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 15th, 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.

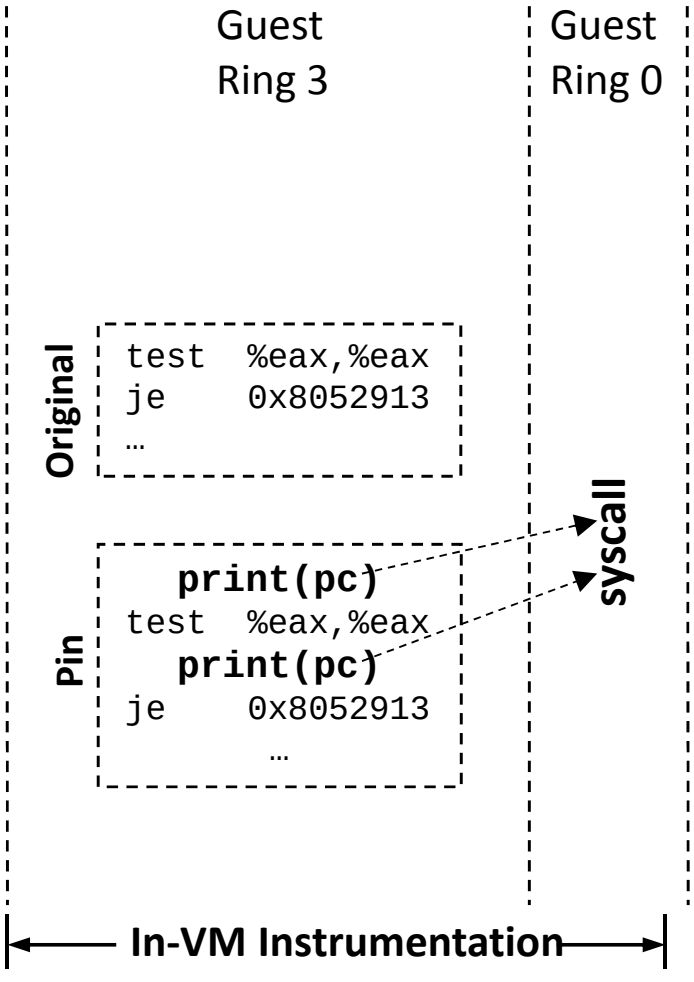
Dynamic Binary Instrumentation (DBI)

