process complex fonts and graphics, and consume the most common web service protocols. Each sub-component provides a unique set of application programming interfaces (APIs) that developers can use to quickly extend their application.

Figure 5
Java 7 conceptual diagram



Applications can be written once and run on a multitude of platforms. Due to these factors, it is no surprise that Java has a widespread adoption in the development community. Java is also quite popular in the financial marketplace and recently made major inroads in the mobile device space. For all of these reasons, the security community has started to focus its efforts on analyzing and auditing this popular application.

[46] http://community.websense.com/blogs/securitylabs/archive/2013/06/04/majority-of-users-still-vulnerable-to-java-exploits.aspx
[47] http://contagiodump.blogspot.ca/2010/06/overview-of-exploit-packs-update.html
[48] http://docs.oracle.com/javase/7/docs/

Vulnerability trending and attack surface

Since early 2011, Oracle has patched almost 300 remotely exploitable vulnerabilities in Java. These issues range from the classic stack-based buffer overflow to the more complicated sandbox bypass vulnerabilities that require the attacker to chain a series of weaknesses to disable the SecurityManager. Every year the number of vulnerabilities being fixed has increased with just over 50 issues patched in all of 2011 to over 180 in 2013.[49] Researchers continue to discover new ways to find holes in the various sub-components of Java and bypass the security architecture.

Vulnerability statistics 2011–2013

Oracle Java patch statistics

Oracle maintains a consistent patch schedule with major security updates released approximately once every three to four months. Along with the software update, it releases a good amount of metadata for the vulnerabilities being fixed. This includes the CVE tracking identifier, a CVSS score, whether it is remotely exploitable, and the location of the vulnerability in the Java architecture. In the example below, CVE-2013-2383[50] seems to be a particularly nasty vulnerability in Java's 2D sub-component.

Figure 6
Oracle risk metric

CVE#	CVE-2013-2383						
Component	JAVA runtime environment						
Protocol	Multiple						
Subcomponent	2D						
Remote exploit without authentication	Yes						
CVSS Version 2.0 Risk (see Risk Matrix Definitions)	**Base score**	**Access vector**	**Access complexity**	**Authentication**	**Confidentiality**	**Integrity**	**Availability**
	10.0	Network	Low	None	Complete	Complete	Complete
Supported versions affected	7 Update 17 and before, 6 update 43 and before, 5.0 update 41 and before						
Notes	See note 1						

This information is useful to application developers when trying to quickly determine whether a particular vulnerability affects a component that their application relies on. It is also extremely useful to security researchers that are looking for the components in the architecture that contain a high number of security-related issues. Researchers can focus their attention on these areas, as they know their work will likely uncover similar issues.

Oracle's patch information over the last three years provides insights into the vulnerabilities being discovered. We observed that only three times in the last three years had a sub-component amassed a double-digit CVE count in a single patch. In February 2013, the Deployment and JavaFX sub-components had a CVE count of 10 and 12 respectively. The Deployment sub-component was again ravaged with a 12 CVE hit count in October 2013. Interestingly enough, all of these large fixes occurred in the 2013[51] patch releases (February and October). Oracle has also corrected security vulnerabilities in the 2D and Deployment sub-components in each of the patch releases since the beginning of 2011 (not including the security alert releases).

[49] oracle.com/technetwork/topics/security/alerts-086861.html

[50] oracle.com/technetwork/topics/security/javacpuapr2013-1928497.html

[51] oracle.com/technetwork/topics/security/javacpufeb2013-1841061.html

Looking at the last three years of patch information, the following sub-components account for half of the remotely exploitable vulnerabilities in Java:[52]

Figure 7
Most vulnerable sub-components

Rank	Sub-component	Average CVSS
1	Deployment	7.31
2	2D	9.12
3	Libraries	7.29
4	JavaFX	8.63
5	AWT	7.49

Ranking these sub-components by the number of unique CVEs, we discover that the Deployment sub-component is the most patched part of the architecture with almost 60 issues. That being said, the 2D sub-component contains the most severe vulnerabilities on average. It could be argued that the 2D sub-component is the riskiest component in the architecture due to the combination of its ranking and average vulnerability severity.

The average CVSS score for a remotely exploitable Java vulnerability is 7.57, which classifies them as High in severity.[53] Almost 50% of the issues fixed by the patches are CVSS 9.0 or higher with over 80 of those occurring in 2013.[54] If we look at what is being targeted year over year, we see that the security research community was focusing on the following sub-components:

Figure 8
Most targeted Java sub-components

Year	Most targeted sub-components
2011	1. Deployment 2. Sound 3. 2D
2012	1. Deployment 2. 2D and libraries 3. Beans and JMX
2013	1. Deployment 2. 2D and libraries 3. JavaFX

Zero Day Initiative (ZDI) submission trends

Many of the researchers working with ZDI take advantage of these statistics and watch for vulnerabilities being patched in specific sub-components. Our researchers typically focus on auditing one or two sub-components and become proficient, yielding new discoveries using a combination of techniques—some mine the patches to understand the weakness pattern and then hunt the attack surface for that pattern. Some simply look for nearby neighbors where Oracle engineers failed to find the same type of issue in the sub-components. Others look for deficiencies in the patch and re-submit those.

