Seven Years of Software Vulnerabilities:

The Ebb and Flow

Hossein Homaei and Hamid Reza Shahriari | Amirkabir University of Technology

A seven-year study charted the most common software vulnerabilities. It revealed that by focusing on just seven categories, security professionals could prevent 60 percent of all vulnerabilities.

Numerous software vulnerabilities are reported every year, some common and others rare. A frequently reported vulnerability in any given year might subside in subsequent years because of prevention and detection methods used by developers. When one vulnerability is addressed and nullified, attackers try to find new vulnerabilities and exploit them to penetrate software. Security professionals then develop more countermeasures to defend systems. In other words, the ebb and flow of vulnerabilities influence both the attackers' behaviors and defenders' concerns.

We studied the fluctuation in software vulnerabilities over a seven-year period, 2008 to 2014, to ascertain relevant vulnerability trends and their severity. Focusing on ascending trends would help security professionals find prevention methods early to warn their customers about more common or more severe vulnerabilities as well as train developers in existing restriction methods. Programmers should also concern themselves with code sanitization and vulnerability prevention to proactively counteract the growing pains associated with software vulnerabilities. Moreover, researchers can develop new prevention and detection methods against more common vulnerabilities to improve cybersecurity.

Tracing Vulnerabilities

Vulnerability repositories and databases can be traced to study trends and find severe vulnerabilities. The three most popular repositories are the Open Source Vulnerability Database (OSVDB; blog.osvdb.org), the Exploit database (www.exploit-db.com), and the National Vulnerability Database (NVD; nvd.nist.gov). OSVDB covers only web application vulnerabilities. The Exploit database archives exploits and vulnerable software; as such, it's not a suitable source for the discovery of vulnerabilities. Therefore, we used NVD as our main source of raw data.

According to its website, "NVD is the US government repository of standards based vulnerability management data represented using the Security Content Automation Protocol." NVD, which contains common vulnerability exposures (CVE) vulnerabilities, US Computer Emergency Readiness Team (US-CERT) alerts, US-CERT vulnerability notes, and Open Vulnerability and Assessment Language (OVAL) queries, is one of the most complete repositories of reported vulnerabilities. Therefore, we used its statistics to reveal trends and uncover each vulnerability's importance. We examined statistics beginning in 2008 because NVD didn't accurately classify vulnerabilities before 2008. For example, NVD reported 6,514

 Copublished by the IEEE Computer and Reliability Societies 1540-7993/17/$33.00 © 2017 IEEE

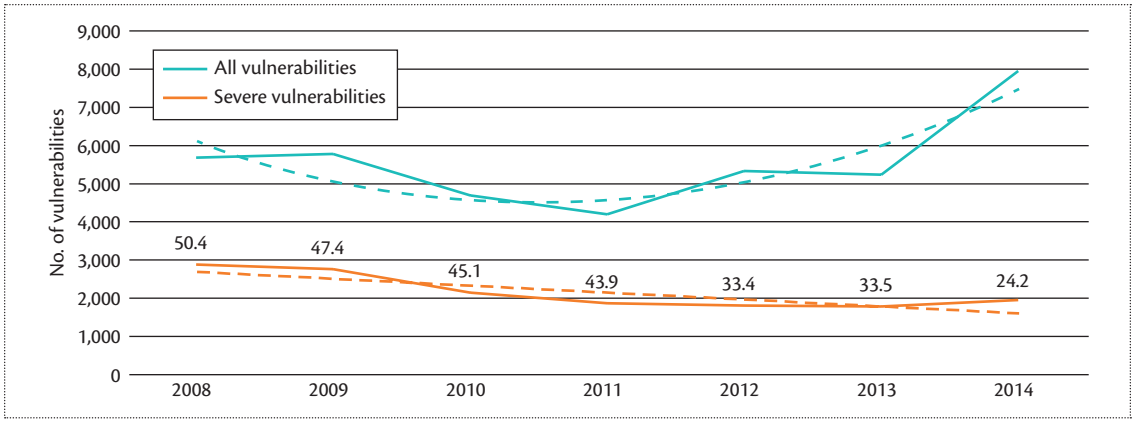

Figure 1. Overall vulnerability trends for the seven-year study. The blue and orange lines represent all vulnerabilities and severe vulnerabilities, respectively, registered on the National Vulnerability Database. Dashed lines represent trends extracted from solid lines.

vulnerabilities in 2007, but only 2,779 of them have been classified.

NVD uses a subset of Common Weakness Enumeration (CWE; cwe.mitre.org) graph construction to categorize vulnerabilities. In the revision applied by NVD in 2014, this classification encompassed 35 categories. However, the earlier version had only 23 categories. Thus, to find trends and compare categories for the entire time period, we merged these two classifications.

