The 2021 State of Open Source Vulnerabilities

FOSSA

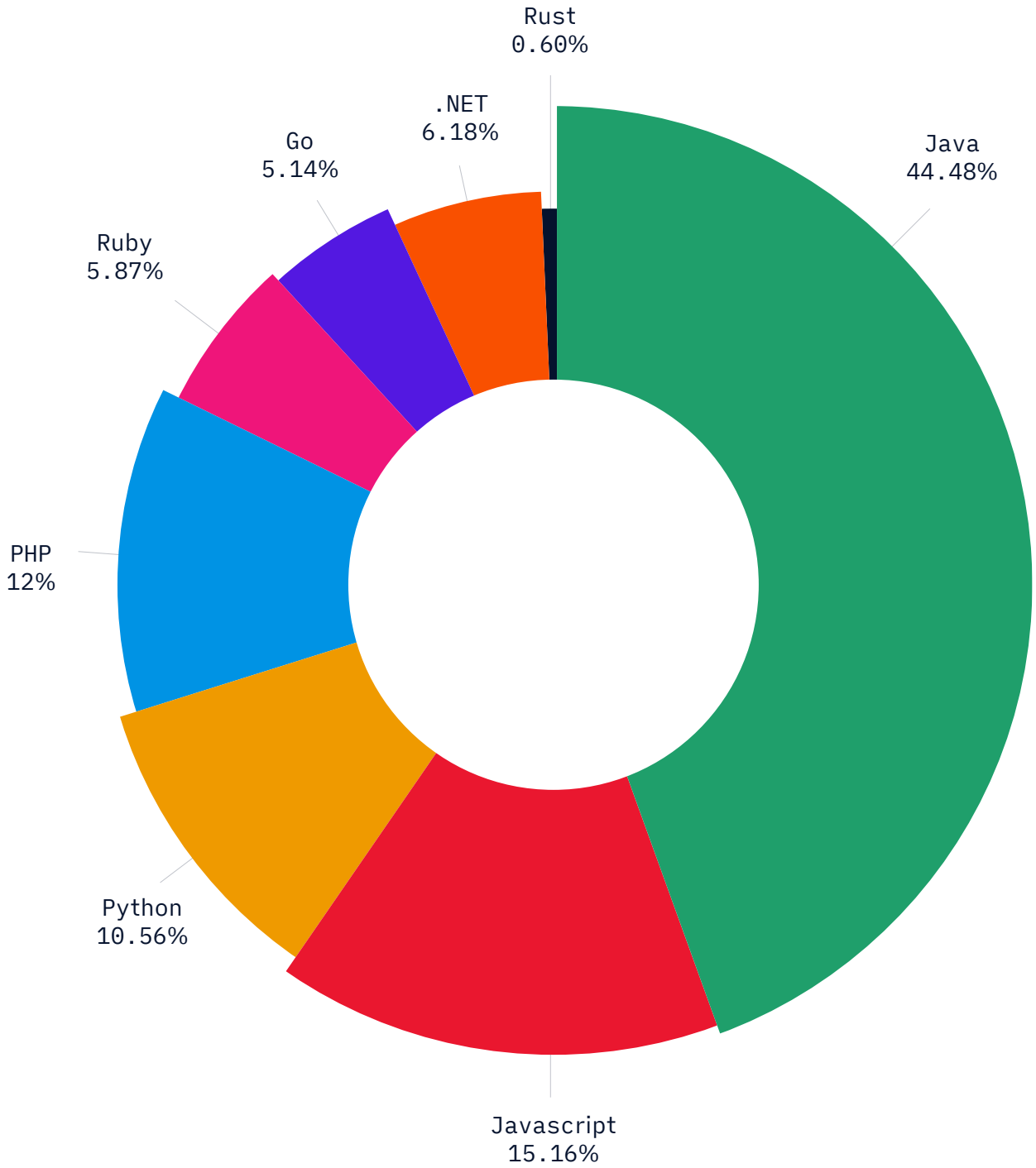
Open source is everywhere. It comprises around 90% of the components of modern applications, and is used by developers across a wide range of industries. Unfortunately, as open source usage has increased, so too have vulnerabilities within open source code.

To better understand the current threat landscape, we recently examined the FOSSA Vulnerability Database — sourced from multiple vulnerability feeds as well as our own research team — to gather insights into trends in open source vulnerabilities.

