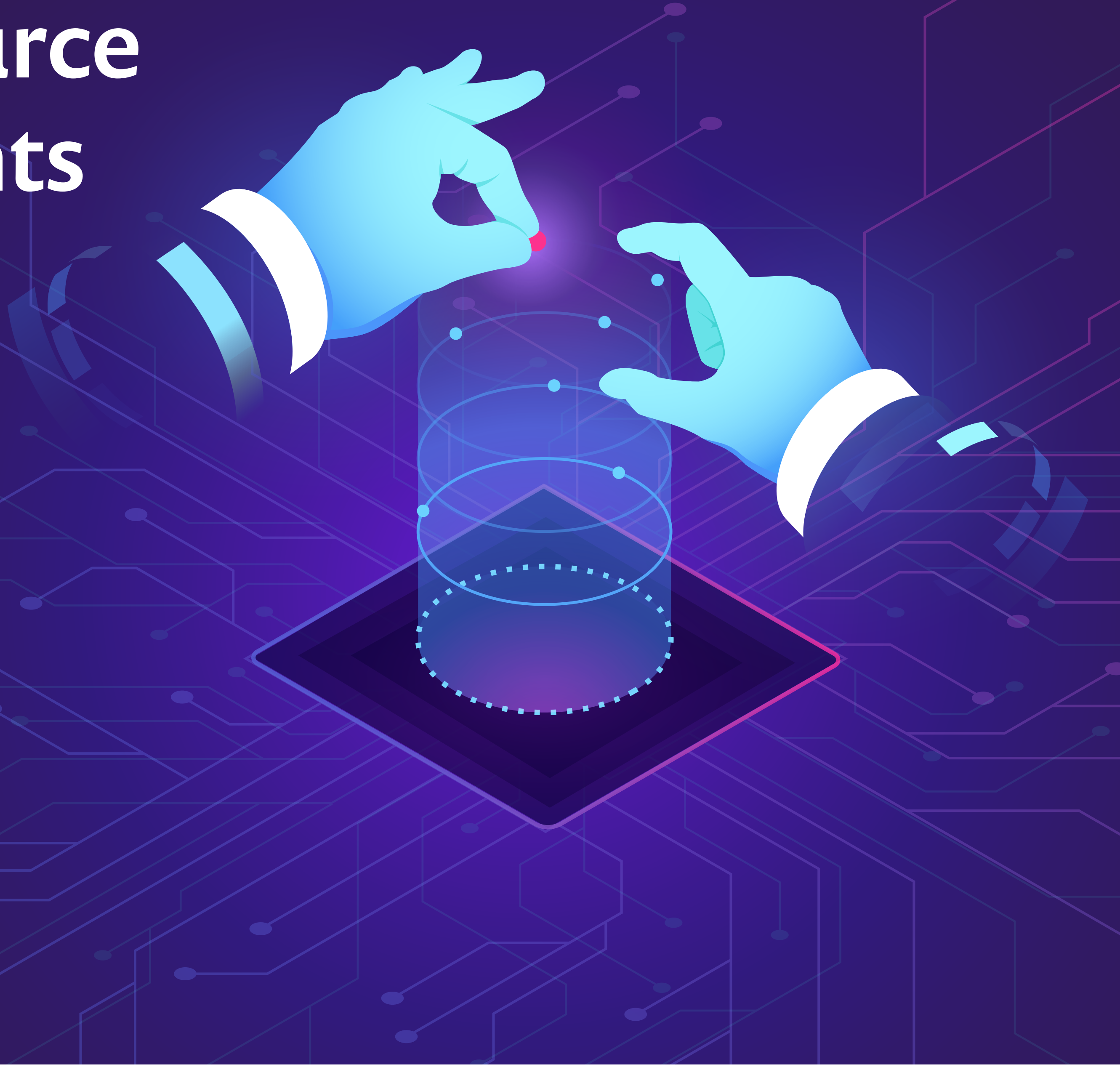
.NET open source security insights



powered by snyk

Table of contents

TL;DR

NuGet stats

- 150,000+ unique packages
- 1.6 million package versions
- 20 billion total downloads
- 26% increase in packages between 2018 and 2019

.NET vulnerability database*

- 100% of vulnerabilities have available remediation
- 71% vulnerabilities rated as high severity
- 2/3 of known vulnerabilities are RCE, XSS, or DoS

Dependency scans*

- 5,744 distinct **direct** dependencies found in .NET projects that we monitor
- 1,819 distinct **indirect** dependencies found in .NET projects that we monitor
- An average project has 11 direct, and 76 indirect dependencies
- An average project has 5 unique high severity vulnerabilities, 2 unique medium severity vulnerabilities, and 1 unique low severity vulnerabilities
- Every project scanned can remediate all vulnerabilities

* Data taken from the Snyk Vulnerability database, and Snyk scan data.

Security takeaways

Bad news

- On average, your apps each have around 8 known vulnerabilities
- The vast majority are high severity

Good news

- You can eliminate **every single** vulnerability by upgrading

.NET security insights

.NET is a growing ecosystem. Whether it is because it has strong industry support from Microsoft, because it has quality developer tools, or because it spans multiple languages and uses, there is no doubt that .NET is holding strong. NuGet is .NET's widely used package manager. It boasts 154,385 unique packages, 1,663,564 package versions, and more than 20 billion package downloads as of the time of this writing. Snyk’s 2019 State of Open Source Security reported a 26% growth in indexed NuGet packages between 2018 and 2019.