We applied the following rules to prune the two existing classifications:

- Each category of 2014 classification that didn't exist in the previous version was merged with its parent node in the tree. If the parent didn't exist in the previous version, the process was repeated until we found an existing category or reached the tree root. The reason for this rule was that the details of a new category in 2014 were unknown in the previous classification. Therefore, we generalized the category to an abstract one. For example, we mapped the 2014 "improper access control" classification to the abstract category "permissions, privileges, and access control issues."
- The root category of 2014 classification (location) was mapped to the "not in CWE" category in the previous version because it didn't exist before 2014.
- The "design errors" category existed in the previous classification, but doesn't exist in the current version. We mapped this category to "not in CWE" for the same reason as the previous rule.
- The "insufficient information" and "other" categories were merged with "not in CWE" and renamed "useless" because we couldn't extract any useful information about vulnerability trends.

Our final modified version had 20 categories: authentication issues (auth); buffer errors; code injection; configuration issues; credential management vulnerabilities; cross-site request forgery (CSRF); cross-site scripting (XSS); cryptographic issues (crypto); format string vulnerabilities; information leakage; insufficient input validation; link following; numeric errors; OS command injection (OS); path traversal; permissions, privileges, and access control issues (access control); race condition; resource management errors; SQL injection (SQLi); and useless. (See cwe.mitre.org for category definitions.)

Vulnerability Trends at a Glance

Figure 1 shows the overall vulnerability trends for 2008 through 2014. The blue and orange lines represent all vulnerabilities and severe vulnerabilities, respectively, registered on NVD. We demonstrate the ratio of severe vulnerabilities to all reported vulnerabilities for each year on the orange line.

Although the number of all vulnerabilities decreased in 2010 and 2011, the overall trend was ascendant. However, the number of severe vulnerabilities decreased. Even in 2014, when an exponential growth occurred in the amount of vulnerabilities, only 24 percent were severe. In other words, although the number of reported vulnerabilities increased, only a limited number of them led to significant consequences when exploited.

Increases and Decreases in Vulnerabilities

Although overall trends are useful for analyzing the ceaseless competition between attackers and defenders, we required more detailed information to discuss future opportunities and threats. We found that whereas the amount of some types of vulnerabilities decreased,

