/ruzzy

Source: `~/.claude/skills/tob-testing-handbook-skills/skills/ruzzy/SKILL.md`

---

name: ruzzy type: fuzzer description: > Ruzzy is a coverage-guided Ruby fuzzer by Trail of Bits. Use for fuzzing pure Ruby code and Ruby C extensions.

Ruzzy

Ruzzy is a coverage-guided fuzzer for Ruby built on libFuzzer. It enables fuzzing both pure Ruby code and Ruby C extensions with sanitizer support for detecting memory corruption and undefined behavior.

When to Use

Ruzzy is currently the only production-ready coverage-guided fuzzer for Ruby.

Choose Ruzzy when:

- Fuzzing Ruby applications or libraries

- Testing Ruby C extensions for memory safety issues
- You need coverage-guided fuzzing for Ruby code
- Working with Ruby gems that have native extensions

Quick Start

Set up environment:

```
export ASAN_OPTIONS="allocator_may_return_null=1:detect_leaks=0:use_sigaltstack=0"
```

Test with the included toy example:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby -e 'require "ruzzy"; Ruzzy.dummy'
```

This should quickly find a crash demonstrating that Ruzzy is working correctly.

Installation

Platform Support

Ruzzy supports Linux x86-64 and AArch64/ARM64. For macOS or Windows, use the Dockerfile or development environment.

Prerequisites

- Linux x86-64 or AArch64/ARM64
- Recent version of clang (tested back to 14.0.0, latest release recommended)
- Ruby with gem installed

Installation Command

Install Ruzzy with clang compiler flags:

```
MAKE="make --environment-overrides V=1" \
CC="/path/to/clang" \
CXX="/path/to/clang++" \
LDSHARED="/path/to/clang -shared" \
```

```
LDSHAREDXX="/path/to/clang++ -shared" \
    gem install ruzzy
```

Environment variables explained:

- `MAKE`: Overrides make to respect subsequent environment variables
- `CC`, `CXX`, `LDSHARED`, `LDSHAREDXX`: Ensure proper clang binaries are used for latest features

Troubleshooting Installation

If installation fails, enable debug output:

```
RUZZY_DEBUG=1 gem install --verbose ruzzy
```

Verification

Verify installation by running the toy example (see Quick Start section).

Writing a Harness

Fuzzing Pure Ruby Code

Pure Ruby fuzzing requires two scripts due to Ruby interpreter implementation details.

Tracer script (`test_tracer.rb`):

```
# frozen_string_literal: true

require 'ruzzy'

Ruzzy.trace('test_harness.rb')
```

Harness script (`test_harness.rb`):

```
# frozen_string_literal: true

require 'ruzzy'

def fuzzing_target(input)
```

```
  # Your code to fuzz here
  if input.length == 4
    if input[0] == 'F'
      if input[1] == 'U'
        if input[2] == 'Z'
          if input[3] == 'Z'
            raise
          end
        end
      end
    end
  end
end

test_one_input = lambda do |data|
  fuzzing_target(data)
  return 0
end

Ruzzy.fuzz(test_one_input)
```

Run with:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby test_tracer.rb
```

Fuzzing Ruby C Extensions

C extensions can be fuzzed with a single harness file, no tracer needed.

Example harness for msgpack (`fuzz_msgpack.rb`):

```
# frozen_string_literal: true

require 'msgpack'
require 'ruzzy'

test_one_input = lambda do |data|
  begin
    MessagePack.unpack(data)
```

```
  rescue Exception
    # We're looking for memory corruption, not Ruby exceptions
  end
  return 0
end

Ruzzy.fuzz(test_one_input)
```

Run with:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby fuzz_msgpack.rb
```

Harness Rules

Do	Don't
Catch Ruby exceptions if testing C extensions	Let Ruby exceptions crash the fuzzer
Return 0 from test_one_input lambda	Return other values
Keep harness deterministic	Use randomness or time-based logic
Use tracer script for pure Ruby	Skip tracer for pure Ruby code

> **See Also:** For detailed harness writing techniques, patterns for handling complex inputs, and advanced strategies, see the **fuzz-harness-writing** technique skill.

Compilation

Installing Gems with Sanitizers

When installing Ruby gems with C extensions for fuzzing, compile with sanitizer flags:

```
MAKE="make --environment-overrides V=1" \
CC="/path/to/clang" \
CXX="/path/to/clang++" \
LDSHARED="/path/to/clang -shared" \
```

```
LDSHAREDXX="/path/to/clang++ -shared" \
CFLAGS="-fsanitize=address,fuzzer-no-link -fno-omit-frame-pointer -fno-common -fPIC -g" \
CXXFLAGS="-fsanitize=address,fuzzer-no-link -fno-omit-frame-pointer -fno-common -fPIC -g" \
    gem install <gem-name>
```

Build Flags

Flag	Purpose
`-fsanitize=address,fuzzer-no-link`	Enable AddressSanitizer and fuzzer instrumentation
`-fno-omit-frame-pointer`	Improve stack trace quality
`-fno-common`	Better compatibility with sanitizers
`-fPIC`	Position-independent code for shared libraries
`-g`	Include debug symbols

Running Campaigns

Environment Setup

Before running any fuzzing campaign, set ASAN_OPTIONS:

```
export ASAN_OPTIONS="allocator_may_return_null=1:detect_leaks=0:use_sigaltstack=0"
```

Options explained:

1. `allocator_may_return_null=1`: Skip common low-impact allocation failures (DoS)
2. `detect_leaks=0`: Ruby interpreter leaks data, ignore these for now
3. `use_sigaltstack=0`: Ruby recommends disabling sigaltstack with ASan

Basic Run

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby harness.rb
```

Note: `LD_PRELOAD` is required for sanitizer injection. Unlike `ASAN_OPTIONS`, do not export it as it may interfere with other programs.

With Corpus

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby harness.rb /path/to/corpus
```

Passing libFuzzer Options

All libFuzzer options can be passed as arguments:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby harness.rb /path/to/corpus -max_len=1024 -timeout=10
```

See libFuzzer options for full reference.

Reproducing Crashes

Re-run a crash case by passing the crash file:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby harness.rb ./crash-253420c1158bc6382093d409ce2e9cff5806e980
```

Interpreting Output

Output	Meaning
`INFO: Running with entropic power schedule`	Fuzzing campaign started
`ERROR: AddressSanitizer: heap-use-after-free`	Memory corruption detected
`SUMMARY: libFuzzer: fuzz target exited`	Ruby exception occurred
`artifact_prefix='./'; Test unit written to ./crash-*`	Crash input saved
`Base64: ...`	Base64 encoding of crash input

Sanitizer Integration

AddressSanitizer (ASan)

Ruzzy includes a pre-compiled AddressSanitizer library:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby harness.rb
```

Use ASan for detecting:

- Heap buffer overflows
- Stack buffer overflows
- Use-after-free
- Double-free
- Memory leaks (disabled by default in Ruzzy)

UndefinedBehaviorSanitizer (UBSan)

Ruzzy also includes UBSan:

```
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::UBSAN_PATH') \
    ruby harness.rb
```

Use UBSan for detecting:

- Signed integer overflow
- Null pointer dereferences
- Misaligned memory access
- Division by zero

Common Sanitizer Issues

Issue	Solution
Ruby interpreter leak warnings	Use `ASAN_OPTIONS=detect_leaks=0`
Sigaltstack conflicts	Use `ASAN_OPTIONS=use_sigaltstack=0`
Allocation failure spam	Use `ASAN_OPTIONS=allocator_may_return_null=1`
LD_PRELOAD interferes with tools	Don't export it; set inline with ruby command

> **See Also:** For detailed sanitizer configuration, common issues, and advanced flags, see the **address-sanitizer** and **undefined-behavior-sanitizer** technique skills.

Real-World Examples

Example: msgpack-ruby

Fuzzing the msgpack MessagePack parser for memory corruption.