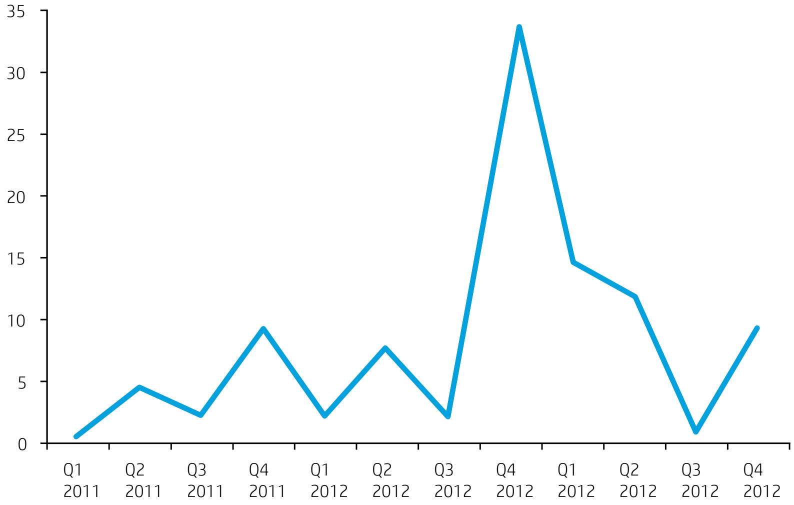
[52] oracle.com/technetwork/topics/security/alerts-086861.html
[53] oracle.com/technetwork/topics/security/alerts-086861.html
[54] oracle.com/technetwork/topics/security/alerts-086861.html

ZDI's submission rate for Java vulnerabilities maintained a consistent rate of approximately seven new vulnerabilities a quarter for the last three years. It is not surprising that the submission rate increased dramatically over the last three quarters with a high of 33 new vulnerabilities in one quarter alone. There are good explanations for this increased activity:

- High profile 0-day vulnerabilities drove researchers to look for related issues.
- Security Exploration's research[55] highlighted sandbox bypasses due to unsafe reflection.

Increased submission rates resulted in the largest patches released by Oracle for Java, with over 50 vulnerabilities fixed in the February and October 2013 patch cycles.

Figure 9
ZDI submission rate



Analyzing the submission trends we observed that the sub-components our researchers were targeting mapped to some of the buggiest parts of the Java architecture. Specifically, our researchers focused on the following sub-components most frequently:

1. 2D
2. Libraries
3. JavaFX
4. Sound
5. Deployment

Of particular note, they focus on the sub-components that produce the highest CVSS scores including 2D and JavaFX. Over the last three years, the average CVSS score for a ZDI submission was 9.38 and the researchers working through the program had accounted for a fourth of Java's vulnerabilities with CVSS score of 9.0 or higher.

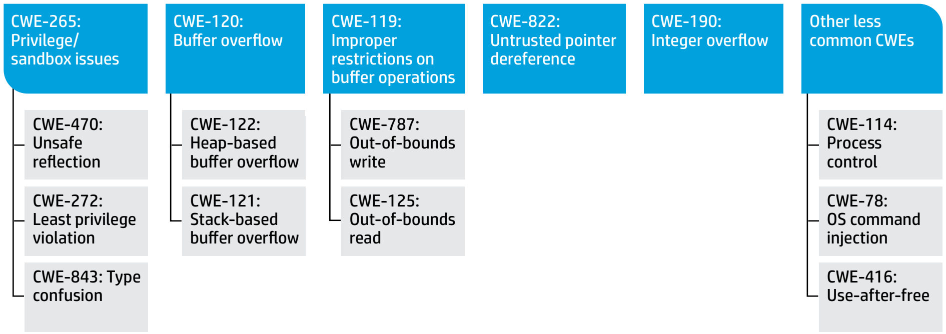
Vulnerability classes

Insights into vulnerability classes (CWE)

By intersecting publicly available vulnerability data with cases submitted to ZDI, we can shed light on what the most popular vulnerability classes are in the Java architecture. Luckily for vulnerability researchers, the architecture is susceptible to every common software weakness from the classic buffer overflow to command injection.

[55] security-explorations.com/en/SE-2012-01.html

Figure 10
Common weaknesses


CWE-265: Privilege/ sandbox issues
CWE-470: Unsafe reflection
CWE-272: Least privilege violation
CWE-843: Type confusion
CWE-120: Buffer overflow
CWE-122: Heap-based buffer overflow
CWE-121: Stack-based buffer overflow
CWE-119: Improper restrictions on buffer operations
CWE-787: Out-of-bounds write
CWE-125: Out-of-bounds read
CWE-822: Untrusted pointer dereference
CWE-190: Integer overflow
Other less common CWEs
CWE-114: Process control
CWE-78: OS command injection
CWE-416: Use-after-free

CWE-265 breakdown and historical timeline

The most prevalent issue in the framework is the ability to bypass the sandbox and execute arbitrary code on the host machine. About half of the vulnerabilities in the sample set had this designation. Not only was it popular with the ZDI researchers, but attackers also seemed to pick up on this weakness with nine CVEs related to the various styles of sandbox bypasses under active exploitation across the last three years. In early 2012, Security Explorations highlighted the sandbox bypass issue with the release of its research paper focused on this weakness.

Figure 11
Timeline of ZDI submission vs. actively exploited CVEs for CWE-265

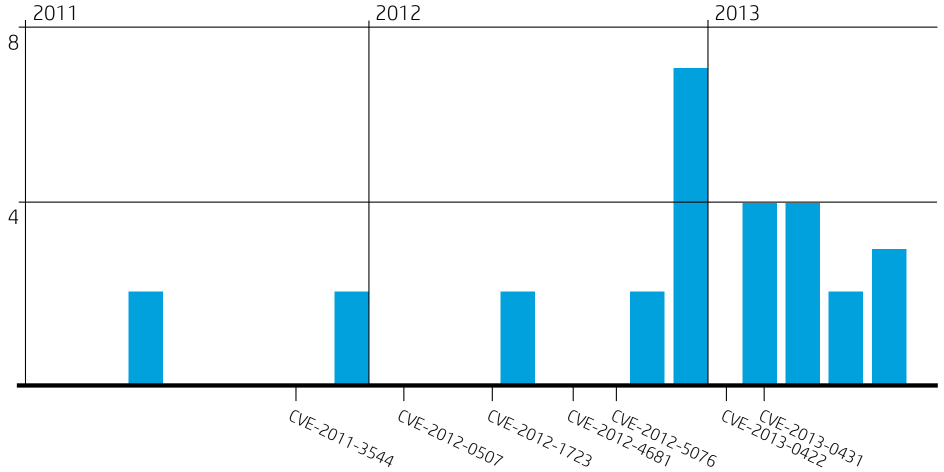
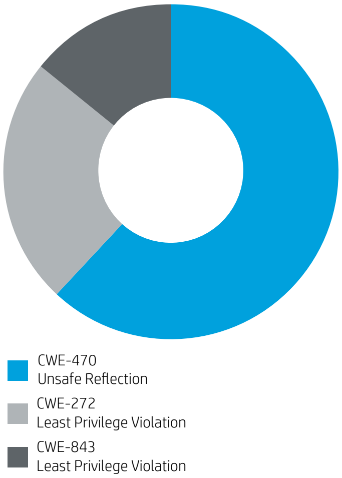


Figure 12
CWE-265 sub-category breakdown



ZDI researchers discovered these vulnerability types as early as April. As previously discussed, the unsafe reflection style of sandbox bypass is the most common technique being utilized with about 60% of the CWE-265 market share. CWE-470 Unsafe Reflection is also becoming the vector of choice for exploit kit authors with three of the most recent active targeted CVEs falling into this category (CVE-2012-5076, CVE-2013-0422, and CVE-2013-431).