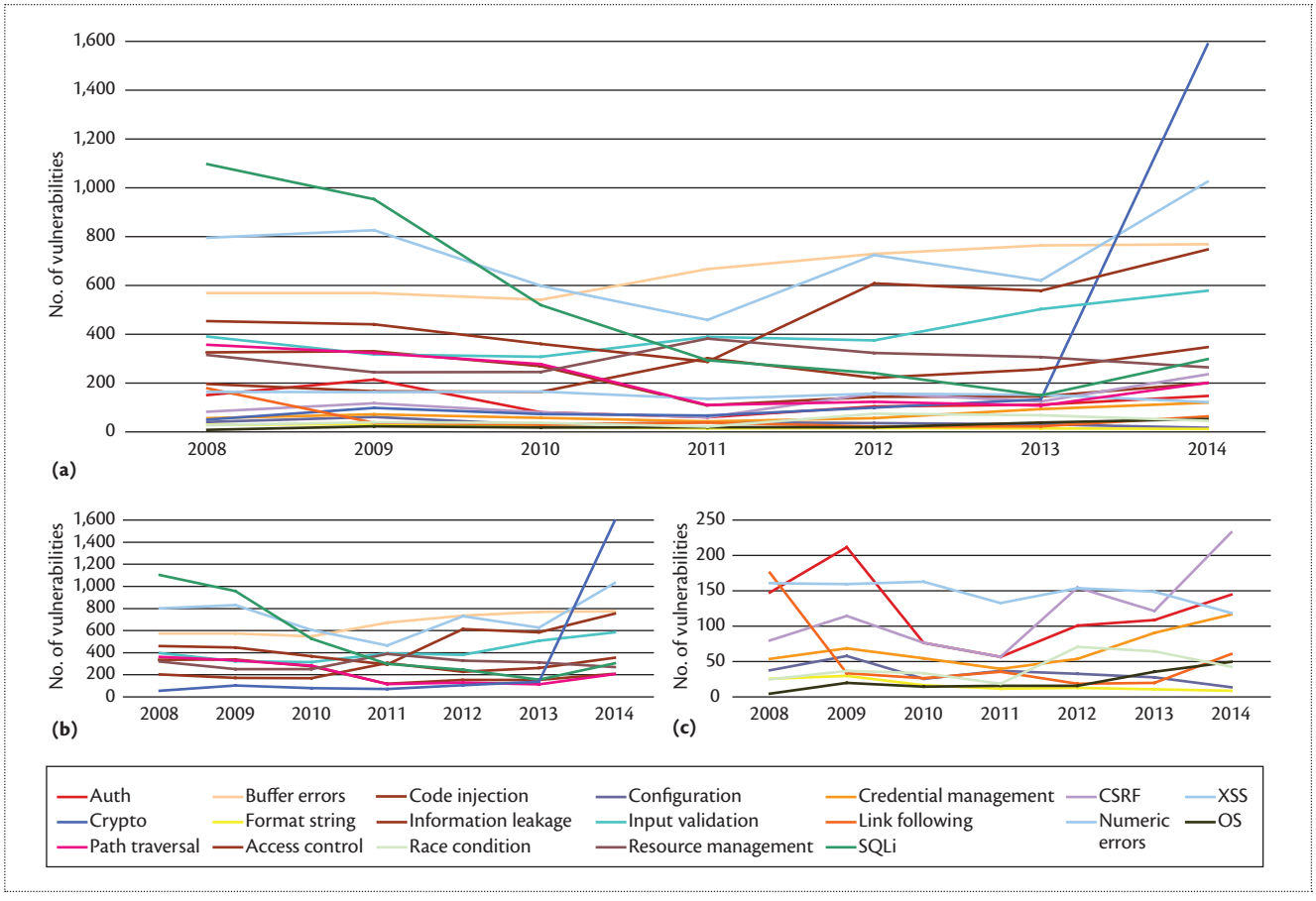

Figure 2. Vulnerability trends for (a) all categories and for (b) the most frequently and (c) the least frequently reported categories (fewer than 250 occurrences per year). Among the less frequent vulnerabilities, cross-site request forgery (CSRF) and link following showed the greatest growth and decline, respectively. A generally upward trend was also indicated in three categories: CSRF, credential management, and authentication vulnerabilities. XSS is cross-site scripting, and SQLi is SQL injection.

other types increased. Along these lines, each category is presented separately in Figure 2. Because the scales of vulnerability occurrences vary from category to category, we further split the main chart into Figures 2b (most frequent) and 2c (least frequent). If a vulnerability occurred fewer than 250 times in each year, it appears in Figure 2c.

As these charts illustrate, most categories had limited ebb and flow in a bounded range. The most interesting observation is the fluctuation patterns in crypto and SQLi.

Crypto vulnerabilities skyrocketed to number one in 2014—a more than 1,100 percent increase over the previous year—and approximately 88 percent of these vulnerabilities occurred in September and October of that year. The main cause of this surge was a specific type of Android application vulnerability. There were many Android applications for which X.509 certification wasn't properly verified, exposing them to man-in-the-middle attacks. However, this vulnerability type decreased in later months, so it could be described as a "flash in the pan." Despite this, security testing of the implementation of cryptographic features should be further researched to restrict possible similar future vulnerabilities, especially for Android applications.

Although SQLi didn't change suddenly in incidence, the number of this vulnerability type gradually—but greatly—decreased over the studied time period. The magnitude of the standard deviation (373), average absolute deviation (300), and maximum-to-minimum difference (947) compared to the average value (503) proves the wide variability in SQLi.

Figure 2c demonstrates that among the less frequent vulnerabilities, CSRF and link following showed the greatest growth and decline, respectively. A generally upward trend was also indicated in three categories: CSRF, credential management, and authentication issues. Thus, we'll likely hear more about these vulnerabilities in the future.