In this report, we'll take an in-depth look at today's open source vulnerability landscape, including areas like:

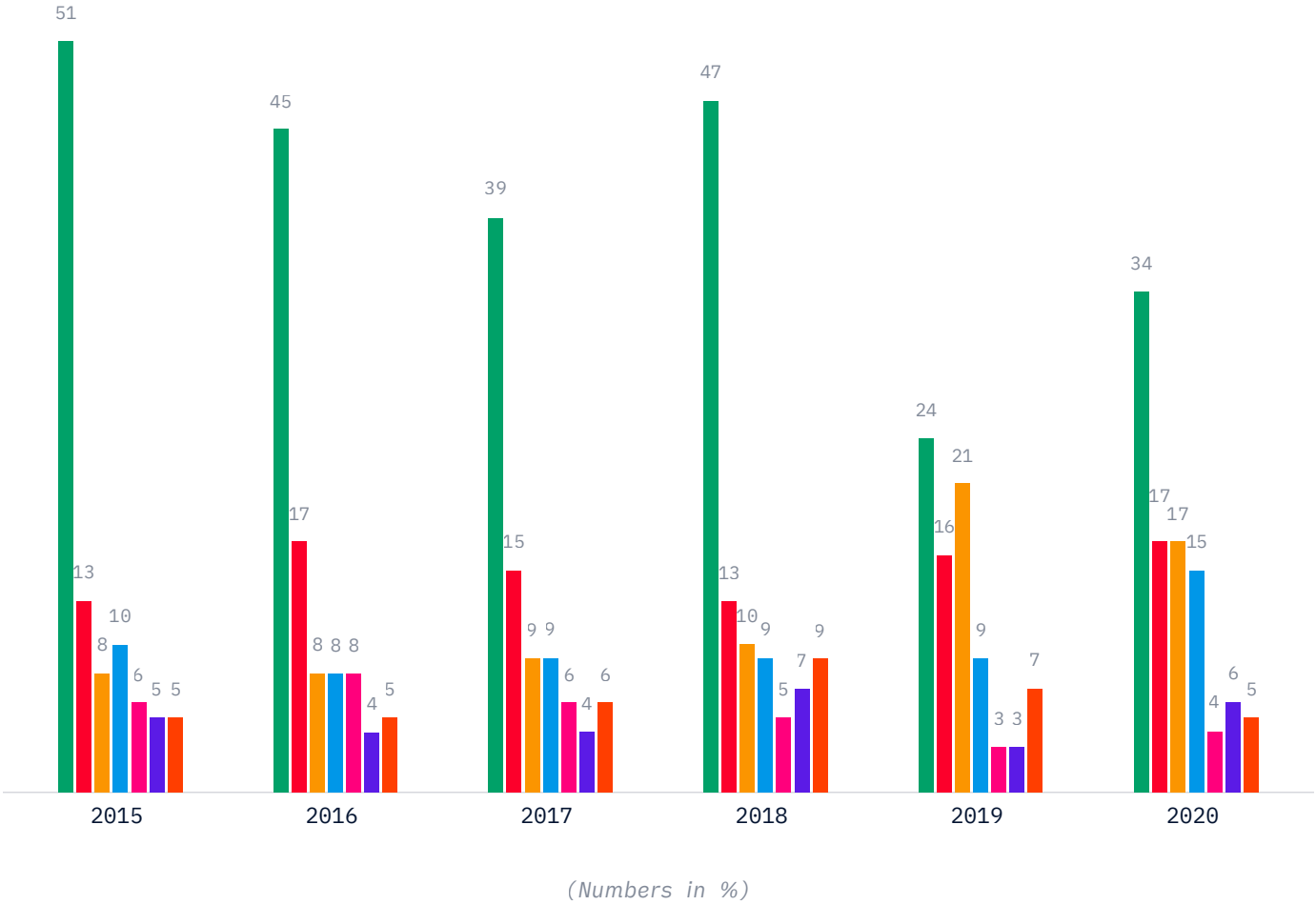
- Common vulnerabilities
- The distribution of vulnerabilities across popular languages
- Longitudinal trends in vulnerabilities over several years
- The most prevalent CWEs in each language
- Libraries with the most vulnerabilities
- Best practices to keep your organization's software free of vulnerabilities

