SPOTLIGHT

Cracks in the Foundation:

Web and Application Framework Vulnerabilities

RISKSENSE®

Introduction

An organization's software stacks and web-facing applications are some of the most fundamentally important assets in an enterprise digital environment. Everything from blogs to the most sophisticated applications are essential to how organizations serve their users and interact with the outside world. The applications' exposure to the outside world also means that they are some of the assets most exposed to attack.

And while resources such as the OWASP Top 10 and Top 25 Programming Most Dangerous Software Errors provide an important way of classifying the most common types of weaknesses, it is often hard for developers and security staff to know how specific choices such as coding language and application frameworks impact the attack surface of their web assets.

This Spotlight Report puts some of the most popular languages and frameworks under the microscope to see:

- The languages and frameworks where vulnerabilities are most common
- Which vulnerabilities are weaponized the most
- The underlying weaknesses that lead to vulnerabilities
- Patterns of weaknesses found in particular frameworks
- How to prioritize vulnerabilities based on real-world context

Key Findings

2019 Vulnerabilities are Down, But Weaponization is Up

Web and application frameworks have largely been immune to the massive spike in vulnerabilities observed in the U.S. National Vulnerability Database (NVD) over the past few years. And while the overall number of framework vulnerabilities is down compared to previous years, the weaponization rate has jumped to 8.6%; more than double that of the NVD average (3.9%) for the same time frame. This uptick in weaponization was primarily due to increased weaponization in Ruby on Rails, WordPress, and Java.

WordPress and Struts are the Most Weaponized

In terms of web and app frameworks, WordPress and Apache Struts were responsible for the most weaponized vulnerabilities – those vulnerabilities for which exploit code exists that can actually take advantage of the weakness. These two frameworks alone accounted for 55% of the weaponized vulnerabilities over the past 10 years. WordPress faced a wide variety of issues, but cross-site scripting (XSS) was the most common problem, while Struts was dominated by input validation problems. Their respective underlying languages, PHP for WordPress and Java for Struts, were also the most weaponized languages in the study.

Keep an Eye on Node.js and Django

As a whole, JavaScript and Python frameworks showed the lowest weaponization of vulnerabilities, however the news wasn't all good. Node.js had a notably higher number of vulnerabilities than other JavaScript frameworks with 56 vulnerabilities, although only one has been weaponized to date. The vulnerabilities covered a wide range of issues including input validation, information exposure, and resource exhaustion issues. Likewise, Django had 66 vulnerabilities with only one weaponized. While weaponization remains low, the large number of vulnerabilities in these frameworks leaves them open for the potential for risk.

Certain Frameworks Produce the Most Vulnerabilities

The majority of weaponized vulnerabilities have consistently been tied to the same technologies over the course of the past decade. WordPress, PHP, Apache Struts, Java, and Drupal remained the top five weaponized technologies when evaluated over both a 10- and 5-year time horizon. While the types of weaknesses have changed considerably over the course of a decade, the same players have remained responsible.

Frameworks Are Getting XSS Under Control

While XSS issues were the most common weakness in the full 10-year dataset, it dropped to 5th when analyzed over the last 5 years. This is a good sign that frameworks are making progress in this important area. XSS weaknesses over the past 5 years have largely been due to issues in WordPress.

Input Validation Becomes the Top Weakness

Input Validation has become the top problem for frameworks, accounting for 24% of all weaponized vulnerabilities over the past five years. Input validation covers a wide range of techniques and is one of the most important capabilities of an application framework. Input validation problems have been largely tied to Struts, WordPress, and Drupal. Access control issues were also on the rise, and mostly attributable to Java.

Injection Weaknesses are Highly Weaponized

Vulnerabilities tied to SQL injection, code injections, and various command injections remained fairly rare, but had some of the highest weaponization rates, often over 50%. In fact, the top three weaknesses by weaponization rate were Command Injection (60% weaponized), OS Command Injection (50% weaponized), and Code Injection (39% weaponized).This often makes them some of the most sought after weaknesses by attackers.

Scoring Systems Need Real-World Context

CVSS v2 and v3 scores were not particularly efficient at predicting vulnerability weaponization. Only 12.4% of CVSS v2 High vulnerabilities were weaponized, while 9% of CVSS v3 Critical vulnerabilities were weaponized. By comparison, models that include intelligence from the wild showed 95% of Critical vulnerabilities were weaponized. This ability to quickly find and work on the issues that pose the greatest risk is crucial for organizations to ensure they spend their effort on the right tasks.

Table of Contents

1. Background on Application Frameworks

Modern applications rely on a wide range of interdependent technologies in order to deliver functionality. These layers of technology are commonly referred to as an application "stack," which can include things like the web server, databases, programming or coding languages and scripting components, virtualization and containers, and a variety of other components.

The most traditional view of the application stack is defined by the LAMP stack. LAMP is an acronym built from the most standard components of the stack including:

- **Operating System – LINUX**
- **Web Server – Apache**
- **Database – MySQL**
- **Language/Scripting – PHP, Perl, Python**

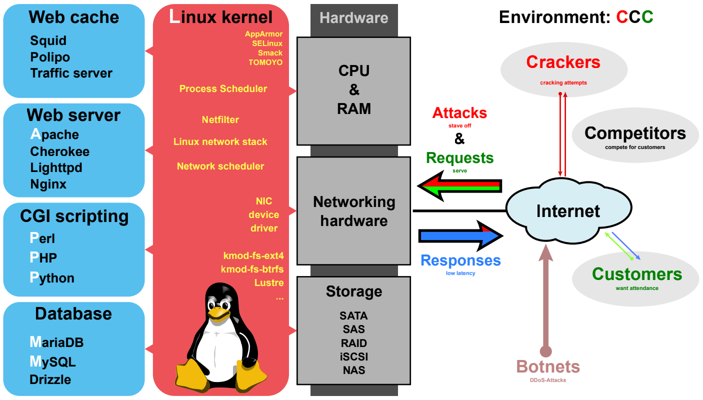

Figure 1(a): The Traditional LAMP Stack
Credit: By Shmuel Csaba Otto Traian, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=28224098

Broadly speaking, the analysis in this report focuses on the "P" in the LAMP stack. Future studies will focus on other parts of the application stack. However, we are not limiting our analysis only to PHP, Perl, and Python. Firstly, there are some very important additional technologies such as JavaScript and Java that are not covered by the three Ps. Additionally, web and app frameworks built on top of the various languages have become essential components of modern applications, and these technologies are likewise featured prominently in our analysis.

What are Application Frameworks?

Web frameworks streamline the development and deployment of applications and websites. Instead of requiring the developer to code every line of PHP, HTML, etc., a framework can provide the developer with ready-made building blocks for many common tasks. This might include things like session management or validating user input to the application. Instead of manually coding these functions over and over, the framework can provide these components automatically. Likewise the framework can handle the dynamic generation of HTML and other code based on user interaction. In short, they vastly simplify the development of dynamic web applications.

These frameworks are likewise built on one of the parent technologies we have seen earlier. For example, the Laravel and Symfony frameworks are built on PHP. Likewise, Node.js, AngularJS, and React are common frameworks built on JavaScript.

The Importance of Vulnerabilities in Frameworks

In general, application frameworks have made application development more secure overall. Instead of relying on manual coding to validate input every time a user interacts with an application, the framework can handle this automatically. With less manual coding frameworks can reduce the chance of manual errors.

However, better does not mean foolproof. Languages and frameworks can both contain their own vulnerabilities that can be exploited by attackers. And vulnerabilities in these fundamental components can leave applications vulnerable even if the developers are following the best coding practices.

Web framework vulnerabilities became mainstream news in 2017 when an Apache Struts vulnerability (CVE-2017-5638) led to the disclosure of 140 million user records in the Equifax breach. However, Struts is far from the only example. Attackers abuse vulnerabilities in frameworks in order to support malvertising campaigns, spread malware, or take over the site completely. For example XSS vulnerabilities such as CVE-2019-9978 in WordPress were recently attacked in the wild that could provide an attacker with control over the victim site.

1. Background on Application Frameworks (Continued)

Most importantly, the nature of web applications means that these vulnerabilities are typically exposed to the internet. For example a recent PHP vulnerability (CVE-2019-11043) allowed even unskilled attackers to force a site to execute a malicious payload simply by using a specially crafted URL. When a new vulnerability is discovered, attackers can easily identify sites built from the vulnerable framework or language and automatically scan for specific vulnerabilities.

2. Overview of Vulnerabilities

Web languages and frameworks have been around for a long time, and likewise have a long history of vulnerabilities. The earliest PHP vulnerabilities reach back to 1999, and the first Drupal and WordPress vulnerabilities were seen in 2002 and 2003 respectively.

And while all vulnerabilities are important, we intentionally focused our analysis on vulnerabilities from the past decade, when frameworks gained prominence. Specifically we analyzed 1,622 vulnerabilities spanning from 2010 through November of 2019.

Next we further honed in on the vulnerabilities that pose the greatest real-world risk to an organization. To this end, we highlighted weaponized vulnerabilities for which exploit code exists that can take advantage of the vulnerability.

These vulnerabilities pose the most immediate, real-world risk to an organization. In general, most vulnerabilities are not weaponized, and this trend holds true with our dataset, with only 209 out of 1,622 CVE entries being weaponized.

Next, we further analyzed the vulnerabilities with the highest impact, including those that enabled remote code execution (RCE) and/or privilege escalation (PE). We found 60 vulnerabilities that enabled remote code execution and another nine that enabled privilege escalation. Notably, all of the RCE or PE capable vulnerabilities were weaponized. As Figure 2(a) shows below, this ability to focus on weaponized and high impact vulnerabilities can greatly help security teams and developers with a highly prioritized and manageable list of to prioritize the vulnerabilities to focus on.