You can learn a lot about your .NET project and how to make it more secure by scanning your repository with Snyk. But you may still have questions. You may be wondering how your project compares to others that have been scanned or you may be wondering about trends within the .NET ecosystem as a whole.

This report aims to cover these questions and includes the following:

- The security footprint of a typical .NET project
- The most common vulnerabilities seen in .NET applications, including information about the corresponding libraries
- An examination of the known vulnerabilities on the ecosystem level, including vulnerability types, severity levels, and more

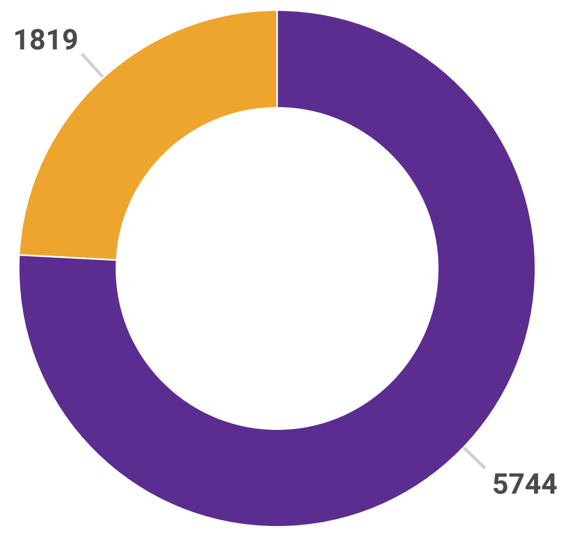
The numbers presented in this report are correct as of the time of writing this report, and are expected to change over time.
They should be viewed in relative terms, and as a starting point of discussion.

The security footprint of a typical .NET project

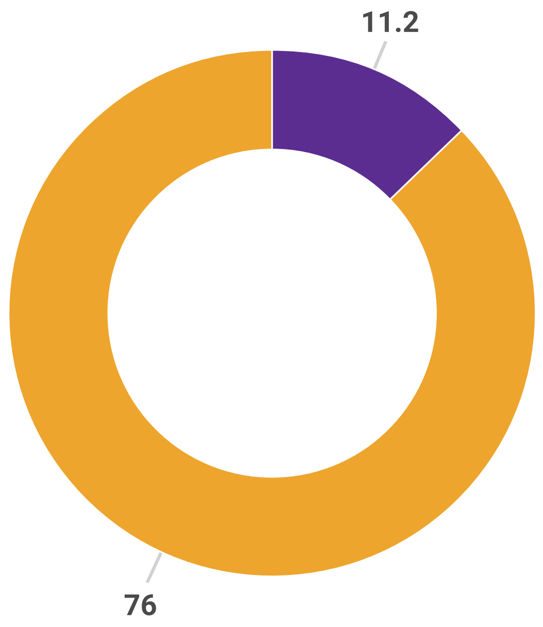
Snyk has already performed thousands of scans for .NET projects since releasing support for the ecosystem in 2017. We can now describe a composite average project, to give our users an idea of what they might find when they try to scan one of their .NET projects.

We have found 5,744 unique direct dependencies and 1,819 indirect dependencies within all of the .NET project scans we have performed. An average project has around 11 direct dependencies and 76 indirect dependencies.

Share of distinct direct vs. distinct indirect dependencies



Average makeup of dependency tree



● Direct dependencies ● Indirect dependencies

Understanding dependencies

Direct dependency

A direct dependency is an open source project that the developer has selected and purposefully installs.

Indirect dependency

An indirect dependency is a dependency of your dependency. Many open source projects utilize other open source projects. When you install your direct dependencies, the indirect dependencies get installed as well. Developers are typically pretty aware of their direct dependencies, but may not be as aware of their indirect dependencies.

Path

A path describes how an open source dependency is introduced to your project. For instance, let's say you have two direct dependencies called *Project A* and *Project B*. Both of these projects introduce indirect dependencies, including *Project C*.

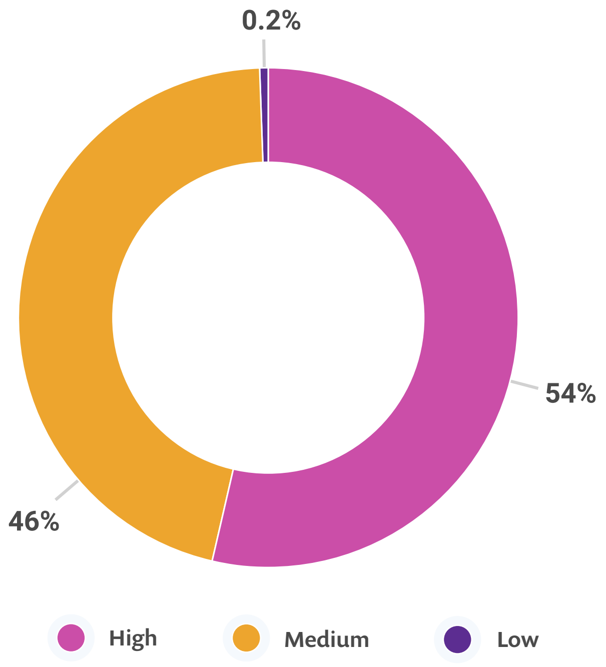
Project C is associated with two different paths, because it is installed by both *Project A* and *Project B*. If *Project C* includes vulnerabilities, a developer must consider both of these paths in order to remediate the vulnerabilities.

The following graphs describe the projects in which we found vulnerabilities. From the scans that have been performed by Snyk, it is clear that for a given project, a vulnerability is likely to be introduced via multiple paths. For instance, you may specify a given package as a direct dependency, but it may show up a second time as an indirect dependency if that same package is also used or referenced by another package within your app. Both cases must be addressed if we want to truly remediate the vulnerability.

