unrubby

@rich0H

richö

- rich-oh!
- Computer Jerk at Stripe
- Duck Enthusiast
- Co-owner of plausibly the world's most ridiculous CVE
- WrongIslandCon jerk
- paraTROOPER

- github.com/richo
- twitter.com/rich0H

Please hold while richo takes a selfie

What this talk is

- Neat tricks with bytecode vms
- Some hilarity inside of the Rubby's VM
- Some reversing fu for people who don't like reversing

- Maybe a little opaque- please ask me questions

What this talk isn't

- Dropping 0day or bugs per se

The Problem

- Someone wants to give you a black box that does computer

- They don't want you to know how it computers

Some terminology

- VM: Virtual machine
- Opcode/Instruction: Used interchangably to refer to operations in the VM
- Bytecode: Internal representation of programs expressed as a series of opcodes

Their Solution

- Obfuscation!

Their Solution

- Obfuscation!

- Not novel:
  - Malware authors are on this case
  - Native code has been doing this for years
  - Obfuscating bytecode isn't new

This kinda sucks in a bytecode VM

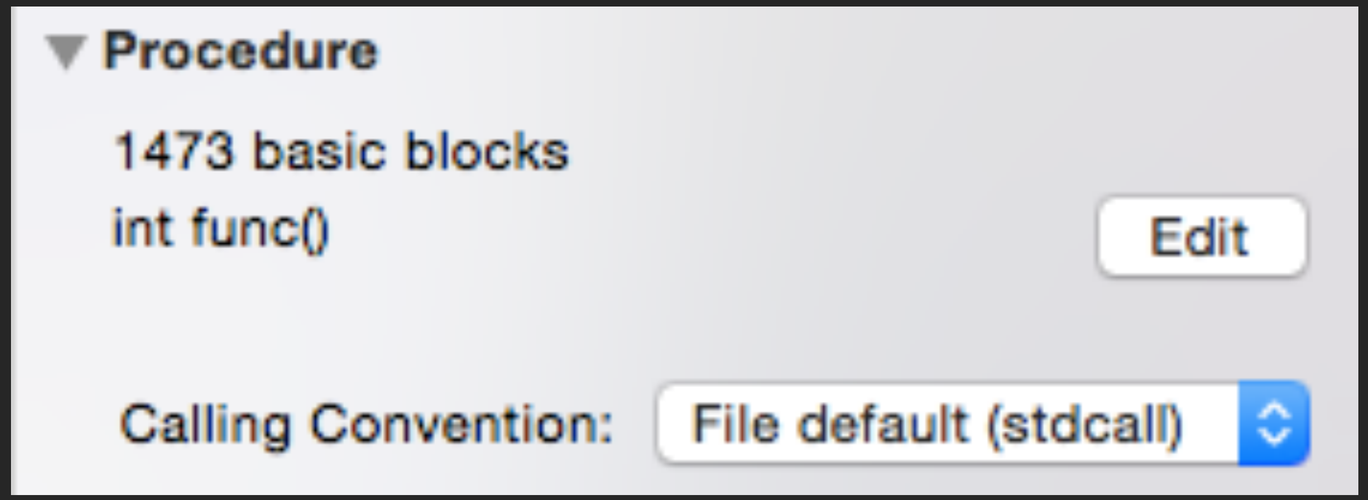
- Your options for detecting fuckery are pretty limited
  - No performance counters
  - Very limited sidechannels
  - No weird instructions to poke

This *really* sucks in a dynamic VM

- Dynamic dispatch means you can't mangle classes and methods
- Lack of a JIT means you can't do anything egregious to method bodies

Code obfuscation

- Typically packs up either source or a build product
- Loaders tend to be really complex



- Messing with RE's is seemingly fun to these people

Some more terminology

- Rubby: An interpreted, dynamic language
- YARV: Yet Another Rubby VM
- MRI: Matz Rubby Interpreter

