

”Zip Slip” 確認してみた

version 2

2023/11/24 勉強会 @salty_byte

目次

- Zip Slipとは
 - 攻撃シナリオ
 - 脆弱なコード例
 - 原因
 - 対策
- ZIPファイル作成方法
- まとめ



Zip Slipとは

Zip Slipとは

- パストラバーサルの種類

- ファイル名に特定の文字列(/ や ../, ¥)を含んだファイルでアーカイブファイルを作り、Webサーバ上でそのファイルを展開処理させることで、**任意の場所にファイルを配置させる**ことができる※脆弱性。
- 任意コード実行(RCE)に繋がる可能性がある。

※Webアプリケーションの権限でできる範囲のみ。

Zip Slipとは

- **影響範囲**

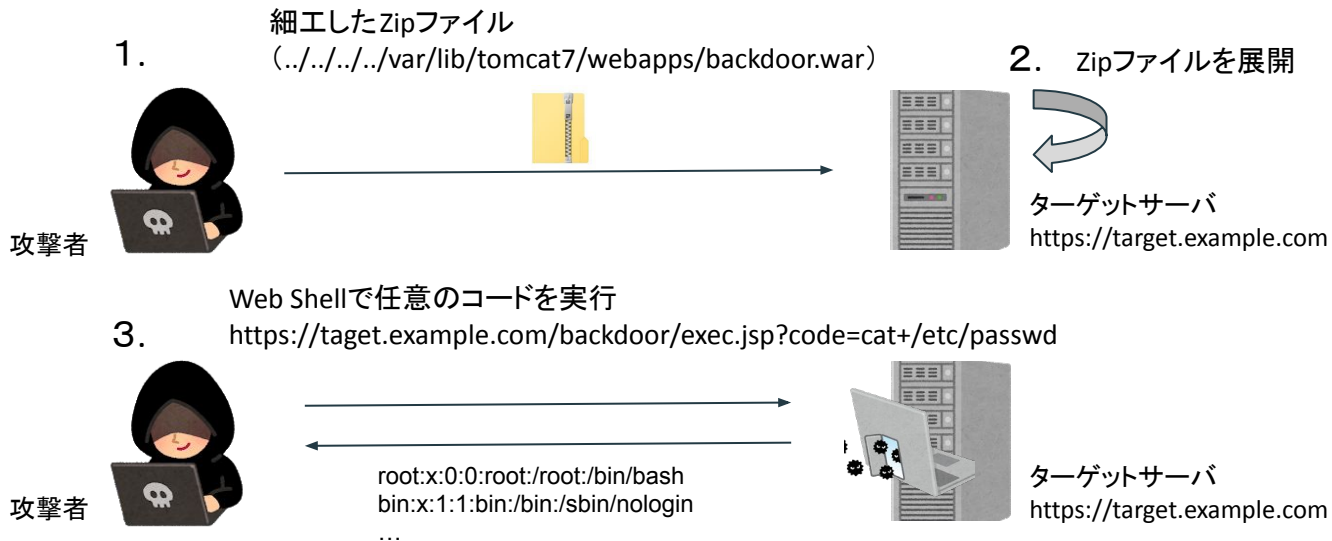
- アーカイブ形式:
 - zip/tar/jar/war/cpio/apk/rar/7z 等
- 開発言語:
 - JavaScript, Groovy, .NET and Go, Java 等
 - StackOverflowなどで脆弱なコードが共有されている場合がある。
 - RubyやPythonは既に脆弱性として過去に修正されている。
- PoCはSnyk(スニーク)によって既に公開されている。

Zip Slipとは

- **報告日**
 - 2018/06/05
- **報告者**
 - Snyk (<https://snyk.io/>)
- **CVE**
 - 各プロダクト毎に割り当てられている。
 - CVE-2018-1002200 ~ CVE-2018-1002209とか

