Defeating the Transparency Features of Dynamic Binary Instrumentation

The detection of DynamoRIO through introspection

Xiaoning Li  
ldpatchguard@gmail.com

Kang Li  
kangli@uga.edu
About us

- Xiaoning
  - Security Researcher

- Kang
  - College Educator
What is Instrumentation

...  

Some Random Piece of Code (from QEMU)

```c
if (size < sizeof(min_buf)) {
    iov_to_buf iov, iovcnt, 0, min_buf, size);
    memset(&min_buf[size], 0, sizeof(min_buf) - size);
}
else if (iov->iov_len < MAXIMUM_ETHERNET_HDR_LEN) {
    /* This is very unlikely, but may happen. */
    iov_to_buf iov, iovcnt, 0, min_buf,
    MAXIMUM_ETHERNET_HDR_LEN);
    filter_buf = min_buf;
}
...
What is Instrumentation

... Some Random Piece of Code (from QEMU)

```c
if (size < sizeof(min_buf)) {
    printf("good size branch \n");
    iov_to_buf(iov, iovcnt, 0, min_buf, size);
    memset(&min_buf[size], 0, sizeof(min_buf) - size);
}
else if (iov->iov_len < MAXIMUM_ETHERNET_HDR_LEN) {
    printf("got a rare case \n");
    /* This is very unlikely, but may happen. */
    iov_to_buf(iov, iovcnt, 0, min_buf,
                MAXIMUM_ETHERNET_HDR_LEN);
    filter_buf = min_buf;
}
...
```

Instrumentation: inserting extra code to observe run-time behavior
Binary Instrumentation

mov    $0x0,%esi
mov    %rax,%rdi
mov    $0x0,%eax
callq  400920 <open@plt>
mov    %eax,-0x9b0(%rbp)
cmpl   $0x0,-0x9b0(%rbp)
jns    400b74 <test_sigcgt+0x7c>

Pre-instruction Hook
Post-instruction Hook
Binary Instrumentation

mov $0x0, %esi
mov %rax, %rdi
mov $0x0, %eax
callq 400920 <open@plt>
mov %eax, -0x9b0(%rbp)
cmpl $0x0, -0x9b0(%rbp)
jns 400b74 <test_sigcgt+0x7c>
Binary Instrumentation

mov $0x0,%esi
mov %rax,%rdi
mov $0x0,%eax
callq 400920 <open@plt>
mov %eax,-0x9b0(%rbp)
cmpl $0x0,-0x9b0(%rbp)
jns 400b74 <test_sigcgt+0x7c>

counter++;
counter++;
counter++;
counter++;
counter++;
counter++;

Concept Similar to Source Level Instrumentation
Binary Instrumentation

Call Graph

Instrumentation can be done at the Code Block level
Binary Instrumentation

Call Graph

Instrumentation can be done at the Code Block level
Dynamic Binary Instrumentation (DBI)

Original Code

Code Cache

Dynamic Instrumentation via Code Cache
Dynamic Binary Instrumentation (DBI)

Original Code

1
2
3
4
5
6
7

Copy code block & start execution in the Code Cache

Code Cache

1

Instrumentation in Code Cache

Dynamic Instrumentation via Code Cache
Dynamic Binary Instrumentation (DBI)

Original Code

Code Cache

Load Block if not already in Cache

Dynamic Instrumentation via Code Cache
Dynamic Binary Instrumentation (DBI)

Original Code

Dynamic Instrumentation via Code Cache

Load more based on execution result
The Increasing Use of DBI

- **Function:**
  - Observing execution
  - Hardening and protection

- **Useful for**
  - Profiling and optimization
  - Reverse engineering
  - Malware analysis
Popular DBI Tools

- Process level:
Demand of Transparency!

- Matching the native behavior
  - E.g.
    - No change to program execution flow
    - No obvious overhead

- Special effort towards transparency
  - E.g.
    - Making no assumptions about memory usage
    - Hide code cache management and instrumentation code
Example of Preserving Transparency

- **Library Transparency in DynamoRIO**

  - **Execution in code cache needs DynamoRIO library calls**
    - E.g.
      - for the start of app from code cache
      - for translation between code cache and app addresses

  - **DynamoRIO uses a custom loader for its libraries**
    - E.g.
      - DLL is loaded to App process space, but “invisible” from App.
      - EnumProcessModules() shows no DLLs from DynamoRIO.
### Transparency Features in DynamoRIO

<table>
<thead>
<tr>
<th>I/O Transparency</th>
<th>Library Transparency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Transparency</td>
<td>Resource Transparency</td>
</tr>
<tr>
<td>Memory Transparency</td>
<td>Address Transparency</td>
</tr>
<tr>
<td></td>
<td>Debugging Transparency</td>
</tr>
</tbody>
</table>
Exposing DBI

DBI detection case studies based on DynamoRIO

Image Source: http://dragonball.wikia.com
Example #1: Cause DynamoRIO to crash
DynamoRIO Crash Code

- Code pieces
  - Works correctly on Native
  - But crashes DynamoRIO if running with it

- For example: Heap as stack
Comparing Code

- **Original Code**

