Plugin development and ESIL

...

SkUaTeR (@sanguinawer)

r2con 2k16 BCN

Agenda

- Introduction
- Get repos & build environment
- ESIL
- Reg Profile
- Plugin Anatomy
- Asm plugins
- Analysis plugins
- IO plugins
- Debug plugins
- Using some plugins:
  - baleful
  - bochs
  - r2k

Introduction

- Why we want to make plugins for r2?
  - Implement new archs
  - Implement new tools / features
  - Make daily work easier

- We need to know
  - C/C++
  - A bit of r2 internals
  - and know what we want to do ...

Introduction

- R2 has many types of plugins:
  - ANALyzer
  - ASM
  - DEBUG
  - IO
  - FS
  - BIN
  - CORE
  - EGG

Introduction

- R2 has a powerful internal API
  - Reg Profile
  - Rutil
  - Conversion routines
  - SDB

This APIs give us the ability to access the r2 internal logic.

r2 is made by a tons of plugins and all plugins are invoked by r_core, this layer of r2 is responsible about select the correct backend to manage the desired architecture, debugger, analyzer , ...

Introduction

- R2 has internal plugins and external plugins but always have same logic.
- External plugins are builded as independent library (dll, o, ...)
- Internal plugins are builded as shared library (libr_bin, libr_asm, ...)

We will learn how to build both type of plugins.

But first let's do some preparations

Get repos && Build Environment

...

Get repos && Build Environment

- I’m a windows guy (plz not hate me and apologize ;))
- I'm alway build from linux and use the cross compiling scripts for other archs.

We need:

- Git
- Radare2
- gcc toolchain
- mingw32-64 (for windows release)

(i hope everyone has ever done a build of r2 on computers that you have brought)

Get repos && Build Environment

Getting Radare2 (the main repo):

```
git clone https://github.com/radare/radare2
```

Getting Radare2 Extras (extra plugins and goodies):

```
git clone https://github.com/radare2/radare2-extras
```

Getting Radare2 Regressions (test suite, to ensure dont break all the project and avoid pancake & minions enter in enrage, if u made and add tests to this repo, r2 team pay to you with tons of love)

```
git clone https://github.com/radare2/radare2-regressions
```

Get repos && Build Environment

Testing for a correct build:

```
cd radare2
sys/install.sh      ← build and install into system (for *nix based systems)
sys/mingw32.sh  ← cross compiling win32 release
sys/mingw64.sh  ← cross compiling win64 release
```

As I commented above, I usually work under linux and would highly recommended for this training work under linux because is more flexible to do the cross compilations builds.

ESIL

...

ESIL

Evaluable Strings Intermediate Language, it aims to describe a Forth-like representation for every target opcode semantics. (ESIL WIKI)

Is virtual machine based on stack. (LIFO)

Examples of ESIL expressions:

sf,!,?{,0x1019,pc,=,}	if (!sf) pc =0x1019
eax,ebx,=	ebx=eax
eax,ebx,^,ebx	ebx=ebx ^ eax

0,sf,=,r_03,r_01,<,sf,=,0,zf,=,r_03,r_01,==,$z,zf,=,0,gf,=,r_03,r_01,>,gf,=	sf = 0; if(r_01 < r_03) sf = 1;
	zf = 0; if (r1 == r_3) zf = 1;
	gf = 0; if (r_01 > r_03) gf = 1;

ESIL Cheat Sheet

ESIL Opcode	Operands	Name	Operation
TRAP	src	Trap	Trap signal
$	src	Syscall	sysccall
$$	src	Instruction address	Get address of current instruction stack=instruction address
==	src,dst	Compare	v = dst - src ; update_eflags(v)
<	src,dst	Smaller	stack = (dst < src)
<=	src,dst	Smaller or Equal	stack = (dst <= src)
>	src,dst	Bigger	stack = (dst > src)
>=	src,dst	Bigger or Equal	stack = (dst > src)
<<	src,dst	Shift Left	stack = dst << src
>>	src,dst	Shift Right	stack = dst >> src
<<<	src,dst	Rotate Left	stack=dst ROL src
>>>	src,dst	Rotate Right	stack=dst ROR src
&	src,dst	AND	stack = dst & src
|	src,dst	OR	stack = dst | src
^	src,dst	XOR	stack = dst ^src
+	src,dst	ADD	stack = dst + src
-	src,dst	SUB	stack = dst - src
*	src,dst	MUL	stack = dst * src

ESIL Opcode	Operands	Name	Operation
/	src,dst	DIV	stack = dst / src
%	src,dst	MOD	stack = dst % src
!	src	NEG	stack = !!!src
++	src	INC	stack = src++
--	src	DEC	stack = src--
+=	src,reg	ADD eq	reg = reg + src
-=	src,reg	SUB eq	reg = reg - src
*=	src,reg	MUL eq	reg = reg * src
/=	src,reg	DIV eq	reg = reg / src
%=	src,reg	MOD eq	reg = reg % src
<<=	src,reg	Shift Left eq	reg = reg << src
>>=	src,reg	Shift Right eq	reg = reg << src
&=	src,reg	AND eq	reg = reg & src
|=	src,reg	OR eq	reg = reg | src
^=	src,reg	XOR eq	reg = reg ^ src
++=	reg	INC eq	reg = reg + 1
--=	reg	DEC eq	reg = reg - 1
!=	reg	NOT eq	reg = !reg
---	---	---	---
=[]	src,dst	poke	*dst=src
[]	src	peek	stack=*src

ESIL Internal VARS

ESIL have some internals variables to manage machine states how EFLAGS.
Theses variables are read only and can be accessed using ‘$’ prefix.

