PEMU: A PIN Highly Compatible Out-of-VM Dynamic Binary Instrumentation Framework

Junyuan Zeng, Yangchun Fu, Zhiqiang Lin

Department of Computer Science
The University of Texas at Dallas

March 15\textsuperscript{th}, 2015
Dynamic Binary Instrumentation (DBI)

- An extremely powerful technique for **program analysis**.
- Dynamically inserts **extra analysis code** into the running binary program to observe how it behaves.
Dynamic Binary Instrumentation (DBI)

- An extremely powerful technique for program analysis.
- Dynamically inserts extra analysis code into the running binary program to observe how it behaves.

Applications

1. Performance profiling
2. Architecture simulation
3. Program shepherding
4. Program optimization
5. Dynamic taint analysis
6. Reverse engineering
7. Malware analysis
8. ...
In-VM DBI

Guest Ring 3

Original

test %eax,%eax
je 0x8052913 ...

Guest Ring 0

Pin

print(pc)
test %eax,%eax
print(pc)
je 0x8052913 ...

syscall

In-VM Instrumentation
In-VM DBI vs. Out-of-VM DBI

Guest Ring 3

test %eax,%eax
je 0x8052913...

Guest Ring 0

print(pc)
test %eax,%eax
print(pc)
jmp 0xae1a8c0b

Host Ring 3

movl $0x8052ca,0x20(%ebp)
movl $0xa9bbcc8d,%eax
jmp 0xb5f3cc15

Host Ring 0

syscall

print(pc)
movl $0x805213,0x20(%ebp)
movl $0xa9bbcc8c,%eax
# State-of-the-Art

<table>
<thead>
<tr>
<th>Platforms</th>
<th>Year</th>
<th>Emulator, Simulator, Virtualizer</th>
<th>Kernel Level Instrumentation</th>
<th>User Level Instrumentation w/ API for instrumentation</th>
<th>Out-of-VM</th>
<th>Guest OS Agnostic</th>
<th>PIN API Compatible</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMBRA [WR96]</td>
<td>1996</td>
<td>✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMWARE [DBR98]</td>
<td>1998</td>
<td>✓ ✓ ✓</td>
<td>X</td>
<td>X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KERNINST [TM99]</td>
<td>1999</td>
<td>X ✓ ✓</td>
<td>✓</td>
<td>X</td>
<td>✓ ✓ ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNINSTAPI [BH00]</td>
<td>2000</td>
<td>X ✓ ✓</td>
<td>X</td>
<td>X</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMO [BDB00]</td>
<td>2000</td>
<td>X ✓ ✓</td>
<td>X</td>
<td>X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOCHS [boc01]</td>
<td>2001</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMICS [MCE+02]</td>
<td>2002</td>
<td>✓ ✓ ✓</td>
<td>X</td>
<td>X</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALGRIND [NS03, NS07]</td>
<td>2003</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRATA [SKV+03]</td>
<td>2003</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMORIO [BDB00, BZA12]</td>
<td>2004</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU [Bel05]</td>
<td>2005</td>
<td>✓ ✓ ✓</td>
<td>X</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN [LCM+05]</td>
<td>2005</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIRVANA [BCdJ+06]</td>
<td>2006</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDTRANS [SSN06]</td>
<td>2006</td>
<td>X X ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRTUALBox [Wat08]</td>
<td>2007</td>
<td>✓ ✓ ✓</td>
<td>X</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PINOS [BL07]</td>
<td>2007</td>
<td>✓ ✓ ✓</td>
<td>X</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMU [YS]</td>
<td>2010</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNINST [MB11]</td>
<td>2010</td>
<td>X ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRK [PFG12]</td>
<td>2013</td>
<td>X ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECAF [HPY+14]</td>
<td>2014</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>X X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEMU</td>
<td>2015</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
<td>✓</td>
<td>✓ X ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Limitations Among Prior Works

**Process Level DBI (e.g., PIN, VALGRIND)**
- Process-level DBI such as PIN and VALGRIND provides rich APIs to analyze user level binary code execution, but the analysis code is executed inside the VM (i.e., in-VM) with the same privilege as the instrumented process.
- No kernel level instrumentation
- Limited type of OS (VALGRIND only for Linux)