Install with sanitizers:

```
MAKE="make --environment-overrides V=1" \
CC="/path/to/clang" \
CXX="/path/to/clang++" \
LDSHARED="/path/to/clang -shared" \
LDSHAREDXX="/path/to/clang++ -shared" \
CFLAGS="-fsanitize=address,fuzzer-no-link -fno-omit-frame-pointer -fno-common -fPIC -g" \
CXXFLAGS="-fsanitize=address,fuzzer-no-link -fno-omit-frame-pointer -fno-common -fPIC -g" \
    gem install msgpack
```

Harness (`fuzz_msgpack.rb`):

```
# frozen_string_literal: true

require 'msgpack'
require 'ruzzy'

test_one_input = lambda do |data|
  begin
    MessagePack.unpack(data)
  rescue Exception
    # We're looking for memory corruption, not Ruby exceptions
  end
  return 0
end

Ruzzy.fuzz(test_one_input)
```

Run:

```
export ASAN_OPTIONS="allocator_may_return_null=1:detect_leaks=0:use_sigaltstack=0"
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby fuzz_msgpack.rb
```

Example: Pure Ruby Target

Fuzzing pure Ruby code with a custom parser.

Tracer (`test_tracer.rb`):

```
# frozen_string_literal: true

require 'ruzzy'

Ruzzy.trace('test_harness.rb')
```

Harness (`test_harness.rb`):

```
# frozen_string_literal: true

require 'ruzzy'
require_relative 'my_parser'

test_one_input = lambda do |data|
  begin
    MyParser.parse(data)
  rescue StandardError
    # Expected exceptions from malformed input
  end
  return 0
end

Ruzzy.fuzz(test_one_input)
```

Run:

```
export ASAN_OPTIONS="allocator_may_return_null=1:detect_leaks=0:use_sigaltstack=0"
LD_PRELOAD=$(ruby -e 'require "ruzzy"; print Ruzzy::ASAN_PATH') \
    ruby test_tracer.rb
```

Troubleshooting

Problem	Cause	Solution
Installation fails	Wrong clang version or path	Verify clang path, use clang 14.0.0+
`cannot open shared object file`	LD_PRELOAD not set	Set LD_PRELOAD inline with ruby command

Problem	Cause	Solution
Fuzzer immediately exits	Missing corpus directory	Create corpus directory or pass as argument
No coverage progress	Pure Ruby needs tracer	Use tracer script for pure Ruby code
Leak detection spam	Ruby interpreter leaks	Set `ASAN_OPTIONS=detect_leaks=0`
Installation debug needed	Compilation errors	Use `RUZZY_DEBUG=1 gem install --verbose ruzzy`

Related Skills

Technique Skills

Skill	Use Case
fuzz-harness-writing	Detailed guidance on writing effective harnesses
address-sanitizer	Memory error detection during fuzzing
undefined-behavior-sanitizer	Detecting undefined behavior in C extensions
libfuzzer	Understanding libFuzzer options (Ruzzy is built on libFuzzer)

Related Fuzzers

Skill	When to Consider
libfuzzer	When fuzzing Ruby C extension code directly in C/C++
aflpp	Alternative approach for fuzzing Ruby by instrumenting Ruby interpreter

Resources

Key External Resources

Introducing Ruzzy, a coverage-guided Ruby fuzzer Official Trail of Bits blog post announcing Ruzzy, covering motivation, architecture, and initial results.

Ruzzy GitHub Repository Source code, additional examples, and development instructions.

libFuzzer Documentation Since Ruzzy is built on libFuzzer, understanding libFuzzer options and behavior is valuable.

Fuzzing Ruby C extensions Detailed guide on fuzzing C extensions with compilation flags and examples.

Fuzzing pure Ruby code Detailed guide on the tracer pattern required for pure Ruby fuzzing.

Revision #5
Created 2026-02-18 08:40:12 UTC by John
Updated 2026-06-21 20:01:29 UTC by John