- $z Zero
- $c Carry
- $p Parity
- $s Sign
- $b Borrow
- $r regsize

ESIL Control Flow

ESIL have few instruction to manage flow into ESIL expression

- `n, SKIP = skip n instructions`
- `n, GOTO = got instruction n`
- `BREAK  = stop evaluating expresion`
- `LOOP  = alias for 0,GOTO`
- `TODO  = stop emulation and print ins value.`
- `?{,  = ESIL if`

`rep movsb byte`	`if !cx break;`
	`5: *di=*si;`
	` if (df) {`
	`  si--;di--`
	`} else {`
	`  si++;di++};`
	` cx--;`
	`if (cx) goto 5`
		`cx,!,?{,BREAK,},`
		`si,[1],di,=[1],`
		`df,?{,1,si,-=,1,di,-=,},`
		
		`df,!,?{,1,si,+=,1,di,+=,},`
		
		`cx,--=,`
		`cx,?{,5,GOTO,}`

ESIL Enviroment

```
asm.bits       - Set bits to work (8,16,32,64)
asm.esil       - Show ESIL representation into disassembler view (true/false)
                    •  Capital o “O” into Visual Mode
asm.emu        - Show Regs value into disassembler view (true/false)
cmd.esil.intr  - Execute command when intr is executed (#!pipe node script.js)
cmd.esil.trap  - Execute command when trap is executed (#!pipe python script.py)
esil.gotolimit - Maximum ESIL instructions executed in a single ESIL Expression
```

```
$ r2 -
[0x00000000]> e asm.bits = 32
[0x00000000]> aer
..
..
[0x00000000]> e asm.bits = 64
[0x00000000]> aer
..
..
```

```
$ r2 /bin/ls
[0x00000000]> e asm.esil =
true
[0x00000000]> pd 2
0x004049a2 rdx,r9,=
0x004049a5 rsp,[8],rsi,=,8,rsp,+=
[0x00000000]> Vpp
Press key O
Press key :
:>e asm.emu = true
```

ESIL COMMANDS

```
aei  - Initialize ESIL VM (analysis ESIL initialize)
aeim - Initialize ESIL Memory (analysis ESIL initialize memory)
aeip - Set ESIL program counter to cursek
    dr PC = $$  | ar PC = $$ | aer PC = $$
```

```
[0x00000000]> e asm.bits = 32
[0x00000000]> aei
[0x00000000]> aeim
[0x00000000]> o
- 4522416 malloc://512 @ 0x0 ; rw size=512
- 5743248 malloc://983040 @ 0x100000 ; rw
size=983040
[0x00000000]> aeim 0x200000 0xffff
[0x00000100]> o
- 3773328 malloc://512 @ 0x0 ; rw size=512
- 3933136 malloc://983040 @ 0x100000 ; rw
size=983040
- 3893776 malloc://65535 @ 0x200000 ; rw size=65535
```

```
[0x00000100]> s 0x100
[0x00000100]> dr eip
0x00000000
[0x00000100]> aeip
[0x00000100]> dr eip
0x00000100
[0x00000100]> dr PC = $$+1
0x00000101
[0x00000100]>
```

```
[0x00000100]> ar eip
0x00000101
[0x00000100]> ar PC = $$+2
[0x00000100]> ar eip
0x00000102
```

ESIL COMMANDS

aer | ar | dr - Show or modify ESIL registry

```
[skuater@leosku ~]$ r2 -
[0x00000000]> e asm.bits = 32
[0x00000000]> ar eax = 1
[0x00000000]> ar eax
0x1
[0x00000000]> dr ebx=1
[0x00000000]> ar
oeax = 0x00000000
eax = 0x00000001
ebx = 0x00000001
ecx = 0x00000000
edx = 0x00000000
esi = 0x00000000
edi = 0x00000000
esp = 0x00000000
ebp = 0x00000000
eip = 0x00000000
eflags = 0x00000000
```

ESIL COMMANDS

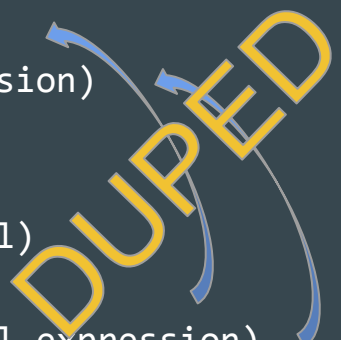
ae - Evaluate ESIL expresion

```
[skuater@leosku ~]$ r2 -
-- ESIL ruined my life
[0x00000000]> e asm.bits = 32
[0x00000000]> ae 1,1,-
0x0
[0x00000000]> ae 1,1,+
0x2
[0x00000000]> ar ebx = 0; ar eax = 0; ar eax; ar ebx
0x00000000
0x00000000
[0x00000000]> ae eax,?{,0x90909090,ebx,=,}; ar eax; ar ebx
0x00000000
0x00000000
[0x00000000]> ar eax = 1; ar ebx = 1; ae eax,?{,0x90909090,ebx,=,}; dr eax; dr ebx
0x00000001
0x90909090
[0x00000000]>
```

ESIL COMMANDS

```
aes   - Single Step  (analysis ESIL step)
aeso  - Single Step Over (analysis ESIL step over)
aesu  - Step until address (analysis ESIL step until)
aesue - Step until ESIL expression (analysis ESIL step until expression)

aec   - Continue until exception (analysis ESIL continue)
aecs  - Continue until syscall (analysis ESIL continue until syscall)
aecu  - Continue until address  (analysis ESIL continue until)
aecue - Continue until Esil expression (analysis ESIL continue until expression)
```



Demo Rovnix

Layer 1: XOR decipher	Fix prefix into decipher code	Layer 2: APACK decipher
r2 -b 16 particionAEscribir.dmp e io.cache=true aeim 0x2000 0xffff aer ds=0x7000 aer cs=0x7000 aer es=0x8000 s 0x7a aeip aesu 0x81 aer ax=0x5000 aesu 0x5a5 aer cx=0x9000 aesu 0x572 aer di=0x9000 aesu 0x57d aesu 0x56f	wx 0x90@0x98d0;wx 0x90@0x98da wx 0x90@0x98e2;wx 0x90@0x98ea wx 0x90@0x9901;wx 0x90@0x9906 wx 0x90@0x990f;wx 0x90@0x9914 wx 0x90@0x991c;wx 0x9090@0x9920 wx 0x90@0x9929;wx 0x90@0x992f wx 0x90@0x9931;wx 0x90@0x9936 wx 0x90@0x9938;wx 0x90@0x993a wx 0x90@0x9943;wx 0x90@0x998e wx 0x90@0x9993;wx 0x90@0x98c0 wx 0x90@0x98c2;wx 0x90@0x98c4 wx 0x90@0x98c9;	aer sp=sp+2 aer bp=bp+0x9000 aer ip=0x9100 e esil.gotolimit=0xfffff aesu 0x98d0 aer di=0xa000 aer bx=0 aesu 0x9943 7 minutes in i7