There is a good reason for the focus on these vulnerability types in exploit kits and targeted attacks. They do not require the attacker to exploit memory corruption style vulnerabilities or bypass modern operating system mitigation techniques like data execution prevention (DEP) and address space layout randomization (ASLR). They do not need to write the exploit to a specific patch level of the operating system or application. This, arguably, takes away some of the challenges in place with other vulnerability classes and provides the attacker a “write once, own everywhere” exploit. In the end, focusing on discovering and productizing these types of issues gives the attacker a high return on investment.

Extrapolating sub-component weaknesses

Vulnerability class to sub-component mapping

You need to understand which packages make up the most vulnerable sub-components to fully grasp Java's attack surface. A sub-component's packages and classes can also be extremely useful when trying to analyze a security update from Oracle as a researcher can greatly reduce the scope of the code that needs to be audited to find the patched vulnerability.

Top seven vulnerability classes in the Java architecture

Based on our available information the top vulnerability classes and affected sub-components can be identified and targeted by the research community. The order of these issues can be further tuned by utilizing the sub-categories generated for the major weaknesses in the Java architecture. The table below provides a more accurate view into Java's attack surface.

Figure 13
Top seven vulnerability classes in Java

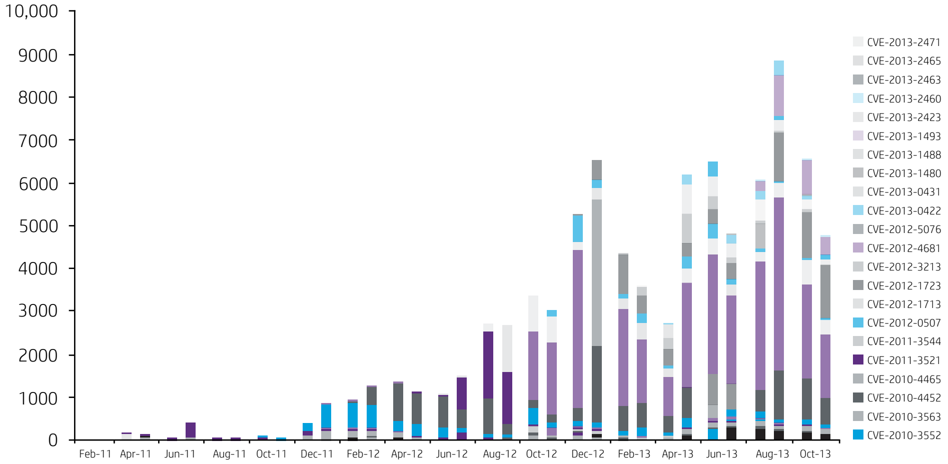
Rank	Common weakness enumeration	Sub-category	Sub-components
1	CWE-265: Privilege/ Sandbox Issues	CWE-470: Unsafe Reflection	AWT Beans HotSpot JAXP JAX-WS JMX Libraries
2	CWE-265: Privilege/ Sandbox Issues	CWE-272: Least Privilege Violation	CORBA JMX Libraries Scripting Sound
3	CWE-122: Heap-based Buffer Overflow	N/A	2D JavaFX
4	CWE-787: Out-of-bounds Write	N/A	2D Sound
5	CWE-822: Untrusted Pointer Dereference	N/A	JavaFX
6	CWE-122: Heap-based Buffer Overflow	CWE-190: Integer Overflow	2D
7	CWE-265: Privilege/ Sandbox Issues	CWE-843: Type Confusion	AWT Concurrency Deserialization Hotspot Libraries Scripting

Leveraging sub-component weaknesses

Exploit kit authors have jumped on the Java bandwagon offering a variety of exploits that leverage different vulnerability types. As stated previously, the kits on average need to offer 2+ Java exploits just to stay competitive in this market. Aligning this with the recent attacks using 0-day vulnerabilities, we derive unique insights into which software weaknesses are actually being leveraged in the threat landscape.

To further our understanding of the landscape, ReversingLabs provided us with a set of 87,000 unique Java malware samples. By looking at the scanning results of multiple anti-malware engines and ReversingLabs proprietary logic, these samples were classified into a set of categories based on the CVE they utilized. From this list, it was clear that 50% of malware classified as exploits carry a CVE identification, and by looking at last seen data, many new CVE-attributed exploits persist years after the vulnerability they target is publicly disclosed. This provided us with a list of the most common weaponized Java vulnerabilities over the last three years. In the graph below, the last three years of unique (by MD5 hash) CVE-attributed Java malware samples per month are shown.

Figure 14
Actively exploited CVEs



It is interesting that this timeline mirrors the increase in vulnerability discoveries by the external community over the last 12 months. Starting in August, the number of unique malware instances quickly shot up to close to the 3000 mark. More surprising is the huge jump in unique instances that begin in December and January of over 6000 against just 10 of the most common CVEs. More than half of those unique instances were labeled as CVE-2012-1723, which is a type confusion vulnerability in the HotSpot sub-component. January 2013 also saw a large increase in the use of CVE-2012-0507, another type confusion vulnerability in the Concurrency sub-component.

Antivirus engines do not always label samples correctly so the exact percentage of the unique samples per CVE inherently includes a small margin for error. As stated at the beginning of this paper we focus on the time period of 2011 to 2013. This graph is limited to the active CVEs during this time. This scope may have resulted in a less than accurate representation of activity in early 2011. The key takeaway is that attackers are significantly upping their game by targeting more CVEs than ever and are increasingly successful at getting their exploits onto victim machines.

