

re|view
reverse engineering
[binary file]
viewer



Open-Source Projekt

2011, Mario Schallner



Was ist **re|view** ?

xR	0805eb60	8B 0D 58 E7 10 08	mov	ecx, [shell_builtins]
	0805eb66	01 F1	add	ecx, esi
	0805eb68	83 C6 18	add	esi, 0x18
	0805eb6b	8B 01	mov	eax, [ecx]
	0805eb6d	85 C0	test	eax, eax
	0805eb6f	74 6F	jz	→ 0x0805EBE0

Open-Source Projekt

Low-Level Binary-Analyseprogramm

Interaktiver Disassembler

Integrierte **abstrakte** Programm-Code Suche

Programmierbar (→ acss, → rv|script)

Flexibles User Interface

Einsatzgebiete

Automatisierte / manuelle Low-Level Code-Analyse,
Binary Auditing, Malware-Analyse (Viren, Trojaner,
Rootkits, Exploits, ...), Ausbildung

Programmiert in **C++** / Qt: ~ 17.000 Zeilen Quelltext

```
65 int disassemble_flow_recursive(  
66     rv_addr_t va,  
67     QMap<rv_addr_t, char> *map,  
68     int &reclevel, rv_addr_t va_from);  
69  
70 QMap<rv_addr_t, char> *disassemble_flow_recursive_from(  
71     rv_addr_t va, QMap<rv_addr_t, char> *M);  
72  
73 int disassemble_single(rv_addr_t va, x86_insn_t *insn);  
74  
75 // ----- oss: disassembly functions -----  
76 QHash<rv_addr_t, rv_addr_t> *process_disassembly_caves(  
77     QMap<rv_addr_t, char> *M);  
78  
79 QHash<rv_addr_t, rv_addr_t> *process_disassembly_hiddenInsns(  
80     QMap<rv_addr_t, char> *M,  
81     QHash<rv_addr_t, rv_addr_t> *allready_hidden = 0);  
82  
83 void format_insn( x86_insn_t *insn, char *buf, int len );  
84 void format_insn4preview( x86_insn_t *insn, char *buf, int len );  
85 void format_mnemonic( x86_insn_t *insn, char *buf, int len );  
86 void format_operands( x86_insn_t *insn, char *buf, int len );  
87 void format_arg_operands( x86_insn_t *insn, char *buf, int len );  
88  
89 int get_dest_controlflow( x86_insn_t *insn, char *buf );  
90 int check_dest_controlflow( x86_insn_t *insn );  
91 int check_dest_memaccess( x86_insn_t *insn );  
92 rv_addr_t rv_x86_get_address(x86_insn_t *i);  
93  
94 // ----- access use in disassembler directly -----  
95 rv_addr_t search_deep(rv_acss_parser *p,  
96     rv_addr_t &va_from, rv_addr_t va_until);  
97  
98 char *disassemble_preview(rv_addr_t va_start,  
99     rv_addr_t va_end, char *rbuf);  
100  
101 // ----- for use on acss click -----  
102 QHash<rv_addr_t, rv_addr_t> *search_disassembly(  
103     rv_acss_parser *p,  
104     QMap<rv_addr_t, char> *va_insnsizes_map,  
105     rv_addr_t va_from = 0, rv_addr_t va_until = 0,  
106     int maxresults = 0);  
107  
108 QHash<rv_addr_t, rv_addr_t> *search_deep_all(  
109     rv_acss_parser *p, int maxresults = 0);  
110  
111 QHash<rv_addr_t, rv_addr_t> *search_deep_range(  
112     rv_acss_parser *p,  
113     rv_addr_t va_from = 0, rv_addr_t va_until = 0,  
114     int maxresults = 0);  
115  
116 public slots:  
117  
118 signals:  
119 void process_changed_to(const QString &);  
120 void debug_print(const QString &);  
121  
122 public:  
123 QMultiMap<rv_addr_t, rv_addr_t> *va_xrefs_mmap;  
124 QString func_reg_acss_filter;  
125  
126 rv_address_descriptor *disasm_ad;  
127 rv_file *disasm_file;  
128 // TODO: remove rv_file parameter from function calls !  
129  
130 }  
131  
132 #endif // REVIEW_H
```