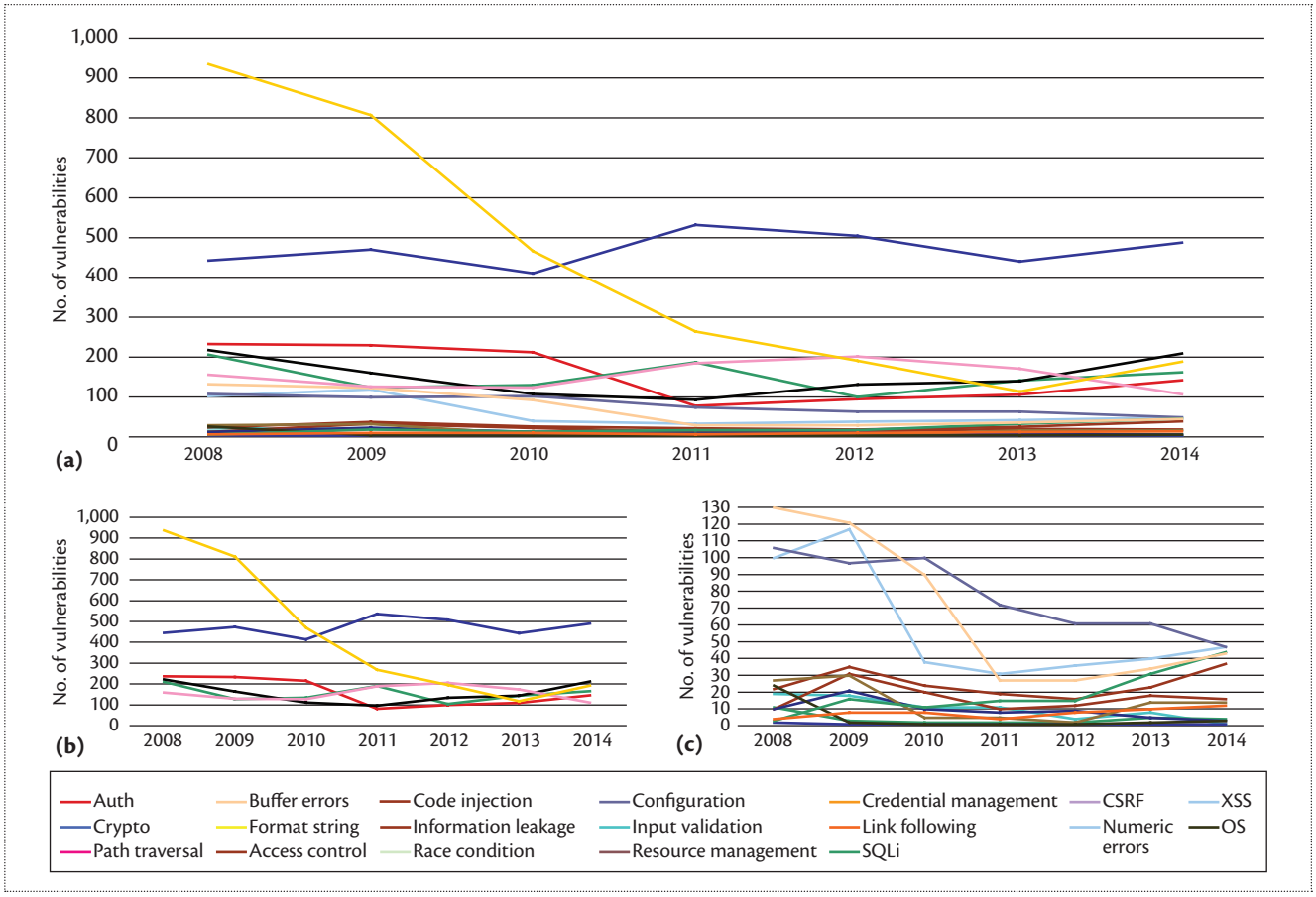

Figure 3. Severe vulnerability trends for (a) all categories and for (b) the most frequent and (c) the least frequent severe categories (fewer than 150 occurrences per year).

Which Vulnerabilities Are the Most Severe?

Although some vulnerabilities are common and reported more than others, they might not have serious consequences for the system. Thus, it's worthwhile to identify the most dangerous vulnerabilities. We used the Common Vulnerability Scoring System (CVSS) base score as a metric to extract the most serious vulnerabilities. The vulnerabilities with high severity (CVSS score of 7 to 10) were assumed to be dangerous.

Figure 3 illustrates the trends for severe vulnerabilities. As with Figure 2, we split the main chart into two parts: Figures 3b and 3c. Figure 3c shows vulnerabilities with fewer than 150 occurrences per year.

Figure 3b shows the results from our tracking of the six most reported severe vulnerabilities. Other categories rarely occurred and are therefore shown in Figure 3c. Comparing the diagrams reveals the following trends:

- Buffer errors are almost always at the top of the severe vulnerability list with an ordinary fluctuation during the study period. The average absolute deviation is 33. This variation is acceptable when compared to the magnitude of the average value (466).
- Although SQLi vulnerabilities are severe, the overall trend for this type is descending.
- The number of severe access control vulnerabilities is in an upward trend. Accordingly, developing more automatic tools to detect access control vulnerabilities will be of high value.
- Fluctuations were normal in some types of severe vulnerabilities. Other vulnerabilities either were reported less or weren't severe. (SQLi vulnerabilities are one exception in that they fluctuated widely.)

NVD has classified many severe vulnerabilities in its "insufficient information" category. We excluded these vulnerabilities from Figure 3 because their details were unknown or unspecified, so little useful information could be extracted.