Threat landscape

Aligning component weaknesses to attacks

As our goal is to understand the weaknesses at play in the landscape, we compared the list of actively targeted CVEs to the CVEs available through penetration testing tools and exploit kits tracked by Contagio.[56] By far, the most common vulnerability type for attack tools is the sandbox bypass using unsafe reflection to gain code execution.

Comparing the most popular software weakness across the attack tools to the most patched vulnerabilities, we see the following:

- Most common weakness included in attack tools
  1. CWE-265 Privilege / Sandbox Issues due to CWE-470 Unsafe Reflection
  2. CWE-265 Privilege / Sandbox Issues due to CWE-843 Type Confusion
  3. CWE-122 Heap-based Buffer Overflow
  4. 4CWE-265 Privilege / Sandbox Issues due to CWE-272 Least Privilege Violation
- Java's most patched weakness
  5. CWE-265 Privilege / Sandbox Issues due to CWE-470 Unsafe Reflection
  6. CWE-265 Privilege / Sandbox Issues due to CWE-272 Least Privilege Violation
  7. CWE-122 Heap-based Buffer Overflow
  8. CWE-787: Out-of-bounds Write

One intriguing occurrence is that the type confusion style of sandbox bypass switches place in the ranks with the least privilege style of sandbox bypass when it came to inclusion in the attack tools. The next logical question is: Which weakness is used more often in the exploit kits? The chart below describes the utilization breakdown for each software weakness across our malware sample set.

The clear "winner" is the type confusion style of sandbox bypass vulnerability with over half of the unique Java malware samples. Heap-based buffer overflow and out-of-band write vulnerabilities barely show up on the diagram due to the sheer volume of unique samples of sandbox issues.

Figure 15
CWEs utilized by attackers

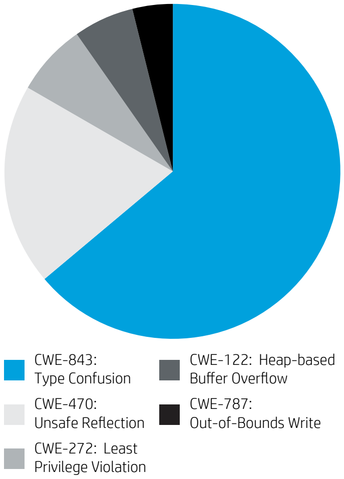


Figure 16
CWEs targeted by Pwn2Own contestants

Contestant	CVE	CWE utilized	
James Forshaw	CVE-2013-1488	CWE-265: Privilege/ Sandbox Issues	CWE-272: Least Privilege Violation
Joshua Drake	CVE-2013-1491	CWE-787: Out-of-bounds Write	CWE-125: Out-of-bounds Read
VUPEN Security	CVE-2013-0402	CWE-122: Heap-based Buffer Overflow	
Ben Murphy	CVE-2013-0401	CWE-265: Privilege/ Sandbox Issues	CWE-470: Unsafe Reflection

[56] http://contagiodump.blogspot.ca/2010/06/overview-of-exploit-packs-update.html

Java conclusions

Oracle has weathered quite the storm over the last eight months. Attackers continually discover and expose weaknesses in the framework and leverage those vulnerabilities to compromise machines. Exploit kit authors are upping the number of Java vulnerabilities they are including in their releases to stay competitive. The external research community is also focusing on the Java framework. Zero Day Initiative researchers continually identify a large number of vulnerabilities resulting in Oracle releasing some of its biggest security patches to date.

Based on this analysis, we have solid evidence that the sandbox bypass due to unsafe reflection is the most prolific issue in the framework but the sandbox bypass due to type confusion is the most exploited vulnerability type. Heap-based buffer overflows in the 2D component produce some of the most severe vulnerabilities but are not commonly used by the exploit community. Interestingly enough, each of the sub-components in the architecture appears to be vulnerable only to a subset of vulnerability types. With this information, researchers will be able to focus their efforts while auditing the sub-components to increase the chance of discovering some fresh 0-days.

We look forward to analyzing the next round of Java issues submitted to the Zero Day Initiative. We hope this information helps you to better understand the attack surface presented by Java and make the appropriate decisions regarding how it is used and how its updates are prioritized and managed in your organization.

Learn more at
zerodayinitiative.com
hp.com/go/hpsr

Pwn2Own 2013

In order to highlight the activity in the landscape, we expanded the scope of the Pwn2Own contest to include the browser plugins: Java, Flash, and Reader. When the contest launched, everyone seemed to be focused on the unsafe reflection style of the sandbox bypass vulnerability so our expectation was that we would only receive those types of bugs at the contest. In fact, our contestants leveraged four unique software weaknesses in order to win the prize money with these weaknesses including the top four vulnerability classes for Java defined earlier in the paper.

South Korean case study—a glimpse into the future and lessons on the nature of targeted attacks

The takeaway

As we discovered in the course of this analysis, even though the malware involved in this particular attack was not that sophisticated, it was still good enough to compromise the networks of several organizations and deliver a damaging payload that caused malicious damage and significant interruptions to normal function. The greatest lesson you can learn from this is that there isn't a single path to take to protect yourself and your vital business assets from threats. You need to take an approach that considers and prioritizes the value of your assets, understands the full attack surface, and realizes, most importantly, that security is a continuous and escalating process of information gathering, sharing, monitoring, and response.

In one of the biggest malware news stories of last year, on 20 March 2013, in a timed and coordinated attack, a malware payload was executed on computers belonging to a number of targeted businesses and organizations in South Korea, effectively crippling them for a short time. The attack targeted computers on networks belonging to several major banks—Shinhan Bank, Jeju, and Nonghyup, and television networks—YTN, MBC, and KBS (YonHap News Agency, 2013).[57] The effect of this attack was to take these networks offline and interrupt broadcasts and the normal operation of online and mobile banking apps and ATMs (ArsTechnica, 2013).[58] Some sources reported that the number of computers affected in the attack was around 48,000 (InformationWeek, 2013).[59]

This attack provides an interesting case study that allows us to see into the future to some degree. As one of the most wired countries in the world, the attacks in South Korea in 2013 provide some valuable lessons as we move in a similar direction. This case study provides a detailed analysis of the malware involved and discusses the possible security implications should this type of attack occur in your organization. We include advice on how to possibly protect your organization from this type of attack and steps you can take to remediate or limit the amount of damage that may be suffered as a consequence of a similar targeted compromise.