Vulnerabilities Per Language



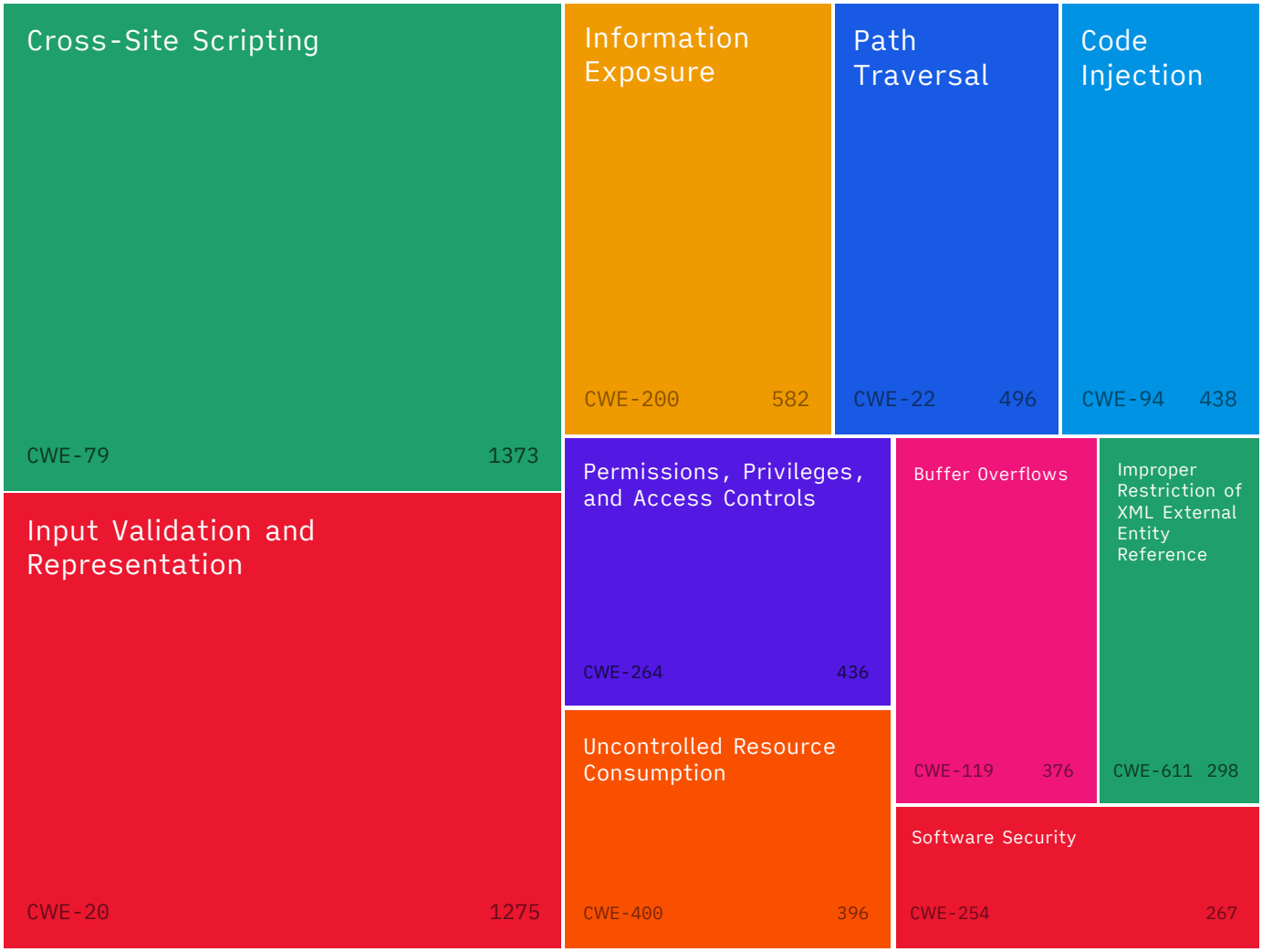
With Java and Javascript being among the most-used languages, it is not a surprise to see them on top of this list. Popular languages tend to attract more security researchers and scrutiny. As newer technologies go mainstream and are adopted more widely, we expect the number of identified vulnerabilities associated with them to grow.

Time Distribution of Vulnerabilities



The yearly distribution of vulnerabilities shows that Java has led the vulnerability count during three of the past five years. In 2019, Python overtook Java as the second-most popular programming language, yet its share of vulnerabilities remains low. In contrast, Go's growing popularity in 2017 and 2018 is reflected in its increase in exploit count.

Top 10 CWE's most commonly found



CWE-79, also known as Cross-Site Scripting (XSS), is one of the most prevalent vulnerabilities in web applications and leads the pack on the most commonly found vulnerabilities list.