Demo Baleful

```
r2 -a baleful vm.cifrada.code
e asm.bits=32
e io.cache = true
aei
aeim 0x200000 0x1024
aer r_data=0x200000
s 0x1000
aeip
aesu 0x1843
aesue 0x001e,r_00,=
aesue0x103a,pc,=
```

r2pipe && ESIL

- r2pipe is a API for different languages, this bring us a way to send command to radare and get the result.
- Don't is needed know r2 internal API.
- Can be use to made scripts or plugins via rlang
- More info at:

  https://github.com/jpenalbae/r2-scripts/blob/master/workshop-slides/r2pipe-Automating_binary_analysis_with_radare-NNC5ed.pdf

Demo r2pipe && msfdecoder

Python example:

```
import os
import r2pipe

r2 = r2pipe.open("/bin/ls")
print (r2.cmd("pd 10"));
r2.quit()
```

MSFDECODER:

```
python msfdecoder.py origen destino
```

REG PROFILE

...

REG Profile (RREG)

- Reg profile is used to define architecture registers
- This profile is defined as string

```
static int set_reg_profile(RAnal *anal) {
        const char *p = \
                        "=pc    pc\n"
                        "=sp    stk\n"
                        "gpr    pc      .32 0   0\n"
                        "gpr    stk     .32 4   0\n"
                        "gpr    zf      .32 8   0\n"
                        "gpr    sf      .32 12  0\n"
                        "gpr    gf      .32 16  0\n"
                        "gpr    r_00    .32 20  0\n"
                        "gpr    r_01    .32 24  0\n"
                        "gpr    r_02    .32 28  0\n"
                        "gpr    r_03    .32 32  0\n"
                        "gpr    r_04    .32 36  0\n"
                        "gpr    r_05    .32 40  0\n"
                        "gpr    r_06    .32 44  0\n"
                        "gpr    r_07    .32 48  0\n"
                        "gpr    r_08    .32 52  0\n"
                        "gpr    r_09    .32 56  0\n"
                        "gpr    r_0a    .32 60  0\n"
                        "gpr    r_0b    .32 64  0\n"
                        "gpr    r_0c    .32 68  0\n"
                                                        550,4-25      87%
```

REG Profile fields

- Type = gpr|drx|fpu|mmx|xmm|flg|seg

Type Of reg	Defined Name into arena	Size of reg into arena	Offset into arena	Internal	end line
drx	dr0	.32	4	0	\n
seg	gs	.32	140	0	\n
gpr	edi	.32	156	0	\n

- Alias = PC|SP|SR|BP|LP|A0-A6|R0-R3|ZF|SF|CF|OF|SN

Alias Name	Arena register	end line
=PC	EIP	\n
=SP	ESP	\n

REG Profile Example

```
static char *r_debug_bf_reg_profile(RDebug *dbg) {
        return strdup (
        "=PC   pc\n"
        "=SP   esp\n"
        "=BP   ptr\n"
        "gpr   pc     .32    0      0\n"
        "gpr   ptr    .32    4      0\n"
        "gpr   esp    .32    8      0\n"
        "gpr   scr    .32    12     0\n"
        "gpr   scri   .32    16     0\n"
        "gpr   inp    .32    20     0\n"
        "gpr   inpi   .32    24     0\n"
        "gpr   mem    .32    28     0\n"
        "gpr   memi   .32    32     0\n"
        );
}
```

```
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F
0123456789ABCDEF
0x00000000  0000 0000 0000 0000 0000 0000 0000 0000
0x00000010  0000 0000 0000 0000 0000 0000 0000 0000
0x00000020  0000 0000
```

PLUGIN ANATOMY

...

Plugin Anatomy

```
RDebugPlugin r_debug_plugin_native= {

}

struct r_lib_struct_t radare_plugin = {
        .type = R_LIB_TYPE_DBG,
        .data = &r_debug_plugin_native,
        .version = R2_VERSION
};
```

Tipo Plugin	Enum Name	Data Struct
io layer	R_LIB_TYPE_IO	RIOPlugin
debugger	R_LIB_TYPE_DBG	RDebugPlugin
assembler	R_LIB_TYPE_ASM	RAsmPlugin
analysis	R_LIB_TYPE_ANAL	RAnalPlugin

Tipo Plugin	Enum Name	Data Struct
language	R_LIB_TYPE_LANG	
parsers	R_LIB_TYPE_PARSE	
bins	R_LIB_TYPE_BIN	
bin extractors	R_LIB_TYPE_BIN_XTR	
breakpoint	R_LIB_TYPE_BP	
syscall	R_LIB_TYPE_SYSCALL	
fastcall	R_LIB_TYPE_FASTCALL	
cryptography	R_LIB_TYPE_CRYPTO	
RCore commands	R_LIB_TYPE_CORE	
r_egg plugin	R_LIB_TYPE_EGG	
r_fs plugin	R_LIB_TYPE_FS	

Plugin Anatomy: Define ASM Plugin

Plugin Definition:	Data Definition:	RAsmPlugin Fields:

`struct r_lib_struct_t radare_plugin = {`
`  .type = R_LIB_TYPE_ASM,`
`  .data = &r_asm_plugin,`
`  .version = R2_VERSION`
`};`
	RAsmPlugin r_asm_plugin = {
	  .name = "baleful",
	  .arch = "baleful",
	  .license = "LGPL3",
	  .bits = 32,
	  .desc = "Baleful",
	  .disassemble = &disassemble,
	  .assemble = &assemble,
	};
		`const char *desc;`
		`const char *license;`
		`void *user; // user data pointer`
		`int bits;`
		`int endian;`
		`bool (*init)(void *user);`
		`bool (*fini)(void *user);`
		
