White paper

# HP 2012 Cyber Risk Report

# Table of contents

# Overview

In the *HP 2012 Cyber Risk Report*, HP Enterprise Security provides a broad view of the vulnerability landscape, ranging from industry-wide data down to a focused look at different technologies, including Web and mobile. The goal of this report is to provide the kind of actionable security that intelligence organizations need to understand the vulnerability landscape as well as best deploy their resources to minimize security risk.

To provide a broad perspective on vulnerabilities, the report draws on the following sources:

- Open Source Vulnerability Database (OSVDB)[1] data
- HP Zero Day Initiative (ZDI)[2] vulnerability data
- HP DVLabs vulnerability and exploit analysis
- HP Fortify on Demand[3] static and dynamic security testing data
- HP Fortify Software Security Research Web vulnerability research
- Security Compass (HP partner) mobile vulnerability data

Based on this data, the report offers the following key findings:

## Critical vulnerabilities are on the decline, but still pose a significant threat

High-severity vulnerabilities (CVSS[4] score of 8 to 10) made up 23 percent of the total scored vulnerabilities submitted to OSVDB in 2011 and dropped to 20 percent in 2012. While this reduction is significant, the data shows that nearly one in five vulnerabilities can still allow attackers to gain total control of the target.

## Mature technologies introduce continued risk

As demonstrated by the recent Department of Homeland Security announcement recommending that the Oracle Java SE platform be universally disabled in Web browsers, seemingly mature technologies still suffer from new exploits. In particular, 2012 data show the number of vulnerabilities disclosed in Supervisory Control And Data Acquisition (SCADA) systems increased from 22 in 2008 to 191 in 2012 (a 768 percent increase).

## Mobile platforms represent a major growth area for vulnerabilities

The explosive adoption of mobile devices and the applications that drive them has resulted in a corresponding boom in mobile vulnerabilities. The last five years have seen a 787 percent increase in mobile application vulnerability disclosures, with novel technologies, such as near-field communications (NFC), introducing previously unseen vulnerability types.

## Web applications remain a substantial source of vulnerabilities

OSVDB data from 2000–2012 shows that of the six most submitted vulnerability types, four—SQL injection, cross-site scripting, cross-site request forgery, and remote file includes—exist primarily or exclusively in Web applications.

## Cross-site scripting remains a major threat to organizations and users

Cross-site scripting (XSS) remains a widespread problem, with 44.5 percent and 44 percent of the applications in our data sets suffering from the vulnerability. In one case, analysis of a multinational corporation showed that just under half (48.32 percent) of their Web applications were vulnerable to some form of XSS. Furthermore, new methods of exploiting this vulnerability continue to be found, as demonstrated by the large portion of ZDI vulnerability submissions focused on XSS.

[1] Open Source Vulnerability Database: osvdb.org/
[2] HP Zero Day Initiative: zerodayinitiative.com/
[3] HP Fortify on Demand: fortifymyapp.com
[4] Common Vulnerability Scoring System: first.org/cvss/cvss-guide.html

### Effective mitigation for cross-frame scripting remains noticeably absent

The first documented cross-frame scripting (XFS) vulnerability, the root cause behind clickjacking attacks, was discovered over 10 years ago. Since then, clickjacking has become a household name, yet less than one percent of 100,000 URLs tested included the best-known mitigation, the X-Frame-Options header.

## Vulnerability trends

Understanding technical security risk begins with knowing how and where vulnerabilities occur within an organization. Vulnerabilities can impact every level of enterprise infrastructure from hardware, to network, to software (both old and new). These vulnerabilities are the gateway that malicious actors use to circumvent security protections and steal or alter data, deny access, and compromise critical business processes.

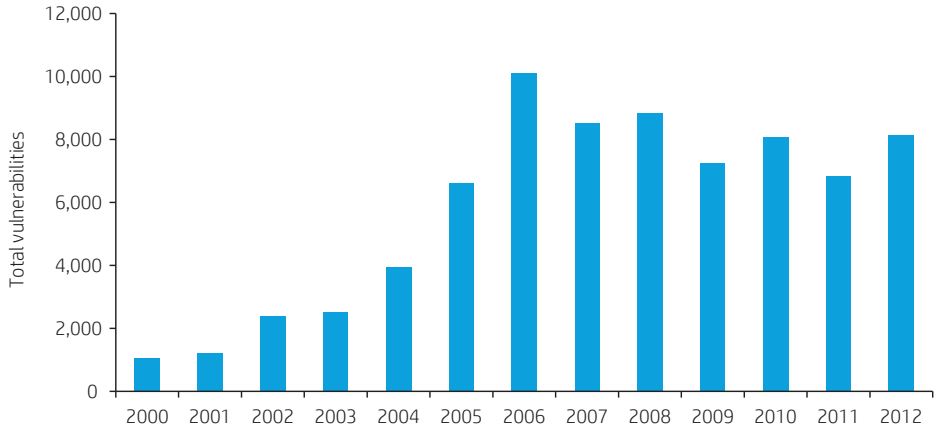
This section of the report uses data from the Open Source Vulnerability Database (OSVDB) and the HP Zero Day Initiative (ZDI) to demonstrate the following global vulnerability trends:

- **The vulnerability arms race—total vulnerability disclosures in 2012 increased 19 percent from 2011.** The total number of vulnerabilities reported provides a snapshot into the world of vulnerabilities and serves to illustrate the nature of a constantly changing threat landscape.
- **Evolving marketplaces and increasing complexity impact discovery and reporting.** Vulnerability disclosure data highlights how changes in the vulnerability marketplace and the technical complexity of systems impact both the number and severity of reported vulnerabilities.
- **Web applications continue to introduce significant technical risk to organizations.** A small number of critical Web application vulnerabilities still represents a large minority of the overall vulnerabilities disclosed in 2012.
- **The maturity of a technology does not govern its vulnerability profile.** Data in 2012 shows an increase of more than 700 percent in vulnerability disclosures impacting both SCADA systems (primarily legacy technology) and mobile devices (the next frontier for IT).

### The vulnerability arms race—total disclosures in 2012 increased 19 percent from 2011

The total number of new vulnerabilities reported during 2012 (8,137) increased by roughly 19 percent from the number of vulnerabilities disclosed in 2011 (6,844), but remained 19 percent lower than the number reported at the peak in 2006. This continued oscillation in the number of reported vulnerabilities demonstrates that the struggle between organizations and the attackers bent on compromising them rages on with no clear victor (see Figure 1).

**Figure 1.** Disclosed vulnerabilities measured by OSVDB, 2000–2012



In the year-by-year view of OSVDB data, it is clear that vulnerability reporting has oscillated since its peak in 2006, with no clear up or down trend in the years that followed. The number of vulnerabilities disclosed in a given year doesn't necessarily measure the overall security of the industry, but rather indicates how changes in the way vulnerabilities are discovered, disclosed, and exploited can vary greatly from year to year. The next section highlights how changes in the vulnerability marketplace are partly responsible for these variations.

## Evolving marketplaces and increasing complexity impact discovery and reporting

While reported vulnerabilities have remained well below the peak seen in 2006, it is important to look at factors that impact the discovery and disclosure of vulnerabilities. Vulnerability information can be disseminated through a number of different outlets, including community programs such as OSVDB or HP ZDI, private security consultants, manufacturer bug bounty programs, and the underground black market.

While OSVDB provides an excellent snapshot of the vulnerability landscape, it can only count vulnerabilities that are disclosed publicly or submitted directly to the organization. Increasingly, specialized security consulting agencies are discovering and purchasing vulnerabilities that are then only disclosed directly to their private groups of clients. This practice leaves a significant quantity of vulnerabilities uncounted in public tallies.

