Monitoring Systems & Binaries

Marcus Botacin[1]

[1]Informatics - Federal University of Parana (UFPR) - Brazil
mfbotacin@inf.ufpr.br

November 2018

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

About Me

- Malware Analyst (2012)
- BsC. Computer Engineer @ UNICAMP (2015)
    - Sandbox Development
- MsC. Computer Science @ UNICAMP (2017)
    - Hardware-Assisted Malware Analysis
- PhD. Computer Science @ UFPR (Present)
    - Hardware-Assisted Malware Detection
    - AntiVirus Evaluation
    - Future Threats
    - **Contextual and Social Malware effects**

Why Monitoring ?

- Policy Enforcement
- Logging
- Forensics
- Debugging
- Malware Analysis
- Reverse Engineer

Real Trace Examples

```
7/4/2014-13:5:1.895|DeleteOperation|2032|C:\deposito.exe|C:\ProgramData\rr.txt|
```

```
7/4/2014-13:3:48.294|CreateProcess|3028|C:\Monitor\Malware\visualizar.exe|2440|C:\Windows\SysWOW64\dll.exe
```

```
2014-05-14 20:02:40.963113      10.10.100.101    XX.YY.ZZ.121      HTTP      290      GET /.swim01/control.php?ia&mi=00B5AB4E-47098BC3 HTTP/1.1
```

Agenda

1 Introduction

2 Software-Based Solutions

3 Evasion Techniques

4 Hardware-Assisted Solutions

5 Attacks

6 Conclusions

7 Extra

Function Interposition

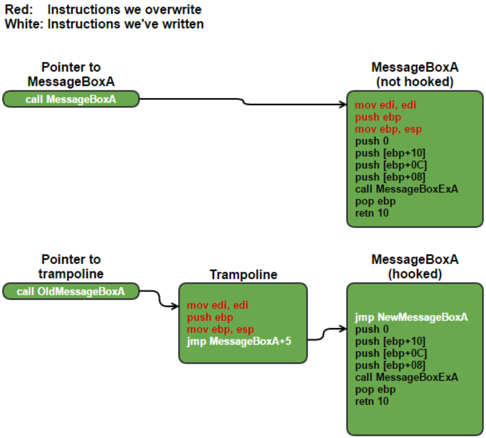


Figure: Source: `https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-1.html`

Techniques I

Kernel Tables

- System Service Dispatch Table (SSDT)
- Interrupt Descriptor Table (IDT)
- Global Descriptor Table (GDT)

Userland Tables

- API hooking
- DLL injection

Techniques II

Binary Patching

- Inline hooking

OS Support

- Detours
- Callbacks

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

In Practice...

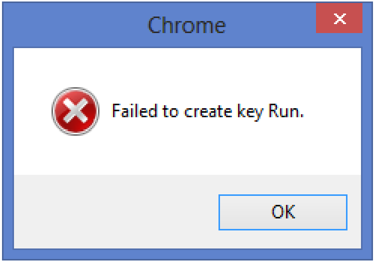


Figure: Real malware claiming a registry problem when an anti-analysis trick succeeded.

In Practice...

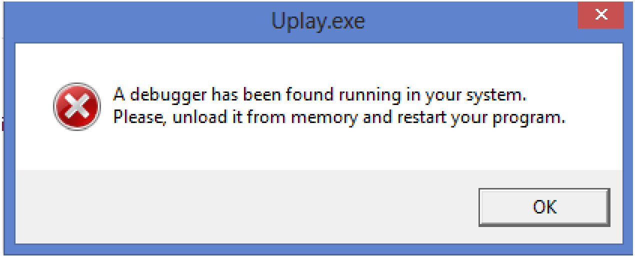


Figure: Commercial solution armored with anti-debug technique.

In Practice...

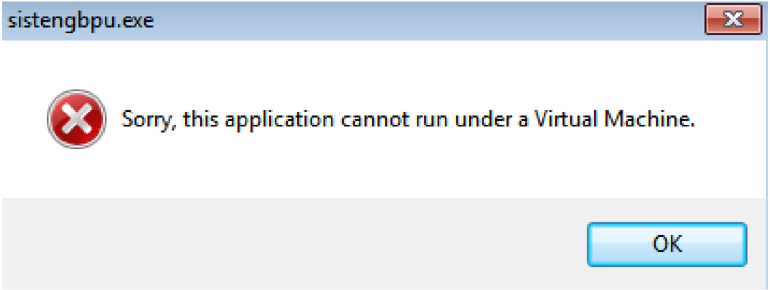


Figure: Real malware impersonating a secure solution which cannot run under an hypervisor.

