What's so wild about exploits in the wild - and how can we prioritize accordingly?

Written by: Rachel Cheyfitz Shani Gal

November 21, 2019 9 mins read

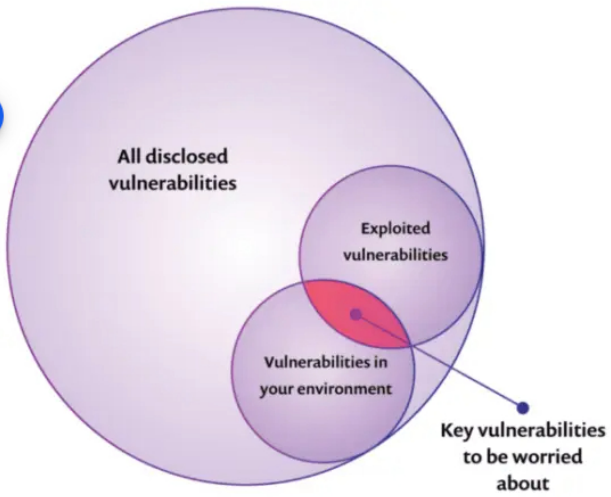
Not all vulnerabilities are created equal. While it's true that you shouldn't ever ignore an open-source vulnerability, it might take some time before some of them can be manipulated and used to hack your system. On the other hand, some known vulnerabilities really do pose a great and immediate risk.

In a world of steadily increasing vulnerabilities, it is challenging to find and attend quickly enough to them all, and we can assume that so many maintainers may be giving up before they're even out of the gate just from sheer frustration in the numbers.

To distinguish and prioritize the different vulnerabilities, we should evaluate the risk posed by each, decide what the minimum risk is that we want to tackle, and then start from there.

One of the main risk factors is how easy it is to hack a specific vulnerability. When someone demonstrates how a vulnerability can be taken advantage of, this is called an exploit. When the exploit is widely published, through sources such as blog posts, forums, exploit-db, or exploitation frameworks like metasploit, it is commonly referred to as an **exploit in the wild**.

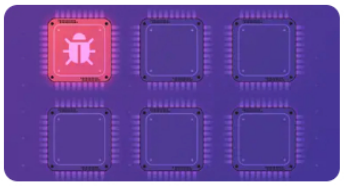
Only a small percentage of known vulnerabilities will be **exploited**, or in other words, used to hack into a system. Vulnerabilities that pose the highest risk are those that have a higher chance of being exploited and therefore should be prioritized and attended to first, as seen in the diagram:



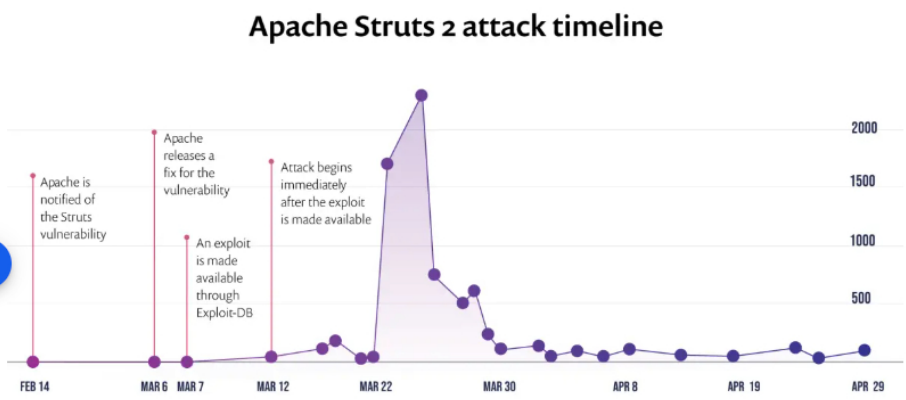
In this post, we detail how exploits in the wild translate into greater risk, how we can evaluate that risk, and how to prioritize and quickly handle your vulnerabilities accordingly.

Apache Struts: an exploit in the wild that led to a real-life hack

You've probably heard of the breach to Equifax a well-known credit-scoring firm that exposed the highly personal data of 145.5 millions of its customers. The breach originated from a known vulnerability in an Apache



Struts package for which there were associated **exploits in the wild published only a few days before the attack**. The availability of the exploit code helped hackers to later breach the Equifax systems, which was attacked two months after the exploit was published — and the results were devastating. See the attacks on Apache Struts in the timeline below:



What is exploit code maturity and how does it influence the risk of a vulnerability?