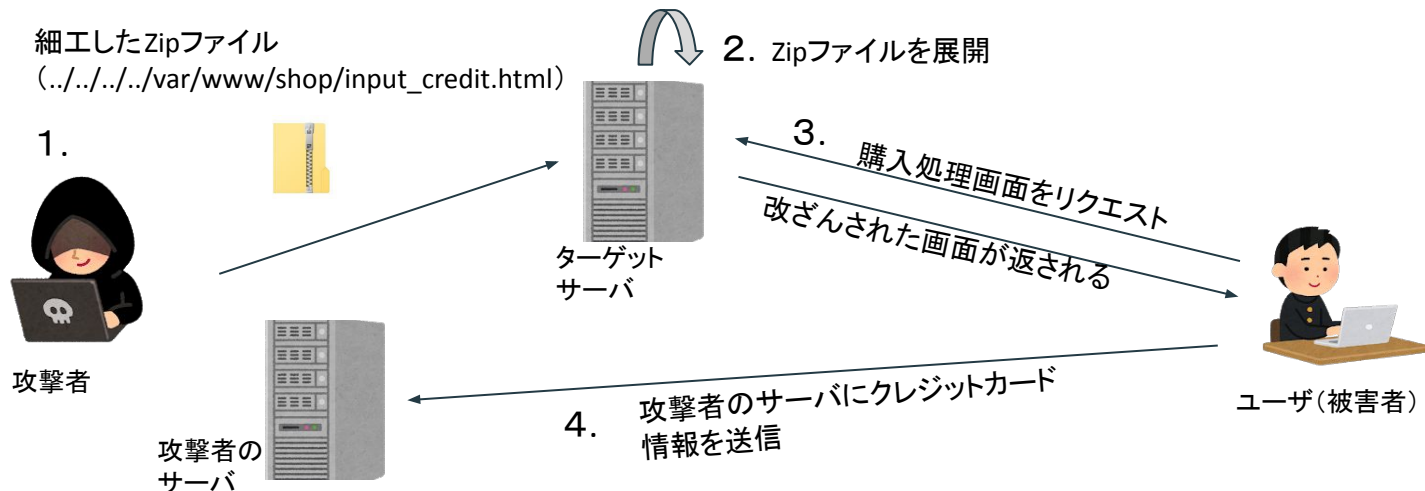
攻撃シナリオ1 (Web Shell配置)

1. 攻撃者がWeb Shell用に細工した圧縮ファイルをアップロードする。
2. アップロードしたファイルが参照可能な場所に展開される。
3. 攻撃者はWeb Shellにアクセスし、任意のコードを実行する。



攻撃シナリオ2(コンテンツ改ざん)

1. 攻撃者が偽のクレジットカード入力画面を含んだ圧縮ファイルをアップロードする。
2. アップロードしたファイルが展開され、元々のクレジットカード入力画面が上書きされる。
3. ユーザが購入処理を行う際に、偽のクレジットカード入力画面が表示される。
4. ユーザが意図せず攻撃者のサーバにクレジットカード情報を送信してしまう。



攻撃シナリオ(その他)

- **実行可能ファイルの上書き**
 - 定期実行ファイル cron
 - DoS
- **Webサイト設定ファイルの上書き**
 - アクセス制限の解除
 - リダイレクト設定の変更

脆弱なコード例

- 言語: Java
- 概要:
 - `java.util.zip.ZipInputStream`を使ったありがちな実装ミス。
- 動作:
 - 指定されたzipファイルを「/tmp/files」配下に解凍する。
- 備考:
 - 現実に「Java zip 解凍」のような検索ワードで、似たような実装が上位に出てくる。
 - ChatGPT 3.5でも同じようなプログラムが生成される。

脆弱なコード例: Java (by ChatGPT 3.5)


```
import java.io.*;
import java.nio.file.*;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

public class UnzipProgram {
    public static void main(String[] args) {
        // ユーザーからの標準入力からZIPファイルのパスを取得
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("ZIPファイルのパスを入力してください: ");
        try {
            String zipFilePath = reader.readLine();
            // ZIPファイルを展開するディレクトリ
            String outputFolder = "/tmp/files";
            // ZIPファイルを展開するメソッドを呼び出す
            unzip(zipFilePath, outputFolder);
            System.out.println("展開が完了しました。");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

脆弱なコード例: Java (by ChatGPT 3.5)