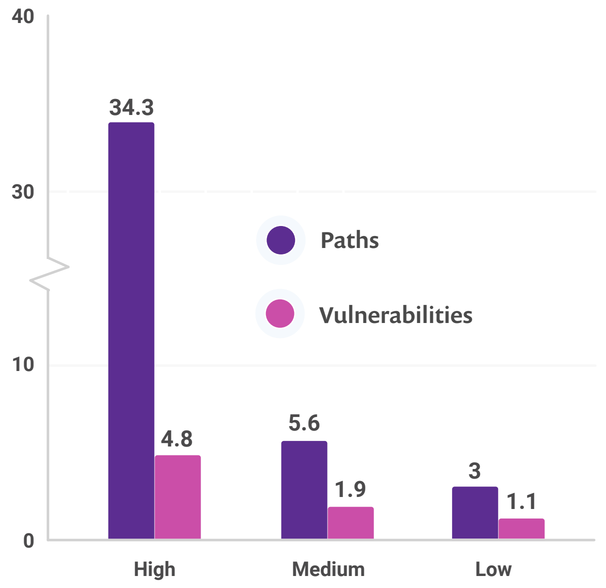
All of the known vulnerabilities found in the .NET ecosystem have available remediation

But the best news is that all of the known vulnerabilities found in the .NET ecosystem have available remediation, meaning that once our users knew of the security vulnerabilities, there were steps they could take to secure their project.

Vulnerability severity breakdown



Vulnerability paths



The most commonly seen vulnerabilities in .NET projects

Now let's look at the top ten libraries that currently are impacting our users most, because they most frequently appear in Snyk project scans.

Let's start by looking at the characteristics of the top 10, and then take a deeper look at three of these libraries that include particularly interesting vulnerabilities. Included in the table are the minimum version upgrades you need to make to move to a vuln-free version.

When reviewing this table, a few things stand out. First, the ASP.NET Core Kestrel cross-platform web server is both popular, and has seen a number of high severity vulnerabilities derived from several different related libraries.

Second, the total number of vulnerabilities for these libraries is generally low, but the severities are generally high. Using the Snyk vulnerability database and data from the NuGet registry, let's dig into the top three of these libraries to learn how they are used, how popular they are, and what known vulnerabilities they contain.

Libraries associated with most commonly occurring vulnerabilities

Place	Library	Use	Vulnerabilites	Minimum known vuln free version	Lifetime Downloads
1	system.net.http	A programming interface for modern HTTP applications	5 ☠ high 1 ⚠ medium	4.3.2	81 million
2	system.io.pipelines	Single producer, single consumer byte buffer management tool	1 ☠ high	4.5.1	8.5 million
3	microsoft.aspnetcore.server.kestrel.core	Core component of ASP.NET Core Kestrel cross-platform web server	2 ☠ high 2 ⚠ medium	2.1.7	22 million
4	system.net.websockets.websocketprotocol	Protocol that enables two-way persistent communication channels over TCP connections	1 ⚠ medium	4.5.3	3.6 million
5	microsoft.data.odata	Data access protocol for the web	1 ☠ high	5.8.4	42.4 million
6	microsoft.aspnetcore.websockets	Web socket middleware for use on top of opaque servers	1 ☠ high 1 ⚠ medium	2.1.7 or 2.2.1	12.5 million
7	system.security.cryptography.xml	Library providing classes to support the creation and validation of XML digital signatures	1 ☠ high	4.4.2	13.6 million
8	microsoft.aspnetcore.server.kestrel.transport.abstractions	Transport abstractions for the ASP.NET Core Kestrel cross-platform web server	1 ☠ high	2.0.3 or 2.1.0-rc1-final	22.1 million
9	system.net.security	Library providing secure network communication between client and server endpoints	3 ☠ high 1 ⚠ medium	4.0.1 or 4.3.1	42.6 million
10	microsoft.aspnetcore.identity	Membership system for building ASP.NET Core web applications	1 ☠ high	2.0.4 or 2.1.2	18.3 million

system.net.http

`system.net.http` provides a programming interface for modern HTTP applications. This includes HTTP client components that allow applications to consume web services over HTTP, and HTTP components that can be used by both clients and servers for parsing HTTP headers.

system.net.http vulnerability breakdown

snyk

Package	Vuln type	Vuln sev	CVSS score	Upgrade available?
system.net.http	Information Exposure	☠ high	7.5	✓
system.net.http	Information Disclosure	☠ high	7.5	✓
system.net.http	Improper Certificate Validation	☠ high	7.5	✓
system.net.http	Denial of Service (DoS)	☠ high	7.5	✓
system.net.http	Privilege Escalation	☠ high	7.3	✓
system.net.http	Authentication Bypass	⚠ medium	5.3	✓

Popularity

`system.net.http` has about 81 million lifetime downloads. The current version (4.3.4) accounts for around 3.4 million downloads. System.http.net averages around 31,000 downloads a day.

Vulnerabilities

There is good and bad news with respect to vulnerabilities in `system.net.http`. First, the good news. The most recent version of this library (4.3.4) has no known vulnerabilities. If you use this library, upgrade to the most recent version!