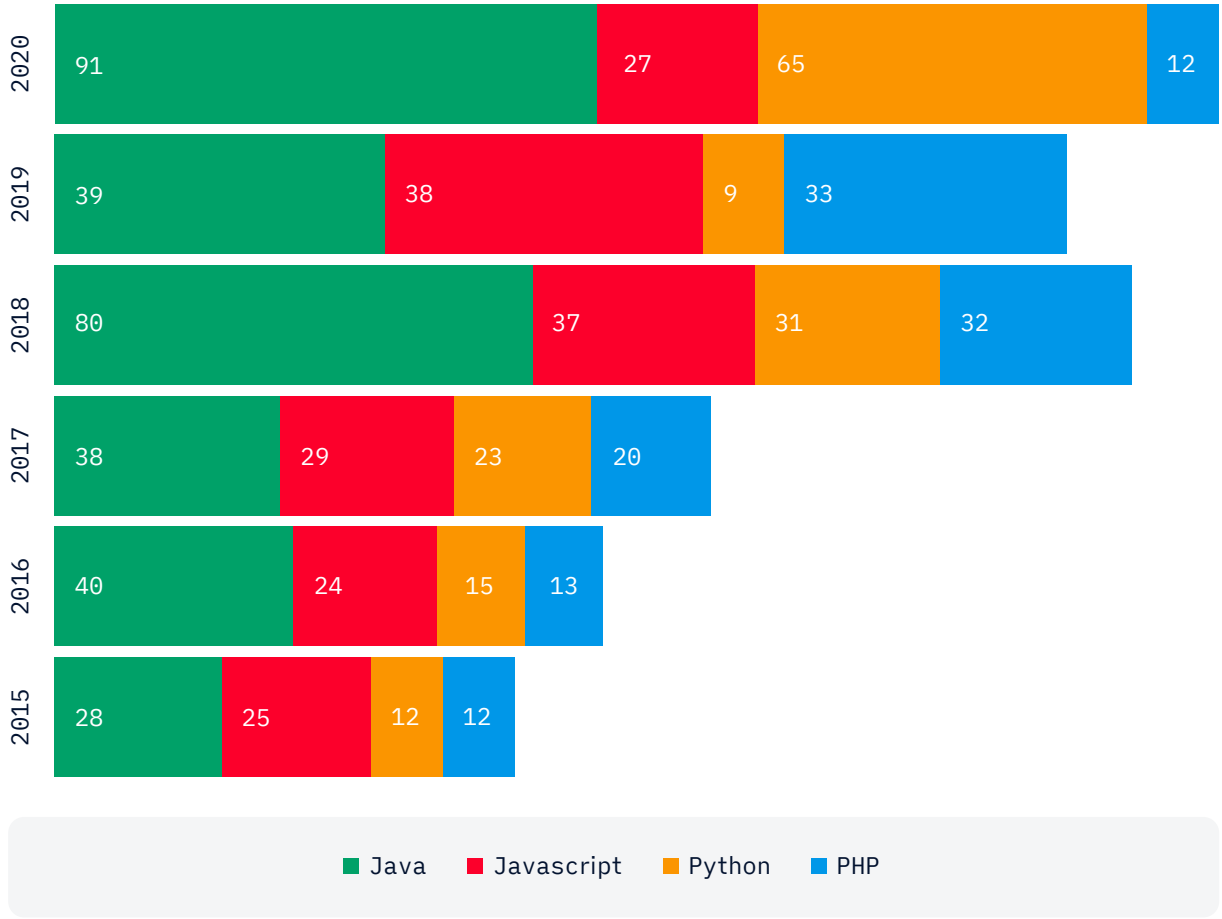
Following CWE-79 is CWE-20, Improper Input Validation. This is a class-level weakness where the product does not validate or incorrectly validates the input. Attackers who

exploit this vulnerability are able to craft an input that is not expected by the rest of the application, leading to unexpected consequences.

Both of these vulnerabilities are related to an input not being validated correctly. Developers should understand the context of all the data and their input sources to avoid vulnerabilities of this kind. Per MITRE:

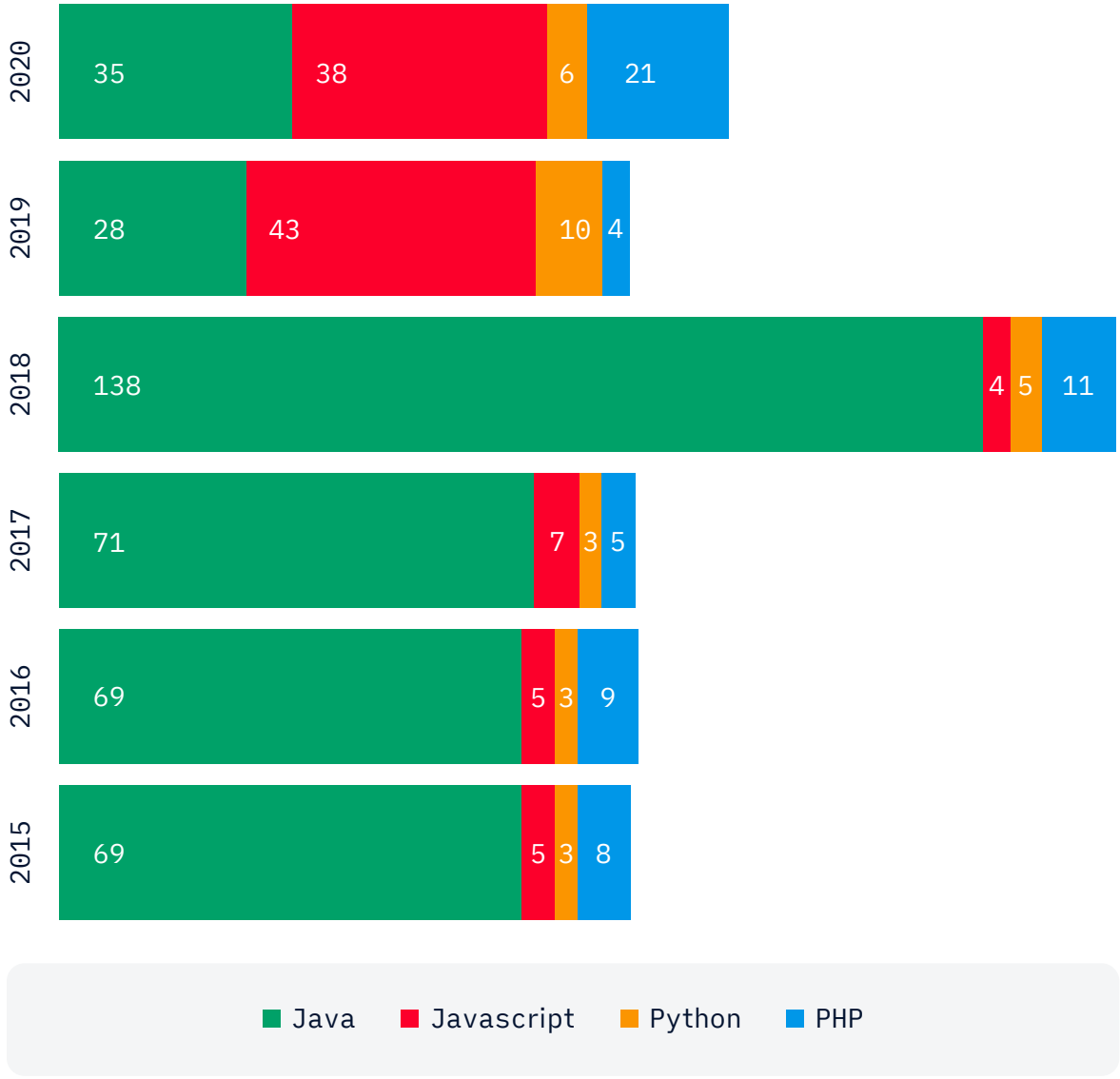
> Assume all inputs are malicious. Use an “accept known good” input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
>
> When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules.
>
> ---
>
> *MITRE, CWE-20: Improper Input Validation. Retrieved January 4, 2020. (https://cwe.mitre.org/data/definitions/20.html)*