```
private static void unzip(String zipFilePath, String outputFolder) {  
    try (ZipInputStream zipInputStream = new ZipInputStream(new FileInputStream(zipFilePath))) {  
        ZipEntry zipEntry = zipInputStream.getNextEntry();  
        while (zipEntry != null) {  
            String entryFileName = zipEntry.getName();  
            Path outputPath = Paths.get(outputFolder, entryFileName);  
            // 新しいディレクトリを作成  
            if (zipEntry.isDirectory()) {  
                Files.createDirectories(outputPath);  
            } else {  
                // ファイルがなければ作成  
                Files.createDirectories(outputPath.getParent());  
                // ファイルに書き込み  
                Files.write(outputPath, zipInputStream.readAllBytes());  
            }  
            zipEntry = zipInputStream.getNextEntry();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

正常ZIPファイル



```
test/  
├─ sample.txt  
└─ demo.txt
```

```
$ zipinfo test.zip  
Archive:  test.zip  
Zip file size: 484 bytes, number of entries: 3  
drwxr-xr-x  3.0 unx      0 bx stor 23-Nov-22 22:59 test/  
-rw-r--r--  3.0 unx      7 tx stor 23-Nov-22 22:58 test/sample.txt  
-rw-r--r--  3.0 unx      5 tx stor 23-Nov-22 22:59 test/demo.txt  
3 files, 12 bytes uncompressed, 12 bytes compressed:  0.0%
```

正常動作

```
$ java UnzipProgram
```

ZIPファイルのパスを入力してください: `test.zip`

展開が完了しました。

```
$ ls /tmp/files/test -al
```

total 84

```
drwxr-xr-x  2 salt salt  4096 Nov 22 23:01 .
```

```
drwxrwxrwt 120 root root 69632 Nov 22 23:01 ..
```

```
-rw-r--r--  1 salt salt    5 Nov 22 23:01 demo.txt
```

```
-rw-r--r--  1 salt salt    7 Nov 22 23:01 sample.txt
```

攻撃用ZIPファイル

evil/
├── ../../../../../../../../../../../../../../tmp/evil.txt
├── sample.txt
└── demo.txt

```
$ zipinfo evil.zip
Archive:  evil.zip
Zip file size: 745 bytes, number of entries: 4
drwxr-xr-x  3.0 unx      0 bx stor 23-Nov-23 00:55 evil/
-rw-r--r--  3.0 unx      5 tx stor 23-Nov-23 00:55
evil/../../../../../../../../../../../../tmp/evil.txt
-rw-r--r--  3.0 unx      7 tx stor 23-Nov-23 00:55 evil/sample.txt
-rw-r--r--  3.0 unx      5 tx stor 23-Nov-23 00:55 evil/demo.txt
4 files, 17 bytes uncompressed, 17 bytes compressed:  0.0%
```

攻撃

```
$ java UnzipProgram
```

ZIPファイルのパスを入力してください: **evil.zip**

展開が完了しました。

```
$ ls /tmp/files/evil -al
```

total 16

drwxr-xr-x 2 salt salt 4096 Nov 23 01:20 .

drwxr-xr-x 3 salt salt 4096 Nov 23 01:20 ..

-rw-r--r-- 1 salt salt 5 Nov 23 01:20 demo.txt

-rw-r--r-- 1 salt salt 7 Nov 23 01:20 sample.txt

```
$ ls /tmp -al | grep evil.txt
```

-rw-r--r-- 1 salt salt 5 Nov 23 01:20 evil.txt

脆弱なコード例: Java (by ChatGPT 3.5)

脆弱な箇所

```
String entryFileName = zipEntry.getName();
Path outputPath = Paths.get(outputFolder, entryFileName);
// 新しいディレクトリを作成
if (zipEntry.isDirectory()) {
    Files.createDirectories(outputPath);
} else {
    // ファイルがなければ作成
    Files.createDirectories(outputPath.getParent());
    // ファイルに書き込み
    Files.write(outputPath, zipInputStream.readAllBytes());
}
```

脆弱なコード例: Java (by ChatGPT 3.5)