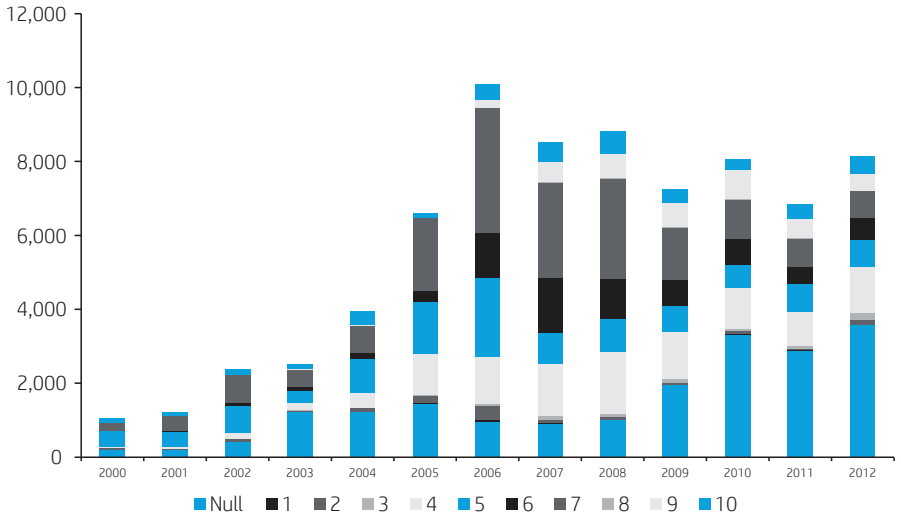
Further complicating the challenge posed by vulnerabilities is the complexity of software today. As security has become an increasing priority, organizations have been fighting back against attackers by building security into their software and adding features to thwart the unauthorized discovery and exploitation of vulnerabilities. These countermeasures have not only helped close a number of holes, but have also made the remaining vulnerabilities more difficult to discover.

The bifurcation of public and private vulnerability marketplaces, combined with the increasing complexity involved in identifying vulnerabilities, has driven up the value of highly exploitable vulnerabilities (those with a CVSS rating of 8 to 10). This increase suggests two conclusions:

- Security researchers must develop extensive expertise in specific systems to remain effective.
- Researchers receive a better return on investment for severe vulnerabilities that fetch a higher price.

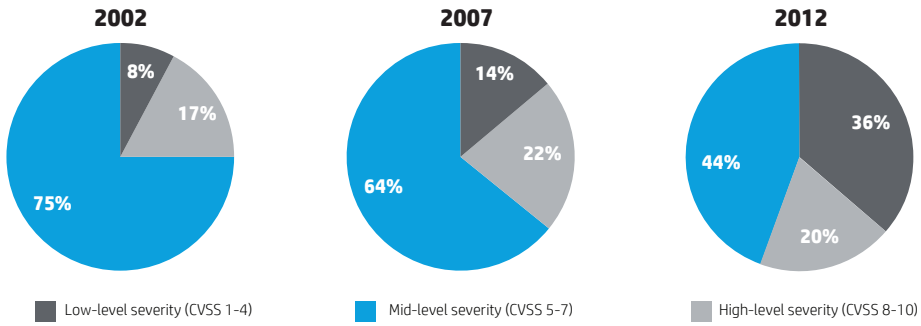
However, while the total number of highly exploitable vulnerabilities reported by OSVDB in 2012 increased nominally, the percentage of the overall vulnerabilities reported with a high CVSS score decreased from 23 percent in 2011 to 20 percent in 2012 (see Figure 2). Note that OSVDB does not require a CVSS score to report vulnerabilities. Vulnerabilities with no CVSS score are listed as Null in the figure.

**Figure 2.** Vulnerability severity landscape using OSVDB data, 2000–2012



In contrast with this modest decrease in the number of highly exploitable vulnerabilities reported in 2012, the overall trend in OSVDB data shows that the percentage of this type of vulnerability has grown substantially and consistently over the last decade (see Figure 3).

**Figure 3.** Severity of OSVDB vulnerabilities broken out over 10 years



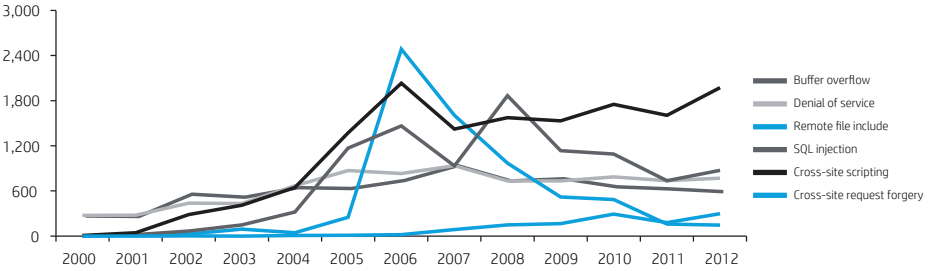

From 2002 through 2007, mid-severity vulnerabilities (CVSS 5 to 7) made up the bulk of the disclosures. This period included a huge increase in the overall number of vulnerabilities reported as fuzzing tools became mainstream and researchers began finding many of the more common, easier-to-find vulnerabilities using automation. This number has since dropped sharply, in part because development organizations have been leveraging automation to identify and resolve these vulnerabilities before researchers can find them.

What, then, explains the decreasing percentage of highly exploitable vulnerabilities reported in 2012? Market data suggests that because of the expertise and time required to uncover and prove the exploitability of very severe vulnerabilities, an ever-increasing number of this type of vulnerability is being sold to private-commercial (gray) or underground (black) markets, keeping them (at least temporarily) out of public counts like OSVDB.

## Web applications still introduce significant risk to enterprises

Figure 4 highlights the six most common vulnerability categories reported in OSVDB: buffer overflow, denial of service, remote file include, SQL injection, cross-site scripting, and cross-site request forgery. All are potentially remotely exploitable.

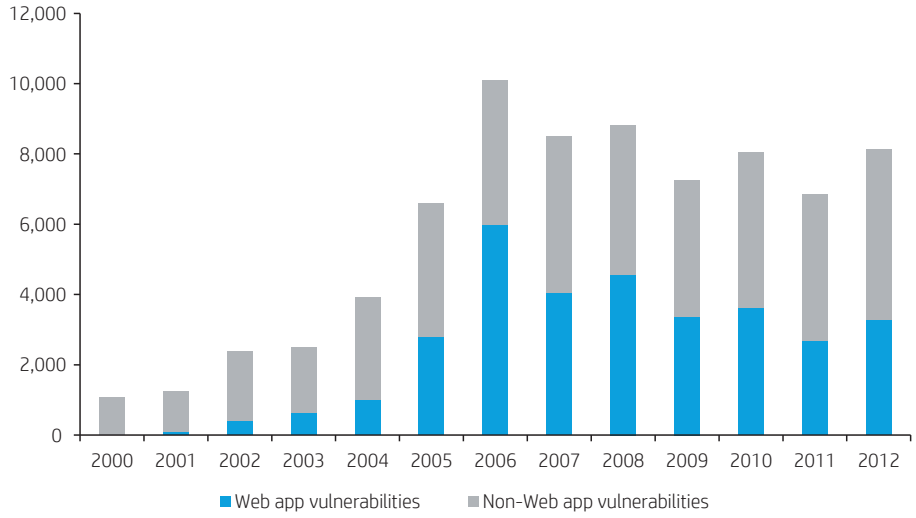
**Figure 4.** Most common vulnerabilities in OSVDB, broken down by category, 2000–2012




Of these six vulnerability categories, four—SQL injection, cross-site scripting, cross-site request forgery, and remote file include—primarily or exclusively impact Web applications and account for 40 percent of overall vulnerability disclosures in 2012 according to OSVDB.

While the number of Web vulnerabilities peaked in 2006, it has since varied in much the same way as the overall vulnerability numbers (see Figure 5).

**Figure 5.** Web application vs. non-Web application vulnerabilities

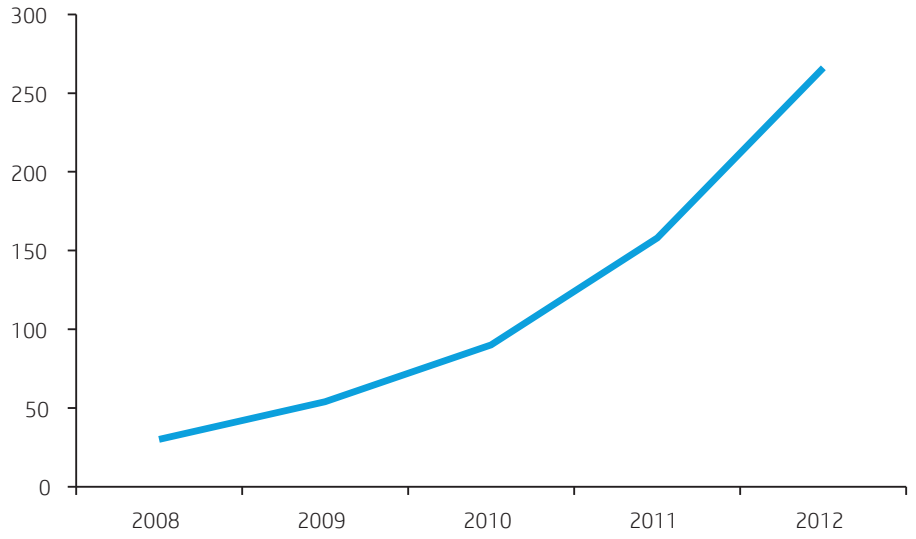



Enterprises can conclude that while application vulnerabilities not connected to the Web still represent the majority of vulnerabilities disclosed, a handful of categories in Web application vulnerabilities still introduce a substantial amount of technical risk to organizations.