  ```
  push eax
  mov eax, dword_49eda8
  xchg eax, esp
  push eax
  call Dst
  pop eax
  xchg eax, esp
  ```

- **Code in Code cache**

  ```
  push eax
  mov eax, dword ptr [drcrash!basebuffer (012aeda8)]
  xchg eax,esp
  push eax
  mov dword ptr fs:[OECh],ecx
  mov ecx,dword ptr [drcrash!lpvResult (012aeda4)]
  push offset drcrash!test_jit_code+0x7d (0121162d)
  jmp 225d1e40
  ```
Example #2: Simple Implementation Artifact
Simple Heuristics for DBI Detection

- Implementation Artifact
  - Parent Process Name
    - Detection by checking who is the parent!
    - InheritedFromUniqueProcessId shows the father is drrun.exe

- “File” Handler Number
  - Handler Count
    - DynamoRIO: 0x17  Native: 0x0d

- Max Open File Handlers
  - 4000 vs. 4096 (on Linux)
Detection by Abnormal Resource Usage

- **Peak Memory Usage**
  - **PeakVirtualSize** (on our sample program)
    - With DynamoRIO: 0x8e7c000 bytes
    - Without: 0x0d73000 bytes

- **Other Anomaly Behavior**
  - E.g. Setting Max Open File handler (on Linux)
    - `setrlimit(RLIMIT_NOFILE, 1024)` fails even when current limit is 1024
Detecting DynamoRIO by Signal Masks

- DynamoRIO capture all signals and relays them
  - To observe all signals while avoiding modify signal handlers
  - To preserve transparency

- Consequence (on Linux):
  - Application with DynamoRIO:
    SIGCGT mask: 0xFFFFFFFFFFFFFFC1FEF
  - Native Application:
    SIGCGT mask: 0x0000000000001000
Example #3: Detecting DynamoRIO Library
Detecting DynamoRIO Library

- **Library Transparency**
  - DynamoRIO library needs to be in the App process
  - DynamoRIO hides its DLL from the Process

- However, the code cache management code has to be in process memory!
Detecting DynamoRIO Library

- Scanning for all PE/DLLs in process memory

- Identify hidden DLLs by comparing with the list from EnumProcessModules()

- Identifying DynamoRIO library
  - Searching hidden library for DynamoRIO data
  - Searching for DynamoRIO code
  - GetProcAddress for DynamoRIO DLL APIs
Example #4: Measuring Error Transparency Behavior
Error Transparency Detection

- Designed code to trigger exception

