Towards Transparent Dynamic Binary Instrumentation using Virtual Machine Introspection

Instrumenting via the Hypervisor

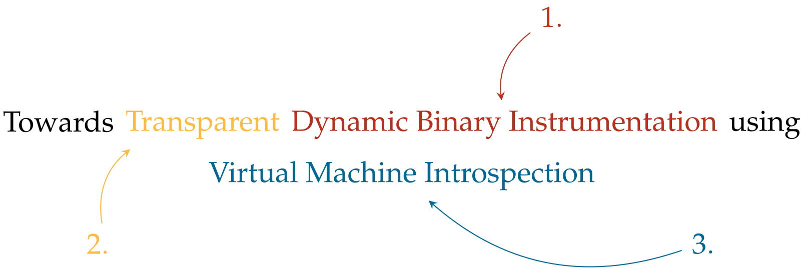
Julian Kirsch

June 19, 2015

About me

- PhD student at Technische Universität München
  - Research: RE techniques in general, deobfuscation & malware analysis in particular (always looking for cool project ideas!)
  - Teaching: advanced binary exploitation, rootkit programming
- Capture-The-Flag addict playing for the H4x0rPsch0rr team (ranked #12 on `https://ctftime.org`'s world-ranking in 2k14)
- Writeups for past RE tasks: `http://hxp.io`
- Contact: `kirschju@sec.in.tum.de`

This Talk



1.

Towards Transparent Dynamic Binary Instrumentation using

Virtual Machine Introspection

2.

3.

Dynamic Binary Instrumentation

A Simple Example

```
_start:
  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob

; do something interesting ...

  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob
  ret
```

```
_my_memfrob:
  xor byte [rdi+rcx-1], 0x42
  loop _my_memfrob
  ret
```

```
  section .data
memfrobbed:
  db 0x77, 0x71, 0x21, 0x30,
  db 0x71, 0x36, 0x1d, 0x32,
  db 0x76, 0x3b, 0x2e, 0x72,
  db 0x76, 0x26, 0x1d, 0x25,
  db 0x72, 0x71, 0x77, 0x1d,
  db 0x2a, 0x71, 0x30, 0x71
```

Dynamic Binary Instrumentation

A Simple Example

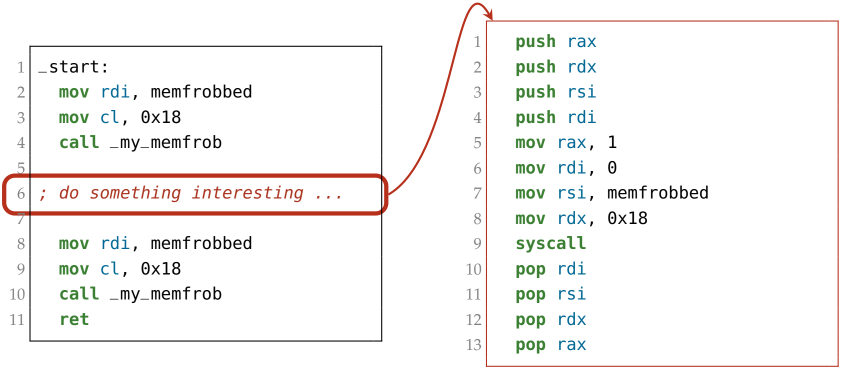
```
_start:
  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob

; do something interesting ...

  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob
  ret
```

```
_my_memfrob:
  xor byte [rdi+rcx-1], 0x42
  loop _my_memfrob
  ret
```

```
  section .data
memfrobbed:
  db 0x77, 0x71, 0x21, 0x30,
  db 0x71, 0x36, 0x1d, 0x32,
  db 0x76, 0x3b, 0x2e, 0x72,
  db 0x76, 0x26, 0x1d, 0x25,
  db 0x72, 0x71, 0x77, 0x1d,
  db 0x2a, 0x71, 0x30, 0x71
```

Dynamic Binary Instrumentation

A Simple Example



```
_start:
  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob

; do something interesting ...

  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob
  ret
```

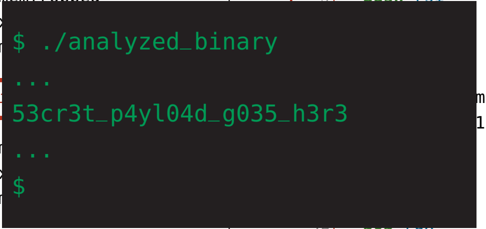
```
push rax
push rdx
push rsi
push rdi
mov rax, 1
mov rdi, 0
mov rsi, memfrobbed
mov rdx, 0x18
syscall
pop rdi
pop rsi
pop rdx
pop rax
```

Dynamic Binary Instrumentation

A Simple Example

```
_start:
  mov rdi, memfrobbed
  mov cl, 0x
  call _my_r

; do somethi

  mov rdi, r
  mov cl, 0x
  call _my_r
  ret
```

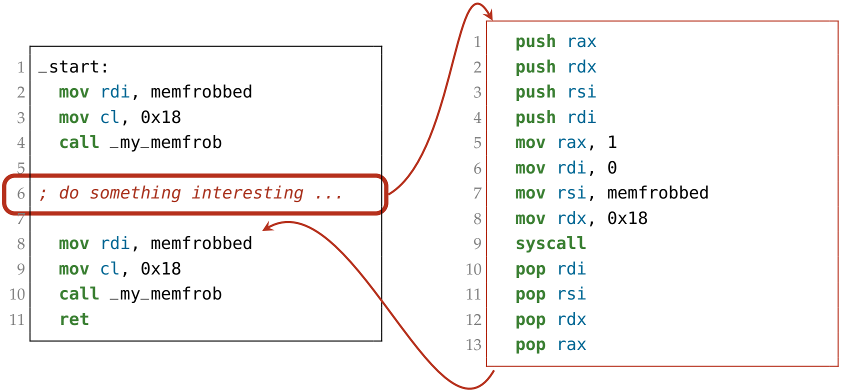
```
push rax
push rdx
push rsi




         mfrobbed
         18




pop rax
pop rax
```



```
$ ./analyzed_binary
...
53cr3t_p4yl04d_g035_h3r3
...
$
```