Cross-Site Scripting



CWE-79 (Cross-Site Scripting) errors are found in almost all web languages. Their prevalence demonstrates how easy it is to make these kinds of errors and how important it is for developers to be mindful of sanitation and input validation as they build their applications. As a rule, assume all inputs are malicious and only accept inputs that have been verified or sanitized.

Improper Input Validation



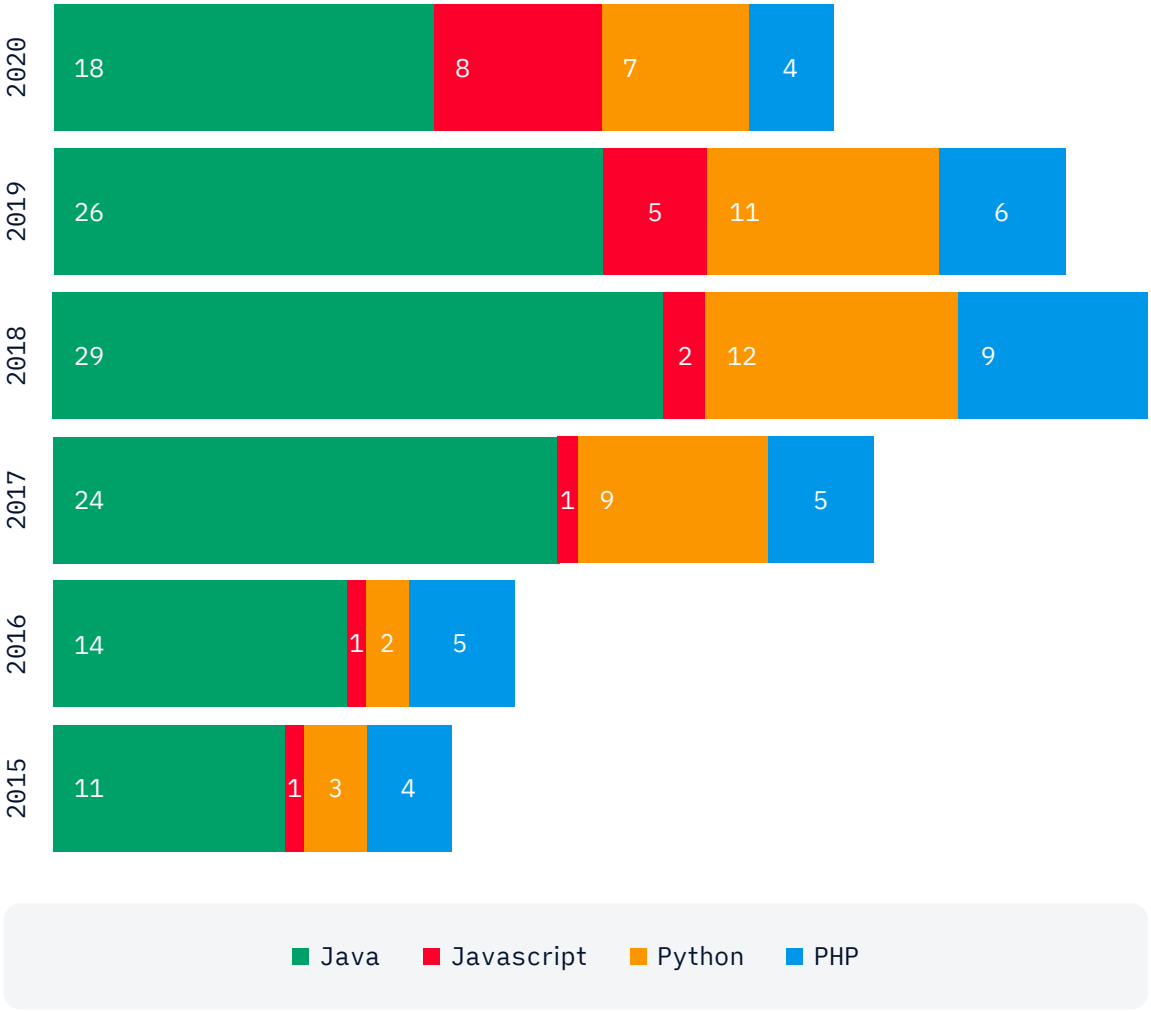
CWE-20, Improper Input Validation, describes unvalidated inputs which alter the expected code output. An example of this would be providing unexpected values to change data flow or access confidential information. CWE-20 is similar to CWE-79 in that it handles inputs, but describes cases where arbitrary code is not executed. Both

