Dynamic Binary Instrumentation Frameworks: I know you're there spying on me

Francisco Falcón – Nahuel Riva

RECon 2012
June 2012
Agenda
Agenda

• Who are we?
• Motivations
• What is Dynamic Binary Instrumentation?
  • What is Pin?
  • How does Pin work?
• Anti-debug and Anti-VM related work
• Anti-instrumentation techniques
• Presentation of eXait
• Applications of our research
• Future work
• Contact info
Who are we?
Who are we?

• We are exploit writers in the Exploit Writers Team of Core Security.
• We have discovered vulnerabilities in software of some major companies (CA, Adobe, HP, Novell, Oracle, IBM, Google).
• We like low-level stuff, like doing kernel exploitation, assembly programming, breaking software protections, etc.
• This is our first talk in a conference!
• We are from small towns in Argentina.
Who are we?

Nahuel is from the World ‘s Capital City of Asado!
Who are we?

Francisco is from a county that looks like the head of a man!
Motivations for our work
Motivations

• Dynamic Binary Instrumentation is becoming more popular.
  
  • Covert debugging (Saffron - Danny Quist – BH USA 2007/Defcon 15)
  • Automatic Unpacking (Piotr Bania - 2009, Ricardo J. Rodriguez - 2012)
  • Shellcode detection (Sebastian Porst – Zynamics - 2010)
  • Taint analysis
  • Instruction tracing
  • Self-modifying code analysis (Tarte Tatin Tools - Daniel Reynaud)
  • Exploitation techniques mitigations (Richard Johnson – Snort 2012)
Motivations

• Dynamic Binary Instrumentation is becoming more popular.
  
  • Light and Dark side of Code Instrumentation - Dmitriy Evdokimov - ConFidEncE 2012
  • Hacking Using Dynamic Binary Instrumentation - Gal Diskin - HITB 2012 AMS
  • Improving Software Security with Dynamic Binary Instrumentation - Richard Johnson - InfoSec Southwest 2012
  • Improvements in the unpacking process using DBI techniques - Ricardo J. Rodriguez - RootedCon 2012
  • Shellcode analysis using dynamic binary instrumentation - Daniel Radu and Bruce Dang - CARO 2011
  • Vulnerability Analysis and Practical Data Flow Analysis & Visualization - Jeong Wook Oh - CanSecWest 2012
Motivations

• If this trend continues, we think that eventually anti-instrumentation techniques will arise.
• Apparently, there isn’t any comprehensive public documentation on anti-instrumentation techniques.
What is Dynamic Binary Instrumentation?
What is Instrumentation?

It’s a technique to analyze and modify the behavior of a program by adding code to it.

It can be done:
- At the source code level
- At the **binary code** level

In turn, it can be:
- **Static**
- **Dynamic**
What is Dynamic Binary Instrumentation?

It’s a technique to analyze and modify the behavior of a binary program by injecting arbitrary code at arbitrary places while it is executing.
What is Pin?
What is Pin?

• It’s the Intel’s Dynamic Binary Instrumentation Framework.

• It works on Windows, Linux and Mac OS X.

• It works on x86, amd64, Itanium and ARM (discontinued).

• Its API allows to inject C/C++ arbitrary code.
How does Pin work?
How does Pin work?

• Pin is a command line tool:

  • `pin.bat -t pintool.dll [pintool args] -- program.exe [program args]`

  • `pin.bat -pid <program pid> -t pintool.dll [pintool args]`
How does Pin work?

- Pin main components:
  - Pin.exe
  - Pinvm.dll

- The code you write to instrument programs using the Pin API is compiled into pintools
How does Pin work?

- JIT compiler.
  - Input: binary code
  - Output: equivalent code with introspection code
  - The code is generated only when it is needed

- The only code that is executed is the code generated by the JIT compiler.

- The original code remains in memory just as a reference but it is never executed.
Anti-debug and Anti-VM related work
Anti-debug and Anti-VM related work

- Anti-debug techniques papers series by Peter Ferrie ([http://pferrie.host22.com/](http://pferrie.host22.com/)).

- Anti-VM techniques papers by Peter Ferrie (same link as above).

- Dan Upton – Detection and Subversion Of Virtual Machines ([http://www.cs.virginia.edu/~dsu9w/upton06detection.pdf](http://www.cs.virginia.edu/~dsu9w/upton06detection.pdf)).
Anti-debug and Anti-VM related work

- Red pill – (Joanna Rutkowska).

