Hardware-Assisted Malware Analysis

Marcus Botacin[1], André Grégio[3], Paulo Lício de Geus[2]

[1]Msc. Candidate
Institute of Computing - UNICAMP
{marcus}@lasca.ic.unicamp.br

[2]Advisor
Institute of Computing - UNICAMP
{paulo}@lasca.ic.unicamp.br

[3]Co-Advisor
Federal University of Paraná (UFPR)
gregio@inf.ufpr.br

July 28^{th}, 2017

Topics

Topics

Malware Threats.

Figure: Washington Post: `https://tinyurl.com/ljo7ekm`

Figure: BBC: `https://tinyurl.com/mfogjhe`

Topics

Malware Analysis.

Figure: Imperva: `https://tinyurl.com/zkbsnl2`

Figure: Enigma: `https://tinyurl.com/kydgwve`

More specifically...

Static Analysis

Analyzing without executing it.

Dynamic Analysis

Analyzing by executing it.

Topics

Malware Analysis.

PetrWrap Crypto Ransomware Blocks Security Researchers From Reverse Engineering Code Samples

JP Buntinx March 16, 2017 News, Security

Figure: Themerkle: `https://tinyurl.com/kasuxcr`

MAY 13, 2017 @ 04:01 AM 95,046 The Little Black I

How One Simple Trick Just Put Out That Huge Ransomware Fire

Figure: Forbes: `https://tinyurl.com/l7ecrex`

What happens to static analysis?

Control Flow Obfuscation

Listing 1: Control flow obfuscation example.

```
1 entry:  xor eax, eax
2         jnz     %l1
3 l1:     xor eax, eax
4         jnz %l2
5 l2:     xor eax, eax
6         jnz %l3
7 ln:     ...
```

What happens to static analysis?

Path Explosion

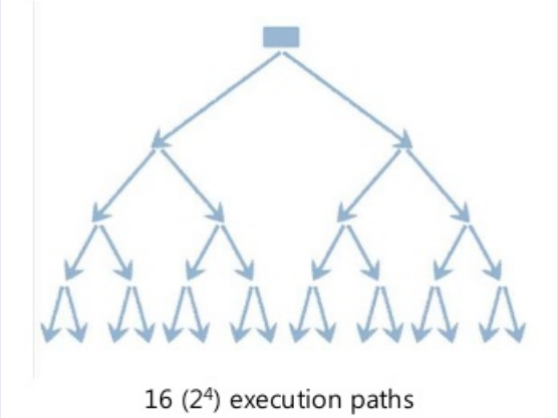


Figure: Source Image: `https://tinyurl.com/n52fz5r`

What happens to dynamic analysis?

Debugger Detection

Listing 2: Anti-debug example.

```
1 if(IsDebuggerPresent())
2         printf("debugged\n");
```

VM Detection

Listing 3: Anti-vm example.

```
1 cpuid(0x40000000, &eax, &ebx, &ecx, &edx);
2 memcpy(hyper_vendor_id + 0, &ebx, 4);
3 if (!strcmp(hyper_vendor_id, "VMwareVMware"))
4         printf("VM\n");
```

And then...

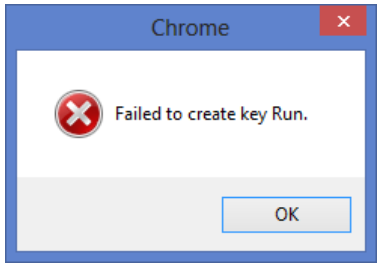


Figure: Real malware claiming a registry problem when an anti-analysis trick succeeded.

And then...

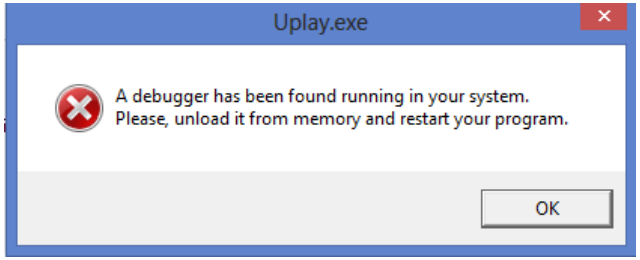


Figure: Commercial solution armored with anti-debug technique.

And then...