00000030	1c	00	18	00	06	00	00	34	00	00	34	80	04	08	[.....4...4...]
00000040	34	04	04	00	20	01	00	20	01	00	05	00	00	00	[4.....T...T...]
00000050	04	00	00	03	00	00	00	54	01	00	54	81	04	08	[.....T...T...]
00000060	54	81	04	00	13	00	00	00	00	00	54	00	00	00	[T.....T...T...]
00000070	01	00	00	00	01	00	00	00	00	00	80	00	04	08	[.....T...T...]
00000080	00	80	04	08	2c	62	01	00	2c	62	05	00	00	00	[.....b...b...]
00000090	09	10	00	00	08	00	00	08	00	00	08	05	08	08	[.....n...n...]
000000a0	08	00	00	08	00	00	00	08	00	00	08	05	08	08	[.....n...n...]
000000b0	00	10	00	02	02	00	00	0c	01	00	0c	0f	05	08	[.....6...6...]
000000c0	0c	0f	05	08	e0	00	00	e0	00	00	05	00	00	00	[.....h...h...]
000000d0	04	00	00	04	00	00	00	68	01	00	68	81	04	08	[.....h...h...]
000000e0	68	81	04	08	44	00	00	44	00	00	04	00	00	00	[h...D...D...]
000000f0	04	00	00	00	e5	74	64	04	61	01	04	e1	05	08	[.....P...d...]



Funktionsumfang – Teil I

Läuft auf MS Windows, Linux, Mac OSX

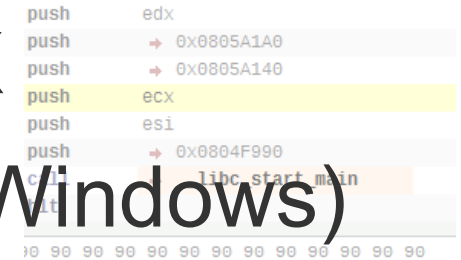
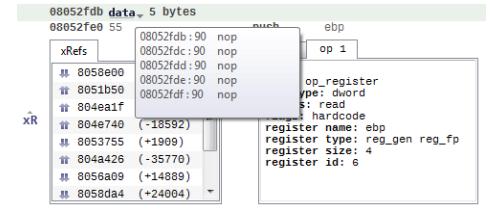
Analyse von **ELF32** (Linux), **PE32** (MS Windows)
Executables, **RAW** Dateien

Robuste Analyse der Datei-Struktur: Standard
Headers, Sections, Segments, Imports, ...

Fehlertolerant, alle Werte uninterpretiert in
Hex einsehbar

Interaktiver Disassembler

Control Flow Analyse, Crossreferences,
String Data References, **dynamische** Detail-Views,
Programmierbar





ACSS Search Results		✕
start	end	
8049ab2	8049ab3	mov [0x000603D0], eax
8049bdc	8049bdf	jz → 0x00051B50
8049e9b	8049e9d	mov [esp+0x10], 0x00049640
8049f98	8049f9a	mov [esp+0xc], 0x00049880
804a29f	804a2a1	mov [esp+0x8], 0x00000000
804a3af	804a3b1	mov [esp+0x4], 0x00000000
804bdbb	804bdbd	[esp], 0x000603E0
804be26	804be28	call → _obstack_begin
804bec0	804bec2	jmp → 0x0004FE1F
		cmp [0x000603C4], 0x00

Abstract Code Sequence Search:

Eigens entwickelte Filtersprache zur abstrakten Suche nach Code-Sequenzen und Algorithmen, extrem schnell durch binäre

on-the-fly Kompilierung

```
Expression
// filter example 5
// find small functions allocating local variables

1 x insn {
    type == push,
    op1_type == register, op1_reg_name == ebp;
};

1 x insn {
    type == mov,
    op2_type == register, op2_reg_name == esp;
};

1 x insn {
    type == sub,
    op1_type == register, op1_reg_name == esp;

    // find sub esp, ... OR
    // and esp, ...

    type == and,
    op1_type == register, op1_reg_name == esp;
};

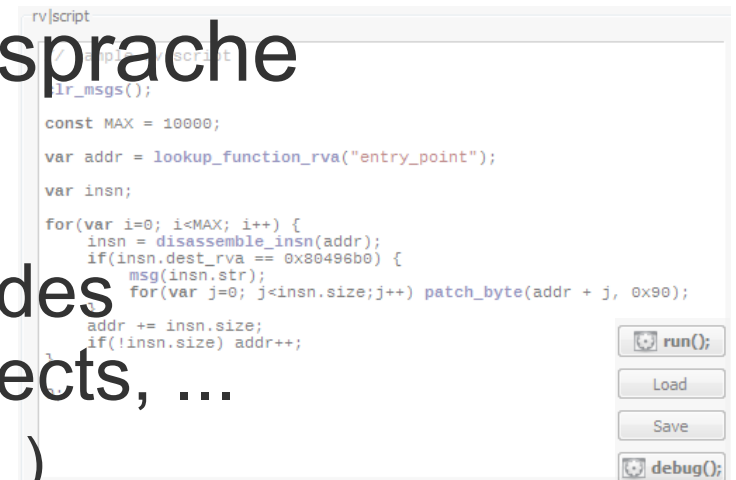
1-200 x insn {
    type != return, type != jmp;
};

1 x insn {
    type == return; // return
    type == jmp; // OR jmp
};
```

rv|script: Objektorientierte Skriptsprache

ECMA basierend („JavaScript“)

Zugriff auf Daten und Funktionen des Disassemblers, Instructions Objects, ... (disassemble_insn(), xrefs_at(), ...)