**VM Monitor Level DBI (e.g., QEMU)**
- No general DBI APIs
## State-of-the-Art

<table>
<thead>
<tr>
<th>Platforms</th>
<th>Year</th>
<th>Emulator, Simulator, Virtualizer</th>
<th>Kernel Level Instrumentation w/ API for instrumentation</th>
<th>User Level Instrumentation w/ API for instrumentation</th>
<th>Out-of-VM</th>
<th>Guest OS Agnostic</th>
<th>PIN API Compatible</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMBRA [WR96]</td>
<td>1996</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMWARE [DBR98]</td>
<td>1998</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KERNEL [TM99]</td>
<td>1999</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNINSTAPI [BH00]</td>
<td>2000</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMO [BDB00]</td>
<td>2000</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOCHS [boc01]</td>
<td>2001</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMICS [MCE+02]</td>
<td>2002</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALGRIND [NS03, NS07]</td>
<td>2003</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRATA [SKV+03]</td>
<td>2003</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMO [BDB00, BZA12]</td>
<td>2004</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU [Bel05]</td>
<td>2005</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PIN [LCM+06]</td>
<td>2005</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>NIRVANA [BCdJ+06]</td>
<td>2006</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDTRANS [SSNB06]</td>
<td>2006</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRTUALBox [Wat08]</td>
<td>2007</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PinOS [BL07]</td>
<td>2007</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TEMU [YS]</td>
<td>2010</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DYNINST [MB11]</td>
<td>2010</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DRK [PFG12]</td>
<td>2013</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DECAF [HPY+14]</td>
<td>2014</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>PEMU 2015</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Introduction of PEMU

PIN + QEMU = PEMU

Objectives

- Rich APIs
  - Provide rich and well-defined APIs for DBI
  - PIN-API compatibility
- Cross-OS
  - OS agnostic for the introspection
- Strong Isolation
  - Out-of-VM Instrumentation to isolate analysis routings with target programs (QEMU)
- VM Introspection
  - Support high level guest object introspection
A sample PIN plugin

```c
1  static UINT64 icount;
2  FILE *pFile;
3  VOID docount(UINT32 c) { icount += c; }
4  VOID Trace(TRACE trace, VOID *v) {
5    for (BBL bbl = TRACE_BblHead(trace);
6        BBL_Valid(bbl); bbl = BBL_Next(bbl)) {
7      BBL_InsertCall(bbl, IPOINT_BEFORE,
8                     (AFUNPTR)docount, IARG_UINT32, BBL_NumIns(bbl),
9                     IARG_END);
10   }
11 }
12  VOID Fini(INT32 code, VOID *v) {
13    fprintf(pFile, "Count %lld\n", icount);
14    fclose(pFile);
15  }
16  INT32 Usage(VOID) {
17    return 0;
18  }
19  int main(int argc, char * argv[]) {
20    if(PIN_Init(argc, argv)) return Usage();
21    pFile = fopen("pemu_count", "w");
22    TRACE_AddInstrumentFunction(Trace, 0);
23    PIN_AddFiniFunction(Fini, 0);
24    PIN_StartProgram();
```
Challenges

1. **How to implement PIN API based on QEMU**
   - QEMU does not have abstraction for TRACE, SEC, Function and IMG etc.
   - QEMU basic block has limited size

2. **Semantic gap between guest OS and host OS**
   - How to find the target process or thread
   - PIN plugins may inspect guest OS state, such as getpid
Challenges

1. **How to implement PIN API based on QEMU**
   - QEMU does not have abstraction for TRACE, SEC, Function and IMG etc.
   - QEMU basic block has limited size
   ⇒ **Instrumentation Engine**

2. **Semantic gap between guest OS and host OS**
   - How to find the target process or thread
   - PIN plugins may inspect guest OS state, such as `getpid`
   ⇒ **Introspection Engine**
Instrumentation Engine