## The maturity of a technology does not govern its vulnerability profile

One of the biggest technical paradigm shifts over the past decade has been the widespread adoption of mobile devices capable of running custom applications. Over the last five years alone, OSVDB data show a 787 percent growth in mobile vulnerability disclosures (see Figure 6). In the last year, the number of mobile vulnerabilities disclosed rose 68 percent from 158 in 2011 to 266 in 2012.

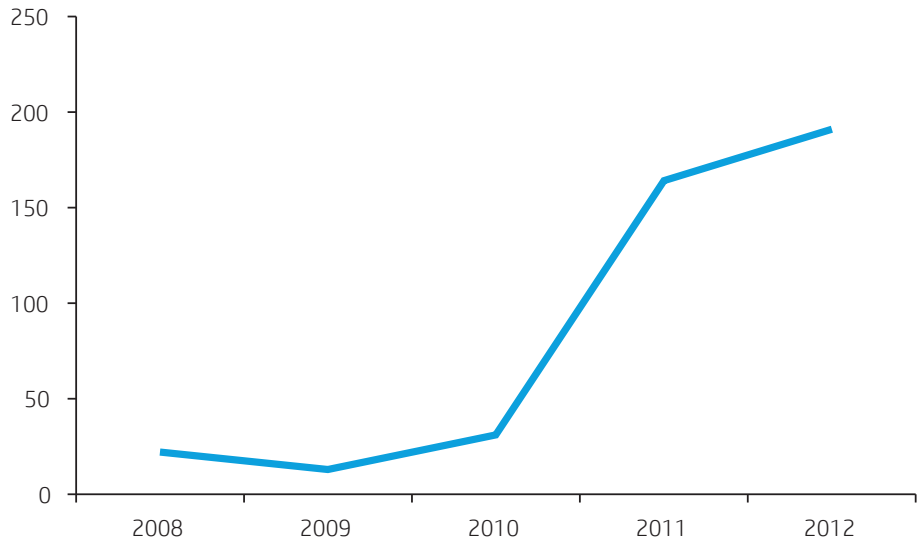
**Figure 6.** Disclosed mobile vulnerabilities measured by OSVDB, 2008–2012




In contrast, Supervisory Control And Data Acquisition (SCADA) systems—which control automated industrial processes such as manufacturing, power generation, mining, and water treatment—rely on considerably more mature technology. These systems, which have historically operated over separate networks with proprietary protocols, have begun to migrate to standard networks and even the Internet to simplify asset management, billing, and operations. As these systems have moved off their separate isolated networks, security problems that were once masked by a restricted attack surface have begun to manifest themselves.

According to OSVDB data, only 76 vulnerabilities were disclosed in SCADA systems from 2008 through 2010. However, after the Stuxnet worm was discovered in an Iranian uranium enrichment plant in 2010, much attention has been focused on the security of SCADA systems. In 2011, there were 164 vulnerabilities disclosed in SCADA systems, and the number rose again to 191 in 2012, representing a 768 percent increase from 2008 numbers (see Figure 7).

**Figure 7.** Disclosed SCADA vulnerabilities measured by OSVDB, 2008–2012




### Zero Day Initiative: a glance at 2012

Due to the ZDI's position as the premiere vulnerability acquisition program, the team's researchers often have the opportunity to analyze some of the most interesting and talked-about vulnerabilities that have occurred over the past year. The following 2012 "buzz-worthy" cases illustrate the intersection between white and black markets.

At the beginning of 2012, a vulnerability in the Microsoft® Remote Desktop (ZDI-12-044, MS12-020, and CVE-2012-0002) received extensive attention:

- The ZDI reported this specific flaw to Microsoft in August 2011.
- There were no known attacks in the wild; however, due to its attractiveness to attackers, Microsoft expected exploitation to be imminent by March 2012.
- This flaw was potentially reachable over the network before authentication was required and existed during error handling while elements were being loaded into an array.

Only a month later, Samba released a much-needed patch based on a flaw found by a ZDI researcher (TPTI-12-04, CVE-2012-1182):

- The ZDI reported this specific flaw to Samba in September 2011.
- There were no known attacks in the wild; however, as this is the most serious vulnerability possible in a program, Samba addressed it quickly.
- This flaw did not require an authenticated connection and resulted in memory corruption that may be exploited by an attacker to gain remote code execution.

The end of 2012 brought a flurry of zero-day activity affecting both Oracle (CVE-2012-0422, CVE-2012-3174) and Microsoft (MS13-008, CVE-2012-4792). The ZDI was at the forefront of the action, having reported a critical Java vulnerability patched with the zero-day activity:

- The ZDI reported this Java flaw to Oracle in December 2012.
- This flaw was exploited in the wild.
- This flaw allowed a malicious applet to execute attacker-supplied code, resulting in remote code execution under the context of the current user.

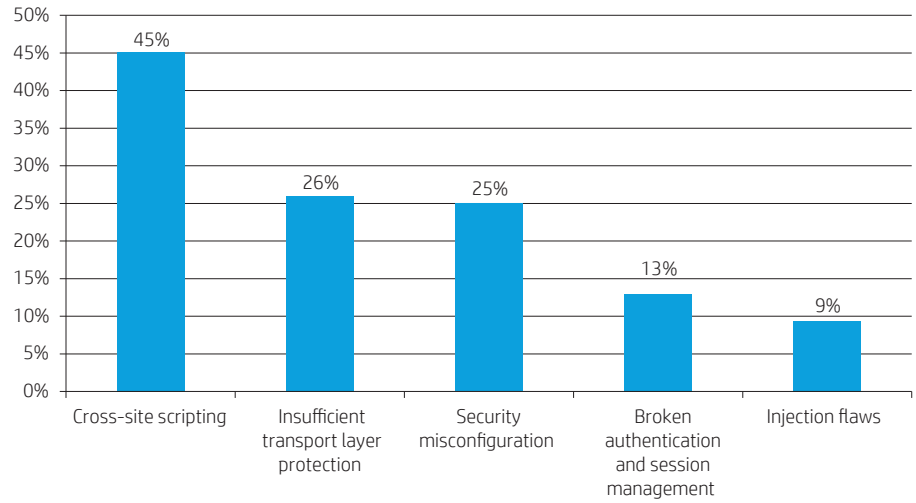
These are only a few cases handled by the ZDI in 2012 that illustrate the intersection between white and black markets in the vulnerability arms race. One thing is certain: our researchers are in the race to find and responsibly disclose vulnerabilities in highly prevalent software technology.

## Web application vulnerabilities

To develop a more complete picture of the current vulnerability landscape, the HP Fortify on Demand team gathered and analyzed the results of thousands of assessments to see how Web application security looks from the inside via code review and penetration testing. A thorough review of both static (examining source code for vulnerabilities without executing it) and dynamic (testing the running software as an attacker would without access to the code itself) analysis results provides a sound basis for investigating the true state of risk in Web applications.

The sample sets used for this analysis comprised over 200 randomly selected applications for dynamic analysis and 800 applications for static analysis. Figure 8 shows the top five vulnerabilities discovered using dynamic analysis in 2012.

**Figure 8.** Top five vulnerabilities discovered with dynamic analysis in 2012 via HP Fortify on Demand




The dynamic results from this sample set as well as other results referenced in this report show that cross-site scripting attacks remain a major threat when attempting to secure Web applications. Considering that the number one vulnerability type purchased last year by ZDI was cross-site scripting, and that repeated testing verifies how widespread cross-site scripting is, organizations have sufficient reason to consider this a primary security concern.

Figure 9 shows how a typical cross-site scripting attack works, as well as how it can be leveraged to steal authentication credentials on critical sites.

