

# Ruby For Pentesters

Mike Tracy, Chris Rohlf, Eric Monti



# Who



★ **Mike Tracy**

★ **Chris Rohlf**

★ **Eric Monti**

# Agenda



- ★ **Why Ruby**
- ★ **Scripted Pen-Testing**
- ★ **Reversing**
- ★ **Fuzzing**
- ★ **Integrating Ruby**

# Why Ruby



# Why Ruby



## ★ See a nail? Ruby is the Hammer

- Versatile
  - **Robust standard library**
  - **Extend existing classes to meet new needs**
  - **Hook existing libraries with Ruby/DL or FFI**
  - **Rubify anything by embedding Ruby**
- Generally easy to write and understand
  - **Language structure lends itself to DSL creation**
- IRB makes a great general-purpose console
  - **Blocks, mixins and monkey patching**

# Why Ruby



## ★ Java is ugly

- ... requires Java. Gross!
- Use JRuby!
  - **A full Ruby runtime inside a JVM**
  - **Ok... So what?**
    - Seamless access to pure Java classes
    - Ruby-style introspection applied to Java
  - **Bounce between Ruby and Java based on need**
  - **More later...**

# And We're Not Alone



## ★ Lots of great security tools in Ruby

- Metasploit
  - **Huge!**
- IdaRub
- Ronin
- More ...
- ... but why isn't this list longer?



## ★ Our approach to Ruby

- Use and extend what is already available to you
  - **Monkey Patches**
  - **Luckily this isn't a Ruby conference ;)**
- Don't reinvent the wheel
  - **Take tools and techniques that work and make them better**
- For example ...





## ★ RBKB - Ruby Black Bag

- A ruby clone of the original Matasano Blackbag written in C
- Extensions to existing Ruby classes and general purpose pen-testing tools
- Great for pen testing and reversing
  - **Example: extending the String class**
    - `"rubyisgreat".{xor, b64, d64, urlenc, urldec, hexdump, hexify, unhexify, blit, entropy, bgrep, crc32}`

# Scripted Pen-Testing



# The Engagement



- **Threat modeling / situational awareness**
- **Logistics challenges**
- **Everything is a webapp (even thick clients)**
- **Must find the bread and butter vulnerabilities**
- **More subtle vulnerabilities might take a back seat**

A1 - Cross Site Scripting (XSS)

A2 - Injection Flaws

A3 - Malicious File Execution

A4 - Insecure Direct Object Reference

A5 - Cross Site Request Forgery (CSRF)

A6 - Information Leakage and Improper Error Handling

A7 - Broken Authentication and Session Management

A8 - Insecure Cryptographic Storage

A9 - Insecure Communications

A10 - Failure to Restrict URL Access

# Tools You Know and Love



|                   |              |
|-------------------|--------------|
| <b>Burp Proxy</b> | WebInspect   |
| WebScarab         | AppScan      |
| Fiddler           | Acunetix     |
| Paros             | Hailstorm    |
| @Stake Proxy      | Grendel-Scan |
| w3af              | Sentinel     |

browser plug-ins

curl + sh

[sorry if I left you out]

# Why Something New?



- Previous success using scrapers and fuzzers to test web applications
- Wanted fine-grained ability to manipulate any input (surgical fuzzing) in any part of the request and detect specific responses
  - Need a console for fuzz prototyping
  - Turn fuzz prototypes into automated scripts
  - Testing thick client apps that use HTTP for transport
  - Test custom form submissions
  - Smarter spidering
- Quickly move the test focus from the bread and butter to more difficult and devastating attacks

# Why Ruby?



- **slides[4].call**
- Awesome core libraries being developed in an active community
- We're a Ruby shop and I didn't have a clue



# What Ruby Brings



- Transport
  - Curb
  - Net/HTTP
  - EventMachine
  - OpenSSL
- Parsing
  - Nokogiri
  - Hpricot
  - URI
- En(de)coding
  - Built-ins
  - Standard Library
  - Easy to mixin custom