脆弱な箇所

```
String entryFileName = zipEntry.getName();
Path outputPath = Paths.get(outputFolder, entryFileName);
// 新しいディレクトリを作成
if (zipEntry.isDirectory()) {
    Files.createDirectories(outputPath);
} else {
    // ファイルがなければ作成
    Files.createDirectories(outputPath.getParent());
    // ファイルに書き込み
    Files.write(outputPath, entry.getContent());
}
```

■入力

outputFolder ⇒ "/tmp/files"

entryFileName ⇒ "../../../../../../../../../../../../tmp/evil.txt"

■パス結合 (Paths.get)

outputPath ⇒ "/tmp/files../../../../../../../../tmp/evil.txt"

■出力

File f = new File("/tmp/evil.txt");

原因

- **パストラバーサルができることが問題**
 - ファイル名を外部から指定することができる。
 - ファイル名として絶対パスや相対パスの形で異なるディレクトリを指定できる。

対策

- 脆弱なライブラリを利用しない
 - 対策済みバージョンにアップデートする。
- ファイル展開処理を自作する場合は必ず検証を入れる
 - 絶対パス(/ や C:\ 等から始まるパス)
 - 相対パス(.. / や ..\ 等を含むパス)

Javaコーディングスタンダード

- IDS04-J. ZipInputStream からファイルを安全に展開する / JPCERT CC
<https://www.jpcert.or.jp/java-rules/ids04-j.html>

“ZIP ファイルに記録されているファイル名情報にはディレクトリパスが含まれている可能性があるということである。(中略)この脆弱性を防ぐには、**展開処理の前に、ファイルパスを正規化(canonicalize)して適切な値であるかどうかを検証するとよい。**”



デモ

脆弱な環境をつくる

- Docker
- CentOS7
- Python2.7.3 (2.7.4でzipFileの脆弱性は対策済)

脆弱な環境をつくる

Dockerfile

```
FROM centos:centos7
```

```
WORKDIR /app
```

```
RUN yum -y install kernel-devel kernel-headers gcc-c++ patch libyaml-devel libffi-devel autoconf automake make libtool bison  
tk-devel zip wget tar gcc zlib zlib-devel bzip2 bzip2-devel readline readline-devel sqlite sqlite-devel openssl openssl-devel git  
gdbm-devel python-devel unzip tree
```

```
ENV HOME /root
```

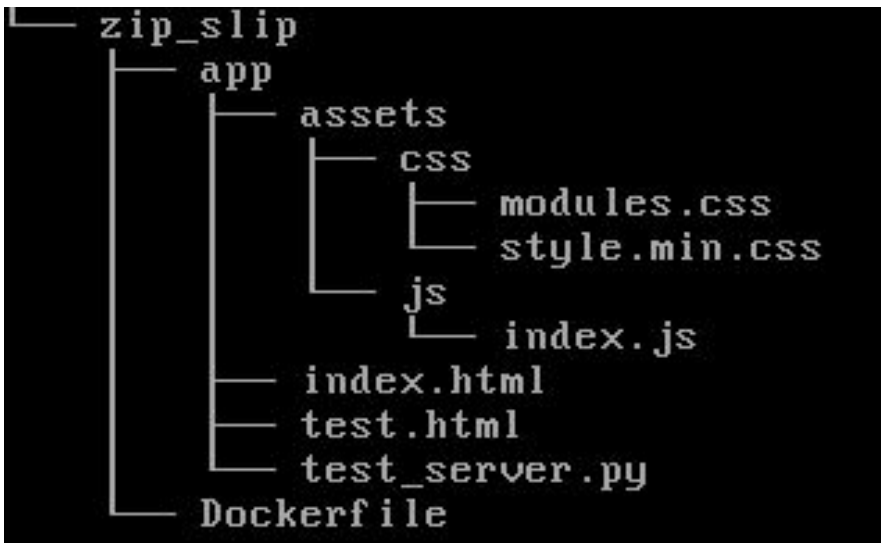
```
ENV PYENV_ROOT $HOME/.pyenv
```

```
ENV PATH $PYENV_ROOT/bin:$PATH
```

```
RUN git clone https://github.com/yyuu/pyenv.git $HOME/.pyenv \  
    && echo 'eval "$(pyenv init -)"' >> ~/.bashrc && eval "$(pyenv init -)" \  
    && pyenv install 2.7.3 && pyenv install 2.7.4 && pyenv global 2.7.3
```