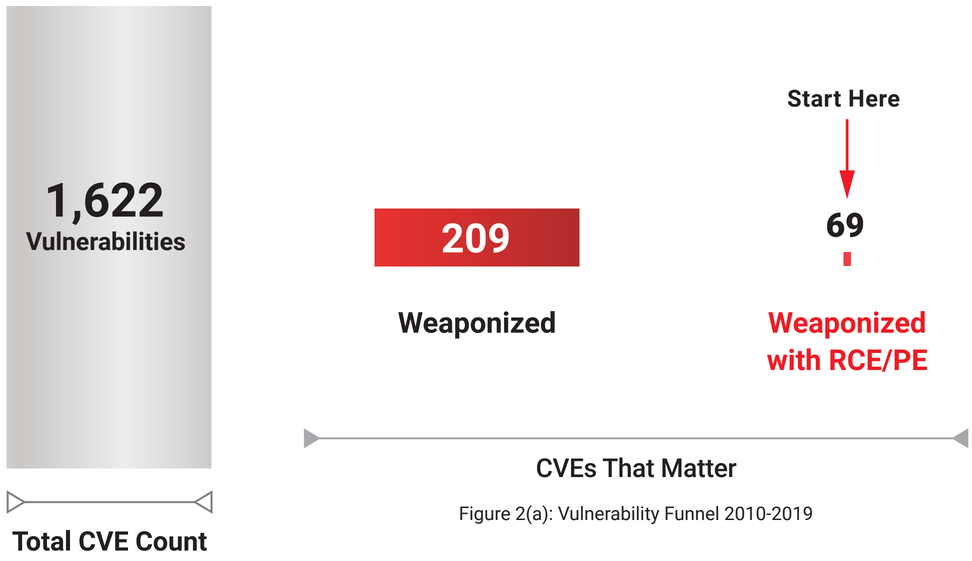

Figure 2(a): Vulnerability Funnel 2010-2019

2. Overview of Vulnerabilities (Continued)

Vulnerabilities by Year

We next analyzed the dataset by year in order to see how the vulnerability landscape is changing year over year. Figure 2(b) shows each year broken out by vulnerabilities that were weaponized versus those that were not. Note that the 2019 data only includes vulnerabilities through the end of October.

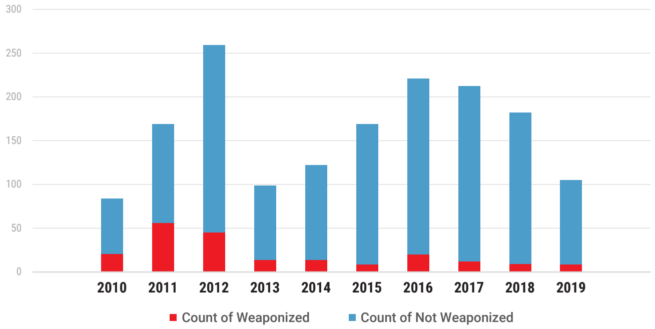

Figure 2(b): Framework CVEs by Year

Here we can see that weaponization rates have fallen considerably since the early part of the decade. 2011 had the highest weaponization rates with just over 1/3 of every language or framework vulnerability being weaponized compared to only 4.9% in 2018.

These results are particularly interesting if we compare them to the overall trends seen across all vulnerabilities seen in the U.S. National Vulnerability Database (NVD). For example, if we look at the list of all vulnerabilities tracked in the NVD in Figure 2(c), we can see a pronounced spike in vulnerabilities beginning in 2017. The total number of vulnerabilities almost tripled between 2016 and 2017, and the following years have maintained this pace. However, this spike is not observed in the framework data. While there are still plenty of vulnerabilities to keep developers and security teams up at night, they at least are not facing the massive spike of vulnerabilities seen in other software.

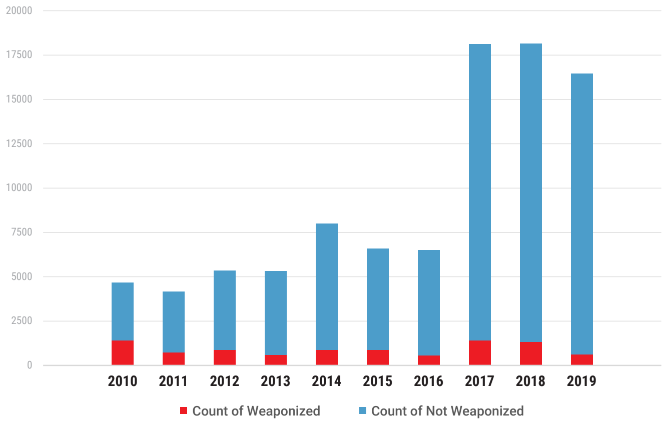

Figure 2(c): All NVD CVEs

We can also see that weaponization rates have generally declined in the overall NVD dataset as well. Figure 2(d) compares the yearly weaponization rates in the frameworks dataset to the overall NVD. Overall, we can see a steady decline in weaponization across both datasets. Notably, 2019 is the only year other than 2011 the framework weaponization rate (8.6%) significantly outpaced the overall NVD average (3.9%). This uptick in weaponization was primarily due to increased weaponization in Ruby on Rails, WordPress, and Java, and we will continue to monitor this area to see if this is an anomaly or part of an emerging trend.

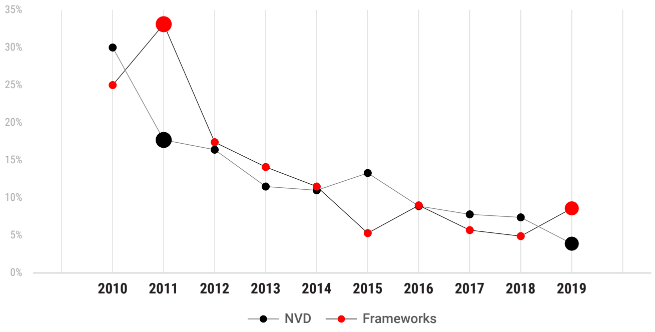

Figure 2(d): Weaponization Rates of Frameworks vs. the Overall NVD

3. The Most Vulnerable Frameworks

For developers, AppSec staff, and executives it is critically important to know which languages and frameworks have the most vulnerabilities and likewise which are weaponized the most. Figure 3(a) begins to bring this into focus. The list is ranked in terms of the greatest number of weaponized vulnerabilities. The maroon bar indicates the percentage of vulnerabilities that were weaponized for each framework.

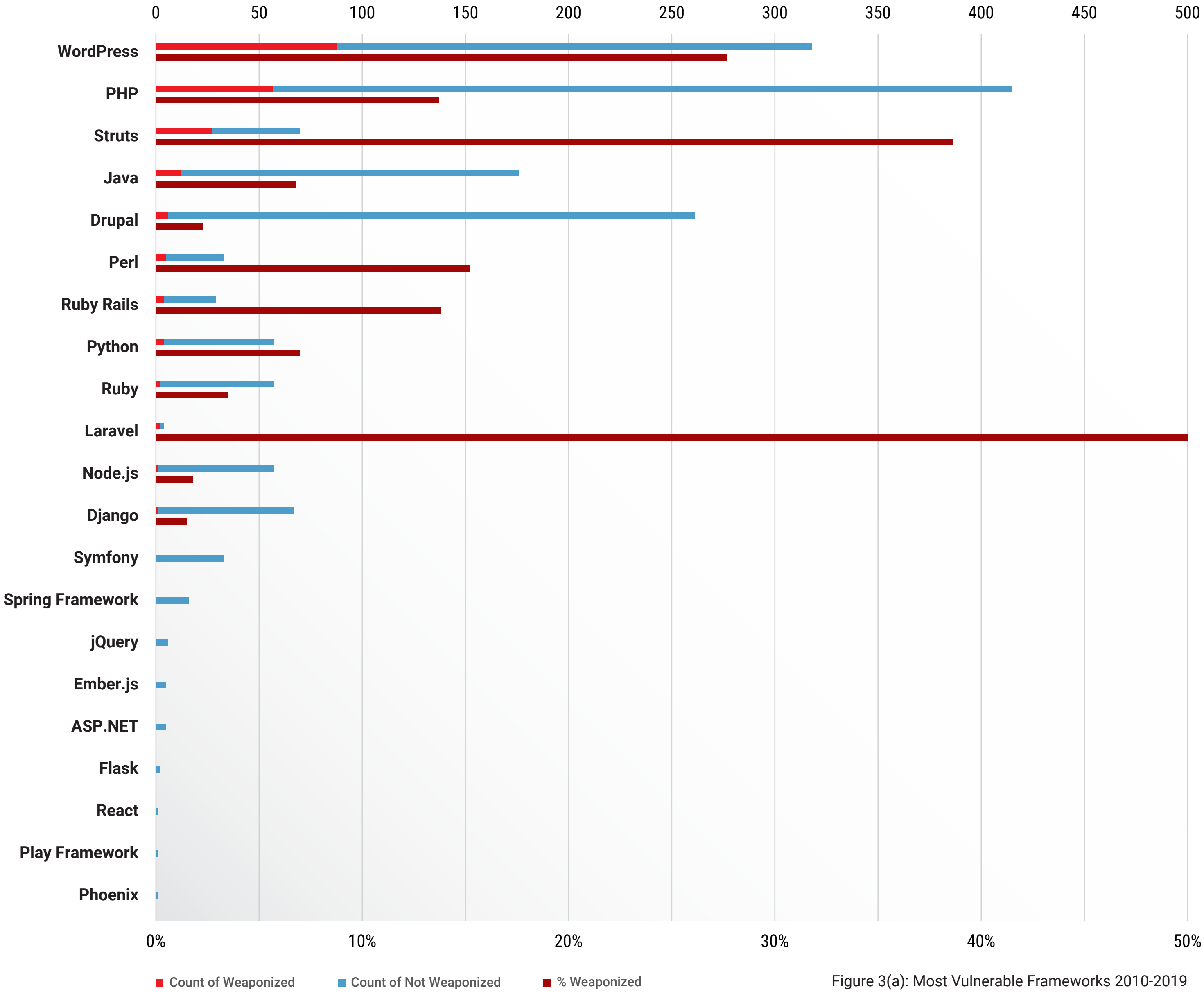

Figure 3(a): Most Vulnerable Frameworks 2010-2019

WordPress and Struts Lead the Way

Analyzing Figure 3(a), we can see that while PHP has the most total vulnerabilities, WordPress actually has the most weaponized vulnerabilities and the third highest weaponization rate. In total, 27.7% of WordPress vulnerabilities were weaponized. Apache Struts had the third most weaponized vulnerabilities and had one of the highest overall weaponization rates across all frameworks. 38.6% of all Struts vulnerabilities were weaponized. Only Laravel had a higher weaponization rate, but that was based on only four total vulnerabilities. In later sections we will dig into the specific weakness underlying these vulnerabilities.