**Figure 9.** Breakdown of a reflected cross-site scripting attack

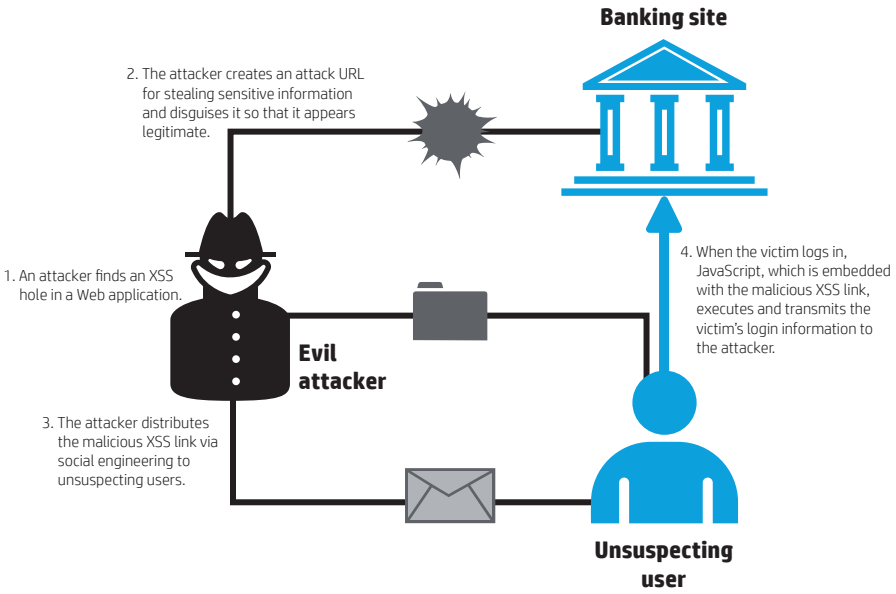
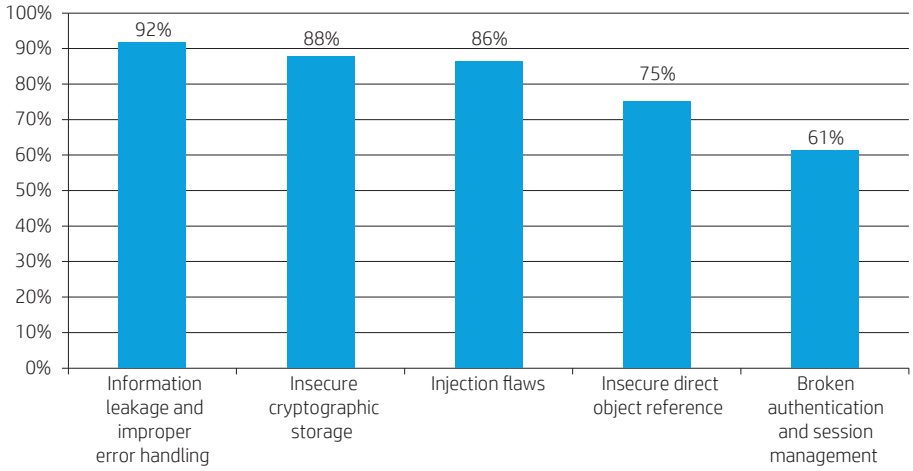



Figure 10 shows the top five vulnerabilities discovered with static analysis in 2012, and includes the relative percentage of applications impacted by each vulnerability category.

**Figure 10.** Top five vulnerabilities discovered with static analysis in 2012 via HP Fortify on Demand




The static results reveal that information leakage, problems with cryptographic storage, and injections flaws were all heavily represented. Not revealed in the top five results is the fact that reflected cross-site scripting (as illustrated in Figure 9) was returned in 51 percent of the applications. While not in the top five, that's still significant when combined with the other finding of this report.

Evidence from both sample sets leaves little doubt that Web applications continue to be plagued by weaknesses deemed critical by various industry standards. The top five vulnerability categories that dominate the findings are often responsible for exposing Web applications to severe risks like information theft, privilege escalation, and so on. Both these data sets lead us to conclude that organizations continue to fail in applying consistent remediation to ubiquitous vulnerabilities such as cross-site scripting and information leakage.

Also worth noting is that the most prolific vulnerability used by attackers in conjunction with other known vulnerabilities is information leakage. While a specific piece of information might not seem important, it might be the one key component that allows an attacker to escalate a technique and conduct a more devastating attack. At the end of the day, a successful attack is prepared through careful information gathering and reconnaissance techniques.

### Web application risk: a case study

**Company profile**
Large (more than 100,000 employees) multinational organization with revenues exceeding $82 billion USD for FY2012

This company has an active security program in place. The findings indicate how much diligence it takes to fully manage application security risk. The following results are from over 1,300 unique dynamic assessments conducted against various sites the company maintains. In the modern application security era, one vulnerability can often be too many.

The results:

- 54.1 percent of the assessments revealed persistent cookies. The European Network and Information Security Agency (ENISA) has referred to these as "bittersweet" cookies. Persistent cookies greatly increase the probability that replay attacks will occur because of the lengthened time cookies remain valid. Information disclosure at "shared" kiosks, etc., can also produce persistent cookies. Because these are used to track behavior, and so on, it's likely that security considerations were somewhat outweighed by the cookies' potential "benefit" to the corporation.
- Just under half (48.32 percent) of the sites were vulnerable to some form of cross-site scripting.
- Almost one-fifth (19.57 percent) of the sites contained a "mixed-scheme" unencrypted login form where information from an HTTP page was posted to an HTTPS page or vice versa.
- 12.77 percent of the assessments were vulnerable to some form of SQL injection. What was really intriguing is that 10.97 percent of the assessments confirmed blind SQL injection vulnerabilities. Considering how particularly nasty blind SQL injection is and that even one such vulnerability can often be used to compromise a system, this percentage is very dangerous.
- Another fifth (19.7 percent) of sites were vulnerable to logins sent over an unencrypted connection. This means that the form did not utilize SSL.
- 5.26 percent of sites were susceptible to local file inclusion/read vulnerabilities. If sensitive enough, the contents of the file could be used to take control of the system.

## Devastating attacks

In addition to statistics, security intelligence also requires observation and insight. To that end, the following section summarizes a list of devastating attacks that HP Fortify on Demand penetration testers discovered in the wild during 2012. Every account was discovered with permission during approved engagements against production sites ranging in technical acuity from trivial to challenging. The common thread among them was that all of the attacks had extremely serious consequences if discovered by a malicious attacker. The customer names have not been included for obvious reasons, but the industries themselves are listed in order to illustrate the danger and relevance of the attacks. While these examples are anecdotal in nature, they do take advantage of the experience and opinions of our penetration testers in a way that simple statistics do not. As always, security is more than just products. It's a process.

### Injection and improper input validation hacks

*Industries: petrochemical, food processing, energy, and software*

**Unsafe file uploads and security misconfiguration:** A thick client program allowed any user to upload malware via its upload function to a Web server that had no anti-malware protection. The program had about 30,000 users worldwide, so this was particularly dangerous. Obviously, it is imperative that file upload functionality exercise extreme prejudice with regard to acceptable file types. Figure 11 shows the consequences of this scenario.

**Figure 11.** Allowing unvalidated user uploads can have severe consequences


**Figure 12.** Blind SQL Injection

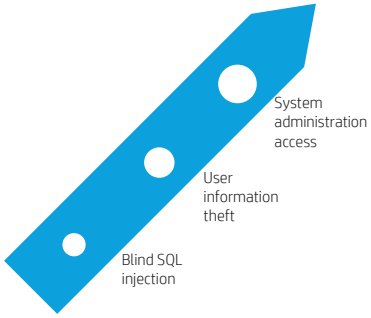



**Figure 13.** Cleartext SQL

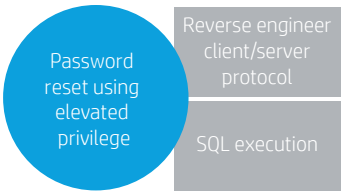
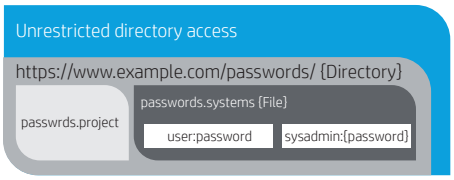



**Figure 14.** Failure to restrict access




**Blind SQL injection:** Guided search functionality (a series of checkboxes designed to help consumers narrow down their search criteria) without input validation resulted in the exposure of 35 databases and database system user IDs and password hashes, including the system administrator account. Blind SQL injection can be difficult to find because it rarely reveals error messages and manifests itself as anomalous application behavior. Parameterized queries, also known as prepared statements, effectively prevent SQL injection attacks when properly implemented.[5] Figure 12 shows this sequence.