		`int (*disassemble)(RAsm *a, RAsmOp *op, const ut8 *buf, int len);`
		
		`int (*assemble)(RAsm *a, RAsmOp *op, const char *buf);`
		
		`RAsmModifyCallback modify;`
		`int (*set_subarch)(RAsm *a, const char *buf);`
		`const char *features;`

Plugin Anatomy: Define Analysis Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_ANAL,`
`.data = &r_anal_plugin,`
`.version = R2_VERSION`
`};`
	RAnalPlugin r_anal_plugin = {
	.name = "baleful",
	.desc = "baleful code analysis plugin",
	.license = "LGPL3",
	.arch = "baleful",
	.bits = 32,
	.esil_init = esil_baleful_init,
	.esil_fini = esil_baleful_fini,
	.esil_intr = esil_baleful_intr,
	.esil = true,
	.op = &baleful_op,
	.set_reg_profile = set_reg_profile,
	};

Plugin Anatomy: Define Debug Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_DBG,`
`.data = &r_debug_plugin,`
`.version = R2_VERSION`
`};`
	RDebugPlugin r_debug_plugin = {
	.name = "bochs",
	.license = "LGPL3",
	.arch = "x86",
	.bits = R_SYS_BITS_16 | R_SYS_BITS_32 | R_SYS_BITS_64,
	.step = r_debug_bochs_step,
	.cont = r_debug_bochs_continue,
	.attach = &r_debug_bochs_attach,
	.detach = &r_debug_bochs_detach,
	.canstep = 1,
	.stop = &r_debug_bochs_stop,
	.wait = &r_debug_bochs_wait,
	.map_get = r_debug_bochs_map_get,
	.breakpoint = &r_debug_bochs_breakpoint,
	.reg_read = &r_debug_bochs_reg_read,
	.reg_write = &r_debug_bochs_reg_write,
	.reg_profile = (void *)r_debug_bochs_reg_profile,
	};

Plugin Anatomy: Define IO Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_IO,`
`.data = &r_io_plugin,`
`.version = R2_VERSION`
`};`
	RIOPlugin r_io_plugin = {
	.name = "bochs",
	.desc = "Attach to a BOCHS debugger",
	.license = "LGPL3",
	.open = __open,
	.close = __close,
	.read = __read,
	.write = __write,
	.check = __plugin_open,
	.lseek = __lseek,
	.system = __system,
	.isdbg = true
	};

Plugin Anatomy: Paths to plugins

Debug Plugins in core:
    radare2/libr/debug/p

Analysis Plugins in core:
    radare2/libr/anal/p

IO Plugins in core:
    radare2/libr/io/p

ASM Plugins in core:
    radare2/lib/asm/p

Time to take a look to some of these paths ….

Best to learn is learn from the best A tiny plugin to inspect can be Brain Fuck they show all the magic and was coded by Pancake they is the best in this , r2 is their other baby.

ASM PLUGINS

...

ASM Plugins: Define ASM Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`  .type = R_LIB_TYPE_ASM,`
`  .data = &r_asm_plugin_myarch,`
`  .version = R2_VERSION`
`};`
	`RAsmPlugin r_asm_plugin_myarch  = {`
	`  .name = "",`
	`  .arch = "",`
	`  .license = "",`
	`  .desc = "",`
	`  .bits = 32,`
	`  .init = 0,`
	`  .fini = 0,`
	`  .disassemble = &disassemble,`
	`  .assemble = &assemble,`
	`};`

ASM Plugins: radare2/libr/asm/p/asm_myarch.c

```
#include <stdio.h>
#include <string.h>
#include <r_types.h>
#include <r_lib.h>
#include <r_asm.h>
static int disassemble(RAsm *a, RAsmOp *op, const ut8 *buf, int len) {
        //TODO
}
static int assemble(RAsm *a, RAsmOp *op, const char *buf) {
        //TODO
}
RAsmPlugin r_asm_plugin_myarch = {
        .name = "myarch",
        .arch = "myarch",
        .license = "LGPL3",
        .bits = 32,
        .endian = R_SYS_ENDIAN_NONE,
        .desc = "my arch skeleton",
        .disassemble = &disassemble,
        .assemble = &assemble
};
#ifndef CORELIB
struct r_lib_struct_t radare_plugin = {
        .type = R_LIB_TYPE_ASM,
        .data = &r_asm_plugin_myarch,
        .version = R2_VERSION
};
#endif
```

ASM Plugins: radare2/libr/asm/p/myarch.mk

```
OBJ_MYARCH=asm_myarch.o

TARGET_MYARCH=asm_myarch.${EXT_SO}
STATIC_OBJ+=${OBJ_MYARCH}

ifeq ($(WITHPIC),1)
ALL_TARGETS+=${TARGET_MYARCH}
${TARGET_MYARCH}: ${OBJ_MYARCH}
    ${CC} $(call libname,asm_myarch) ${LDFLAGS} ${CFLAGS} -o ${TARGET_MYARCH} ${OBJ_MYARCH}
endif
```

ASM Plugins: radare2/libr/asm/p/Makefile

Edit Makefile and add myasm.mk

```
include ../../config.mk
include ../../../mk/platform.mk
CFLAGS+=-I$(TOP)/shlr -I../../include -I../arch/ -I../arch/include
CFLAGS+=-Wall -shared ${PIC_CFLAGS} ${LDFLAGS_LIB} ${LDFLAGS_LINKPATH}..
LDFLAGS+=-L../../util -L../../lib -L../../flags
LDFLAGS+=${LINK} -lr_flags -lr_util
CURDIR=
foo: all
ALL_TARGETS=
# TODO: rename to enabled plugins
ARCHS=mips_gnu.mk x86_cs.mk sparc_cs.mk sparc_gnu.mk java.mk bf.mk arm_gnu.mk dalvik.mk
ARCHS+=x86_as.mk x86_nz.mk cris_gnu.mk vax.mk
ARCHS+=ppc_gnu.mk ppc_cs.mk x86_olly.mk x86_udis.mk xap.mk x86_nasm.mk avr.mk
ARCHS+=sh.mk arm_winedbg.mk tms320.mk gb.mk snes.mk ebc.mk malbolge.mk ws.mk
ARCHS+=6502.mk h8300.mk cr16.mk v850.mk spc700.mk propeller.mk msp430.mk i4004.mk z80_cr.mk
ARCHS+=lh5801.mk v810.mk mcs96.mk lm32.mk myarch.mk
ARCHS+=riscv.mk rsp.mk
include $(ARCHS)
all: ${ALL_TARGETS}
clean:
        -rm -f *.${EXT_SO} *.o ${STATIC_OBJ}
```