Dynamic Binary Instrumentation

A Simple Example

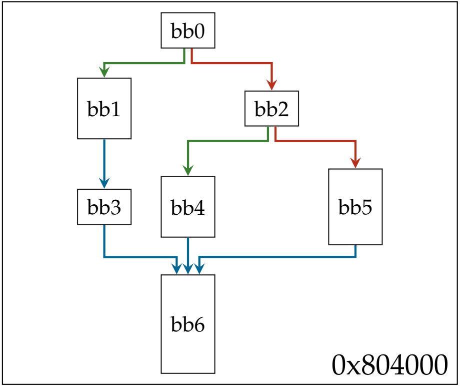


```
_start:
  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob

; do something interesting ...

  mov rdi, memfrobbed
  mov cl, 0x18
  call _my_memfrob
  ret
```

```
  push rax
  push rdx
  push rsi
  push rdi
  mov rax, 1
  mov rdi, 0
  mov rsi, memfrobbed
  mov rdx, 0x18
  syscall
  pop rdi
  pop rsi
  pop rdx
  pop rax
```

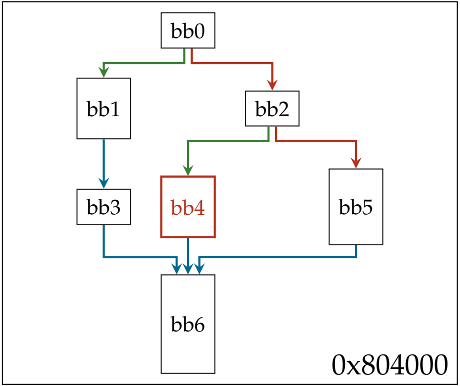
Dynamic Binary Instrumentation

Inner Workings & Code Caches



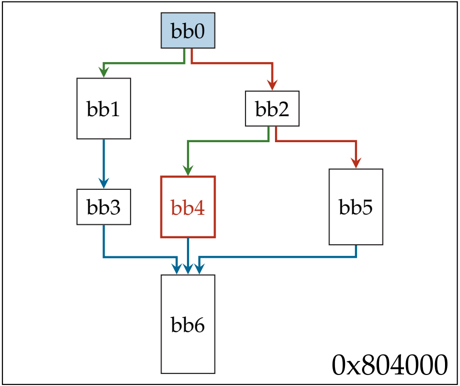
Dynamic Binary Instrumentation

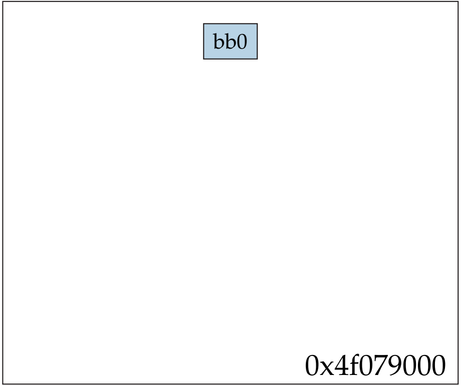
Inner Workings & Code Caches



Dynamic Binary Instrumentation

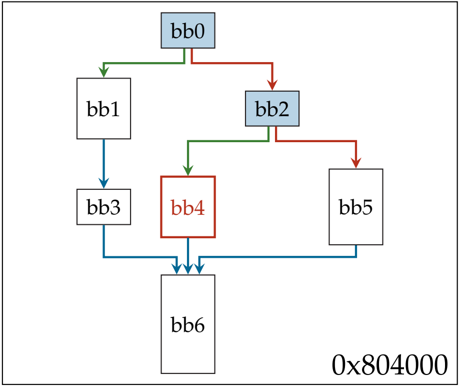
Inner Workings & Code Caches

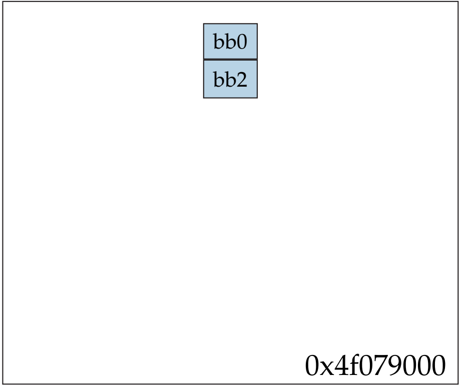




Dynamic Binary Instrumentation

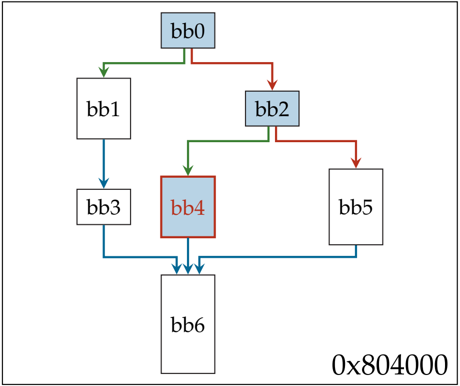
Inner Workings & Code Caches

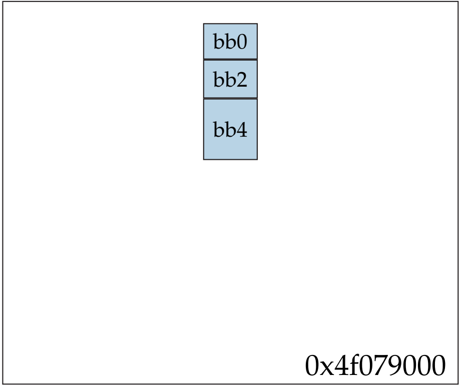




Dynamic Binary Instrumentation

Inner Workings & Code Caches

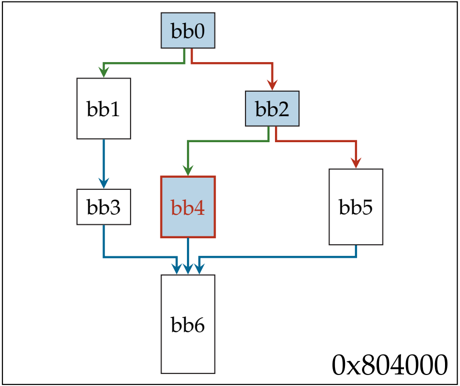


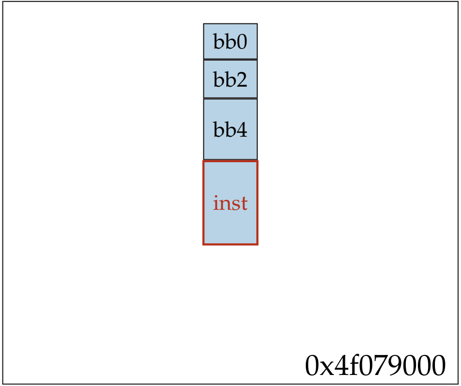


Dynamic Binary Instrumentation

Inner Workings & Code Caches

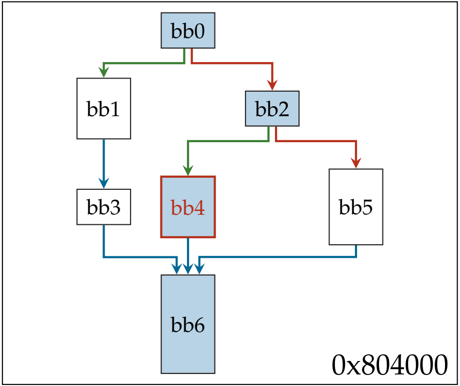
Technische Universität München

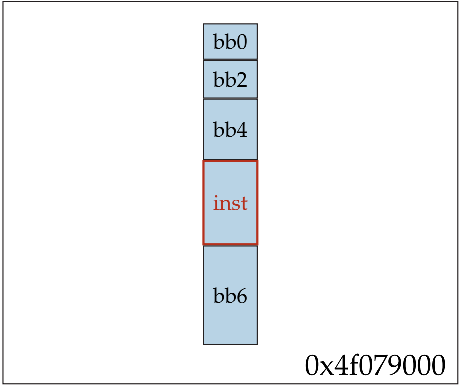




Dynamic Binary Instrumentation

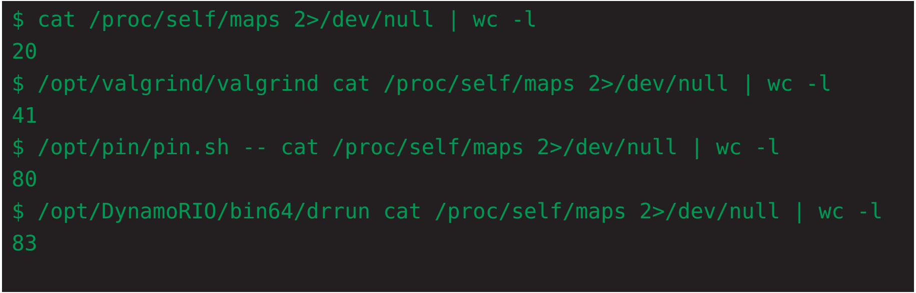
Inner Workings & Code Caches





Dynamic Binary Instrumentation

Transparency… not



```
$ cat /proc/self/maps 2>/dev/null | wc -l
20
$ /opt/valgrind/valgrind cat /proc/self/maps 2>/dev/null | wc -l
41
$ /opt/pin/pin.sh -- cat /proc/self/maps 2>/dev/null | wc -l
80
$ /opt/DynamoRIO/bin64/drrun cat /proc/self/maps 2>/dev/null | wc -l
83
```

Dynamic Binary Instrumentation

Non-Transparency

- *Dynamic Binary Instrumentation Frameworks: I know you're there spying on me*. Francisco Falcón and Nahuel Riva, REcon 2012
  - Detection using library hooks, page permissions, argv, . . .
- *Defeating the Transparency Features of Dynamic Binary Instrumentation*. Xiaoning Li and Kang Li, BlackHat USA 2014
  - Detection using file descriptor numbers, sigmasks, abnormal resource usage, . . .