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

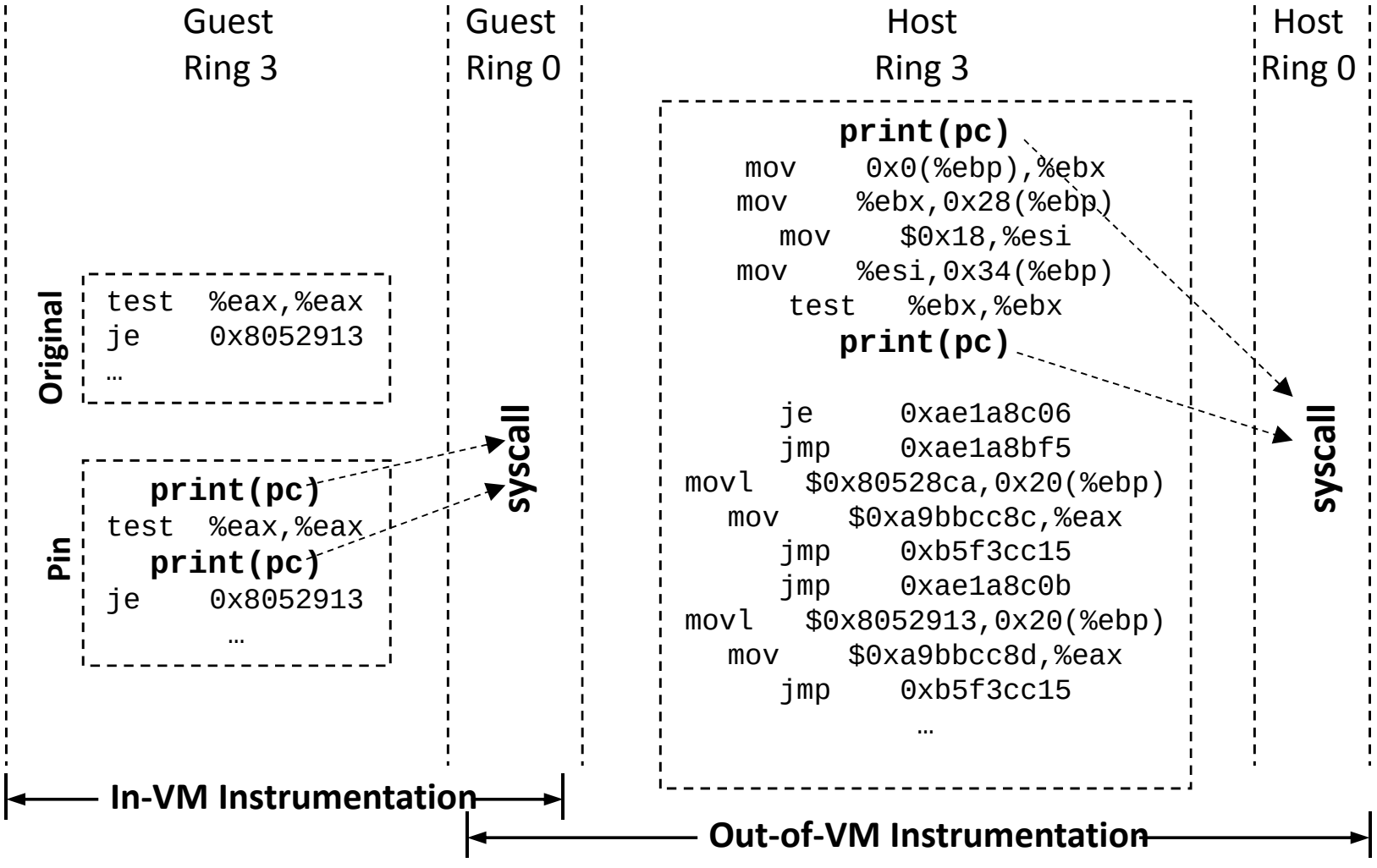
Applications

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
Guest
Ring 0
Original
test %eax,%eax
je 0x8052913
…
Pin
print(pc)
test %eax,%eax
print(pc)
je 0x8052913
…
syscall
In-VM Instrumentation

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



State-of-the-Art

Platforms	Year	Emulator, Simulator, Virtualizer	Kernel Level Instrumentation	User Level Instrumentation	w/ API for instrumentation	Out-of-VM	Guest OS Agnostic	PIN API Compatible	Open Source
EMBRA [WR96]	1996	✓	✓	✓	✗	✓	✗	✗	✗
VMWARE [DBR98]	1998	✓	✓	✓	✗	✓	✗	✗	✗
KERNINST [TM99]	1999	✗	✓	✓	✓	✗	✓	✗	✓
DYNINSTAPI [BH00]	2000	✗	✓	✓	✓	✗	✓	✗	✓
DYNAMO [BDB00]	2000	✗	✓	✓	✗	✗	✓	✗	✗
BOCHS [boc01]	2001	✓	✓	✓	✗	✓	✗	✗	✓
SIMICS [MCE+02]	2002	✓	✓	✓	✗	✓	✗	✗	✗
VALGRIND [NS03, NS07]	2003	✗	✗	✓	✓	✗	✓	✗	✓
STRATA [SKV+03]	2003	✗	✗	✓	✓	✗	✓	✗	✓
DYNAMORIO [BDB00, BZA12]	2004	✗	✗	✓	✓	✗	✓	✗	✓
QEMU [Bel05]	2005	✓	✓	✓	✗	✓	✗	✗	✓
PIN [LCM+05]	2005	✗	✗	✓	✓	✗	✓	✓	✗
NIRVANA [BCdJ+06]	2006	✗	✗	✓	✓	✗	✓	✗	✗
HDTRANS [SSNB06]	2006	✗	✗	✓	✓	✗	✓	✗	✓
VIRTUALBOX [Wat08]	2007	✓	✓	✓	✗	✓	✗	✗	✓
PINOS [BL07]	2007	✓	✓	✓	✓	✗	✗	✓	✗
TEMU [YS]	2010	✓	✓	✓	✓	✓	✓	✗	✓
DYNINST [MB11]	2010	✗	✓	✓	✓	✗	✓	✗	✓
DRK [PFG12]	2013	✗	✓	✓	✓	✗	✓	✗	✓
DECAF [HPY+14]	2014	✓	✓	✓	✓	✓	✗	✗	✓
PEMU	2015	✓	✓	✓	✓	✓	✓	✓	✓

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

Platforms	Year	Emulator, Simulator, Virtualizer	Kernel Level Instrumentation	User Level Instrumentation	w/ API for instrumentation	Out-of-VM	Guest OS Agnostic	PIN API Compatible	Open Source
EMBRA [WR96]	1996	✓	✓	✓	✗	✓	✗	✗	✗
VMWARE [DBR98]	1998	✓	✓	✓	✗	✓	✗	✗	✗
KERNINST [TM99]	1999	✗	✓	✓	✓	✗	✓	✗	✓
DYNINSTAPI [BH00]	2000	✗	✓	✓	✓	✗	✓	✗	✓
DYNAMO [BDB00]	2000	✗	✓	✓	✗	✗	✓	✗	✗
BOCHS [boc01]	2001	✓	✓	✓	✗	✓	✗	✗	✓
SIMICS [MCE+02]	2002	✓	✓	✓	✗	✓	✗	✗	✗
VALGRIND [NS03, NS07]	2003	✗	✗	✓	✓	✗	✓	✗	✓
STRATA [SKV+03]	2003	✗	✗	✓	✓	✗	✓	✗	✓
DYNAMORIO [BDB00, BZA12]	2004	✗	✗	✓	✓	✗	✓	✗	✓
QEMU [Bel05]	2005	✓	✓	✓	✗	✓	✗	✗	✓
PIN [LCM+05]	2005	✗	✗	✓	✓	✗	✓	✓	✗
NIRVANA [BCdJ+06]	2006	✗	✗	✓	✓	✗	✓	✗	✗
HDTRANS [SSNB06]	2006	✗	✗	✓	✓	✗	✓	✗	✓
VIRTUALBOX [Wat08]	2007	✓	✓	✓	✗	✓	✗	✗	✓
PINOS [BL07]	2007	✓	✓	✓	✓	✗	✗	✓	✗
TEMU [YS]	2010	✓	✓	✓	✓	✓	✓	✗	✓
DYNINST [MB11]	2010	✗	✓	✓	✓	✗	✓	✗	✓
DRK [PFG12]	2013	✗	✓	✓	✓	✗	✓	✗	✓
DECAF [HPY+14]	2014	✓	✓	✓	✓	✓	✗	✗	✓
PEMU	2015	✓	✓	✓	✓	✓	✓	✓	✓

