NBS System

Giovanni Dante Grazioli

Twitter: @der0ad
GitHub: @wargio

r2dec

How to not write a decompiler

5-8 September 2018
r2con

New r2dec version (1.0)!

will be merged after the talk.

Special thanks to

Eli Cohen-Nehemia (elicn)

> For assisting with the new core, and rewriting the x86-64 arch

And also to all the previous contributors of the current code in the master

> Thanks a lot !

radare2 and r2dec

radare2 is:

A reverse engineering framework, and is useful for forensics, reversing, exploiting, and more.

r2dec is:

radare2 plugin mostly written in JavaScript
Tries to convert assembly to pseudo-C code
Extends radare2 functionality

Short History of the project

The project started in May 2017

- Idea & Implementation:
  - Just write a simple decompiler for any tool.
  - It was Node JS based.
  - Each architecture had its own logic to do the analysis/optimizations/etc..
  - Support initially was only for PowerPC (32/64 bits).

1st Core rewrite (current master) started in December 2017 & 2nd Core rewrite (Aug, 2018)

- Idea & Implementation:
  - Write a core that was able to handle the given data independently of the architecture.
  - Support any architecture in an easy way.
  - Binary independent bug fixing.
  - Easy user support and issue creation on GH.
  - Transition to duktape for portability from NodeJS in March 2018

Current Supported Architectures

- arm (soon also arm64)
- avr
- mips
- m68k (experimental)
- ppc
- sparc
- v850
- wasm (partial)
- x86-64 (intel syntax)

How to install it

```
$ r2pm install r2dec
```

done.

Protip - when you update your r2 bin, be sure to also run *r2pm install r2dec* again.

Usage

Load your binary to radare2

```
$ r2 crackme
[0x080483e0]>
```

Analyze it via `af` or `aaa` (if you are brave)

```
[0x080483e0]> s main
[0x08048494]> af
```

Call r2dec via `pdd`

```
[0x08048494]> pdd
/* r2dec pseudo C output */
#include <stdint.h>

int64_t main (void) {
    char * s1;
    printf ("Password: ");
    [...]
    fgets (rax, 0x20, &s1);
    rax = &s1;
    eax = strcmp (rax, "th3p4ss");
    if (eax == 0) {
        puts ("Password Correct!");
        eax = 0;
    } else {
        puts ("Invalid Password");
        eax = 1;
    }
    return rax;
}
```

r2dec arguments (1)

```
[0x00000000]> pdd --help
r2dec [options]
 --help      | this help message
 --assembly  | shows pseudo next to the assembly
 --blocks    | shows only scopes blocks
 --colors    | enables syntax colors
 --casts     | shows all casts in the pseudo code
 --debug     | do not catch exceptions
 --html      | outputs html data instead of text
 --issue     | generates the json used for the test suite
 --paddr     | all xrefs uses physical addresses instead of virtual
addresses
 --xrefs     | shows also instruction xrefs in the pseudo code
```

r2dec arguments (2)

```
[0x00000000]> pdd?
Usage: pdd [args] - core plugin for r2dec
 pdd   - decompile current function
 pdd?  - show this help
 pdda  - decompile current function with side assembly
 pddb  - decompile current function but shows only scopes
 pddi  - generates the issue data
 pddu  - install/upgrade r2dec via r2pm
```

- `pdda` is the short from for `` `pdd --assembly` ``
- `pddb` = `` `pdd --blocks` ``
- `pddi` = `` `pdd --issue` ``

radare2 evaluable variables (e <var>)

People has its own preferred configurations and inputting each time command line arguments is sometimes boring.
r2dec has also the evaluable variables which can be added to `~/.radare2rc`

```
r2dec.casts      | if false, hides all casts in the pseudo code.
r2dec.asm        | if true, shows pseudo next to the assembly.
r2dec.blocks     | if true, shows only scopes blocks.
r2dec.offset     | if true, shows pseudo next to the offset.
r2dec.paddr      | if true, all xrefs uses physical addresses compare.
r2dec.xrefs      | if true, shows all xrefs in the pseudo code.
r2dec.theme      | defines the color theme to be used on r2dec.
```

r2dec can also use the radare2 standard evaluable variables

```
scr.html         | outputs html data instead of text.
scr.color        | enables syntax colors.
```