Detecting Analysis Procedures

```
if(IsDebuggerPresent())
        printf("debugged\n");
else
        printf("NO DBG\n");
```

```
cmp  [eax+0xe9], eax  ;; 0xe9 = JMP
pop      rbp
```

Anti-Analysis Summary

Table: **Anti-Analysis**: Tricks summary. Malware samples may employ multiple techniques to evade distinct analysis procedures.

Technique	Description	Reason	Implementation
Anti Debug	Check if running inside a debugger	Blocks reverse engineering attempts	Fingerprinting
Anti VM	Check if running inside a VM	Analysts use VMs for scalability	Execution Side-effect
Anti Disassembly	Fool disassemblers to generate wrong opcodes	AV signatures may be based on opcodes	Undecidable Constructions

Agenda

1 Introduction

2 Software-Based Solutions

3 Evasion Techniques

4 Hardware-Assisted Solutions

5 Attacks

6 Conclusions

7 Extra

Transparency

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

Hardware Features Summary

Technique	PROS	CONS	Gaps
HVM	Ring -1	Hypervisor development	High overhead
SMM	Ring -2	BIOS development	High implementation cost
AMT	Ring -3	Chipset code change	No malware analysis solution
HPCs	Lightweight	Context-limited information	No malware analysis solution
GPU	Easy to program	No register data	No introspection procedures
SGX	Isolates goodware	Also isolates malware	No enclave inspection
SOCs	Tamper-proof	Passive components	Raise alarms

HVM

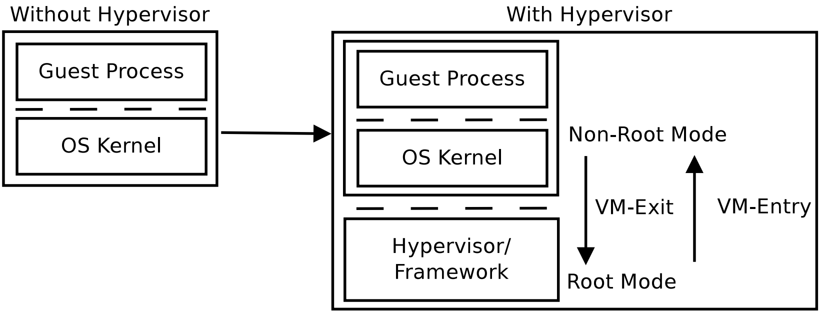


Figure: HVM operating layers

.

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: Ether Sandbox Exits.

SMM

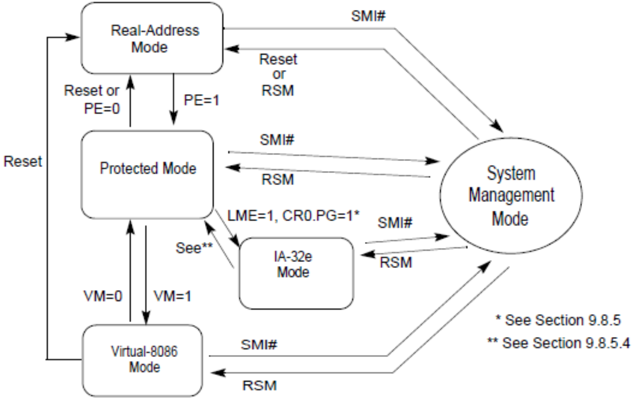


Figure: Operation modes. Source: `https://tinyurl.com/l2uqr8d`

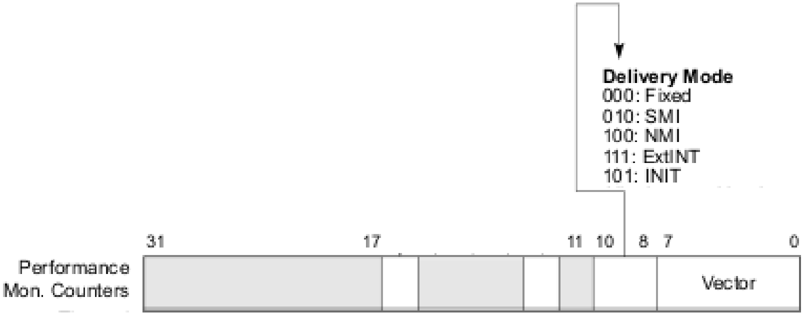


Figure: **SMI generation.**

A ring to rule them all!