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

```
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`

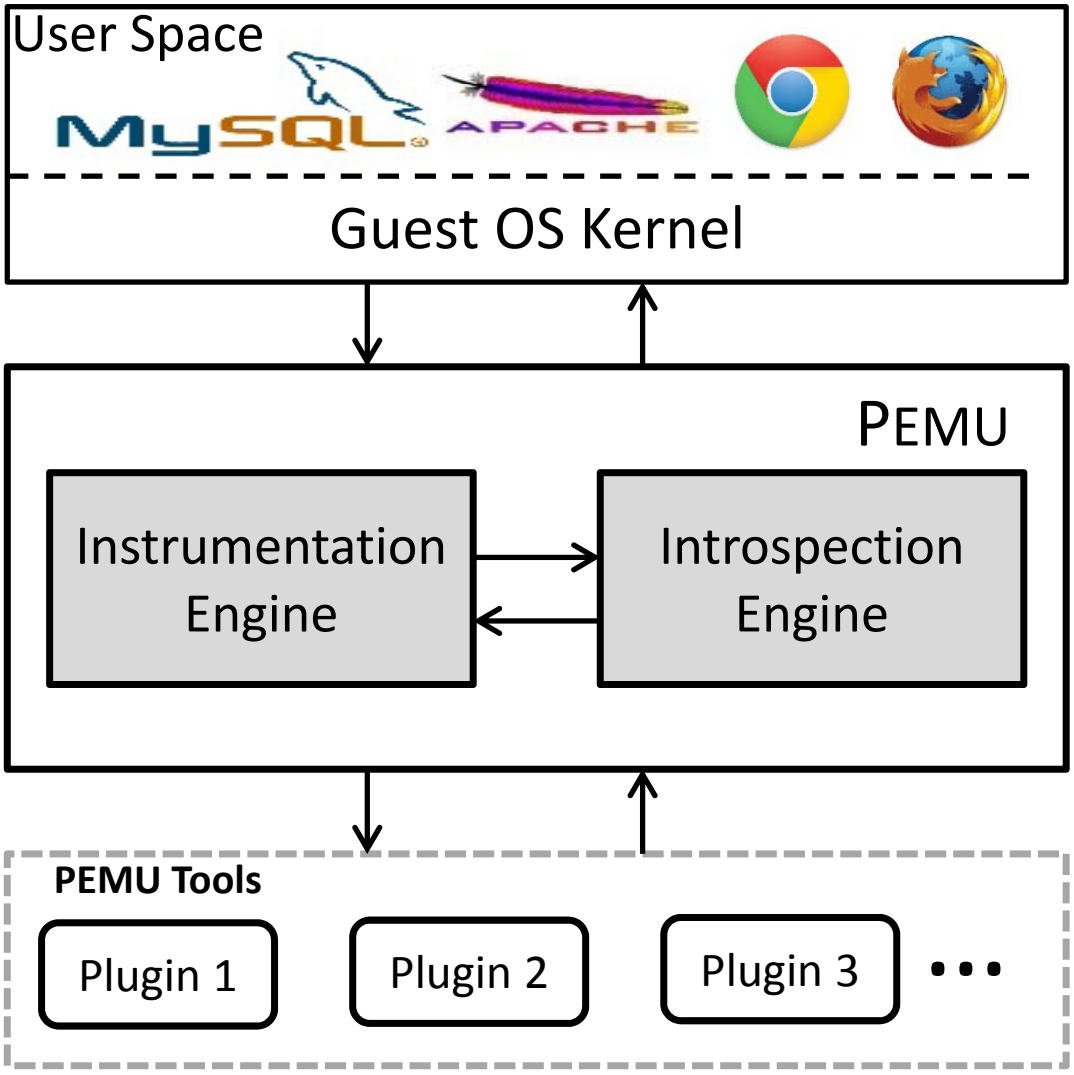
Challenges

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

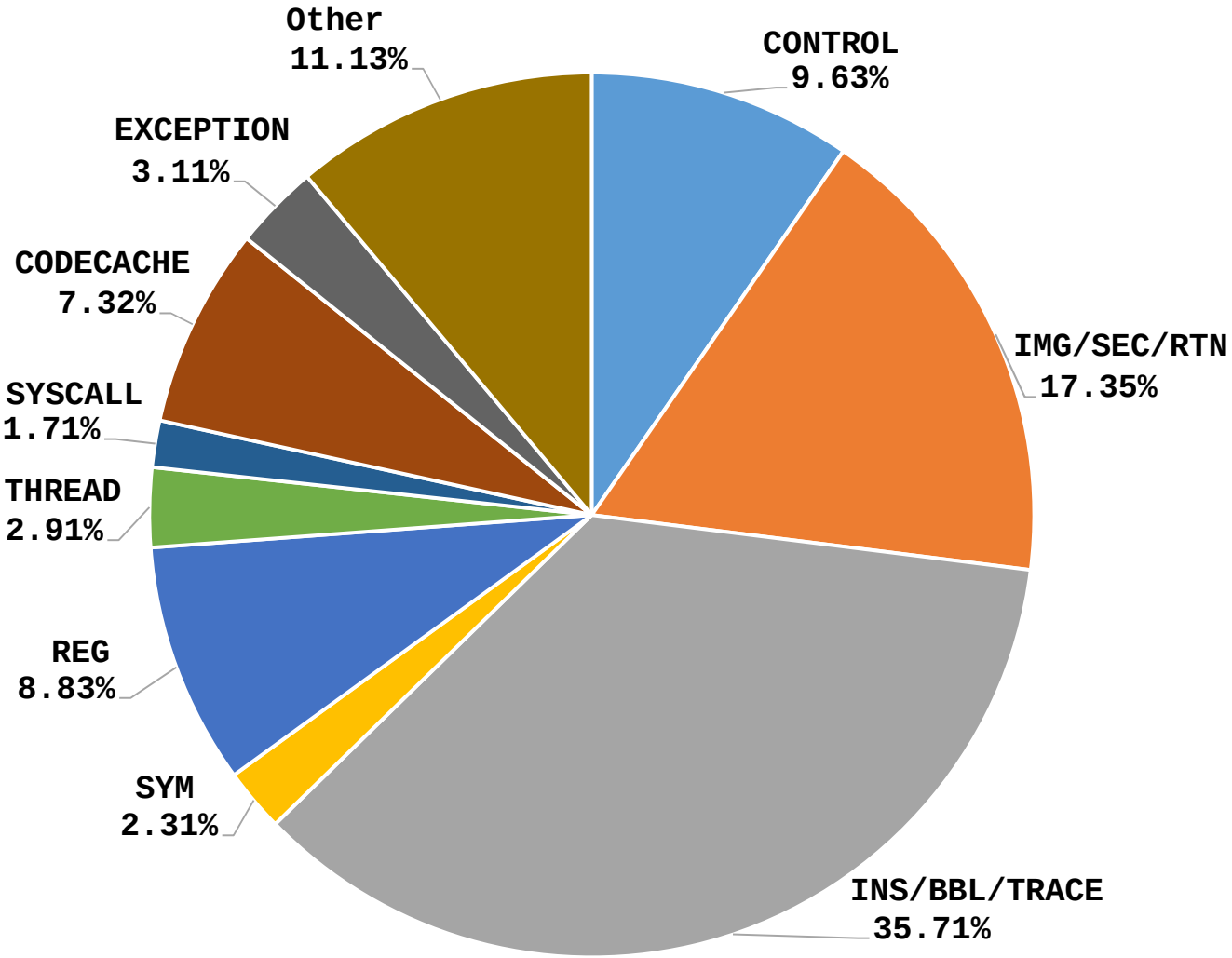
   $\Rightarrow$ Instrumentation Engine
2. **Semantic gap between guest OS and host OS**