脆弱な環境をつくる

フォルダ階層



脆弱な環境をつくる

実行

コンテナを起動し、コンテナ内でサーバを起動する

```
$ docker run --rm -it -p 8080:80 -v $(pwd)/app:/app zip_slip_test bash
```

```
$ python -m http.server 80
```

コンテナを起動し、同時に脆弱なバージョンで pythonサーバを起動する

```
$ docker run --rm -p 8080:80 -v $(pwd)/app:/app zip_slip_test bash -c 'eval "$(pyenv init -)" && python test_server.py'
```

コンテナを起動し、同時に pythonサーバを起動する(脆弱性なし)

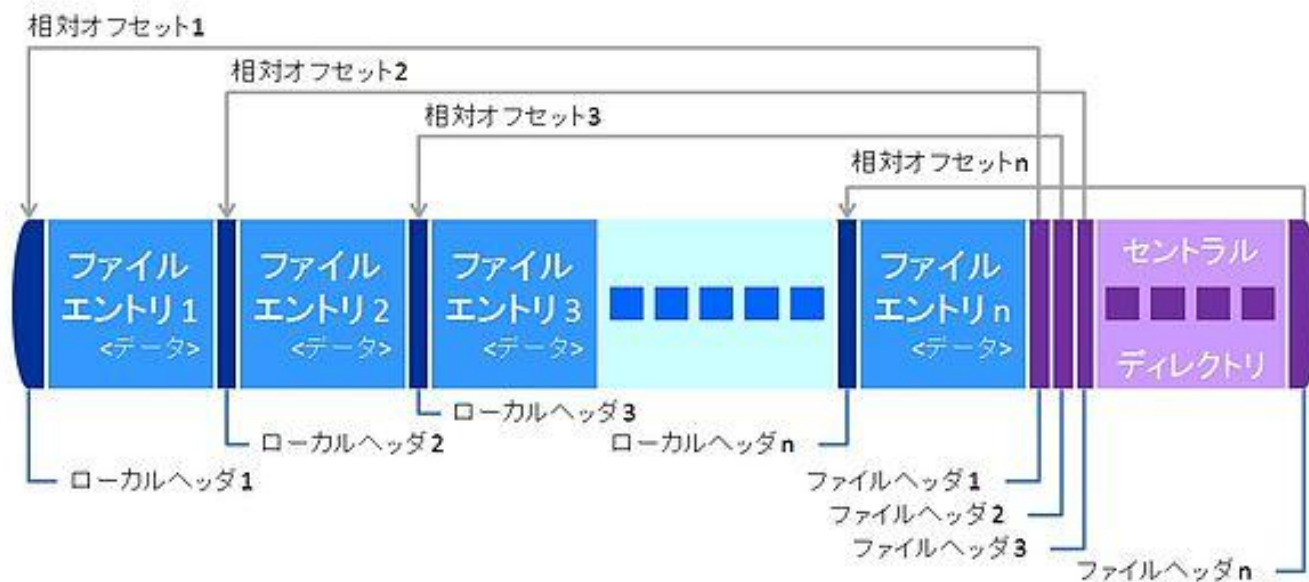
```
$ docker run --rm -p 8080:80 -v $(pwd)/app:/app zip_slip_test python -m http.server 80
```

デモ



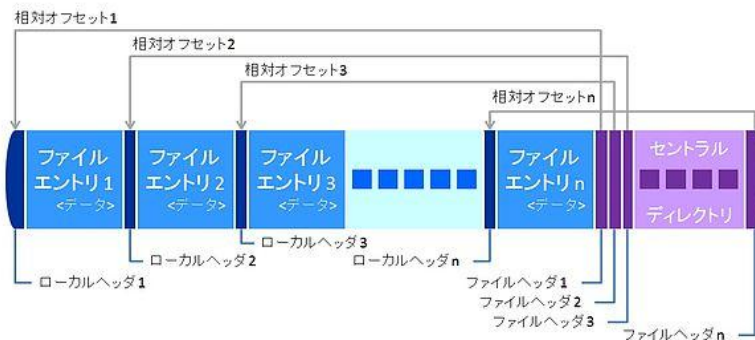
ZIP作成方法

ZIPファイル基本構造



引用: [https://ja.wikipedia.org/wiki/ZIP_\(ファイルフォーマット\)](https://ja.wikipedia.org/wiki/ZIP_(ファイルフォーマット))