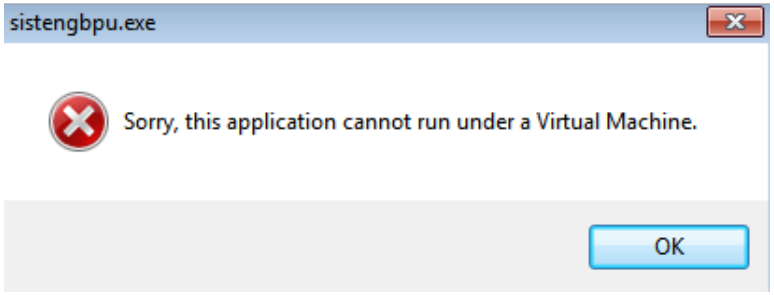


Figure: Real malware impersonating a secure solution which cannot run under an hypervisor.

Topics

Definitions

1. **Higher privileged.**
2. **No non-privileged side-effects.**
3. **Identical Basic Instruction Semantics.**
4. **Transparent Exception Handling.**
5. **Identical Measurement of Time.**

Topics

HVM

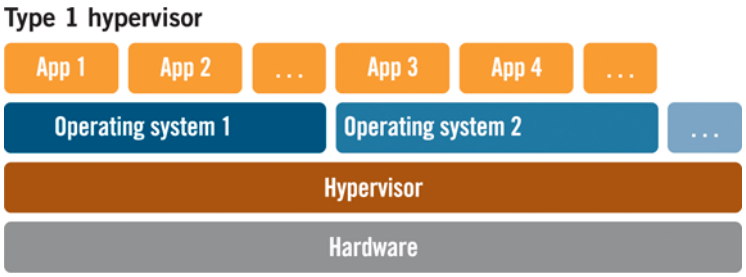


Figure 2. A Type 1 or bare-metal hypervisor sits directly on the host hardware.

Figure: Type-1 Hypervisor. Source: `https://tinyurl.com/ku7oho5`

HVM

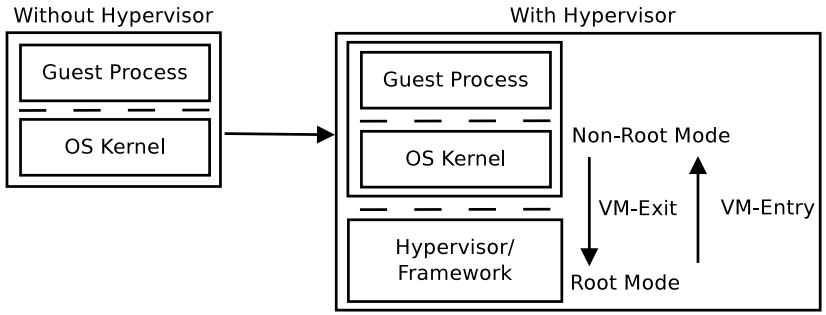


Figure: HVM operating layers

.

HVM

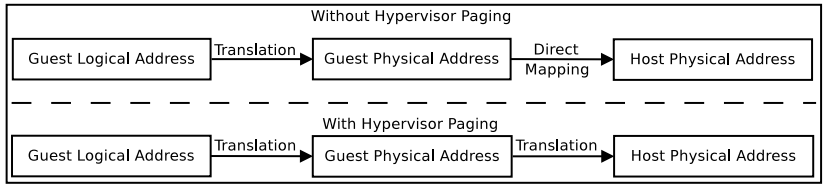


Figure: HVM memory operation.

HVM

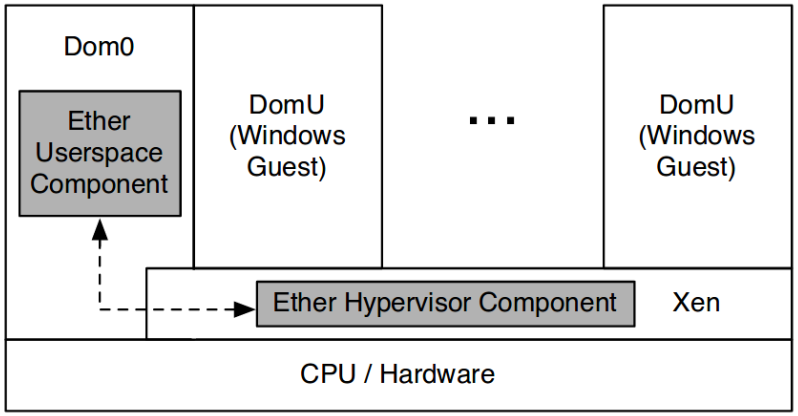


Figure: Ether's architecture.