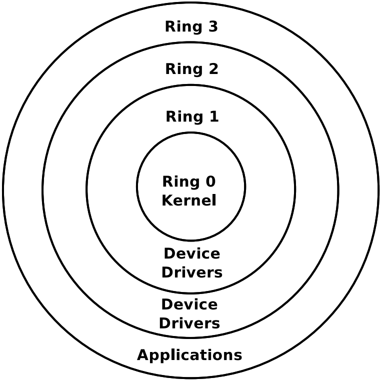


Figure: Privileged rings.

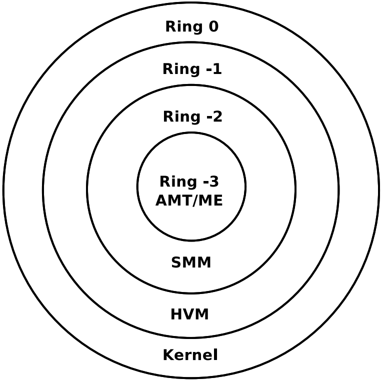


Figure: New privileged rings.

Isolated Enclaves

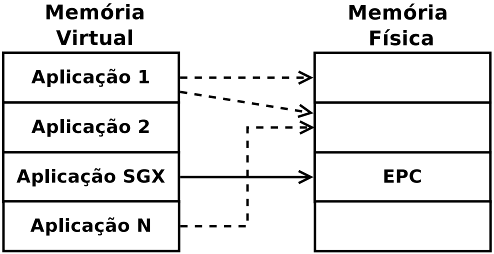


Figure: SGX Memory Protection

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

DMA Attacks I

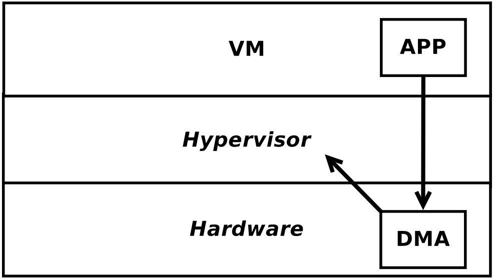
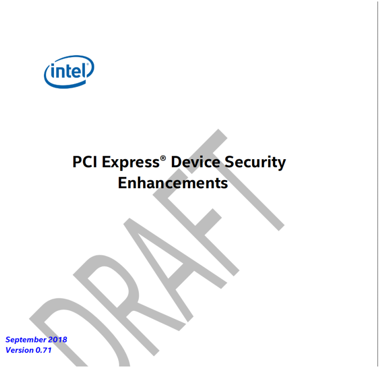


Figure: Hypervisor Attack

.

DMA Attacks II



intel

PCI Express® Device Security Enhancements

DRAFT

September 2018
Version 0.71

Figure: Source: `https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/pcie-device-security-enhancements.pdf`

SGX Malware

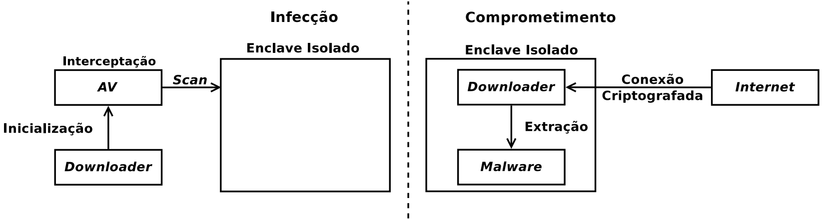


Figure: SGX Malware

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

References

- *Who watches the watchmen: A security-focused review on current state-of-the-art techniques, tools and methods for systems and binary analysis on modern platforms*—ACM Computing Surveys.
- *Enhancing Branch Monitoring for Security Purposes: From Control Flow Integrity to Malware Analysis and Debugging*—ACM Transactions on Privacy and Security.
- *The other guys: automated analysis of marginalized malware*—Journal of Computer Virology and Hacking techniques.

Conclusions

- Thanks Tilo for hosting me.
- Open to hear your questions.

Agenda

1. Introduction
2. Software-Based Solutions
3. Evasion Techniques
4. Hardware-Assisted Solutions
5. Attacks
6. Conclusions
7. Extra

Proposed Framework

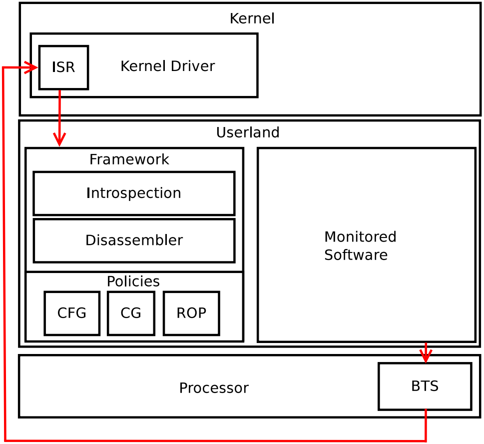


Figure: Proposed framework architecture.