So which categories are the most severe? We used two factors to determine each category's severity: the ratio of the number of occurrences of a specific severe

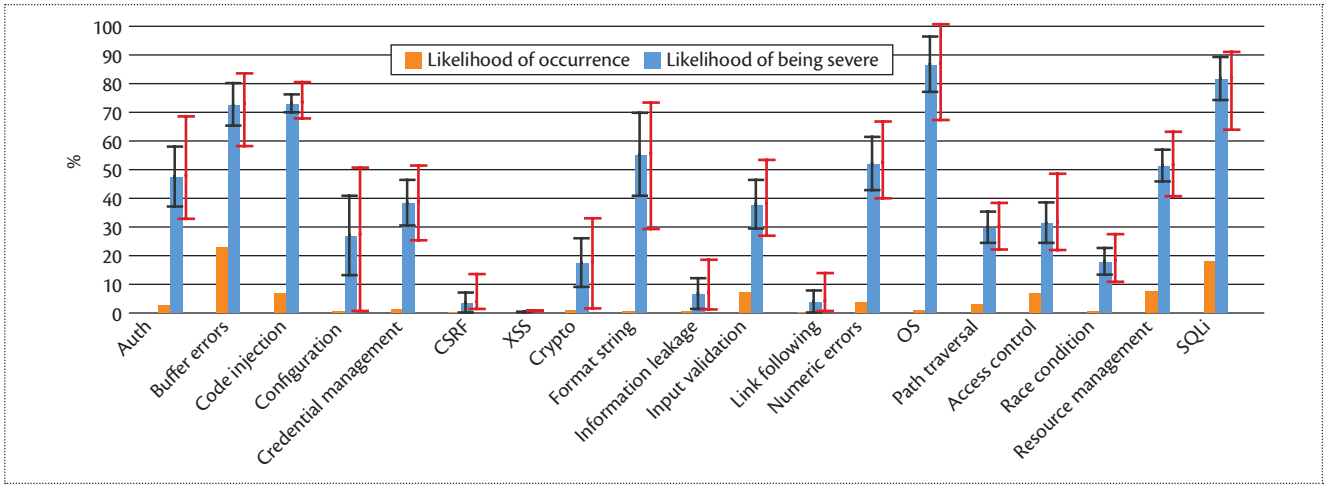

Figure 4. Severity of vulnerability types. We determined the ratio of the number of occurrences of a specific type of severe vulnerability to all reported severe vulnerabilities as well as the number of all reported vulnerabilities in the corresponding category. The black error bar shows the 95 percent confidence interval of the standard deviation; the red error bar illustrates the minimum to maximum values.

vulnerability type to all reported severe vulnerabilities, and the number of all reported vulnerabilities in the corresponding category. These factors reflect the importance of knowing not just the number of occurrences for any given severe vulnerability in each category but also the probability that the reported vulnerability will cause severe damage.

Figure 4 shows these two factors simultaneously. The quantities are calculated based on the average values over the seven years. We also added two error bars for better visualization of the results' variation: the black error bar shows the 95 percent confidence interval of the standard deviation, and the red error bar illustrates the range of changes from minimum to maximum values for 2008 through 2014.

The diagram illustrates that buffer errors, SQLi, code injections, and resource management errors are the most severe vulnerabilities, because they're more common than other vulnerabilities (the probability of each of them occurring is greater than 5 percent) and often have severe effects (the probability of causing damage is greater than 50 percent). To give one example, nearly 23 percent of all severe vulnerabilities were buffer errors and 72 percent of buffer errors were severe. Also worth mentioning is that OS command injection rarely occurred; however, when it did, its exploitation had a high probability—nearly 85 percent—of causing severe software problems.

Great Benefit with Relatively Little Effort

We'll now examine whether the most common vulnerability types are the same every year, and, if they're repeated every year, how many vulnerabilities could be prevented by avoiding these common types. In other words, can we find a rule similar to the Pareto principle for vulnerability prevention? Which vulnerability types should be prevented first to gain maximum benefit?

Figure 5 shows the five most often reported vulnerabilities in each of the past seven years. These vulnerabilities account for approximately 50 percent of all reported vulnerabilities in the year. Therefore, this figure provides an abstract view of the most prevalent vulnerabilities each year.

Two of the most often reported web application vulnerabilities were SQLi and XSS. SQLi was the most reported vulnerability in 2008 and 2009, but the number declined in 2010 and finally disappeared from the top five list in 2011. XSS was almost always the second most popular vulnerability, except in 2010 when it was first.

Although specific web application vulnerabilities were noteworthy until 2010, buffer errors gradually became the most often reported vulnerability from 2011 to 2013.

