BREAK OUT OF THE TRUMAN SHOW

ACTIVE DETECTION AND ESCAPE OF DYNAMIC BINARY INSTRUMENTATION

Ke Sun  wildsator@gmail.com
Xiaoning Li  ldpatchguard@gmail.com
Ya Ou  perfectno2015@gmail.com
About Us

- Xiaoning
  - Security Researcher

- Dr Ke Sun/Dr Ya Ou
  - Independent Security Researcher
Typical DBI’s Software Architecture

- Instrumentation Plugins
- Instrumentation Interface
- JIT Compiler
- Emulation Interface
- Code Cache
- OS/Hardware

Application
DBI Detections Talks

- **Pintools**

  CORE SECURITY
  Dynamic Binary Instrumentation Frameworks: I know you’re there spying on me
  Francisco Falcón – Nahuel Riva
  RECon 2012
  June 2012

- **DynamoRIO**

  Defeating the Transparency Features of Dynamic Binary Instrumentation
  The detection of DynamoRIO through introspection
  Xiaoning Li
  Kang Li
  ldpatchguard@gmail.com
  kangli@uga.edu

- **SafeMachine**

  malware needs love, too
  Martin Hron, Jakub Jermář
  AVAST Software, research
## Published DBI Detections Methodologies

<table>
<thead>
<tr>
<th>DBI Tool</th>
<th>Core Security @ RECON 2012</th>
<th>SafeMachine @ VB 2014</th>
<th>Xiaoning Li et al @ BH 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Detection Method</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pinvm</td>
<td>Code</td>
<td></td>
<td>Code</td>
</tr>
<tr>
<td>&amp; pintools</td>
<td>String</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>Exported Func</td>
<td></td>
<td>APIs</td>
</tr>
<tr>
<td></td>
<td>Section Names</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Handles</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ntdll</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hooks</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Fingerprint in Memory</strong></td>
<td>Execution Time</td>
<td></td>
<td>Peak Memory Usage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Handler Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max Open Handlers</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Real EIP Leak</strong></td>
<td>FNXSAVE, FNSAVE, FNSTENV</td>
<td>FNXSAVE, FNSAVE, FNSTENV</td>
<td></td>
</tr>
<tr>
<td>Interrupt (32-bit only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Page Permission</strong></td>
<td>Page Permission</td>
<td>Page Permission</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ZwAllocateVirtualMemory</td>
<td>PAGE_GUARD</td>
<td></td>
</tr>
<tr>
<td><strong>Parent process</strong></td>
<td>argv List of pin.exe</td>
<td>Parent Process Name</td>
<td>Parent Process Name</td>
</tr>
<tr>
<td></td>
<td>Parent Process Name</td>
<td></td>
<td>Customized Stack</td>
</tr>
<tr>
<td><strong>Stack</strong></td>
<td></td>
<td>Used Stack</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Setrlimit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Designed Exception Distance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Checksum Triggered Exception</td>
</tr>
<tr>
<td><strong>Implementation Bugs</strong></td>
<td>Bug in Emulating Instruction</td>
<td>SYSENTER</td>
<td></td>
</tr>
</tbody>
</table>
New Approaches to Detect DBI
New Detections

- Passive Detections with Gating Code
  - Unsupported Instructions
  - Unsupported Behaviors
- Active Detection
  - Xmode Code
  - Code Cache Detection
  - Thread Local Storage
  - Unexpected Context
Passive Detections with Gating Code

- Unsupported instructions
  - `Retf`
    - `-rdata:543A0A14` 00000021 C  Pin doesn't support FAR RET (IP
    - `-rdata:543A0A38` 0000000E C  on IRET (IP

C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\pin-2.14-7131
3-msvc12\windows>pin.exe -- ..\..\exe\64ret32.exe
CS selector = 33
E: Pin doesn't support FAR RET (IP 0x0004011c3) with transfer to different code
segment (from 0x0033 to 0x0000)

- Unsupported behaviors
  - Does not support mode switch in WoW64

C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\Xmode\BT\DynamoRIO-Win
dows-5.1.0-RC1\bin32\drrun.exe ..\..\..\exe\32_64_compcode.exe
CS selctor = **23**
Return value under32bit = 4
CS selector = **23**
Return value under64bit = 4

32_64_compcode.exe has stopped working
Windows is checking for a solution to the problem...
Active Detections with Xmode Code

- CPU mode is determined by the “L” bit in the segment descriptor of the code segment (CS).