**Cleartext SQL:** During testing of a thick client application, it was discovered that the client was executing SQL statements directly to the back-end SQL server with administrator-level permissions over HTTP. By reverse engineering the protocol and through manipulation, user privilege escalation and password modification was achieved (see Figure 13).

**Local file inclusion:** Directory traversal and local file inclusion techniques were used to view the contents of the Web server's backup security accounts manager (SAM) file, which allowed the passwords to be cracked. Within 10 minutes, local administrator access to the system was gained, allowing for complete compromise. Input validation routines, both inherent within the application and at the Web server configuration level, would have eliminated this vector.

**Security misconfiguration hacks**

*Industries: petrochemical and international banking*

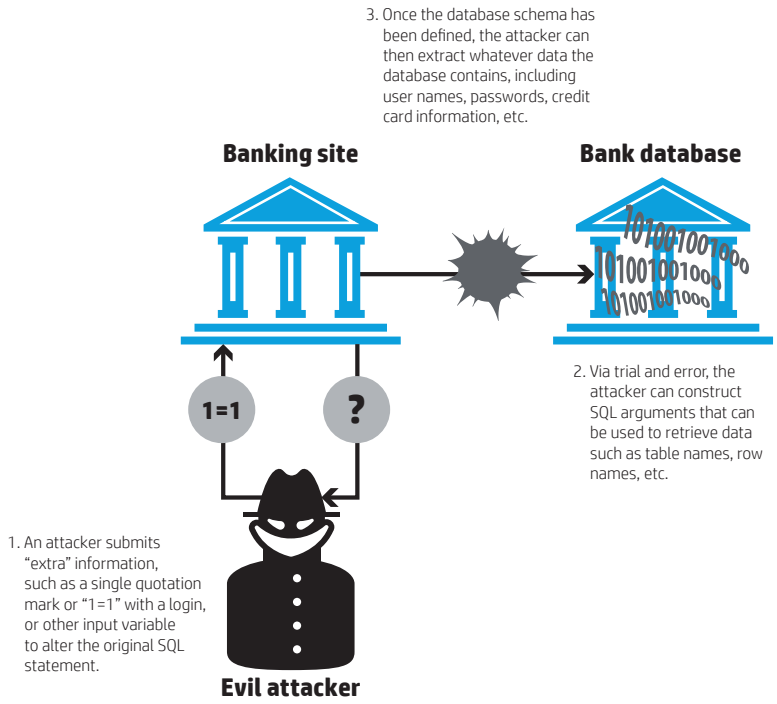
**Failure to restrict access to sensitive directories:** In this case, the discovered directory was "https://www.example.com/passwords/". Understandably, there shouldn't be a "passwords" folder, at least not publicly. The folder was accessible via Web browsers with no authentication and included a directory listing of text files with names like "passwords.project", including "passwords.systems". Clicking on a file opened a text document with a list of users and passwords in the format: user:password. One of the lists even contained "sysadmin:{password} admin:{password} etc..." (see Figure 14). This illustrates the need for and the importance of post-automated scan validation, as this seemingly low-risk vulnerability could easily be disregarded by the customer. In addition, the problem could have been avoided if access to the directory had been properly restricted.

**WebDAV enabled allowing remote write:** WebDAV was enabled on a particular Web server in a way that allowed remote application users to interact with the host and write files to arbitrary directories. By leveraging this vulnerability, a custom backdoor was uploaded and subsequently executed by browsing to the URL path of the newly transferred file. Once executed, it allowed full control of the Web server. If scoped for further testing, the tester would have been able to continue the attack against other hosts on the internal corporate network—a process known as *pivoting*. Removing WebDAV access and extraneous HTTP methods could have prevented this attack.

**SQL injection and weak input validation controls:** A SQL filter was filtering everything but the "OR" operator. This basically means that by including "OR" in a SQL statement that any command after it would be executed on the system. Parameterizing queries will prevent SQL injection attacks. Figure 15 breaks down the sequence of a SQL injection attack.

[5] https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

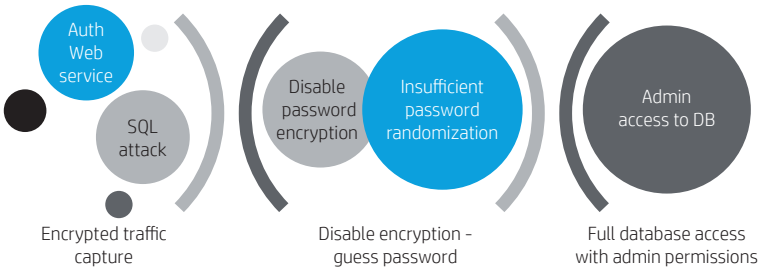
**Figure 15.** Breakdown of a SQL injection attack




**SAP misconfiguration:** The entire credit card database was accessed and dumped to a file. The SAP implementation had poorly configured controls, allowing customer service representatives to run sensitive transactions, including the HR data browser (SE16). This capability was used to browse and load the entire contents of the customer credit card data table. Given access in this manner, the entire data set could easily be exported using lowest-privilege SAP accounts.

**Authentication, session, logic, and miscellaneous hacks**
*Industries: airline, international banking, and energy*
**Enumeration of airline tickets through mobile QR code Web services:** Testers were able to reverse engineer part of the Web service function to create ticket numbers. Fake ticket QR codes for airline flights were then generated. Hypothetically speaking, exploitation may have enabled attackers to fly for free by allowing them to check in via their mobile devices in the absence of effective compensating controls. Using an industry-standard randomization function for ticket numbers could also have prevented this problem.

**Figure 16.** Easily reversible dynamic password generation



**Easily reversible dynamic password generation:** The thick client application being tested communicated with a Web service that first authenticated the user attempting to log in. Next, the application requested the password to the back-end database server. The password was originally encrypted during transmission, but by combining another SQL attack, a flag in the database was able to be flipped, and thereby turn off this encryption. It appeared the system implemented dynamic passwords that changed frequently; however, after gathering multiple passwords, a pattern emerged. The password was time based and therefore easily guessable. Direct access to the database with administrative permissions was confirmed, allowing for complete compromise of the system (see Figure 16).

**Web service allowed direct SQL queries:** An application allowed connections to its back-end database via a Web-based interpreter that was accessible to the Internet without authentication. With limited knowledge of SQL, an attacker could easily retrieve all records in the client database, including full user account details: user names, passwords, and email addresses, among other personally identifiable information (PII).

By sharing these accounts of real-world penetration testing findings, our goal is to draw attention to the limitless creativity and endless determination of malicious attackers. Penetration testers emulate these techniques in an effort to effectively test each security control to help ensure that sensitive data and application capabilities are protected. As demonstrated in these accounts, automated security testing alone is an incomplete measure of an environment's overall security posture and must be supplemented with manual analysis.

In the next section, HP's software security researchers aim to highlight a crucial vulnerability often used in combination with many of the attacks and techniques mentioned earlier whose effective mitigation has yet to be holistically adopted.

## The X-Frame-Options header—a failure to launch

In order to illustrate the constantly changing nature of Web application security, researchers from the HP Software Security Research group examined an established vulnerability formally referred to as cross-frame scripting (XFS). XFS is an important tool for any attacker trying to craft a phishing or social engineering attack and can be exploited to load a vulnerable website inside an HTML iFrame tag on an attacker-controlled malicious page. This enables the attacker to capture events, such as keystrokes, invoked by any user on the victim website.

XFS vulnerabilities also pave the way for clickjacking[6] attacks, which deceive users into clicking certain elements on the victim website loaded inside an invisible iFrame tag. Often, this results in unintended and possibly even privileged actions. For years, researchers have warned against the ineffectiveness of script-based frame-busting protections in mitigating the threat of XFS. Developers and supporting server administrators, however, continue to rely solely on the JavaScript-based mechanism (Figure 17) to detect XFS-based exploits.

Various mitigations proposed against XFS have repeatedly proven to be incomplete or ineffective. The most popular of these mitigations is the use of JavaScript frame-busting logic, or essentially a client-side script that resembles the one shown in Figure 17.