ZIPファイル構造



4.3.6 Overall .ZIP file format:

<https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT>

[local file header 1] ← ローカルファイルヘッダ

[encryption header 1]

[file data 1] ← ファイルデータ

[data descriptor 1]

...

[local file header n]

[encryption header n]

[file data n]

[data descriptor n]

[archive decryption header]

[archive extra data record]

[central directory header 1] ← セントラル
ディレクトリヘッダ

...

[central directory header n]

[zip64 end of central directory record]

[zip64 end of central directory locator]

[end of central directory record] ← セントラル
ディレクトリ終端

Local file header

各ヘッダ及び終端の判別に 4 バイトのシグネチャが使われる。
値はリトルエンディアンで格納される。

local file header signature	4 bytes (0x04034b50)
version needed to extract	2 bytes
general purpose bit flag	2 bytes
compression method	2 bytes
last mod file time	2 bytes
last mod file date	2 bytes
crc-32	4 bytes
compressed size	4 bytes
uncompressed size	4 bytes
file name length	2 bytes
extra field length	2 bytes
file name	(variable size)
extra field	(variable size)

central file header signature	4 bytes (0x02014b50)
version made by	2 bytes
version needed to extract	2 bytes
general purpose bit flag	2 bytes
compression method	2 bytes
last mod file time	2 bytes
last mod file date	2 bytes
crc-32	4 bytes
compressed size	4 bytes
uncompressed size	4 bytes
file name length	2 bytes
extra field length	2 bytes
file comment length	2 bytes
disk number start	2 bytes
internal file attributes	2 bytes
external file attributes	4 bytes
relative offset of local header	4 bytes
file name	(variable size)
extra field	(variable size)
file comment	(variable size)

Central directory header

End of central directory record

end of central dir signature	4 bytes (0x06054b50)
number of this disk	2 bytes
number of the disk with the start of the central directory	2 bytes
total number of entries in the central directory on this disk	2 bytes
total number of entries in the central directory	2 bytes
size of the central directory	4 bytes
offset of start of central directory with respect to the starting disk number	4 bytes
.ZIP file comment length	2 bytes
.ZIP file comment	(variable size)

バイナリを見てみる

無圧縮のZIP

```
$ echo "zip data" > data.txt
```

```
$ zip data.zip data.txt
```

```
$ xxd data.zip
```

```
00000000: 504b 0304 0a00 0000 0000 ab64 7757 f701 PK.....dwW..  
00000010: 0787 0900 0000 0900 0000 0800 1c00 6461 .....da  
00000020: 7461 2e74 7874 5554 0900 03f1 c85e 65f1 ta.txtUT.....^e.  
00000030: c85e 6575 780b 0001 04e8 0300 0004 e803 .^eux.....  
00000040: 0000 7a69 7020 6461 7461 0a50 4b01 021e ..zip data.PK...  
00000050: 030a 0000 0000 00ab 6477 57f7 0107 8709 .....dwW.....  
00000060: 0000 0009 0000 0008 0018 0000 0000 0001 .....  
00000070: 0000 00a4 8100 0000 0064 6174 612e 7478 .....data.tx  
00000080: 7455 5405 0003 f1c8 5e65 7578 0b00 0104 tUT.....^eux....  
00000090: e803 0000 04e8 0300 0050 4b05 0600 0000 .....PK.....  
000000a0: 0001 0001 004e 0000 004b 0000 0000 00 .....N...K.....
```