[XPath searching an HTML DOM is incredibly useful]

```
module WWMD_Utf7
  def to_utf7
    self.scan(/./m).map { |b|
      "+" + [b.toutf16].pack("m").strip[0..2] + "-"
    }.join("")
  end
end

class String
  include WWMD_Utf7
end
```



- **Page:** all the heavy lifting
- **Scrape:** pull useful goo from pages
- **Spider:** find where everything is
- **Form\*:** manipulate and submit HTML forms
  - and GET parameters and other things
- **UrlParse:** re-inventing the wheel
- **ViewState:** deserializer / serializer / fuzzer
- **Lots of utilities for everyday tasks**
  - Parse, cut and paste from and use burp/webscarab logs
  - FormFuzzer templates
  - URLlists / Fuzzlists
  - Convenience methods to make fuzzing web services easier



# What Can I Do With It?



- A tool like scapy but for webapp pen-testing
- Integrate with the tools you already use
- Manipulate the entire request from a shell prompt
  - POST and GET parameters
  - headers, bodies and bespoke request types
- Easy shift between character encodings
- Focused customization of attack strings and wordlists
  - or fuzz using generators
- XPath searches of response bodies to create a smart fuzzer
- Instantaneous (almost) testing of exploits and concept proofs
- Trivial to automate spidering, scraping and exploit generation
- Find something new, mixin a method and it's yours forever

# Walkthrough



**And now... some code**

# welcome to example.com



example.com  
providing examples since 1992

Login:

Password:

login

# let's figure out how to login

```
> wwmd
wwmd> OPTS = { :base_url => "http://www.example.com/example" }
=> {:base_url=>"http://www.example.com/example"}
wwmd> page = Page.new(OPTS)
=> ...
wwmd> page.get "http://www.example.com/example"
=> [200, 663]
wwmd> page.now
=> "http://www.example.com/example/login.php"
wwmd> form = page.get_form
=> [{"username", nil}, {"password", nil}]
wwmd> form.type
=> "post"
wwmd> form.action
=> "http://www.example.com/example/login_handler.php"
```

# login method example



```
module WWMD
  class Page
    attr_reader :logged_in
    def login(url,uname,passwd)
      self.get(url)           ;# GET the login page
      form = self.get_form    ;# get the login form
                               ;# did we actually get a form?
      return (self.logged_in = false) unless form
      form["username"] = uname ;# set form username
      form["password"] = passwd ;# set form password
      self.submit(form)       ;# submit the form

      # naively check for password fields to see if we're still on login page
      self.logged_in = (self.search("//input[@type='password']").size == 0)
    end
  end
end
```

# login method test



```
#!/usr/bin/env ruby
require 'wvmd'
require 'example_mixins'
include WVMD

opts = { :base_url => "http://www.example.com" }
page = Page.new(opts)
page.login((page.base_url + "/example"),"jqpublic","password")
raise "not logged in" unless page.logged_in
puts page.search("//div[@class='loggedin']").first.text
```

```
>./login_test.rb
```

```
you are logged in as jqpublic [logout]
```

# what's in here?

# example.com

providing examples since 1992

you are logged in as jqpublic [\[logout\]](#)

## your user profile

|                    |                      |                        |    |                   |          |
|--------------------|----------------------|------------------------|----|-------------------|----------|
| <b>First Name:</b> | John                 | <b>Middle Initial:</b> | Q  | <b>Last Name:</b> | Public   |
| <b>Address:</b>    | 3501 S. Shields      |                        |    | <b>Apt:</b>       | Apt. 301 |
| <b>City:</b>       | Chicago              | <b>State:</b>          | IL | <b>Zip:</b>       | 60616    |
| <b>Phone:</b>      | 312-744-1000         |                        |    |                   |          |
| <b>Email:</b>      | jqpublic@example.com |                        |    |                   |          |
| <b>SSN:</b>        | ###-##-####          |                        |    |                   |          |

[edit profile](#)

### Things To Do:

[view profile](#)  
[generate report](#)