Besonderheiten von **re|view** – Teil I

Freie Software



FSF FREE SOFTWARE
FOUNDATION



Speziell entwickelt für Anforderungen der
Malware-Analyse:

Instruction-Metadaten-**Suche** (acss) im Disassembly,
oder über gesamtes Binary:

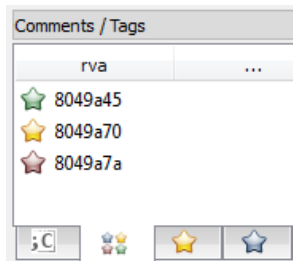
→ auch in **Datenblöcken**

acss kann zur **abstrakten Suche** von Algorithmen
eingesetzt werden, wo **Signatur-Scanning versagt**

Umgang mit „hostile binaries“ (korrupte Headers,
anti-disassembling, ...)

Besonderheiten von **re|view** – Teil II

Schwerpunkt komfortables und flexibles  User Interface:



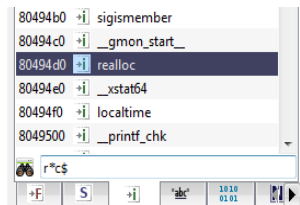
Dynamische Detail-Views, Dock Windows

Navigation mittels **Tags**, Comments, Names, ...

Integrierte Programmier-Möglichkeiten (— Programmierung und Verarbeitung der Ergebnisse im GUI möglich)

```
var addr = lookup_function_rva("entry_point");
var insn;
for(var i=0; i<MAX; i++) {
    insn = disassemble_insn(addr);
    if(insn.dest_rva == 0x80496b0) {
        msg(insn.str);
        for(var j=0; j<insn.size;j++)
            patch_byte(addr + j, 0x90);
    }
    addr += insn.size;
    if(!insn.size) addr++;
}
```

Syntax-Highlighting, **Regular Expression** Search in allen Suchfenstern



„Command-Line“: Direkte Befehls-eingabe im GUI

Besonderheiten **r|v** – Disassembler



Spezielle Darstellungsform für die komfortable Analyse

Strings im Klartext (

`mov [esp+0x4], \"../bash/shell.c\"`

Daten bei indirekter Addressierung im Disassembly

dargestellt (`mov eax, [0x080602A0] ; [0x080602A0] = 1`), ...

Dynamische Darstellung von Metadaten,

Crossreferences, Datenblöcken ...

on demand, direkt im Disassembly:

```
0804ac86 E9 85 ED FF FF      jmp      → strcmp
0804ac8b : 90 8d 74 26 00 55 31 c9 89 e5 83 ec 18 8b 45 0c : ..t&.U1.....E.
0804ac9b : 89 75 f8 8b 55 08 89 5d f4 89 7d fc 8b 58 68 8b : .u..U..]...Xh.
0804acab : 7a 68 83 fb 09 0f 94 c1 83 fb 03 0f 94 c3 89 ce : zh.....
0804acbb : 0f b6 db 09 de 31 db 83 ff 09 0f 94 c3 31 c9 83 : .....1.....1.
0804accb : ff 03 0f 94 c1 09 cb 75 7c 89 f1 84 c9 75 3e 8b : .....U|...U>.
0804acdb : 58 38 8b 4a 38 8b 78 34 8b 72 34 39 cb 7d 16 8b : X8.J8.x4.r49.}..
0804aceb : 5d f4 b8 ff ff ff ff 8b 75 f8 8b 7d fc 89 ec 5d : ]......U..]...
0804acfb : c3 8d 74 26 00 7f 16 39 f7 72 e4 39 cb 7c 26 39 : ..t&...9.r.9.|&9
0804ad0b : f7 8d 74 26 00 76 1e 8d b6 00 00 00 8b 5d f4 : ..t&.v.....].
0804ad1b : b8 01 00 00 00 8b 75 f8 8b 7d fc 89 ec 5d c3 8d : .....U..]...
0804ad2b : b6 00 00 00 00 8b 12 8b 00 8b 5d f4 8b 75 f8 89 : .....]...U..
0804ad3b : 55 08 8b 7d fc 89 45 0c 89 ec 5d e9 c5 ec ff ff : U..}..E...].
0804ad4b : 90 8d 74 26 00 89 f1 84 c9 75 84 eb 92 90 8d b4 : ..t&...U.....
0804ad5b : 26 00 00 00 00 55 31 c9 89 e5 83 ec 18 8b 55 0c : &....U1.....U.
0804ad6b : 89 75 f8 8b 45 08 89 5d f4 89 7d fc 8b 5a 68 8b : .u..F..l...}..7h.

0804ae30 55      push    ebp
0804ae31 89 E5      mov     ebp, esp
```

```
010098f4 4A      dec     edx
010098f5 75 0B      jnz     → 0x01009902
010098f7 33 C0      xor     eax, eax

xRefs
↓ 100990e (+23)
↓ 100990a (+19)
↓ 10098ff (+8)
↑ 10098d7 (-32)

insn  op 1  op 2
xor eax, eax
group:logic
type:xor
size:2
flags set: (37) carry_set
zero_set oflow_set sign_set
parity_set
flags tested: (0)
prefix: (0)
stack modified?:0
stack mod val:0
operand count:2

010098f9 EB 18      jmp     → 0x01009913
010098fb 24 C0      and     al, 0xC0
```