Dynamic Binary Instrumentation

Non-Transparency

- *Dynamic Binary Instrumentation Frameworks: I know you're there spying on me*. Francisco Falcón and Nahuel Riva, REcon 2012
  - Detection using library hooks, page permissions, argv, ...
- *Defeating the Transparency Features of Dynamic Binary Instrumentation*. Xiaoning Li and Kang Li, BlackHat USA 2014
  - Detection using file descriptor numbers, sigmasks, abnormal resource usage, ...

→ Hiding all DBI artifacts is still a problem in 2k15!

Dynamic Binary Instrumentation

Let's try something else . . .

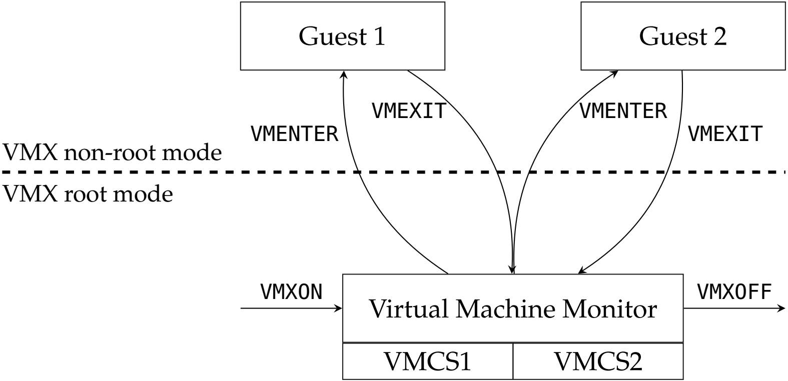
Observations:

- In the best case the instrumenting code is present only during its execution.
- Separate analysis platform from analyzed target.

→ Maybe virtualization can help . . . ?

Virtual Machine Introspection

Hardware supported virtualization



Virtual Machine Introspection

Hardware supported virtualization

- VMM maintains control of the complete guest state
  - In particular: VMM can configure the guest to hand over execution whenever a predefined condition is met.
  - Even better: VMM is able to shadow many parts of the hardware.
  - But: Reconstructing the high-level state of the guest from the available low-level information is challenging (semantic gap)

DBI using VMI

Location-based Instrumentation

Goal: Execute callback whenever an instruction at a certain address in the virtual address space of the target binary is executed.

1. Wait for the target binary being launched in the VM
2. Configure the guest to trap to the VMM if the desired location is reached
3. Inject instrumenting code
4. Profit

DBI using VMI

Location-based Instrumentation

1.1. Notify the VMM of debug exceptions:

- `GUEST_CR4.DE` = 1 (debug enable)
- `GUEST_DR7.RW0` = 0 (break on instruction execution)
- `GUEST_DR7.G0` = 1 (enable breakpoint 0)
- `VMCS.EXCEPTION_BITMAP[1] = 1` (trap DE to VMM)

DBI using VMI

Location-based Instrumentation

1.2. Notify the VMM if a new process is execve’d:

DBI using VMI

Location-based Instrumentation

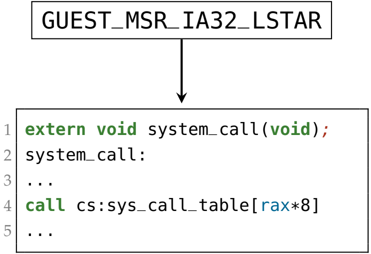
1.2. Notify the VMM if a new process is `execve`'d:

```
GUEST_MSR_IA32_LSTAR
```