**Figure 17.** Typical JavaScript frame-busting logic

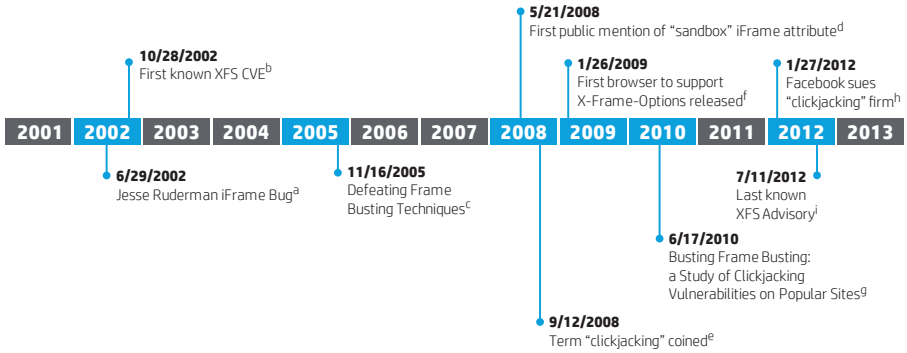
```
<script>
      if (top!=self) top.location.href = self.location.href;
</script>
```

[6] cvedetails.com/cve/CVE-2002-1217/

Aside from many variations to this technique that can easily be found online, counter-mitigations have also been created that defeat these variants as well, essentially amounting to the classic game of whack-a-mole.

As shown in Figure 18, the first documented XFS vulnerability was discovered over 10 years ago.[7] Since then, clickjacking has become a household vulnerability, yet less than one percent of our sample set included the best known mitigation, the X-Frame-Options header.

**Figure 18.** A brief history of significant events stemming from the discovery of cross-frame scripting (XFS)




[a] https://bugzilla.mozilla.org/show_bug.cgi?id=154957
[b] http://www.cvedetails.com/cve/CVE-2002-1217/
[c] http://crypto.stanford.edu/framebust/
[d] http://html5.org/tools/web-apps-tracker?from=1642&to=1643
[e] http://www.sectheory.com/clickjacking.htm
[f] http://blogs.msdn.com/b/ie/archive/2009/01/26/internet-explorer-8-release-candidate-now-available.aspx
[g] http://cdn.ly.tl/publications/busting-frame-busting-a-study-of-clickjacking-vulnerabilities-on-popular-sites.pdf
[h] http://www.guardian.co.uk/technology/2012/jan/27/facebook-sues-clickjacking-firm-adscend
[i] http://osvdb.org/show/osvdb/83762

The history of XFS begins as a rather benign report of behavior observed by Jesse Ruderman in 2002[8] and matured into an enabler for what's commonly referred to as *clickjacking* today. A key component of our identified security regression was the discussion of adding the sandbox attribute to the draft HTML5 specification for the iFrame tag in mid-2008.

Researchers have demonstrated numerous counter-mitigations made possible by either browser bugs or JavaScript tricks. The latest addition to this list, as shown in Figure 19, is the use of a security feature introduced in the HTML5 specification in the form of the iFrame sandbox attribute that attackers can exploit to disable JavaScript frame-busting protections.

**Figure 19.** iFrame sandbox attribute

```
<iframe sandbox="allow-scripts" src="http://example.com/"></iframe>
```

The use of the sandbox attribute is to grant script execution permission (allow-scripts) to the framed victim. As a side effect, any frame-busting logic used by example.com is effectively disabled by denying it the allow-top-navigation permission in its sandbox specification.

By using the sandbox attribute, this technique effectively allows an attacker to bypass any frame-busting attempts by developers to protect against third-party site framing and clickjacking. Browser vendors have introduced and adopted a much stronger, policy-based mitigation technique using the X-Frame-Options header. Developers can use this header to instruct the browser about appropriate actions to perform if their site is included inside an iFrame. Unfortunately, the adoption rate of this feature among Web developers is very low, which allows attackers a vast playground of potentially exploitable sites as they discover new techniques to circumvent frame-busting scripts. This makes the adoption and implementation of the X-Frame-Options header that much more critical, as it's the de facto solution to this issue, and remains effective even when faced with the new HTML5 sandbox attribute. Herein lies the impetus for this research project and the formation of our initial question, "How many domains include X-Frame-Options headers, and how many of them do so correctly?"

[7] cvedetails.com/cve/CVE-2002-1217/
[8] https://bugzilla.mozilla.org/show_bug.cgi?id=154957

**Figure 20.** Cross-frame scripting mitigations

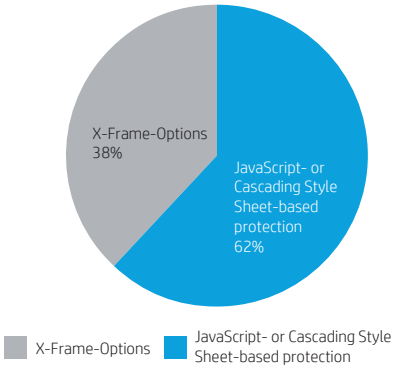
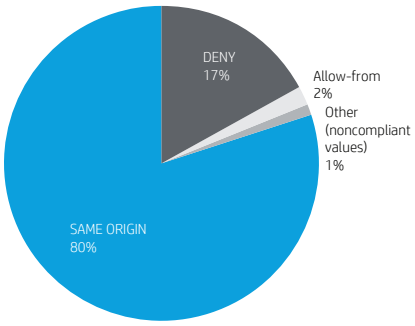



**Figure 21.** Breakdown of X-Frame-Options values supplied




## Sample set collection

As part of this study, the HP Fortify Software Security Research Group conducted research to gauge how widespread weak XFS mitigation practices continue to be adopted and relied upon. Our findings also highlight the reluctance and slow adoption rate of more secure practices recommended by browser and industry security experts.

Alexa[9] was used to populate a collection of 100,000 of the most popular websites in order to produce a list distributed over many different industries. After the initial list of 100,000 URLs was created, the top-level requests were assembled by concatenating "http://" to the resulting URL, for example:

"http://" + example.com = http://example.com

Next, the newly formed URL was requested, and the resulting response was passively examined for the following values:

- The presence of the X-Frame-Options header and its respective values.
- The presence of a password field in the response from the top-level request. It is assumed that a password field indicates a level of sensitivity that must be protected by the application.
- If the presence of the X-Frame-Options header was not observed, additional testing was performed on the target URL in order to discern whether any attempt to thwart XFS attacks had been implemented. Use of JavaScript- or Cascading Style Sheet–based mitigation was recorded for further analysis.

## Findings

**90 percent** of the samples analyzed made no attempt to protect themselves against XFS. Of the domains with XFS protection in place, 62 percent are still relying on the weak script-based mitigation. Of the 100,000 domains tested, 19,848 had password fields present on the top-level request, with only 101 of those specifying an X-Frame-Options header. **That's 0.1 percent of the sample that's protected.** Furthermore, over 99 percent of the sample set neglected to specify an X-Frame-Options header, while 19 percent of the sample set had a reason to include the X-Frame-Options header but didn't.

The findings indicate that developers are either still lacking sufficient awareness of the threat posed by XFS vulnerabilities or are unwilling to invest the effort into protecting their visitors from being victimized. While 2,307 samples employed protections against XFS, only 38 percent opted to use the recommended X-Frame-Options header. **1,432** samples were found to be still using either JavaScript-based or Cascading Style Sheet–based mitigation techniques, which have proven insufficient (see Figure 20).

The 875 domains relying on the X-Frame-Options header displayed the following distribution of values:

1. 702 of 875 domains specified with an X-Frame-Options header were "SAMEORIGIN"
2. 151 of 875 domains specified with an X-Frame-Options header were "DENY"
3. 13 of 875 were specified using "Allow-from"
4. 8 of 875 supplied values inconsistent with the IETF draft[10]

None of the 13 domains using the Allow-from attribute specified the wildcard (*) value to mitigate the risk of open access policy. Figure 21 shows these values as percentages.

[9] https://bugzilla.mozilla.org/show_bug.cgi?id=154957

[10] http://tools.ietf.org/html/draft-ietf-websec-x-frame-options-01

**Cross-frame scripting: in conclusion**

We realize that not every resource needs to be protected against XFS. However, the fact remains that there are valid situations that demand adequate protection. One such metric applied to identify these cases during the study was the presence of a password input form field. Authentication pages face the highest risk from a clickjacking attack, and by focusing on pages with password fields, the analysis attempts to correlate the sensitivity of a resource and the effort invested in protecting it. One of the more compelling statistics from the analysis is that only 103 out of 100,000 domains adequately protected pages with sensitive password fields present. That's a staggering 0.10 percent and suggests that the power and effectiveness of the X-Frame-Options header has not been adequately evangelized, adopted by server administrators, or requested by developers.