ASM Plugins: radare2/libr/include/r_asm.h:

Edit libr/include/r_asm.h and define a extern with the new plugin:

RAsmPlugin r_asm_plugin_myasm

```
...
...
extern RAsmPlugin r_asm_plugin_riscv;
extern RAsmPlugin r_asm_plugin_vax;
extern RAsmPlugin r_asm_plugin_lanai_gnu;
extern RAsmPlugin r_asm_plugin_xtensa;
extern RAsmPlugin r_asm_plugin_tricore;
extern RAsmPlugin r_asm_plugin_pic18c;
extern RAsmPlugin r_asm_plugin_rsp;
extern RAsmPlugin r_asm_plugin_myarch;
#endif

#ifdef __cplusplus
}
#endif

#endif
```

ASM Plugins: radare2/plugins.def.cfg

Edit plugins.def.cfg and add asm.myasm

```
STATIC="anal.8051
anal.arc
anal.arm_cs
anal.arm_gnu
anal.avr
...
...
asm.pic18c
asm.myarch
bin.any
bin.art
bin.bf
...
parse.x86_pseudo"
SHARED="io.shm"
```

Plugin Task 1: Build ASM Plugin Skeleton

```
1)  Take a look to libr/asm/p/asm_bf.c
2)  implement a new asm plugin skeleton called myarch.

Remember u must add/edit all theses files:
     radare2/libr/asm/p/asm_myarch.c
     radare2/libr/asm/p/myarch.mk
     radare2/libr/asm/p/Makefile
     radare2/libr/include/r_asm.h
     radare2/plugins.def.cfg
```

ASM Plugins: Basic callbacks

```
static int disassemble(RAsm *a, RAsmOp *op, const ut8 *buf, int len) {
        ut32 *dw;
        int size = 1;
        strcpy(op->buf_asm,"SUPER OPCODE");
        op->size = size;
        return size;
}
```

RAsmOp members:

```
typedef struct r_asm_op_t {
        int size; // instruction size
        int payload; // size of payload (opsize = (size-payload))
        ut8  buf[R_ASM_BUFSIZE + 1];     // bytes opcode
        char buf_asm[R_ASM_BUFSIZE + 1]; // representation opcode
        char buf_hex[R_ASM_BUFSIZE + 1]; // ????
} RAsmOp;
```

ut8 buf : Receive a buffer with opcodes bytes to interpreter

ASM Plugins: Basic callbacks

```
static int assemble(RAsm *a, RAsmOp *op, const char *buf) {
        int n = 0;
        if (!strncmp (buf, "r1 = 0", 6)) {
                op->buf[0] = 0x01;
                n = 1;
          }
        return n;
}
```

ut8 buf : Receive a buffer with string representation of opcode to encode

Plugin Task 2: Implement this arch for disassembly

Opcode	Param	Representation	Pseudo
0x00	1 DWORD (4 bytes)	LDR stk,DWORD	stk = 32 BITS VALUE
0x01	NONE	LDR r1,stk	r1 = stk
0x02	NONE	LDR r2,stk	r2 = stk
0x03	NONE	ADD stk,r1,r2	stk = r1 + r2
0x04	NONE	SUB stk,r1,r2	stk = r1 - r2
0x05	NONE	PRINT stk	print value of stk in ascii

Plugin Task 3: Implement assembler callback

ANALYSIS PLUGINS

...

Analysis Plugins: Define Analysis Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_ANAL,`
`.data = &r_anal_plugin_myarch,`
`.version = R2_VERSION`
`};`
	RAnalPlugin r_anal_plugin_myarch = {
	.name = "myarch",
	.desc = "myarch code analysis plugin",
	.license = "LGPL3",
	.arch = "myarch",
	.bits = 32,
	.esil = true,
	.op = &myanal_op,
	.get_reg_profile = get_reg_profile,
	.set_reg_profile = set_reg_profile,
	
	};

Analysis Plugins: radare2/libr/anal/p/anal_myarch.c

```
#include <string.h>
#include <r_types.h>
#include <r_lib.h>
#include <r_asm.h>
#include <r_anal.h>

static int myanal_op(RAnal *anal, RAnalOp *op, ut64 addr, const ut8 *buf, int len) {
    ut32 *dw;
    if (op == NULL)
        return 1;
    memset (op, 0, sizeof (RAnalOp));
    op->type = R_ANAL_OP_TYPE_NULL;
    op->delay = 0;
    op->jump = op->fail = -1;
    op->ptr = op->val = -1;
    op->addr = addr;
    op->refptr = 0;
    op->size = 0;
    r_strbuf_setf (&op->esil, "nop");
    op->size = 1;
    break;
    return op->size;
}
```

Analysis Plugins: radare2/libr/anal/p/anal_myarch.c

```
static char *get_reg_profile(RAnal *anal) {
    return strdup (
        "=PC    pc\n"
        "=A0    r1\n"
        "=A1    r2\n"
        "gpr    pc    .32    0    0\n" // program counter
        "gpr    stk   .32    4    0\n" // temp reg
        "gpr    r1    .32    8    0\n" // reg1
        "gpr    r2    .32    12   0\n" // reg2
    );
}

static int set_reg_profile(RAnal *anal) {
    const char *p = \
        "=PC    pc\n"
        "=A0    r1\n"
        "=A1    r2\n"
        "gpr    pc    .32    0    0\n"  // program counter
        "gpr    stk   .32    4    0\n"  // temp reg
        "gpr    r1    .32    8    0\n"  // reg1
        "gpr    r2    .32    12   0\n"; // reg2
    return r_reg_set_profile_string (anal->reg, p);
}
```