HVM

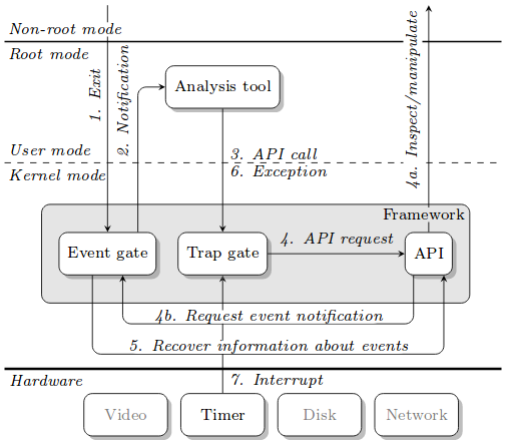


Figure: Fattori's architecture.

HVM

Event	Description	Arguments
`ProcessSwitch`	Context (process) switch	—
`Exception`	Execution	Exception vector, faulty instruction, error code
`Interrupt`	Hardware or software interrupt	Interrupt vector, requesting instruction
`BreakpointHit`	Execution breakpoint	Breakpoint address
`WatchpointHit`	Watchpoint on data read/write	Watchpoint address, access type, hitting instruction
`FunctionEntry`	Function call	Function name/address, caller/return address
`FunctionExit`	Return from function	Function name/address, return address
`SyscallEntry`	System call invocation	System call number, caller/return address
`SyscallExit`	Return from system call	System call number, return address
`IOOperationPort`	I/O operation throught hardware port	Port number, access type
`IOOperationMmap`	Memory-mapped I/O operation	Memory address, access type

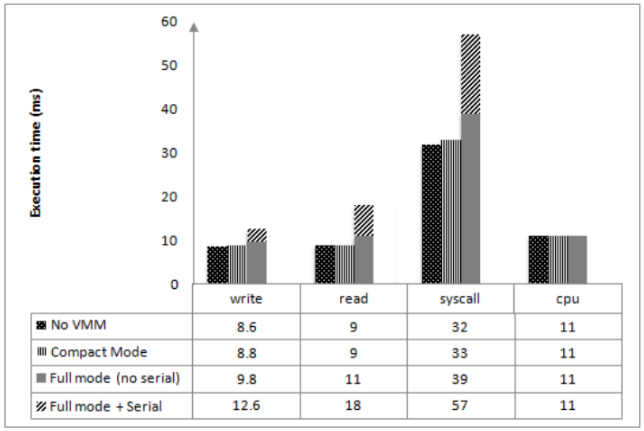
Figure: Fattori's traceable events.

HVM

Event	Exit cause	Native exit
`ProcessSwitch`	Change of page table address	√
`Exception`	Exception	√
`Interrupt`	Interrupt	√
`BreakpointHit`	Debug except. / Page fault except.	
`WatchpointHit`	Page fault except.	
`FunctionEntry`	Breakpoint on function entry point	
`FunctionExit`	Breakpoint on return address	
`SyscallEntry`	Breakpoint on syscall entry point	
`SyscallExit`	Breakpoint on return address	
`IOOperationPort`	Port read/write	√
`IOOperationMmap`	Watchpoint on device memory	

Figure: Fattori's techniques.

HVM



	write	read	syscall	cpu
No VMM	8.6	9	32	11
Compact Mode	8.8	9	33	11
Full mode (no serial)	9.8	11	39	11
Full mode + Serial	12.6	18	57	11

Figure: MAVMM's overhead.