It's worth mentioning that while a little over 80 percent of the responses didn't include a password field, that doesn't mean the content is not worth protecting. Simply stated, the sample didn't include a password field on the top-level request, so it's very likely that many more sites are at risk when delving deeper into the site tree. The more important metric to pay attention to is the percentage of sites without an X-Frame-Options headers present—an overwhelming 99 percent.

After closely analyzing specialized research covering XFS analysis, the next section will refocus attention to arguably the most popular information security topic of 2012, mobile application security.

## Mobile application security

It's obvious from standing in line for more than 10 seconds anywhere that the use of mobile devices has exploded. In fact, 2012 for the first time saw more smartphones sold than laptops and desktops combined.[11] That rise in usage has also come with a commensurate rise in risk, especially as businesses try to capitalize on the advantages mobility provides. As we saw earlier, the OSVDB reported a 68 percent increase in submission of mobile vulnerabilities since 2011—a 787 percent increase over the last five years. During our testing, one thing has repeatedly rung true. If you have data, attackers will come for it.

To gauge the current state of mobile application security, The HP Fortify on Demand team gathered results from over 70 mobile applications for security vulnerabilities. This sample set covered more than 50 unique organizations and multiple industries ranging from small to global, so it should serve as a fairly accurate representation of an average mobile application. The results show that the same security vulnerabilities that affect regular applications also affect mobile ones. The results also showed that one problem stood out above the rest. Given the realities of the modern IT landscape, information leakage is an especially widespread and pernicious problem. The increasing demand for information from the mobile workforce—combined with increasingly pervasive cloud services and a wave of unmanaged consumer-grade devices on the corporate network—presents a noteworthy challenge for many organizations.

### Sensitive data leakage over insecure channels

Data leakage has been a long-standing issue among Web applications. While data leakage can seem low key, it is often a seemingly innocuous piece of information that lets an attacker escalate his or her attack methodology to conduct a more dangerous one. The same is true in mobile applications. Over three-fourths of the mobile applications (77 percent) in our survey were susceptible to information leakage vulnerabilities. We discovered that a user's personal data was often sent over unencrypted network protocols such as HTTP. Much of this information was simple, such as names, addresses, and phone numbers. However, across our sample set this data also included the current location of the user, as well as the specific device identifier (aka the UDID).

[11] voices.washingtonpost.com/posttech/2010/11/smartphone_sales_to_pass_compu.html

The device identifier is very important, in that it can be leveraged for incredibly targeted attacks against specific users. If the geolocation, unique device identifier, and personal details of the device owner could all be intercepted via a vulnerable application, the real-world implications are staggering. An attacker could locate a "target" in the real world, and then what might happen is open-ended. It presents a frightening scenario most people don't imagine. Of course, simple run-of-the-mill application exploitation could also take place. Imagine this scenario: if the application has been sending the UDID, full name, address, and so on, to a vulnerable Web service, and that Web service is susceptible to SQL injection, then it's easily conceivable that every bit of data on that mobile device could be accessed. It's amazing how far information leakage can take an attacker given the right set of circumstances, and none of them is out of the realm of the probable, let alone possible.

Data transmitted over insecure channels was not limited to personal data—application data was also not secure. We discovered login information, user credentials, session IDs, tokens, and sensitive company data all being sent over unencrypted network protocols like HTTP. Imagine the consequences for a vulnerable banking application. If credentials, session identifiers, identifiable personal information, or other sensitive data are being transmitted to a back-end server, the transmission should be secure. Otherwise, data could be intercepted by an attacker using common network packet-capturing tools or apps (e.g., DroidSheep).

The research showed that 37.5 percent of the of the applications were susceptible to some form of authorization vulnerability, including cleartext passwords, hardcoded passwords, and passwords included as part of the response. That's a much higher percentage than we saw in "traditional" applications and provides another indication that mobile developers need to do a better job taking care of their data.

Other vulnerabilities that registered in important numbers included stack smashing. More than half of the vulnerabilities (55.45) did not include proper protection against stack smashing attacks. While this oversight will not lead to code execution, it can cause the vulnerable application to crash indiscriminately.

In addition, 13.5 percent of the applications were vulnerable to XSS. This was actually a surprise, as every other set of applications we've tested, both mobile and traditional, showed higher numbers of XSS vulnerabilities. When the numbers were reviewed more closely, they revealed that the vulnerable applications consisted of both financial and database management applications. In other words, the low percentage did not diminish the potential impact.

## Unauthorized access affects almost half of mobile apps

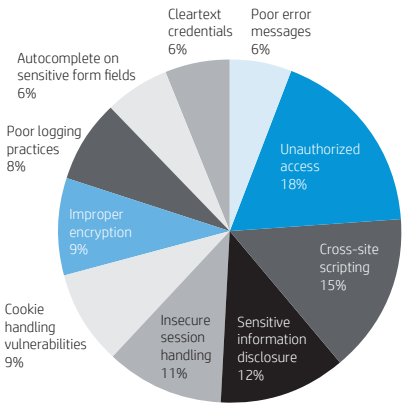
We sought out more data by seeing what results our partner Security Compass gained from testing an additional set of applications. In many ways, they confirm our earlier numbers.

**48 percent** of the applications were susceptible to unauthorized access vulnerabilities. These validate the authentication vulnerabilities (37.5 percent) that we encountered in our earlier sample. The numbers show that mobile developers need to concentrate on preventing unauthorized access to mobile applications as much as making them easy for legitimate users to access.

**37 percent** of the applications contained sensitive information disclosure issues, 26 percent utilized poor logging practices, and 19 percent contained poor error messages that revealed information that could be used by potential attackers. This corroborates our earlier set of data as we saw the significance of information leakage. When coding mobile applications, developers are not considering the security implications of how they store, transmit, and access data.

**33 percent** of the applications were vulnerable to XSS attacks. Repeated testing shows that XSS poses just as much of a threat to mobile applications as it does to their "grounded" counterparts. This set of results is in line with what we see when testing traditional applications.

**Figure 22.** Top 10 vulnerability percentages by prevalence




**26 percent** of the applications employed improper encryption. Encryption on corporate PCs is now standard protocol for most Fortune 500 companies. Ten years ago the news was rife with stories of data stolen from lost PCs. This has definitely been curtailed in part because of legislative requirements, and in part because corporations have learned their lessons the hard way. However, these same standards are not yet being applied to mobile devices, and in the age of bring your own device (BYOD), that's dangerous.

## Top 10 mobile vulnerabilities

HP partner Security Compass also tracked mobile vulnerabilities and flagged them by type (see Figure 22). In other words, What are average applications most vulnerable to by vulnerability prevalence? What types of attacks by incident are mobile applications most vulnerable to?

Of the top 10 mobile vulnerabilities, XSS was flagged at the second highest rate, and was 15 percent of all the vulnerabilities discovered in that sample set. The numbers also confirm that when flagged by occurrence, information leakage and unauthorized access/authentication vulnerabilities show mobile developers how they can better secure their applications by focusing their efforts on those areas.

## Mobile research—near-field communication (NFC) for mobile payment applications

As we've seen the rise of mobile application vulnerabilities, it's also important to look ahead and see what potential new vulnerabilities are on the horizon. New technologies always introduce possible security vulnerabilities. The mobile platform is no different. One of these new technologies is near-field communication (NFC). NFC is a method of contactless communication of data between devices in close proximity. It is a technology that has already been adopted in Europe and Asia and has recently been gaining traction in North America. The NFC Forum (comprised of members from Sony, Nokia, and Philips) enforce strict standards for manufacturers designing NFC-compatible devices. This provides a secure architecture and compatible framework for application vendors to harness this technology for mobile payment capabilities and data sharing (e.g., peer-to-peer money exchange).

Currently, there are several NFC-compatible smartphones such as the Google™ Nexus 5, the Samsung Galaxy S II, and the BlackBerry Bold 9900 and 9930. Several companies are currently leveraging this technology:

- Google Wallet—a secure container for credit card information that facilitates NFC transactions
- MasterCard PayPass—a contactless payment service (currently supported on Google Wallet)
- Visa payWave—a contactless payment service
- PayPal—a "bump" method to transfer money or make payments between users
- Apple iPhone—expected to have NFC support in a future release (not confirmed)[12]

### Security concerns

There are several attack scenarios to consider when sensitive information such as credit card or account number data is being transmitted through an NFC channel on a mobile device. The following are examples of potential real-world cases discussed in the security arena more recently:

*Case 1*
NFC technology is used as a part of consumer payment solutions in two separate implementations:

- NFC chip in consumer credit cards such as MasterCard PayPass and Visa payWave
- NFC used by mobile wallet solutions

[12] There are currently vendors such as DeviceFidelity, which provide third-party components to mimic NFC support for iPhone devices.