**CS = 0023: 32-bit (L=0)**

**CS = 0033: 64-bit (L=1)**
Active Detections with Xmode Code

- Dynamic mode switch can be carried out by far branches to the corresponding segment
  - Far Jump
  - Far Call
  - Far Return
  - IRet

```assembly
db 0eah
dd Enter64bit_Ret  jmp far 0033: Enter64bit_Ret
db 033h
db 000h
```

Switch from 32-bit to 64-bit mode
Active Detections with Xmode Code

- Instruction compatibility
  - Compatible instructions
    - Same binary code has same meaning under 32-bit/64-bit mode
  - Incompatible instructions
    - Same binary code has different meaning under 32-bit/64-bit mode

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/</th>
<th>64-Bit</th>
<th>Compat/</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0+ rb ib</td>
<td>MOV r8, imm8</td>
<td>0I</td>
<td>Valid</td>
<td>Valid</td>
<td>Move imm8 to r8.</td>
</tr>
<tr>
<td>REX + B0+ rb ib</td>
<td>MOV r8**, imm8</td>
<td>0I</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move imm8 to r8.</td>
</tr>
<tr>
<td>B8+ rw lw</td>
<td>MOV r16, imm16</td>
<td>0I</td>
<td>Valid</td>
<td>Valid</td>
<td>Move imm16 to r16.</td>
</tr>
<tr>
<td>B8+ rd id</td>
<td>MOV r32, imm32</td>
<td>0I</td>
<td>Valid</td>
<td>Valid</td>
<td>Move imm32 to r32.</td>
</tr>
<tr>
<td>REX.W + B8+ rd io</td>
<td>MOV r64, imm64</td>
<td>0I</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move imm64 to r64.</td>
</tr>
<tr>
<td>C6 /0 ib</td>
<td>MOV r/m8, imm8</td>
<td>MI</td>
<td>Valid</td>
<td>Valid</td>
<td>Move imm8 to r/m8.</td>
</tr>
</tbody>
</table>

compatible code

incompatible code
Active Detections with Xmode Code

- Compatible instructions has exactly the same binary & disassembly under 32-bit and 64-bit mode, but still can have different results due to different stack frame size.

### 64-bit mode
```
test!Callback64bit: 00000000`00b215c3 8bc4
00000000`00b215c5 e800000000
00000000`00b215ca 8bdc
00000000`00b215cc 2bc3
```

- 64-bit mode

### 32-bit mode
```
test!Callback64bit: 001615c3 8bc4
001615c5 e800000000
001615ca 8bdc
001615cc 2bc3
```

- 32-bit mode

#### after code execution
- **eax = 8**

#### after code execution
- **eax = 4**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Comp/Compat Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B /r</td>
<td>MOV r32,r/m32</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m32 to r32.</td>
</tr>
<tr>
<td>68 cd</td>
<td>CALL rel32</td>
<td>M</td>
<td>Valid</td>
<td>Valid</td>
<td>Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.</td>
</tr>
</tbody>
</table>
Active Detections with Xmode Code

- Direct execution in command line

C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\xmode_detect\xmode_detect_bkup2\Release>test.exe
Local Variable Address: 2efdb0

Current CS Selector = 23
Current Stack Frame Size = 4

Current CS Selector = 33
Current Stack Frame Size = 8

- Executed under DBI tools (DynamoRIO)

C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DynamoRIO-Windows-6.0.0-6\bin32\drrun.exe ..\..\xmode_detect\xmode_detect_bkup2\Release\test.exe
Local Variable Address: 2bfc78

Current CS Selector = 23
Current Stack Frame Size = 4

DBI detected!
Active Detection with Code Cache

- Code Cache Signature
  - 0xfeedbeaf in Pin Code Cache

- Direct execution by command line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DBI\detect\codecatch\Release>ebxcatch.exe

Searching for PIN signature "feedbeaf"
Memory search completed, signature count: 0