# simple spider



```
#!/usr/bin/env ruby
require 'wwmd'
require 'example_mixins'
include WWMD

opts = { :base_url => "http://www.example.com" }
page = Page.new(opts)
spider = page.spider                ;# use page's spider object
spider.set_ignore([ /logout/i, /login/i ]) ;# ignore login and logout
page.login((page.base_url + "/example"), "jqpublic", "password")
raise "not logged in" unless page.logged_in
while (url = spider.next)           ;# shift from collected urls
  code,size = page.get(url)         ;# get the shifted url
  page.summary                       ;# report on the page
end
```

```
>./spider_example.rb
```

```
XXXX[Ljfc] | 200 | OK | http://www.example.com/example/generate_report.php?userid=1045 | 818
XXXX[Ljfc] | 200 | OK | http://www.example.com/example/edit_profile.php?userid=1045 | 2740
XXXX[ljfc] | 200 | OK | http://www.example.com/example/downloads/TEMP1053623.pdf?userid=1045 | 21741
XXXX[Ljfc] | 200 | OK | http://www.example.com/example/edit_profile_handler.php?userid=1045 | 2039
```



# simple xss fuzzer



```
...
fuzz = File.read("xss_fuzzlist.txt").split("\n")
while (url = spider.next)
  code,size = page.get(url)
  next unless (form = page.get_form)           ;# page has a form?
  oform = form.clone                          ;# copy the original form
  form.each do |k,v|                          ;# each key=value in the form
    fuzz.each do |f|                          ;# each entry in the fuzzlist
      form[k] = f                             ;# set value to our fuzz string
      r = Regexp.new(Regexp.escape(f),"i")    ;# create regexp to match
      page.submit(form)                      ;# submit the form
      form = oform.clone                     ;# reset the form
      next unless page.body_data.match(r)    ;# is our string reflected?
      puts "XSS in #{k} | #{form.action}"    ;# yes
    end
  end
end
page.submit(oform)                           ;# leave things as we found them
end
```

# found some XSS

The screenshot shows a web application interface for 'example.com' with the tagline 'providing examples since 1992'. The main heading is 'your user profile'. A modal dialog box is open, displaying the URL 'http://www.example.com' and a PHP session ID 'PHPSESSID=687dupc4i5t68ffvpm529jdv74'. The dialog includes a compass icon and an 'OK' button. In the top right corner, there is a '[logout]' link. The user profile details are as follows:

|                    |                 |                        |    |                   |          |
|--------------------|-----------------|------------------------|----|-------------------|----------|
| <b>First Name:</b> | John            | <b>Middle Initial:</b> | Q  | <b>Last Name:</b> | Public   |
| <b>Address:</b>    | 3501 S. Shields |                        |    | <b>Apt:</b>       | Apt. 301 |
| <b>City:</b>       | Chicago         | <b>State:</b>          | IL | <b>Zip:</b>       | 60616    |
| <b>Phone:</b>      | 312-744-1000    |                        |    |                   |          |
| <b>Email:</b>      |                 |                        |    |                   |          |
| <b>SSN:</b>        | ###-##-####     |                        |    |                   |          |

Below the profile details is an 'edit profile' button. To the right, under the heading 'Things To Do:', there are two links: 'view profile' and 'generate report'.

```
> ./form_fuzzer_example.rb
```

```
XSS in address_2 | http://www.example.com/example/edit_profile_handler.php?userid=1045
```

```
XSS in email | http://www.example.com/example/edit_profile_handler.php?userid=1045
```

# viewstate example



```
wwmd> page = Page.new()
wwmd> vs = ViewState.new()
wwmd> page.get "http://www.example.com/vstest/test.html"
=> [200, 287]
wwmd> vs.debug = true
wwmd> page.get "http://www.example.com/vstest/test.html"
=> [200, 287]
wwmd> vs.deserialize(page.get_form['__VIEWSTATE'])
00000002 [0x0f] pair: next = string
00000003 [0x05] string: wwmd viewstate
00000013 [0x05] string: decoder
wwmd> puts vs.to_xml.pp
<ViewState version_string='ff01' version='/wE='>
  <VSPair>
    <VSString>wwmd viewstate</VSString>
    <VSString>decoder</VSString>
  </VSPair>
</ViewState>
```