Reporting an Issue

1. Open your file on radare2
2. Analyze the function that r2dec fails to decompile (`af`).
3. Call `pddi` or `pdd --issue` to get the JSON.
4. Copy the JSON data (or upload it) into the issue on GitHub page.
5. Done.

r2dec debug & patching

1. Open your file on radare2
2. Analyze the function that r2dec fails to decompile (`af`).
3. Call `pdd --debug` to see the JavaScript error and where it fails.
4. Patch the code while r2 is running.
5. Call `pdd --debug` to see the JavaScript error is gone.
6. Commit and submit a PR.

Demo

r2dec design

Typical Decompiler Design

- each architecture requires a plugin
- each architecture has its own analysis loop
- each architecture has its own way of parsing machine code or assembly
- tries to optimize the output from the machine/assembly code
- often also does some deobfuscation

This was also r2dec design before the 1st core rewrite

r2dec design

- Javascript based (running on duktape js engine)
  > Avoids recompilation and it is great with strings
- Doesn’t parse any binaries, just use JSON data from r2
- Generic operations that can be used on any architecture
  > add, subtract, multiply, divide, rotate left, etc..
- Generic control flow analysis
  > One analysis to rule them all
- Optimizations are done mostly via the generic operations
  > You add a new architecture and you get 80% of the features including optimizations
- Issues can be reproduced via JSON maps provided by r2dec itself (`pddi`)
  > No binary sharing required!!!

r2dec internals - generic instruction

The machine code usually looks like:

```
<mnemonic> <operand0> <operand1> (etc..)
```

A generic operation is a common instruction (available everywhere), that can be generalized.

For example:

```
; intel x86 asm
; rax = rax + 10
add rax, 10
; memn op0 op1

// Generic  DST  A    B
   Base.add(op0, op0, op1)
// outputs rax += 10
```

```
; renesas v850 asm
; r7 = r10 ^ 0x28
xori 0x28, r10, r7
; memn op0 op1 op2

// Generic  DST  A    B
   Base.xor(op2, op1, op0)
// outputs r7 = r10 ^ 0x28
```

```
; mips asm
; $s0 = $t3 & 0
and $s0, $t3, $zero
; memn op0 op1 op2

// Generic  DST  A    B
   Base.and(op0, op1, op2)
// outputs $s0 = 0
```

`Base` is the r2dec object that abstracts a "*generic instruction*"

r2dec internals - optimizations

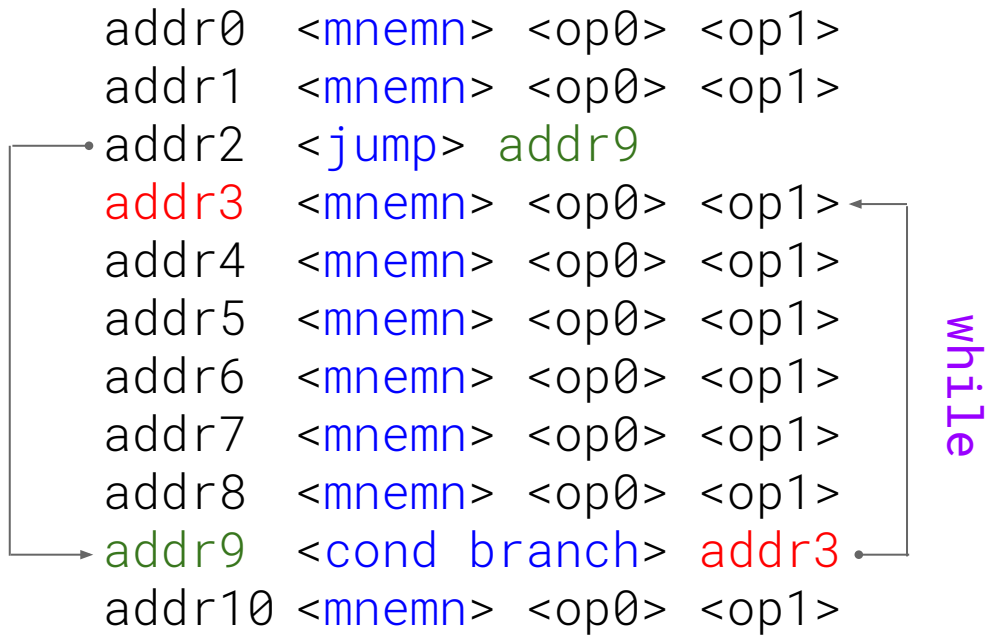
As said a generic operation is a common instruction (available everywhere), that can be generalized; for example:

```
Base.add(DST, A, B):

    If DST == A && B == 1:
        return "DST ++"
    Else If DST == A:
        return "DST += B"
    return "DST = A + B"
```

```
Base.xor(DST, A, B):

    If A == B:
        return "DST = 0"
    Else If A == 0:
        return "DST = B"
    Else If B == 0:
        return "DST = A"
    return "DST = A ^ B"
```