vulnerabilities can occur together.

Historically, there have been a disproportionate number of CWE-20 exploits in Java. However, in 2019, there was an increase in the number of CWE-20 vulnerabilities in Javascript and we see that trend continue in 2020 as well. Again, always verify inputs.

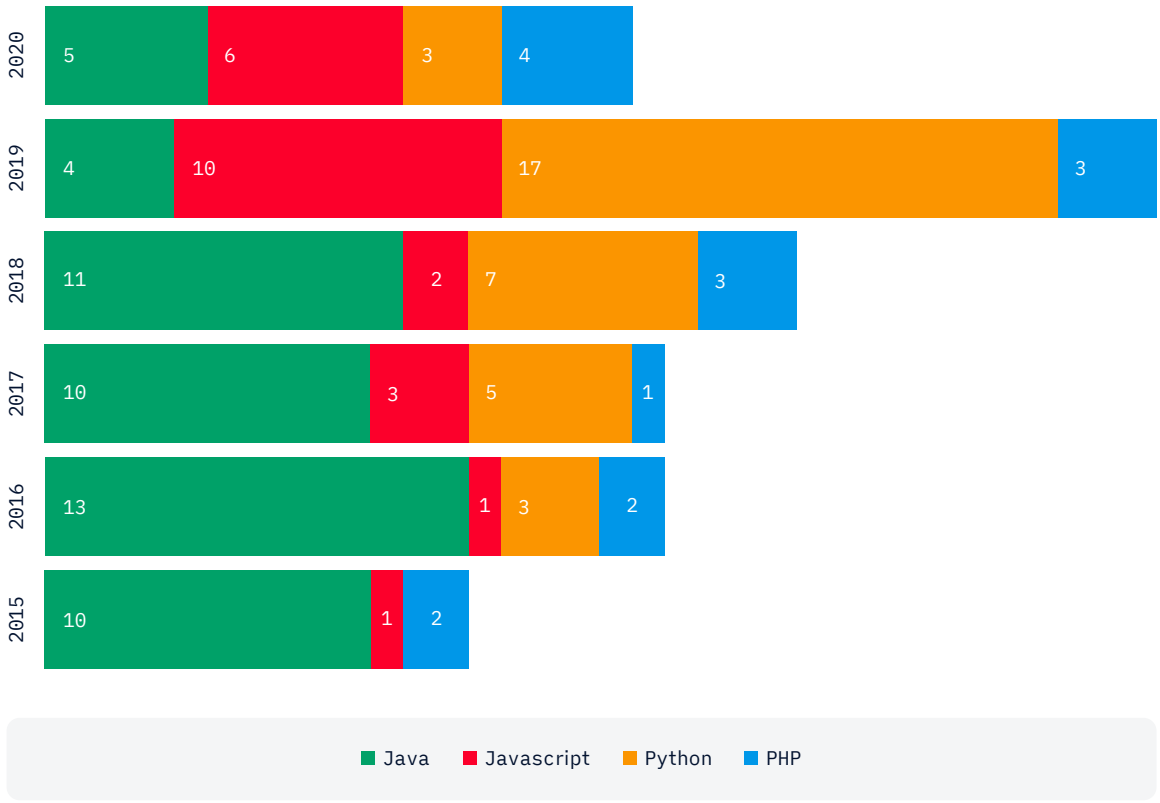
Another common mistake developers make is doing only client-side validation to perform bound-checking. While necessary and important for UI functionality and initial validation, it is not a substitute for server-side validation. As a best practice, input validation must be performed both on the client side and server side.