1. **Trace Instrumentation**: occurs immediately before a code sequence is executed
   - Instruction Level, e.g., \texttt{INS\_InsertCall(ins, IPOINT\_BEFORE, ...)}
   - Basic Block Level, e.g., \texttt{BBL\_InsertCall(bbl, IPOINT\_BEFORE, ...)}
   - Trace Level, e.g., \texttt{TRACE\_InsertCall(trace, IPOINT\_BEFORE, ...)}

2. **Ahead-of-time Instrumentation**: caches the instrumentation before the execution.
   - IMG instrumentation, e.g., \texttt{IMG\_AddInstrumentFunction}
   - RTN instrumentation, e.g., \texttt{RTN\_InsertCall}
Instrumentation Engine

1. **Trace Instrumentation**: occurs immediately before a code sequence is executed
   - Instruction Level, e.g., `INS_InsertCall(ins, IPOINT_BEFORE, ..)`
   - Basic Block Level, e.g., `BBL_InsertCall(bbl, IPOINT_BEFORE, ..)`
   - Trace Level, e.g., `TRACE_InsertCall(trace, IPOINT_BEFORE, ..)`

   ⇒ Adding our own disassembler (TRACE Constructor)

2. **Ahead-of-time Instrumentation**: caches the instrumentation before the execution.
   - IMG instrumentation, e.g.,
     `IMG_AddInstrumentFunction`
   - RTN instrumentation, e.g., `RTN_InsertCall`

   ⇒ Instrumenting code when image is loaded
Instrumentation Engine

流程图说明：
1. Guest CPU 给 Instrumentation Engine 发送信号。
2. Code Injector 处理信号。
3. 调用 TRACE Constructor。
4. Guest Code 进入 Host Code。
5. Meta Data 处理。
6. Instrumentation Engine 向 Guest CPU 回应。
7. Host CPU 处理回应信号。
Instrumentation Engine

Overview

PEMU Design

Evaluations

Discussion

Conclusion

References

Instrumentation Engine

Meta Data

Instrumentation API

QEMU-DBT

Code Injector

TRACE Constructor

Guest CPU

push %ebp;
mov %esp,%ebp
push %ebx

Guest Code

Host CPU

push %ebp;
mov %esp,%ebp
push %ebx

Host Code

Host CPU

TRACE

TRACE_Address

TRACE_Size

TRACE_NumBbl

TRACE_BblHead*

PCi, PCj, PCk, …

NULL

CALL

Instr_func

PCi

PCj

PCk

Type_i, Func_i, List_of_args

Type_k, Func_k, List_of_args

Hooking Point

Hash-table (HPHT)

TPC

Instances of TRACE, BBL and INS

TRACE

INC

INS_Address

INS_Size

INS_NumIns

INS_Next*

INS

INS_Address

INS_Size

INS_NumIns

INS_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*

PC

PC_Address

PC_Size

PC_NumIns

PC_Next*

NULL

BBL

BBL_Address

BBL_Size

BBL_NumIns

BBL_Next*

BBL_InsHead*
Introspection Engine

Goals and Solutions

- Identification of the monitoring process/threads
  - \( \Rightarrow \) Capture the data life time of PGD (\( \text{CR3 in x86} \)) to identify the new process
  - \( \Rightarrow \) Kernel \( \text{esp} \) to identify new threads

- Bridging the semantic gap for the out-of-VM plugins
  - \( \Rightarrow \) Using a system call redirection/forwarding approach
  - HyperShell [YFL14]
Introspection Engine: Syscall Redirection

User

Plugin:

PEMU_glibc:

Open(...)