# viewstate example



```
#!/usr/bin/env ruby
require 'wwmd'
include WWMD

OPTS = { :base_url => "http://www.example.com/example" }
page = Page.new(OPTS)
vs = ViewState.new()
page.get(page.base_url + "/binary_serialized_test.html")
vs.deserialize(page.get_form["__VIEWSTATE"])
vs.to_xml.search("//VSBinarySerialized").each do |node|
  puts "====[ #{node.text.size}"
  puts node.text.b64d.hexdump
end
```



## ★ Java Remote Method Invocation

- Translates:
  - **Transparent network serialization of objects between clients and servers**
- Been around 10+ years.
  - **But it crops up all over enterprise apps**
  - **We see this stuff everywhere by now**
- Examples:
  - **JMX rides on RMI**
  - **grep 'extends UnicastRemoteObject'**



## ★ Risks

- A4 - Insecure Direct Object Reference
  - ... and how
- An RMI client program will often tell you:
  - **AUTHENTICATION REQUIRED**
    - oh really?
- But where are the JRMI security testing tools?



## ★ JRMI From JRuby - a primer

- Fire up JIRB and load RMI stub classes
  - JRMI needs the client to have 'Stubs' for remote endpoints
  - In Ruby, this usually just comes down to this:  
`Dir["*.jar"].each {|jarfile| require jarfile }`
- Get a remote JRMI registry reference to walk the endpoints and their exposed methods:

```
import java.rmi.Naming      # reads just like it does in Java
registry = Naming.lookup("//victimhost:1099")
registry.list.each do |remote_name| # walk the remote endpoints
  remote = registry.lookup(remote_name)
  # walk its instance methods
  remote.java_class.declared_instance_methods.each do |meth|
    puts "#{meth.to_s}" # produce a Java method prototype
  end
end
```

# Scripted Pen-Testing



## ★ JRMI - Remote Method Invocation cont...

- Next, don't be shocked to type things like
  - `remote.getSystemConfiguration()`
  - `remote.getUserPassword('admin')`
  - `remote.executeCommand('/bin/pwn')`
- We've beaten numerous enterprise Java apps using little more than 'jirb' and a jar file.
- ... and we didn't write a single line of Java



# Reversing





## ★ Reverse Engineering

- Having a dynamic language for reversing is a must
- Ruby excels in this role
  - **Many of the built-ins feel like they were made for reversing**
  - **What isn't built is easily added**



## ★ Network Protocols

- You have to start somewhere
  - **Plugboards**
    - Blit, Plug, Telson
    - Using IRB to get inline
- More advanced ...
  - **Protocol awareness**
    - Ruckus



## ★ Network Protocols

- Blit
  - A simple OOB IPC mechanism for sending messages to blit enabled tools
- Plug
  - A reverse TCP proxy between one or more network connections
- Telson
  - Sets up a network connection and listens for messages from a blit client



## ★ Network Protocols

- Reversing a proprietary network protocol
  - **We capture a session and use Black Bag's cap2files to extract the TCP payloads**
    - cap2files will dump each payload as a small binary file with ordered file names
    - We will need these files later
  - **Read in each payload file to an array**

```
pl_ary = Array.new
d = Dir.entries('./saved_packets/')
d.delete_if do |x| x == '.' end
d.delete_if do |x| x == '..' end
d.each do |x| pl_ary.push(File.read(x)) end
```



## ★ Network Protocols

- ... continued
- **Lets try a replay attack with some modified fields**
  - modify a length field in each payload at offset 5
    - `pl_ary.each do |x| x[5] = rand(256); end`
  - connect to target with Telson
    - `telson -r 192.168.1.1:1234`
  - start up a conversation from within IRB
    - `pl_ary[0].blit`
  - or automate it with Black Bag's feed utility
    - `feed --from-files=* -r 192.168.1.1:1234`
    - `cap2files` names them in order for a reason!