No PIN Detected.
```

- Executed by Pin

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313-msvc12-windows>pin.exe -- ..\DBI\detect\codecatch\Release\ebxcatch.exe

Searching for PIN signature "feedbeaf"
Memory search completed, signature count: 87

PIN Detected!!
```
Active Detection with Code Cache

- Use predefined signature and memory search
  - Direct execution only 1 hit
  - Execution under DBI gives 2 hits: one in original PE image one in code cache

- Signature can be certain code or data

```asm
__asm{
    nop
    nop
    push eax
    pop eax
    nop
    nop
    push eax
    pop eax
}
```

Signature: 90 90 50 58

```asm
__asm{
    push ebx
    mov ebx, 0x12345678
    pop ebx
}
```

Signature: 78 56 34 12
Active Detection: Code Cache Detection

- Execute & Search

**Signature Function**

```c
void test()
{
    __asm{
        nop
        nop
        push eax
        pop eax
        nop
        nop
        push eax
        pop eax
        nop
        nop
        push eax
        pop eax
        nop
        nop
    }

    printf("\nsignature function executed.\n");
    return;
}
```

**Main Function**

```c
int sig_count = 0;
test();

for (int i = 0; i<0x80000; i++)
{
    data = (unsigned char*)(i * 0x1000);
    for (int j = 0; j<0xffff; j++)
    {
        data = (unsigned char*)(i * 0x1000 + j);
        __try{
            if (data[0] == 0x90 &&
                data[1] == 0x90 &&
                data[2] == 0x50 &&
                data[3] == 0x58)
            {
                printf("signature found @ 0x%x\n", data);
                sig_count++;

                break;
            }
        }
        __except (filter(GetExceptionCode(), GetExceptionInformation())){
            continue;
        }
    }
}
```
Active Detection: Code Cache Detection

- Direct execution by command line
  
  ```
  C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DBI\codecache\Release>codecache_detect.exe

  signature function executed.
  signature found @ 0x13c1038
  memory search completed, signature count 1
  ```

- Executed by Pin
  
  ```
  C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313-msvc12-windows>pin.exe -- ..\DBI\codecachesch\Release\codecache_detect.exe

  signature function executed.
  signature found @ 0x1161038
  signature found @ 0x18478b2
  memory search completed, signature count 2
  DBI Detected!!
  ```

- Executed by DynamoRIO
  
  ```
  C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DynamoRIO-Windows-6.0.0-6\bin32>drrun.exe ..\..\DBI\codecachesch\Release\codecache_detect.exe

  signature function executed.
  signature found @ 0x1301038
  signature found @ 0x17f47500
  memory search completed, signature count 2
  DBI Detected!!
  ```
Active Detection: Code Cache Detection

- Signature location in code cache can be confirmed to be RWE without calling memory APIs
- Executed by Pin

```text
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313-msvc12-windows\pin.exe -- ..\DBI\detect\codecachwt\Release\ebxcatch.exe

signature function executed.
signature found @ 0x1271038
signature found @ 0x18473da
memory search completed, signature count: 2

1st signature Location not RWE
2nd signature read from writing location: 0x12345678

2nd signature Location RWE, DBI detected!
```

- Executed by DynamoRIO

```text
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DynamoRIO-Windows-6.0.0-6\bin32\drrun.exe ..\..\DBI\detect\codecachwt\Release\ebxcatch.exe

signature function executed.
signature found @ 0x51038
signature found @ 0x1c707420
memory search completed, signature count: 2

1st signature Location not RWE
2nd signature read from writing location: 0x12345678

2nd signature Location RWE, DBI detected!
```
Active Detection with Thread Local Storage

- Thread Local Storage (TLS) is the method by which each thread in a given multithreaded process can allocate locations in which to store thread-specific data.

- Dynamically bound (run-time) thread-specific data is supported by way of the TLS API (TlsAlloc, TlsGetValue, TlsSetValue, and TlsFree).