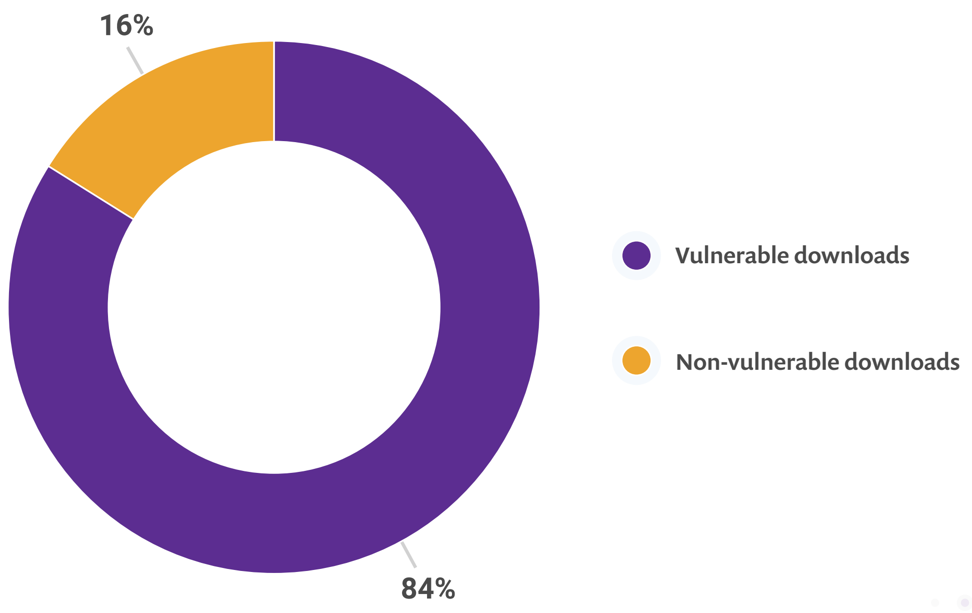
But now for the bad news. This library includes a number of high severity vulnerabilities in its other versions. One of these vulnerabilities is only present in very old versions, but there are four high severity vulnerabilities and one medium severity vulnerability for versions below 4.1.2 and for versions 4.3 - 4.3.2 inclusive.

These vulnerabilities are of multiple types, including information disclosure, improper certificate validation, privilege escalation, authentication bypass, and denial of service. NuGet provides some interesting statistics for this library, including what versions have been downloaded and how often and even includes granular information from the last 6 weeks.

Unfortunately, these statistics show that only 16% of the downloads of this library in the past six weeks were for the most recent version (the only version completely free from known vulnerabilities). That means that over the last six weeks, 84% of package downloads were of versions that include multiple, known, high severity vulnerabilities.

The numbers on NuGet do provide a ray of hope. The most recent version (4.3.4) has been out for 7 months, but it looks like the pace of adoption has been picking up recently. Version 4.3.4 has been downloaded approximately 3.3 million times, but 1.2 million of the downloads occurred in the last six weeks.

Vulnerable system.net.http package downloads



84% of system.net.http package downloads contained multiple high severity vulnerabilities

system.io.pipelines

The `system.io.pipelines` library is a single producer, single consumer byte buffer management tool. In short, this library makes it easier to do high performance I/O (input/output) in .NET.

Popularity

`system.io.pipelines` has been downloaded approximately 8.4 million times, including around 600k downloads of the most recent version (4.5.3). Version 4.6.0 is currently in preview.

Vulnerabilities

Currently, there is only a single known vulnerability associated with the `system.io.pipelines` library. Similar to the previously discussed system.net.http library, there is good and bad news with respect to the vulnerability in `system.io.pipelines`.

First the bad news. The single high severity vulnerability associated with this library is a denial of service vulnerability—which has the capacity to crash your website. The vulnerability in question has a high severity score, and if exploited can prevent legitimate users from accessing your website, run up your server costs, and cause you many headaches.

Now for the good news. The single high severity vulnerability associated with this library is a denial of service vulnerability.

Yes! This is good news as well. Unlike other vulnerabilities, denial of service attacks usually do not aim at breaching security. Despite the headaches that such a vulnerability can cause, it is reassuring that although it is likely to cost you your uptime rate, the vulnerability is not likely to lead to a loss of data or personal information.

Our recommendation? Upgrade system.io.pipelines to version 4.5.1 or higher and keep an eye on the security status of this new and useful library!

system.io.pipelines vulnerability breakdown

snyk

Package	Vuln type	Vuln sev	CVSS score	Upgrade available?
system.io.pipelines	Denial of Service	☠ high	7.5	✓

microsoft.aspnetcore.server.kestrel.core

The `microsoft.aspnetcore.server.kestrel.core` library is the core component of ASP.NET Core Kestrel cross-platform web server. Kestrel is an event-driven, asynchronous I/O-based server and is generally considered to be the favored web server for new asp.net applications.

Popularity

`microsoft.aspnetcore.server.kestrel.core` claims more than 22 million lifetime downloads. Around 630k of these downloads are for the most recent version (2.2.0), which was released in late 2018.

microsoft.aspnetcore.server.kestrel.core vulnerability breakdown

snyk

Package	Vuln type	Vuln sev	CVSS score	Upgrade available?
microsoft.aspnetcore.server.kestrel.core	Denial of Service	high	8.8	✓
microsoft.aspnetcore.server.kestrel.core	Privilege Escalation	high	8.8	✓
microsoft.aspnetcore.server.kestrel.core	Denial of Service	⚠ medium	6.5	✓
microsoft.aspnetcore.server.kestrel.core	Denial of Service	⚠ medium	5.8	✓

Vulnerabilities

This library currently has four known vulnerabilities, including two denial of service vulnerabilities of medium severity and a denial of service and a privilege escalation of high severity. The good news for this library is that the two most recent versions (2.1.7 and 2.2.0) are free from known vulnerabilities.