DBI using VMI

Location-based Instrumentation

1.2. Notify the VMM if a new process is execve'd:



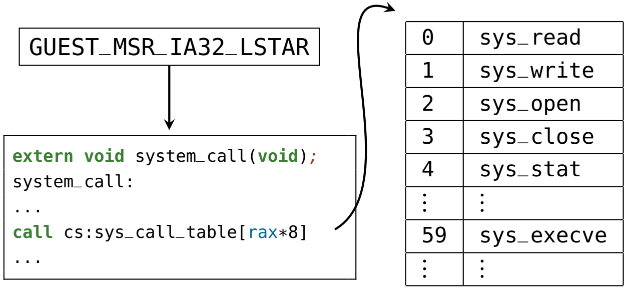
GUEST_MSR_IA32_LSTAR

```
extern void system_call(void);
system_call:
...
call cs:sys_call_table[rax*8]
...
```

DBI using VMI

Location-based Instrumentation

1.2. Notify the VMM if a new process is execve'd:



GUEST_MSR_IA32_LSTAR

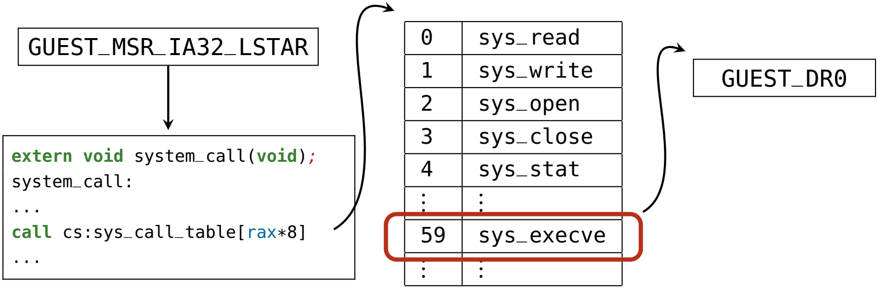
```
extern void system_call(void);
system_call:
...
call cs:sys_call_table[rax*8]
...
```

0	sys_read
1	sys_write
2	sys_open
3	sys_close
4	sys_stat
⋮	⋮
59	sys_execve
⋮	⋮

DBI using VMI

Location-based Instrumentation

1.2. Notify the VMM if a new process is execve'd:



GUEST_MSR_IA32_LSTAR

```
extern void system_call(void);
system_call:
...
call cs:sys_call_table[rax*8]
...
```

0	sys_read
1	sys_write
2	sys_open
3	sys_close
4	sys_stat
⋮	⋮
59	sys_execve
⋮	⋮

GUEST_DR0

DBI using VMI

Location-based Instrumentation

2. Identify the spawned process by its CR3 and place desired trap:
   - Check first argument of `do_execve` (`struct filename *`).
   - Resulting `CR3` identifies the instrumented process
   - `VMCS.CR3_LOAD_EXITING` = 1 (notify VMM on `mov cr3, reg`)
   - `GUEST_DR2` = instrumented location
   - `GUEST_DR7.RW1` = 0 (break on instruction execution)
   - `GUEST_DR7.G1` = 1 (enable breakpoint 1)
   - Wait for trap caused by breakpoint 1

DBI using VMI

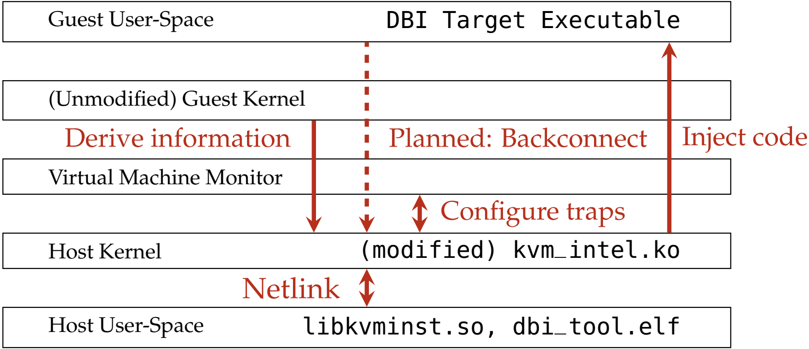
Location-based Instrumentation

3.1. Inject code into running process

- Establish a new mapping in the page table of the process
  - Currently 3 pages: code (`.txt`), data (`.bss`), stack
- Copy (position independent) instrumentation code into the guest
- Save the process execution context
- Execute instrumentation code
- Store instrumentation context (currently `.bss` only!)
- Restore execution context (on `return`/scheduling event)

DBI using VMI

Software Stack



Guest User-Space DBI Target Executable

(Unmodified) Guest Kernel

Derive information

Planned: Backconnect

Inject code

Virtual Machine Monitor

Configure traps

Host Kernel (modified) kvm_intel.ko

Netlink

Host User-Space libkvminst.so, dbi_tool.elf

DBI using VMI

SigINT CTF 2013: 0x90

- Decrypts file by $\oplus$ing with a homebrewed hash
- Hash is calculated from
  - return value of `ptrace`
  - `argv`, parts of `envp`
  - opcode bytes of all `r-x` mapped pages within the process
  - fd numbers
  - x_α via the following iteration:

$$
\begin{aligned}
x_0 &= 0 \\
x_{i+1} &= \frac{\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10}{3\left(\sqrt{\tan^{-1}(\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10)} + 10^{-28}\right)}
\end{aligned}
$$

DBI using VMI

SigINT CTF 2013: 0x90

- Decrypts file by $\oplus$ing with a homebrewed hash
- Hash is calculated from
    - return value of `ptrace`
    - `argv`, parts of `envp`
    - opcode bytes of all `r-x` mapped pages within the process
    - fd numbers
    - x_α via the following iteration:

$$
\begin{aligned}
x_0 &= 0 \\
x_{i+1} &= \frac{\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10}{3\left(\sqrt{\tan^{-1}(\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10)} + 10^{-28}\right)}
\end{aligned}
$$

    - Problem: $\alpha = 10000000000000 = 10^{13}$ = a lot

DBI using VMI

SigINT CTF 2013: 0x90

$$
\begin{aligned}
x_0 &= 0 \\
x_{i+1} &= \frac{\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10}{3\left(\sqrt{\tan^{-1}(\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10)} + 10^{-28}\right)}
\end{aligned}
$$

DBI using VMI

SigINT CTF 2013: 0x90

$$x_0 = 0$$

$$x_{i+1} = \frac{\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10}{3\left(\sqrt{\tan^{-1}(\lfloor \cos(\sqrt{x_i}) \rfloor \mod 10)} + 10^{-28}\right)}$$

- Observation:

x_0	x_1	x_2	x_3	x_4	x_5	$\cdots$
0	2.4827	0	2.4827	0	2.4827	$\cdots$

- i.e.: $x_{10000000000000} = x_0$
- Problem: How to patch this out?