SMM

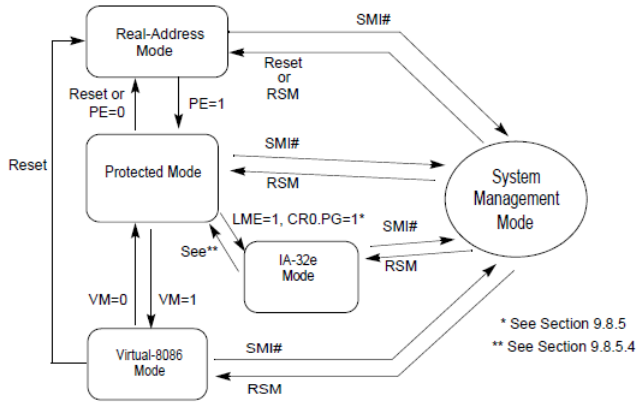


Figure: Operation modes. Source: `https://tinyurl.com/l2uqr8d`

SMM

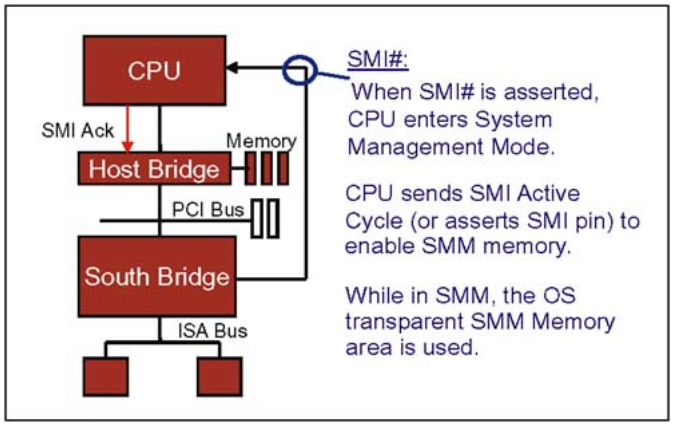


Figure: SMIs. Source: `https://tinyurl.com/kuqlmau`

Battle of the rings

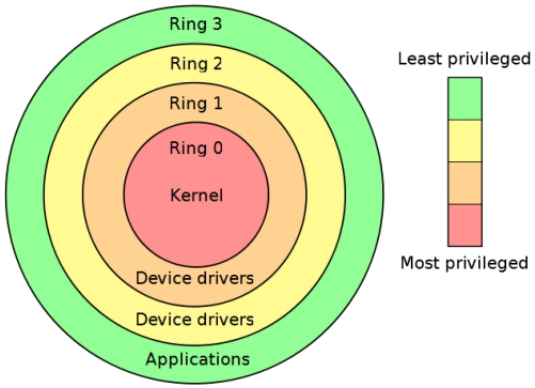


Figure: Processor rings. Source: `https://tinyurl.com/mzjezkg`

AMT/ME

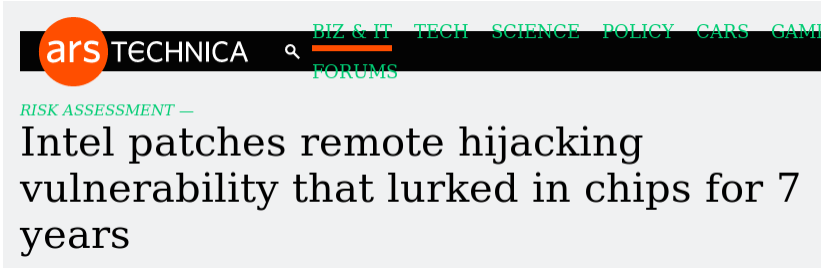


Figure: Arstechnica. Source: `https://tinyurl.com/kc62fw8`

SGX

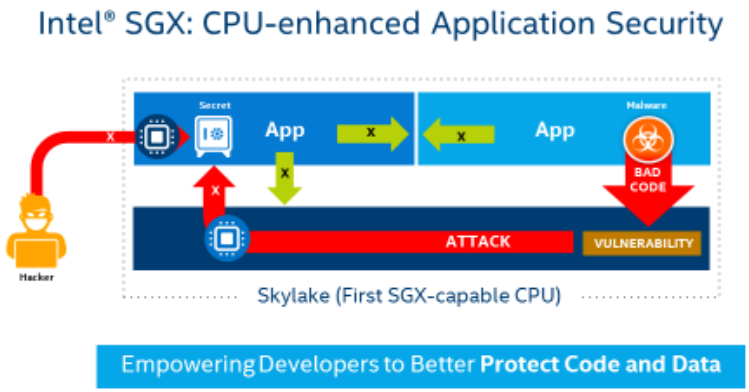


Figure: SGX enclaves. Source: `https://tinyurl.com/ln6eb4g`

External Hardware

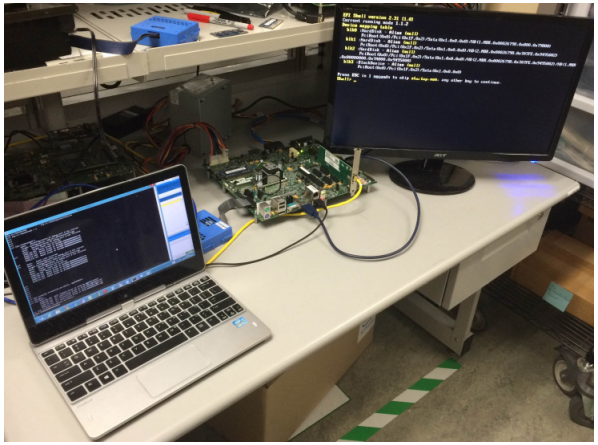


Figure: JTAG. Source: `https://tinyurl.com/m8r79e8`

Branch Monitor

FROM	TO
0x0FFFF418	0x0FFF8F36
0x0FFF1510	0x0FFFCF2E
0x0FFF8014	0x0FFF0523
0x0FFF81b3	0x0FFFE057

Figure: Branch Stack.