The bad news with respect to this library is that a related library (`microsoft.aspnetcore.server.kestrel.transport.abstractions`) also appears on the list. If you are using `microsoft.aspnetcore.server.kestrel.core`, you are also likely to be using other libraries with common vulnerabilities.

Additionally, it is important to remember that Kestrel is not a fully featured web server and is often run behind another library. Your choice to run Kestrel independently versus running it behind a library like NGINX or IIS is going to have an impact on how you approach your security.

The remainder of the 20 most popular libraries containing the most commonly seen are described in the following table.

Now let's look at vulnerability types, severities, and remediations related to these vulnerabilities.

More of the most commonly seen vulnerabilities

snyk

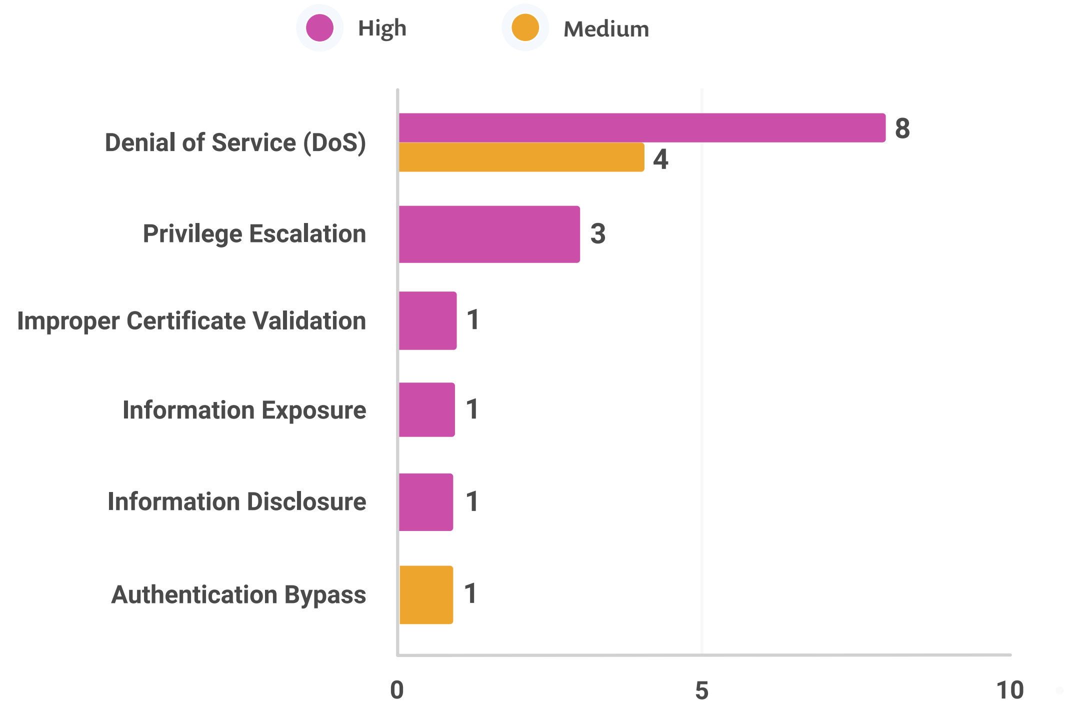
Package	Vuln type	Vuln sev	CVSS score	Upgrade available?
system.net.websockets .websocketprotocol	Denial of Service	⚠ medium	5.9	✓
microsoft.data.odata	Denial of Service	☠ high	7.5	✓
microsoft.aspnetcore. websockets	Denial of Service	☠ high	7.5	✓
microsoft.aspnetcore. websockets	Denial of Service	⚠ medium	5.9	✓
system.security.cryptography. xml	Denial of Service	☠ high	7.5	✓
microsoft.aspnetcore.server. kestrel.transport.abstractions	Denial of Service	☠ high	8.8	✓
system.net.security	Improper Certificate Validation	☠ high	7.5	✓
system.net.security	Denial of Service	☠ high	7.5	✓
system.net.security	Privilege Escalation	☠ high	7.3	✓

The most commonly found .NET vulnerability types

Within the most commonly seen vulnerabilities, we see a variety of vulnerability types. More than half of the vulnerabilities in the top ten however, are denial of service vulnerabilities. Denial of service (DoS) describes a family of attacks, all aimed at making a system inaccessible to its intended and legitimate users.

This has interesting security implications because unlike other vulnerabilities, DoS attacks usually do not aim at breaching security. Rather, they are focused on making websites and services unavailable to genuine users resulting in downtime.

Most common vulnerabilities by type



One popular denial of service vulnerability is a DDoS (distributed denial of service), an attack that attempts to clog network pipes to the system by generating a large volume of traffic from many machines.

When it comes to open source libraries, DoS vulnerabilities allow attackers to trigger such a crash or crippling of the service by using a flaw either in the application code or from the use of open source libraries.

Two common types of DoS vulnerabilities are:

- High CPU/Memory Consumption- An attacker sending crafted requests that could cause the system to take a disproportionate amount of time to process.
- Crash - An attacker sending crafted requests that could cause the system to crash.

Although no one wants a costly denial of service attack and the bad press, downtime, and lost revenue potentially associated with it, a breach of security that surfaces sensitive information could be much worse.