## ★ Network Protocols

- Ruckus
  - **A DOM-Inspired Ruby Smart Fuzzer**
    - Declare structures like your writing C
    - Define network protocol headers
    - Built in mutators for fuzzing
    - No giant XML configuration files
    - Define your protocol in code

```
class Foo < Ruckus::Structure
  byte :id
  byte :len
  str :string
  relate_size:string, :to => :len
  relate_value :len, :to => :string, :through => :size
end

r = Foo.new
r.capture(some_packet)
pp r.to_human
```



## ★ Network Protocols

- Ruckus
  - Capture a packet in IRB
  - Define your Ruckus structure on the fly
  - Inspect the packet
  - Modify the packet
  - Print the packet

```
puts r.to_human
```

```
Foo
```

```
  id = 49 (0x31)
```

```
  len = 48 (0x30)
```

```
  string =
```

```
%%
```

```
00000000 31 30 31 6c 6b 73 6a 64 6b 6c 73 61 6a 64 00 00 |101lksjdklsajd..|
```

```
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
%%
```





## ★ Static analysis

- Extracting embedded data using Black Bag
  - **deezee**
    - Yah we ported it from the original blackbag
    - Extract embedded Zlib compressed images
  - **magicripper**
    - Go through binary blobs looking for magic numbers with libmagic
- Other handy things in Black Bag for your Strings
  - **hexify, dedump, rstrings, bgrep, ...**



## ★ A Disassembler For Your Scripts

- Frasm
  - **Distorm wrapped with Ruby**
    - Distorm is a 32/64bit x86 disassembler library written in C
    - Wrapped in a Ruby extension, and now we have frasm

```
#!/usr/bin/env ruby

require 'frasm'

d = Frasm::DistormDecoder.new
f = File.read('/bin/ls')
d.decode(f).each do |l|
  puts "#{l.mnem} #{l.size} #{l.offset} #{l.raw}"
end
```



## ★ Static Analysis

- Ruckus
  - We mentioned Ruckus earlier
  - It can be used for file formats too
  - Define structures like PE/ELF and parse up binaries just like network packets
  - Fuzz file formats with Ruckus mutators
  - Dump file format structures on the fly



## ★ Static Analysis

- There is no point in disassembling all of `/bin/l`s
  - **We need file format awareness**
- Ruckus Examples
  - **rElf**
    - Parse ELF structures with Ruckus
  - **ruPe**
    - Parse PE structures with Ruckus



## ★ Dynamic Analysis

- Ragweed
  - Sort of like 'PyDBG' except in Ruby
  - Support for Windows, OSX and Linux
  - Run Ruby blocks when breakpoints are hit
  - Write hit tracers in minutes
  - Example:

```
#!/usr/bin/env ruby

require 'ragweed'

pid = Ragweed::Debuggertux.find_by_regex(/gcalctool/)
d = Ragweed::Debuggertux.new(pid.to_i)
d.attach
d.continue
d.loop
```



## ★ Dynamic Java Analysis



- Java Debugging Interface (JDI)
  - **"jdi\_hook" drives JDI via JRuby**
    - Think kernel32 debugging API for the JVM
    - Next, think PyDBG for Java
- Why?
  - **JAD/JODE are an incomplete solution**
  - **Obfuscated Java code!**
  - **Have YOU used "jdb"?**



# Demo: Hit-tracing with “jdi\_hook”

# Reversing

- ★ **JRuby for other dynamic Java tasks**
  - Use the target against itself
    - **Hook right into its proprietary network protocols**
    - **... and proprietary crypto algorithms?**
  - Bonus
    - **Divide and conquer the debugged target**
    - **"jirb" as your debuggee for class steering**



