Fuzzing native Ruby code with Kisaten

Ariel Zelivansky / Twistlock
About

- Security research lead of Twistlock Labs
- From Tel-Aviv, Israel
- Auditing security of cloud native and open-source projects
- Publishing write-ups
- Personally interested in Ruby
Agenda

- Introduction to fuzzing
- Ruby fuzzing
- Developing a native fuzzer
- Kisaten usage
- Kisaten future
What is fuzzing?

- Technique for testing software by providing it with random, unexpected or invalid input
- Fuzzing finds many bugs
  - Including complex bugs and security issues that can be missed by other techniques
- Fuzzers vary by
  - Source code awareness (Whitebox/greybox/blackbox)
  - Input awareness
Simple fuzzing cycle

1. **Corpus**
2. **Fuzzer**
   - Magic algorithm: Generates input
   - Spawn: Feed input
   - Crashes/hangs
3. **Program**

Diagram:
- Corpus
- Fuzzer
- Program
- Magic algorithm: Generates input
- Crashes/hangs
- Spawn: Feed input
American fuzzy lop (afl)

- Security-oriented greybox fuzzer
- Mutation-based fuzzing algorithm
  - Uses **binary instrumentation**
  - Able to synthesize file formats
- Fast
- Easy setup
- It works!
On top of this, the fuzzer helped make countless non-security improvements to core tools (e.g., sed, awk, make, nis, yacc, PHP, ImageMagick, freedesktop.org, patch, libtasn1, libvorbis, zsh, lua, ninja, ruby, busbox, gqrep, vim, Tor, poppler, libqubes, BSD shell, gcc, gdb, qemu, wget, ssh, dropbear, libtorrent, git, rust, gravity, easypgp, etc); found security issues in all sorts of less-widely used software (e.g., parrot, lodepng, json-glib, cabextract, libmspack, gzip, gnu-libc, dnf, awk, awk, zoo, gzip, libgcrypt, libjpeg, split, zpaq, diff, cspan, cygcheck, fasm, catdoc, pngcrush, emark, p7zip, libjpeg2, aqphoto, tuxtiles, apnglentic, gimp, libvorbis, freetype, universal-engines, uiparser, jpeg, lua, adobe, gnuplot, libexp, teseq, cimg, libcert, policycoreutils, libsemanage, rename, metasploit, openssl, mysql, perl, lug, UEFItool, libusb, ltdpd, pngquant, hmparser, mchilo, pythoon, sudo, Overpass-API, fish-shell, gnu-prc, murbox-gl-native, rapidjson, libjson, FLIF, MultiMarkdown, astyle, php-rtls, zziplib, PyPDF, spiffing, apkg, yppdmp, icontils, msitools, dos2tools, schova, MozillaShade, and so on); and is likely responsible for quite a few other things that weren't publicly attributed to the tool.
Binary Instrumentation (afl)

- Tracing program code at runtime by compilation time modification
  - Through gcc (rewriting the assembly produced)
  - Through LLVM
- No source code?
  - QEMU or Blackbox mode
- Other languages
afl and Ruby

- Fuzzing the interpreter
  - Slow…!
  - Maybe good for finding bugs within the interpreter
- Fuzzing for native C Ruby extensions
afl and Ruby

- Need to instrument *native Ruby* code
- Solution - Ruby (MRI) extension to mimic afl instrumentation
Kisaten

- **Ruby gem** for fuzzing for native Ruby code
- Integrates with MRI (Matz’s Ruby Interpreter) to trace Ruby lines
- Integrates with afl-fuzz for fuzzing logic
- Simple user experience
  - require ‘kisaten’
  - Kisaten.init
Ruby Instrumentation

- Using **TracePoint** API
  - Since Ruby 2.0
  - Similar to Kernel.#set_trace_func
- C code calls **rb_tracepoint_new** with RUBY_EVENT_LINE
- **rb_tracepoint_enable**
- Tracepoint info gives file path and line number
  - Combined into a hash that identifies each node in the execution
- Kisaten writes each tracepoint to afl’s shared memory
Integration with afl

- afl was ready
  - Set AFL_SKIP_BIN_CHECK=1 environment variable
  - Prepared for python-afl development
- Shares instrumentation data through shared memory
- Implements the forserver
  - Kisaten synchronizes with afl through pipes
    - Start, stop (timeout)
    - Return status
Forkserver Flow

```
Forkserver Flow

afl (vanilla) → afl → Executive() → Binary Fork server → Binary
```

Shared Memory

pipe()s

fork()
Forkserver Flow

- `afl (Ruby)`
- `execve()`
- `pipe()`s
- `fork()`
- `script.rb`
- `init`
Persistent mode

while Kisaten.loop 10000
    gc_food = Placeholder.logic(ARGV[0])
end

- **Purpose:** run N times without forking
  - Faster fuzzing!
  - (For stateless code)
- **Implemented by sending** SIGSTOP **signal from child Kisaten**
- **Forkserver sends** SIGCONT **to child when ready**
Ruby Exceptions

- afl determines “crash” if program was terminated by unhandled signal
  - For example, segmentation fault signal
- For Ruby code, most bugs will result in Exceptions
- Current solution
  - `RUBY_EVENT_RAISE` tracepoints to catch all exceptions
  - Kisaten lets the user decide how to handle each exception (before init)
  - Kisaten.crash_at [Crash array, Ignore array, Crash signal]

```ruby
Kisaten.crash_at [Exception], [], Signal.list['USR1']
```
Hangs

- Don’t ignore hangs!
- Hangs may indicate an infinite loop in the code
  - Or just other bugs causing suspicious slowing
Kisaten Live Demo
Bugs found

- Found bugs in Ruby gems and Ruby Standard Library gems
- Security issues will likely be DoS
Kisaten Future

- Better crash handling
  - Catching only unhandled exceptions
- Easier fuzzing setup?
  - Find Rubyist way to integrate fuzzing with Kisaten
- **Fuzz more code**
  - Fuzz more code
  - [https://github.com/twistlock/kisaten](https://github.com/twistlock/kisaten)
THANK YOU!

Ariel Zelivansky
ariel@twistlock.com
Twistlock.com/labs
@TwistlockLabs