```assembly
call $+5
pop eax
Invalid ISA
```

- In exception handler, exception record eax/eip distance should be one
- Trigger this code via self modified code
On Native Windows 7 32-bits

code base: 1061ebe
Exception Eip: 1061ebc
Exception Eax: 1061ebb
unknownisa = 1
EXCEPTION_ACCESS_VIOLATION_counter = 0
EXCEPTION_DATATYPE_MISALIGNMENT_counter = 0
EXCEPTION_BREAKPOINT_counter = 1
EXCEPTION_SINGLE_STEP_counter = 0
EXCEPTION_ARRAY_BOUNDS_EXCEEDED_counter = 0
EXCEPTION_FLT_DENORMAL_OPERAND_counter = 0
EXCEPTION_FLT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_FLT_INEXACT_RESULT_counter = 0
EXCEPTION_FLT_INVALID_OPERATION_counter = 0
EXCEPTION_FLT_OVERFLOW_counter = 0
EXCEPTION_FLT_STACK_CHECK_counter = 0
EXCEPTION_FLT_UNDERFLOW_counter = 0
EXCEPTION_INT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_INT_OVERFLOW_counter = 0
EXCEPTION_PRIV_INSTRUCTION_counter = 0
EXCEPTION_IN_PAGE_ERROR_counter = 0
EXCEPTION_ILLEGAL_INSTRUCTION_counter = 1
EXCEPTION_NONCONTINUABLE_EXCEPTION_counter = 0
EXCEPTION_STACK_OVERFLOW_counter = 0
EXCEPTION_INVALID_DISPOSITION_counter = 0
EXCEPTION_GUARD_PAGE_counter = 0
EXCEPTION_INVALID_HANDLE_counter = 0
EXCEPTION_INVALID_LOCK_SEQUENCE_counter = 0
unknownisa_others = 0
Code at Runtime

```
01061e9d 60    pushad
01061e9e 894598    mov    dword ptr [ebp-68h].eax
01061ea1 895dbc    mov    dword ptr [ebp-44h].ebx
01061ea4 894da4    mov    dword ptr [ebp-5Ch].ecx
01061ea7 8955a0    mov    dword ptr [ebp-60h].edx
01061eaa 897da8    mov    dword ptr [ebp-58h].edi
01061ead 89f5cc    mov    dword ptr [ebp-34h].esi
01061eb0 896dc4    mov    dword ptr [ebp-3Ch].ebp
01061eb3 8965b0    mov    dword ptr [ebp-50h].esp
01061eb6 e80000000    call    dr_detection_exception+0x1ebb (01061ebb)
01061ebb 58    pop    eax
01061ebc 66    ret
01061ebd 0f    ???
01061ee0 0f0000    sldt    word ptr [eax]
01061ec1 90    nop
01061ec2 90    nop
01061ec3 90    nop
01061ec4 90    nop
01061ec5 90    nop
01061ec6 90    nop
01061ec7 90    nop
01061ec8 90    nop
01061ec9 90    nop
01061eca 90    nop
01061ecb 90    nop
01061ecc 89459c    mov    dword ptr [ebp-64h].eax
01061ecf 895dd0    mov    dword ptr [ebp-30h].ebx
01061ed2 894dd4    mov    dword ptr [ebp-2Ch].ecx
01061ed5 8955b8    mov    dword ptr [ebp-48h].edx
01061ed8 897dc0    mov    dword ptr [ebp-40h].edi
01061edb 89f5cc    mov    dword ptr [ebp-38h].esi
01061ede 896db4    mov    dword ptr [ebp-4Ch].ebp
01061ee1 8965ac    mov    dword ptr [ebp-54h].esp
01061ee4 8bb65b0    mov    esp,dword ptr [ebp-50h]
01061ee7 61    popad
```
Code Property

0:001> !address 1061ebc
ProcessParameters 00281948 in range 00280000 00291000
Environment 002807f0 in range 00280000 00291000
  01060000 : 01061000 - 00001000
  Type          01000000 MEM_IMAGE
  Protect       00000040 PAGE_EXECUTE_READWRITE
  State         00001000 MEM_COMMIT
  Usage         RegionUsageImage
On Native Windows 7 32-bits + DynamoRIO
Code in Runtime