Analysis Plugins: radare2/libr/anal/p/anal_myarch.c

```
struct r_anal_plugin_t r_anal_plugin_myarch = {
    .name = "myarch",
    .desc = "myarch code analysis plugin",
    .license = "LGPL3",
    .arch = "myarch",
    .bits = 32,
    .esil = true,
    .op = &myanal_op,
    .get_reg_profile = get_reg_profile,
    .set_reg_profile = set_reg_profile,
};

#ifndef CORELIB
struct r_lib_struct_t radare_plugin = {
    .type = R_LIB_TYPE_ANAL,
    .data = &r_anal_plugin_myarch,
    .version = R2_VERSION
};
#endif
```

Analysis Plugins: radare2/libr/anal/p/myarch.mk

```
OBJ_MYARCH=anal_myarch.o

STATIC_OBJ+=${OBJ_MYARCH}
TARGET_MYARCH=anal_myarch.${EXT_SO}

ALL_TARGETS+=${TARGET_MYARCH}

${TARGET_MYARCH}: ${OBJ_MYARCH}
    ${CC} $(call libname,anal_myarch) ${LDFLAGS} ${CFLAGS} -o anal_myarch.${EXT_SO} ${OBJ_MYARCH}
```

Analysis Plugins: radare2/libr/anal/p/Makefile

```
include ../../config.mk
include ../../../mk/platform.mk
CFLAGS+=-I../../include -I../arch -Wall -shared $(PIC_CFLAGS) ${LDFLAGS_LIB} ${LDFLAGS_LINKPATH}..
CFLAGS+=-L../../util -lr_util -L../../anal -lr_anal -L../../reg -lr_reg
LDFLAGS+=${LINK}
CURDIR=
ifeq ($(WITHPIC),1)
all: ${ALL_TARGETS} ;
ALL_TARGETS=
# TODO: rename to enabled plugins
ARCHS=null.mk x86_udis.mk ppc_gnu.mk ppc_cs.mk arm_gnu.mk avr.mk xap.mk dalvik.mk sh.mk ebc.mk gb.mk malbolge.mk ws.mk
h8300.mk cr16.mk v850.mk msp430.mk sparc_gnu.mk sparc_cs.mk x86_cs.mk cris.mk 6502.mk snes.mk riscv.mk vax.mk xtensa.mk
rsp.mk myarch.mk
include $(ARCHS)
clean:
	-rm -f *.${EXT_SO} *.o ${STATIC_OBJ}
mrproper: clean
	-rm -f *.d ../arch/*/*/*.d
.PHONY: all clean mrproper
else
all clean mrproper:
.PHONY: all clean mrproper
endif
```

Analysis Plugins: radare2/libr/include/r_anal.h:

Edit libr/include/r_anal.h and define a extern with the new plugin:

RAsmPlugin r_anal_plugin_myarch

```
extern RAnalPlugin r_anal_plugin_cris;
extern RAnalPlugin r_anal_plugin_v810;
extern RAnalPlugin r_anal_plugin_6502;
extern RAnalPlugin r_anal_plugin_snes;
extern RAnalPlugin r_anal_plugin_riscv;
extern RAnalPlugin r_anal_plugin_vax;
extern RAnalPlugin r_anal_plugin_i4004;
extern RAnalPlugin r_anal_plugin_xtensa;
extern RAnalPlugin r_anal_plugin_pic18c;
extern RAnalPlugin r_anal_plugin_rsp;
extern RAnalPlugin r_anal_plugin_myarch;
#ifdef __cplusplus
}
#endif

#endif
#endif
```

Analysis Plugins: radare2/plugins.def.cfg

Edit plugins.def.cfg and add anal.myarch

```
STATIC="anal.8051
anal.arc
anal.arm_cs
anal.arm_gnu
asm.myarch
anal.avr
...
...
asm.pic18c
bin.any
bin.art
bin.bf
...
parse.x86_pseudo"
SHARED="io.shm"
```

Plugin Task 1: Build ANAL Plugin Skeleton

1) Take a look to libr/anal/p/asm_bf.c
2) implement a new anal plugin skeleton called myarch.

Remember u must add/edit all theses files:

```
radare2/libr/anal/p/anal_myarch.c
radare2/libr/anal/p/myarch.mk
radare2/libr/anal/p/Makefile
radare2/libr/include/r_anal.h
radare2/plugins.def.cfg
```

Analysis Plugins: Basic callbacks

```
static char *get_reg_profile(RAnal *anal) {
     return strdup (
                "=PC  pc\n"
                "gpr  pc    .32   0     0\n" // program counter
     );
}
static int set_reg_profile(RAnal *anal) {
const char *p = \
                "=PC  pc\n"
            "gpr      pc    .32   0     0\n";  // program counter
     return r_reg_set_profile_string (anal->reg, p);
}
```

Analysis Plugins: Basic callbacks

```
static int myanal_op(RAnal *anal, RAnalOp *op, ut64 addr, const ut8 *buf, int len) {
   if (op == NULL) return 1;
    memset (op, 0, sizeof (RAnalOp));
    op->type = R_ANAL_OP_TYPE_NULL;
    op->delay = 0;
    op->jump = op->fail = -1;
    op->ptr = op->val = -1;
    op->addr = addr;
    op->refptr = 0;
    op->size = 0;
    r_strbuf_setf (&op->esil, "nop");
    op->size = 1;
}
```

ut64 addr: Receive the address of opcode bytes

ut8 buf : Receive a buffer with opcode bytes

Plugin Task 2: Implement this arch for analysis

Opcode	Param	Representation	Pseudo
0x00	1 DWORD (4 bytes)	LDR stk,DWORD	stk = 32 BITS VALUE
0x01	NONE	LDR r1,stk	r1 = stk
0x02	NONE	LDR r2,stk	r2 = stk
0x03	NONE	ADD stk,r1,r2	stk = r1 + r2
0x04	NONE	SUB stk,r1,r2	stk = r1 - r2
0x05	NONE	PRINT stk	print value of stk in ascii

REGISTER	SIZE	DESCRIPTION
PC	32 bits	Instruction Pointer
stk	32 bits	Temporal reg
r1	32 bits	General Purpose
r2	32 bits	General Purpose

IO PLUGINS

...