3. The Most Vulnerable Frameworks (Continued)

Language and Framework Breakout

To make a more apples to apples comparison, we broke out the languages from the frameworks. Figure 3(b) provides a view of only the languages, while Figure 3(c) shows the frameworks organized by the language they are built on.

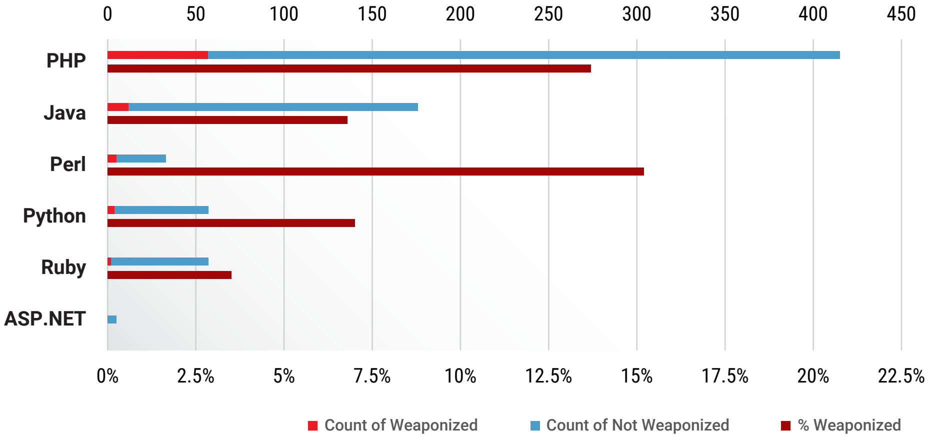

Figure 3(b): Languages 2010-2019

Here we can see that the PHP-based frameworks had both a high rate of vulnerabilities as well as weaponization. The Java frameworks were dominated by Struts vulnerabilities. Most notably the JavaScript and Python-based frameworks fared far better across the board in terms of weaponization. However Node.js and Django stood out in terms of overall vulnerabilities within JavaScript and Python respectively.

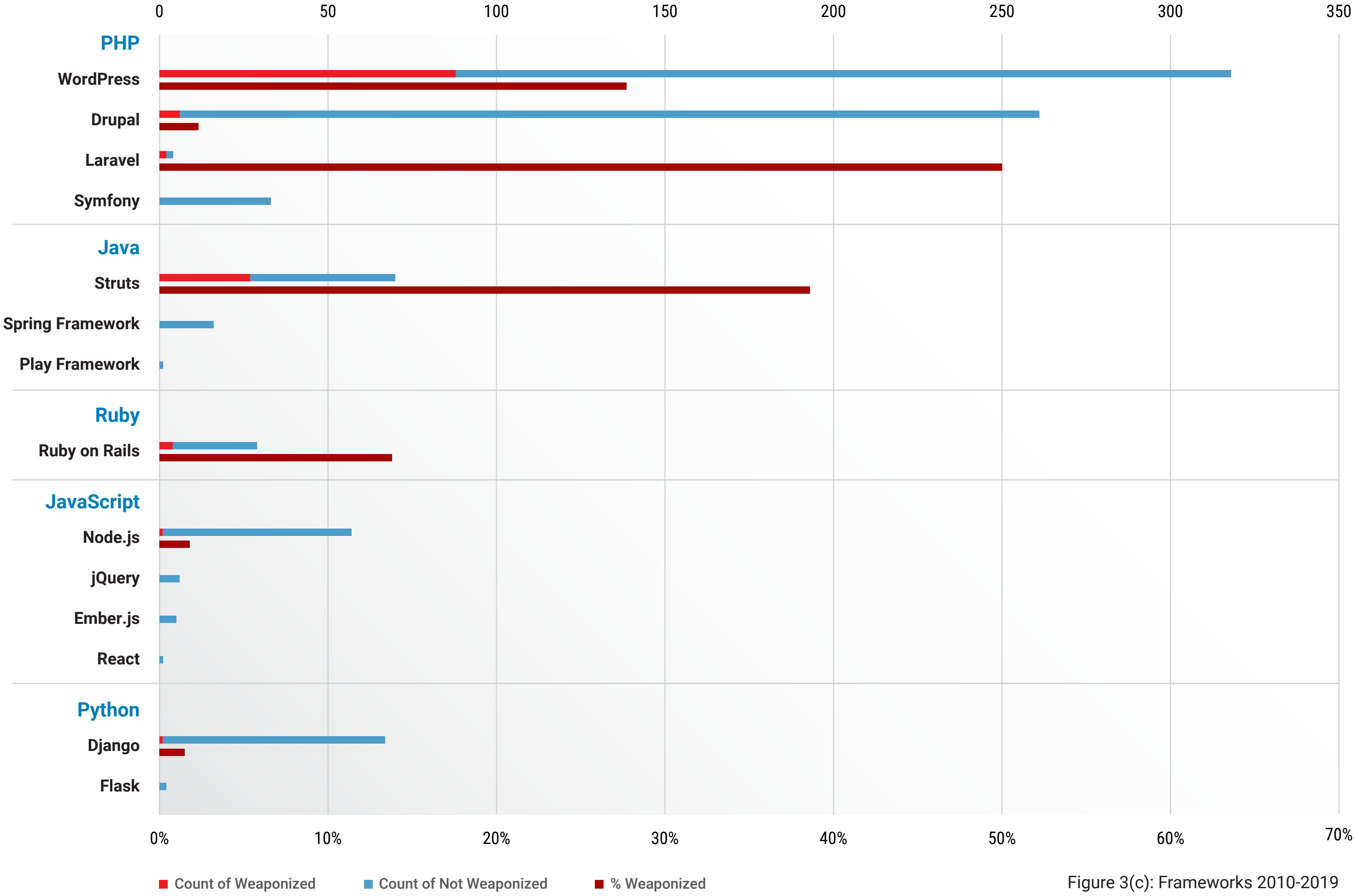

Figure 3(c): Frameworks 2010-2019

3. The Most Vulnerable Frameworks (Continued)

Trends in 2015-2019

Given that the adoption of frameworks has changed considerably over the past several years, it made sense to refocus our analysis on the most recent five years. Figure 3(d) shows the data from 2015 through the end of October 2019.