Information Exposure



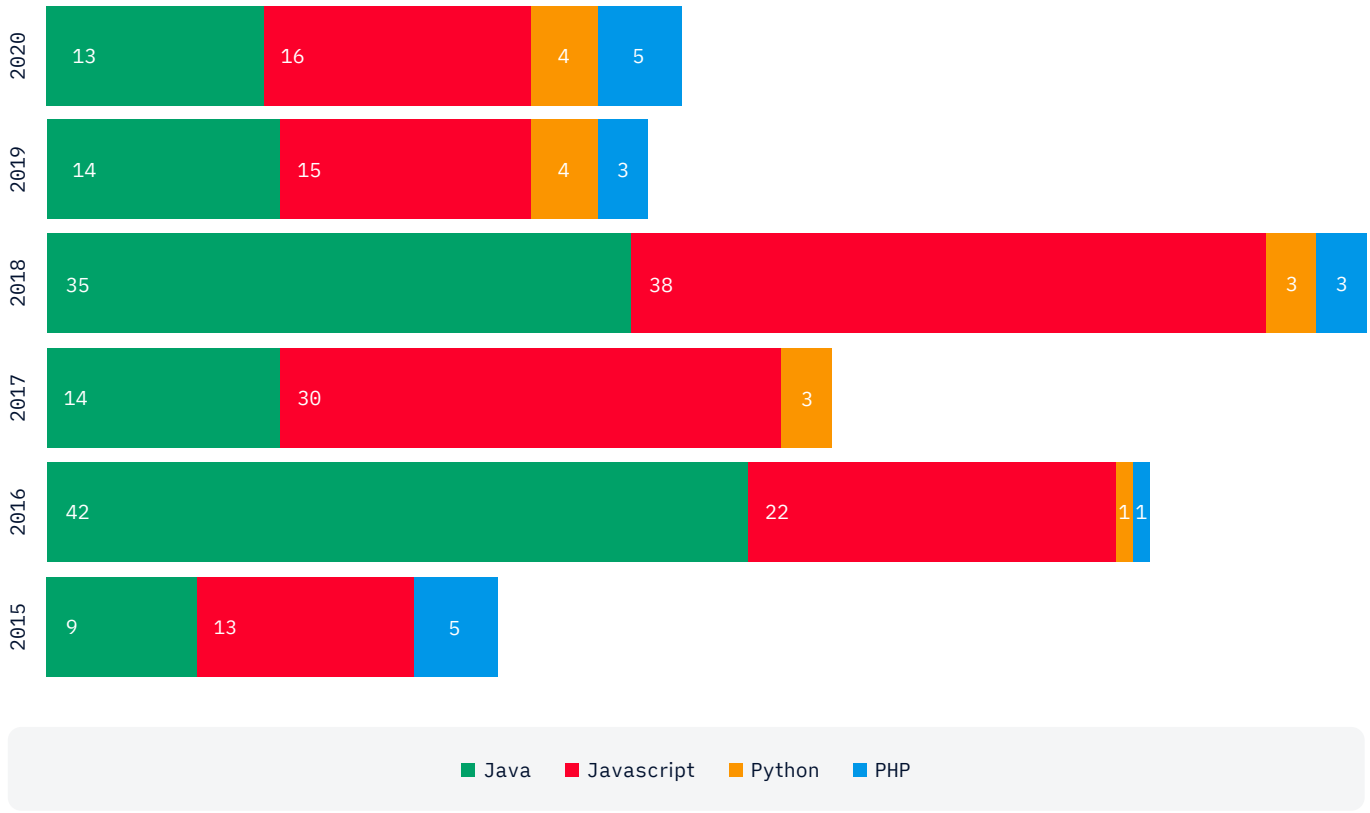
CWE-200, which is related to the unintended information exposure of private or otherwise sensitive information, has been among the most prevalent vulnerabilities in the past few years. Amongst the ecosystems that we have examined, Java followed by php see the most instances of this particular type of vulnerability.

Code Injection



CWE-94, Improper Control of Generation of Code, occurs when software allows a user's input to contain code syntax. This creates a scenario where an attacker can develop code in a manner that will impact the control flow of the software. Similar to other CWEs mentioned, this happens primarily due to unsanitized inputs within an application. This is another reason why it's important that developers always sanitize inputs.

Path Traversal



CWE-22, or Path Traversal errors, are caused by user input being used to construct a pathname for a file or directory potentially leading to files outside the scope of the application. Attackers can potentially view sensitive files like application code and data, credentials, and sensitive OS system files. Javascript sees a higher incidence of path traversal errors than the other languages.