IO Plugins: Define IO Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_IO,`
`.data = &r_io_plugin,`
`.version = R2_VERSION`
`};`
	`RIOPlugin r_io_plugin = {`
	`.name = "bochs",`
	`.desc = "Attach to a BOCHS debugger",`
	`.license = "LGPL3",`
	`.open = __open,`
	`.close = __close,`
	`.read = __read,`
	`.write = __write,`
	`.check = __plugin_open,`
	`.lseek = __lseek,`
	`.system = __system,`
	`.isdbg = true`
	`};`

IO Plugins: radare2/libr/io/p/io_bochs.c

```
#include <r_io.h>
#include <r_lib.h>
#include <r_util.h>
#include <libbochs.h>

typedef struct {
        libbochs_t desc;
} RIOBochs;

static libbochs_t *desc = NULL;
static RIODesc *riobochs = NULL;
extern RIOPlugin r_io_plugin_bochs; // forward declaration

static bool __plugin_open(RIO *io, const char *file, bool many) {
        return !strncmp (file, "bochs://", strlen ("bochs://"));
}

static RIODesc *__open(RIO *io, const char *file, int rw, int mode) {
        RIOBochs  *riob;
        lprintf("io_open\n");
        const char *i;
        char * fileBochs = NULL;
        char * fileCfg = NULL;
        int l;
        if (!__plugin_open (io, file, 0)) {
                return NULL;
        }
```

IO Plugins: radare2/libr/io/p/io_bochs.c

```
if (r_sandbox_enable (false)) {
                eprintf ("sandbox exit\n");
                return NULL;
        }
        if (riobochs) {
                return riobochs;
        }

        i = strstr (file + 8, "#");
        if (i) {
                l = i - file - 8;
                fileBochs = r_str_ndup (file + 8, l);
                l = strlen (i + 1);
                fileCfg = strdup (i + 1);
        } else {
                free (fileCfg);
                eprintf ("Error cant find :\n");
                return NULL;
        }
        riob = R_NEW0 (RIOBochs);

        // Inicializamos
        if (bochs_open (&riob->desc, fileBochs, fileCfg) == true) {
                desc = &riob->desc;
                riobochs = r_io_desc_new (&r_io_plugin_bochs, -1, file, rw, mode, riob);
                //riogdb = r_io_desc_new (&r_io_plugin_gdb, riog->desc.sock->fd, file, rw, mode, riog);
                free(fileBochs);
```

IO Plugins: radare2/libr/io/p/io_bochs.c

```
free(fileCfg);
                return riobochs;
        }
        lprintf ("bochsio.open: Cannot connect to bochs.\n");
        free (riob);
        free (fileBochs);
        free (fileCfg);
        return NULL;
}
static int __write(RIO *io, RIODesc *fd, const ut8 *buf, int count) {
        lprintf("io_write\n");
        return -1;
}
static ut64 __lseek(RIO *io, RIODesc *fd, ut64 offset, int whence) {
        lprintf("io_seek %016"PFMT64x" \n",offset);
        return offset;
}
static int __read(RIO *io, RIODesc *fd, ut8 *buf, int count) {
        memset (buf, 0xff, count);
        ut64 addr = io->off;
        if (!desc || !desc->data)
                return -1;
        lprintf ("io_read ofs= %016"PFMT64x" count= %x\n", io->off, count);
        bochs_read (desc,addr,count,buf);
        return count;
}
```

IO Plugins: radare2/libr/io/p/io_bochs.c

```
static int __close(RIODesc *fd) {
        lprintf("io_close\n");
        bochs_close (desc);
        return true;
}

static int __system(RIO *io, RIODesc *fd, const char *cmd) {
        eprintf ("system command (%s)\n", cmd);
        if (!strcmp (cmd, "help")) {
                eprintf ("Usage: =!cmd args\n"
                        " =!:<bochscmd>         - Send a bochs command.\n"
                        " =!dobreak             - pause bochs.\n");
                eprintf ("io_system: Enviando comando bochs\n");
                bochs_send_cmd (desc, &cmd[1], true);
                io->cb_printf ("%s\n", desc->data);
                return 1;
        } else if (!strncmp (cmd, "dobreak", 7)) {
                bochs_cmd_stop (desc);
                io->cb_printf ("%s\n", desc->data);
                return 1;
        }
        return true;
}
```

IO Plugins: radare2/libr/io/p/io_bochs.c

```
RIOPlugin r_io_plugin_bochs = {
        .name = "bochs",
        .desc = "Attach to a BOCHS debugger",
        .license = "LGPL3",
        .open = __open,
        .close = __close,
        .read = __read,
        .write = __write,
        .check = __plugin_open,
        .lseek = __lseek,
        .system = __system,
        .isdbg = true
};

#ifndef CORELIB
struct r_lib_struct_t radare_plugin = {
        .type = R_LIB_TYPE_IO,
        .data = &r_io_plugin_bochs,
        .version = R2_VERSION
};
```

IO Plugins: radare2/libr/io/p/bochs.mk

```
OBJ_BOCHS=io_bochs.o
STATIC_OBJ+=${OBJ_BOCHS}
TARGET_BOCHS=io_bochs.${EXT_SO}
ALL_TARGETS+=${TARGET_BOCHS}
LIB_PATH=$(SHLR)/bochs/
CFLAGS+=-I$(SHLR)/bochs/include/
LDFLAGS+=$(SHLR)/bochs/lib/libbochs.a

include $(LIBR)/socket/deps.mk
ifeq (${WITHPIC},0)
LINKFLAGS=../../socket/libr_socket.a
LINKFLAGS+=../../util/libr_util.a
LINKFLAGS+=../../io/libr_io.a
else
LINKFLAGS=-L../../socket -lr_socket
LINKFLAGS+=-L../../util -lr_util
LINKFLAGS+=-L.. -lr_io
endif

${TARGET_BOCHS}: ${OBJ_BOCHS}
	${CC} $(call libname,io_bochs) ${OBJ_BOCHS} ${CFLAGS} \
		${LINKFLAGS} ${LDFLAGS_LIB} $(LDFLAGS)
```