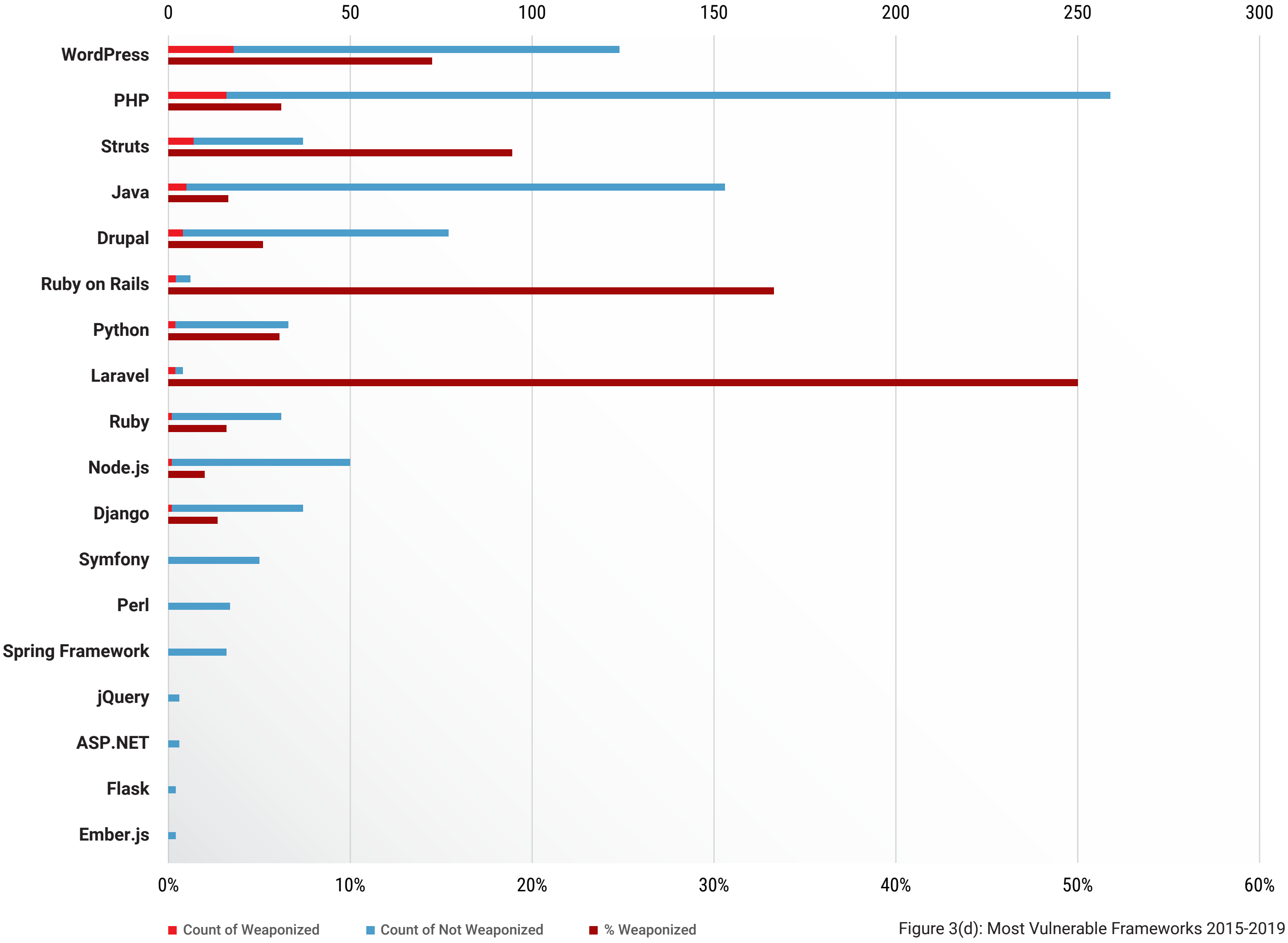

Figure 3(d): Most Vulnerable Frameworks 2015-2019

Interestingly, even though the overall count of vulnerabilities and weaponization rates decreased, the overall ranking of the frameworks remained almost unchanged. The top five weaponized technologies remained in the exact same order – WordPress, PHP, Struts, Java, and Drupal. The only significant changes were that both the Perl and Ruby languages dropped considerably, while all others remained in their previous order. This is particularly interesting as it shows that **while frameworks have undergone considerable changes between the front half and back half of the decade, the vulnerability hot spots have remained remarkably predictable.**

The same core group remained with minor changes even as we focused on 2019. PHP, WordPress, Java, and Drupal had the most vulnerabilities and weaponization. The only change was that Rails had more weaponized vulnerabilities in 2019 than Apache Struts.

4. Analyzing Framework Weaknesses

Once we've established where the vulnerabilities are, it is equally important to know what kind of weaknesses led to the vulnerabilities in the first place. Resources like the OWASP Top Ten Project and CWE Top 25 Most Dangerous Software Errors provide a regularly updated list of the most common types of weaknesses within applications.

As part of our analysis we looked at the Common Weakness Enumeration (CWE) codes that were associated with the vulnerabilities in our dataset. Unfortunately, 107 out of the 1,622 CVEs in our dataset did not have an associated CWE code, thus we limited our analysis to the remaining 1,515 CVEs.

The Most Weaponized Weaknesses of the 2010s

We began analyzing the full 2010 to 2019 dataset based on CWEs, once again prioritizing by those where the CVE was weaponized. Figure 4(a) provides a ranked list based on the total number of weaponized vulnerabilities.

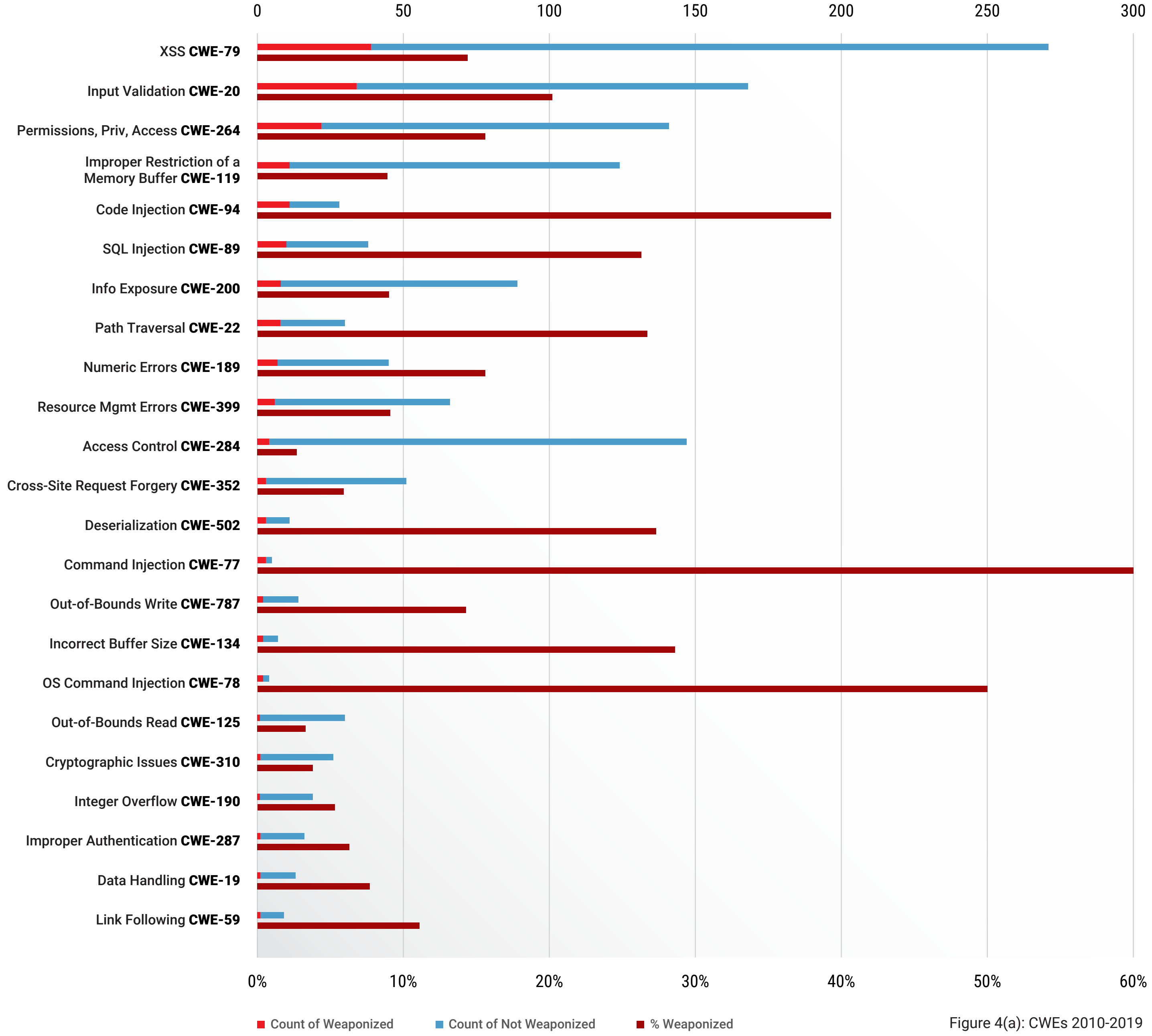

Figure 4(a): CWEs 2010-2019

4. Analyzing Framework Weaknesses (Continued)

Here we can see a few interesting findings:

- **Cross-Site Scripting (XSS) Leads the Way:** For the 2010s, XSS vulnerabilities were the most common type of weakness as well as the most weaponized. This aligns with recent analysis from security firm HackerOne, which found that XSS vulnerabilities were the most-rewarded type of vulnerability in bug bounty programs. However, we will see that frameworks have markedly improved in terms of avoiding XSS weaknesses in recent years.

- **Improper Input Validation (CWE-20):** While XSS easily had the most associated vulnerabilities, it was closely followed by Improper Input Validation in terms of weaponization. Unfortunately CWE-20 is also one of the most amorphous and generic CWE codes. Improper Input Validation could include a wide range of injection techniques, buffer overflows, and encoding issues to name a few. In fact, the Common Attack Pattern Enumeration and Classification list (CAPEC) associates 53 different attack patterns with CWE-20. However, it remains an important finding because input validation is one of the key capabilities that organizations often rely on a framework to address.

- **Injections Are Highly Weaponized:** Injection is the number one category in the latest OWASP Top 10, and this category was well represented in our framework data as well. Most interestingly, they showed some of the highest rates of weaponization. For example, Command Injection (CWE-77) showed a weaponization rate of 60%, OS Command Injection (CWE-78) at 50%, Code Injection (CWE-94) at 39%, and SQL Injection (CWE-89) at 26%. It is also important to note that many injection techniques not specified by a CWE are included in the Improper Input Validation CWE discussed above.

- **Improper Restriction Within a Memory Buffer:** CWE-119, “Improper Restriction of Operations within the Bounds of a Memory Buffer” is the #1 coding error on the CWE Top 25 and was the 4th most common weakness seen in our data. While this weakness is often associated with traditional applications, it likewise applies to web applications and frameworks as well. In our data these weaknesses were primarily tied to PHP with 9 associated CVEs, 2 tied to Python, and 1 tied to Java.

Injection Weaknesses

are rare, but highly weaponized, often over 50%

4. Analyzing Framework Weaknesses (Continued)

2015-2019: Big Changes in the Weakness Landscape

As we narrowed our focus to the last five years of the decade, we noticed significant changes in the types of weaknesses that were leading to problems in languages and frameworks. Table 4(b) shows the most commonly weaponized CWEs from 2015 through 2019.

