CSAT

Pierre-Marie de Rodat

pmderodat@lse.epita.fr
http://lse.epita.fr

July 16, 2012

What is a disassembler?

- `objdump(1)`
- It takes an executable (e.g.: ELF)
- It reads sections table
- ... and only disassemble what it expects to see
- Works only for very nice binaries (those yielded by a compiler)
- Useful for debugging, but not reverse engineering

Then static analysis?

- Static != dynamic
- Analysis of some program without executing it
- Relies heavily on disassembling meta-information
- Hardest part: predict control flow

IDA

- Excellent interactive static analysis tool (Windows/Linux/OSX, not free)
- Enables one to interactively give information in order to continue disassembling where it stopped
- Handles a *huge* list of binary formats and architectures
- Can even decompile to C
- Accept extensions (C and Python API as far as I know)
- But badly engineered and documented API...

IDA: disassembler

CSAT

Pierre-Marie de Rodat

IDA View-A

```
.text:004011A1
.text:004011A1 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:004011A1
.text:004011A1 ; Attributes: bp-based frame
.text:004011A1
.text:004011A1 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:004011A1 _main   proc near         ; DATA XREF: .data:0040A0D0↓o
.text:004011A1
.text:004011A1 array   = byte ptr -1Ch
.text:004011A1 size    = dword ptr -8
.text:004011A1 ptr     = dword ptr -4
.text:004011A1 argc    = dword ptr  8
.text:004011A1 argv    = dword ptr  0Ch
.text:004011A1 envp    = dword ptr  10h
.text:004011A1
.text:004011A1         push    ebp
.text:004011A2         mov     ebp, esp
.text:004011A4         add     esp, 0FFFFFFE4h
.text:004011A7         push    esi
.text:004011A8         push    edi
.text:004011A9         mov     eax, init_val
.text:004011AF         mov     [ebp+size], eax
.text:004011B2         mov     al, byte_40A12C
.text:004011B8         mov     byte ptr [ebp+ptr],
.text:004011BB         mov     esi, offset default
.text:004011C0         lea     edi, [ebp+array]
.text:004011C3         mov     ecx, 5
.text:004011C8         rep movsd
.text:004011CA         push    5
.text:004011CC         lea     eax, [ebp+size]
.text:004011CF         push    eax
.text:004011D0         call    adjust_array
.text:004011D5         add     esp, 8
.text:004011D8         movsx   edx, al
.text:004011DB         push    edx
.text:004011DC         push    offset format ; format
.text:004011E1         call    _printf
```