Performance Monitor

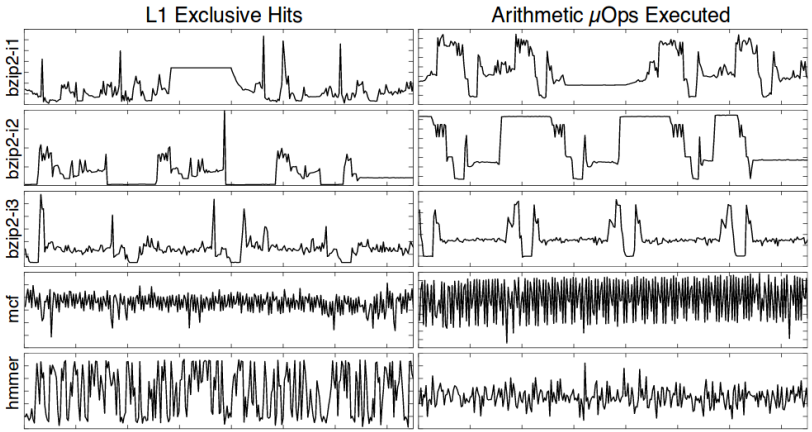


Figure: PMU-based side-effect detection.
Source: `https://tinyurl.com/m9zgphk`

GPUs

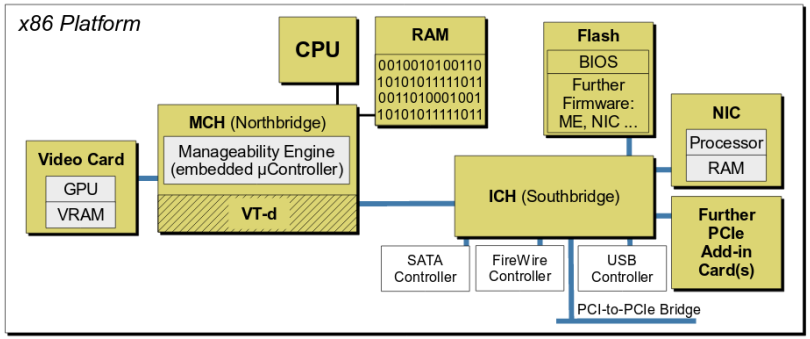


Figure: GPU. Source: `https://tinyurl.com/lnhufn9`

A Survey

TSX

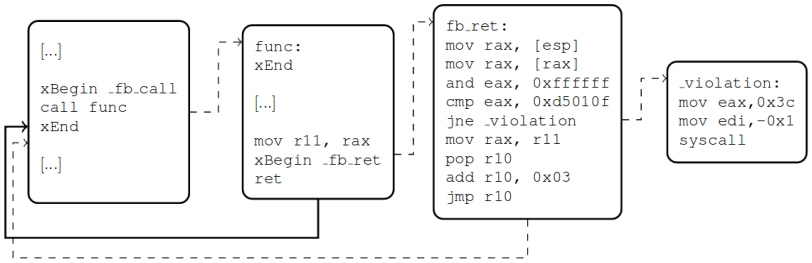


Figure: TSX-based CFI. Source: `https://tinyurl.com/l8jfycj`

Topics

Proposed Framework

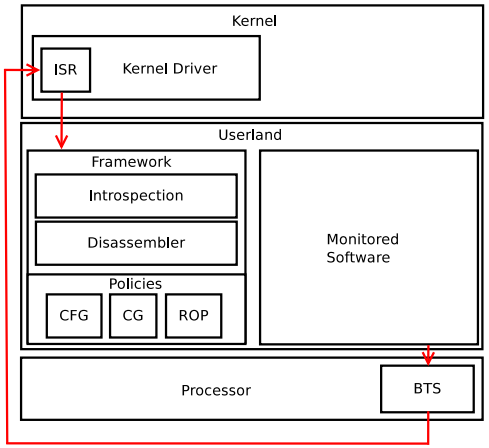


Figure: Proposed framework architecture.

Could I develop a performance-counter-based malware analyzer?

Could I isolate processes' actions?