DBI using VMI

SigINT CTF 2013: 0x90

```
00402D80 ; __int64 __fastcall benchmark_kernel()
00402D80 _Z16benchmark_kerneldi_W proc near       ; CODE XREF: benchmark_kernel(double, int)+A
00402D80
00402D80
00402D80 48 83 EC 18                              sub     rsp, 18h
00402D84 BF 01 00 00 00                           mov     edi, 1
00402D89 E8 F2 00 10 00                           call    srandom
00402D8E 33 D2                                    xor     edx, edx
00402D90 48 B8 00 A0 72 4E 18 09+                 mov     rax, 9184E72A000h
00402D9A C5 F9 57 C0                              vxorpd  xmm0, xmm0, xmm0
00402D9E 4C 89 64 24 08                           mov     [rsp+18h+var_10], r12
<% --- snip --- %>
```

DBI using VMI

SigINT CTF 2013: 0x90

```
00402D80 ; __int64 __fastcall benchmark_kernel()
00402D80 _Z16benchmark_kerneldi_W proc near      ; CODE XREF: benchmark_kernel(double, int)+A
00402D80
00402D80
00402D80 48 83 EC 18                             sub     rsp, 18h
00402D84 BF 01 00 00 00                          mov     edi, 1
00402D89 E8 F2 00 10 00                          call    srandom
00402D8E 33 D2                                   xor     edx, edx
00402D90 48 B8 00 A0 72 4E 18 09+                mov     rax, 9184E72A000h
00402D9A C5 F9 57 C0                             vxorpd  xmm0, xmm0, xmm0
00402D9E 4C 89 64 24 08                          mov     [rsp+18h+var_10], r12
<% --- snip --- %>
```

DBI using VMI

SigINT CTF 2013: 0x90

```c
#include "my_regpatch_cb.h"

/* Callback to adjust the rax register */
int my_regpatch_cb(struct kvm_inst_ctxt *ctxt)
{
    /* Sanity check */
    if (ctxt->rax == 0x9184E72A000ULL) {
        ctxt->rax = 0x2;
        return 0;
    } else {
        return -1;
    }
}
```

my_regpatch_cb.c

```c
#ifndef __MY_REGPATCH_CB_H
#define __MY_REGPATCH_CB_H

#include "kvm-inst.h"

int my_regpatch_cb(
    struct kvm_inst_ctxt *);

#endif
```

my_regpatch_cb.h

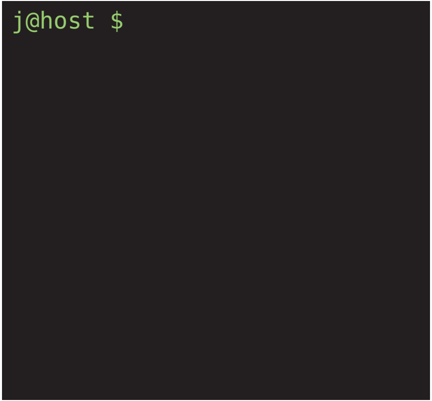
DBI using VMI

SigINT CTF 2013: 0x90

```c
#include <stdio.h>
#include "kvm-inst.h"
#include "my_regpatch_cb.h"

int main(int argc, char **argv)
{
    puts("[+] Setting instrumentation target ...");
    if (kvm_inst_target("/home/vm/0x90/0x90.run")) {
        puts("[-] Could not set target process, is KVM
               running?");
        return -1;
    }
    if (kvm_inst_reg_loc_cb(0x00402D9A, my_regpatch_cb)
           ) {
        puts("[-] Could not register location
               instrumentation callback.");
        return -2;
    }
    switch (kvm_inst_eventloop()) {
        case KVM_INST_TARGET_START:
            puts("[+] The target program was started in
                    the VM.");
        break;
        case KVM_INST_TARGET_EXIT:
            puts("[+] The target program exited.");
        break;
        case KVM_INST_LOC_CB_RET:
            puts("[+] Callback function returned %d.",
                kvm_inst_get_last_retval());
        break;
        default:
            puts("[-] Fatal: Unhandled instrumentation
                   event.");
            return -3;
    }
}
```

DBI using VMI

SigINT CTF 2013: 0x90

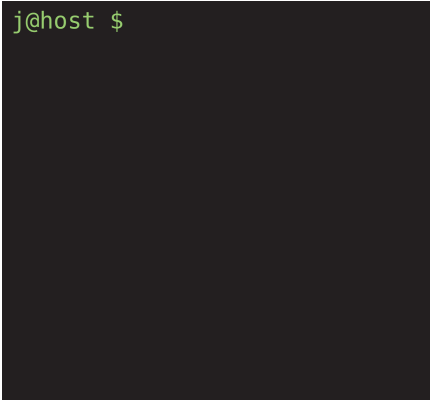


```
j@host $
```

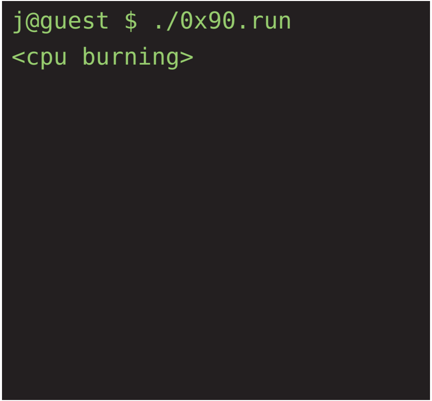
```
j@guest $
```

DBI using VMI

SigINT CTF 2013: 0x90



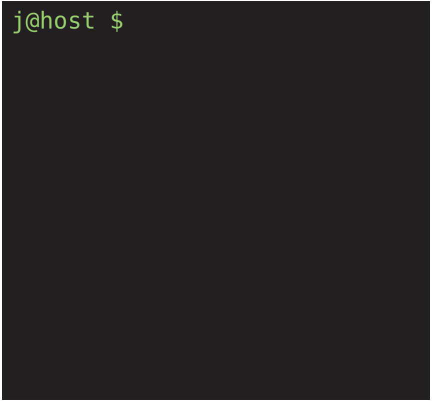
```
j@host $
```



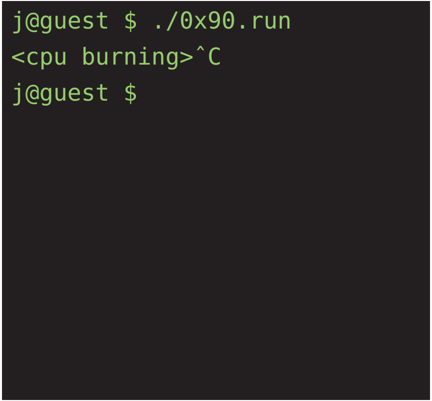
```
j@guest $ ./0x90.run
<cpu burning>
```