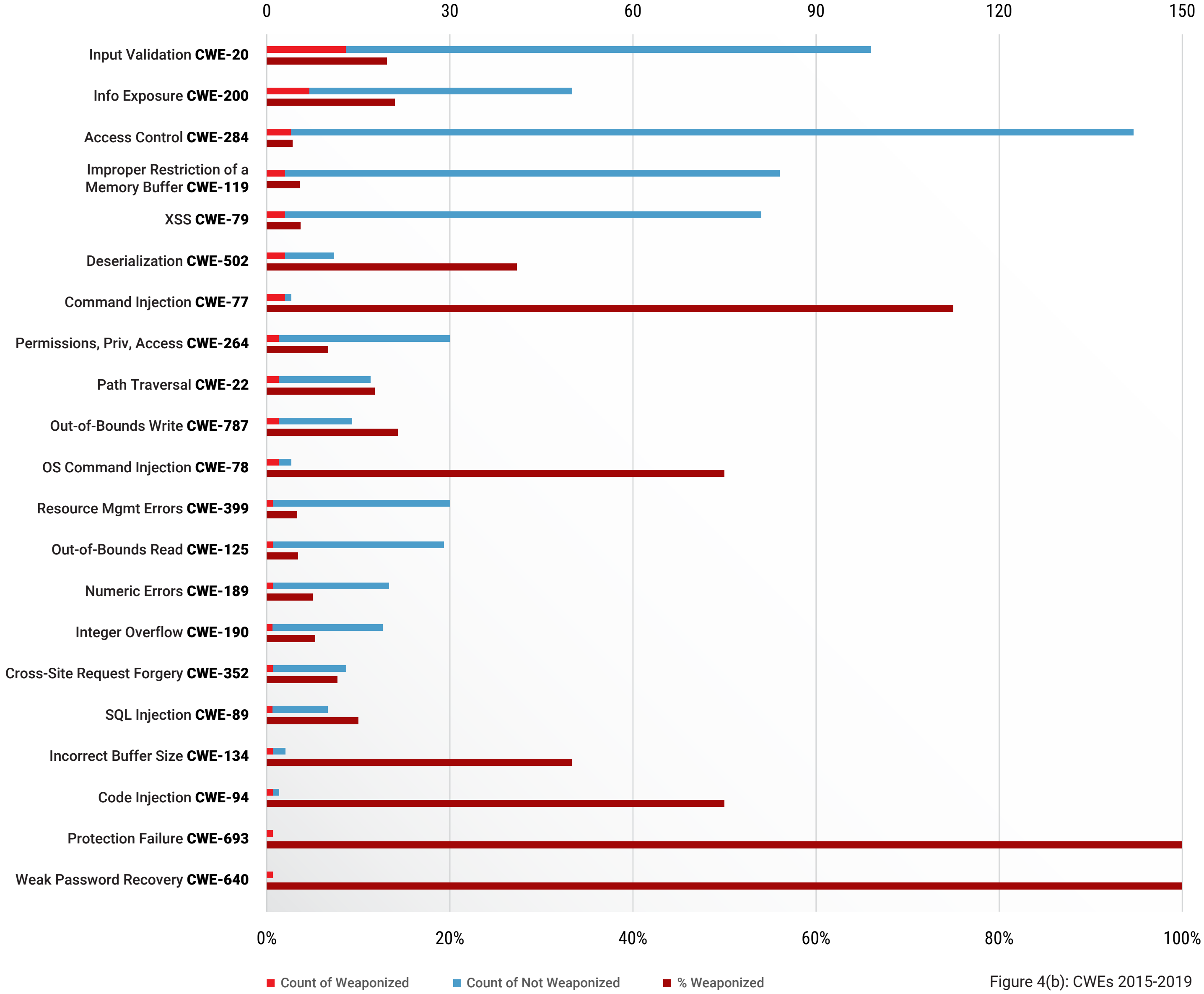

Figure 4(b): CWEs 2015-2019

4. Analyzing Framework Weaknesses (Continued)

Comparing this 5-year dataset to the 10-year set, we see some interesting changes:

- **Big Drop in XSS:** While XSS led all weaknesses in both overall volume and weaponization in the previous data, it dropped to 4th in terms of overall volume, and 5th in terms of weaponization. XSS was actually tied with Deserialization of Untrusted Data (CWE-502) and Command Injection (CWE-77) in terms of total weaponization. In the first half of the decade XSS accounted for 27% of all weaponized vulnerabilities, but dropped to 5.5% in the second half of the decade. This is a clear indication that frameworks have made good progress in preventing cross-site scripting. In terms of specific frameworks, WordPress was the primary source of both total and weaponized vulnerabilities.

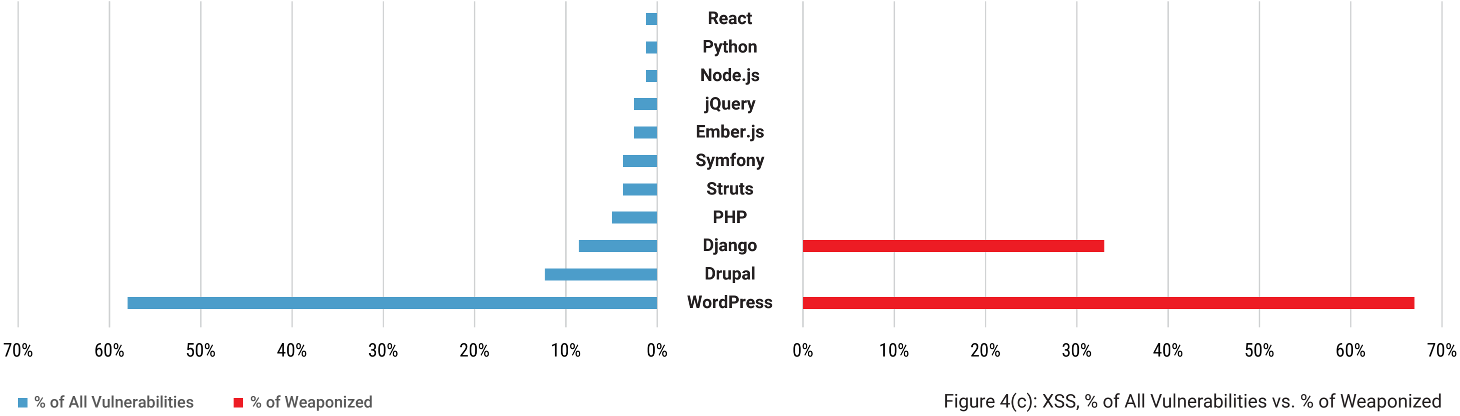

Figure 4(c): XSS, % of All Vulnerabilities vs. % of Weaponized

- **Input Validation Takes the Top Spot for Weaponization:** As XSS fell, Input Validation errors (CWE-20) remained very common and rose to the most weaponized vulnerability overall. As shown below, a variety of frameworks had input validation problems however Apache Struts was notably the most weaponized framework and the second biggest contributor of vulnerabilities overall.

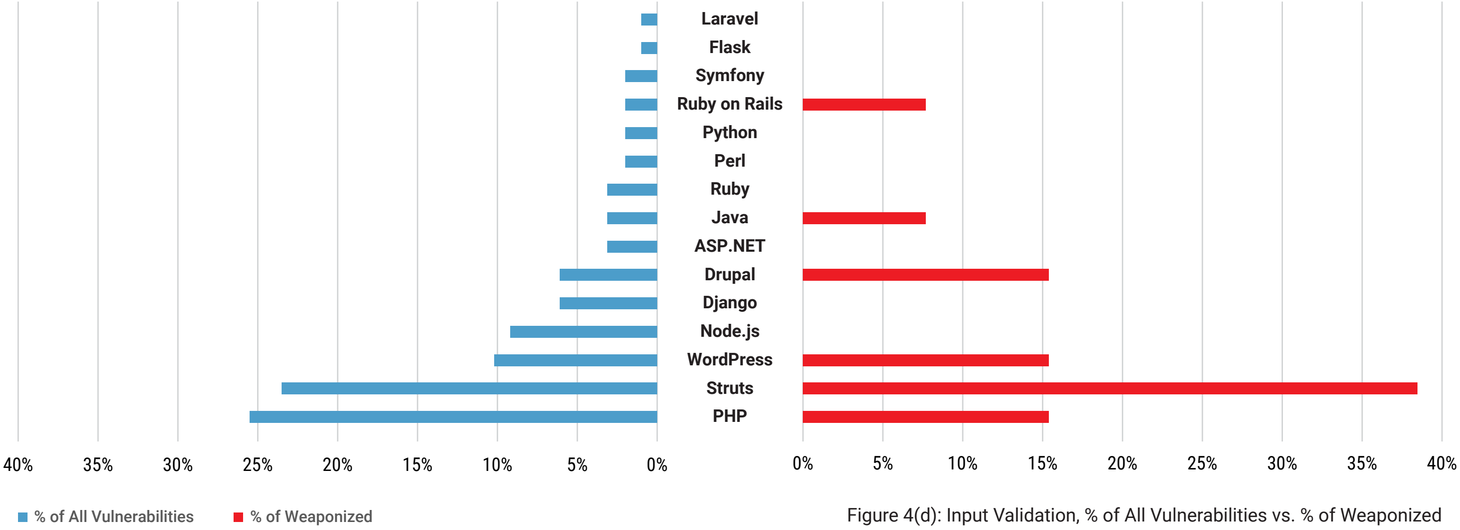

Figure 4(d): Input Validation, % of All Vulnerabilities vs. % of Weaponized

4. Analyzing Framework Weaknesses (Continued)

- **Information Exposure Jumps to #2:** As we focused on the back half of the decade, CWE-200 made a big jump from #7 to #2. This particular weakness affected several frameworks, but Java, Node.js, and Drupal had notably higher rates of vulnerabilities in this area than their respective averages. For example Node.js only accounted for 5.7% of vulnerabilities overall, yet accounted for 10% of information exposures. Yet, once again, when it came to weaponization, WordPress once again led the way.