   - How to find the target process or thread
   - PIN plugins may inspect guest OS state, such as `getpid`

   $\Rightarrow$ Introspection Engine

PEMU Architecture



Instrumentation Engine: PIN APIs



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,..)`

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

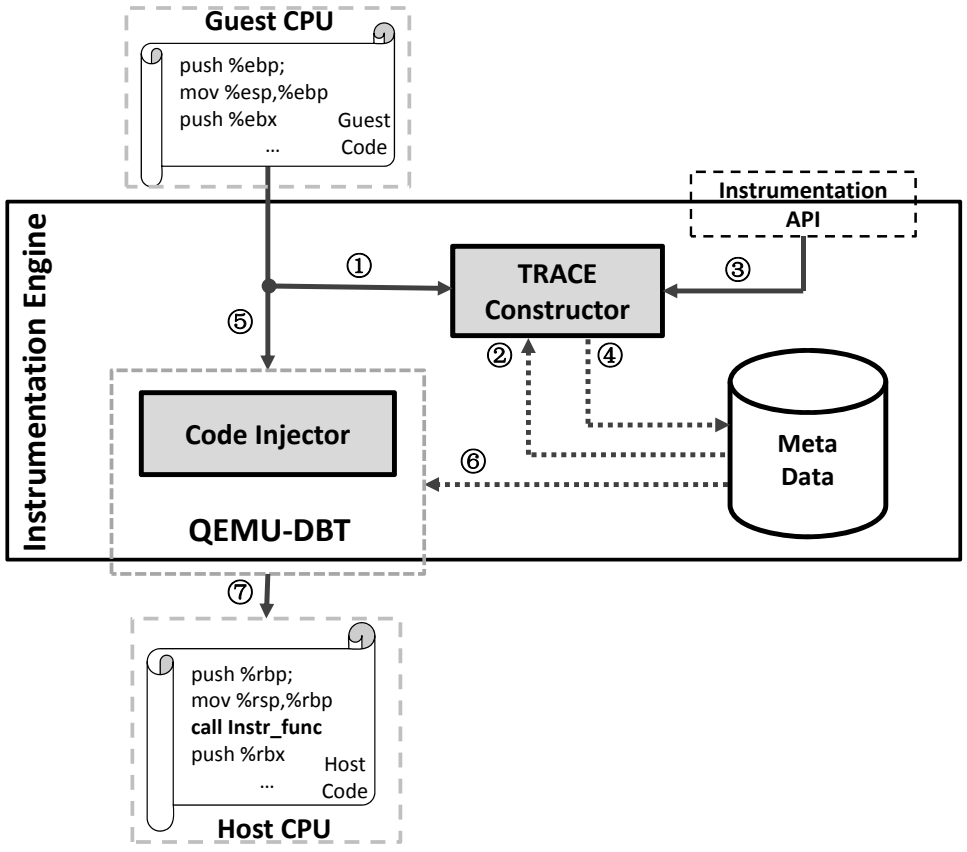
Instrumentation Engine

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,..)`

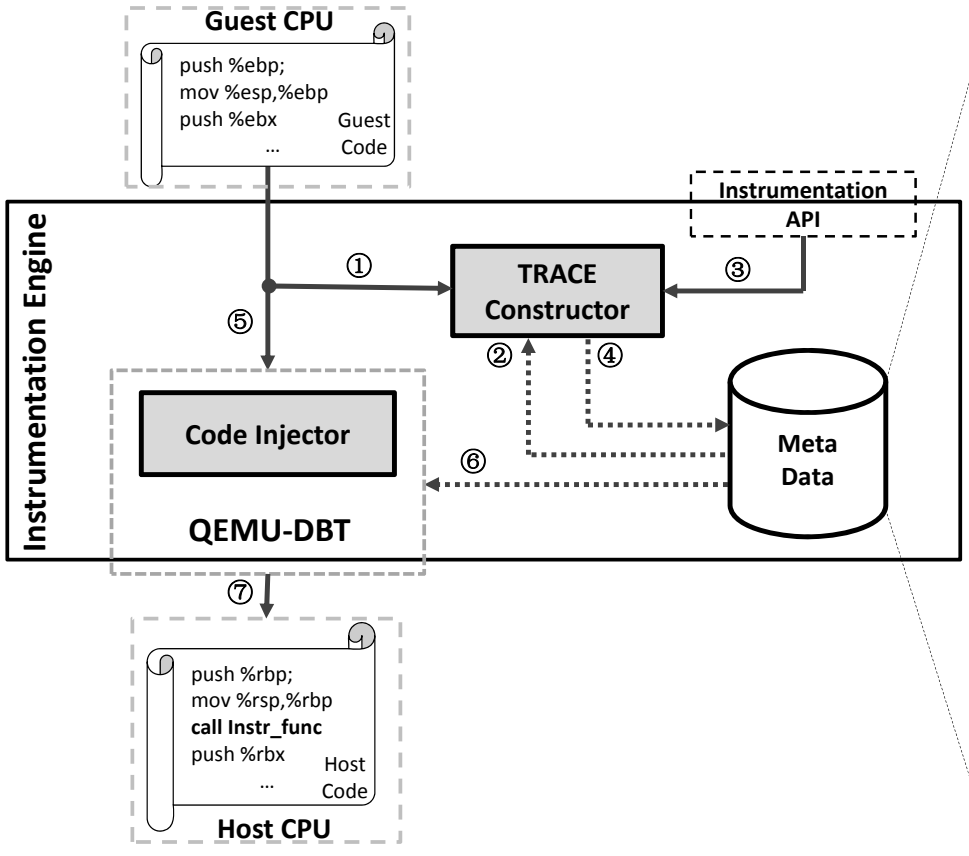
   $\Rightarrow$ Adding our own disassembler (TRACE Constructor)
2. **Ahead-of-time Instrumentation**: caches the instrumentation before the execution.
   - IMG instrumentation, e.g., `IMG_AddInstrumentFunction`