Besonderheiten **r|v** – Disassembler

Alle Address-Parameter „clickable“, auch bei non-controlflow instructions (zB.: `push` → `0x0805A140`)



Darstellung von File-Offset und Virtual Address

Darstellung von in Instructions codierten ASCII Strings direkt im Disassembly (zB.:

```
0805f3c5 C7 00 2F 62 69 6E  mov    [eax], 0x6E69622F
0805f3c7                               /bin )
```

zB. zur Erkennung von Shellcodes

Control Flow Analyse erkennt u. markiert
disaligned Instructions:

08048060	E8 01 00 00 00	call	→ 0x08048066
08048065	E9 58 90 05 0B	jmp	0x130A10C2
xR 08048066	58	pop	eax
08048067	90	nop	
08048068	05 0B 00 00 00	add	eax, 0x0000000B
0804806d	50	push	eax
0804806e	C3	ret	
0804806f	data 16 bytes		

Besonderheiten r|v – Disassembler

Tooltip Disassemblies für Datenblöcke und Sprungziele:

Datenblock

```
0804ac8b data 421 bytes
0804ae30 55          push     ebp
0804ae31 89          0804ac8b:90 nop
0804ae33 83          0804ac8c:8D 74 26 00 lea     esi, [esi+]
0804ae36 8B          0804ac90:55 push     ebp
0804ae39 8B          0804ac91:31 C9 xor      ecx, ecx
0804ae3c 8B          0804ac93:89 E5 mov      ebp, esp
0804ae3e 8B          0804ac95:83 EC 18 sub     esp, 0x18
0804ae3e 8B          0804ac98:8B 45 0C mov     eax, [ebp+0xC]
0804ae40 89          0804ac9b:89 75 F8 mov     [ebp-0x8], esi
0804ae43 89          0804ac9e:8B 55 08 mov     edx, [ebp+0x8]
0804ae46 C9          0804aca1:89 5D F4 mov     [ebp-0xC], ebx
0804ae47 E9          0804aca4:89 7D FC mov     [ebp-0x4], edi
0804ae47 E9          0804aca7:8B 58 68 mov     ebx, [eax+0x68]
0804ae4c dat      0804aca7:8B 58 68 mov     edi, [edx+0x68]
0804ae4c dat      0804acaa:8B 7A 68 mov     edi, [edx+0x68]
0804ae50 55          0804acad:83 FB 09 cmp     ebx, 0x09
0804ae51 89          0804acb0:0F 94 C1 setz    cl
0804ae53 83          0804acb3:83 FB 03 cmp     ebx, 0x03
0804ae53 83          0804acb6:0F 94 C3 setz    bl
0804ae56 8B          0804acb9:89 CE mov     esi, ecx
0804ae59 8B          0804acbb:0F B6 DB movzx   ebx, bl
0804ae5c 8B          0804acbe:09 DE or      esi, ebx
0804ae5e 8B          0804acc0:31 DB xor     ebx, ebx
0804ae60 89          0804acc2:83 FF 09 cmp     edi, 0x09
0804ae63 89          0804acc5:0F 94 C3 setz    bl
0804ae63 89          0804acc8:31 C9 xor     ecx, ecx
0804ae63 89          0804acca:83 FF 03 cmp     edi, 0x03
```

Sprungziel

```
jnz      → 0x08050D3F
cmp      [0x80601C8], 0x01 ; [0x80601C8] = 1
jz       → 0x08050D3F
call     → 0x08050D3F
mov      eax, [0x080602E0]
xor      ebx, ebx
mov      [esp], eax
xor      ebx, ebx
mov      [esp], ebx
call     ff1
jmp      → 0x08050D64
```