As we'll discover during the course of our analysis, there are some key lessons to be learned from the experience of the affected South Korean entities that can be applied to the broader understanding and application of security. These lessons reflect the changing nature of risk in light of well-resourced, financially motivated malicious actors in the context of an increasingly ubiquitous Internet.

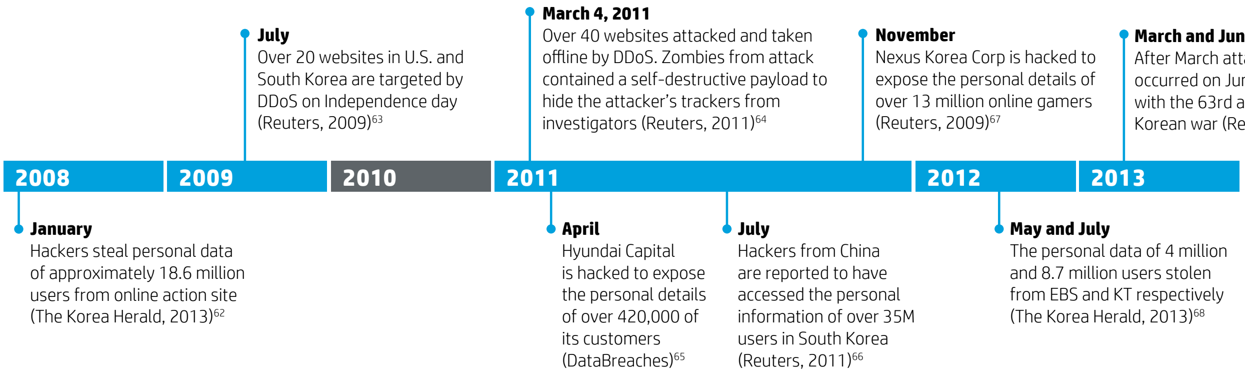
[57] http://english.yonhapnews.co.kr/news/2013/03/20/0200000000AEN20130320007951315.HTML
[58] http://arstechnica.com/tech-policy/2013/03/networks-of-south-korean-banks-broadcasters-hit-in-cyber-attack/
[59] informationweek.com/security/attacks/north-korea-behind-bank-malware-south-ko/240152644

Background

These attacks were not necessarily a new development for South Korea, which has experienced a number of notable cyber attacks since 2008. As an early adopter of Internet technology, South Korea is especially vulnerable to cyber attack as one of the most wired countries in the world, with figures from 2011 showing that more than 95% of households have permanent Internet access (Reuters, 2011).[60]

There are other economic and rather obvious political features that make South Korea a target for types of cyber crime or espionage. South Korea's strong economy and focus on industries where IP is critical makes it an attractive target for those so motivated—not to mention the strong competition that exists in these markets as an additional impetus. South Koreans enjoy a high standard of living compared to most of Asia. It has Asia's fourth largest economy and is the 15th largest in the world (by nominal GDP).[61]

While not a comprehensive list, South Korea has recorded multiple attacks in recent years:



2008 2009 2010 2011 2012 2013

January
Hackers steal personal data of approximately 18.6 million users from online action site (The Korea Herald, 2013)[62]

July
Over 20 websites in U.S. and South Korea are targeted by DDoS on Independence day (Reuters, 2009)[63]

March 4, 2011
Over 40 websites attacked and taken offline by DDoS. Zombies from attack contained a self-destructive payload to hide the attacker's trackers from investigators (Reuters, 2011)[64]

April
Hyundai Capital is hacked to expose the personal details of over 420,000 of its customers (DataBreaches)[65]

July
Hackers from China are reported to have accessed the personal information of over 35M users in South Korea (Reuters, 2011)[66]

November
Nexus Korea Corp is hacked to expose the personal details of over 13 million online gamers (Reuters, 2009)[67]

May and July
The personal data of 4 million and 8.7 million users stolen from EBS and KT respectively (The Korea Herald, 2013)[68]

March and June
After March attacks, more attacks occurred on June 25, coinciding with the 63rd anniversary of the Korean war (Reuters, 2013)[69]

The malware

As is typical for today's malware's modus operandi, these attacks were carried out using several different malware components. There were two main components to the attack that have been highlighted by several sources. These components include a Trojan dropper, which was used to deliver the wiper component, and the wiper component itself.

We do not have sufficient detail to state anything categorical regarding the nature of the initial compromise. However, it would be reasonable to suggest that the organizations were most likely initially compromised by attackers exploiting vulnerabilities in either the organization's technology or people (or a combination of the two). Spear-phishing and vulnerability exploits are commonly used in this context to make an initial point of access that can then be used by the attacker in furtherance of the attack. Publicly available information suggests that the outcome of the initial compromise was the installation of a remote access Trojan, possibly via spear phishing or watering hole attack, which was then used to install the Trojan dropper component.

[60] reuters.com/article/2011/06/14/us-cyber-south-korea-idUSTRE75D19P20110614
[61] princeton.edu/~achaney/tmve/wiki100k/docs/Economy_of_South_Korea.html
[62] koreaherald.com/view.php?ud=20130719000708
[63] http://english.yonhapnews.co.kr/news/2013/03/20/0200000000AEN20130320007951315.HTML
[64] reuters.com/article/2011/07/28/us-hackers-attack-idUSTRE76R19M20110728
[65] databreaches.net/hyundai-capital-in-south-korea-to-notify-420000-customers-of-data-breach-financial-watchdog-opens-investigation/
[66] reuters.com/article/2011/07/06/us-korea-cyberattack-idUSTRE76479M20110706
[67] reuters.com/article/2011/11/26/us-korea-hacking-nexon-idUSTRE7AP09H20111126
[68] koreaherald.com/view.php?ud=20130719000708
[69] reuters.com/article/2013/07/16/net-us-korea-cyber-idUSBRE96F0A920130716

The following sample was examined for this analysis:

Dropper
MD5: 9263e40d9823aecf9388b64de34eae54
Also known as/detected as :
Dropper-FDH (McAfee)
Trojan:Win32/Dembr.A (Microsoft)
Trojan.Jokra (Symantec)

The dropper component that we examined was distributed as a UPX-packed binary.