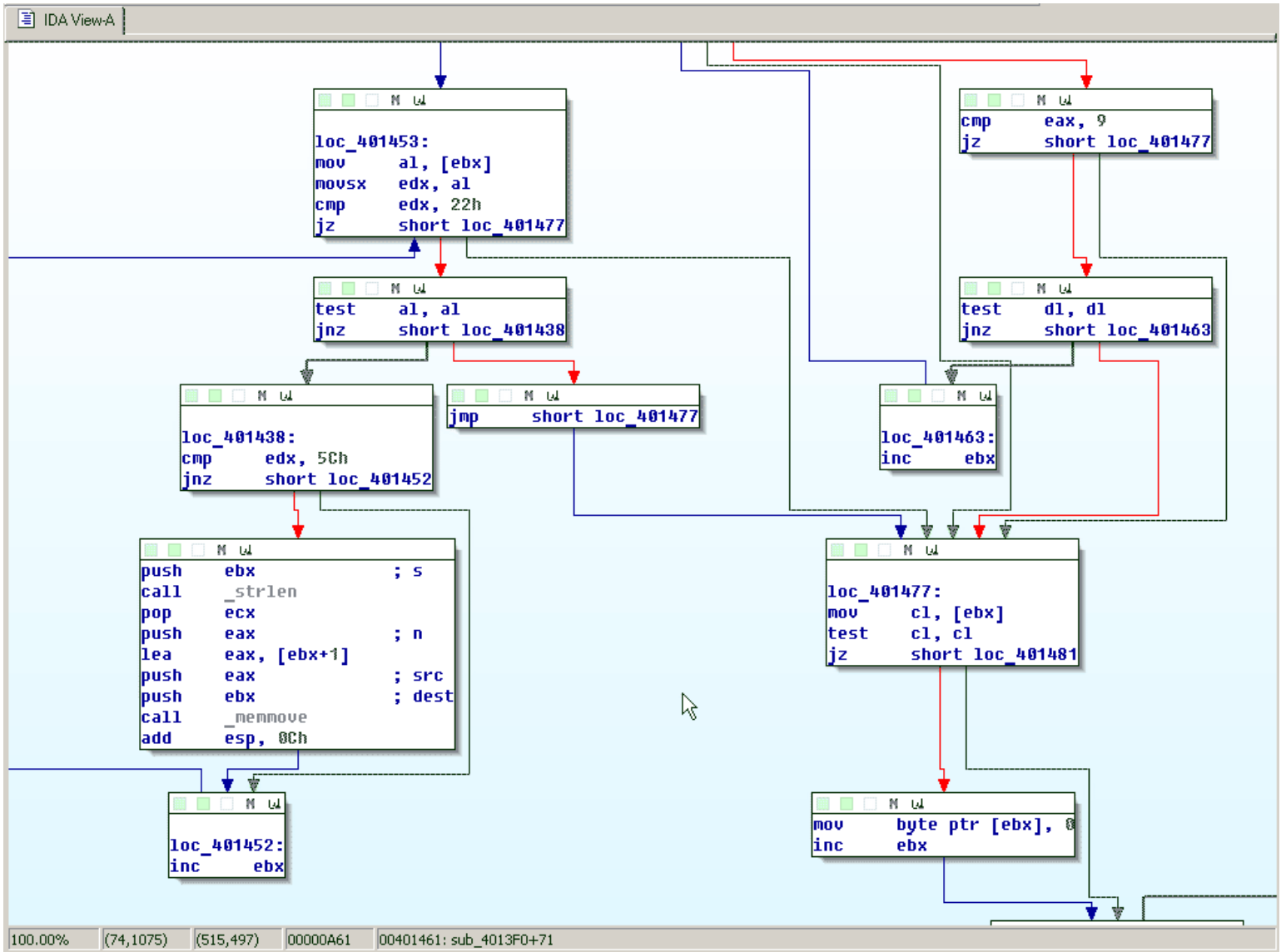
Stack of _main

Edit Jump Search

```
-0000001C ; Ins/Del : create/delete structur
-0000001C ; D/A/*   : create structure membe
-0000001C ; N       : rename structure or st
-0000001C ; U       : delete structure membe
-0000001C ; Use data definition commands to
-0000001C ; Two special fields " r" and " s"
-0000001C ; Frame size: 1C; Saved regs: 4; P
-0000001C ;
-0000001C
-0000001C array   db 20 dup(?)      ; char
-00000008 size    dd ?              ; base 10
-00000004 ptr     dd ?              ; offset
+00000000  s      db 4 dup(?)
+00000004  r      db 4 dup(?)
+00000008 argc    dd ?
+0000000C argv    dd ?              ; offset
+00000010 envp    dd ?              ; offset
+00000014
+00000014 ; end of stack variables
```

SP++00000000

IDA: graph


IDA View-A
loc_401453:
mov al, [ebx]
movsx edx, al
cmp edx, 22h
jz short loc_401477
cmp eax, 9
jz short loc_401477
test al, al
jnz short loc_401438
test dl, dl
jnz short loc_401463
loc_401438:
cmp edx, 5Ch
jnz short loc_401452
jmp short loc_401477
loc_401463:
inc ebx
push ebx ; s
call _strlen
pop ecx
push eax ; n
lea eax, [ebx+1]
push eax ; src
push ebx ; dest
call _memmove
add esp, 0Ch
loc_401477:
mov cl, [ebx]
test cl, cl
jz short loc_401481
loc_401452:
inc ebx
mov byte ptr [ebx], 0
inc ebx
100.00% (74,1075) (515,497) 00000A61 00401461: sub_4013F0+71

- Collaborative Static Analysis Tool
- Written using Python3
- Mad idea about making another IDA
- Learning things and trying to have fun

Binary formats

- Given a file descriptor and its filename, a binary format "plug-in" must accept or refuse the binary
- When accepted, it must define some architecture properties (endianess, addresses size, entry point, architecture)
- Then, load segments into a memory space
- Optionally, add segments (`.text`, ...)
- CSAT can then start disassembling on its own

Instructions decoding

- Fetch as many bytes as needed, and decode them
- CSAT provides data structures to be filled (instruction, operand, register, ...)
- It also provides utilities like bit fields, or dispatchers
- This part, alone, is easy: just act like a processor!

Disassembling methods (1/2)

- There are more than one way to disassemble code
- `objdump` just starts at the beginning of the `.text` section and continues as long as possible
- What if code is not nicely aligned by a compiler?