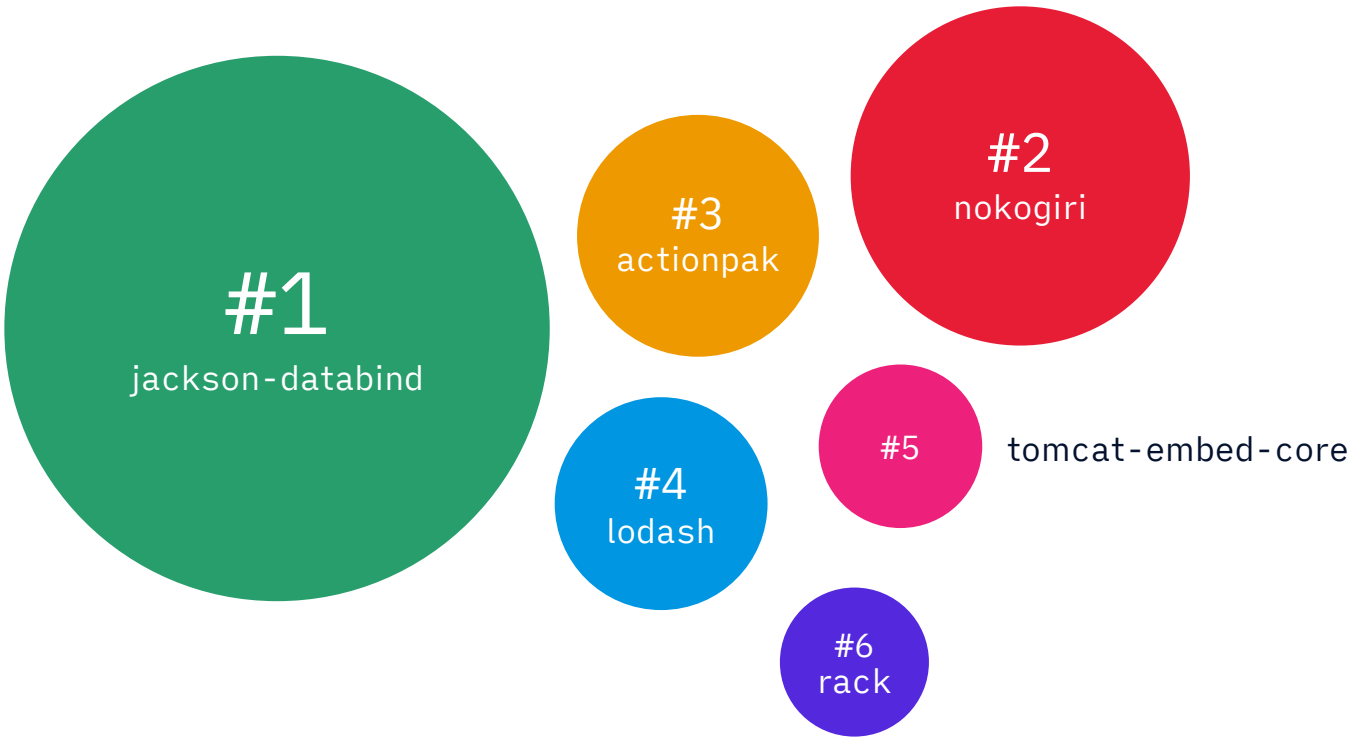
Top 5 by Ecosystem

Java	Javascript	Python	PHP
CWE-20	CWE-79	CWE-79	CWE-79
CWE-79	CWE-22	CWE-20	CWE-89
CWE-611	CWE-20	CWE-200	CWE-200,CWE-94
CWE-264	CWE-400	CWE-264	CWE-352,CWE-20
CWE-200	CWE-74	CWE-399	

As shown, several vulnerabilities are common to all language ecosystems, while a few seem to occur with higher frequency in certain languages. Additionally, the majority of CWEs discussed involve input validation and sanitization issues, further demonstrating how susceptible software is to these types of weaknesses.

Across all ecosystems, CWE-79 (Cross-Site Scripting) leads as the most prevalent weakness, which is understandable considering how dynamic the internet is today. CWE-20 and CWE-200 are the other weaknesses that are seen in high numbers across all languages.

Most Prevalent Vulnerabilities Identified in Our Enterprise Customers



Not surprisingly, libraries that deal with handling user inputs top our list of most vulnerabilities found in enterprise software. Jackson-databind is an extremely popular Java library for parsing JSON and is used in many enterprise applications. Similarly, nokogiri is a Ruby library that parses HTML and XML input. Incorrect usage of inputs or unvalidated inputs are a source of many vulnerabilities, which explains why parsing libraries are often affected. Developers should take proactive measures to prevent these exploits from sneaking into their code and ensure third-party libraries are used safely.

Protecting Against Open Source Vulnerabilities

The open source vulnerability landscape is constantly evolving. Each year brings new libraries, new threats, and new cybersecurity technologies. Our data indicates that in 2019, CWE-79 (Cross-Site Scripting) was the most common CWE, Java the programming language with the most vulnerabilities, and jackson-databind the library with the most vulnerabilities. But future years may be very different.

So while it's impossible to know what, exactly, the threat landscape will look like as we move forward in 2021 and beyond, we do know that following a set of core best practices can go a long way toward helping organizations reduce the security risk in their use of open source:

- Assume all inputs are malicious only accept inputs that have been verified or sanitized.
- Conduct input validation on both the client side and server side.
- Consider adopting SCA tools that integrate directly into the CI/CD pipeline

You can also visit our website for more information on open source vulnerability management.

About FOSSA

Up to 90% of any piece of software is from open source, creating countless dependencies and areas of risk to manage. FOSSA is the most reliable automated policy engine for security management, license compliance, and code quality across the open source stack. With FOSSA, engineering, security, and legal teams all get complete and continuous risk mitigation for the entire software supply chain, integrated into each of their existing workflows. FOSSA enables organizations like Uber, Zendesk, Twitter, Verizon, Fitbit, and UiPath to manage their open source at scale and drive continuous innovation. Learn more at https://fossa.com.