Installation
When executed it creates the following files in the affected user's %Temp% directory:

- alg.exe: A legitimate binary used to open SSH connections with remote servers
  MD5 e45cd9052dd3dd502685dfd9aa2575ca
  Size: 166,912 bytes
- conime.exe: A legitimate binary used to open SSH connections with remote servers
  MD5: 6a702342e8d9911bde134129542a045b
  Size: 153,600 bytes
- ~pr1.tmp: Payload - A destructive bash script
  MD5: dc789dee20087c5e1552804492b042cd
  Size: 1,186 bytes
  Also known as/detected as:
  KillMBR-FBIA (McAfee)
  Trojan:SH/Kofornix.A (Microsoft)
  Trojan.Jokra (Symantec)
- AgentBase.exe: Payload - Win32 wiper component (see details below)
  MD5: db4bbdc36a78a8807ad9b15a562515c4
  Size: 24,576

Payload—attempts to connect to remote servers and upload a destructive bash script

After determining the location of user profile directories on the affected computer, the malware searches these directories for configuration files and directories that may be associated with the connection manager clients mRemote and SecureCRT.

- mRemote—an open source tool for centrally managing remote server connections using a GUI (Kevin Kline, 2008).[70] This tool is no longer being actively developed or supported.
- SecureCRT—a commercial SSH and Telnet client by VanDyke Software.

If an mRemote installation is located, the dropper reads the configuration file and checks if there's a NODE that is defined with "Username=root", "Protocol=SSH", and a password that is not blank. If those conditions are satisfied it extracts the information. The password is decrypted after being extracted.

If a SecureCRT installation is located, the dropper extracts information from sessions that have Username=root, Protocol=SSH and a saved password. If these conditions are satisfied, the username, hostname, port, and password are extracted. The password is then decrypted.

After extracting these connection and server details, the dropper uses the previously dropped alg.exe and conime.exe to attempt to connect remote servers, upload and run the bash script ~pr1.tmp.

The bash script initially checks which UNIX it is running on (of HP-UX, SunOS, Linux, or AIX) and then attempts to wipe the /kernel, /usr /etc and /home directories, thus rendering the machine inoperative.

[70] http://sqlmag.com/net-framework/mremote

Win32 Wiper component

When the AgentBase.exe component is executed, it first attempts to stop the following processes, presumably in order to evade detection:

- pasvc.exe – policy agent from AhnLab
- clisvc.exe – ViRobot ISMS from Hauri

It then enumerates all physical drives and overwrites the first 512 bytes with the string: "princpes", effectively destroying the MBR (master boot record) of the affected drive.

It continues to look for removable and fixed drives, locates the root directory on these drives, and then attempts to delete all files and folders in this directory.

Finally, the affected computer is shut down and rebooted, although if the wiping mechanisms were successful then the machine will not be able to boot.

Apart from the obvious, disruptive payload of these attacks against particular targets, there has been some speculation that the ultimate goal of these attacks was sensitive data exfiltration. There has also been significant discussion regarding the perpetrators of these attacks and possible foreign state sponsorship. However, we do not have any further evidence beyond what is currently publicly available in this area and therefore can make no assertions in these matters at this time.

What can you learn, and more importantly, what can you do to protect your organization from this type of attack?

Regardless of the identity of the attackers or their motivations, there are lessons that can be learned from these events that can be meaningfully applied to reduce your organization's possible exposure to similar threats. Before we start with the specifics, let's briefly touch on some standard best practices for securing your organization. (Undoubtedly, you will already be familiar with the advice that follows, but these basics continue to remain important, and are not necessarily well implemented in practice):

- Maintain up-to-date antivirus software.
- Keep all operating systems and software patched against known vulnerabilities. Remove programs and disable services not in use to reduce the possible attack surface.
- Use strong passwords—apply policies as appropriate to ensure that your users create strong passwords and that they are changed regularly.
- As much as is practicable, apply the principle of least privilege to ensure that users and processes have the minimum privileges required to perform their functions and no more. For example, limit which users are able to download and run software.
- Use an NGIPS to detect and block malicious or suspicious behavior on the network.
- Use threat intelligence—subscribe to services and participate in communities that can identify when activities are being planned that may affect your industry.

Beyond the basic best practices above, reflecting on the South Korean attacks as an example of modern threat behavior provides us with the following additional lessons for protecting your organization.

Don't rely solely on traditional defensive perimeter security

While no one is denying the utility of more traditional, defensive perimeter security solutions, such as antivirus and firewall (we even recommend them—see above), the nature of today's threats means that these types of protections, while still important, are not enough. There's a couple of reasons for this, but one of the main reasons is that while many new developments have occurred in antivirus technology in recent years, in the main, most information security technologies such as signature based security tools still rely on reactive signatures in order to detect and remove threats. Which signatures are created, and therefore, which threats get detected rely in the main on intelligence that is gathered from the wild. Even more modern, proactive, behavioral signature detections rely on good intelligence on current threat behavior.

All well and good, but one of the key characters of targeted threats is their ability to get in under the radar—their ability to compromise networks and keep a low profile in order to avoid detection and removal, remaining in place until they perform their payload (regardless of what the payload is)—exactly as occurred in the Korean cyber attacks. Unfortunately, signature-based detection methods don't know what they don't know. The often bespoke, advanced (or otherwise as in this case) type of threats that are most likely to compromise your organization and steal your IP and financial and customer data are exactly the types of threats that signature-based detection product companies are less likely to have intelligence on, and therefore detect. You're trying to stop attackers at the perimeter, but ensure that your network security considerations don't stop at the perimeter as well.