- DBI tools use TLS to store tool-specific data, which can be detected by using TLS API: TlsGetValue

Thread Local Storage in Native App

- Executed by command line

```bash
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btoscape\DBI\tlsdetect\Release\TLS_detect.exe
TLS Slots:
[0]: 0x0
[1]: 0x0
[2]: 0x0
[3]: 0x0
[4]: 0x0
[5]: 0x0
[6]: 0x0
[7]: 0x0
[8]: 0x0
[9]: 0x0
[10]: 0x0
[11]: 0x0
[12]: 0x0
[13]: 0x0
[14]: 0x0
[15]: 0x0
[16]: 0x0
[17]: 0x0
[18]: 0x0
[19]: 0x0
[20]: 0x0
[21]: 0x0
[22]: 0x0
[23]: 0x0
[24]: 0x0
[25]: 0x0
[26]: 0x0
[27]: 0x0
[28]: 0x0
[29]: 0x0
[30]: 0x0
[31]: 0x0
[32]: 0x0
[33]: 0x0
[34]: 0x0
[35]: 0x0
[36]: 0x0
[37]: 0x0
[38]: 0x0
[39]: 0x0
[40]: 0x0
[41]: 0x0
[42]: 0x0
[43]: 0x0
[44]: 0x0
[45]: 0x0
[46]: 0x0
[47]: 0x0
[48]: 0x0
[49]: 0x0
[50]: 0x0
[51]: 0x0
[52]: 0x0
[53]: 0x0
[54]: 0x0
[55]: 0x0
[56]: 0x0
[57]: 0x0
[58]: 0x0
[59]: 0x0
[60]: 0x0
[61]: 0x0
[62]: 0x0
[63]: 0x0
```
Thread Local Storage in Pin Context

Executed by Pin

C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313-msuc12-windowns>pin.exe -- ..\DBI\tlsdetect\Release\TLS_detect.exe
TLS Slots:
[0]: 0x0
[1]: 0x90100
[2]: 0x16c0010
[3]: 0x0
[4]: 0x0
[5]: 0x0
[6]: 0x0
[7]: 0x0
[8]: 0x0
[9]: 0x0
[10]: 0x0

...  

[58]: 0x0
[59]: 0x0
[60]: 0x0
[61]: 0x0
[62]: 0x0
[63]: 0x0

DBI Detected!!

DBI tool = PIN!
Thread Local Storage in DynamoRIO Context

- Executed by DynamoRIO

```c
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DynamoRIO-Windows-6.0.0-6\bin32\drrun.exe ..\..\DBI\tlsdetect\Release\TLS_detect.exe

TLS Slots:
[0]: 0x0
[1]: 0x0
[2]: 0x0
[3]: 0x0
[4]: 0x0
[5]: 0x0
[6]: 0x0
...

[51]: 0x0
[52]: 0x0
[53]: 0x1dd49034
[54]: 0x0
[55]: 0x73ee3071
[56]: 0x37
[57]: 0x1dd1b480
[58]: 0x7f
[59]: 0x1dd91240
[60]: 0x7f
[61]: 0x1dd91680
[62]: 0x7f
[63]: 0x1dd91ac0

DBI Detected!!
DBI tool = DynamoRIO!
```
Active Detection with Pin-specific Context

- Pin JIT hides EBX from application usage in code cache

```
00c21c2f  90    nop
00c21c30  90    nop
00c21c31  50    push  eax
00c21c32  58    pop   eax
00c21c33  90    nop
00c21c34  90    nop
00c21c35  50    push  eax
00c21c36  58    pop   eax
00c21c37  90    nop
00c21c38  90    nop
00c21c39  50    push  eax
00c21c3a  58    pop   eax
00c21c3b  90    nop
00c21c3c  90    nop
00c21c3d  50    push  eax
00c21c3e  58    pop   eax
00c21c3f  50    push  eax
00c21c40  53    push  ebx
00c21c41  85c3   mov  eax,ebx
00c21c42  0f     pop   eax
00c21c43  89f4   mov  dword_ptr[ebp-0Ch],eax
```

```
02686ae1  90    nop
02686ae2  90    nop
02686ae3  50    push  eax
02686ae4  58    pop   eax
02686ae5  90    nop
02686ae6  90    nop
02686ae7  50    push  eax
02686ae8  58    pop   eax
02686ae9  90    nop
02686aee  90    nop
02686aeb  50    push  eax
02686aec  58    pop   eax
02686aed  90    nop
02686aef  50    push  eax
02686af0  58    pop   eax
02686af1  50    push  eax
02686af2  56    push  esi
02686af3  89f0   mov  eax,esi
02686af5  58    pop   eax
02686af6  8945f4  mov  dword_ptr[ebp-0Ch],eax
```
Active Detection with Pin-specific Context

