




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 Java


 JavaScript


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6


 XML





TypeScript static code analysis


Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TYPESCRIPT code


All rules279


 Vulnerability27

 Bug51


 Security Hotspot43


 Code Smell158

 Quick Fix50


 Bug


The output of functions that don't return anything should not be used

 Bug


 Bug


Comma and logical OR operators should not be used in switch cases

 Bug


 Bug


Generators should "yield" something

 Bug


 Bug


"new" operators should be used with functions

 Bug


 Bug


Non-existent operators '+=', '=-' and '!=' should not be used

 Bug


 Bug


"NaN" should not be used in comparisons

 Bug


 Bug


A "for" loop update clause should move the counter in the right direction

 Bug

 Bug

Return values from functions without side effects should not be ignored

 Bug

 Bug

Special identifiers should not be bound or assigned

 Bug

 Bug

Values should not be uselessly incremented

 Bug

 Bug

Related "if/else if" statements should not have the same condition

 Bug

 Bug


Objects should not be created to be


Tags ▾


Search by name... 🔍

Expanding archive files without controlling resource consumption is security-sensitive

Analyze your code

 Security Hotspot

 Critical

 cwe

owasp

Successful Zip Bomb attacks occur when an application expands untrusted archive files without controlling the size of the expanded data, which can lead to denial of service. A Zip bomb is usually a malicious archive file of a few kilobytes of compressed data but turned into gigabytes of uncompressed data. To achieve this extreme **compression ratio**, attackers will compress irrelevant data (eg: a long string of repeated bytes).

Ask Yourself Whether

Archives to expand are untrusted and:

- There is no validation of the number of entries in the archive.
- There is no validation of the total size of the uncompressed data.
- There is no validation of the ratio between the compressed and uncompressed archive entry.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Define and control the ratio between compressed and uncompressed data, in general the data compression ratio for most of the legit archives is 1 to 3.
- Define and control the threshold for maximum total size of the uncompressed data.
- Count the number of file entries extracted from the archive and abort the extraction if their number is greater than a predefined threshold, in particular it's not recommended to recursively expand archives (an entry of an archive could be also an archive).

Sensitive Code Example

For **tar** module:

```
const tar = require('tar');

tar.x({ // Sensitive
  file: 'foo.tar.gz'
});
```

For **adm-zip** module:





```
const AdmZip = require('adm-zip');

let zip = new AdmZip("./foo.zip");
zip.extractAllTo("."); // Sensitive
```

For **jszip** module:

https://rules.sonarsource.com/typescript/RSPEC-5042

1/5

Objects should not be created to be dropped immediately without being used
 Bug
Identical expressions should not be used on both sides of a binary operator
 Bug
All code should be reachable
 Bug
Loops with at most one iteration should be refactored
 Bug

```
const fs = require("fs");
const JSZip = require("jszip");

fs.readFile("foo.zip", function(err, data) {
  if (err) throw err;
  JSZip.loadAsync(data).then(function (zip) { // Sensitive
    zip.forEach(function (relativePath, zipEntry) {
      if (!zip.file(zipEntry.name)) {
        fs.mkdirSync(zipEntry.name);
      } else {
        zip.file(zipEntry.name).async('nodebuffer').then(fun
          fs.writeFileSync(zipEntry.name, content);
        });
      }
    });
  });
});
```

For [yauzl](#) module

```
const yauzl = require('yauzl');

yauzl.open('foo.zip', function (err, zipfile) {
  if (err) throw err;

  zipfile.on("entry", function(entry) {
    zipfile.openReadStream(entry, function(err, readStream)
      if (err) throw err;
      // TODO: extract
    });
  });
});
```

For [extract-zip](#) module:

```
const extract = require('extract-zip')

async function main() {
  let target = __dirname + '/test';
  await extract('test.zip', { dir: target }); // Sensitive
}

main();
```

Compliant Solution

For [tar](#) module:

```
const tar = require('tar');
const MAX_FILES = 10000;
const MAX_SIZE = 10000000000; // 1 GB

let fileCount = 0;
let totalSize = 0;

tar.x({
  file: 'foo.tar.gz',
  filter: (path, entry) => {
    fileCount++;
    if (fileCount > MAX_FILES) {
      throw 'Reached max. number of files';
    }

    totalSize += entry.size;
    if (totalSize > MAX_SIZE) {
      throw 'Reached max. size';
    }

    return true;
  }
});
```