With that in mind, and considering the capture and use of authentication and session details from SSH clients in order to access remote servers and upload a destructive script, as occurred in this case, we recommend that you consider stronger methods of authentication for connections to critical servers. Consider using public key as opposed to passwords (and ideally use a client that doesn't insecurely store or otherwise allow access to authentication and/or session details).

Remember that people are part of your organization's perimeter too

While the initial vector for the compromise of the affected networks in these attacks remains unknown, there are a number of different methods that are commonly used by attackers to bypass protections and gain access to networks. In the case of targeted attacks, as these were, one of the more common ways of making a compromise involves attacks that exploit particular vulnerabilities. However, as opposed to exploiting vulnerabilities in hardware or software, these attacks focus on taking advantage of and exploiting vulnerabilities in people.

These attacks fall under the broad umbrella of social engineering and commonly include, more specifically, spear phishing and targeted drive by downloads. Both attacks utilize initial research on the target:

- **Spear phishing** is essentially a targeted phishing attack. Phishing is a process by which attackers broadcast messages to victims, masquerading as a trusted entity (say, a bank) in order to request and steal sensitive information (such as user names and passwords). Spear phishing occurs when an attacker gathers specific information on an individual or group of individuals and uses that information to construct more convincing messages that are targeted to those individuals. Security researchers have speculated that the initial compromise in the South Korean attacks could have been accomplished by spear phishing the targeted organizations' employees.
- A **drive by download** is where an attacker compromises a website in such a way that he is able to install malware on visitors' computers, often by exploiting software vulnerabilities. A **watering hole attack** is one where intelligence is first gathered to determine which websites and services might be frequented by employees of the targeted organization and then those websites are specifically targeted for compromise and hosting malicious code.

Don't forget about social media

Our examinations of the tactics used by the Syrian Electronic Army (SEA) this year have taught us some important lessons about the risks posed by social media and how it contributes to the attack surface. As we saw in our Threat Briefing no. 3, the SEA has been proactive in its use of social media in targeting its aims—for example, by hacking the credentials of existing accounts in order to spread propaganda or leak captured information, distribute malware, or participate in phishing attacks.

Remember your social media assets and take appropriate hardening precautions:

- Monitor corporate Facebook pages for spam comments.
- Monitor Facebook and Twitter accounts for compromise.
- Enforce strong passwords.
- Be particularly vigilant to monitor for phishing attacks.
- Maintain unique passwords for each social media site. Avoid re-using passwords.
- Monitor infrastructure for DDOS and SQL injection.
- Monitor corporate websites for any out-of-process changes.
- Be aware of fake sites. Double check the domain and URL. If your experience with the site seems to be abnormal—such as advertisements appear out of the ordinary or your friend lists look different—avoid from posting any information until you can verify that the site is real.

Consider your people part of the possible attack surface and take steps to "harden" them as you would your hardware and software. While you can't patch people per se, you can educate them on:

- Types of attack they may be targeted by
- Identifying sensitive information assets and how those assets should be used
- Types of information request to view with suspicion and how to recognize illegitimate influence attempts
- Types of behavior that might be indicative of a possible compromise
- How to report suspicious messages or unexpected behavior

While technology, policy, and process can be used to minimize the attack surface, it is unlikely that you would be able to completely eliminate the people from your organization. Forewarned is forearmed—teach your people to be a security asset.

Note: While it is strongly recommended that you have an ongoing security education program for your people, in order to back this defense up, consider both limiting users' privileges to ensure that they are unable to introduce unknown binaries to your network, or at least use a reputation service that will allow for vetting of possibly malicious traffic in your environment.

Expect to be compromised

It's an uncomfortable observation, often made by experts in the field that it is near impossible for organizations to secure their networks completely in the face of APTs. This disquieting truth needs to be taken into account when planning how to keep assets secure. Expect the worst and plan appropriately. The South Korean attacks of March 2013 were only successful because the malware controllers were able to secretly and stealthily establish a beachhead on up to 48,000 computers from where they could launch their destructive payload. When the disk-wiping payload was delivered on the day of the attack, the malware that downloaded was speculated to have been in place on the affected networks for some time. This malware persisted, undetected in secret, until the malware controllers wanted to use it to perform this damaging attack.

To that end, have tested, well-communicated, verified plans in place for when—not if—the worst happens. At a bare minimum:

- Perform daily backups of critical assets with storage offsite and offline and practice recovery so if your data gets wiped (as in this particular example) you can restore it.
- Create an emergency communications plan ready to be used if the network fails.
- Have accessible references ready on how to restore critical systems and resources dedicated to contingency.
- Test your plans and documentation.
- Isolate critical systems from other parts of the business and ensure that they can be brought back online independently of other less-critical systems.
- Encrypt sensitive or business-critical information (make the information unusable even if it does get captured).

Understand that not all information and network assets are equal

During the course of investigating these attacks, an interesting fact came to light that goes some way to explaining how the attackers were able to deliver the payload component to so many computers in such an effective fashion. By getting hold of the credentials for an enterprise patch management server, the attackers were able to use it to serve the malware to multiple computers, as you would updates (Ahnlab, 2013). Systems that facilitate centralized management functions and play a central role in establishing trust in networks are highly prized targets of compromise and could be used to make other security controls useless. Prioritize resources to identifying and protecting critical assets first.

Make security and response a continuous process

Unfortunately, the security of an organization isn't something you can "set and forget." An organization's security is accomplished, or rather, increased, through a system of continuous, at times, escalating processes that reduce the risks of adverse outcomes created by today's threats.

In these cyber attacks, attackers were able to bypass traditional protection measures in order to compromise the targeted networks and persist until such time that they chose to deliver a damaging payload. The attackers needed to accomplish a number of different steps in order to accomplish their end goal. It is likely, that at least some of those steps or processes would appear anomalous to the regular pattern of use or behavior of users in that particular environment.

If traditional signature detection mechanisms are not as useful, and compromise occurs, there must still be mechanisms in place to flag aberrant behavior and respond appropriately. Do you know what's normal behavior on your network? Would you be able to identify meaningfully unusual behavior on your network and respond in real time to an attack before damage, or further damage occurs?