バイナリを見てみる

```
$ xxd data.zip
00000000: 504b 0304 0a00 0000 0000 ab64 7757 f701 PK.....dwW..
00000010: 0787 0900 0000 0900 0000 0800 1c00 6461 .....da
00000020: 7461 2e74 7874 5554 0900 03f1 c85e 65f1 ta.txtUT.....^e.
00000030: c85e 6575 780b 0001 04e8 0300 0004 e803 .^eux.....
00000040: 0000 7a69 7020 6461 7461 0a50 4b01 021e ..zip data.PK...
00000050: 030a 0000 0000 00ab 6477 57f7 0107 8709 .....dwW.....
00000060: 0000 0009 0000 0008 0018 0000 0000 0001 .....
00000070: 0000 00a4 8100 0000 0064 6174 612e 7478 .....data.tx
00000080: 7455 5405 0003 f1c8 5e65 7578 0b00 0104 tUT.....^eux....
00000090: e803 0000 04e8 0300 0050 4b05 0600 0000 .....PK.....
000000a0: 0001 0001 004e 0000 004b 0000 0000 00 .....N...K.....
```

バイナリを見てみる

Local file header: 青

File data: 緑

Central directory header: 橙

End of central directory record: 赤

```
$ xxd data.zip
```

```
00000000: 504b 0304 0a00 0000 0000 ab64 7757 f701 PK.....dwW..
00000010: 0787 0900 0000 0900 0000 0800 1c00 6461 .....da
00000020: 7461 2e74 7874 5554 0900 03f1 c85e 65f1 ta.txtUT.....^e.
00000030: c85e 6575 780b 0001 04e8 0300 0004 e803 .^eux.....
00000040: 0000 7a69 7020 6461 7461 0a50 4b01 021e ..zip data.PK...
00000050: 030a 0000 0000 00ab 6477 57f7 0107 8709 .....dwW.....
00000060: 0000 0009 0000 0008 0018 0000 0000 0001 .....
00000070: 0000 00a4 8100 0000 0064 6174 612e 7478 .....data.tx
00000080: 7455 5405 0003 f1c8 5e65 7578 0b00 0104 tUT.....^eux....
00000090: e803 0000 04e8 0300 0050 4b05 0600 0000 .....PK.....
000000a0: 0001 0001 004e 0000 004b 0000 0000 00 ....N...K.....
```

バイナリを見てみる

Local file header: 青

File data: 緑

Central directory header: 橙

End of central directory record: 赤

```
$ xxd data.zip
```

```
00000000: 504b 0304 0a00 0000 0000 ab64 7757 f701
00000010: 0787 0900 0000 0900 0000 0800 1c00 6461
00000020: 7461 2e74 7874 5554 0900 03f1 c85e 65f1
00000030: c85e 6575 780b 0001 04e8 0300 0004 e803
00000040: 0000 7a69 7020 6461 7461 0a50 4b01 021e
00000050: 030a 0000 0000 00ab 6477 57f7 0107 8709
00000060: 0000 0009 0000 0008 0018 0000 0000 0001
00000070: 0000 00a4 8100 0000 0064 6174 612e 7478
00000080: 7455 5405 0003 f1c8 5e65 7578 0b00 0104
00000090: e803 0000 04e8 0300 0050 4b05 0600 0000
000000a0: 0001 0001 004e 0000 004b 0000 0000 00
```

```
PK.....dWl..
.....da
ta.txtUT.....^e.
.^eux.....
..zip data.PK...
.....dWl....
.....
.....data.tx
tUT.....^eux....
.....PK.....
.....N...K.....
```

ZIP仕様

<https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT>



まとめ

まとめ (Zip Slip)

- ❖ 細工されたアーカイブファイルの展開処理に対する脆弱性
 - 特定の場所にファイルを配置可能。
- ❖ アーカイブファイル内のファイルパスをそのまま使用することで脆弱になる
 - ライブラリをアップデートする。
 - ファイルパスを検証する。

Thank you for listening!

参考文献

1. <https://snyk.io/research/zip-slip-vulnerability>
2. <https://github.com/snyk/zip-slip-vulnerability>
3. <https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT>
4. <https://bombrary.github.io/blog/posts/zip-format-report/>
5. [https://ja.wikipedia.org/wiki/ZIP_\(ファイルフォーマット\)](https://ja.wikipedia.org/wiki/ZIP_(ファイルフォーマット))