```
00031e9d 60          pushad
00031e9e 894598      mov    dword ptr [ebp-68h].eax
00031ea1 895dbc      mov    dword ptr [ebp-44h].ebx
00031ea4 894da4      mov    dword ptr [ebp-5Ch].ecx
00031ea7 8955a0      mov    dword ptr [ebp-60h].edx
00031eaa 897da8      mov    dword ptr [ebp-58h].edi
00031ead 8975cc      mov    dword ptr [ebp-34h].esi
00031eb0 896dc4      mov    dword ptr [ebp-3Ch].ebp
00031eb3 8965b0      mov    dword ptr [ebp-50h].esp
00031eb6 e800000000  call   dr_detection_exception+0x1ebb (00031ebb)
00031ebb 58          pop    eax
00031ebc 66          ???
00031ebd 0f          ???
00031ebe 0f0000      sldt   word ptr [eax]
00031ec1 90          nop
00031ec2 90          nop
00031ec3 90          nop
00031ec4 90          nop
00031ec5 90          nop
00031ec6 90          nop
00031ec7 90          nop
00031ec8 90          nop
00031ec9 90          nop
00031ecb 90          nop
00031eccc 89459c     mov    dword ptr [ebp-64h].eax
00031ecf 895dd0      mov    dword ptr [ebp-30h].ebx
00031ed2 894dd4      mov    dword ptr [ebp-2Ch].ecx
00031ed5 8955b8      mov    dword ptr [ebp-48h].edx
00031ed8 897dc0      mov    dword ptr [ebp-40h].edi
00031edb 8975c8      mov    dword ptr [ebp-38h].esi
00031ede 896db4      mov    dword ptr [ebp-4Ch].ebp
00031eef 8965ac      mov    dword ptr [ebp-54h].esp
```
Code Property

0:001> !address 00031ecc
ProcessParameters 00301948 in range 00300000 00311000
Environment 003007f0 in range 00300000 00311000
  00030000 : 00031000 - 0004d000
  Type 01000000 MEM_IMAGE
  Protect 00000020 PAGE_EXECUTE_READ
  State 00001000 MEM_COMMIT
  Usage RegionUsageImage
Fixed by revision r2688 😊
(May, 2014)
Example #5: Unexpected Exception
Calculate Code Checksum