◦ **JMP Stack**
speichert Rücksprungsadressen
während der Navigation:

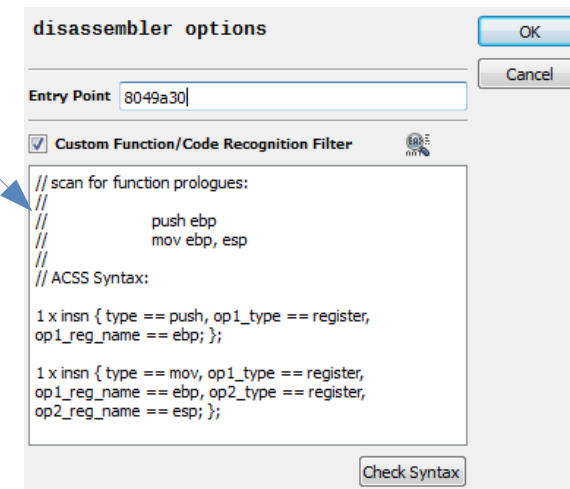
Markierung der Adressen im
Disassembly:

JMP stack	
←	100527f
←	100156a
←	1004353
←	1003b51
←	1003695

```
0100155f data 5 bytes
01001564 3B 0D 10 C0 00 01 cmp     ecx, [0x100C010] ;
0100156a 0F 85 97 89 00 00 jnz     → 0x01009F07 ←
01001570 C3 ret
```

Besonderheiten **r|v** – Disassembler

Disassembler benutzt „acss“ zur Analyse des Binaries,
parametrisierbarer (!)
Function-Recognition Filter
zum Aufspüren von nicht-
referenzierten Code-Blöcken

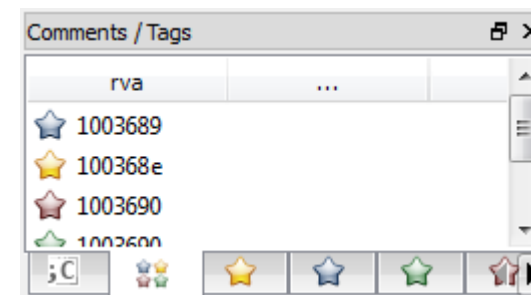


Steuerbar über Commandline – Befehle



Direktes Taggen von Positionen im Disassembly

★	01003689	E8 C5 F9 FF FF	call	→ 0x01003053
★	0100368e	6A 58	push	0x58
★ ★ ★ ★	01003690	68 A0 37 00 01	push	0x010037A0





r/v Metadatenuche – ACSS

Filtersprache zur abstrakten Formulierung von Code-Sequenzen und Algorithmen:

```
// filter example 4
// evil calls I
// call register

1 x insn {
  op1_type == register,
  op1_access == (&execute);
};
```

```
// filter example 2

4-8 x insn {
  group == move,
  op2_type == immediate,
  op2_datatype == dword;
};

1 x insn { type == call; };
```

```
1 x insn {
  type == sub,
  op1_type == register, op1_reg_name == esp;

  // find sub esp, ... OR
  // and esp, ...

  type == and,
  op1_type == register, op1_reg_name == esp;
};

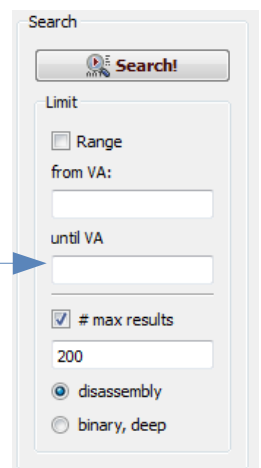
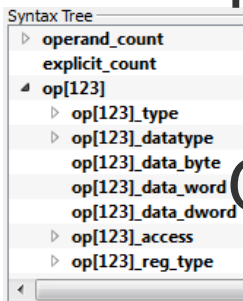
1-200 x insn {
  type != return, type != jmp;
};
```

Metadaten inkludieren uA.: instruction group, type, flags-status, operands, registers, data, access type, ...

Operatoren: ==, !=, implizites AND und OR (über Struktur)

Suche im Disassembly, oder über gesamtes Binary möglich, Bereichsangaben, etc ...

Filter organisiert in Libraries, werden in binäre Form kompiliert **extrem schnelle Suche**





r|v Metadatenuche – ACSS

Suchergebnisse über GUI zugreifbar und hinterlegt:

Ergebnis

The screenshot shows the r|v GUI with the following components:

- Main Window:** Displays search results for ACSS. The table has columns: start, end, rva, hex, mnemonic, and operands. The results are filtered by the 'xR' filter.
- Right Sidebar:** Shows a list of symbols with columns: start, end, and a numerical value.
- Bottom Status Bar:** Indicates the current command is 'disassembly 1'.