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

Could I isolate processes' actions?

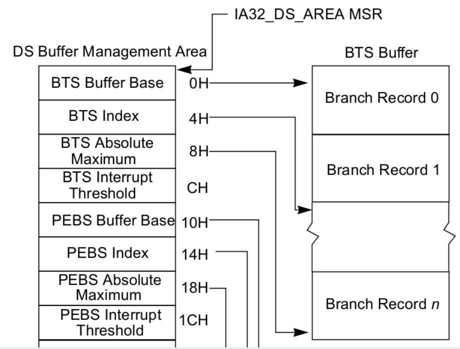


Figure: Data Storage (DS) AREA.

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

Could I isolate processes' actions?

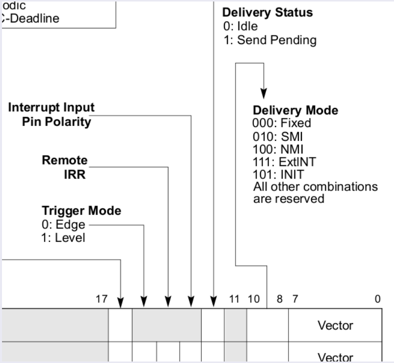


Figure: Local Vector Table (LVT).

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?

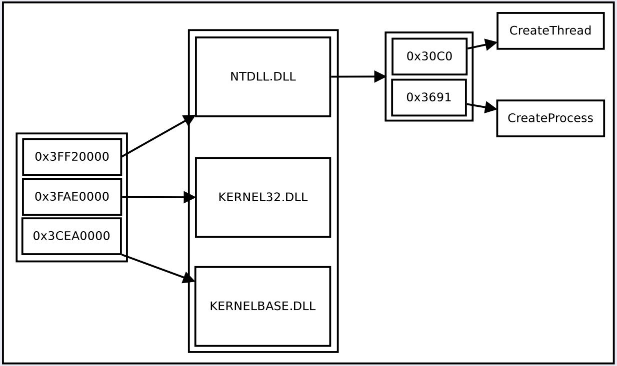


Figure: Introspection Mechanism.

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

Is CG reconstruction possible?

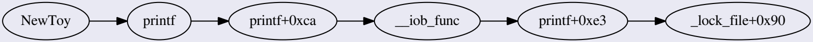


Figure: Step Into.

Figure: Step Over.

Is CFG reconstruction possible?

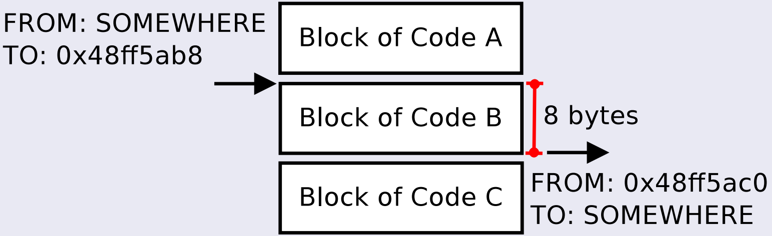


Figure: Code block identification.

Is the final solution transparent?

Identified tricks

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

```
0x180 push 0x10a
0x185 ret
```

Is the final solution transparent?

Identified tricks

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

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

Is the final solution transparent?

Deviating Behavior

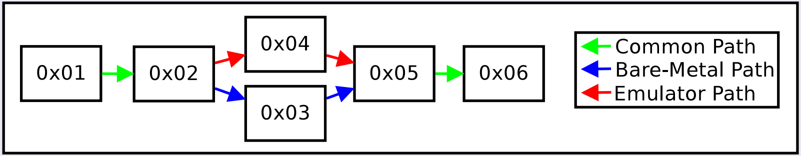


Figure: Deviating behavior identification.

Is the final solution transparent?

Deviating Behavior

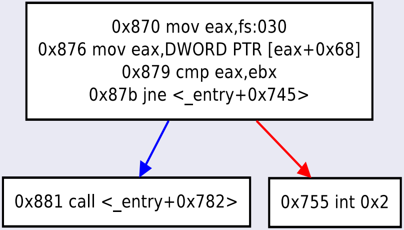


Figure: Divergence: True Positive.

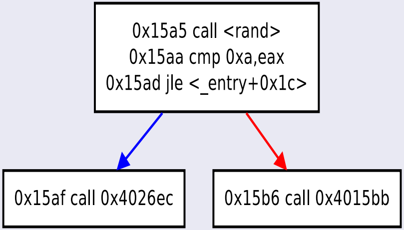


Figure: Divergence: False Positive.

