



# Emulating Code In Radare2

...

pancake  
Lacon 2015

# Overview

Emulation allows us to simulate the execution of code of the same or different CPU in order to understand what a specific snippet of code is doing or avoid the common risks running native code have (malware, etc).

This technique have been used to run games from old consoles.

But there have been used too in debuggers and code analyzers in order to ease the understanding



# Understanding the problem

## Emulate CPU

Requires lot of arch-specific code that needs to be rewritten for each architecture, and if we want to be really fine-grained for each CPU model.

## Performance

This is important to run games, or huge pieces of code (like an entire operating system).

Implies JIT and increases security risk.

## Intermediate Language

Emulators are usually implemented at low level, and that makes internal representation not available for deeper analysis of code.

**How To Solve All Those Problems?**

**Just use R2... but how?**

# Strings



**H**  
**HD**  
HISTORY.COM

ONIONS

# Evaluable Strings Intermediate Language

I defined a forth-like programming language to describe what every instruction of every CPU does. Some kind of micro-code.

It can be extended with native plugins to create new commands, hook on events, implement syscalls, etc

---

# Why Strings?

Strings are human-friendly.

Easy to generate, parse and modify.

Extensible by definition.

Can be redefined by the user.

Easy to translate to other forms.

---



# How Does It Look?

```
sub rsp, 0x648
```

```
1608, rsp, -=, $c, cf, =, $z, zf, =, $s, sf, =, $o, of, =
```

# Easy To Translate: REIL

```
[0x100001058]> aetr 2,3,+,4,*,rax,=  
0000.00:      ADD          3:64 ,          2:64 ,          V_00:64  
0001.00:      MUL          4:64 ,          V_00:64 ,          V_01:64  
0002.00:      STR          R_rax:64 ,          ,          V_02:64  
0002.01:      STR          V_01:64 ,          ,          R_rax:64  
[0x100001058]> █
```

ESIL is managed in r2 with the `ae` command.

# Other Translations

Radeco is the experimental decompiler for r2, it is written in Rust, as part of the GSoC-2015, and performs several computations like SSA, DCE, Constant Propagation and Verifications and can output the results in C-like form or graph.

# Current Applications For ESIL in Radare2

## Code Emulation

- Emulate a block of code
- Used in real malware samples to decrypt

## Search Conditions

- Evaluate an ESIL expression on every offset
- Useful for complex conditionals in exploiting

## Branch Prediction

- Linear emulation: e asm.emu
- Catch data refs, conditional branch, reg calls..

## Assisted Debugging

- Implements Software Watchpoints
- Step in every instruction and evaluate ESIL

## VM Emulation

- Fully emulates Baleful VM
- Can easily support Themida or ZeusVM

# Current Applications For ESIL in Radare2

## Code Analysis

- a2f
- Used in real malware samples to decrypt

## Decompilation

- Creates AST from ESIL
- Feeds the passes with info from r2

## And More!

- ...

# Expression Dependencies

## Registers

This is handled by the `r_reg` API from `r2`.

- Profile defined in plain text, supports packed register, overlapped, bitfields, and more!
- Reimplemented in Rust for Radeco.
- Each expression needs to know the offset where it is.
- ESIL have its own internal registers prefixed with '\$'

## Memory

Served by the `r_io` API.

- Virtual/Physical addresses
- Page protections and exceptions
- Allows to map different files and data at different virtual addresses.
- Emulate the Stack, Heap and BSS
- Loaded from RBin by default.
- `io.cache` to avoid real memory writes

# Hooks

**ESIL API permits to hook on every internal step of the expression evaluation.**

- register read / write
- memory read / write
- trap / exception
- syscalls
- scriptable with r2pipe
- custom/unknown instruction



# Architectures

Does not yet supports all instructions, but basic ones are enough for most uses, and grows every day a bit depending on user needs.

- Arm, Thumb, Aarch64
- Mips
- GameBoy/z80
- Powerpc
- 8051
- 6502
- Avr
- X86 (32, 64)
- Brainfuck
- Baleful
- H8300



# Alternatives?

Last Blackhat, the Capstone guy presented the Unicorn project that aims to be like ESIL:


- Using qemu as code base (not in sync)
- GPL (licensing problems)
- Provides API and bindings (Go, Java, Py)
- Cannot emulate dynamic VMs
- Uses JIT (-secure, +slower, complexity++)
- Not yet released
- Register profiles are static
- Emulates MMU with generic io api
- No IL access
- No memory cache
- Hard to implement new archs



# Unicorn Also Works in R2 (radare2-extras/unicorn)

```
pair:pe pancake$ r2 -D unicorn /bin/ls
[UNICORN] Using arch x86 bits 64

[UNICORN] dpa # reattach to initialize the unicorn
[UNICORN] dr rip=entry0 # set program counter to the entrypoint
[UNICORN] No code mapped into the Unicorn. Use `dpa` to attach and transfer
[UNICORN] Set Program Counter 0x00000000
[UNICORN] Define 64 KB stack at 0x07000000
Debugging pid = 8, tid = 8 now
-- Show offsets in graphs with 'e graph.offset = true'
[0x00001058]> █
```



# Demo Time!

## e asm.emu=true

```
0x00014e8a  ldr r1, [r0]                ; r1=0x7fe3feedface -> 0xffffffff00
0x00014e8c  mov r0, r4                  ; r0=0x0
0x00014e8e  blx sym.imp.objc_msgSend    ;[1]
    ^- 0x009a537c() ; sym.imp.objc_msgSend; lr=0x14e92 -> 0xf1904600; pc=0x9a537c
0x00014e92  mov r0, r5                  ; r0=0x0
0x00014e94  blx sym.imp.objc_release    ;[2]
    ^- 0x009a53cc() ; sym.imp.objc_release; lr=0x14e98 -> 0x10acf600; pc=0x9a53cc
0x00014e98  movw r0, 0xf9ac             ; r0=0xf9ac -> 0xf1954600
0x00014e9c  movt r0, 0xe0               ; r0=0xe0f9ac -> 0xb67400
0x00014ea0  add r0, pc                  ; r0=0xe24850 -> 0xa71c00
0x00014ea2  ldr.w sl, [r0]              ; sl=0x7fe300a71c83 -> 0xffffffff00
0x00014ea6  mov r0, r4                  ; r0=0x0
0x00014ea8  mov r1, sl                  ; r1=0xa71c83 "topBorder"
0x00014eaa  blx sym.imp.objc_msgSend    ;[1]
    ^- 0x009a537c() ; sym.imp.objc_msgSend; lr=0x14eae -> 0xf1904600; pc=0x9a537c
0x00014eae  mov r7, r7                  ; r7=0x0
0x00014eb0  blx sym.imp.objc_retainAutoreleasedReturnValue ;[3]
```

cd radare2-bindings/r2pipe/nodejs/examples/syscall  
-> emulates shellcode + syscalls using r2pipe and esil

**Thanks For Listening!**

Questions?