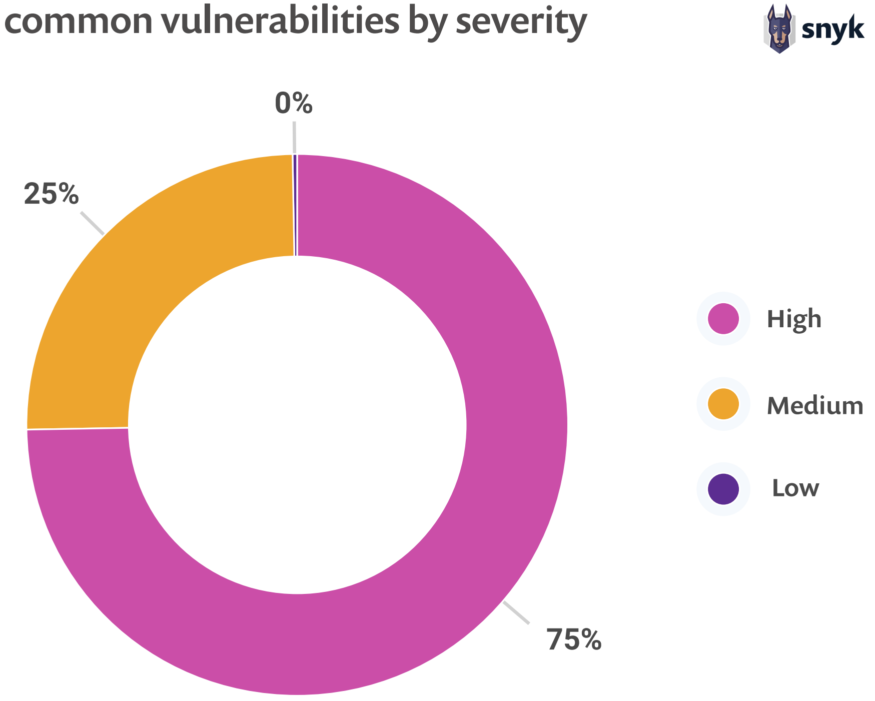
Vulnerability severity

The bad news that comes from this data is that three quarters of the most commonly seen vulnerabilities have a high severity rating. Medium severity vulnerabilities account for the remaining quarter of the top twenty vulnerabilities. We saw no low severity vulnerabilities. CVSS scores ranged from 5.3 through 8.8, with a median of 7.5.

A general trend we have seen in the .NET ecosystem is that projects had tended not to have high numbers of vulnerabilities. This may lead people to think that they can put security on the back burner, but the statistics on severity ratings show why this is faulty thinking. Known vulnerabilities may not be as common within the .NET ecosystem, but the ones that are present can cause serious disruptions and damage if not addressed.

Every single vulnerability within Snyk's top 20 had available remediation

Most common vulnerabilities by severity



Every vulnerability can be fixed

The majority of the vulnerabilities in the top twenty may be rated as high severity, but there is good news as well. Every single vulnerability within Snyk's top 20 had available remediation. This is excellent news. It is good to know about a security vulnerability, even if remediation is not available, but nothing beats actionable advice that leads to a more secure project.

What can Snyk tell us about security in the .NET ecosystem?

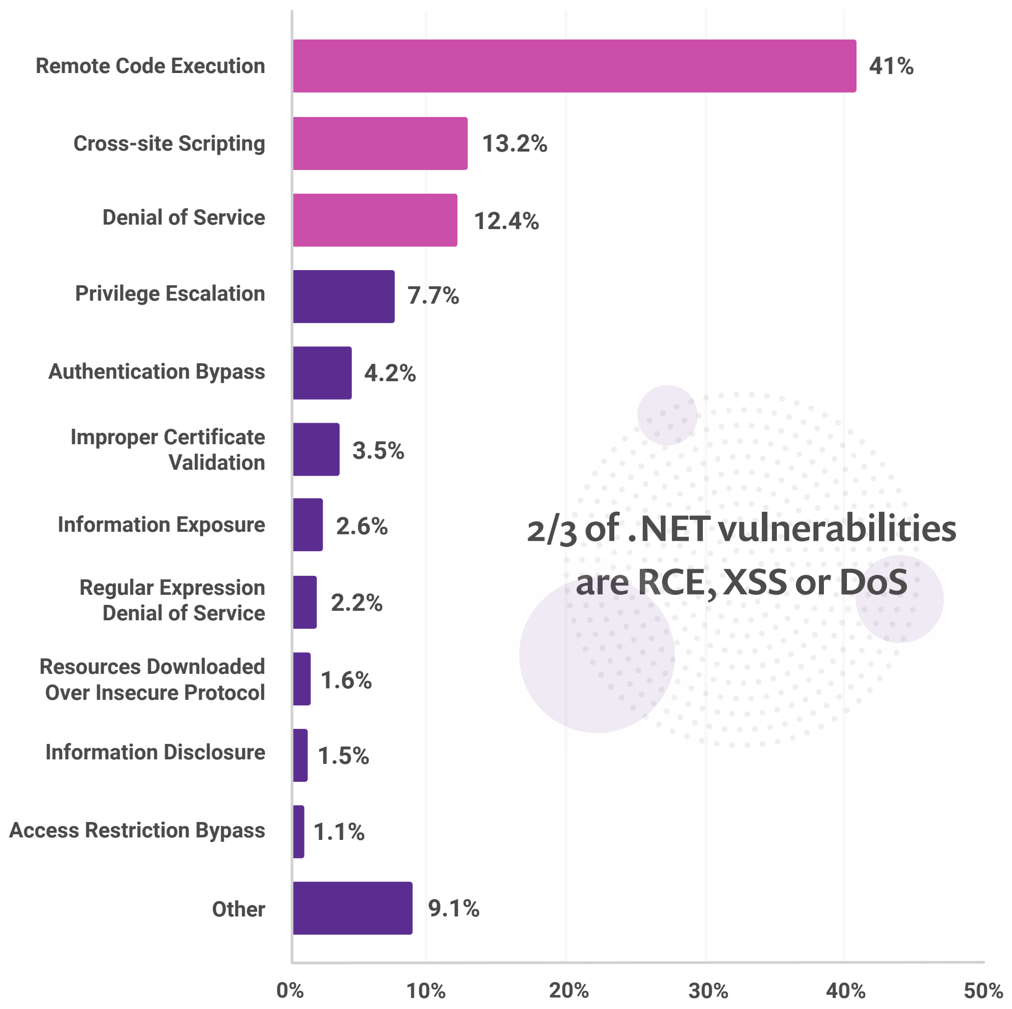
Now that we have considered the most commonly seen vulnerabilities, let's take a higher level look at the .NET ecosystem. What types of vulnerabilities are common within the ecosystem? Does the trend of high severity vulnerabilities hold true when we look at the ecosystem as a whole?