The figure also illustrates that just nine categories encompass all the vulnerabilities among the top five most reported during the seven years. We also conclude that five vulnerability categories—buffer errors, XSS, access control vulnerabilities, SQLi, and insufficient input validation—are more interesting from the attacker point of view. Thus, it's worth paying particular attention to these vulnerability categories and cautioning developers to avoid them.

Figure 6 shows the top four severe vulnerability categories in each year. Like the previous chart, these vulnerabilities account for approximately 50 percent of all severe vulnerabilities reported each year. It's amazing that just six categories encompass the top four reported

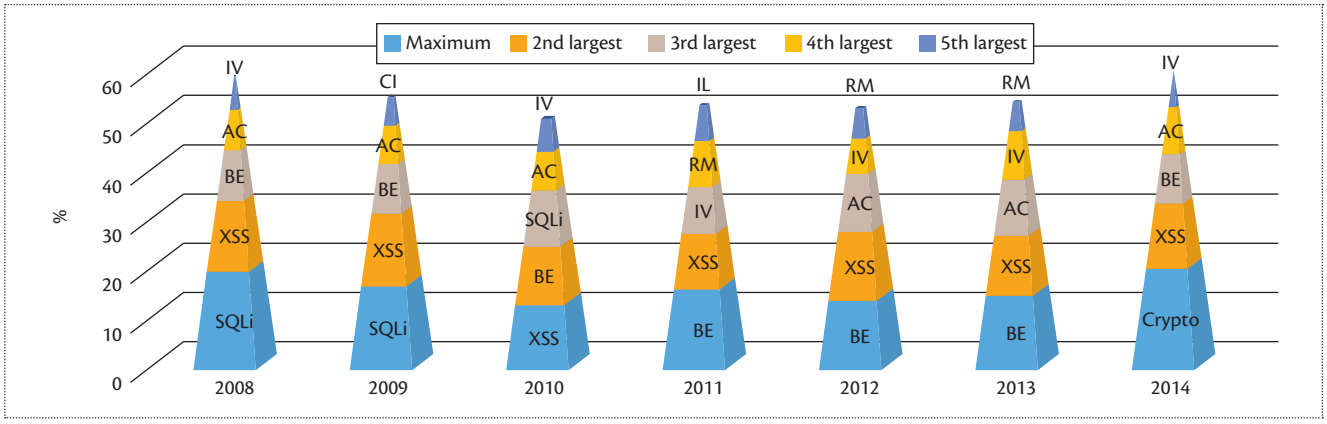

Figure 5. The top five yearly vulnerabilities. AC is access control vulnerabilities, BE is buffer errors, CI is code injection, IL is information leakage, IV is insufficient input validation, and RM is resource management errors.

Figure 6. The top four severe vulnerabilities each year. Six categories encompass the top four reported severe vulnerabilities: buffer errors, SQL injection, access control vulnerabilities, insufficient input validation, resource management errors, and code injection.

severe vulnerabilities: buffer errors, SQLi, access control vulnerabilities, insufficient input validation, resource management errors, and code injection.

We have reviewed how SQLi and XSS vulnerabilities change over time (as shown in Figure 5). But there are some interesting points to be noted when considering severity. Surprisingly, XSS, which has been the second most often reported vulnerability, has never been the most often reported severe vulnerability. In other words, although many XSS vulnerabilities are reported every year, they seldom cause serious problems. Moreover, in spite of the reduction in the amount of reported SQLi, this category is still one of the most dangerous vulnerabilities.

It's also useful to look at the most reported vulnerabilities over the entire seven years. This information helped us determine which vulnerabilities were most important for the entire time period and decide not only which vulnerability types should be prevented first to maximize benefit but also how many vulnerabilities could be prevented by avoiding these vulnerability types.

Figure 7 illustrates the average number of the most reported vulnerabilities from 2008 through 2014. We chose only those categories with more than 5 percent of the average number of reported vulnerabilities. We aggregated the other vulnerability types, representing them in the figure as "other categories." The categories are arranged clockwise from most to least reported vulnerability.

