Ruby for Pentesters: The Workshop

Timur Duehr
Cory Scott
Mike Tracy

PROGRAMMING Language

matasano SECURITY

agenda

1415: Introductions
1420: The 20 minute tour of Ruby
1440: Blackbag
1445: Webby Blackbag
1500: Protocol Blackbag
1515: Break
1530: Fuzzing and Redis
1550: Ragweed: Part 1
1610: Ragweed: Part 2
1630: Coffee Service
1700: Making Burp better with Buby
1715: JRuby
1725: FFI

Setup

Ruby in 20 Minutes

ruby for pentesters

Gems and packages

Stuff we use

Nokogiri
eventmachine
rbkb
ragweed
Nerve
Buby
lots of others

Stuff we like

Metasploit
ronin
watir
whatweb
Arachni

Lab: The basics

Ruby basics

Ruby Blackbag (rbkb)

Do less typing

Command line tools

plugboards
encoders / decoders
utilities

Object mixins

The same stuff but
for scripts and IRB

Lab: rbkb

IRB + rbkb

Scripted Webby Stuff

What do we need to script a webapp?

Transport

Parsing

Encoding / Decoding

Lab: Simple SQLi scanner

`Curb,Nokogiri,rbkb`

Protocol Reversing w/ Blackbag

ruby for pentesters

matasano SECURITY

General protocol approach

Establish the flow

Observe it

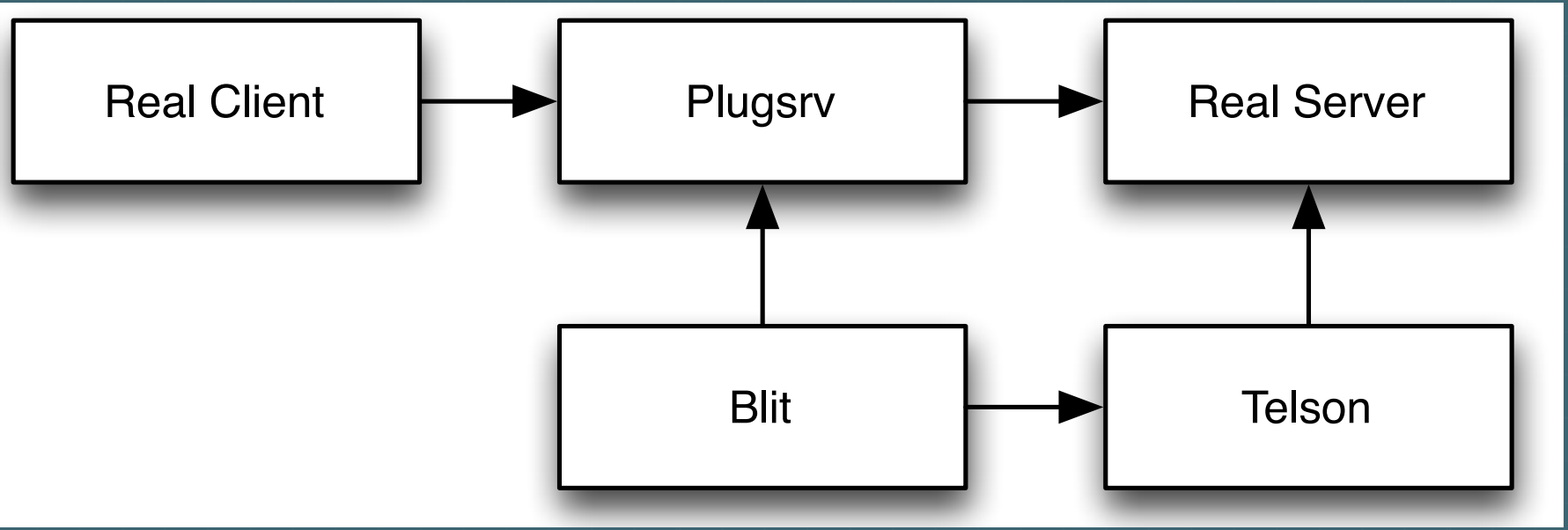
Understand it

Manipulate it

demo: when all you have is pcap...

`FeedImport`

demo: the blackbag flow



matasano SECURITY

exercise: tcp protocol lab

Get in the middle

Observe and replay

Manipulate

Exploit

exercise: build protocol structures

`BinData`

demo: eventmachine and UDP

event loops

manipulate dns

demo: TLS tricks

TLS MITM & self signed certs

SSL version scanning

Fuzzing

the what

smart

Protocol Aware
User Aware
Session Aware
Error State Aware

dumb

Bit Flipping
Boundary Trampling
Switch-a-roo
Random random everywhere

matasano SECURITY

the why

memory corruption

unexpected behavior

crypto analysis

parsers

access control test

demo: generator patterns

DFuzz + BinData

demo: the harder stuff

process control

instrumentation

binning

intro to redis

redis-server

key:value

redis object

set & get
add & delete
push & pop

redis data types

strings, lists, sets

lab: fuzzing with redis

grab your structs

mutate & send

store in redis

query

Ragweed:
Instrumentation & Getting Started

ruby for pentesters

Why a scriptable debugger?