Syscall()

{
  set_up_syscall_arg()
  PEMU_syscall()
}

Kernel
## Compatibility Testing With PIN Plugins

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Description</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>calltrace.so</td>
<td>Call trace tracing</td>
<td>✓</td>
</tr>
<tr>
<td>extmix.so</td>
<td>Instruction extension mix profile</td>
<td>✓</td>
</tr>
<tr>
<td>inscount2_vregs.so</td>
<td>Counting executing instructions</td>
<td>✓</td>
</tr>
<tr>
<td>pinatrace.so</td>
<td>Memory address tracing</td>
<td>✓</td>
</tr>
<tr>
<td>xed-cache.so</td>
<td>Decode cache profile</td>
<td>✓</td>
</tr>
<tr>
<td>catmix.so</td>
<td>Instruction category mix profile</td>
<td>✓</td>
</tr>
<tr>
<td>fence.so</td>
<td>Runtime text modification guard</td>
<td>✓</td>
</tr>
<tr>
<td>jumpmix.so</td>
<td>Jmp/branch/call profiling</td>
<td>✓</td>
</tr>
<tr>
<td>regmix.so</td>
<td>Register usage mix profile</td>
<td>✓</td>
</tr>
<tr>
<td>xed-print.so</td>
<td>XED usage testing</td>
<td>✓</td>
</tr>
<tr>
<td>coco.so</td>
<td>Code coverage analyzer</td>
<td>✓</td>
</tr>
<tr>
<td>icount.so</td>
<td>Counting executing instructions</td>
<td>✓</td>
</tr>
<tr>
<td>ldstmix.so</td>
<td>Register/memory operand profiler</td>
<td>✓</td>
</tr>
<tr>
<td>topopcode.so</td>
<td>Opcode mix profiler</td>
<td>✓</td>
</tr>
<tr>
<td>xed-use.so</td>
<td>XED interface usage testing</td>
<td>✓</td>
</tr>
<tr>
<td>dcache.so</td>
<td>Data cache simulation</td>
<td>X</td>
</tr>
<tr>
<td>ilenmix.so</td>
<td>Instruction length mix profiler</td>
<td>✓</td>
</tr>
<tr>
<td>malloctrace.so</td>
<td>Tracing calls to malloc</td>
<td>✓</td>
</tr>
<tr>
<td>toprtn.so</td>
<td>Hostest routines profiling</td>
<td>✓</td>
</tr>
<tr>
<td>edgcnt.so</td>
<td>Control flow edge profiler</td>
<td>✓</td>
</tr>
<tr>
<td>inscount2_mt.so</td>
<td>Counting executing instructions</td>
<td>✓</td>
</tr>
<tr>
<td>opcodemix.so</td>
<td>Opcode mix profiler</td>
<td>X</td>
</tr>
<tr>
<td>trace.so</td>
<td>Compressed instruction tracer</td>
<td>✓</td>
</tr>
</tbody>
</table>
### Performance Evaluation: Speed