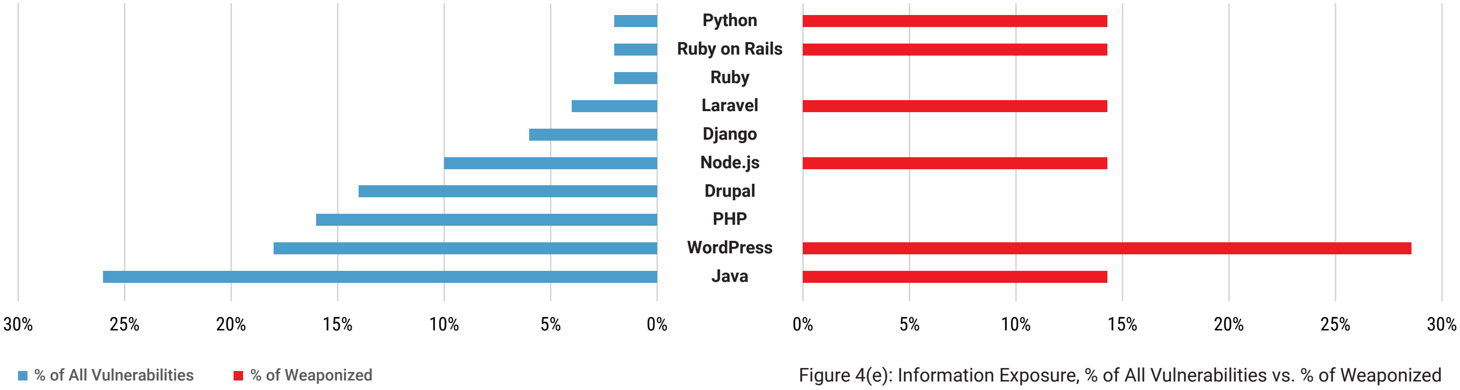

Figure 4(e): Information Exposure, % of All Vulnerabilities vs. % of Weaponized

- **Changes in Injection Attacks:** Overall, the high weaponization rates of injection attacks was one of the trends that remained consistent in both datasets. Command Injection (CWE-77) was weaponized a whopping 80% of the time, and was dominated once again by Apache Struts and WordPress. OS Command Injection (CWE-78) was weaponized 40% of the time and was tied to both PHP and Ruby. However, as a bit of good news, SQL Injection (CWE-89) decreased dramatically with the overall number of vulnerabilities dropping by 74% and the weaponization rate dropping from 26% to 10%.

- **Increase In Access Control Weaknesses:** Improper Access Control (CWE-284) was a notable riser as we looked at the 2015-2019 data. In fact, it was the #1 source of vulnerabilities overall. Weaponization rates remained low at 2.8%, but that was still enough to earn it 3rd place as in terms of weaponization. These issues were notably tied to Java in both cases.

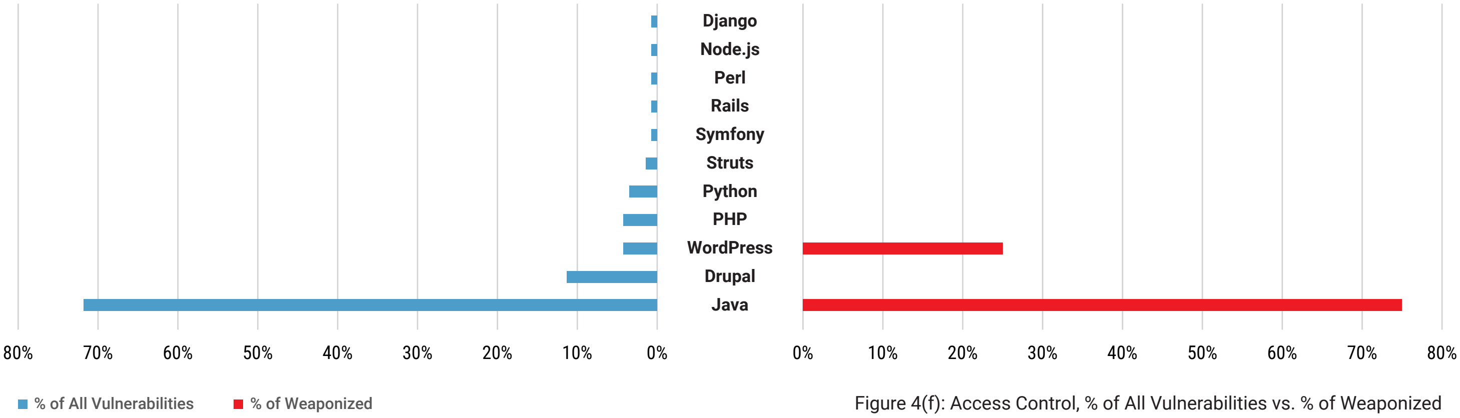

Figure 4(f): Access Control, % of All Vulnerabilities vs. % of Weaponized

4. Analyzing Framework Weaknesses (Continued)

The diagram in Figure 4(g) summarizes the relationship between all CWEs and all languages/frameworks for the 2015 to 2019 time frame.

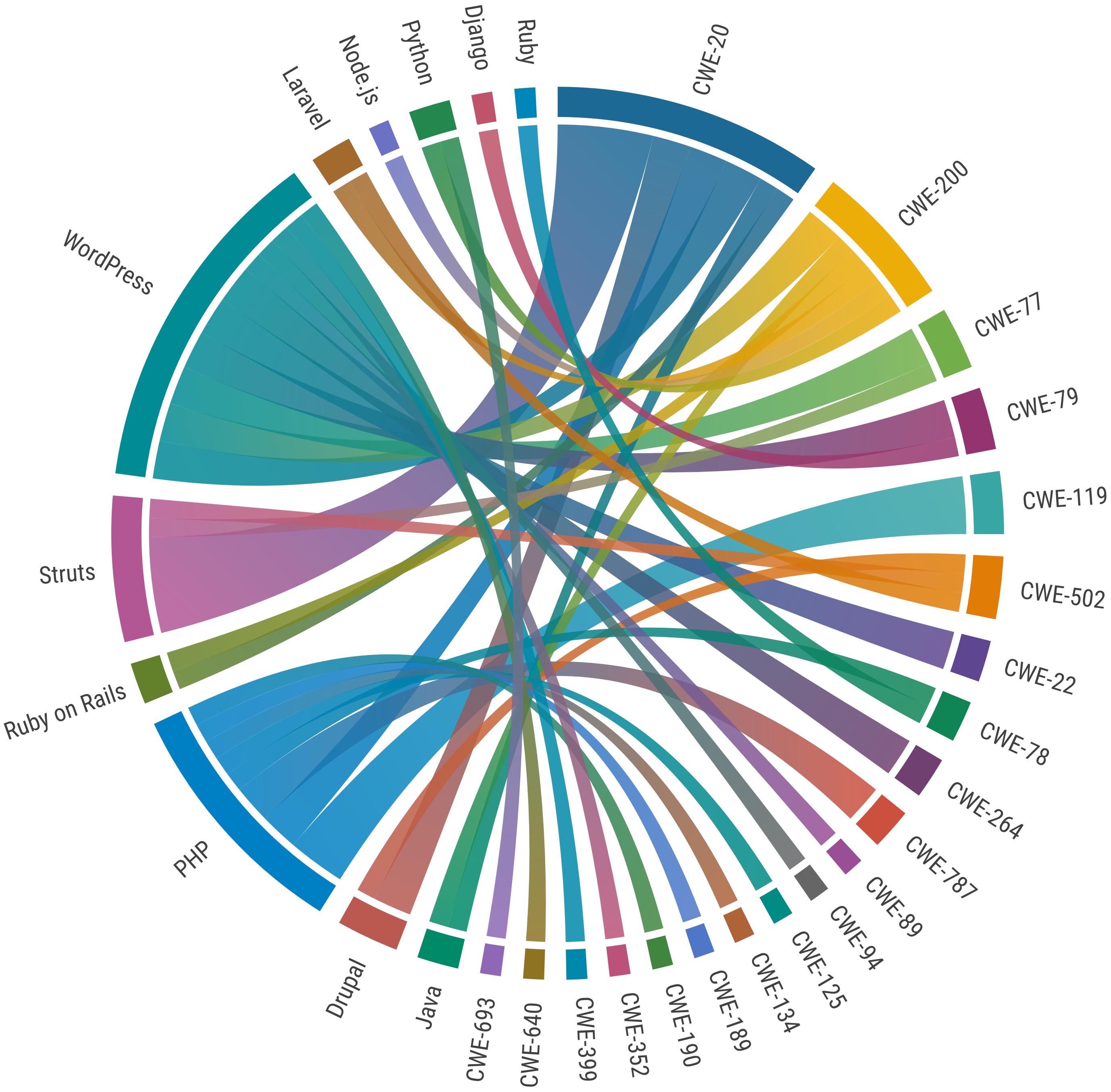

Figure 4(g): Relationship Between All CWEs and Languages/Frameworks, 2015-2019

4. Analyzing Framework Weaknesses (Continued)

Summary of Key Frameworks

The section below summarizes some of the key findings for some of the more notable technologies in the report.

- **WordPress:** WordPress was the most weaponized framework in the study and one of the top sources of vulnerabilities overall. Unfortunately, WordPress was a fairly equal opportunity offender with a variety of underlying weaknesses and weaponization patterns. XSS was by far the most common type of weakness found, however when it came to weaponization it was relatively equal footing with Input Validation, Information Exposure, Command Injection, Access Control, and Path Traversal issues.