What if you're really lazy

The Rubby VM


source_file.rb
READ
CODEGEN


TROOPERS
MAKE THE WORLD A SAFER PLACE

The Rubby VM

```ruby
module InstanceMethods
  def do_a_thing(a)
    puts "I'm doing a thing: #{a}"
  end
end

module ClassMethods
  def operation(*args)
    puts "Doing an operation on #{self} with #{args.inspect}"
  end
end

class SuperSekrit
  include InstanceMethods
  extend ClassMethods

  operation :hi, :there

  def butts(a)
    do_a_thing(a)
  end
end

SuperSekrit.new.butts("richo")
```

```
elektra % ruby decode.rb test.rb
== disasm: <RubyVM::InstructionSequence:<compiled>@<compiled>>==========
0000 trace            1                                               (   1)
0002 putspecialobject 3
0004 putnil
0005 defineclass      :InstanceMethods, <module:InstanceMethods>, 2
0009 pop
0010 trace            1                                               (   7)
0012 putspecialobject 3
0014 putnil
0015 defineclass      :ClassMethods, <module:ClassMethods>, 2
0019 pop
0020 trace            1                                               (  13)
0022 putspecialobject 3
0024 putnil
0025 defineclass      :SuperSekrit, <class:SuperSekrit>, 0
0029 pop
0030 trace            1                                               (  24)
0032 getinlinecache   39, <ic:0>
0035 getconstant      :SuperSekrit
0037 setinlinecache   <ic:0>
0039 opt_send_simple  <callinfo!mid:new, argc:0, ARGS_SKIP>
0041 putstring        "richo"
0043 opt_send_simple  <callinfo!mid:butts, argc:1, ARGS_SKIP>
0045 leave
```

The Rubby VM

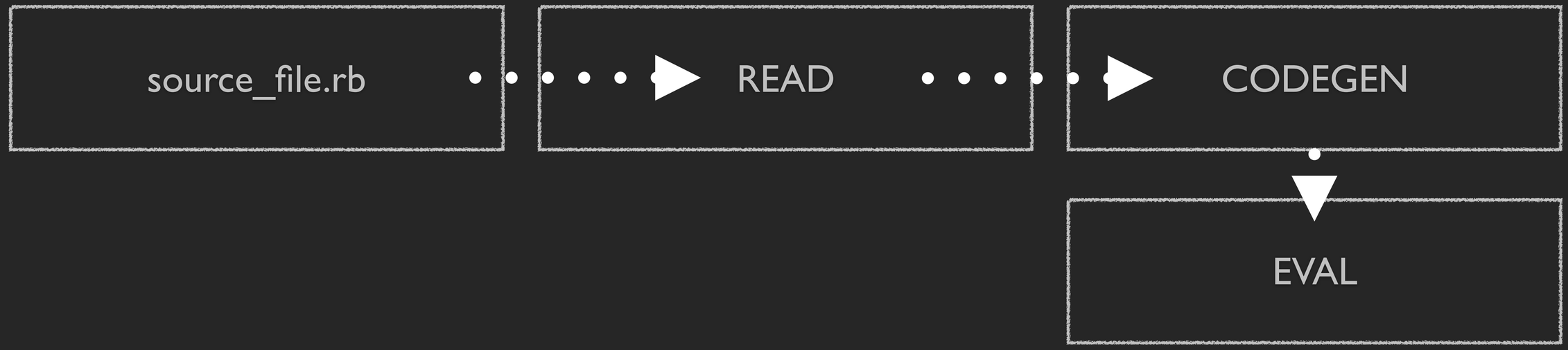
```ruby
module InstanceMethods
  def do_a_thing(a)
    puts "I'm doing a thing: #{a}"
  end
end

module ClassMethods
  def operation(*args)
    puts "Doing an operation on #{self} with #{args.inspect}"
  end
end

class SuperSekrit
  include InstanceMethods
  extend ClassMethods

  operation :hi, :there

  def butts(a)
    do_a_thing(a)
  end
end

SuperSekrit.new.butts("richo")
```

```
[:defineclass,
 :InstanceMethods,
 ["YARVInstructionSequence/SimpleDataFormat",
  2,
  0,
  1,
  {:arg_size=>0, :local_size=>1, :stack_max=>4},
  "<module:InstanceMethods>",
  "<compiled>",
  nil,
  1,
  :class,
  [],
  0,
  [],
  [1,
   [:trace, 2],
   2,
   [:trace, 1],
   [:putspecialobject, 1],
   [:putspecialobject, 2],
   [:putobject, :do_a_thing],
   [:putiseq,
    ["YARVInstructionSequence/SimpleDataFormat",
```