We refer to the maturity of any exploit code (exploit maturity for short) to measure how practical a vulnerability really is in the real world, relative to:

- Whether the exploit has been published (is it "in the wild"?); and
- The actual "helpfulness" of those published exploits - meaning, whether the vulnerability really can be taken advantage of more easily due to those publications.

In other words, if there is no exploit available at all, it will most likely be more difficult to take advantage of a vulnerability; while at the same time, even if there *is* an exploit available, that *also* does not necessarily mean that the vulnerability can still be easily breached.

Factors that influence the risk posed by a published exploit

For this post, let's discuss two of the factors that influence the maturity of exploit code.

Factor I: how practical is it to exploit this vulnerability?

The first factor is whether there is a gap between the academic theory versus its practical implementation, and how big that gap is. To understand the risks posed by an exploit in the wild, we should evaluate whether it is:

- Practical and can be applied in the real world, or if it's currently only a theory; and
- whether it can be applied to all cases or is limited by certain conditions

An exploit that has only been discussed in theory might present far less risk than a published exploit that has been tested and proven. Likewise, an exploit that is applicable to only 1% of use cases in which the vulnerability appears poses far less risk than an exploit demonstrating how to easily hack the vulnerability regardless of circumstance.

Factor II: What expertise level is required?

The second factor is the level of expertise necessary to actually exploit the vulnerability. Do you need to be an expert hacker to manipulate this vulnerability, or can even beginners implement it? The easier it is to use, the higher the chances are that someone will.

Now that we've put things into perspective, it's easier to understand how a published exploit is typically

correlated with vulnerabilities that are ultimately hacked. It's no wonder that[this study](#) shows that when there is a published exploit available, the vulnerability is **four times more likely to actually be exploited**. Additional studies have even asserted that the publication of an exploit increases the risk by 7 times!!

If we already have CVSS why do we need exploit maturity too?

The Common Vulnerability Scoring System (CVSS score) already weighs a few risk factors in its calculation, including code maturity, which measures if a relevant public exploit code is available. However, as the number of known vulnerabilities increases exponentially over time, the comprehensive scoring system does not always reflect the true risk (or lack thereof) that any given vulnerability may present. In [his blog post earlier this year](#), Liran Tal observed that while the CVSS "vulnerability score is determined by any of a number of recognized parties, the complex system is comprised of over a dozen key characteristics and without proper guidance, experience and supporting information, mistakes are easy to make."

Prioritizing according to exploit maturity is not only right, but also effective

When prioritizing specifically according to exploit maturity we can effectively pinpoint the riskiest vulnerabilities, narrowing the prioritized set down to only about 10% of the total. Amongst Snyk customers, only 4-12% of their vulnerabilities have a mature exploit available, with this number varying according to ecosystem (see the table below). This finding is consistent with additional numbers, such as the stat [found here](#), indicating that 5.5% of published vulnerabilities have exploits which have been observed in the wild.

Read Next

How to prioritize vulnerabilities based on risk

e vulnerabilities out of all vulnerabilities can vary between ecosystems, depending

le below we can see the average data among Snyk customers from sample key

Ecosystem	able vulnerabilities **(aggregated data from Snyk)**
Java Script	19.1%
Java	3.9%
Python	11.6%

Should we fix other vulnerabilities? Well, of course. There's risk in any vulnerability (albeit, some higher than others) and there are many other ways to prioritize - all of which we'll cover in the future. Bottom line though: since every vulnerability poses a risk, we must be vigilant in prioritizing and the best way to get started is by evaluating exploit code maturity.

So how do I know which of my vulnerabilities have exploits in the wild?

In order to support our users and protect them, we are now enabling them to prioritize the detected vulnerabilities in their projects according to exploit maturity!