   - RTN instrumentation, eg.., `RTN_InsertCall`

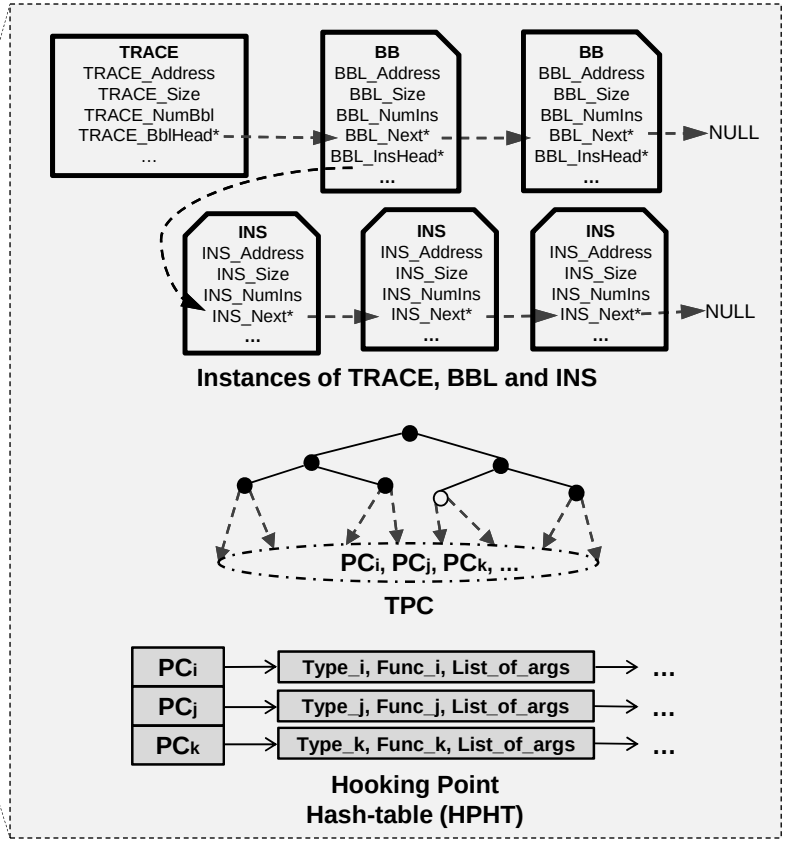
   $\Rightarrow$ Instrumenting code when image is loaded

Instrumentation Engine



Instrumentation Engine



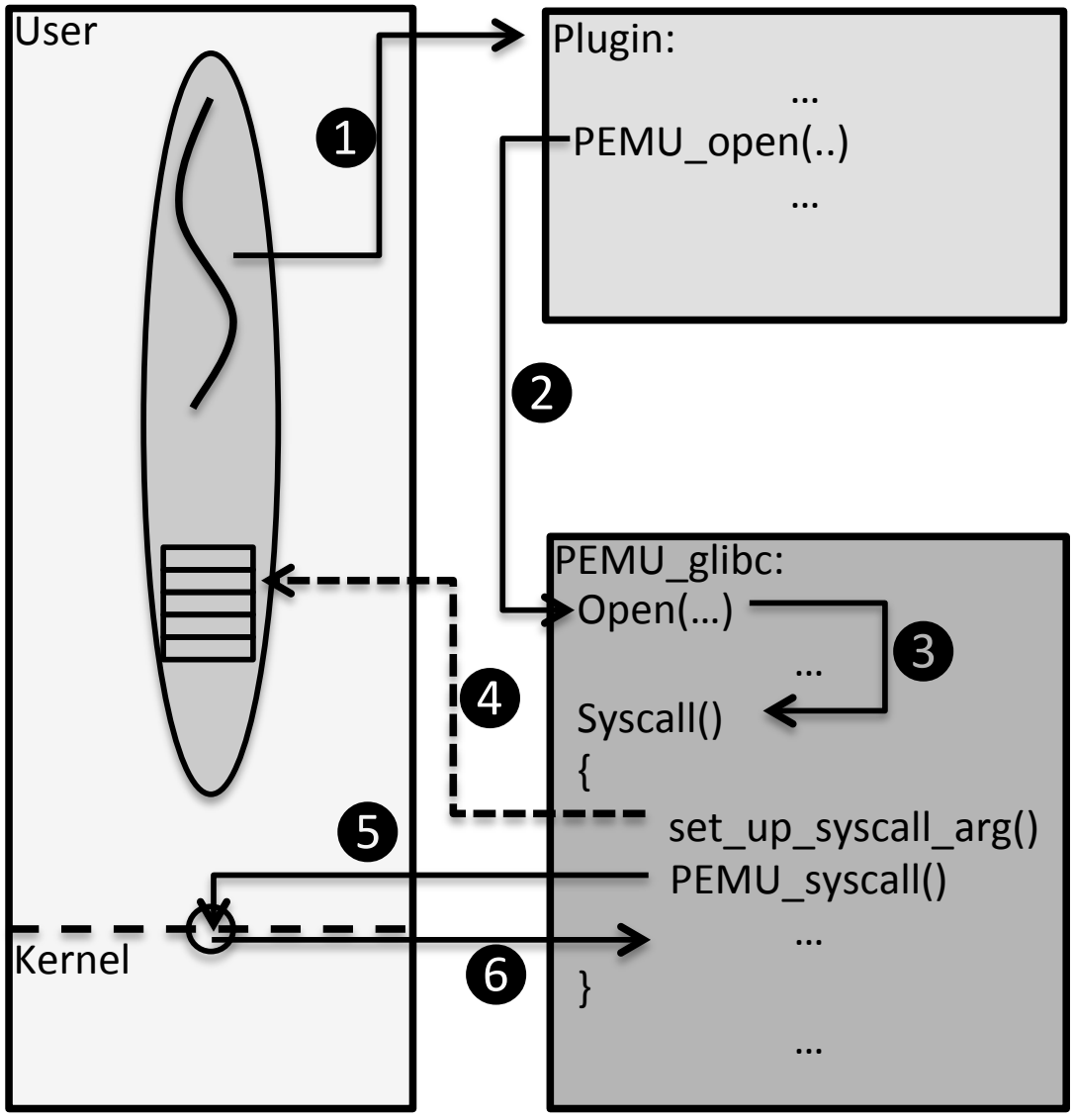


Introspection Engine

Goals and Solutions

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