```
00110000  push  eax
00110001  push  ebx
00110002  call  $+5
00110007  pop   eax
00110008  xor   ebx, ebx
0011000A  add   ebx, [eax+20h]
0011000D  add   ebx, [eax+21h]
00110010  add   ebx, [eax+22h]
00110013  add   ebx, [eax+23h]
00110016  add   ebx, [eax+24h]
00110019  add   ebx, [eax+25h]
0011001C  add   ebx, [eax+26h]
0011001F  add   ebx, [eax+27h]
00110022  add   ebx, [eax+28h]
00110025  add   ebx, [eax+29h]
00110028  add   ebx, [eax+2Ah]
0011002B  add   ebx, [eax+2Bh]
```
On Native Windows 7 32-bits

codebase: f0000
checksum = a2a2a270

unknownisa = 0
EXCEPTION_ACCESS_VIOLATION_counter = 0
EXCEPTION_DATATYPE_MISALIGNMENT_counter = 0
EXCEPTION_BREAKPOINT_counter = 0
EXCEPTION_SINGLE_STEP_counter = 0
EXCEPTION_ARRAY_BOUNDS_EXCEEDED_counter = 0
EXCEPTION_FLT_DENORMAL_OPERAND_counter = 0
EXCEPTION_FLT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_FLT_INEXACT_RESULT_counter = 0
EXCEPTION_FLT_INVALID_OPERATION_counter = 0
EXCEPTION_FLT_OVERFLOW_counter = 0
EXCEPTION_FLT_STACK_CHECK_counter = 0
EXCEPTION_FLT_UNDERFLOW_counter = 0
EXCEPTION_INT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_INT_OVERFLOW_counter = 0
EXCEPTION_PRIV_INSTRUCTION_counter = 0
EXCEPTION_IN_PAGE_ERROR_counter = 0
EXCEPTION_ILLEGAL_INSTRUCTION_counter = 0
EXCEPTION_NONCONTINUABLE_EXCEPTION_counter = 0
EXCEPTION_STACK_OVERFLOW_counter = 0
EXCEPTION_INVALID_DISPOSITION_counter = 0
EXCEPTION_GUARD_PAGE_counter = 0
EXCEPTION_INVALID_HANDLE_counter = 0
EXCEPTION_INVALID_LOCK_SEQUENCE_counter = 0
unknownisa_others = 0
On Native Windows 7 32-bits + DynamoRIO

codebase: 2500000
GetExceptionCode() = c0000005
Eip: 00260000

unknownisa = 0
EXCEPTION_ACCESS_VIOLATION_counter = 1
EXCEPTION_DATATYPE_MISALIGNMENT_counter = 0
EXCEPTION_BREAKPOINT_counter = 0
EXCEPTION_SINGLE_STEP_counter = 0
EXCEPTION_ARRAY_BOUNDS_EXCEEDED_counter = 0
EXCEPTION_FLT_DENORMAL_OPERAND_counter = 0
EXCEPTION_FLT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_FLT_INEXACT_RESULT_counter = 0
EXCEPTION_FLT_INUVALID_OPERATION_counter = 0
EXCEPTION_FLT_OVERFLOW_counter = 0
EXCEPTION_FLT_STACK_CHECK_counter = 0
EXCEPTION_FLT_UNDERFLOW_counter = 0
EXCEPTION_INT_DIVIDE_BY_ZERO_counter = 0
EXCEPTION_INT_OVERFLOW_counter = 0
EXCEPTION_PRIV_INSTRUCTION_counter = 0
EXCEPTION_IN_PAGE_ERROR_counter = 0
EXCEPTION_ILLEGAL_INSTRUCTION_counter = 0
EXCEPTION_NONCONTINUABLE_EXCEPTION_counter = 0
EXCEPTION_STACK_OVERFLOW_counter = 0
EXCEPTION_INUVALID_DISPOSITION_counter = 0
EXCEPTION_GUARD_PAGE_counter = 0
EXCEPTION_INUVALID_HANDLE_counter = 0
EXCEPTION_INUVALID_LOCK_SEQUENCE_counter = 0
unknownisa_others = 0
What more can be done?
What can be done?

- To improve DBI transparency (evade detection)
  - Avoid implementation artifacts
  - A challenging task in general …

- To detect DBI
  - More systematic fuzzing
    - Comparing regular App and DBI-App side-by-side
  - Performance based detection
    - Design binary that triggers the most code cache overhead
Summary

- The increasing use of BT and DBI
  - Runtime program analysis

- Transparency is preserved very well for
  - regular applications, and even buggy applications that make invalid memory accesses

- Transparency is easily broken by detecting anomaly in
  - Resource usage
  - Hidden libraries
  - Exception Handling
Disclaimers and Acknowledgment

- **DynamoRIO Developers**
  - Providing Powerful Open Source DBI Framework
  - Targets are Benign Applications
  - Not Intentionally Designed for Evading Detection

- **Dr. Qin Zhao @ Google**
  - Respond to reports
  - Feedback to our slides

- **Research Support**
  - Dr. Kang Li’s research is partially supported by NSF award 1319115
Bonus Materials

Multiple Bytes NOPs
NOps

- No Operation Instruction
- 0x90 decoded as “xchg eax, eax”
- 1-9 bytes for X86

Examples:

66 NOP                       - 66 90H
NOP DWORD ptr [EAX]          - 0F 1F 00H
NOP DWORD ptr [EAX + 00H]     - 0F 1F 40 00H
NOP DWORD ptr [EAX + EAX*1 + 00H] - 0F 1F 44 00 00H
66 NOP DWORD ptr [EAX + EAX*1 + 00H] - 0F 1F 44 00 00H
NOP DWORD ptr [EAX + 00000000H] - 0F 1F 80 00 00 00 00H
NOP DWORD ptr [EAX + EAX*1 + 00000000H] - 0F 1F 84 00 00 00 00 00H
66 NOP DWORD ptr [EAX + EAX*1 + 00000000H] - 0F 1F 84 00 00 00 00 00H
4 Byte NOPs

- 0x0F,0x18,0x60,0x70 is a 4 byte NOP
- Output from XED:

0F186070
ICLASS: NOP   CATEGORY: WIDENOP   EXTENSION: BASE   IFORM: NOP_MEMv_0F18r4   ISA_SET: PPRO
SHORT: nop dword ptr [eax+0x70]
Why Position Independent NOPs

- X86 instruction with different offsets could be decoded as different instructions