<table>
<thead>
<tr>
<th>Program</th>
<th>#Inst (M)</th>
<th>$T_{Qemu}$ (s)</th>
<th>$T_{Pemu}$ (s)</th>
<th>$T_{Pemu} / T_{Qemu}$</th>
<th>$T_{Pin}$ (s)</th>
<th>$T_{Qemu} / T_{Pin}$</th>
<th>$T_{Pemu} / T_{Pin}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>401.bzip2</td>
<td>11500.27</td>
<td>24.55</td>
<td>81.15</td>
<td>3.31</td>
<td>11.17</td>
<td>2.20</td>
<td>7.26</td>
</tr>
<tr>
<td>403.gcc</td>
<td>4940.36</td>
<td>18.35</td>
<td>169.21</td>
<td>9.22</td>
<td>13.56</td>
<td>1.35</td>
<td>12.48</td>
</tr>
<tr>
<td>410.bwaves</td>
<td>29360.09</td>
<td>419.99</td>
<td>1336.57</td>
<td>3.18</td>
<td>7.44</td>
<td>56.45</td>
<td>179.65</td>
</tr>
<tr>
<td>416.gamess</td>
<td>2121.15</td>
<td>23.19</td>
<td>84.99</td>
<td>3.66</td>
<td>3.35</td>
<td>6.92</td>
<td>25.37</td>
</tr>
<tr>
<td>429.mcf</td>
<td>3562.67</td>
<td>23.91</td>
<td>70.58</td>
<td>2.95</td>
<td>3.55</td>
<td>6.74</td>
<td>19.88</td>
</tr>
<tr>
<td>433.milc</td>
<td>39509.49</td>
<td>779.07</td>
<td>2570.44</td>
<td>3.30</td>
<td>9.28</td>
<td>83.95</td>
<td>276.99</td>
</tr>
<tr>
<td>435.gromacs</td>
<td>4907.53</td>
<td>106.28</td>
<td>334.74</td>
<td>3.15</td>
<td>3.43</td>
<td>30.99</td>
<td>97.59</td>
</tr>
<tr>
<td>436.cactusADM</td>
<td>9730.11</td>
<td>304.89</td>
<td>1019.89</td>
<td>3.35</td>
<td>4.42</td>
<td>68.98</td>
<td>230.74</td>
</tr>
<tr>
<td>437.leslie3d</td>
<td>55857.54</td>
<td>900.01</td>
<td>3009.06</td>
<td>3.34</td>
<td>15.38</td>
<td>58.52</td>
<td>195.65</td>
</tr>
<tr>
<td>444.namd</td>
<td>74037.63</td>
<td>1523.78</td>
<td>5037.00</td>
<td>3.31</td>
<td>16.22</td>
<td>93.94</td>
<td>310.54</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>314.88</td>
<td>2.43</td>
<td>4.17</td>
<td>1.72</td>
<td>1.71</td>
<td>1.42</td>
<td>2.44</td>
</tr>
<tr>
<td>450.soplex</td>
<td>63.67</td>
<td>1.49</td>
<td>2.22</td>
<td>1.49</td>
<td>1.80</td>
<td>0.83</td>
<td>1.23</td>
</tr>
<tr>
<td>453.povray</td>
<td>2987.41</td>
<td>36.16</td>
<td>193.17</td>
<td>5.34</td>
<td>3.52</td>
<td>10.27</td>
<td>54.88</td>
</tr>
<tr>
<td>454.calculix</td>
<td>187.33</td>
<td>2.53</td>
<td>6.61</td>
<td>2.61</td>
<td>2.46</td>
<td>1.03</td>
<td>2.69</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>17862.2</td>
<td>46.43</td>
<td>260.56</td>
<td>5.61</td>
<td>6.95</td>
<td>6.68</td>
<td>37.49</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>15514.49</td>
<td>48.40</td>
<td>432.79</td>
<td>8.94</td>
<td>14.38</td>
<td>3.37</td>
<td>30.10</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>408.63</td>
<td>0.77</td>
<td>2.01</td>
<td>2.61</td>
<td>0.62</td>
<td>1.24</td>
<td>3.24</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>98144.32</td>
<td>392.21</td>
<td>2751.31</td>
<td>7.01</td>
<td>34.01</td>
<td>11.53</td>
<td>80.90</td>
</tr>
<tr>
<td>465.tonto</td>
<td>3571.85</td>
<td>48.23</td>
<td>195.16</td>
<td>4.05</td>
<td>5.44</td>
<td>8.87</td>
<td>35.88</td>
</tr>
<tr>
<td>470.lbm</td>
<td>7744.81</td>
<td>161.22</td>
<td>692.51</td>
<td>4.30</td>
<td>2.92</td>
<td>55.21</td>
<td>237.16</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>2209.23</td>
<td>16.24</td>
<td>136.63</td>
<td>8.41</td>
<td>3.19</td>
<td>5.09</td>
<td>42.83</td>
</tr>
<tr>
<td>473.astar</td>
<td>26645.10</td>
<td>102.95</td>
<td>734.52</td>
<td>7.13</td>
<td>13.67</td>
<td>7.53</td>
<td>53.73</td>
</tr>
<tr>
<td>482.sphinx3</td>
<td>6198.21</td>
<td>77.95</td>
<td>322.26</td>
<td>4.13</td>
<td>4.94</td>
<td>15.78</td>
<td>65.23</td>
</tr>
<tr>
<td>999.specrand</td>
<td>6198.21</td>
<td>1.42</td>
<td>2.46</td>
<td>1.73</td>
<td>0.92</td>
<td>1.54</td>
<td>2.67</td>
</tr>
<tr>
<td>Avg.</td>
<td>17649.05</td>
<td>210.94</td>
<td>810.42</td>
<td>4.33</td>
<td>7.68</td>
<td>22.52</td>
<td>83.61</td>
</tr>
</tbody>
</table>
We compiled it into a plugin architecture, and each technique is implemented like a tool to test anti-instrumentation techniques. We then use PEMU and PEMU plugin with the loaded DLLs. We found that 17 anti-instrumentation techniques detect the presence of PEMU. We ran successfully on PEMU and PEMU plugin with the same source code. PEMU has open.