DBI using VMI

SigINT CTF 2013: 0x90



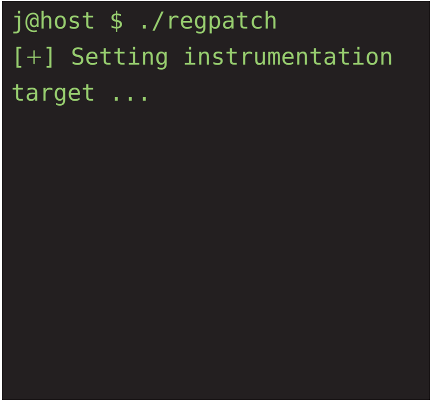
```
j@host $
```



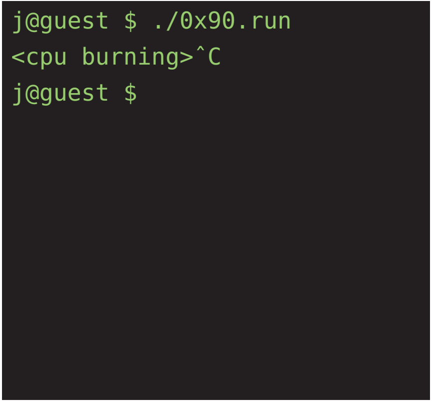
```
j@guest $ ./0x90.run
<cpu burning>^C
j@guest $
```

DBI using VMI

SigINT CTF 2013: 0x90



```
j@host $ ./regpatch
[+] Setting instrumentation
target ...
```



```
j@guest $ ./0x90.run
<cpu burning>^C
j@guest $
```

DBI using VMI

SigINT CTF 2013: 0x90

```
j@host $ ./regpatch
[+] Setting instrumentation
target ...
[+] The target program was
started in the VM.
```

```
j@guest $ ./0x90.run
<cpu burning>^C
j@guest $ ./0x90.run
```

DBI using VMI

SigINT CTF 2013: 0x90

```
j@host $ ./regpatch
[+] Setting instrumentation
target ...
[+] The target program was
started in the VM.
[+] Callback function
returned 0.
[+] The target program
exited.
j@host $
```

```
j@guest $ ./0x90.run
<cpu burning>^C
j@guest $ ./0x90.run
sigint_mcHamm3R
j@guest $
```

DBI using VMI

Event-based Instrumentation

- Execute code based on events, not a particular executed instruction
- Examples: Instrument on
  - taken branches
  - function calls/returns
  - ...
- Challenge: Catch these events in the guest and deliver them to the VMM

DBI using VMI

Event-based Instrumentation

Idea: (Ab)use CPU's built-in hardware performance counters (PMCs)

- Counted PMC events are highly configurable

DBI using VMI

Event-based Instrumentation

Idea: (Ab)use CPU's built-in hardware performance counters (PMCs)

- Counted PMC events are highly configurable (i.e. see pages 2584 – 2780 of the Intel Software Developer's Manual)

DBI using VMI

Event-based Instrumentation

Idea: (Ab)use CPU's built-in hardware performance counters (PMCs)

- Counted PMC events are highly configurable (i.e. see pages 2584 – 2780 of the Intel Software Developer's Manual)
- Problem: No Mechanism to trap an PMC increment to the VMM

DBI using VMI

Event-based Instrumentation

Idea: (Ab)use CPU's built-in hardware performance counters (PMCs)

- Counted PMC events are highly configurable (i.e. see pages 2584 – 2780 of the Intel Software Developer's Manual)
- Problem: No Mechanism to trap an PMC increment to the VMM
- But: Local APIC can be configured to generate a debug exception on PMC overflows which can be trapped!

DBI using VMI

Event-based Instrumentation

Goal: Execute callback whenever a certain event during the execution of the target binary occurs.

1. Select desired entry from the list of countable events (e.g. `0xc4`, `0x04` for `BR_INST_RETIRED.ALL_BRANCHES`), use these values for `GUEST_MSR_IA32_PERFEVTSEL0.EVENT` and `.UMASK` respectively.
2. `GUEST_MSR_IA32_PERFEVTSEL.USR` = 1, `.INT` = 1, `.EN` = 1
3. `GUEST_MSR_CORE_PERF_GLOBAL_CTRL[0]` = 1 (enable counter 0)

DBI using VMI

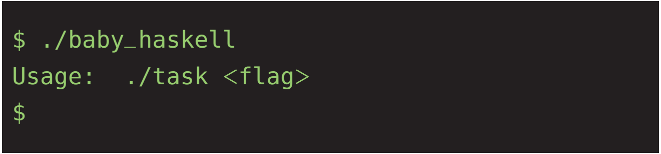
Event-based Instrumentation

Goal: Execute callback whenever a certain event during the execution of the target binary occurs.

1. Select desired entry from the list of countable events (e.g. `0xc4`, `0x04` for `BR_INST_RETIRED.ALL_BRANCHES`), use these values for `GUEST_MSR_IA32_PERFEVTSEL0.EVENT` and `.UMASK` respectively.
2. `GUEST_MSR_IA32_PERFEVTSEL.USR` = 1, `.INT` = 1, `.EN` = 1
3. `GUEST_MSR_CORE_PERF_GLOBAL_CTRL[0]` = 1 (enable counter 0)
4. `GUEST_MSR_IA32_PMC0 = 0xffffffffffffffff` (-1 ☺)

DBI using VMI

Insomni’hack Teaser 2015: baby_haskell

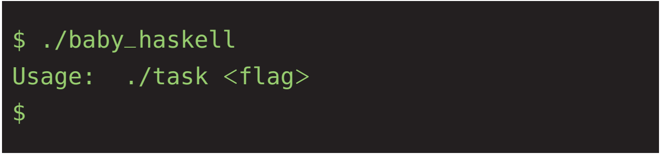


```
$ ./baby_haskell
Usage:  ./task <flag>
$
```

- Supply the flag as an argument
- Program verifies input
- Problem: The binary consists of compiled Haskell code

DBI using VMI

Insomni'hack Teaser 2015: baby_haskell



```
$ ./baby_haskell
Usage:  ./task <flag>
$
```

- Supply the flag as an argument
- Program verifies input
- Problem: The binary consists of compiled Haskell code
- Maybe comparison is done character wise?