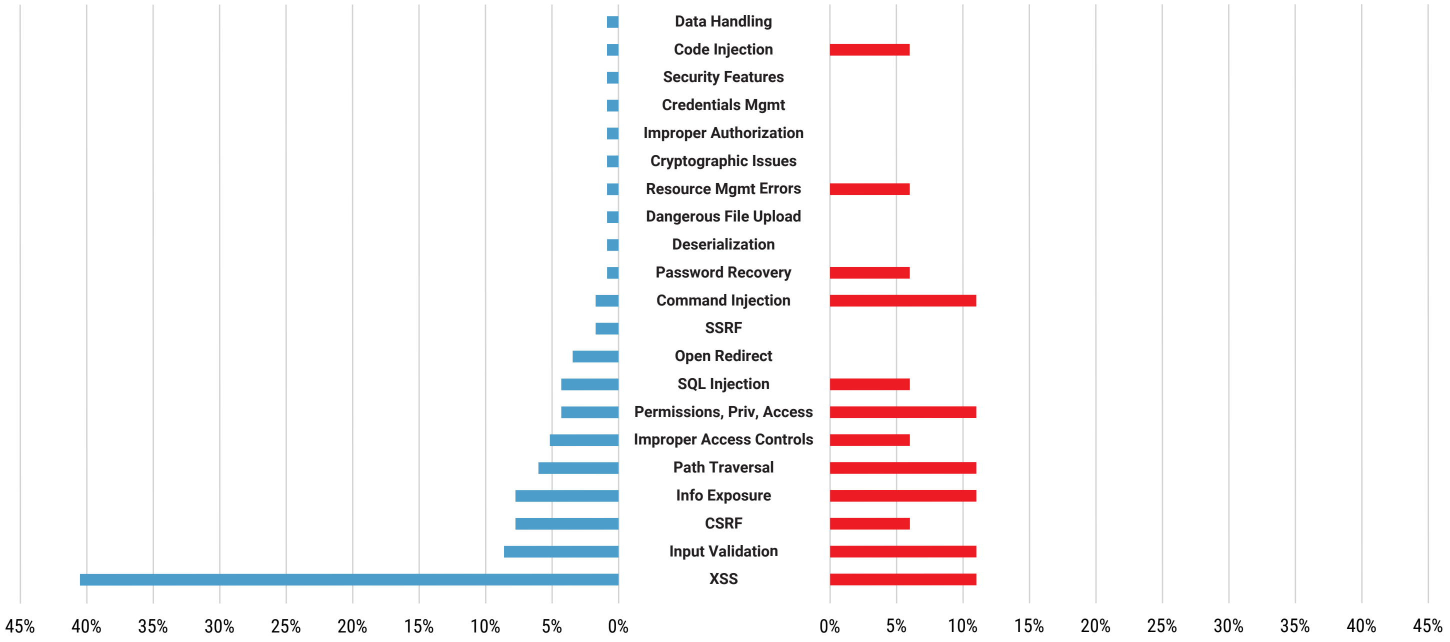

Figure 4(h): WordPress, % of All Vulnerabilities vs. % of Weaponized 2015-2019

- **Apache Struts:** Despite its relatively modest total number of vulnerabilities, Apache Struts was the 3rd most weaponized technology behind WordPress and PHP. In both cases, Improper Input Validation was by far the most common weakness, with deserialization and command injection weaknesses also being weaponized. Notable Struts CVEs include CVE-2017-5638, which was used in the Equifax breach as well as CVE-2018-11776 which was also exploited in the wild. Additional in depth analysis of Apache Struts can be found in the RiskSense's Apache Struts Spotlight Report.

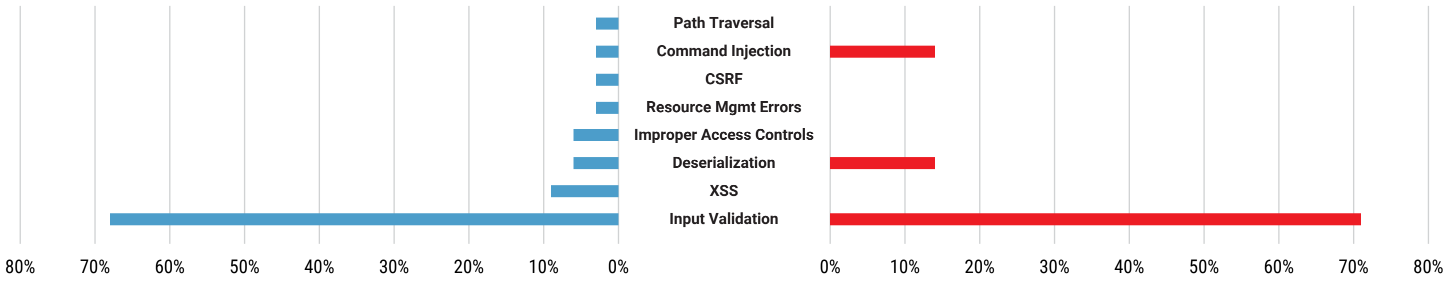

Figure 4(i): Struts, % of All Vulnerabilities vs. % of Weaponized 2015-2019

4. Analyzing Framework Weaknesses (Continued)

- **Node.js:** Node.js has received relatively little coverage in this report, largely due to the fact that only one of its vulnerabilities was weaponized, CVE-2018-5407, also known as the Portsmash vulnerability. However, it had by far the most vulnerabilities of the JavaScript-based frameworks we analyzed with 48 vulnerabilities between 2015 and 2019, and 56 overall. This at least leaves open the potential for greater risk, so developers should be aware of the types of issues that have been found. A wide variety of weaknesses have been observed including Input Validation issues, Information Exposures, and Resource Management Errors. However the framework has also shown a variety of key management issues, cryptography weakness, and susceptibility to resource exhaustion.

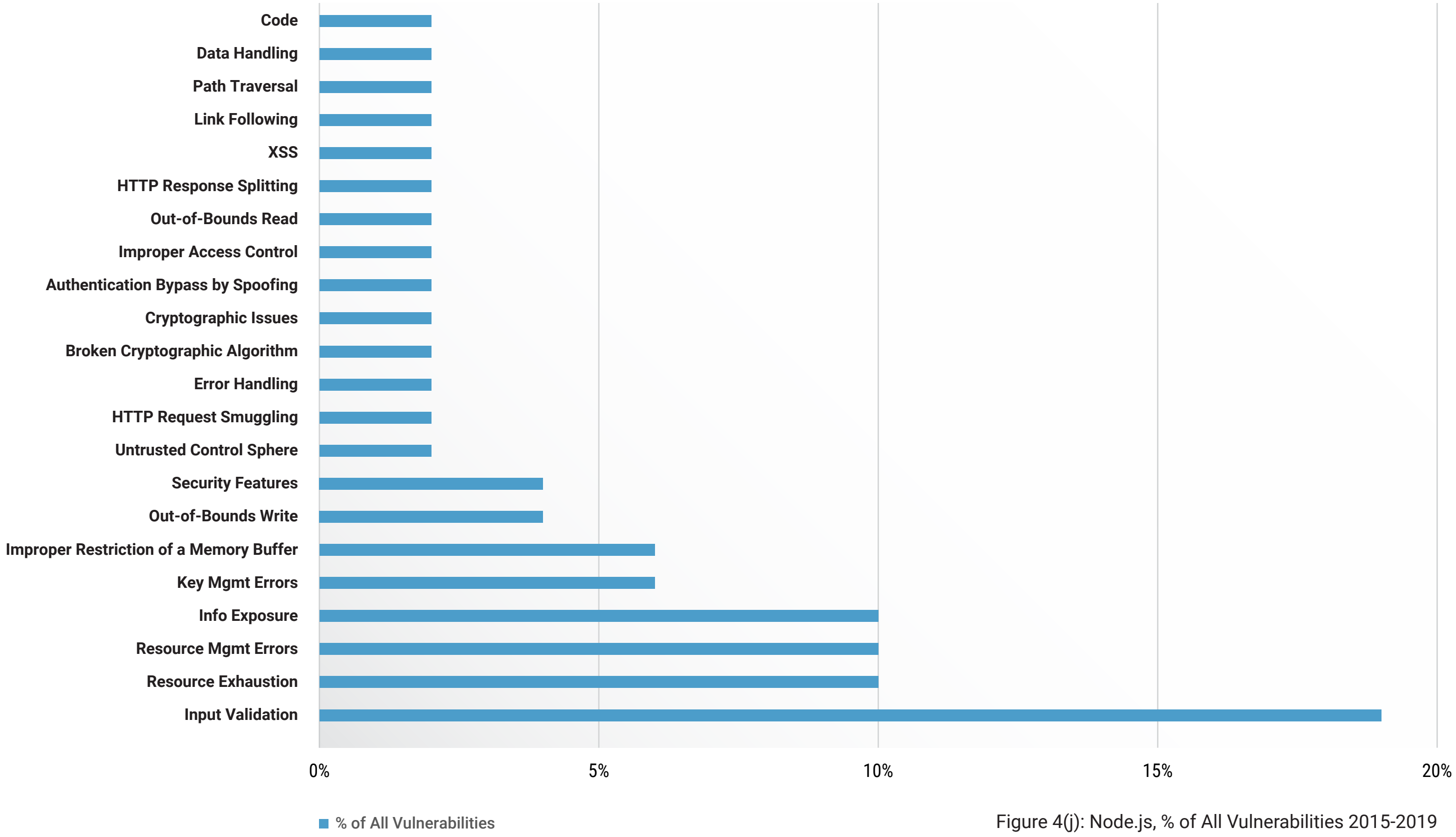

Figure 4(j): Node.js, % of All Vulnerabilities 2015-2019

5. Comparison of Vulnerability Scoring Systems

As new vulnerabilities are identified it is critically important to be able to prioritize them based on their risk. Organizations often rely on vulnerability scores such as the Common Vulnerability Scoring System (CVSS) in order to set these priorities, so we wanted to see how various models performed in terms of reflecting real-world risk.

However, one of the limitations of CVSS scores is that they lack attack context from the real world, such as whether the vulnerability is weaponized, its use in the wild, and so on. So in addition to analyzing CVSS scores we also looked at the vulnerabilities in terms of RiskSense's Vulnerability Risk Rating (VRR) scores. These scores follow the same 0 to 10 scoring found in the CVSS and the same Critical, High, Medium, Low classification breakpoints used in CVSS v3 scores. However, the important difference is that in addition to analyzing the intrinsic attributes of a vulnerability (impact, exploitability, etc.), RiskSense Vulnerability Risk Ratings also incorporate real-world threat context gathered from internal research teams, pen testers, and other threat intelligence sources.

