

Metasm Feelings

Alexandre Gazet

Sogeti / ESEC R&D

alexandre.gazet(at)sogeti.com

Yoann Guillot

Sogeti / ESEC R&D

yoann.guillot(at)sogeti.com



RECON 2010 - Montreal

Plan

- 1 Metasm
- 2 Tracer
- 3 MSR
- 4 NIC

Metasm: prends-moi

- opensource (LGPL)
- full ruby
- <http://metasm.cr0.org/>
- extensible
- scriptable
- full interactive documentation over IRC¹
- yadda yadda

¹<irc://freenode/metasm>

Main features

- Cross-arch assembler
- Disassembler
 - Intelligent
 - Emulation
 - Instruction semantics
- Debugger
 - Linux
 - Win32
 - GDBServer
- C Parser
- C Compiler
- Decompiler
- xtrem GUI

Target architectures

OS

- Windows
- Linux

Arch

- X86
- X64
- ARM
- MIPS
- PowerPC
- Sh4

Previous works

- Mainly static analysis of software protections:
 - Deobfuscation
 - Devirtualisation
- SSTIC (fr) - challenges (T2 2007, poeut, ...)
- HITB 2009 - realworld software protection
- Check the papers (with translation) on <http://metasm.cr0.org/>

Sample uses

```
require 'metasm'  
include Metasm  
  
np = OS.current.find_process('notepad')  
# read the virtual memory of a running process  
p np.memory[0x40000, 0x200]  
  
# inject a shellcode  
sc = Shellcode.assemble(ia32.new, 'jmp esp').encode_string  
np.memory[0x40000, sc.length] = sc
```



Sample uses

```
require 'open-uri'

puts ''retrieving source code...''
html = open('http://recon.cx/2010/cfp.html').read
source = html[ /unsigned.*\}/m ]
abort ''no source found :('' if not source

elf = Metasm::ELF.compile_c(Metasm::Ia32.new, source)
elf.encode_string ; elf.decode
dasm = elf.disassembler

# selfmodify plugin, handles trivial decoding loops
dasm.load_plugin 'selfmodify'
```


Plan

- 1 Metasm
- 2 Tracer
- 3 MSR
- 4 NIC

Trace engine

Scriptable debugger

- High level API: \Rightarrow `bpx()`, `stepover()`, etc.
- Disassembler: code graph

Building tools

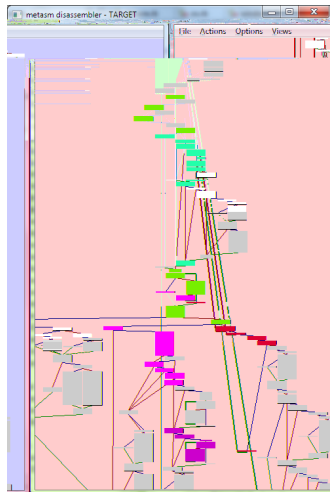
- Block-by-block trace algorithm implemented using the API
 \Rightarrow can be used on any supported target
- Database support for free with `rubygems`^a
 \Rightarrow Trivial to store execution traces

^aCurrently using DataMapper

Trace visualization

Scriptable GUI as well

- Execution path coloring
- Possible to add controls: trace selection, etc.
- Advanced data manipulation
 - Trace “replay”
 - Trace diffing



Demo

Trace creation
Trace diffing
Trace replay

Plan

- 1 Metasm
- 2 Tracer
- 3 MSR
- 4 NIC

MSR debug registers

- Hardware registers from processors (Intel/AMD)
- Branch tracing MSR: IA32_DEBUGCTL (0x1D9)
- Can trigger an INT1 whenever a branch is taken with TF set
- Implementation for WinXP in 2006 by Pedram Amini (NtSystemDebugControl)
Branch Tracing with Intel MSR Registers ²
- Same concepts can be applied on Windows 7 through the KD module³:
 - Issue IOCTLs to `\\.\kldbgdrv` device
 - Need to boot with `/DEBUG` option

²http://www.openrce.org/blog/view/535/Branch_Tracing_with_Intel_MSR_Registers

³<http://www.ivanlefeu.tuxfamily.org/?p=382>

DynLdr

We'd like to call `Kernel32!DeviceIOControl()` from our ruby script

DynLdr

DynLdr is a Metasm component to generate ruby bindings from C headers

- Handles arbitrary C function prototypes (incl. `__stdcall`)
- Handles most numeric macros/enums
- Wrapper for C structures

MSR: C definitions

```
class MSR < Metasm::DynLdr
  new_api_c <<EOS

  typedef struct _KLDBG {
    SYSDBG_COMMAND DbgCommandClass;
    PVOID DbgCommand;
    DWORD DbgCommandLen;
  } KLDBG, * PKLDBG;

  typedef enum _SYSDBG_COMMAND {
    SysDbgReadVirtual = 8,
    SysDbgWriteVirtual = 9,
    SysDbgReadMsr = 16,
    SysDbgWriteMsr = 17,
  }

  HANDLE WINAPI OpenServiceA(
    HANDLE hSCManager,
    LPCSTR lpServiceName,
    DWORD dwDesiredAccess
  );
EOS
end
```


MSR: from Ruby to kernel device

```
def readmsr(addr)
  msr = alloc_c_struct('SYSDBG.MSR', 'msraddress' => addr)

  kldbg = alloc_c_struct('KLDBG')
  kldbg['dbgcommandclass'] = SYSDBGREADMSR
  kldbg['dbgcommand'] = msr
  kldbg['dbgcommandlen'] = msr.length

  lpBytesReturned = 0.chr*8

  deviceiocontrol(@hdevice, IOCTL, kldbg, kldbg.length, \
    msr, msr.length, lpBytesReturned, NULL)

  return msr['data']
end
```

Glue the bricks together

- Tweak debugger's interpretation of INT1
 - Handle trap flag logic
 - Add support for MSR:
 - Load kldbdrv driver
 - Read/Write the IA32_DEBUGCTL MSR
 - Integrate within Bintrace module

 - Keep in mind that MSR are core-specific on SMP systems
- ⇒ OMGWTFBBQ

Port to Linux

- MSR interaction trivial, via mainline `msr` LKM
creates char device: `/dev/cpu/*/msr`
- ~30 lines of Ruby code

Demo

Branch tracing of `hostname.exe`

Plan

- 1 Metasm
- 2 Tracer
- 3 MSR
- 4 NIC

Broadcom NIC firmware

- Runs on the card
 - Independant MIPS processor
 - Dedicated ram&flash
 - Controlled from host through MMIO
-
- Already examined by Perez&Dufлот (CSW10)
(custom-designed lowlevel interface)
 - Guillaume Delugré, from our lab, has done similar works
 - a custom LKM exposes a char device
 - read/write to card memory space via a cli interface
 - set breakpoints, read regs, etc. through MMIO

NICDBG

- Metasm comes with a MIPS disassembler
 - ⇒ **Get a disassembler/debugger GUI for free**
- Include Bintrace modules in the debugger
 - ⇒ **Record live trace of execution**

Demo

NIC debugging
Trace card initialisation routines

Conclusion

Metasm is good for you

- Covers lots of everyday reversing tasks
- Avoid reinventing the wheel
- Plan, do, check, act
- The tracing module will be published soon⁴

⁴<http://esec.fr.sogeti.com/blog/>

Questions ?

<http://metasm.cr0.org/>

<http://esec.fr.sogeti.com/blog/>