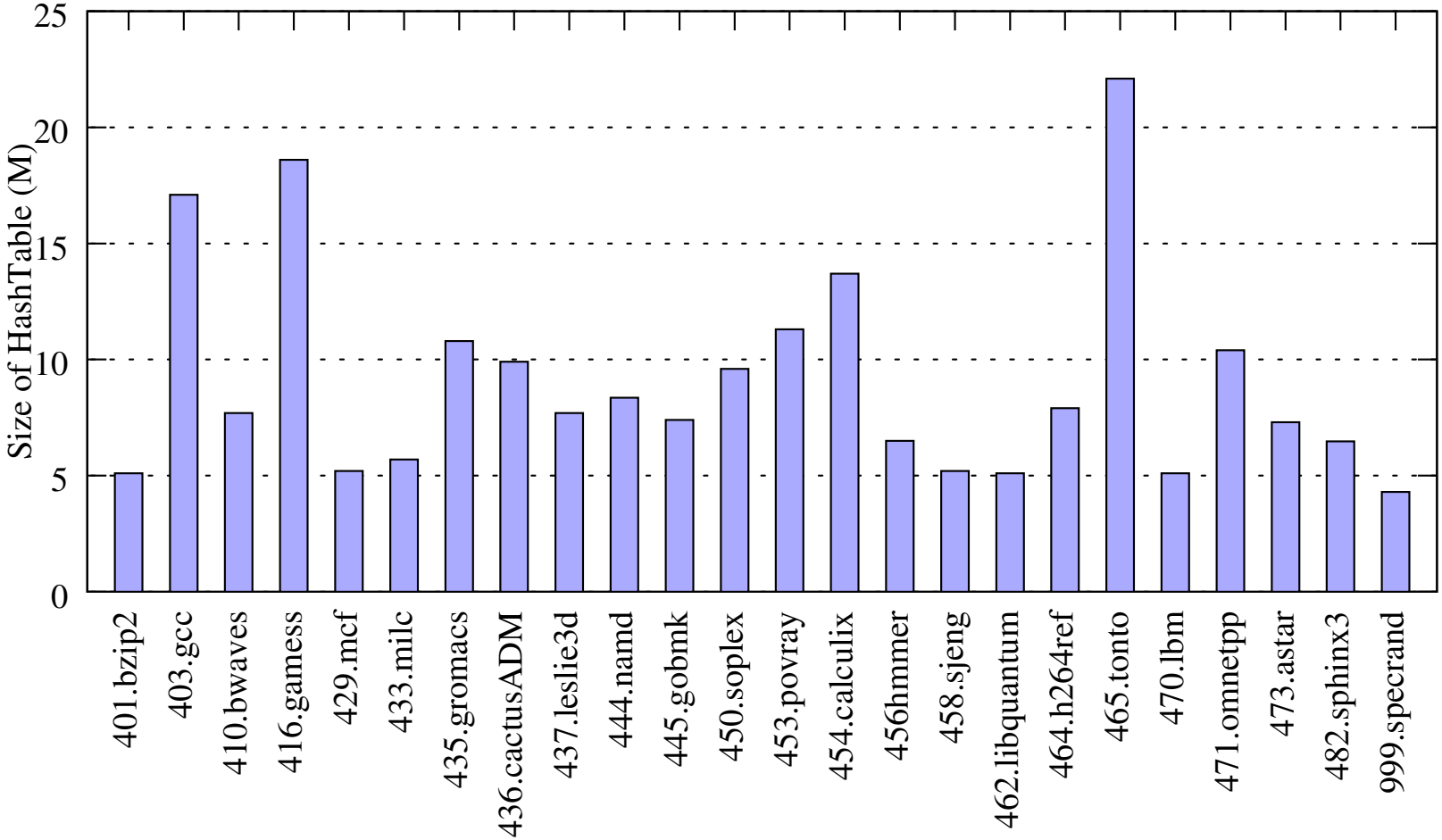
Compatibility Testing With PIN Plugins

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

Performance Evaluation: Speed

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

Performance Evaluation: Memory Cost



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