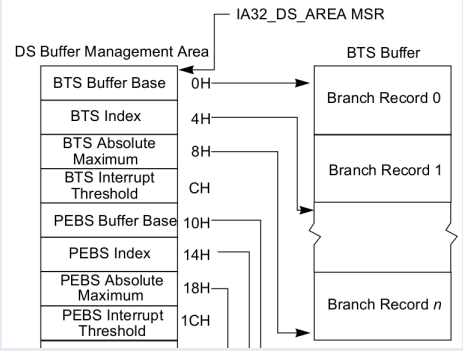


Figure: Data Storage (DS) AREA.

Could I develop a performance-counter-based malware analyzer?

Could I isolate processes' actions?

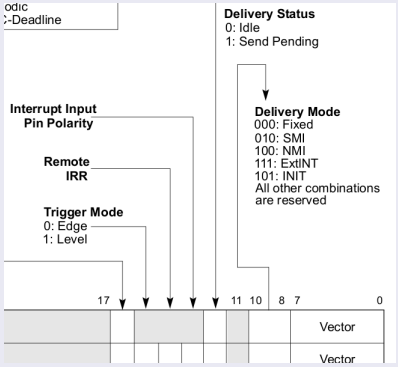


Figure: Local Vector Table (LVT).

Could I develop a performance-counter-based malware analyzer?

Is CG reconstruction possible?

Table: ASLR - Library placement after two consecutive reboots.

Library	NTDLL	KERNEL32	KERNELBASE
Address 1	0xBAF80000	0xB9610000	0xB8190000
Address 2	0x987B0000	0x98670000	0x958C0000

Could I develop a performance-counter-based malware analyzer?

Is CG reconstruction possible?

Table: Function Offsets from `ntdll.dll` library.

Function	***Offset***
NtCreateProcess	0x3691
NtCreateProcessEx	0x30B0
NtCreateProfile	0x36A1
NtCreateResourceManager	0x36C1
NtCreateSemaphore	0x36D1
NtCreateSymbolicLinkObject	0x36E1
NtCreateThread	0x30C0
NtCreateThreadEx	0x36F1

Could I develop a performance-counter-based malware analyzer?

Is CG reconstruction possible?

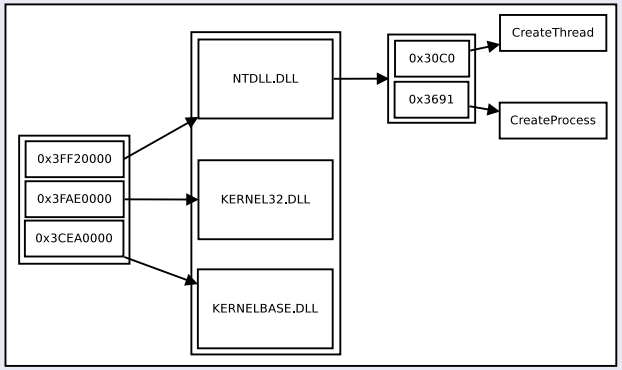


Figure: Introspection Mechanism.

Could I develop a performance-counter-based malware analyzer?

Is CG reconstruction possible?

Figure: Step Into.

Figure: Step Over.

Is CFG reconstruction possible?

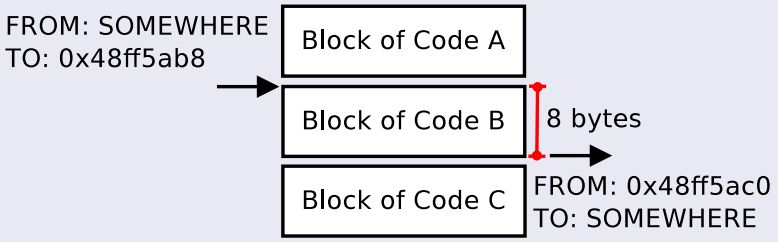


Figure: Code block identification.

Is the final solution transparent?

Identified tricks

Listing 4: Fake Conditional.

```
1 0x190 xor eax, eax
2 0x192 jnz 0x19c
```

Listing 5: Control Flow Change.

```
1 0x180 push 0x10a
2 0x185 ret
```

Is the final solution transparent?

Identified tricks

Listing 6: Hook Detection.

```
1 0x340 cmp eax,0xe9
2 0x345 jnz 0x347
```

Listing 7: Hardware Debugger Detection.

```
1 0x400 QWORD PTR fs:0x0,rsp
2 0x409 mov rax,QWORD PTR [rsp+0xc]
3 0x40e cmp rbx,QWORD PTR [rax+0x4]
```

Is the final solution transparent?

