R2 ESIL LIMITS

R2CON 2017

GERARDO GARCÍA PEÑA

WHAT IS ESIL?

- ESIL is an intermediate language.
- It is based on evaluable strings, with a Polish-like order of evaluation.
- It is used for building a virtual machine.
- It runs securely with its own stack, registers and instruction set.
- It is used in r2 for two purposes:
  - Describing the behavior of assembly instructions.
  - Emulation.
- It is a feature of Radare2.

WHY DID I TOOK CONTACT WITH ESIL?

- Last year I rewrote from scratch with help of @brainstorm and @SkUaTeR the Atmel AVR CPU emulator.
  - In r2con 2016 @brainstorm explained how he reversed a quad-copter using radare2, and how he used r2 for fixing the CPU of one of the four motors.
  - The motors were controlled by a TinyAVR microcontroller.
  - After the talk we wanted to reverse the firmware, emulating it with the r2 AVR emulator.
  - Then we discovered that the anal module of AVR was incomplete and buggy.
  - So we rewrote it.
- After this little adventure I observed a lot of limitations and difficulties in the ESIL environment and design.
- The purpose of this talk is to share my observations for helping to find solutions for improving the ESIL architecture.

BUT… FIRST OF ALL

- I think that ESIL paradigm is great:
  - It is simple
  - It allows to easily write a small emulator.
  - Performance is not the main goal here, giving a high degree of flexibility that you would lose in other emulators like QEMU.
- And most important: it is not intended for booting a complete Windows system or running a PS4 game.
  - So if you think that the solution is embedding a QEMU inside r2 you are missing the point; you should connect to a qemu instance using the r2 gdb plugin.
- But… I think that it should be possible to emulate a simple hardware like an Arduino or C64 inside r2.
  - This feature would make possible to interactively reverse an embedded firmware or a complex piece of code.
  - At this moment I think that ESIL is fun, but not enough.

WHERE IS USED ESIL?

- Basically ESIL is a language used by a analysis (anal) plugin, used to feed the ESIL virtual machine.
- An anal plugin is a complement for a disassembler plugin (disasm).
- Usually an architecture implements both plugins, but it is not mandatory.
- There are architectures without an anal plugin and their users seem happy.
  - It is a pity…
- (taking a look into an anal plugin) (why not)

FIRST PROBLEMS

- Radare2 design does not…
  - Enforce coupled disasm and anal plugins.
  - An anal plugin usually needs more information than the r2 opcode disasm info gives.
  - Anyway an anal plugin does not receive input from the disasm plugin, so it must disasm or call explicitly some disassembler.
- This has several implications:
  - Disasm and anal plugins may return different information!
  - A bug fixed in a disasm/anal plugin may not be fixed in the related anal/disasm plugin.
  - This promotes code redundancy.

AN IDEAL ESIL STRING

- This is the ESIL of our dreams:

```
sub rsp, 8 ; (intel x64)
    8,rsp,-=,
    $o,of,=,
    $s,sf,=,
    $z,zf,=,
    $p,pf,=,
    $b8,cf,=
```

BUT A COMMON ESIL STRING

- This is not so nice:

```
sub r24, r18 ; (avr)
```

```
r18,r24,-
,r24,0x08,&,!,r18,0x08,&,!,!,&,r18,0x08,&,!,!,0,RPICK,0x08,&,!,!,&,r24,0x0
8,&,!,0,RPICK,0x08,&,!,!,&,|,|,hf,=,r24,0x80,&,!,!,r18,0x80,&,!,&,0,RPICK,
0x80,&,!,&,r24,0x80,&,!,r18,0x80,&,!,!,&,0,RPICK,0x80,&,!,!,&,|,vf,=,0,RPI
CK,0x80,&,!,!,nf,=,0,RPICK,!,zf,=,r24,0x80,&,!,r18,0x80,&,!,!,&,r18,0x80,&
,!,!,0,RPICK,0x80,&,!,!,&,r24,0x80,&,!,0,RPICK,0x80,&,!,!,&,|,|,cf,=,vf,nf
,^,sf,=,r24,=
```

BUT A COMMON ESIL STRING

- This is not so nice:

```
sub r24, r18 ; (avr)
    r18,r24,-,
    r24,0x08,&,!,r18,0x08,&,!,!,&,r18,0x08,&,!,!,0,RPICK,0x08,&,!,!,&,r24,0x08,&,!,0,RPIC
    K,0x08,&,!,!,&,|,|,hf,=,
    r24,0x80,&,!,!,r18,0x80,&,!,&,0,RPICK,0x80,&,!,&,r24,0x80,&,!,r18,0x80,&,!,!,&,0,RPIC
    K,0x80,&,!,!,&,|,vf,=,
    0,RPICK,0x80,&,!,!,nf,=,
    0,RPICK,!,zf,=,
    r24,0x80,&,!,r18,0x80,&,!,!,&,r18,0x80,&,!,!,0,RPICK,0x80,&,!,!,&,r24,0x80,&,!,0,RPIC
    K,0x80,&,!,!,&,|,|,cf,=,
    vf,nf,^,sf,=,
    r24,=
```

WHAT'S THE PROBLEM?