- Pin use a specific location in memory for EBX backup
- Real EBX value in runtime control by Pin is the base address for registers’ backup location

```assembly
00df100c 90   nop
00df100d 50   push eax
00df100e 58   pop eax
00df100f 90   nop
00df1010 90   nop
00df1011 50   push eax
00df1012 58   pop eax
00df1013 90   nop
00df1014 90   nop
00df1015 50   push eax
00df1016 58   pop eax
00df1017 90   nop
00df1018 90   nop
00df1019 50   push eax
00df101a 8bc3 mov eax, ebx
00df101b 8945fc mov dword ptr [ebp-4], eax
00df101c 8955fc mov dword ptr [ebp-4], eax
00df101d 58   pop eax
00df101e ff75fc push dword ptr [ebp-4]
```

```assembly
eax=00000000 ebx=012c0080 ecx=6e9ce000 edx=00000000 esi=00000001 edi=00000000
eip=018573ec esp=000000f0 ebp=000004e4 iopl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246
```
Pin-specific Context in EBX

- Part of Code Cache Area

eax=ab0819af  ebx=01520080  ecx=00000000  edx=0044f5f4  esi=00000000  edi=0044f5ec
eip=01d09619  esp=0044f510  ebp=0044f5ec  iopl=0  nv up ei ng nz na pe nc
 cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b  efl=00000286
01d09619 90  nop
0:000> !address ebx

Usage:       <unknown>
Base Address: 01520000
End Address:  01930000
Region Size:  00410000  (  4.063 MB)
State:       00001000   MEM_COMMIT
Protect:     00000040   PAGE_EXECUTE_READWRITE
Type:        00020000   MEM_PRIVATE
Allocation Base: 01520000
Allocation Protect: 00000040   PAGE_EXECUTE_READWRITE

0:000> dd 01520000
01520000  feedback  00410000  00000000  00000000
01520010  00000000  00000000  00000000  00000000
01520020  00000000  00000000  00000000  00000000
01520030  00000000  00000000  00000000  00000000
01520040  00000000  00000000  00000000  00000000
01520050  00000000  00000000  00000000  00000000
01520060  00000000  00000000  00000000  00000000
01520070  00000000  00000000  00000000  00000000
Original Registers in Pin-specific Context

- ebx
- ebx+1Ch
- ebx+20h
- ebx+24h
- ebx+2Ch
- ebx+30h
- ebp backup location
- esp backup location
- ebx backup location
- ecx backup location
- eax backup location
Active Detection with EBX signature

- Detection Method:
  - Directly write signature code to EBX backup location \([EBX+24h]\)
  - Read EBX to see if signature can be found

Direct execution by command line

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DBI\ebxcatch4\Release>ebxcatch.exe
```

Write signature 0x1234 to PIN’s EBX backup location.
Read EBX value is 0x0
Signature not found in EBX.

Executed by Pin

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313-msvc12-windows>pin.exe -- ..\DBI\ebxcatch4\Release\ebxcatch.exe
```

Write signature 0x1234 to PIN’s EBX backup location.
Read EBX value is 0x1234
Signature found in EBX. PIN detected!!
DBI Escape Criteria
How to measure DBI escape

- Run banned instructions
- Run controlled instructions with controlled DBI context
- Run controlled instructions with DBI stack
- Run controlled instructions in DBI critical context
- Run controlled instructions hijacking DBI control flow
- Run controlled instructions tampering instrumentation client

All around how to break the limitation from DBI
DBI Escape Tracing with Hardware Features

- Performance Monitor Counter
Hardware Event with Native/DBI

- Indirect calls captured by PMI for the same binary with/without DBI

![Bar chart showing the number of indirect calls captured by PMI for different categories: Native, PIN, and DR. The chart indicates a significantly higher number of indirect calls captured by PIN compared to Native and DR.](image-url)
Indirect Calls with Hardware Events

