

Zip Slip Vulnerability

Introduction

Exploitable Application Flow

Are you Vulnerable?

What action should you take?

Java

Groovy

JavaScript

.Net

Go

Ruby and Python

Other vulnerable projects

Credits

Disclosure Timeline

Zip Slip is a widespread arbitrary file overwrite critical vulnerability, which typically results in remote command execution. It was discovered and responsibly disclosed by the Snyk Security team ahead of a public disclosure on 5th June 2018, and affects thousands of projects, including ones from HP, Amazon, Apache, Pivotal and [many more \(CVEs and full list here\)](#) . Of course, this type of vulnerability has existed before, but recently it has manifested itself in a much larger number of projects and libraries.



The vulnerability has been found in multiple ecosystems, including JavaScript, Ruby, .NET and Go, but is especially prevalent in Java, where there is no central library offering high level processing of archive (e.g. zip) files. The lack of such a library led to vulnerable code snippets being hand crafted and shared among developer communities such as [StackOverflow](#) .

The vulnerability is exploited using a specially crafted archive that holds **directory traversal** filenames (e.g. `../../../../evil.sh`). The Zip Slip vulnerability can affect numerous archive formats, including `tar`, `jar`, `war`, `cpio`, `apk`, `rar` and `7z`. If you'd like the information on this page in a downloadable technical white paper, click the button below.

[Download Technical White Paper](#) (PDF 170KB)

Zip Slip is a form of directory traversal that can be exploited by extracting files from an archive. The premise of the directory traversal vulnerability is that an attacker can gain access to parts of

the file system outside of the target folder in which they should reside. The attacker can then overwrite executable files and either invoke them remotely or wait for the system or user to call them, thus achieving **remote command execution** on the victim's machine. The vulnerability can also cause damage by overwriting configuration files or other sensitive resources, and can be exploited on both client (user) machines and servers.

Exploitable Application Flow

The two parts required to exploit this vulnerability is a malicious archive and extraction code that does not perform validation checking. Let's look through each of these in turn. First of all, the contents of the zip file needs to have one or more files that break out of the target directory when extracted. In the example below, we can see the contents of a zip file. It has two files, a `good.sh` file which would be extracted into the target directory and an `evil.sh` file which is trying to traverse up the directory tree to hit the root and then add a file into the tmp directory. When you attempt to `cd ..` in the root directory, you still find yourself in the root directory, so a malicious path could contain many levels of `../` to stand a better chance of reaching the root directory, before trying to traverse to sensitive files.

```
5 Tue Jun 5 11:04:29 BST 2018 good.sh
20 Tue Jun 5 11:04:42 BST 2018 ../../../../../../tmp/evil.sh
```

The contents of this zip file have to be hand crafted. Archive creation tools don't typically allow users to add files with these paths, despite the zip specification allowing it. However, with the right tools, it's easy to create files with these paths.

The second thing you'll need to exploit this vulnerability is to extract the archive, either using your own code or a library. The vulnerability exists when the extraction code omits validation on the file paths in the archive. An example of a vulnerable code snippet (example shown in Java) can be seen below.

```
1 Enumeration<ZipEntry> entries = zip.getEntries();
2 while (entries.hasMoreElements()) {
3     ZipEntry e = entries.nextElement();
4     File f = new File(destinationDir, e.getName());
5     InputStream input = zip.getInputStream(e);
```

```
6     IOUtils.copy(input, write(f));  
7 }
```

You can see on line 4, `e.getName()` is concatenated with the target directory, `dir`, without being validated. At this point, when our zip archive gets to our `evil.sh`, it will append the full path (including every `../`) of the zip entry to the target directory resulting in `evil.sh` being written outside of the target directory.

To see Zip Slip in action, [watch us exploit the vulnerable java-goo application](#) , a sample application used to show many known vulnerabilities.

Are you Vulnerable?

You are vulnerable if you are either using a library which contains the Zip Slip vulnerability or your project directly contains vulnerable code, which extracts files from an archive without the necessary directory traversal validation. Snyk is maintaining a [GitHub repository](#) listing all projects that have been found vulnerable to Zip Slip and have been responsibly disclosed to, including fix dates and versions. The repository is open to contributions from the wider community to ensure it holds the most up to date status.

What action should you take?