Deviating Behavior

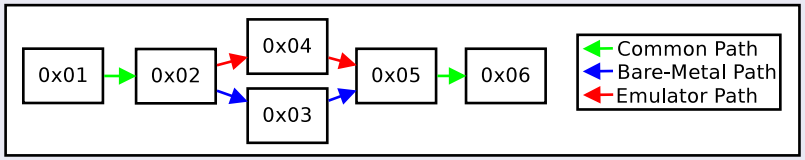


Figure: Deviating behavior identification.

Is the final solution transparent?

Deviating Behavior

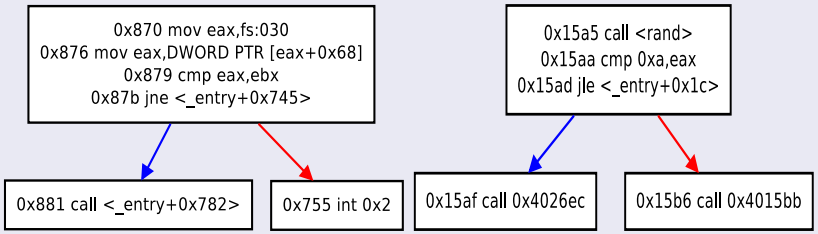


Figure: Divergence: True Positive.

Figure: Divergence: False Positive.

Could I develop a Debugger?

Inverted I/O

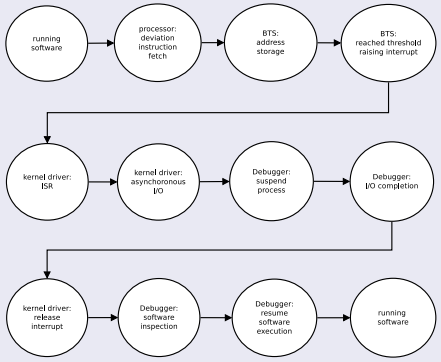


Figure: Debugger's working mechanism.

Could I develop a Debugger?

Suspending Processes

- EnumProcessThreads + SuspendThread.
- DebugActiveProcess.
- NtSuspendProcess.

Could I develop a Debugger?

Integration

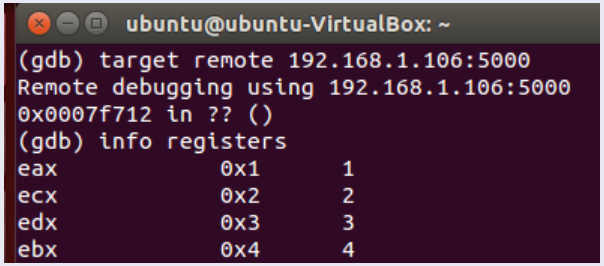


Figure: GDB integration.

Does the solution handle ROP attacks?

ROP Attacks

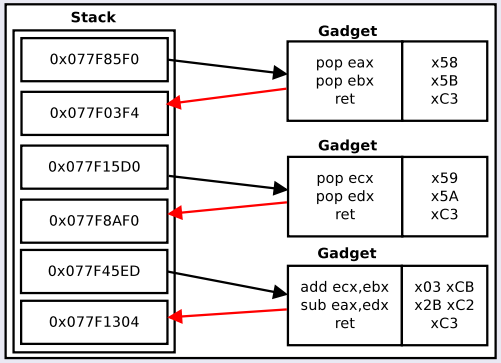


Figure: ROP chain example.

Does the solution handle ROP attacks?

CALL-RET Policy

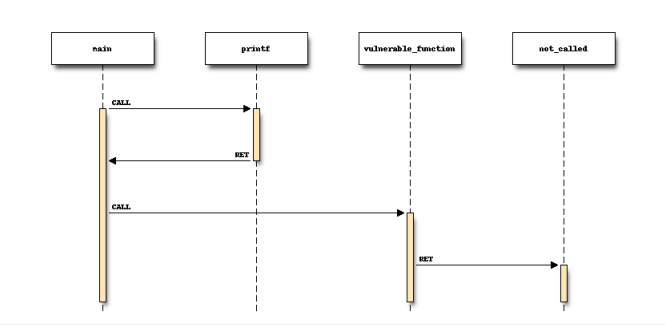


Figure: CALL-RET CFI policy.

Does the solution handle ROP attacks?

Gadget-size policy

LBR Stack

	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

Figure: KBouncer's exploit stack.

Does the solution handle ROP attacks?

Exploit Analysis

Table: Excerpt of the branch window of the ROP payload.