Captured Indirect Calls for dummy_func

Without DBI

With DBI
Red dot is the Ind Call to dummy_func at 1880

Running Without DBI

Running under Pin

Escaped from Pin

Calling dummy_func under Pin without escape

1880 Address is absent
DynamoRIO

Running Without DBI

Red dot is the Ind Call to dummy_func at 1880

Running under DR

Escaped from DR

Calling dummy_func under DR without escape

1880 Address is absent
DBI Escape Approaches
Simplified Attack Surfaces

Application

Instrumentation Plugins

Instrumentation Interface

JIT Compiler

Emulation Interface

OS/Hardware

Code Cache
DBI Escape Approaches

- Code Cache Manipulation
  - Run, Modify, Run
  - Run, Modify Current Code Cache
- Critical Data Structures
  - Pin Stack
  - Pin/Pinclient callbacks
  - Pin/Pinclient Data
- Demo with retf and Xmode Code
DBI Escape Research in Past

Escaping DynamoRIO and Pin - or why it's a worse-than-you-think idea to run untrusted code or to input untrusted data

Before we begin, I want to clarify that both DynamoRIO and Pin are great tools that I use all the time. Dynamic Binary Modification is a very powerful technique in general. However, both implementations have a limitation which can have serious security implications for some use cases and which, as far as I can tell, is not documented in the user manuals. I got in touch with people involved in both projects and they've explained that they consider it low risk for the typical usage scenario and that fixing it would add performance overhead. This is a perfectly reasonable position, but I think this sort of low risk / high impact issue should be very well and visibly documented.

Background

It all started after I've watched this Black Hat talk on detecting execution under a DBM tool. That's interesting enough, but at the moment it's more or less a trivial problem. Now, escaping from the control of a DBI tool should be more challenging, right? Well, not so much.
Code Cache – Run/Modify/Run

- **Escape under Pin**
  - Extra codes executed while escape not counted by Pin

Extra codes not executed

Extra codes executed while escape

Extra codes executed when no escape
Code Cache – Run/Modify/Run

- Escape under DynamoRIO
  - Extra codes executed while escape not counted by DynamoRIO

Extra codes executed when no escape

Extra codes executed while escape

Extra codes not executed
Code Cache – Self Modify in Code Cache

- Search Signature & Modify NOPs
- Signature
- NOPs

Modify Code Cache

Modify Code Cache
DBI Key Context - Stack

- Pin has dedicated stack to run Pin’s code
- Jitted code in code cache uses OS allocated stack

```
01d30fe0 896320  mov   dword ptr [ebx+20h],esp
01d30fe3 8ba378050000 mov   esp,dword ptr [ebx+578h]
01d30fe9 ff93a4050000 call  dword ptr [ebx+5A4h]
```

```
Usage: <unknown>
Base Address: 01520000
End Address: 01930000
Region Size: 00410000 ( 4.063 MB)
State: 00001000 MEM_COMMIT
Protect: 00000040 PAGE_EXECUTE_READWRITE
Type: 00020000 MEM_PRIVATE
Allocation Base: 01520000
Allocation Protect: 00000040 PAGE_EXECUTE_READWRITE
```
DBI Key Context - TLS

- DBI has critical context point saved in TLS

```
[0]: 0x0
[1]: 0x500100
[2]: 0x1930010

0:000> dd 01930000
01930000 feedbeaf 000000f0 00000000 00000000
01930010 537bf50c 00000000 00000000 01520080
```

eax=ab0819af ebx=01520080 ecx=00000000 edx=0044f5f4 esi=00000000 edi=0044f5ec
eip=01d09619 esp=0044f510 ebp=0044f5ec iopl=0 ng up ei npx na pe nc

cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b
01d09619 90 nop
0:000> !address ebx

Usage: <unknown>
Base Address: 01520000
End Address: 01930000
Region Size: 00410000 (4.063 MB)
State: 00001000 MEM_COMMIT
Protect: 00000040 PAGE_EXECUTE_READWRITE
Type: 00020000 MEM_PRIVATE
Allocation Base: 01520000
Allocation Protect: 00000040 PAGE_EXECUTE_READWRITE
Pin/Pinclient Critical Sections

- .charmve section in Pin Dll

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>End</th>
<th>R</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>54001000</td>
<td>5437A000</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.idata</td>
<td>5437A000</td>
<td>5437A20C</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.rdata</td>
<td>544E0000</td>
<td>5466D000</td>
<td>R</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>.data</td>
<td>5466D000</td>
<td>5466E000</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.charmve</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- Both .charmve and .pinclie sections in Pintool Plugin