inside an InstructionSequence

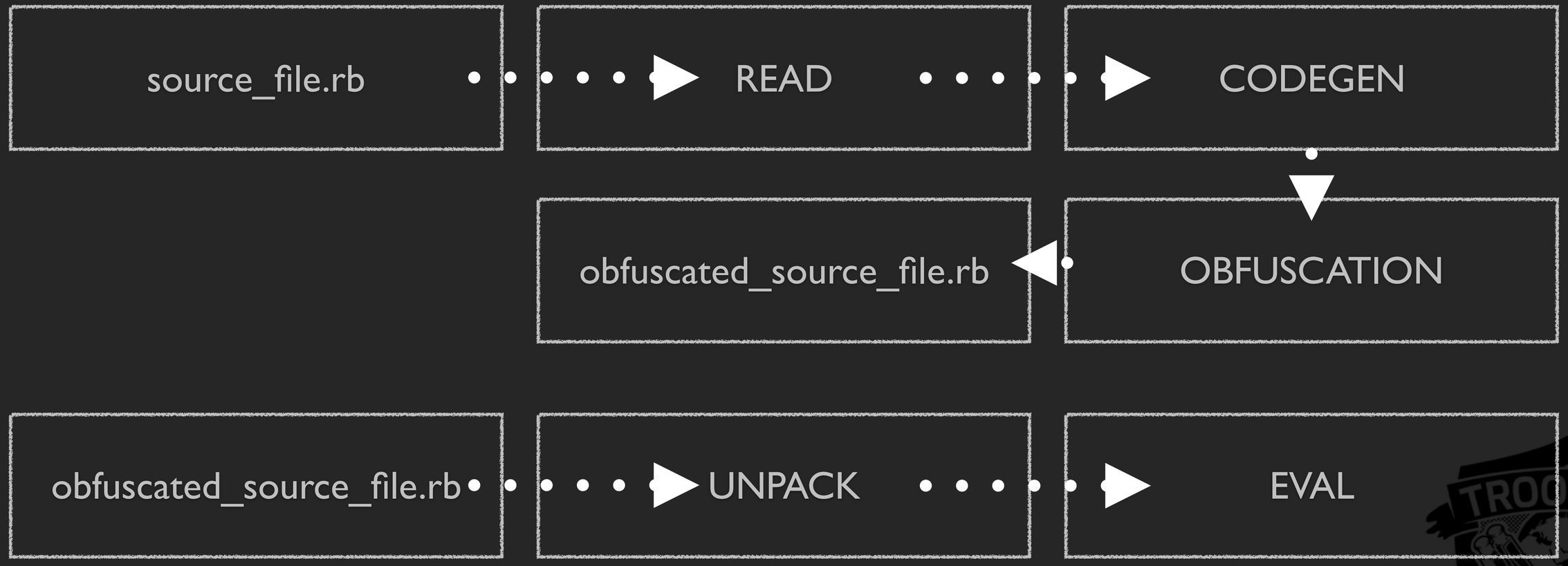
```
/* TODO: compatibility issue */
/*
 * [:magic, :major_version, :minor_version, :format_type, :misc,
 *  :name, :path, :absolute_path, :start_lineno, :type, :locals, :args,
 *  :catch_table, :bytecode]
 */
```

The Rubby VM


source_file.rb
READ
CODEGEN
EVAL


TROOPERS
MAKE THE WORLD A SAFER PLACE

The Obfuscated Rubby VM



Packed code

```
require 'loader.so'
Loader.load('bW9kdWxlIEluc3RhbmNlTWV0aG9kcwogIGRlZiBkb19hX3RoaW5n
KGEpCiAgICBwdXRzICJ20gZG9pbmcgYSB0aGluZzogI3thfSIKICBlbmQKZW5kCgp
tb2R1bGUgQ2xhc3NNZXRob2RzCiAgZGVmIG9wZXJhdGlvbigqYXJncykKICAgIHB1
dHMgIkRvaW5nIG9wZXJhdGlvbiBvbiAje3NlbGZ9IHdpdGggI3thcmdzLmluc
3BlY3R9IgogIGVuZAplbmQKCmNsYXNzIFN1cGVyU2Vrcml0CiAgaW5jbHVkZSBJbn
N0YW5jZU1ldGhvZHMKICBleHRlbmQgQ2xhc3NNZXRob2RzCgogIG9wZXJhA
6aGksIDp0aGVyZQoKICBkZWYgYnV0dHMoYSkKICAgIGRvX2FfdGhpbmcoYSkKICBl
bmQKZW5kCgpTdXBlclNla3JpdC5uZXcuYnV0dHMoInJpY2hvIikK')
```

Dynamic VM is Dynamic

- We can trivially insert instrumentation
- This.. sort of works.
  - Tack binding.pry calls everywhere
  - Attach a debugger, do a lot of `call rb_f_eval`

- Defeats for this are fairly plausible and costly to bypass
- Dynamism is a double edged sword

Rubby

- Open Source!
- We can just slam our own debug interfaces in

- Worked entirely with the reference implementation
- All mainstream loaders target it anyway
  - Typically see a loader for each of the more recent rubbies