Could I develop a Debugger?

Inverted I/O

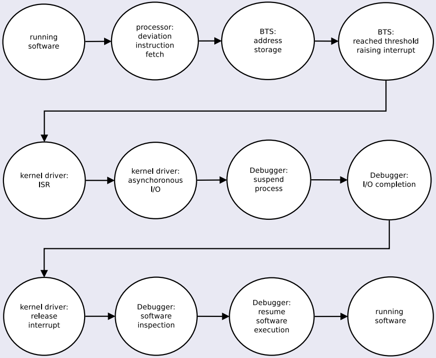


Figure: Debugger's working mechanism.

Could I develop a Debugger?

Suspending Processes

- EnumProcessThreads + SuspendThread.
- DebugActiveProcess.
- NtSuspendProcess.

Could I develop a Debugger?

Integration

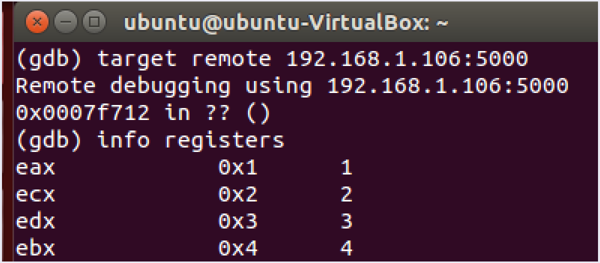


Figure: GDB integration.

Does the solution handle ROP attacks?

ROP Attacks

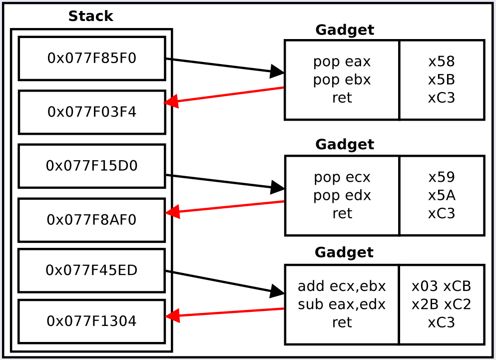


Figure: ROP chain example.

Does the solution handle ROP attacks?

CALL-RET Policy

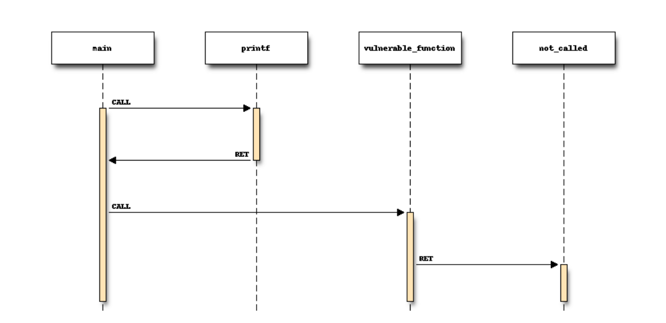


Figure: CALL-RET CFI policy.

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

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

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

Is the solution easy to implement?

Lines of Code comparison

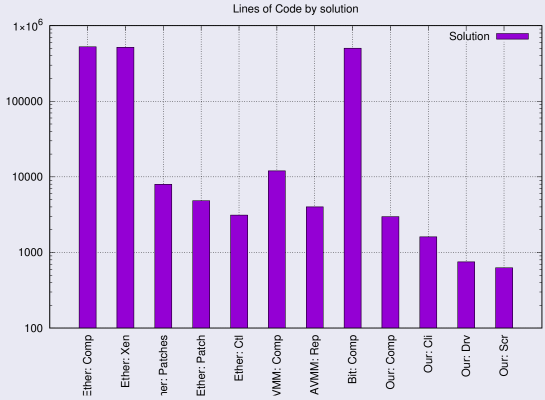


Figure: Lines of Code by solution.

Is the solution portable?

Talking about Linux

```
static __init int bts_init(void)
bts_pmu.capabilities      = PERF_PMU_CAP_AUX_NO_SG
      | PERF_PMU_CAP_ITRACE
bts_pmu.task_ctx_nr       = perf_sw_context;
bts_pmu.event_init        = bts_event_init;
bts_pmu.add               = bts_event_add;
bts_pmu.del               = bts_event_del;
bts_pmu.start             = bts_event_start;
bts_pmu.stop              = bts_event_stop;
bts_pmu.read              = bts_event_read;
return perf_pmu_register(&bts_pmu,
"intel_bts",-1)
```

Is the solution portable?

Talking about Linux

```
perf_init(&pe, MMAP_PAGES);
fcntl(gbl_status.fd_evt, F_SETOWN,
    getpid());
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%