```
0F 18 60 70
```

```
0F
18
60
70
```
PIN (Position Independent NOP)

- Always NOP instructions even decoded at different offsets

```
00401EBF  db  0F3h
00401EC0  db  26h
00401EC1  db  0F2h
00401EC2  db  90h
00401EC3  nop
00401EC4  nop
00401EC5  nop
00401EC6  nop
```

```
00401EBF  repne nop
00401EC0  repne nop
00401EC1  repne nop
00401EC2  repne nop
00401EC3  nop
00401EC4  nop
00401EC5  nop
00401EC6  nop
```

```
F3  26  F2  90
```
How to create a 4 byte PIN

- Single byte NOP
  [0x90], [0x90], 0x90, 0x90
- 2 byte NOP
  [0xF2, [0x90], 0xF2, 0x90]
- 3 byte NOP
  [0x90], [0x26, [0xF2, 0x90]]
- 4 byte NOP
  [0xF3, [0x26, [0xF2, [0x90]]]]
2 Byte PINs

- Examples
  - 0x26, 0x90
  - 0x2E, 0x90
  - 0x36, 0x90
  - 0x3E, 0x90
  - 0x64, 0x90
  - 0x65, 0x90
  - 0x66, 0x90
  - 0x67, 0x90
  - 0xF2, 0x90
  - ...
3 Byte PINs

- Examples
  - 0x2E, 0x26, 0x90
  - 0x2E, 0x2E, 0x90
  - 0x2E, 0x36, 0x90
  - 0x2E, 0x3E, 0x90
  - 0x2E, 0x64, 0x90
  - 0x2E, 0x65, 0x90
  - 0x2E, 0x66, 0x90
  - 0x2E, 0x67, 0x90
  - 0x2E, 0xF2, 0x90
  - 0x36, 0x26, 0x90
  - ...

4 Byte PINs

Examples

- 0x2E, 0x2E, 0x26, 0x90
- 0x36, 0x2E, 0x26, 0x90
- 0x3E, 0x2E, 0x26, 0x90
- 0x64, 0x2E, 0x26, 0x90
- 0x65, 0x2E, 0x26, 0x90
- 0x66, 0x2E, 0x26, 0x90
- 0x67, 0x2E, 0x26, 0x90
- 0xF2, 0x2E, 0x26, 0x90
- ...
Thanks!

ldpatchguard@gmail.com
kangli@uga.edu
Reference

[1] Transparent Dynamic Instrumentation, Derek Bruening, Qin Zhao, Saman Amarasinghe, International Conference on Virtual Execution Environments (VEE-12), 2012