Disassembling methods (2/2)

- Another approach is to start disassembling at the entry point
- Then to follow control flow instructions (calls, jumps, ...)
- Decoders have to yield instructions *and* metadata like control flow type (branch, jump, call, ret, ...)
- You then build a control flow graph (CFG) as long as you can follow these instructions

Disassembling in CSAT

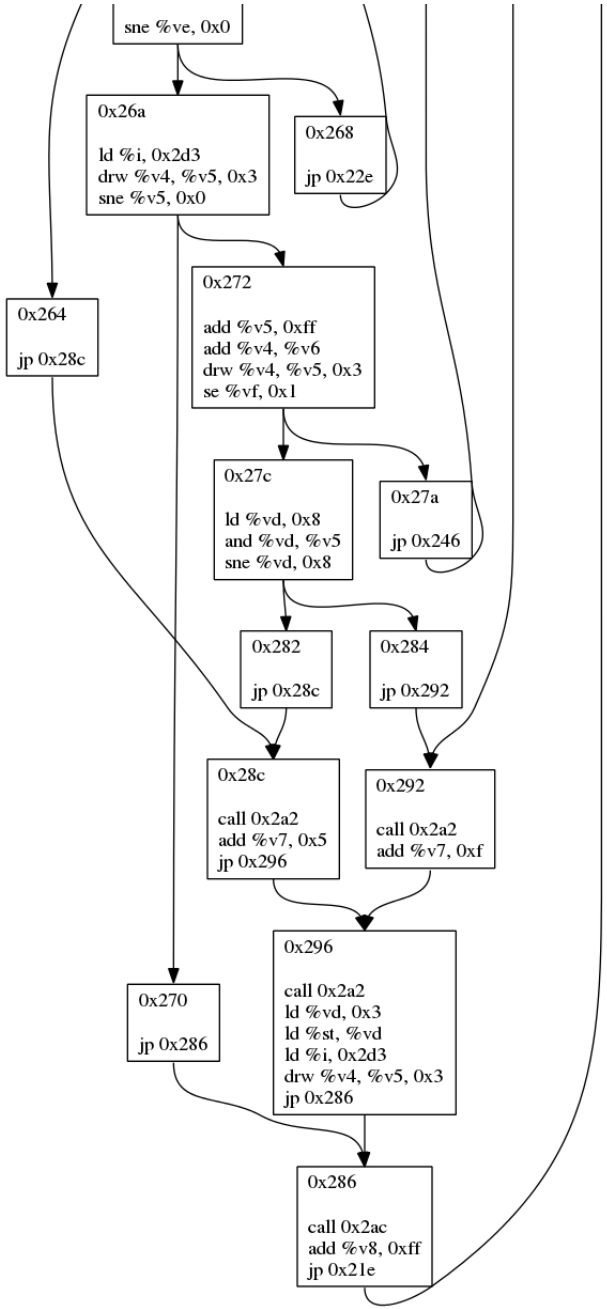
- To handle some architecture, CSAT just needs:
  - some metadata for architectures (delay slot, ...)
  - a `decode` method that fetch bytes and return an instruction and associated metadata (control flow type, etc.)
- It takes care of control flow handling and just call the provided `decode` method

Parallel decoding


```
c7 45 fc c3 00 00 00    movl   $0xc3,-0x4(%ebp)
```

- What if we jump in the middle of this instruction?
- `c3  ret`
- For some architectures, instructions don't have to be word-aligned (and it doesn't even make sense on most CISC!)
- Thus, code bytes may have different interpretations
- IDA assumes that one chunk of bytes can have only one interpretation: it is up to the user to determine which one is correct
- CSAT keeps every interpretation it discovers as reachable

Graph from CSAT (using dot)



UI!!

- To be interactive, CSAT needs a user interface...
- Originally, we thought about one based on Qt, but I am alone on this project right now
- I started one text-based, but it is not convenient
- Big task: a lot of things to display, but it has to stay usable
- UI may help development

Looking for functions

- Mechanisms to follow as many jump as possible are to be done
- Beyond that, some functions may stay unreachable from the static analysis point of view
- IDA can look through the memory space looking for things that look like function prologue/epilogue
- Disassemblers should expose more information to CSAT to do that

Recognize standard library functions

- From IDA (`http://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml`)
- The point is that a great part of programs are just standard library functions
- Detecting them from a signature database helps saving reversing time...

Interpreting code, predicting jumps

- When following jumps, the hard part is to know where goes `jump %r0`...
- To determine what could be the value of a register at one point, the disassembler should be able to have some understanding of the ISA
- Again, there is a tradeoff to find between...
- Factorisation: put as much code as possible in CSAT, not in modules
- And power: some architecture have unusual control flow instructions (like looping ones)

Stack analysis

- Once a function has been recognized, its stack frame can be analysed
- Every memory access that is relative to the stack pointer can be a local variable (or a function argument)
- Once again, the disassembler has to be aware of some ISA specificities

Did you talk about collaboration?

- Well... at the beginning, we were quite enthousiastic about a collaborative tool, mainly to work with during CTFs
- For the coming months, the target is to "only" have some handy interactive static analysis tool ;-)
- -> There is no thought about concurrency issues right now

Conclusion

- A lot of work to do!
- Shame on me: no test yet...
- A complex architecture (x86?) is to be tried to test the current API

Questions?

- Temporarily on `http://bitbucket.org/delroth/csat`