r2dec internals - control flow (while)

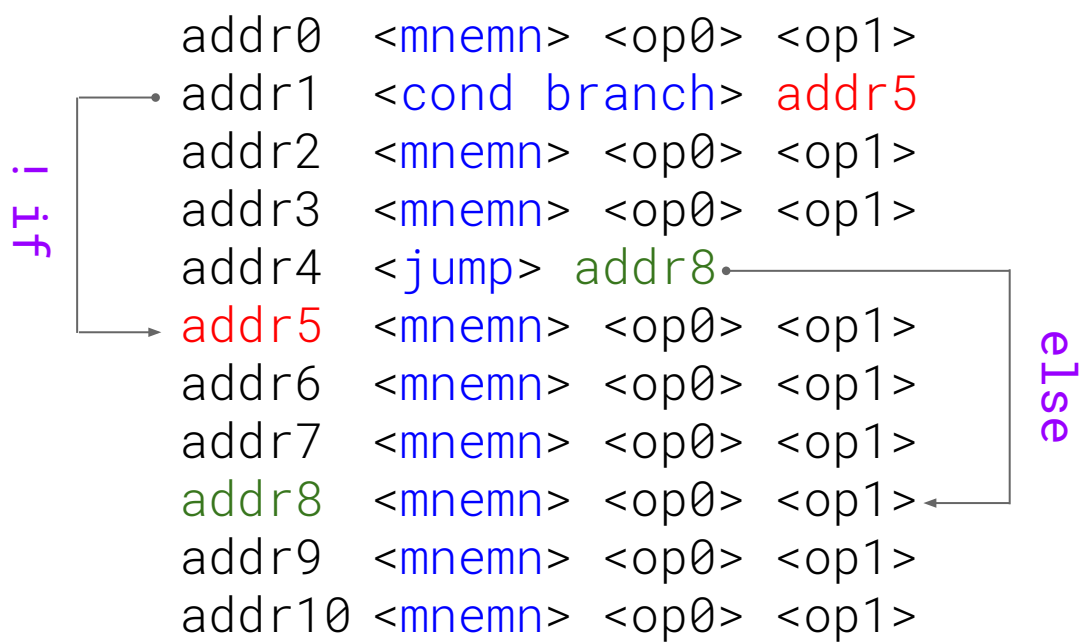
`while` loops are always jumping back and conditions are defined as `do{}while` loops.



The `addr9` conditional branch can be also an **unconditional jump**

r2dec internals - control flow (if/else)

`if/else` conditions jump ahead and conditions are usually negated in the assembly code.



Extending r2dec

r2dec paths

r2dec is usually installed under:

```
R2DEC_PATH=$HOME/.local/share/radare2/r2pm/git/r2dec-js
```

The required files/folders can be found under:

```
$R2DEC_PATH/libdec/arch/      | r2dec architectures folder
$R2DEC_PATH/libdec/archs.js   | r2dec supported architectures map
```

The new architecture shall be placed under the *$R2DEC_PATH/libdec/arch/* folder and shall be mapped under the *$R2DEC_PATH/libdec/archs.js* (it shall match the output of `e asm.arch` output on radare2)

Before starting developing

Hints and the js template needed for the new architecture can be found at:

`https://github.com/wargio/r2dec-js/blob/master/DEVELOPERS.md`

Important things to know:

- r2dec disallow the usage of the following functions outside the `libdec/core/r2util.js` file.
  - `___internal_*()`
  - `r2cmd()`
- r2dec allows pre and post analysis modifications of the data.
  > Useful especially for those architectures that has delayed branch pipeline.
- r2dec allows the usage of a '`context`' object to keep memory of specific informations during the analysis of the data that might be needed later.