IO Plugins: radare2/libr/io/p/Makefile

```
CFLAGS+=-I../../include -Wall -DWORDSIZE=64 ${PIC_CFLAGS} ${LDFLAGS_LIB} ${LDFLAGS_LINKPATH}.. -DCORELIB

include ../../config.mk

ifeq ($(WITHPIC),1)
# on solaris only
ifeq (${OSTYPE},solaris)
CFLAGS+=-lsocket
endif
# windows
ifeq (${OSTYPE},windows)
CFLAGS+=-lws2_32
endif

foo: all

ALL_TARGETS=
PLUGINS=ptrace.mk debug.mk gdb.mk malloc.mk shm.mk mach.mk w32dbg.mk procpid.mk windbg.mk bochs.mk qnx.mk r2k.mk
#zip.mk
#PLUGINS=ptrace.mk debug.mk gdb.mk malloc.mk mach.mk w32dbg.mk procpid.mk
include ${PLUGINS}
..
..
```

IO Plugins: radare2/libr/include/r_io.h:

Edit libr/include/r_anal.h and define a extern with the new plugin:

RAsmPlugin r_anal_plugin_myarch

```
extern RIOPlugin r_io_plugin_gzip;
extern RIOPlugin r_io_plugin_windbg;
extern RIOPlugin r_io_plugin_r2pipe;
extern RIOPlugin r_io_plugin_r2web;
extern RIOPlugin r_io_plugin_bochs;
extern RIOPlugin r_io_plugin_r2k;
#endif

#ifdef __cplusplus
}
#endif

#endif
```

IO Plugins: radare2/plugins.def.cfg

Edit plugins.def.cfg and add io.bochs

```
fs.ufs
fs.xfs
io.bfdbg
io.bochs
io.debug
io.default
io.gdb
io.qnx
io.r2pipe
io.gzip
```

DEBUG PLUGINS

...

Debug Plugins: Define Debug Plugin

Plugin Definition:	Data Definition:

`struct r_lib_struct_t radare_plugin = {`
`.type = R_LIB_TYPE_DBG,`
`.data = &r_debug_plugin,`
`.version = R2_VERSION`
`};`
	RDebugPlugin r_debug_plugin = {
	.name = "bochs",
	.license = "LGPL3",
	.arch = "x86",
	.bits = R_SYS_BITS_16 | R_SYS_BITS_32 | R_SYS_BITS_64,
	.canstep = 1,
	.step = r_debug_bochs_step,
	.cont = r_debug_bochs_continue,
	.attach = &r_debug_bochs_attach,
	.detach = &r_debug_bochs_detach,
	.stop = &r_debug_bochs_stop,
	.wait = &r_debug_bochs_wait,
	.map_get = r_debug_bochs_map_get,
	.breakpoint = &r_debug_bochs_breakpoint,
	.reg_read = &r_debug_bochs_reg_read,
	.reg_write = &r_debug_bochs_reg_write,
	.reg_profile = (void *)r_debug_bochs_reg_profile,
	};

Debug Plugins: Too many to show correctly here

The debug plugin have too many lines of code, and is needed more of 1 hours to do a fresh plugin.
The steps to build a new plugin are same way of another plugins.

- libr/debug/p/debug_myarch.c
- libr/debug/p/myarch.mk
- libr/debug/p/Makefile
- libr/include/r_debug.h
- plugins.def.cfg

Now we inspect the code from BOCHS debug plugin.

Take a brew and ….

…. May the Force be with you


U KNOW WHAT DAY IT IS?
IT'S THE FINAL COUNTDOWN
memegenerator.net

Plugin Task: The Final CountDown

1) Implement into io_bochs the write callback

Tips:

- Look at shlr/bochs/src/libbochs.c and inspect bochs_read
- bochs command to write memory is: setpmem add size value (size 1,2,4)

2) Implement into debug_bochs the r_debug_bochs_reg_write callback

Tips:

- Look at libr/debug/p/debug_bochs.c and inspect r_debug_bochs_reg_read
- bochs command to write registers is: set regname = value
-

Prepara sistema: UBUNTU x64

apt-get install git
apt-get install mingw-w64
apt-get install vim tmux
apt-get install python-pip nodejs npm
git clone http://github.com/radare/radare2
git clone http://github.com/radare/radare2-extras
git clone http://github.com/radare/radare2-regressions
radare2/sys/install.sh
cd radare2-extras
./configure
cd baleful
make
r2 -hh (to get the plugin dir)
cp -f asm/asm_baleful.so anal/anal_baleful.so ~/.config/radare2/plugins/
pip install r2pipe
npm install syspipe
sudo ln -s /usr/bin/nodejs /usr/bin/node

Build BOCHS

```
sudo apt-get install libsdl1.2-dev
sudo apt-get install libx11-dev
sudo apt-get install libncurses-dev
sudo apt-get install libxrandr-dev
wget http://bochs.sourceforge.net/svn-snapshot/bochs-20160830.tar.gz
tar -zxvf bochs-20160830.tar.gz
cd bochs-20160830/

./configure --with-x11 --with-sdl --with-term  --with-nogui
--enable-sb16 --enable-ne2000 --enable-all-optimizations
--enable-cpu-level=6 --enable-x86-64 --enable-vmx=2 --enable-pci
--enable-clgd54xx --enable-voodoo --enable-usb --enable-usb-ohci
--enable-usb-xhci --enable-es1370 --enable-e1000 --enable-show-ips
--disable-readline --enable-x86-debugger --enable-debugger
make
su sudo
make install
```

TEST1:
cd ~/taller/demobochs
r2 -d bochs:///usr/bin/bochs#stage4.bxrc

TEST2:
cd ~/taller/demobaleful
r2 -a baleful -e io.cache=true vm.cifrada.code