Scoring by Year

Figure 5(a) shows how severity ratings differ between CVSS v2, CVSS v3, and the RiskSense VRR. It is important to note that CVSS v2 uses 3 tiers of severity (Low, Medium, High) while CVSS v3 and RiskSense use a fourth tier (Low, Medium, High, Critical). CVSS v3 was first introduced in 2015, so data was not available for earlier years.

Comparing CVSS v2 to v3 we can see a strong trend of vulnerabilities that are classified as Medium in v2 being upgraded to High or Critical in v3. In fact, in every year well over 50% of framework vulnerabilities were classified as Critical or High in v3. This can make it difficult for organizations to properly prioritize issues when the majority of vulnerabilities are classified in the top tiers. By contrast, RiskSense Vulnerability Risk Ratings are able to use real-world context to hone the High and Critical vulnerabilities to a small and manageable set of CVEs.

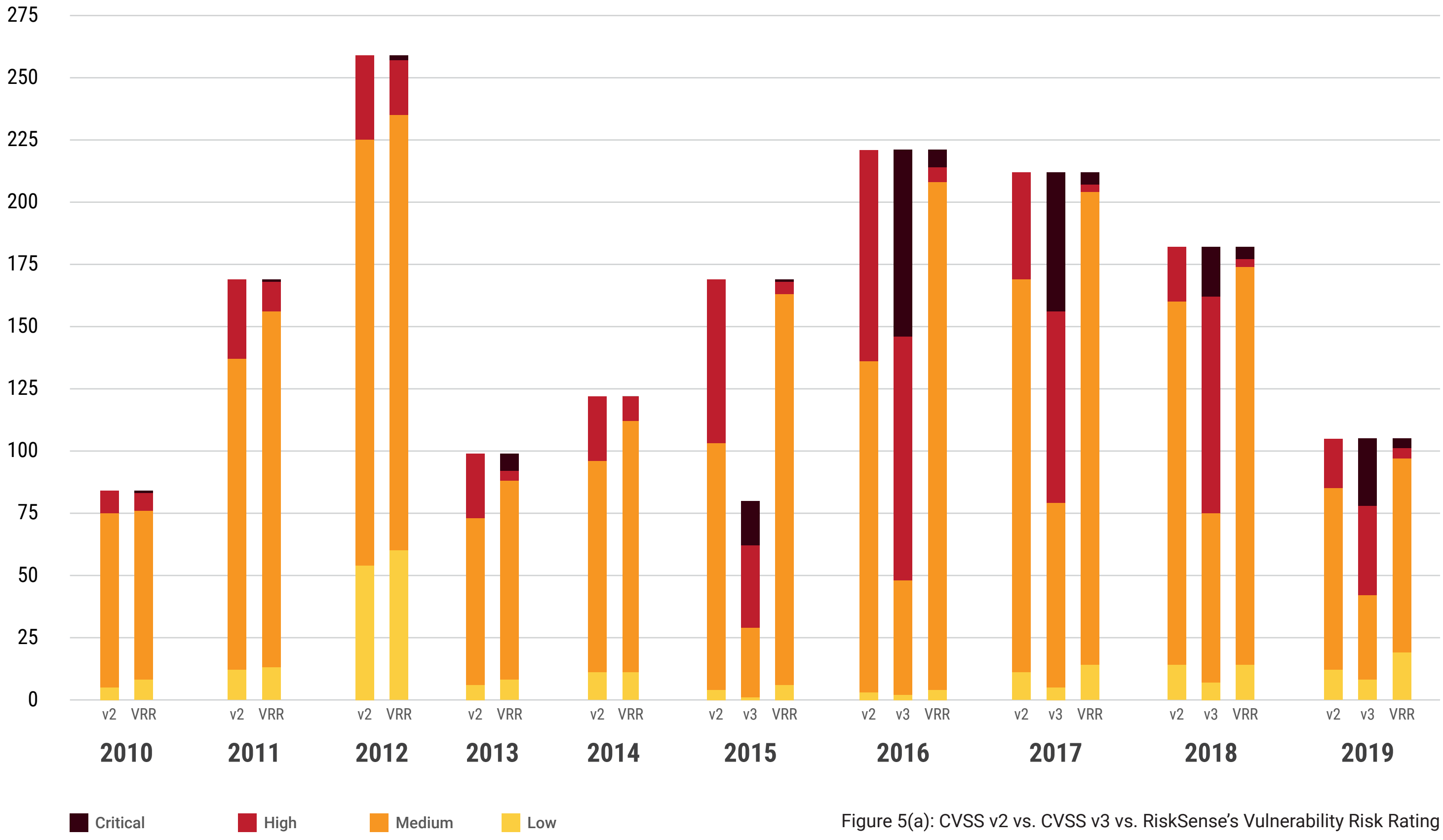

Figure 5(a): CVSS v2 vs. CVSS v3 vs. RiskSense's Vulnerability Risk Rating

5. Comparison of Vulnerability Scoring Systems (Continued)

The Importance of Real-World Context

Next we wanted to see how the scoring models performed in terms of weaponization – specifically, how many weaponized vulnerabilities were in each severity rating and how efficient each category was in terms of weaponized vulnerabilities. Since CVSS v3 scores weren't implemented until mid-2015, we limited our analysis to 2016 to 2019 in order to ensure an equal comparison between the three models. The table in Figure 5(b) shows the results:

Scoring	LOW All / Weaponized	%	MEDIUM All / Weaponized	%	HIGH All / Weaponized	%	CRITICAL All / Weaponized	%	TOTAL All / Weaponized
CVSS v2	40 / **1**	**2.5%**	510 / **28**	**5.5%**	170 / **21**	**12.4%**	0 / **0**	**0%**	720 / **50**
CVSS v3	22 / **1**	**4.5%**	222 / **10**	**4.5%**	298 / **23**	**7.7%**	178 / **16**	**9%**	720 / **50**
RiskSense VRR	51 / **0**	**0%**	632 / **14**	**2.2%**	16 / **16**	**100%**	21 / **20**	**95.2%**	720 / **50**

Figure 5(b): Score Comparison, CVSS v2 vs. CVSS v3 vs. RiskSense's Vulnerability Risk Rating

First, we can see that the RiskSense model was far more efficient in terms of capturing weaponized vulnerabilities in its highest severity ratings. 95.2% of vulnerabilities classified as Critical were weaponized, while 100% of the High severity were weaponized. By comparison, none of the CVSS categories fared better than 12.4%

By nature, CVSS scores don't factor in real-world context such as business criticality or weaponization, so these results were not necessarily unexpected. However, given how heavily organizations rely on these scores, it is important to know their limitations and how they can be complemented by additional context.

On the other hand, it was interesting to see how CVSS v2 compared to v3. CVSS v2 actually fared better than v3 in the highest categories. The High severity of CVSS v2 was actually more efficient at capturing weaponized vulnerabilities (12.4%) than the Critical category of v3 (9%). In other words, staff would have been better off addressing all v2 High vulnerabilities than v3 Criticals. However, v2 lost this advantage in lower severities, with the Medium category having a large number of weaponized vulnerabilities, but a low percentage overall (5.5%). CVSS v2 was the proverbial haystack where most of the weaponized vulnerabilities were hidden.

Conclusion

In this report we have seen how web frameworks have evolved over the past decade, and the impact they can have on the security of an organization's applications. Any weaknesses within a web framework are particularly insidious given that developers often rely on the framework to handle important security-related functions like validating input. For a developer, a vulnerable framework is analogous to giving a mathematician a faulty calculator – the problem can be easy to miss in the moment, but the results can be devastating.

This makes it particularly important for developers and security teams to be aware of how framework decisions can ultimately impact the security of their apps and the organization itself. As with all software, the most recent framework versions are typically the most secure, at least in terms of addressing known vulnerabilities. However, upgrading frameworks can be arduous and potentially risky. Upgrading an underlying framework may cause a given application to break, often leading to a "if it's not broken, don't fix it" mentality. In these cases, it is critical that DevSecOps teams have clear insight into the real-world risk of any vulnerabilities in order to make responsible upgrade decisions and plans.

It also puts an increased priority on selecting the right framework in the first place. As our data shows, specific languages and frameworks have consistently generated high numbers of vulnerabilities and shown high rates of weaponization. WordPress, PHP, Apache Struts, Java, Drupal, and more recently, Ruby on Rails have shown predictable levels of vulnerabilities and weaponization. The popularity of these frameworks make them attractive hunting grounds for attackers, thus vulnerabilities in these areas are more likely to be weaponized.

On the other hand, teams shouldn't conflate a lack of weaponization with safety. Frameworks such as Node.js have thus far shown relatively low weaponization, yet have relatively high numbers of vulnerabilities overall, which carry potential risk for the organization. Ultimately organizations need to have a solid understanding of the overall vulnerability attack surface of their frameworks, the types of specific weaknesses that they contain, and a constantly up to date view into how those weaknesses are being used in the wild.

If you have questions about the data in this report or want to learn more about applying risk-based principles to managing vulnerabilities in your web applications and frameworks, please contact the RiskSense team at info@risksense.com.

About RiskSense

RiskSense®, Inc. provides vulnerability management and prioritization to measure and control cybersecurity risk. The cloud-based RiskSense platform uses a foundation of risk-based scoring, analytics, and technology-accelerated pen testing to identify critical security weaknesses with corresponding remediation action plans, dramatically improving security and IT team efficiency and effectiveness. For more information, visit www.risksense.com or follow us on Twitter at @RiskSense.

RiskSense – the industry's only full spectrum risk-based vulnerability management and prioritization platform.

Contact us today to learn more about RiskSense
RiskSense, Inc. | +1 844.234.RISK | +1 505.217.9422 | risksense.com

CONTACT US | SCHEDULE A DEMO | READ OUR BLOG