For [adm-zip](#) module:

```
const AdmZip = require('adm-zip');
const MAX_FILES = 10000;
```

```

const MAX_SIZE = 10000000000; // 1 GB
const THRESHOLD_RATIO = 10;

let fileCount = 0;
let totalSize = 0;
let zip = new AdmZip("./foo.zip");
let zipEntries = zip.getEntries();
zipEntries.forEach(function(zipEntry) {
    fileCount++;
    if (fileCount > MAX_FILES) {
        throw 'Reached max. number of files';
    }

    let entrySize = zipEntry.getData().length;
    totalSize += entrySize;
    if (totalSize > MAX_SIZE) {
        throw 'Reached max. size';
    }

    let compressionRatio = entrySize / zipEntry.header.compr
    if (compressionRatio > THRESHOLD_RATIO) {
        throw 'Reached max. compression ratio';
    }

    if (!zipEntry.isDirectory) {
        zip.extractEntryTo(zipEntry.entryName, ".");
    }
});

```

For [jszip](#) module:

```

const fs = require("fs");
const pathmodule = require("path");
const JSZip = require("jszip");

const MAX_FILES = 10000;
const MAX_SIZE = 10000000000; // 1 GB

let fileCount = 0;
let totalSize = 0;
let targetDirectory = __dirname + '/archive_tmp';

fs.readFile("foo.zip", function(err, data) {
    if (err) throw err;
    JSZip.loadAsync(data).then(function (zip) {
        zip.forEach(function (relativePath, zipEntry) {
            fileCount++;
            if (fileCount > MAX_FILES) {
                throw 'Reached max. number of files';
            }

            // Prevent ZipSlip path traversal (S6096)
            const resolvedPath = pathmodule.join(targetDirectory,
            if (!resolvedPath.startsWith(targetDirectory)) {
                throw 'Path traversal detected';
            }

            if (!zip.file(zipEntry.name)) {
                fs.mkdirSync(resolvedPath);
            } else {
                zip.file(zipEntry.name).async('nodebuffer').then(fun
                totalSize += content.length;
                if (totalSize > MAX_SIZE) {
                    throw 'Reached max. size';
                }

                fs.writeFileSync(resolvedPath, content);
            });
        }
    });
});
});
});
});

```

Be aware that due to the similar structure of sensitive and compliant code the issue will be raised in both cases. It is up to the developer to decide if the implementation is secure.

For [yauzl](#) module

```

const yauzl = require('yauzl');

const MAX_FILES = 10000;
const MAX_SIZE = 1000000000; // 1 GB
const THRESHOLD_RATIO = 10;

yauzl.open('foo.zip', function (err, zipfile) {
  if (err) throw err;

  let fileCount = 0;
  let totalSize = 0;

  zipfile.on("entry", function(entry) {
    fileCount++;
    if (fileCount > MAX_FILES) {
      throw 'Reached max. number of files';
    }

    // The uncompressedSize comes from the zip headers, so i
    // Alternatively, calculate the size from the readStream
    let entrySize = entry.uncompressedSize;
    totalSize += entrySize;
    if (totalSize > MAX_SIZE) {
      throw 'Reached max. size';
    }

    if (entry.compressedSize > 0) {
      let compressionRatio = entrySize / entry.compressedSiz
      if (compressionRatio > THRESHOLD_RATIO) {
        throw 'Reached max. compression ratio';
      }
    }

    zipfile.openReadStream(entry, function(err, readStream)
      if (err) throw err;
      // TODO: extract
    ));
  });
});

```

Be aware that due to the similar structure of sensitive and compliant code the issue will be raised in both cases. It is up to the developer to decide if the implementation is secure.

For [extract-zip](#) module:

```

const extract = require('extract-zip')

const MAX_FILES = 10000;
const MAX_SIZE = 1000000000; // 1 GB
const THRESHOLD_RATIO = 10;

async function main() {
  let fileCount = 0;
  let totalSize = 0;

  let target = __dirname + '/foo';
  await extract('foo.zip', {
    dir: target,
    onEntry: function(entry, zipfile) {
      fileCount++;
      if (fileCount > MAX_FILES) {
        throw 'Reached max. number of files';
      }

      // The uncompressedSize comes from the zip headers, so
      // Alternatively, calculate the size from the readStre
      let entrySize = entry.uncompressedSize;
      totalSize += entrySize;
      if (totalSize > MAX_SIZE) {
        throw 'Reached max. size';
      }

      if (entry.compressedSize > 0) {
        let compressionRatio = entrySize / entry.compressedS
        if (compressionRatio > THRESHOLD_RATIO) {
          throw 'Reached max. compression ratio';
        }
      }
    }
  });
}

```

```
    }  
  });  
}  
main();
```

See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2021 Category A5](#) - Security Misconfiguration
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-409](#) - Improper Handling of Highly Compressed Data (Data Amplification)
- [bamssoftware.com](#) - A better Zip Bomb

Available In:

sonarcloud  **sonarqube** 

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)