DBI using VMI

Insomni’hack Teaser 2015: baby_haskell

```c
#include "hs_pwn.h"

unsigned int branches, max_branches;
char curr_guess = 0x20, best_guess = 0;
unsigned int curr_pos, init_pos;
char *hs_argv0;
int init_cb(struct kvm_inst_ctxt *ctxt) {
    if (!init_once) {
        hs_argv0 = *ctxt->rdi;
        curr_pos = strlen(hs_argv0);
    }
    hs_argv0[curr_pos] = curr_guess;
    return 0;
}
```

```c
int restart_cb(struct kvm_inst_ctxt *ctxt) {
    if (curr_guess < 0x7f) {
        if (branches > max_branches)
            best_guess = curr_guess;
        curr_guess++;
        ctxt->rip = 0x46e130;
    } else {
        printf("%c\n", best_guess);
    }
    return 0;
}
int branch_cb(struct kvm_inst_ctxt *ctxt) {
    branches++; return 0;
}
```

DBI using VMI

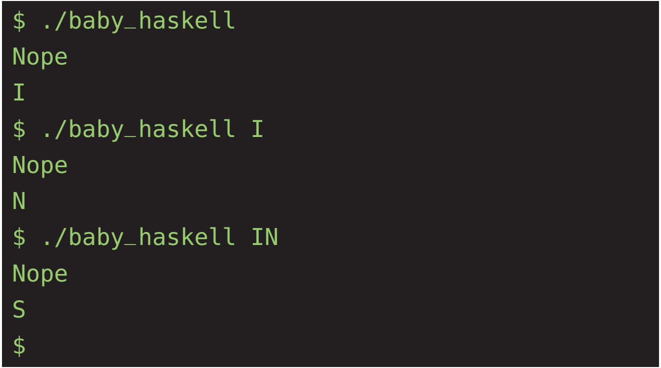
Insomni'hack Teaser 2015: baby_haskell

```c
#include <stdio.h>
#include "kvm-inst.h"
#include "hs_pwn.h"

int main(int argc, char **argv)
{
    puts("[+] Setting instrumentation target ...");
    if (kvm_inst_target("/home/vm/baby_haskell/baby_haskell")) {
        puts("[-] Could not set target process, is KVM running?")
               ;
        return -1;
    }
    if (kvm_inst_reg_loc_cb(0x0046e130, init_cb)) {
        puts("[-] Could not register init callback.");
        return -2;
    }
    if (kvm_inst_reg_loc_cb(0x0046e1d7, restart_cb)) {
        puts("[-] Could not register restart callback.");
        return -3;
    }
```

```c
    if (kvm_inst_reg_pmc_cb(0xc4, 0x04,
            branch_cb)) {
        puts("[-] Could not register branch
               callback.");
        return -4;
    }

    switch (kvm_inst_eventloop()) {
        case KVM_INST_TARGET_START:
        case KVM_INST_TARGET_EXIT:
        case KVM_INST_LOC_CB_RET:
        case KVM_INST_PMC_CB_RET:
            puts("KVM_INST_EVENT");
        break;
        default:
            puts("[-] Fatal: Unhandled
                   instrumentation event.");
            return -3;
    }
}
```

DBI using VMI

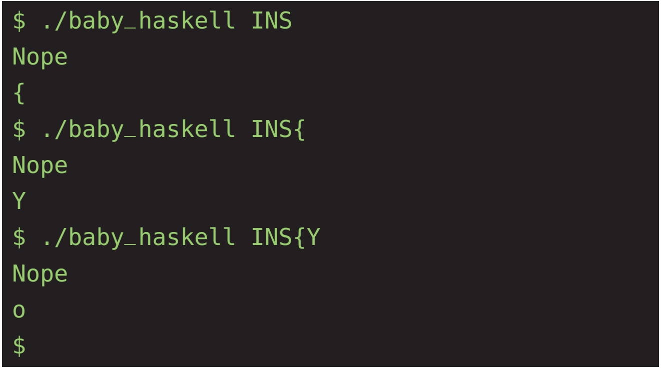
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell
Nope
I
$ ./baby_haskell I
Nope
N
$ ./baby_haskell IN
Nope
S
$
```

DBI using VMI

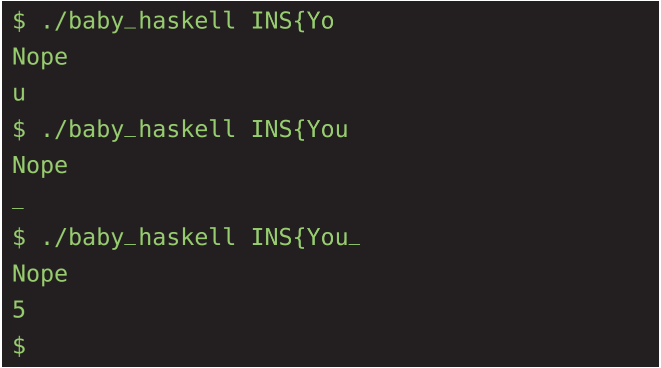
Insomni'hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS
Nope
{
$ ./baby_haskell INS{
Nope
Y
$ ./baby_haskell INS{Y
Nope
o
$
```

DBI using VMI

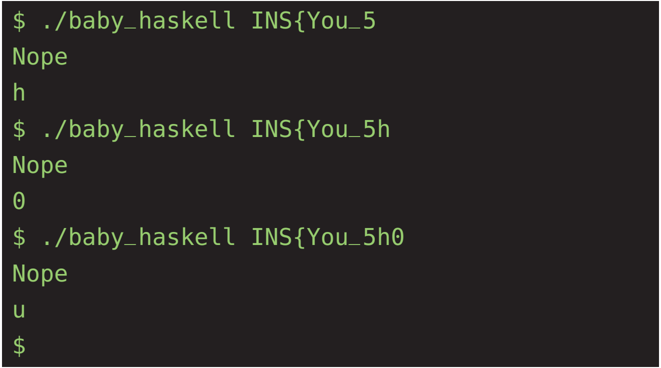
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{Yo
Nope
u
$ ./baby_haskell INS{You
Nope
_
$ ./baby_haskell INS{You_
Nope
5
$
```

DBI using VMI

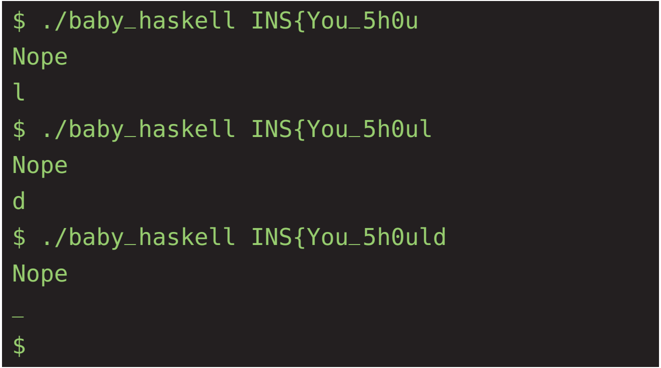
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5
Nope
h
$ ./baby_haskell INS{You_5h
Nope
0
$ ./baby_haskell INS{You_5h0
Nope
u
$
```