Of course, being able to flag and respond to unusual events in real time is no small feat for any organization, but by using continuous monitoring and gathering data that enables you to know what's normal and what's not within the context of your organization you can increase the likelihood of detecting advanced threats that have breached the perimeter before they take hold and create more damage (possibly through malicious disruption or data exfiltration).

We also recommend that you perform active analysis of event data to detect anomalous activities and to identify indicators of compromise that are not already known. The most successful companies utilize a host of methods and tools that range from data visualization and SIEM to big data solutions and data scientists leveraging custom algorithms.

Seek out credible and reliable security intelligence

It's difficult to protect your organization and its assets unless you know your most critical assets, what's being targeted, and what is threatening you—how it works, its methods. You need to know your enemy and be familiar with its current modus operandi. You can use this intelligence to ensure that your perimeter security and ongoing monitoring are appropriately configured for current conditions.

Having sources of accurate information that you trust and can use in this way to keep your protections both appropriately reactive to current threats and proactive against new trends is imperative. Ideally, partner with an authoritative source of security intelligence that captures data from multiple sources on the threat landscape and analysis from experienced security experts.

It is very difficult if not impossible for any one entity to get a good picture of the breadth and depth of the threat landscape. While threat information is often proprietary in some examples, enterprises should be encouraged to share their experiences of compromise with the greater security community and other enterprises. Often attackers use campaigns that target multiple organizations at once—sharing information on current attacks and methods makes everyone less vulnerable and the attackers less likely to succeed. We are stronger together.

Conclusions

The complexity and difficulty of securing enterprises only grows with the passage of time. But with the right information, organizations can significantly reduce their attack surface, substantially mitigate risks, and prevent the losses and damages associated with successful attacks. To that end, we examined the areas that significantly contributed to the growing attack surface in 2013—Oracle Java, mobile devices, and web software security. We provided insight into the current vulnerability landscape and the growing market of Zero Day research describing the most targeted software and corresponding implications to the overall threat landscape. And finally, we examined the current state of threats to smartphones and the implications for those who plan to develop applications for such devices. Our results reveal that securing enterprises remains a tenuous proposition, and significant opportunities for improving security efforts exist.

The results of our analysis on vulnerability data from the HP Security Research Zero Day Initiative demonstrate the increasing overlap in focus between researchers and adversaries on what software to target. Put simply, researchers are keen to work collaboratively with heavily targeted software vendors in an attempt to find, fix, and eliminate such issues from the software before adversaries identify and add them to their arsenal leaving defenders without a patch to secure their environment. Results also show there is room for improvement from an operational perspective. More specifically, we continue to struggle with using risk management techniques for patch deployment, which introduces a significant security gap—one that software itself cannot address. Defenders must understand what software adversaries are targeting and leverage that information to prioritize their patch management efforts.

In a similar vein, using information from our analysis into the weaknesses in web software security by Fortify on Demand it is clear that a large percentage of vulnerabilities lie with insecure configuration. Eliminating bugs and the resultant vulnerabilities from code won't fix this—even perfectly coded software can be dangerously vulnerable when misconfigured. Just as important as using risk management in our patching efforts defenders must also remain vigilant by auditing for insecure application deployment and similar weaknesses (such as providing too much error information) that make it trivial for adversaries to compromise and commandeer your web applications for nefarious means (such as redirecting to exploit kit sites).

Considering the increasing use of smartphones to store and access sensitive corporate data, we examined the prevalent OS platforms from a malware, vulnerability, and exploit perspective. The results of our analysis were clear, unfortunately significantly favoring the adversary. Our Mobile Pwn2Own competition showed that existing vulnerabilities could be exploited to see the same kinds of resulting compromises and payloads as we see on more traditional platforms. Same methods of attack, same avenues of compromise, same targeted information and resources, same payloads. The real difference is users don't expect these attacks on mobile platforms and hence aren't modifying their behavior accordingly considering the level of risk. While the results of the competition proved the viability of such attacks our research into mobile malware as well as the mobile development ecosystem clearly demonstrated the existence of much lower hanging fruit. As is the case with cyber criminals they will always take the path of least resistance (as measured by time, complexity, and cost to carry out an attack), which means we generally aren't likely to see the types of attacks carried out during our Mobile Pwn2Own competition gain any kind of significant prevalence for some time.

The world is abuzz on the "Internet of Things" and what that means to the future. Organizations are increasingly looking to build mobile apps that promise newfound productivity and efficiency gains, connecting people to corporate assets and sensitive data. Looking at the results of our analysis on smartphones it is clear that if we're not careful history—in this case from the desktop arena—will repeat itself in more ways than should be allowed. The future of technology is an exciting one and even more challenging from a security perspective as every year passes by. However, with greater cooperation and a strong understanding of the attack surface of today and tomorrow we can gain traction and take back ground from the adversaries by making it more costly for them to be successful.

For more information on the HP Enterprise Security tools mentioned, go to hp.com/go/sirm.

Authors and contributors

The HP 2013 Cyber Risk Report is an annual collaboration among groups within HP Enterprise Security Products, including: HP Security Research (spanning HP DVLabs, HP Fortify Software Security Research, and the HP Zero Day Initiative), HP TippingPoint, HP Fortify, and HP ArcSight.

We would like to sincerely thank Mario Vuksan and his team at ReversingLabs for the contribution of research, data, and reprint rights.

Authors	Contributors
Joy Marie Forsythe	Jesse Michael Emerson
Brian Gorenc	Alexander Hoole
Heather Goudey	Prajakta Jagdale
Abdul-Aziz Hariri	Jason Lancaster
Scott Lambert	Matias Madou
John Park	Matt Molinyawe
Joe Sechman	Alvaro Munoz
Nidhi Shah	Sasi Siddharth Muthurajan
Jasiel Spelman	Yekaterina Tsipenyuk O'Neil
Jewel Timpe	Ted Ross
Jacob West	Shannon Sabens
	Todd Tabor
	Mario Vuksan (ReversingLabs)
	Dave Weinstein
	Stephanie Wisdom
	Diana Wong

4AA4-5495ENW, January 2014, Rev. 2

This is an HP Indigo digital print.