The Rubby VM

- Interesting symbols to start with:
  - rb_eval_iseq

```
VALUE
rb_iseq_eval(VALUE iseqval)
{
  VALUE reversal;
  if (reversal = get_reversal()) {
    if (getenv("UNRUBBY_FULL_ISEQ")) {
      VALUE bytecode = rb_funcall(iseqval, rb_intern("disasm"), 0);
      rb_funcall(rb_stdout, rb_intern("puts"), 1, bytecode);
    }
  }
```

The Rubby VM

- Interesting symbols to start with:
  - rb_eval_iseq
  - rb_define_method
    - vm_define_method

The Rubby VM

- Interesting symbols to start with:
  - rb_eval_iseq
  - rb_define_method
    - vm_define_method

- rb_f_eval (lol)

Ok so we have bytecode right

- Now what?

A stack of Rubbies

- Rubby's VM is a stack machine
- Opcodes consume operands from the stack and leave values on it
- A few simple registers for storing branch conditions, pc, etc

Deeper into the YARV

```ruby
pp RubyVM::InstructionSequence.new("@a = Math.sqrt(49)").to_a
```

```
[1,
 [:trace, 1],
 [:getinlinecache, :label_9, 0],
 [:getconstant, :Math],
 [:setinlinecache, 0],
 :label_9,
 [:putobject, 49],
 [:opt_send_simple, {:mid=>:sqrt, :flag=>256, :orig_argc=>1, :blockptr=>nil}],
 [:dup],
 [:setinstancevariable, :@a, 1],
 [:leave]]
```

Expressive IR is nice

- YARV bytecode is pretty easy to read
- Auditing by hand isn't too bad

- Happily it's also sufficiently expressive that decompilation is pretty tenable

Reversal

- Research project from Michael Edgar @ dartmouth
- Similar in operation to pyRETic by Rich Smith

Reversal

- Over the course of this research I found several versions of rubby that simply won't compile
- Several debug flags that cause rubby simply not to build
- The VM has gained more instructions since 2010

Aside: instructions

- bitblt:

```
/**
  @c joke
  @e BLT
  @j BLT
 */
DEFINE_INSN
bitblt
()
()
(VALUE ret)
{
    ret = rb_str_new2("a bit of bacon, lettuce and tomato");
}
```

Aside: Docs

- Rubby is an english language (now)

- This is.. not super true for large chunks of the codebase

```
expandarray
/**
  @c put
  @e expand array to num objects.
  @j スタックトップのオブジェクトが配列であれば、それを展開する。
     配列オブジェクトの要素数が num以下ならば、代わりに nil を積む。num以上なら、
     num以上の要素は切り捨てる。
     配列オブジェクトでなければ、num - 1 個の nil を積む。
     もし flag が真なら、残り要素の配列を積む
     flag: 0x01 - 最後を配列に
     flag: 0x02 - postarg 用
     flag: 0x04 - reverse?
```

Reviving Reversal

- Patched reversal until it started working again
- Added support for rubby 1.9.3
  - And it's delightful new instructions

Presenting: unrubby

- Hacked up rubby VM
- Lots and lots of hooks into internal behaviour
- Reaches out to reversal for decompilation
- Gives you back source!

Why not just reversal

- Reversal's mode of operation is a bit fragile
- Unrubby hooks the behaviour of the VM, not the format of the bytecode
- Attempts to defeat unrubby would in turn be fragile

Digging further in

- Reversal suggests it can take the whole program and turn it back into source.
- This is largely untrue in my experience.

Obfuscation at many layers

- Problem space includes two layers:
  - Obfuscation of the bytecode itself
  - Difficult to read bytecode

Obfuscation at many layers

```
Creating a class with `defineclass`
== disasm: <RubyVM::InstructionSequence:<compiled>@<compiled>>==========
0000 trace            1                                               (   1)
0002 putspecialobject 3
0004 putnil
0005 defineclass      :Foobar, <class:Foobar>, 0
0009 leave
```

Obfuscation at many layers

```
Creating a class without `defineclass`
== disasm: <RubyVM::InstructionSequence:<compiled>@<compiled>>==========
0000 trace            1                                               (   1)
0002 getinlinecache   9, <is:0>
0005 getconstant      :Class
0007 setinlinecache   <is:0>
0009 opt_send_simple  <callinfo!mid:new, argc:0, ARGS_SKIP>
0011 dup
0012 putspecialobject 3
0014 setconstant      :Foobar
0016 leave
```

Digging further in

- We can keep abusing the runtime behaviour of the VM

- hook more stuff!
  - rb_mod_include
  - rb_obj_extend
  - rb_define_class
  - rb_define_method

