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
Overview

But it’s in fact way more than just this.

- CPU / FPU
- MMU (pagination, permissions, …)
- Exceptions, Traps, Etc
- Syscalls
- System Libraries
- Devices
## Understanding the problem

<table>
<thead>
<tr>
<th>Emulate CPU</th>
<th>Performance</th>
<th>Intermediate Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires lot of architecture-specific code that needs to be rewritten for each architecture, and if we want to be really fine-grained for each CPU model.</td>
<td>This is important to run games, or huge pieces of code (like an entire operating system). Implies JIT and increases security risk.</td>
<td>Emulators are usually implemented at low level, and that makes internal representation not available for deeper analysis of code.</td>
</tr>
</tbody>
</table>
How To Solve All Those Problems?

Just use R2... but how?
String
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?

```assembly
sub rsp, 0x648
1608, rsp, -=, $c, cf, =, $z, zf, =, $s, sf, =, $o, of, =
```
ESIL is managed in r2 with the `ae` command.
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.
<table>
<thead>
<tr>
<th>Current Applications For ESIL in Radare2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Emulation</strong></td>
</tr>
<tr>
<td>• Emulate a block of code</td>
</tr>
<tr>
<td>• Used in real malware samples to decrypt</td>
</tr>
<tr>
<td><strong>Search Conditions</strong></td>
</tr>
<tr>
<td>• Evaluate an ESIL expression on every offset</td>
</tr>
<tr>
<td>• Useful for complex conditionals in exploiting</td>
</tr>
<tr>
<td><strong>Branch Prediction</strong></td>
</tr>
<tr>
<td>• Linear emulation: e asm.emu</td>
</tr>
<tr>
<td>• Catch data refs, conditional branch, reg calls..</td>
</tr>
<tr>
<td><strong>Assisted Debugging</strong></td>
</tr>
<tr>
<td>• Implements Software Watchpoints</td>
</tr>
<tr>
<td>• Step in every instruction and evaluate ESIL</td>
</tr>
<tr>
<td><strong>VM Emulation</strong></td>
</tr>
<tr>
<td>• Fully emulates Baleful VM</td>
</tr>
<tr>
<td>• Can easily support Themida or ZeusVM</td>
</tr>
</tbody>
</table>
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 support 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
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

```
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'
```

(radare2-extras/unicorn)
Demo Time!
e
```
e asm.emu=true
cd radare2-bindings/r2pipe/nodejs/examples/syscall
-> emulates shellcode + syscalls using r2pipe and esil
```
Thanks For Listening!

Questions?