Vulnerability types

The following table breaks down the types of vulnerabilities present, for all vulnerabilities with one or more percent share across the .NET ecosystem. It includes the number of distinct vulnerabilities of a given type found in Snyk's vulnerability database and its percent share.

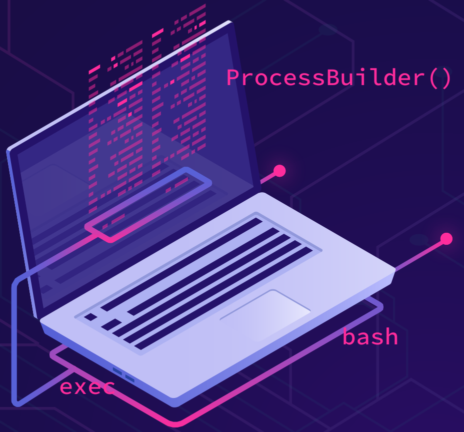
Despite the large variety, just three vulnerability types make up the majority of the vulnerabilities found. Remote code execution (RCE), cross-site scripting (XSS), and denial of service (DoS) vulnerabilities account for 2/3 of .NET vulnerabilities found in Snyk's vulnerability database.

Vulnerability types at the ecosystem level



Vulnerability spotlight

This section is a handy review for anyone wanting more information on the top three vulnerability types in the .NET ecosystem.



Cross-site scripting

A cross-site scripting attack occurs when the attacker tricks a legitimate web-based application or site to accept a request as originating from a trusted source. This is done by escaping the context of the web application; the web application then delivers that data to its users along with other trusted dynamic content, without validating it.

Remote code execution

Remote code execution describes a type of vulnerability that occurs when an attacker is able to execute arbitrary commands or code in your application. The code execution can happen over a network and is therefore not tied to any specific geography.

Denial of service

Denial of service describes a family of attacks, all aimed at making a system inaccessible to its intended and legitimate users. Unlike other vulnerabilities, DoS attacks usually do not aim at breaching security. Rather, they are focused on making websites and services unavailable to genuine users resulting in downtime.

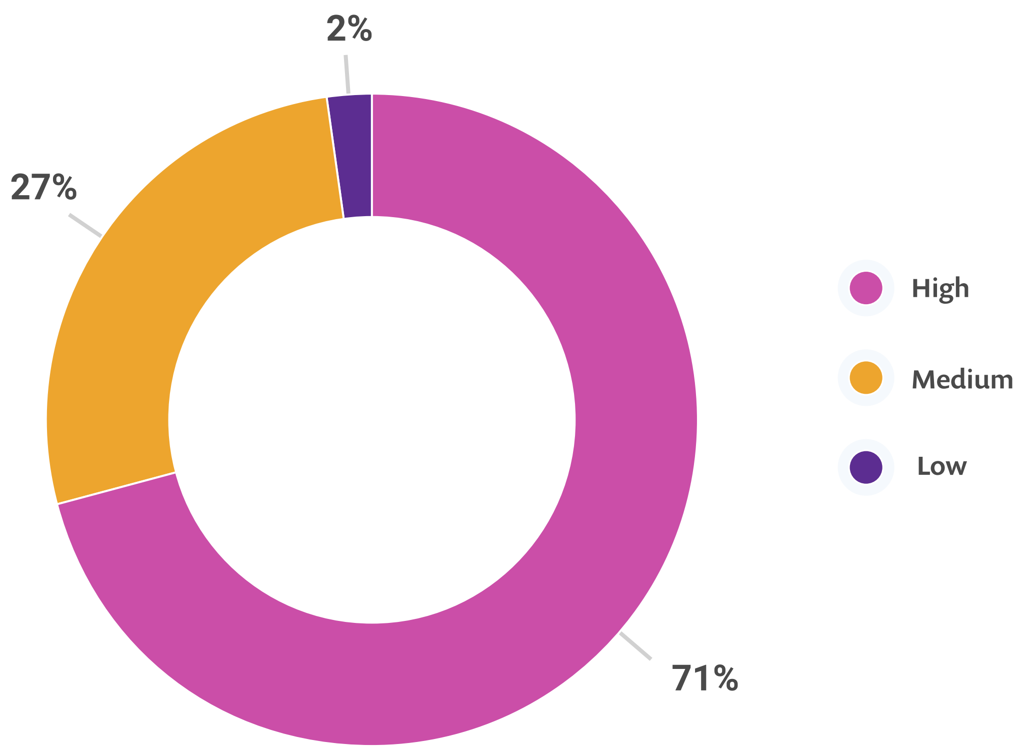
Known .NET vulnerabilities tend to be high severity

Earlier, when we we considered the top 20 most commonly seen vulnerabilities, we found that the majority were high severity. Does this trend hold true when considering all the .NET vulnerabilities in Snyk's database? Yes! High severity vulnerabilities account for 70.7% of total. Medium severity vulnerabilities account for the next largest share, at 26.9%. Only 2.4% of .NET vulnerabilities in Snyk's database are considered low severity.