The challenge with the first implementation is twofold: one is that for most credit card providers, this is a mandatory installation and second is that by design the NFC chip in credit cards is always "on." For example, if a user's credit card is in the field of an active NFC reader, such as the one in a point-of-sale (POS) terminal, the credit card automatically transmits the user's credit card number to the receiving NFC reader. Now, consider the scenario that involves most modern mobile devices that have built-in NFC chips. In the Android world, there are applications that are designed to activate the mobile device's NFC chip to emulate the behavior of a POS terminal's NFC reader. By using such a mobile device and application, an attacker can potentially activate the NFC chip in his or her Android device and bump into people in a crowded setting, attempting to scan their credit cards to collect their permanent account numbers (PANs).

The NFC mobile wallet solutions tend to be vulnerable to the same bump-and-steal attack if the solution in question does not "turn off" or disengage the NFC chip after the user has completed a transaction. Fortunately, this vulnerability can be easily remediated by mobile wallet solution developers if they disengage the NFC chip after use.

*Case 2*
There are several attack scenarios to consider when sensitive information such as credit card or account number data is being transmitted through an NFC channel:

- Eavesdropping: attempting to intercept the NFC transmission data communication (e.g., NFC proxy)
- Data manipulation: attempting to manipulate the NFC transmission data communication (e.g., to determine erroneous outcomes)
- Interception attacks: attempting to take advantage of active-passive modes of the device to send and receive NFC transmission data communication
- Theft: attempting to gain unauthorized access to the mobile payment application (as if the device were stolen or lost) and reviewing the file storage on the device for sensitive information

*Case 3*
At the very core of the NFC functionality is the component called the *Secure Element*. This has two implementation types: (1) those embedded in the device and (2) those loaded on a SIM module. In cases where the Secure Element is loaded onto a SIM module, there is a possibility for the SIM module to be removed and swapped from an unsuspecting user. This results in possible attack scenarios such as:

- SIM swap to another (same type) device to attempt access to the contents of the Secure Element. In some cases, using another device of a different type may offer up additional access to information of the Secure Element if implemented insecurely. For example, a risk of bypassing the payment application- or phone-level password protection by swapping SIMs into a matching phone with the payment application and no password.
- Identify data containing sensitive information in the Secure Element located in the SIM module after a restore operation to another device. This may also include possible attacks to clone or copy the SIM using an external hardware device.

It should be noted that the difficulty in gaining access to the Secure Element provides some level of assurance about the security posture of the Secure Element on the target device.

*Case 4*
In some cases, implementing the VMPA on the Secure Element may not have taken all security elements into consideration. For example, insufficient signing and access control of the VMPA applet such that any application can initialize and make requests to it. This may lead to the ability of an unauthorized application to invoke the VMPA applet through the APDU command (JSR 177). A well-crafted request may potentially allow the NFC radio to be turned on and transmit the credit card information.

Although meaningful information and analysis can be obtained from a forensic device (if access to such hardware is available), at this time there is no known tool that allows access to the Secure Element information. Secure Element, as the name implies, provides specific security mechanisms (e.g., access control, encryption, and segregation) to prevent extraction of possible sensitive information. Therefore, explicit forensic access of the Secure Element is not currently possible even with a forensic tool from market leaders.

### Recommendations

There are certain actions that organizations can take to mitigate the risk from mobile application security vulnerabilities. First, applications need to be manually audited and assessed before the products are launched to determine if any input injection vulnerabilities or information leakage vulnerabilities are present. The code should be analyzed via static analysis when being developed to find code-based vulnerabilities. As with any application, it's much less expensive to address security vulnerabilities during development than once it has been released.

Secure data transmission standards should be included as part of any application requirements, especially if those applications are being developed by third-party developers. The same is true for secure data storage and application logging. Reasonable inter-application communication exposure and permissions in application requirements should be stringently defined. These concerns should all be addressed in the requirements phase and tested during development. When performing security testing and analysis on mobile applications, the server-side Web services and APIs that the mobile clients talk to should be taken in context and analyzed for vulnerabilities. High-risk vulnerabilities may be missed if the two are tested out of context with each other.

## Conclusions

For us, this report is a way to provide organizations with the security intelligence they need to better understand how to deploy their limited resources so that they can best minimize their security risk. As such, there are a number of lessons and trends from 2012 that should not be ignored either in the coming year or beyond.

### Vulnerability weaponization

We will continue to see attackers weaponize vulnerabilities to carry out their malicious agendas. Those who build exploit kits will continue to focus on vulnerabilities in prevalent software, often targeting browsers and browser plug-ins, such as Oracle Java and Adobe® Flash. Similarly, we will continue to see a steady rise in the number of publicly disclosed attacks targeting specific technologies and organizations. Crime organizations, nation states, and hacktivists will continue to use cyber attacks as a method of leveling the playing field against wealthy or powerful targets, though the true motivations behind attacks will often remain difficult to determine.

### Mobile vulnerabilities

The growth in adoption of mobile technology and its intersection with use in the enterprise will continue to introduce considerable risk. As many have noted, the growth of malware on both the Android and Apple marketplaces continues to climb. The problem is only exacerbated by the fact that enterprises don't have the same types of control over these devices as they do PCs. As BYOD becomes the enterprise norm, and the adoption of mobile devices continues to grow, expect the commensurate rise in mobile application vulnerabilities to continue unabated for the foreseeable future.

## Mature technologies, continued risk

It's not only new technologies that introduce vulnerabilities. Attackers continue to leverage existing and seemingly mature technologies to introduce enterprise risk. From the rise in SCADA vulnerabilities to the recent Department of Homeland Security announcement recommending that the Oracle Java SE platform be universally disabled in browsers, new methods of attack are constantly being discovered in old technology. When coupled with a lack of best practices concerning existing vulnerabilities (such as the lack of cross-frame scripting prevention), it's easy to see that securing the enterprise becomes that much harder when even mature technologies remain stubbornly vulnerable.

## Web applications remain vulnerable

Many companies and individuals assume that "their websites" are not "interesting" to attackers. Nothing could be further from the truth. In fact, the lack of secure programming and IT security best practices only serve as an enabler for the proliferation of malware. In addition, the lack of proper input sanitization in Web applications, as well as the information "leaked" by them, shows that developers still have a long way to go to secure their applications properly.

Many of the documented attacks in 2012 (and earlier), whether by hacktivists or those seeking to enable crimeware, have leveraged long-standing vulnerabilities such as SQL injection. The high disclosure rate of XSS vulnerabilities coupled with its frequent appearance in testing gives us no reason to expect it to drop in popularity anytime soon. In the future, we expect more Injection type vulnerabilities, such as PHP injection, to continue to gain in popularity as the payoff of successful exploitation can be high.

**Learn more at**
**hp.com/go/SIRM**

## Contributors

The *HP 2012 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, and HP Fortify on Demand. We would like to sincerely thank the Open Source Vulnerability Database (OSVDB) for allowing print rights to its data in this report. Special thanks also go to Sahba Kazerooni, Takeaki Chijiiwa, Subramanian Ramanathan, and Jevonne Peters of HP partner Security Compass for their contribution of content, research, and data.

| Contributor | Title |
|---|---|
| Jason Haddix | Director of Penetration Testing, HP Fortify on Demand |
| Brian Hein | Security Researcher, HP Security Research |
| Patrick Hill | Senior Product Line Manager, HP DVLabs |
| Adam Hils | Senior Product Line Manager, HP TippingPoint |
| Präjaktä Jagdälé | Software Security Researcher, HP Security Research |
| Jason Lancaster | Security Researcher, HP Security Research |
| Sasi Siddharth Muthurajan | Software Security Researcher, HP Security Research |
| Mark Painter | Product Marketing Manager, HP Fortify |
| John Pirc | Director, Security Intelligence, HP Security Research |
| Joe Sechman | Manager, Software Security Research, HP Security Research |
| Ryan Strecker | Product Marketing Manager, HP TippingPoint |
| Jewel Timpe | Manager, Malware Research, HP Security Research |



4AA4-5495ENW, March 2013, Rev. 1