DBI using VMI

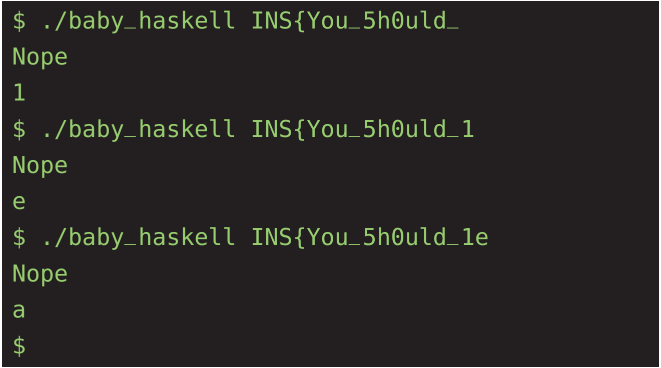
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0u
Nope
l
$ ./baby_haskell INS{You_5h0ul
Nope
d
$ ./baby_haskell INS{You_5h0uld
Nope
_
$
```

DBI using VMI

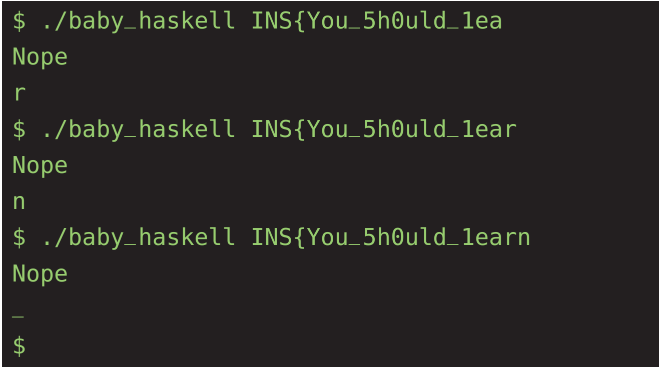
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0uld_
Nope
1
$ ./baby_haskell INS{You_5h0uld_1
Nope
e
$ ./baby_haskell INS{You_5h0uld_1e
Nope
a
$
```

DBI using VMI

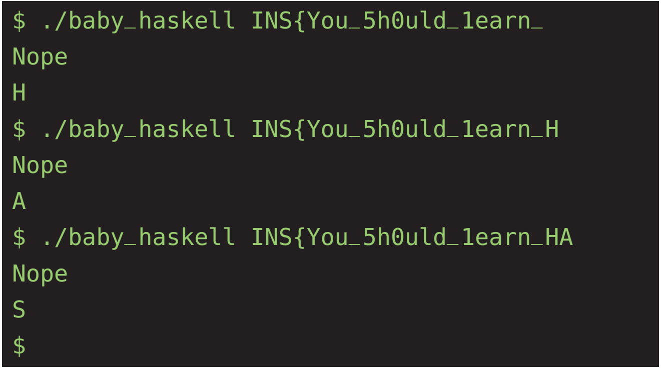
Insomni'hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0uld_1ea
Nope
r
$ ./baby_haskell INS{You_5h0uld_1ear
Nope
n
$ ./baby_haskell INS{You_5h0uld_1earn
Nope
_
$
```

DBI using VMI

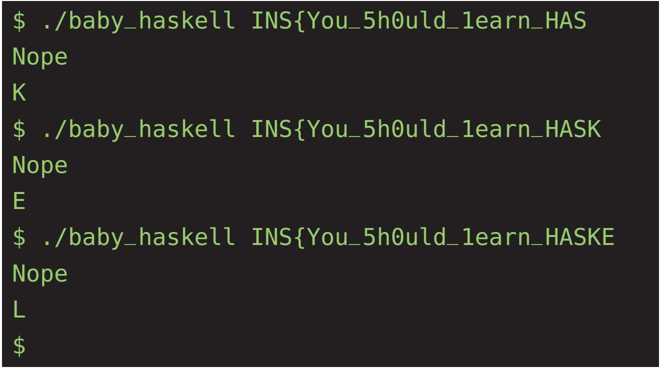
Insomni'hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0uld_1earn_
Nope
H
$ ./baby_haskell INS{You_5h0uld_1earn_H
Nope
A
$ ./baby_haskell INS{You_5h0uld_1earn_HA
Nope
S
$
```

DBI using VMI

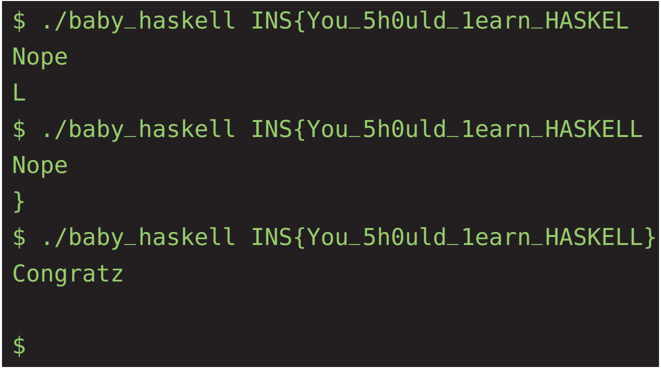
Insomni’hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0uld_1earn_HAS
Nope
K
$ ./baby_haskell INS{You_5h0uld_1earn_HASK
Nope
E
$ ./baby_haskell INS{You_5h0uld_1earn_HASKE
Nope
L
$
```

DBI using VMI

Insomni'hack Teaser 2015: baby_haskell



```
$ ./baby_haskell INS{You_5h0uld_1earn_HASKEL
Nope
L
$ ./baby_haskell INS{You_5h0uld_1earn_HASKELL
Nope
}
$ ./baby_haskell INS{You_5h0uld_1earn_HASKELL}
Congratz

$
```

Limitations

- Timing
- Concurrency
- Trust the OS kernel
- # of Hardware Breakpoints & Performance Counters
- Resource Conflicts

Questions?

Thanks!

```
git://kirschju.re/kvm-inst.git
```