Hittracing

Fuzzing

less clicking

runtime changes

What do we script?

Events

sigtrap
sigterm
event_fork

Actions

get registers
set breakpoints
manipulate memory

How?

memory locations

IDA
objdump
runtime calls

inspection/manipulation

get_registers
shared_libraries
read

set_registers
write

The target

in_circle
out_circle

```
eroot@Caladan32:~$ objdump -tj .text monte

monte:     file format elf32-i386

SYMBOL TABLE:
08048450 l    d  .text  00000000              .text
08048480 l     F .text  00000000              __do_global_dtors_aux
080484e0 l     F .text  00000000              frame_dummy
08048740 l     F .text  00000000              __do_global_ctors_aux
08048730 g     F .text  00000005              __libc_csu_fini
08048450 g     F .text  00000000              _start
080486d0 g     F .text  0000005a              __libc_csu_init
080486b6 g     F .text  00000015              out_circle
080486a1 g     F .text  00000015              in_circle
08048735 g     F .text  00000000              .hidden __i686.get_pc_thunk.bx
08048504 g     F .text  0000019d              main
```

Demo: arguments and registers

reading memory

getting registers

Exercise: function arguments

`SSL_read`

`SSL_write`

read SSL requests that our client is making

Walkthrough: function arguments

```
int SSL_read(SSL *ssl, void *buf, int num);

int SSL_write(SSL *ssl, const void *buf, int num);
```

Ragweed:
Hit Tracing and in Memory Fuzzing

ruby for pentesters

What do we mean by that?

Tracking function calls
and logic flow

Modifying memory locations
as the program runs

Automate this!

accounting

- Hash
- CSV
- IPC
- Redis

breakpoints

- CSV
- Nerve
- metaprogramming

Exercise: Break stuff!

in memory fuzzing

hit tracing

read arguments
write new arguments

output function hit
track order and count

Ragweed: the intermission

Recap

in memory fuzzing

screenshot TBA

hittracing

screenshot TBA

Burp + Jruby = Buby

buby is your friend

Access Burp data

- Proxy History
- Repeater
- Intruder
- Scan Results

Extend it

- Extend with modules
- Inline extraction
- Inline manipulation

Lab: CookieMonster

Grab and decode all cookies

evt_http_message	Fires when an HTTP message is received
IHttpRequestResponse	Extender object
#response_headers	Buby convenience method

Lab: CookieMunger

Modify cookie values inline

```
evt_http_message
IHttpRequestResponse
#get_request
#request=
```

```
Automatically:
Grab the request
Modify the cookie
Forward the request
```

Jruby tricks

demo: extending our buby example

```
load jar
```

```
import objects
```

```
make pretty graphs
```

FFI: interfacing with C

No gem, no problem

rubypython

libusb

ffi-libc

ragweed

ffi-pcap

OpenCV-FFI

the actual list is unimportant

This is your C struct

```
struct timezone {
        int     tz_minuteswest;
        int     tz_dsttime;
};

struct timeval {
        time_t          tv_sec;
        suseconds_t     tv_usec;
};
```

This is your C struct on Ruby

```ruby
module Libc
  extend FFI::Library
  ffi_lib 'libc'

  class Timezone < FFI::Struct
    layout :tz_minutewest, :int,
           :tz_dsttime, :int
  end

  class Timeval < FFI::Struct
    layout :tv_sec, :time_t,
           :tv_usec, :suseconds_t
  end
end
```

Calling C functions

definition

```
int
printf(const char *restrict format, ...);
```

setup

```
attach_function 'printf', [:string, :varargs], :int
```

call

```
Libc.printf("cputs %s %x %d", :string, "demo", :int, 0x33, :int, 42)
```

Exercise: execv

definition

```
int
execv(const char *path, char *const argv[]);
```

a pointer

```
argv = FFI::MemoryPointer.new(:pointer, args.size + 2)
argv[0].put_pointer(0, FFI::MemoryPointer.from_string(path.to_s))
```

OK 1-2-3 GO!

Walkthrough: execv spoiler

```
def execv(path, *args)
  FFI.errno = 0
  args.flatten!
  argv = FFI::MemoryPointer.new(:pointer, args.size + 2)
  argv[0].put_pointer(0, FFI::MemoryPointer.from_string(path.to_s))
  args.each_with_index do |arg, i|
    argv[i + 1].put_pointer(0, FFI::MemoryPointer.from_string(arg.to_s))
  end
  argv[args.size + 1].put_pointer(0, nil)

  Libc.execv(path, argv)
  raise SystemCallError :execv, FFI.errno
end
```

Follow along with me