# Fuzzing





## ★ Start Somewhere

- Dumb fuzzers in Seconds

```
def random_string(size = 8)
  chars = (0..255).map {|c| c.chr }
  (1..size).map { chars[rand(chars.size)] }.join
end
# irb(main)> random_string.unpack("H*")
# => ["c9064583d92e2598"]
# irb(main)> random_string(16).unpack("H*")
# => ["ce4074302ce90fcc8049b58e77dab7bc"]
# irb(main)> random_string(32).unpack("H*")
# => ["7d21adcc67f36d349d8470a4c2279347861175e25d6548e6e774de8876c3f0bc"]

require 'generator'
def power_A(a="A", p = 16)
  Generator.new( (0..p).map {|p| a*(1<<p)} )
end
# irb(main)> gen = power_A()
# irb(main)> gen.next
# => "A"
# irb(main)> gen.next
# => "AA"
# irb(main)> gen.next
# => "AAAA"
# irb(main)> gen.next
# => "AAAAAAAA"
# irb(main)> gen.next
# => "AAAAAAAAAAAAAAAA"
# irb(main)> gen.next
# => "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```

# Fuzzing

## ★ Pretty Soon, Design Something Cleaner

- DFuzz

```
strs = DFuzz::String.new()
while strs.next?
  target.send( strs.next )
end
```

- Thanks Dino!



## ★ Intelligent Fuzzing: Structure Awareness

- Mutation based fuzzing
- **Start with a structure (using ruckus)**

```
class DataField < Ruckus::Structure
  byte :id
  byte :len
  str :string
  relate_size :string, :to => :len
  relate_value :len, :to => :string, :through => :size
end
```

- **Now lets fuzz the 'info' field**

```
dat = DataField.new
dat.id = 0xff
dat.len = 5
dat.string.value = Ruckus::Mutator::Str.new 'A', [Ruckus::Mutator::Multiplier]
dat.string.permute => "AA"
send(dat)
dat.string.permute => "AAAA"
send(dat)
dat.string.permute => "AAAAAAAAA"
send(dat)
...
```



## ★ win32ole

- ActiveX controls are historically ripe with bugs
- COM can be awkward to work with
- WIN32OLE is Ruby's native COM API
- Plenty to work with for writing ActiveX and COM fuzzers



## ★ win32ole

- We need something a bit more automated ...
- AxRub is our ActiveX Ruby fuzzer
  - **Uses win32ole to:**
    - Enumerate methods and arguments
    - Enumerate properties
  - **Uses Ruby to:**
    - Setup a fake web server
    - Serve up HTML with fuzzed ActiveX stuff

```
a = AxRub.new(clsid, 'blacklist.txt')  
a.fuzz
```

- **Just sit back and wait for the bugs**



# Demo: ActiveX fuzzing with "axrub"

# Integrating Ruby





# Integrating Ruby



- ★ **Your old tools suck. Give them Ruby!**
  - Ruby Extensions
    - **Wrap C libraries and expose them in Ruby**
  - JRuby
    - **Java classes are all just "there" in JRuby**
  - Embedded Ruby and JRuby
    - **Ruby runtimes piggy-backing other apps**



## ★ qRub

- libnetfilter\_queue C code with embedded Ruby
  - **Was an existing tool called QueFuzz**
    - It sucked, but had a lot of useful code
  - **We ditched all the C fuzzing code and embedded Ruby instead**
  - **Easily intercept and modify packets**
  - **Drop into IRB for quick modifications**
  - **Hook into Ruby Black Bag**
  - **Reverse network protocols inline**



## ★ LeafRub



- Leaf is an extendable ELF analysis and disassembly tool written in C
- LeafRub is a Leaf plugin that embeds Ruby
  - **Analyze disassembly output using Ruby**
  - **Use Ruby extensions for different output**
    - There are gems for SQL, XML, HTML and just about anything else you want
  - **Write plugins to implement your ideas in half the time**



# Demo: Using "LeafRub"

# Integrating Ruby



## ★ Buby



- Portswigger BurpSuite is our 3rd-party web pesting tool of choice
  - ... **but it needs more Ruby**
- Burp + JRuby = Buby
  - **Burp's API exposed fully to Ruby**

# The end



# Questions?