Template explanation (1)

```
module.exports = (function() {
    const Base = require('libdec/core/base');
    const Variable = require('libdec/core/variable');
    const Extra = require('libdec/core/extra');
    return {
        preanalisys: function(instructions, context) {},
        postanalisys: function(instructions, context) {},
        localvars: function(context) {
            return [];
        },
        globalvars: function(context) {
            return [];
        },
        arguments: function(context) {
            return [];
        },
        returns: function(context) {
            return 'void';
        },
[...]
```

Generics and Utils

Pre/Post analysis functions

Local variables definition

Global variables definition

Routine arguments definition

Routine return type definition

Template explanation (2)

```
[...]
        instructions: {
            add: function(instr, context, instructions) {
                var opd = instr.parsed.opd;
                return Base.add(opd[0], opd[1], opd[2]);
            },
            nop: function() {
                return Base.nop();
            }
        },
        parse: function(assembly) {
            var tokens = assembly.trim().split(' ');
            return { mnem: tokens.shift(), opd: tokens };
        },
        context: function() {
            return { cond: { a: '?', b: '?' } };
        },
    };
})();
```

Instructions

Instructions parsing function

New architecture context function

Instruction Object

```
Instruction.location {Long object}
     contains the current program counter address
Instruction.jump {Long object}
     contains the instruction jump address
Instruction.assembly {String}
     contains the original assembly code
Instruction.comments {Array of Strings}
     is an array of strings that will be printed as a comment
Instruction.code {Base Object}
     contains the result of a Base.* function.
Instruction.parsed {Parse Object}
     contains the parsed assembly code
Instruction.valid {Boolean}
     is a boolean used to define if it should be printed or not.
Instruction.string {String}
     contains the string that the instruction xref to.
Instruction.conditional (A,B,COND) {Function}
     is used to store a conditional value which output will be <ctrlflow> (A COND B)
Instruction.setBadJump () {Function}
     can be used to invalidate Instruction.jump value.
```

Useful instruction data to use when improving r2dec

Complex instruction handling

Example of complex instruction handled via *Base.composed(array_of_ops)*

PPC RLWIMI := Rotate Left Word Immediate then Mask Insert

```
// rlwimi <destination>, <source>, <shift>, <maskbegin>, <maskend>
// rlwimi = (dst & ~mask) | (rotate_left_32_bit(src, sh) & mask)

var _rlwimi = function(dst, src, sh, mb, me) {
     var mask = mask32(mb, me);    // mask = ppcbitmask(mb, me)
     var minv = mask32inv(mb, me); // ~mask
     var ops = [];
     var value0 = Variable.uniqueName('local_');// returns a unique variable name
     var value1 = Variable.uniqueName('local_');// like local_XXX where XXX is a number
     ops.push(Base.rotate_left(value0, src, sh, 32));
     ops.push(Base.and(value0, value0, '0x' + mask.toString(16)));
     ops.push(Base.and(value1, dst, '0x' + minv.toString(16)));
     ops.push(Base.or(dst, value1, value0));
     return Base.composed(ops);
};
```

Questions?

Github: https://github.com/wargio/r2dec-js

Twitter: @der0ad

Backup

Project Folder Structure

```
r2dec-js
|  git repository/root folder
├── libdec
|    |  contains r2dec libs/deps/interfaces/etc..
|    ├── arch
|    |       contains all the architectures supported by r2dec
|    ├── colors
|    |       r2dec color library to print colors on screen
|    ├── core
|    |       r2dec core files (base.js, extra.js, variables.js, etc..)
|    └── db
|             database folder for known functions (C, Cpp, macros, etc..)
├── p
|       contains the C code to interface with r2/js and the duktape core.
├── r2dec-duk.js
|       r2dec main function
├── r2dec-test.js
|       r2dec-regressions main function
└── themes
         theme folder for r2dec
```