Patchy patchy

```
+/* If we're in rubby, and reversal is loaded, return a reference to Reversal.
+ * Otherwise return NULL
+ */
+VALUE get_reversal(void) {
+  if (rubby) {
+    if (rb_const_defined(rb_cObject, rb_intern("Reversal"))) {
+      VALUE reversal = rb_path2class("Reversal");
+      if (rb_const_defined(reversal, rb_intern("LOADED"))) {
+        return reversal;
+      }
+    }
+  }
+  return NULL;
+}
```

Patchy patchy

```
@@ -1959,6 +1968,12 @@ vm_define_method(rb_thread_t *th, VALUE obj, ID id, VALUE iseqval,
         klass = rb_singleton_class(obj);
         noex = NOEX_PUBLIC;
     }
+    VALUE reversal;
+    if (reversal = get_reversal()) {
+      if (getenv("UNRUBBY_METHODS")) {
+        rb_funcall(reversal, rb_intern("decompile_into"), 2, iseqval, obj);
+      }
+    }
```

```
@@ -1455,6 +1456,14 @@ vm_exec(rb_thread_t *th)
 VALUE
 rb_iseq_eval(VALUE iseqval)
 {
+  VALUE reversal;
+  if (reversal = get_reversal()) {
+    if (getenv("UNRUBBY_FULL_ISEQ")) {
+      VALUE reversed = rb_funcall(reversal, rb_intern("decompile"), 1, iseqval);
+      rb_funcall(rb_stdout, rb_intern("puts"), 1, reversed);
+    }
+  }
+
```

Bonus

- This also gives us a more flexible intermediate state

- Write your own hooks in ruhby!

```
@@klassmap = Hash.new do |h, k|
  h[k] = {
    :methods => [],
    :includes => [],
    :extends => [],
    :super => nil,
  }
end
```

More bonus

- This has the impact of "unfurling" metaprogramming
- We get dynamically generated methods as well

Aside: Classes

- Rubby classes are weird
- If you think that hooking rb_define_class is enough you would be sadly mistaken

- Luckily our hook function is idempotent

- Skim class.c and hook *everything*

Demo time!

Making it go

- Rubby's insanity is super useful to us
- We can preload our library, then hijack execution flow during the eval step
- An atexit(3) hook will just dump the code to stdout

Real world breaking

- Things have dependencies
- Things want to talk to databases

- Rubby to the rescue again!

Naively

- Reimplement rails without any bodies

Rubby: richo has feels

- Rubby lets you do a bunch of things it ought not to:
  - method_missing
  - const_missing
  - reopening classes
  - monkey patching

  - etc

Or!

```ruby
class Stub
  def self.method_missing(sym, *args)
    return Stub.new
  end
end

class Object
  def self.const_missing(const)
    Stub
  end
end
```

Stealth

- Reversing things is kinda noisy
- Do this in an unroutable vm
- Unroutable vm's are miserable to work with

Stealth

- Reversing things is kinda noisy
- Do this in an unroutable vm
- Unroutable vm's are misrable to work with

- Compromises end up getting made

What's in the box?

- Rubby source tree
- Patched version of reversal
- A rails shim that ought to appease many applications

- Please play with it!
  - Please report bugs!
  - I'll drop some tips in the readme for how to report bugs without coughing up privileged code
  - UNRUBBY_REPORT_BUG

More goodies

- Lots of environment variables to control what gets emitted
  - `UNRUBBY_FULL_ISEQ`
  - `UNRUBBY_METHODS`
  - `YOLO`

- Abusing the autoloader can yield results

Care and Feeding

- unrubby currently targets rubby 2.1
- Vendors typically ship shims for their rubby
  - Upstream vendors make loader bundles available

- Autoloaded packages can make you sad
  - Implement your own entrypoint
  - Overwrite their bundled rubby

How would I defeat it?

- No super obvious way
- Unfortunately Rubby is just a really obtuse VM to target
- Cat and mouse games abound:
  - Checksum argv[0]
  - Recalculate internal offsets
- Best I came up with was to shove everything into .rodata and statically link a binary

Go forth!

- No obvious way to defeat the attack!
- Cost of attack:defense way in favour of attacker

- Novel technique that can be applied to other VMs easily

- Go reverse stuff

Gr33tz and shit

- Rich Smith - pyRETic
- Michael Edgar - Reversal

- TROOPERS for having me

- Whoever I'm missing

Questions?

Resources

- https://github.com/richo/unrubby
- https://github.com/michaeledgar/reversal

- I'll toot the link to these slides - `@rich0H`