The numbers on the chart indicate each category's average number of vulnerabilities. For example, code injection on average makes up 7 percent of all reported severe vulnerabilities. But only 4 percent of all vulnerabilities are code injections. Some slices of the doughnut contain two numbers because we determined the

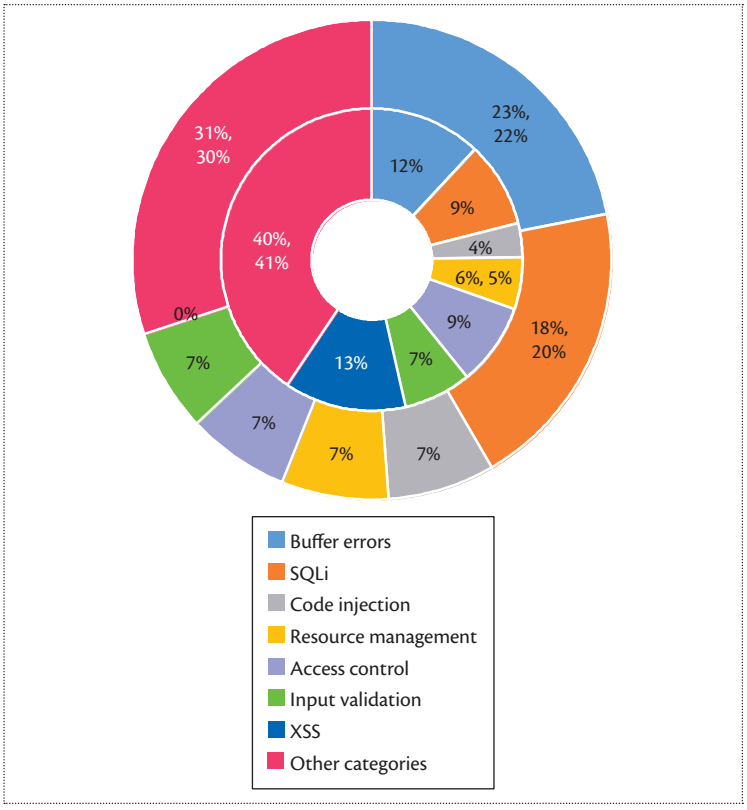

Figure 7. Average percentage of occurrence of each vulnerability type. The inside doughnut represents all vulnerability categories, whereas the outside doughnut represents only severe vulnerabilities. Where two numbers are shown, the left number is the average percentage of vulnerabilities in each category in each year, and the right number is the total number of reported vulnerabilities in the category divided by the total number of all reported vulnerabilities.

average quantity in two ways: the left number was calculated by averaging the percentage of vulnerabilities in each category in each year, and the right number by dividing the total number of reported vulnerabilities in the category by the total number of all reported vulnerabilities. (For the outer doughnut, the number of vulnerabilities is replaced with the number of severe vulnerabilities in the formula.) We show only one value if the two formulas yield equal results.

Astonishingly, the average number of reported severe vulnerabilities for just six categories is more than 5 percent. Therefore, approximately 70 percent of severe vulnerabilities and 46 percent of all vulnerabilities could be prevented by avoiding buffer errors, SQLi, code injection, resource management errors, access control issues, and insufficient input validation. Moreover, if programmers watch out for XSS in addition to these six categories, nearly 60 percent of all vulnerabilities might be prevented.

Figure 8 shows the distribution of vulnerabilities in each category from 2008 through 2014. Each category can be compared with others in the same year or in other years. The sum of the portions isn't 100 percent in each year because we excluded the "useless" category from the chart. Although this category comprises some special types that can't be integrated into our merged classification, it predominantly comprises vulnerabilities from NVD's "insufficient information" class. For example, more than 18 percent of vulnerabilities reported in 2013 didn't have sufficient information to be classified into the existing categories.

Analyzing seven years of NVD statistics for software vulnerabilities yields the following results:

- The total number of reported vulnerabilities is ascending. However, the portion of vulnerabilities that might cause severe problems is descending.
- Buffer errors, XSS, and access control problems were on the list of most reported vulnerabilities almost every year. Furthermore, buffer errors are almost always at the top of the severe vulnerabilities list.
- There was an upward fluctuation in insufficient input validation, CSRF, and credential management vulnerabilities. But the number of reported vulnerabilities grew most for crypto in September and October 2014. However, this type of vulnerability decreased in the following months. Thus, we guess that a valley will be formed near this peak.
- OS command injections would cause severe consequences if exploited, but fortunately, this exploitation rarely happened during the study period. However, buffer errors, SQLi, code injections, and resource management errors are the most severe—they're not only more common but also lead to severe problems.
- More than 70 percent of severe vulnerabilities could be prevented by avoiding these categories: buffer errors, SQLi, access control, insufficient input validation, resource management errors, and code injection. Moreover, 60 percent of all vulnerabilities wouldn't have occurred if programmers had prevented XSS in addition to these six categories.