The search results table is as follows:

start	end	rva	hex	mnemonic	operands
804c600	804c633	08053938	E8 C3 5A FF FF	call	→ __errno_location
804c650	804c665	0805393d	C7 00 5F 00 00 00	mov	[eax], 0x0000005F
804c690	804c6ae	08053943	B8 FF FF FF FF	mov	eax, ".gnu_debuglink"
804d030	804d0c9	08053948	EB B0	jmp	→ 0x080538FA
804ee40	804ee66	0805394a	data, 6 bytes		
804ef26	804ef26	08053950	55	push	ebp
804ef40	804ef40	08053951	89 E5	mov	ebp, esp
804ef60	804ef60	08053953	83 EC 28	sub	esp, 0x28
804ef80	804ef80	08053956	89 75 F8	mov	[ebp-0x8], esi
804efa0	804efa0	08053959	8B 45 08	mov	eax, [ebp+0x8]
804efc0	804efc0	0805395c	8B 75 0C	mov	esi, [ebp+0xC]
804f000	804f000	0805395f	89 5D F4	mov	[ebp-0xC], ebx
804f020	804f020	08053962	89 7D FC	mov	[ebp-0x4], edi
804f040	804f040	08053965	89 04 24	mov	[esp], eax
804f060	804f060	08053968	89 74 24 04	mov	[esp+0x4], esi
804f080	804f080	0805396c	E8 EF 5E FF FF	call	→ fgetfilecon
804f100	804f100	08053971	85 C0	test	eax, eax
804f120	804f120	08053973	74 43	jz	→ 0x08053988
804f140	804f140	08053975	83 F8 0A	cmp	eax, 0x0A
804f160	804f160	08053978	74 0E	jz	→ 0x08053988
804f180	804f180	0805397a	8B 5D F4	mov	ebx, [ebp-0xC]
804f200	804f200	0805397d	8B 75 F8	mov	esi, [ebp-0x8]
804f220	804f220	08053980	8B 7D FC	mov	edi, [ebp-0x4]
804f240	804f240	08053983	89 EC	mov	esp, ebp
804f260	804f260	08053985	5D	pop	ebp
804f280	804f280	08053986	C3	ret	
804f300	804f300	08053987	data, 1 byte		
804f320	804f320	08053988	8B 16	mov	edx, [esi]
804f340	804f340	0805398a	B8 AC A2 05 08	mov	edi, "unlabeled"
804f360	804f360	0805398f	B9 0A 00 00 00	mov	ecx, 0x0000000A
804f380	804f380	08053994	89 D6	mov	esi, edx
804f400	804f400	08053996	F3 A6	repz cmps	es:[edi], ds:[esi]
804f420	804f420	08053998	75 E0	jnz	→ 0x0805397A



r/v Metadatenuche – ACSS

Möglichkeit **acss** auch **ohne Disassembly** (ohne vorherige Control-Flow Analyse) einzusetzen:

Suchergebnisse werden rot hinterlegt, bei Doppelklick wird vom Ergebnis weg ein Control-Flow Analyse gestartet, und Ergebnis grün hinterlegt

D.h. Es wurden **nur die Instructions verfolgt, die von diesem Code-Teil aus erreicht werden**. Alles andere bleibt ausgeblendet (!)

zB:

```
Expression
// filter example 3
// nop zones

8-200 x insn {
  type == nop;
};
```

Filter

Name

Store

Filters

- func end
- func 4-8 const args
- nop zones

auf leeres Disassembly angewendet:



r|v Metadatenuche – ACSS

Suchergebnis 0x8053a16:

RE | view

File View Settings Disassembler Help

ACSS Search Results

start	end	rva	hex	mnemonic	operands
8049a52	8049a5f	080532fa			
8049ae3	8049aef	080533c6			
8049c5f3	804c5ff	08053821			
804cf33	804cf3f	08053a16 90		nop	
8051e37	8051e3f	08053a17 90		nop	
80521e5	80521ef	08053a18 90		nop	
80526b2	80526bf	08053a19 90		nop	
8052956	805295f	08053a1a 90		nop	
8053423	805342f	08053a1b 90		nop	
8053462	805346f	08053a1c 90		nop	
8053565	805356f	08053a1d 90		nop	
8053722	805372f	08053a1e 90		nop	
8053842	805384f	08053a1f 90		nop	
8054193	805419f	08053a20 55		push	ebp
8054273	805427f	08053a21 89 E5		mov	ebp, esp
8054463	805446f	08053a23 83 EC 18		sub	esp, 0x18
8055c06	8055c0f	08053a26 8B 45 08		mov	eax, [ebp+0x8]
8055ed3	8055edf	08053a29 89 75 F8		mov	[ebp-0x8], esi
8058c33	8058c3f	08053a2c 89 7D FC		mov	[ebp-0x4], edi
8058cf4	8058cff	08053a2f C7 44 24 04 00 00 0...		mov	[esp+0x4], 0x00000000
8058e67	8058e6f	08053a37 89 04 24		mov	[esp], eax
8058f27	8058f2f	08053a3a E8 91 5D FF FF		call	⇒ setlocale
8059344	805934f	08053a3f 89 C6		mov	esi, eax
8059b06	8059b0f	08053a41 B8 01 00 00 00		mov	eax, 0x00000001
8059b86	8059b8f	08053a46 85 F6		test	esi, esi
8059d07	8059d0f	08053a48 74 19		jz	⇒ 0x08053A63
8059fe4	8059fef	08053a4a 0F B6 05 31 D8 05 08		movzx	eax, [0x805D831]
805a135	805a13f	08053a51 38 06		cmp	[esi], al
		08053a53 75 1B		jnz	⇒ 0x08053A70
		08053a55 0F B6 05 32 D8 05 08		movzx	eax, [0x805D832]

Symbols

start	end	size (hex)
8048000	80493e0	5088 (0x13e0)
80493ec	80497d0	996 (0x3e4)
80497e0	8053a16	41526 (0xa236)
8053a6d	8053a70	3 (0x3)
8053a89	805e22c	42915 (0xa7a3)
805fef8	80602c0	968 (0x3c8)

Comments / Tags

JMP stack

1 2

disassembly 1 disassembly 2 acss rv | script hex view

Cmd: SEL RUN dist↓



Erweiterung mit **r|v script**

Objektorientierte Programmiersprache
nach dem ECMA Standard („JavaScript“)

inklusive Debugger

Bietet Zugriff auf die Disassembling-Engine

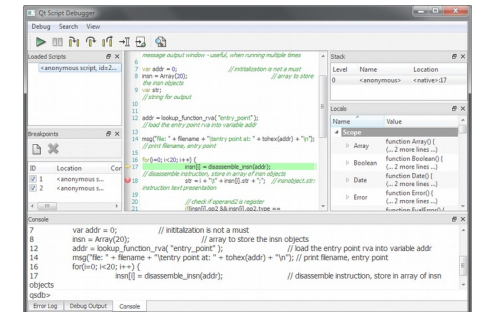
Stellt Objekte des Typs „Instruction“ zur
Verfügung, welche alle Metadaten enthalten

Stellt Funktionen der Disassembling-Engine zur Verfügung:
zB: `disassemble_insn()`, `lookup_function_rva()`, ...

Stellt Listen-Objekte für Crossreferences zur Verfügung
zu füllen per `xrefs_at()`;

Syntax Highlighting, Ausgabefenster für Meldungen, ...

```
// sample r|vscript
clr_msgs();
const MAX = 10000;
var addr = lookup_function_rva("entry_point");
var insn;
for(var i=0; i<MAX; i++) {
    insn = disassemble_insn(addr);
    if(insn.dest_rva == 0x80498b0) {
        msg(insn.str);
        for(var j=0; j<insn.size; j++)
            patch_byte(addr + j, 0x90);
        addr += insn.size;
        if(!insn.size) addr++;
    }
}
0;
```





r/v script

Beispielscript:

rv|script

```
// sample rv|script
// print the disassembly of the entrypoint, do demo checks on insn.group, insn.type, insn.op1.type,
// and store insns in an array (20 lines)

clr_msgs();           // clear the message output window - useful, when running multiple times

var addr = 0;         // initialization is not a must
insn = Array(20);     // array to store the insn objects
var str;              // string for output

addr = lookup_function_rva( "entry_point" );           // load the entry point rva into variable addr
msg("file: " + filename + "\tentry point at: " + tohex(addr) + "\n"); // print filename, entry point
for(i=0; i<20; i++) {
    insn[i] = disassemble_insn(addr);           // disassemble instruction, store in array of insn objects
    str = i + "\t" + insn[i].str + ";";         // insnobject.str: instruction text presentation

    // check if operand2 is register
    if(insn[i].op2 && insn[i].op2.type == OP_REGISTER) str += "\t(insn.op2.reg.name: '" + insn[i].op2.reg.name + "')";

    // check if operand1 is immediate value
    if(insn[i].op1 && insn[i].op1.type == OP_IMMEDIATE) str += "\t(insn.op1.data (immediate): " + tohex(insn[i].op1.data) + ")";

    if(insn[i].group == INSN_LOGIC) str += "\tinsn.group: LOGIC !";
    if(insn[i].type == INSN_CALL) str += "\tinsn.type: CALL !";

    msg(str);
    addr += insn[i].size;           // set addr to rva of next instruction
}

msg("\n\n == insn[5] ==:\n" + insn[5].str);

0;
```