We show there is a need for out-of-VM alternatives. Also, even though future malware may escape PEVerifier, we should be able to add countermeasures against them, given that the source code of PEMU is open.
Applications: anti-PIN malware analysis

Background
- Anti-PIN malware: malware exits when it detects it is run inside PIN

Case Studies
- **tElock, Safengine Shielden**: two widely used tools to build anti-analysis software.
- **eXait**: a benchmark-like tool to test anti-instrumentation techniques
Applications: anti-PIN malware analysis

```c
1   FILE *trace;
2   VOID SysBefore(ADDRINT ip, ADDRINT num) {
3       fprintf(trace,"0x%x: %ld\n",
4               (unsigned long)ip, (long)num);
5   }
6   VOID SyscallEntry(THREADID threadIndex,
7            CONTEXT *ctxt, SYSCALL_STANDARD std, VOID *v) {
8       SysBefore(PIN_GetContextReg(ctxt, REG_INST_PTR),
9       PIN_GetSyscallNumber(ctxt, std));
10   }
11   VOID Fini(INT32 code, VOID *v) {
12       printf("program exit()\n");
13   }
14   INT32 Usage(VOID){
15       return 0;
16   }
17   int main(int argc, char * argv[]){
18       if(PIN_Init(argc, argv)) return Usage();
19       trace = fopen("strace.out", "w");
20       PIN_AddSyscallEntryFunction(SyscallEntry, 0);
21       PIN_AddFiniFunction(Fini, 0);
22       PIN_StartProgram();
23       return 0;
24   }
```
Applications: anti-PIN malware analysis

Our Result

- **tElock, Safengine Shielden:**
  - PIN failed to run the testing programs generated by tElock and Safengine Shielden, which detected the presence of PIN and exited at early stages.
  - The testing programs ran successfully on PEMU.

- **eXait:**
  - 17 out of 21 anti-instrumentation techniques detect the presence of PIN,
  - None of them detect the presence of PEMU.
Applications: Virtual Machine Introspection

Building VMI tools

- **PEMU** can be used to build many out-of-VM introspection tools.
- A number of such tools have been built atop **PEMU**:
  - **VMST** [FL12]
  - **EXTERIOR** [FL13]
Limitation and Future Work

- Currently Incomplete support of the PIN-APIs
- A weak semantic gap [JBZ+14]
- No optimization with the generated instrumentation and analysis routine yet
Currently Incomplete support of the PIN-APIs
⇒ Make PEMU open source

A weak semantic gap [JBZ+14]
⇒ Not trust the guest OS kernel at all

No optimization with the generated instrumentation and analysis routine yet
Conclusion

**PEMU**

A new dynamic binary code instrumentation framework

1. PIN-API compatibility
2. Cross-OS (support both Windows and Linux)
3. Out-of-VM (strong isolation with the analysis routine and the target code)
Availability

The source code of PEMU is available at:
https://github.com/utds3lab/pemu
Thank you


———, Exterior: Using a dual-vm based external shell for guest-os introspection, configuration, and recovery, Proceedings of the Ninth Annual International Conference on Virtual Execution Environments (Houston, TX), March 2013.


Barton P. Miller and Andrew R. Bernat, Anywhere, any time binary instrumentation, September 2011.

Nicholas Nethercote and Julian Seward, *Valgrind: A program supervision framework*, In Third Workshop on Runtime Verification (RV’03), 2003.


Heng Yin and Dawn Song, *Temu: Binary code analysis via whole-system layered annotative execution*, Technical Report UCB/EECS-2010-3, EECS Department, University of California, Berkeley, Jan 2010.