We recommend that programmers learn more about input validation, XSS, and CSRF. Furthermore, we encourage researchers to develop more methods and tools to detect cryptographic problems and access control vulnerabilities. Developers should also be aware of the severe consequences of buffer errors, SQLi, and code injections. ■

Hossein Homaei is a PhD candidate in the Department of Computer Engineering and Information Technology at the Amirkabir University of Technology

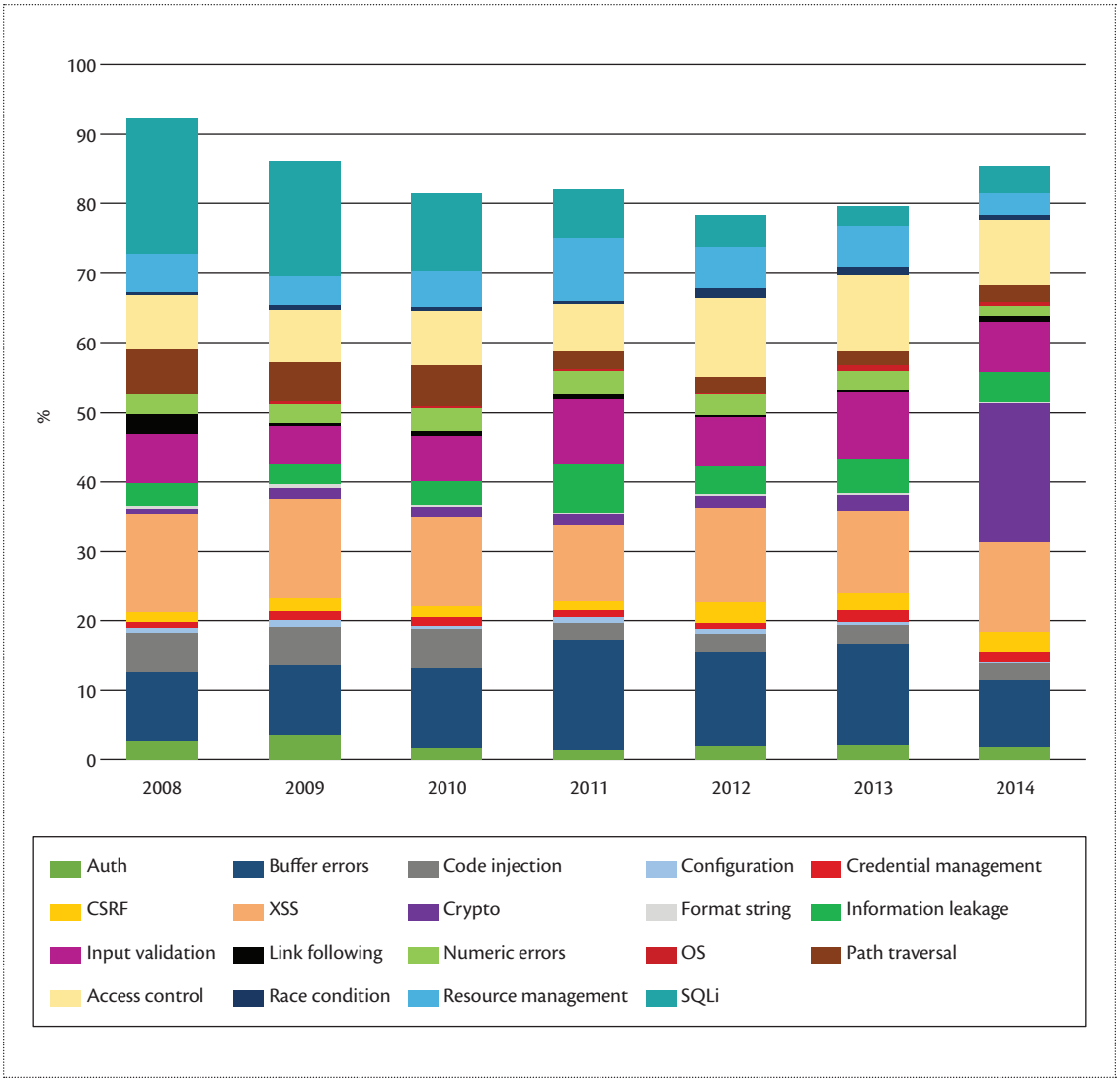

Figure 8. Vulnerability distribution. Shown are the number of reported vulnerabilities in each category from 2008 through 2014.

(Tehran Polytechnic). His research interests include formal methods in computer security, vulnerability analysis, and security quantification. Homaei received an MS in information security from the Amirkabir University of Technology. Contact him at homayi@aut.ac.ir.

Hamid Reza Shahriari is an assistant professor in the Department of Computer Engineering and Information Technology at the Amirkabir University of Technology. His research interests include information security, especially vulnerability analysis, security in e-commerce, trust and reputation models, and database security. Shahriari received a PhD in computer engineering from the Sharif University of Technology. Contact him at shahriari@aut.ac.ir.