FROM	TO
—-	0x7c346c0a
0x7c346c0b	0x7c37a140
0x7c37a141	—-

Does the solution handle ROP attacks?

Exploit Analysis

Listing 8: Static disassembly of the `MSVCR71.dll` library.

```
1 7c346c08: f2 0f 58 c3        addsd  %xmm3,%xmm0
2 7c346c0c: 66 0f 13 44 24 04  movlpd %xmm0,0x4(%esp)
```

Listing 9: Dynamic disassembly of the `MSVC71.dll` executed code.

```
1 0x1000 (size=1)  pop    rax
2 0x1001 (size=1)  ret
```

Is the solution easy to implement?

Lines of Code comparison

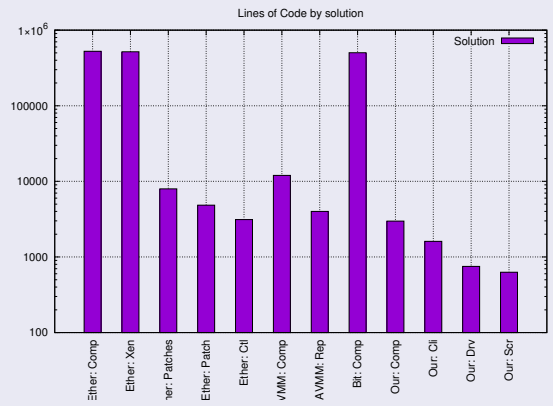


Figure: Lines of Code by solution.

Is the solution portable?

Talking about Linux

Listing 10: Linux BTS support (Source:https://tinyurl.com/mws6v8x

```
1  static __init int bts_init(void)
2  bts_pmu.capabilities    = PERF_PMU_CAP_AUX_NO_SG
       | PERF_PMU_CAP_ITRACE
3  bts_pmu.task_ctx_nr     = perf_sw_context;
4  bts_pmu.event_init      = bts_event_init;
5  bts_pmu.add             = bts_event_add;
6  bts_pmu.del             = bts_event_del;
7  bts_pmu.start           = bts_event_start;
8  bts_pmu.stop            = bts_event_stop;
9  bts_pmu.read            = bts_event_read;
10 return perf_pmu_register(&bts_pmu,
11 "intel_bts",-1)
```

Is the solution portable?

Talking about Linux

Listing 11: Linux BTS support (Source:https://tinyurl.com/mx759d7)

```
1 perf_init(&pe, MMAP_PAGES);
2 fcntl(gbl_status.fd_evt, F_SETOWN,
      getpid());
3 monitor_loop(pid_child, s_outfile);
```

Is solution's overhead acceptable?

Could the solution run in real-time?

Task	Base value	System monitoring	Penalty	Benchmark monitoring	Penalty
Floating-point operations (op/s)	101530464	99221196	2.27%	97295048	4.17%
Integer operations (op/s)	285649964	221666796	22.40%	219928736	23.01%
MD5 Hashes (hash/s)	777633	568486	26.90%	568435	26.90%
RAM transfer (MB/s)	7633	6628	13.17%	6224	18.46%
HDD transfer (MB/s)	90	80	11.11%	75	16.67%
Overall (benchm. pt)	518	470	9.27%	439	15.25%

Topics

Lessons learned

- Transparency is essential.
- Hardware-assisted approaches may fulfill transparency requirements.
- There are open problems on hardware monitoring.
- Security, performance, and development efforts as trade-offs (really?).
- Performance monitors as lightweight alternatives.

Topics

And now?

- Multi-core monitor.
- Linux Monitor.
- Malware clustering.

Solution's Availability

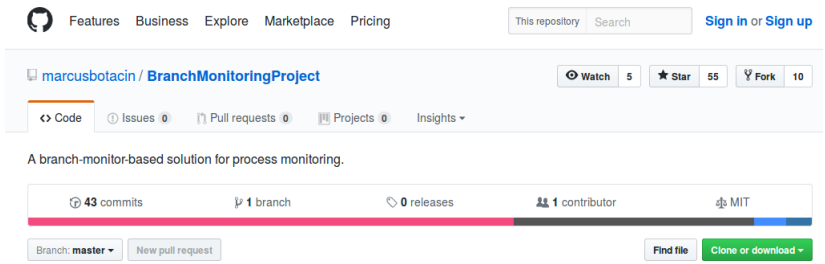


Figure: Solution's Availability. Solution is public on github. `https://github.com/marcusbotacin/BranchMonitoringProject`

Topics

Questions?