

# 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 IRC<sup>1</sup>
- yadda yadda

---

<sup>1</sup><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/>
- Trainings (CSW, HITB)

## 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'

w = Metasm::Gui::DasmWindow.new('recon-cfp', dasm, 'buf')

Metasm::Gui.main
```



# 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`<sup>a</sup>  
 $\Rightarrow$  Trivial to store execution traces

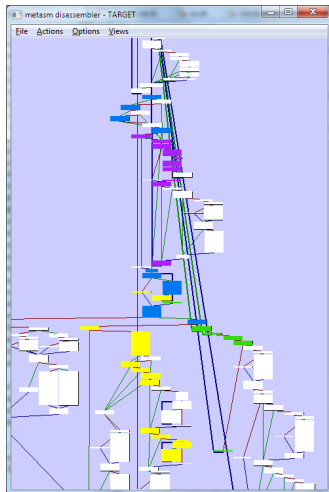
---

<sup>a</sup>Currently 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 <sup>2</sup>
- Same concepts can be applied on Windows 7 through the KD module<sup>3</sup>:
  - Issue IOCTLs to `\\.\kldbgdrv` device
  - Need to boot with `/DEBUG` option

---

<sup>2</sup>[http://www.openrce.org/blog/view/535/Branch\\_Tracing\\_with\\_Intel\\_MSR\\_Registers](http://www.openrce.org/blog/view/535/Branch_Tracing_with_Intel_MSR_Registers)

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

# DynLdr

We'd like to call `Kernel132!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'] = SYSDBG_READMSR
  kldbg['dbgcommand'] = msr
  kldbg['dbgcommandlen'] = msr.length

  lpBytesReturned = 0.chr*8

  deviceioctl(@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 soon<sup>4</sup>

### Thanks

- to Guillaume & Ivan for their work
- all the lab for their feedback
- bruce & natron for IRC animation

---

<sup>4</sup><http://esec.fr.sogeti.com/blog/>



## Questions ?

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

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