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>End</th>
<th>R</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>55001000</td>
<td>552E6000</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.idata</td>
<td>552E6000</td>
<td>552E6210</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.rdata</td>
<td>552E6210</td>
<td>55420000</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.data</td>
<td>55420000</td>
<td>5544C000</td>
<td>R</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>.pinclie</td>
<td>5544C000</td>
<td>5544D000</td>
<td>R</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>.charmve</td>
<td>5544D000</td>
<td>5544E000</td>
<td>R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pin Callbacks/Data

call BrokerClient

; Exported entry 1. BrokerClient

; SUBROUTINE ====

public BrokerClient

BrokerClient proc near

mov eax, dword_54665970
ret

.endp

.data:54665970 dword_54665970 dd ?
.data:54665970
data:54665974 align 8
Pinclient Callbacks/Data

- PinClient Callbacks can be addressed as data structure from memory

```assembly
loc_55040E66:
  push  eax
  call  sub_550B8590

; CODE XREF: sub_55040D90+2C↑j
; sub_55040D90+38↑j
  call  sub_550A7F20
  movzx  eax, al
  push  eax
  call  ds:dword_5544C018
  add  esp, 4
```
Escape under Pin

- 32-bit / 64-bit mode switch can be carried out after escape

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\pin-2.14-71313\msvc12-windows>pin.exe -- ..\DBI\escape\escape_xmode\Release\ebxcatch.exe
test called
sig: 0x222000
sig: 0x225000
sig: 0x23f000
sig: 0x374000
sig: 0x10c0000
signature: 0x10c10d3
signature: 0x1847523
mem search completed, signature count:2
argc = 1
Running under DBI!

escaped!
Array a[10] Base Address = 18feb4
Current CS Selector = 23
Current Stack Frame Size = 4

Array a[10] Base Address = 18feb4
Current CS Selector = 33
Current Stack Frame Size = 8
```

test called
Escape with Xmode Code

- Escape under DynamoRIO
  - 32-bit / 64-bit mode switch can be carried out after escape

```
C:\Users\Wild Sator\Documents\Visual Studio 2013\Projects\btescape\DynamoRIO-Windows-6.0.0-6\bin32>drrun.exe ..\..\DBI\escape\escape_xmode\Release\ebxcatch.exe
test called
sig: 0xd2000
sig: 0xd5000
sig: 0xef000
sig: 0x2c0000
signature: 0x2c10d3
signature: 0x21ec7433
mem search completed, signature count:2
argc = 1
Running under DBI!

escaped!
Array a[10] Base Address = 49f760
Current CS Selctor = 23
Current Stack Frame Size = 4

Array a[10] Base Address = 49f760
Current CS Selctor = 33
Current Stack Frame Size = 8
test called
```
Demo
Negative Impacts on Exploit Defense
Create New Attack Surfaces

- Code cache provides perfect place for shell code with full memory read/write
- DBI escape can be easily applied by exploit to hijack control flow and activate exploit/shell code

Defend old exploits, but make new exploit easier!
Summary

- Disclosed New Detection Methodologies
- Discussed DBI Escape Criteria
- In-depth Discussion of DBI Escape with Different Ways
  - Tampering Code Cache
  - Tampering Critical DBI Contexts/Callbacks/Data
- DBI is a powerful tool to defend existing exploits, but it also opens a big surface for new exploit utilizing RWE code cache
Reference

- Dynamic Binary Instrumentation Frameworks: I know you're there spying on me, Francisco Falcón / Nahuel Riva, RECon 2012
- SafeMachine malware needs love, too, Martin Hron / Jakub Jermář, VB 2014
- Defeating the Transparency Features of Dynamic Binary Instrumentation, Xiaoning Li / Kang Li, BlackHat USA 2014
- Obfuscation to Defeat Static Analysis Using Cross-mode Coding, Ke Sun / Xiaoning Li, Source Seattle 2015
- https://github.com/lgeek/dynamorio_pin_escape
- http://www.dynamorio.org/
Thanks!

wildsator@gmail.com
ldpatchguard@gmail.com
perfectno2015@gmail.com