100% of .Net vulnerabilities can be remediated

This may seem to paint a bleak picture of .NET security. That is a large share of high severity vulnerabilities. However, at the time of this writing, every vulnerability found by Snyk in a dependency scan has had a remediation available. Although we cannot be 100% certain as to why this is true, it is possible that this is in part because of the support that the .NET ecosystem receives from Microsoft.

Vulnerability severity at the ecosystem level



Security spotlight: fixing vulnerabilities

Upgrade

Once a vulnerability is found, project maintainers will typically include a fix (if possible) in a future version, though the timeline can vary widely. Keeping up to date with release versions is generally a good way to stay on top of security vulnerabilities.

Sometimes it is difficult to upgrade a dependency. This can be because dependencies interact with each other and with your code.

Direct vs indirect

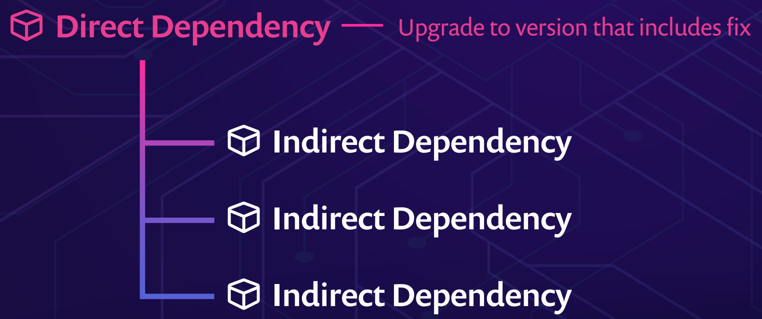
Remediating vulnerabilities in direct dependencies is usually straightforward. Upgrade the dependency to the minimum version that includes the fix.

Remediating vulnerabilities in indirect dependencies requires two things: a fixed version of the indirect dependency and a version of the direct dependency that utilizes that fixed version.

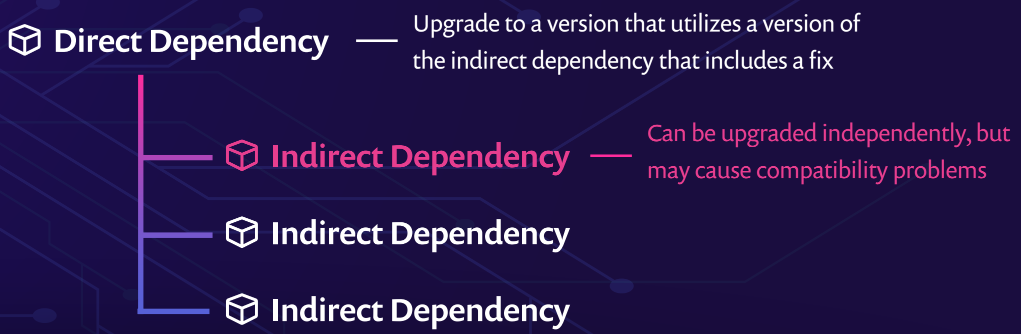
If these two conditions are met, upgrading the associated direct dependency to a version that utilizes the fixed version of the indirect dependency will remediate the issue.

If no fix is available at the level of the direct dependency, developers can upgrade the indirect dependency to resolve the issue. However, this has the potential to cause compatibility problems between the dependencies.

Vulnerability in direct dependency



Vulnerability in indirect dependency


Direct Dependency
Upgrade to a version that utilizes a version of the indirect dependency that includes a fix
Indirect Dependency
Can be upgraded independently, but may cause compatibility problems
Indirect Dependency
Indirect Dependency

Conclusion

In the .NET ecosystem, the number of vulnerabilities per package is low, but the severity of those vulnerabilities tends to be high. This means less time is likely needed to resolve the problems (lower investment) but you are addressing potentially dangerous security problems (high return). In other words, by addressing known vulnerabilities in the .NET packages that you use, you are taking a security step that has a high return on investment.

At Snyk, our goal is to help people use open source and stay secure. Part of that goal includes building tools to help people find and automatically fix known vulnerabilities in their dependencies, but identifying and properly disclosing new vulnerabilities is important as well. The .NET ecosystem does not currently have a centralized place to report vulnerabilities in open source libraries. Snyk is a CVE Numbering Authority (CNA), meaning we are able to assign a new vulnerability a CVE number (basically an ID for a vulnerability) and add the vulnerability to relevant databases. As a CNA, Snyk can help you responsibly report vulnerabilities.

You can learn more about this process here: https://snyk.io/vulnerability-disclosure.

Snyk helps you use open source and stay secure.

Get started at snyk.io

Twitter: @snyksec

Web: https://snyk.io

Office info

London

1 Mark Square
London EC2A 4EG

Tel Aviv

40 Yavne st.,
first floor

Boston

WeWork 9th Floor
501 Boylston St
Boston, MA 02116

Report author

Hayley Denbraver (@hayleydenb)

Report design

Growth Labs (@GrowthLabsMKTG)