Here are some steps you can take to check if your project's dependencies of code contain the Zip Slip vulnerability:

1. Search through your projects for vulnerable code.

In each ecosystem section, you'll see example snippets of code highlighting the specific vulnerability. The accompanying validation code can be added to the vulnerable snippet to test for directory traversal. You should search through your code for similar extract patterns, and ensure you're on the [fixed versions of the archive processing libraries](#) that we have found to be vulnerable.

► Java

► Groovy

► JavaScript

► .NET

► Go

► Ruby & Python

2. Add Zip Slip Security Testing to your application build pipeline

If you'd prefer not to search through your direct and transitive dependencies (of which you likely have hundreds) to determine if you're using a vulnerable library, you can choose a dependency vulnerability scanning tool, like Snyk. It's a good practice to add security testing into your development lifecycle stages, such as during development, CI, deployment and production. You can test your own projects (all the ecosystems mentioned above are supported) to determine if they are vulnerable to Zip Slip.

Other vulnerable projects

Vulnerable projects include projects in various ecosystems that either use the libraries mentioned above or directly include vulnerable code. Of the many thousands of projects that have contained similar vulnerable code samples or accessed vulnerable libraries, the most significant include: Oracle, Amazon, Spring/Pivotal, LinkedIn, Twitter, Alibaba, Jenkinsci, Eclipse, OWASP, SonarQube, OpenTable, Arduino, Elasticsearch, Selenium, JetBrains and Google.






Thank you!

The Snyk security team would like to thank all the vendors, project owners and the community members that helped raise awareness, find and fix vulnerabilities in projects across many ecosystems.

Zip Slip Disclosure Timeline

This disclosure timeline details our actions from the first private disclosure on April 15th 2018.

DATE	EVENT
Apr 15th 2018	Initial private disclosure to codehaus/plexus-archiver , zip4j , adm-zip , unzipper , mholt/archiver
Apr 15th 2018	Confirmed codehaus/plexus-archiver , zip4j , mholt/archiver
Apr 17th 2018	Confirmed adm-zip
Apr 17th 2018	unzipper fix released, v0.8.13, (CVE-2018-1002203) 
Apr 17th 2018	Snyk submitted a fix to mholt/archiver and unzipper
Apr 17th 2018	mholt/archiver fix released (CVE-2018-1002207) 
Apr 18th 2018	Private disclosure to ZeroTurnaround zt-zip
Apr 19th 2018	Disclosed to Apache (multiple projects affected)
Apr 20th 2018	Apache confirmed the issue, collaborated triage began
Apr 21st 2018	Apache Ant fix released, v1.9.12 
Apr 22nd 2018	Snyk submitted a fix to adm-fix
Apr 23rd 2018	adm-zip fix released, v0.4.9 (CVE-2018-1002204) 
Apr 25th 2018	Disclosed to DotNetZip , Semverd , SharpCompress
Apr 26th 2018	zt-zip confirmed and fixed, v1.13 (CVE-2018-1002201) 
Apr 27th 2018	HP Fortify Cloud Scan Jenkins Plugin fix released, v1.5.1 
May 02 2018	Snyk submitted a fix to SharpCompress 
May 02 2018	Apache Storm confirmed vulnerable and fixed (CVE-2018-8008)
May 3rd 2018	Private disclosure to OWASP DependencyCheck
May 3rd 2018	Private disclosure to SonarQube
May 4th 2018	OWASP DependencyCheck fix released (3.2.0)
May 4th 2018	SonarQube fixed 

May 6th 2018	Snyk submitted a fix to plexus-archiver, fix released, v3.6.0 (CVE-2018-1002200) 
May 7th 2018	DotNetZip.Semverd fix released, 1.11.0 (CVE-2018-1002205) 
May 9th 2018	Private disclosure to Pivotal Security Team
May 9th 2018	Private disclosure to Oracle Security Team
May 9th 2018	Pivotal spring-zip-integration fixed (CVE-2018-1261) 
May 9th 2018	SharpCompress fix released, v0.21.0 (CVE-2018-1002206)
May 12th 2018	Apache Commons Compress documentation fixed
May 14th 2018	Pivotal eclipse-integration-gradle fixed, v3.9.4 
May 22nd 2018	LinkedIn removed vulnerable implementation from Pinot project 
May 23rd 2018	Apache Hadoop and Hive confirmed vulnerable and fixed (CVE-2018-8009)
May 26th 2018	Oracle fixed documentation
May 31st 2018	Amazon aws-toolkit-eclipse fix released, v201805311643 