- ESIL is designed with Intel x86/x64 in mind
- Flags, operators, instructions… all are designed for translating x86/x64 opcodes into nice ESIL strings.
  - …but it does not work so well with other archs.
- The previous example also put in relieve two problems:
  - ESIL is not very readable.
  - Maybe ESIL is a too low level language.
- And most important: ESIL is not good for describing instructions in a simple way.

ABOUT THE CPU INTERNAL STATE

- Let's assume that ESIL is OK, and then we decide to continue with our AVR emulator.
- We have seen that the ESIL virtual machine is basically an interpreter of ESIL strings. The anal plugin only must generate these strings in a stateless way, without assuming that the ESIL string will be executed.
- This implies that the internal state will be managed by the ESIL virtual machine.
- Because of that a register profile must be declared by the anal plugin.
  - (moment to take a look into source code)

PROBLEMS SPOTTED

- If we want to emulate a real CPU it is not enough storing the public registers.
  - We must manage also a lot of internal registers that usually are not known (or documented)… and they will be published to the radare2 user, confusing him a bit more.
  - We also have to include some "configuration options" stored like registers (specially on highly customizable CPUs like those from the AVR family).
- But… what if we want to do more complex things?
  - Where can we store this CPU state information?
  - How can we reconfigure the register profile in case of changing the CPU model?
  - Or do certain things when some register changes?
    - Like changing between real and protected mode?

ADDRESS SPACE (OR SPACES)

- What is our address space?
- ESIL was thought as a mechanism for emulating small pieces of code, so it basically does not emulate a real computer address space.
- The address space by default is the address space defined during the binary load.
- This means that only one address space exists, maybe complemented with some dynamically reserved memory.
  - This is an important thing to keep in mind: if you run ESIL code on a binary opened with the -w flag, if some instruction writes on memory, all changes will be written on your binary!
- Anal plugin cannot manage or create new or existing address spaces.

THE ADDRESS SPACE NIGHTMARE

- Some archs, like AVR, have more tan one address space
  - In AVR we have a totally differentiated address spaces for CODE, RAM, IO and EEPROM.
  - But IO is mapped onto RAM, also.
  - And the CPU registers are mapped on the IO address space.
  - Party!
- Other archs have several address spaces depending on the CPU mode (protected mode vs real mode).
- And a lot of archs have mapped IO address spaces or ports in RAM.
- The ESIL virtual machine does not give any facility for this.

EXCEPTIONS, SIGNALS AND INTERRUPTIONS

- These features are essential for managing…
  - …the most basic I/O operations of a CPU (interactions with real world)
    - Signals from the outside world
    - Counters or timers hit
    - Sensors
    - …
  - …reacting to internal unexpected situations
    - Bad instructions
    - Illegal addresses
    - Division by zero
    - …

IS IT POSSIBLE TO SOLVE THIS?

- Software exceptions (i.e. division by zero) could be "easily" managed from ESIL strings, but they over bloat the ESIL code and may introduce redundancy on it.
- Other software exceptions (i.e. page fault) would introduce too much code on each memory access ESIL string, making the whole system unmaintainable and error prone.
- Some other exceptions like timers are simply impossible to write without getting mad.
- And there is not any way to declare and signal external interrupts, so it is not possible to activate them from the r2 command line or API.
- And the worst: there is no possibility of importing or loading "hardware plugins":
  - A chipset emulator (sound chips, video chips)
  - I/O devices or DMA transfers
  - Sensors
  - Etc.

AND FINALLY: THE I/O ITSELF

- A computer without I/O is stupid.
- Sooner or later we have to receive data or write data.
- Data can be represented in several ways:
  - Like analogic waves.
  - Digital values (1 or 0).
  - Screen.
  - Sound waves.
  - At least a fucking led.

NO I/O :'(

- There is not any mechanism for
  - Reading
  - Writing
  - Or presenting data (graphical buffers, sound, etc).
- I think that at least it should be possible to attach I/O to input and output files (or pipes).
  - To write or read wave files.
  - Or digital inputs.
  - Even piping those channels to external applications.
- I think this feature would bring a new dimension to the firmware reversing in r2.

SO...

- I have exposed all these shortcomings and problems and I have put them together in a PPTX.
- I know that this is not enough for improving the ESIL emulator, but at least I have made a critical review of the ESIL, putting all these limitations together.
- But I am sure that I have not covered all the limitations of ESIL or needs that a decent low level emulator should have.
- So I think that a great next step would be to make a roadmap for improving ESIL.
- I do not have too much time, but I think that building a powerful but simple multi-arch emulator in r2 would be a fantastic experience.
- So I will be glad to hear any question, opinion or proposal.

BYE!

- Any questions:

Gerardo García <killabytenow@gmail.com>

- Greetings to:
  - @Brainstorm – For the idea of implementing my own anal architecture.
  - @SkUaTeR – For his lessons on the r2 anal discipline.
  - @pancake – He invented the r2 anal style.
  - R2con ppl!