run()
 Load
 Save
 debug()

wordwrap
 maximize



r/v script

Ausführung des Beispielscripts:

```
messages
file: F:/_mario/_prog/_ELFs/ls      entry point at: 8049A30
0  08049a30 : xor    ebp, ebp;      (insn.op2.reg.name: 'ebp')    insn.group: LOGIC !
1  08049a32 : pop    esi;
2  08049a33 : mov    ecx, esp;      (insn.op2.reg.name: 'esp')
3  08049a35 : and    esp, 0xF0;    insn.group: LOGIC !
4  08049a38 : push   eax;
5  08049a39 : push   esp;
6  08049a3a : push   edx;
7  08049a3b : push   0x0805A1A0; (insn.op1.data (immediate): 805A1A0)
8  08049a40 : push   0x0805A140; (insn.op1.data (immediate): 805A140)
9  08049a45 : push   ecx;
10 08049a46 : push   esi;
11 08049a47 : push   0x0804F990; (insn.op1.data (immediate): 804F990)
12 08049a4c : call   0x08049580; insn.type: CALL !
13 08049a51 : hlt    ;
14 08049a52 : nop    ;
15 08049a53 : nop    ;
16 08049a54 : nop    ;
17 08049a55 : nop    ;
18 08049a56 : nop    ;
19 08049a57 : nop    ;

== insn[5] ==:
08049a39 : push   esp
--- script returned: '0'
--- script terminated ---
```



re|view

RE | view

File View Settings Disassembler Help

offset	label	rva	hex	mnemonic	operands	comment
00003f6e		0804bf6e	A3 98 04 06 08	mov	[0x08060498], eax	
00003f73		0804bf73	83 C4 10	add	esp, 0x10	
00003f76		0804bf76	5B	pop	ebx	
00003f77	my_name1	0804bf77	5E	pop	esi	
00003f78		0804bf78	5D	pop	ebp	
00003f79		0804bf79	C3	ret		
00003f7a		0804bf7a	data 6 bytes			
00003f80		0804bf80	89 54 24 0C	mov	[esp+0xC], edx	
<div><div>xRefs</div><div>↑ 804bf0e (-114)</div><div>xR</div></div> <div><div>insn op 1 op 2</div><div>mov [esp+0xC], edx</div><div>group:move</div><div>type:mov</div><div>size:4</div><div>flags set: (0)</div><div>flags tested: (0)</div><div>prefix: (0)</div><div>stack modified?:0</div><div>stack mod val:0</div><div>operand count:2</div><div>op explicit count:2</div><div>addr size:4</div></div>						
00003f84		0804bf84	89 F3	mov	ebx, esi	
00003f86		0804bf86	89 4C 24 08	mov	[esp+0x8], ecx	
00003f8a		0804bf8a	C7 44 24 04 6A A2 0...	mov	[esp+0x4], "%*lu "	
00003f92		0804bf92	C7 04 24 01 00 00 00	mov	[esp], 0x00000001	
00003f99		0804bf99	E8 62 D5 FF FF	call	→ __printf_chk	
00003f9e		0804bf9e	EB C5	jmp	→ 0x0804BF65	
00003fa0		0804bfa0	C7 44 24 04 20 00 00...	mov	[esp+0x4], 0x00000020	
00003fa8		0804bfa8	89 04 24	mov	[esp], eax	
00003fab		0804bfab	E8 C0 D6 FF FF	call	→ __overflow	
00003fb0		0804bfbb	EB AF	jmp	→ 0x0804BF61	
00003fb2		0804bfbb	data 1			
00003fc0		0804bfc0	55	push	ebp	
<div><div>xRefs</div><div>↑ 804d9dd (+6685)</div><div>↑ 804d9ad (+6637)</div></div> <div><div>insn op 1</div><div>push ebp</div><div>group:stack</div><div>type:push</div></div>						

Cmd: 804bfc0

Symbols

rva	name
80493f0	abort
8049400	__errno_location
8049410	sigemptyset
8049420	localeconv
8049430	dirfd
8049440	__cxa_atexit
8049450	strcoll
8049460	memcmp
8049470	fputs_unlocked
8049480	__ctype_get_mb_cur_max
8049490	__fprintf_chk
80494a0	signal
80494b0	sigismember
80494c0	__gmon_start__
80494d0	realloc
80494e0	__xstat64
80494f0	localtime
8049500	__printf_chk

Comments / Tags

rva	...
804bf6e	
804bf8a	

JMP stack

8049ae3
8049ab5
8049646
8049636
804b770

2