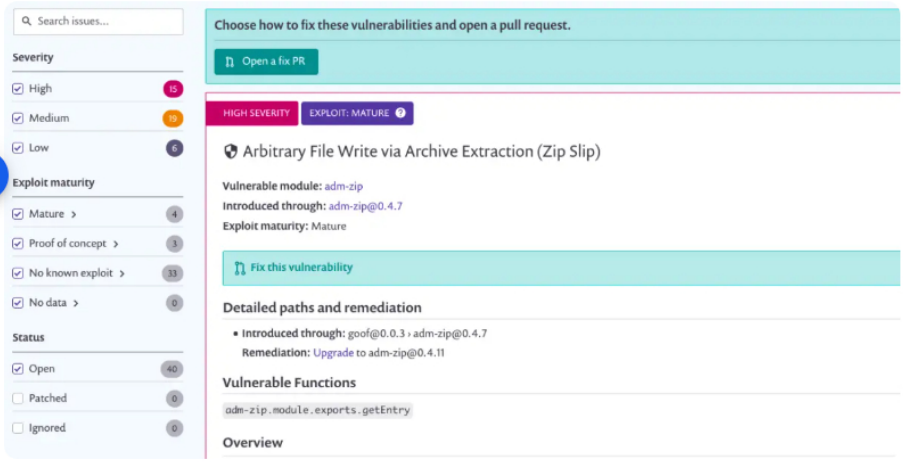
We've decided to use three vectors that we based on our research and on CVSS:

- **Mature:** a published code exploit that can easily be used for this vulnerability is available.
- **Proof of concept:** a published, theoretical proof-of-concept or detailed explanation that demonstrates

how to exploit this vulnerability is available.

- **No known exploit:** neither a proof-of-concept code nor an exploit were found for this vulnerability, or are not publicly available.

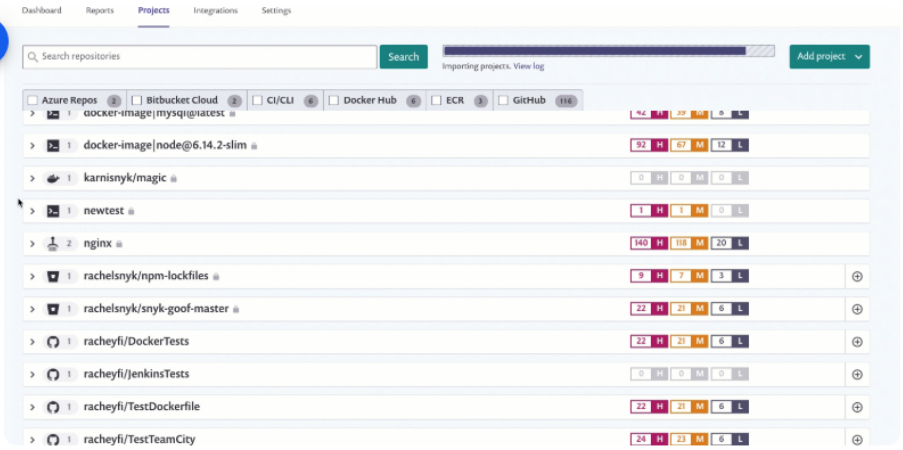
When viewing your project vulnerabilities you can now see if a specific vulnerability is an exploit in the wild. You can also filter and prioritize your scan results and then remediate accordingly, and view an aggregate report. In this way, you can prioritize and attend to the most important and risky vulnerabilities first.



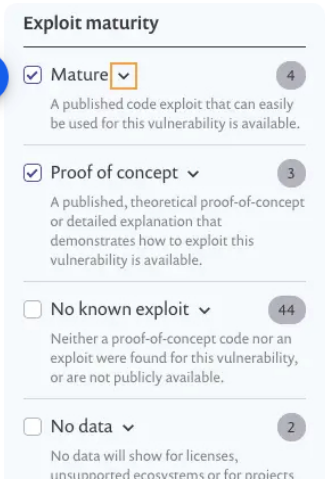
Get started now!

It's easy to get started:

1. Log in to Snyk and go to the detailed**Projects** page for any of your projects:



2. You can see the new filters on the left side:



that require a re-scan. See docs for more details

3. Click **Mature** to view the riskiest, based on exploit maturity, and you're ready to get started remediating.

See our docs for more details.

What's next?

Now that we've released the ability to prioritize according to **exploit maturity**, we'll continue to present additional prioritization methods for vulnerability remediation, supporting our users in the effective protection of their dependencies.

Get started in capture the flag

Learn how to solve capture the flag challenges by watching our virtual 101 workshop on demand.

Watch now

Posted in: Vulnerability Insights