```
1  FILE *trace;
2  VOID SysBefore(ADDRINT ip, ADDRINT num) {
3     fprintf(trace,"0x%lx: %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

Limitation and Future Work

- Currently Incomplete support of the PIN-APIs

$\Rightarrow$ Make PEMU open source

- A weak semantic gap [JBZ+14]

$\Rightarrow$ 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

References I


Sanjay Bhansali, Wen-Ke Chen, Stuart de Jong, Andrew Edwards, Ron Murray, Milenko Drinić, Darek Mihočka, and Joe Chau, Framework for instruction-level tracing and analysis of program executions, Proceedings of the 2Nd International Conference on Virtual Execution Environments (New York, NY, USA), VEE '06, ACM, 2006, pp. 154–163.


Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia, Dynamo: A transparent dynamic optimization system, Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (New York, NY, USA), PLDI '00, ACM, 2000, pp. 1–12.


Fabrice Bellard, Qemu, a fast and portable dynamic translator, Proceedings of the annual conference on USENIX Annual Technical Conference (Berkeley, CA, USA), ATEC '05, USENIX Association, 2005.


Bryan Buck and Jeffrey K. Hollingsworth, An api for runtime code patching, Int. J. High Perform. Comput. Appl. **14** (2000), no. 4, 317–329.


Prashanth P. Bungale and Chi-Keung Luk, Pinos: A programmable framework for whole-system dynamic instrumentation, Proceedings of the 3rd international conference on Virtual execution environments, 2007, pp. 137–147.


bochs: The open source ia-32 emulation project, 2001, http://bochs.sourceforge.net/.


Derek Bruening, Qin Zhao, and Saman Amarasinghe, Transparent dynamic instrumentation, Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (New York, NY, USA), VEE '12, ACM, 2012, pp. 133–144.


Scott W. Devine, Edouard Bugnion, and Mendel Rosenblum, Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture, United States Patent 6,397,242 (1998).

References II


Yangchun Fu and Zhiqiang Lin, Space traveling across vm: Automatically bridging the semantic-gap in virtual machine introspection via online kernel data redirection, Proceedings of the 2012 IEEE Symposium on Security and Privacy (San Francisco, CA), May 2012.


_____, 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.


Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin, Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform, Proceedings of the 2014 International Symposium on Software Testing and Analysis (New York, NY, USA), ISSTA 2014, ACM, 2014, pp. 248–258.


Bhushan Jain, Mirza Basim Baig, Dongli Zhang, Donald E. Porter, and Radu Sion, Sok: Introspections on trust and the semantic gap, Proceedings of the 2014 IEEE Symposium on Security and Privacy (Washington, DC, USA), SP '14, IEEE Computer Society, 2014, pp. 605–620.


Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, Pin: building customized program analysis tools with dynamic instrumentation, Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (New York, NY, USA), PLDI '05, ACM, 2005, pp. 190–200.


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


Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, Simics: A full system simulation platform, Computer **35** (2002), no. 2, 50–58.

References III


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


Nicholas Nethercote and Julian Seward, Valgrind: A framework for heavyweight dynamic binary instrumentation, Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (New York, NY, USA), PLDI '07, ACM, 2007, pp. 89–100.


Angela Demke Brown Peter Feiner and Ashvin Goel, Comprehensive kernel instrumentation via dynamic binary translation, Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems, 2012.


K. Scott, N. Kumar, S. Velusamy, B. Childers, J. W. Davidson, and M. L. Soffa, Retargetable and reconfigurable software dynamic translation, Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (Washington, DC, USA), CGO '03, IEEE Computer Society, 2003, pp. 36–47.


Swaroop Sridhar, Jonathan S. Shapiro, Eric Northup, and Prashanth P. Bungale, Hdtrans: An open source, low-level dynamic instrumentation system, Proceedings of the 2Nd International Conference on Virtual Execution Environments (New York, NY, USA), VEE '06, ACM, 2006, pp. 175–185.


Ariel Tamches and Barton P. Miller, Fine-grained dynamic instrumentation of commodity operating system kernels, Proceedings of the Third Symposium on Operating Systems Design and Implementation (Berkeley, CA, USA), OSDI '99, USENIX Association, 1999, pp. 117–130.


Jon Watson, Virtualbox: Bits and bytes masquerading as machines, Linux J. **2008** (2008), no. 166.

References IV


Emmett Witchel and Mendel Rosenblum, Embra: Fast and flexible machine simulation, Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (New York, NY, USA), SIGMETRICS '96, ACM, 1996, pp. 68–79.


Junyuan Zeng Yangchun Fu and Zhiqiang Lin, Hypershell: A practical hypervisor layer guest os shell for automated in-vm management, USENIX ATC'14 Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX Association Berkeley, CA, USA), USENIX Association, 2014, pp. 85–96.


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.