- On the Cutting Edge: Thwarting Virtual Machine Detection (Tom Liston – Ed Skoudis
Anti-instrumentation techniques
Anti-instrumentation techniques

- Code and data fingerprinting of pinvm.dll
- PE characteristics fingerprint
- Handles inspection
- Time detection
- Pin’s JIT compiler code fingerprint
- Real EIP value
- Misc techniques
Anti-instrumentation techniques – Fingerprinting pinvm.dll

• Code and data fingerprinting of pinvm.dll
  • Detect by searching string patterns
  • Detect by code patterns
Fingerprinting pinvm.dll – Detect by string patterns

- Detect by string patterns
  - "@CHARM-VERSION: $Id:"
  - "build\Source\pin\internal-include-windows-ia32\bigarray.H"
  - "LEVEL_BASE::ARRAYBASE::SetTotal"
  - "Source\pin\base\bigarray.cpp"
Fingerprinting pinvm.dll – Detect by code patterns

• Detect by code patterns (pattern 1)

5418D4A6 897424 04  MOV DWORD PTR SS:[ESP+4], ESI
5418D4AA 895C24 10  MOV DWORD PTR SS:[ESP+10], EBX
5418D4AE 895424 14  MOV DWORD PTR SS:[ESP+14], EDX
5418D4B2 894C24 18  MOV DWORD PTR SS:[ESP+18], ECX
5418D4B6 894424 1C  MOV DWORD PTR SS:[ESP+1C], EAX
5418D4BA 33C0   XOR EAX, EAX
5418D4BC 894424 20  MOV DWORD PTR SS:[ESP+20], EAX
5418D4C0 8C4C24 20  MOV WORD PTR SS:[ESP+20], CS
5418D4C4 894424 28  MOV DWORD PTR SS:[ESP+28], EAX
5418D4C8 8C5C24 28  MOV WORD PTR SS:[ESP+28], DS
5418D4CC 894424 24  MOV DWORD PTR SS:[ESP+24], EAX
5418D4D0 8C5424 24  MOV WORD PTR SS:[ESP+24], SS
5418D4D4 894424 2C  MOV DWORD PTR SS:[ESP+2C], EAX
5418D4D8 8C4424 2C  MOV WORD PTR SS:[ESP+2C], ES
5418D4DC 894424 30  MOV DWORD PTR SS:[ESP+30], EAX
5418D4E0 8C6424 30  MOV WORD PTR SS:[ESP+30], FS
5418D4E4 894424 34  MOV DWORD PTR SS:[ESP+34], EAX
5418D4E8 8C6C24 34  MOV WORD PTR SS:[ESP+34], GS
Fingerprinting pinvm.dll – Detect by code patterns

• Detect by code patterns (pattern 2)

```
01750110  CD  00    INT  0
01750112  E9  0B080000  JMP 01750922
01750117  90  NOP
01750118  CD  01    INT  1
0175011A  E9  03080000  JMP 01750922
0175011F  90  NOP
01750120  CD  02    INT  2
01750122  E9  FB070000  JMP 01750922
01750127  90  NOP
01750128  CD  03    INT  3
0175012A  E9  F3070000  JMP 01750922
0175012F  90  NOP
01750130  CD  04    INT  4
01750132  E9  EB070000  JMP 01750922
01750137  90  NOP
01750138  CD  05    INT  5
0175013A  E9  E3070000  JMP 01750922
[...]
It continues until INT FF
```
Anti-instrumentation techniques – Detect by PE characteristics

- Detect by PE characteristics
  - Detect by pinvm.dll presence
  - Detect by pinvm exported functions
  - Detect by pintools exported functions
  - Detect by sections names
Detect by PE characteristics – Detect by pinvm.dll presence

• Detect by pinvm.dll presence
Detect by PE characteristics – Detect by pinvm exported functions

- Detect by pinvm.dll exported functions
  - PinWinMain
  - CharmVersionC

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Function RVA</th>
<th>Name Ordinal</th>
<th>Name RVA</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nFunctions)</td>
<td>Dword</td>
<td>Word</td>
<td>Dword</td>
<td>szAnsi</td>
</tr>
<tr>
<td>00000001</td>
<td>00019980</td>
<td>0000</td>
<td>003A041C</td>
<td>CharmVersionC</td>
</tr>
<tr>
<td>00000002</td>
<td>001D7430</td>
<td>0001</td>
<td>003A042A</td>
<td>CrtEnableThreadCallbacks</td>
</tr>
<tr>
<td>00000003</td>
<td>001D7370</td>
<td>0002</td>
<td>003A0443</td>
<td>DeleteCriticalSection</td>
</tr>
<tr>
<td>00000004</td>
<td>001D7080</td>
<td>0003</td>
<td>003A0459</td>
<td>FlsAlloc</td>
</tr>
<tr>
<td>00000005</td>
<td>001D7120</td>
<td>0004</td>
<td>003A0462</td>
<td>FlsFree</td>
</tr>
<tr>
<td>00000006</td>
<td>001D71F0</td>
<td>0005</td>
<td>003A046A</td>
<td>FlsGetValue</td>
</tr>
<tr>
<td>00000007</td>
<td>001D70D0</td>
<td>0006</td>
<td>003A0476</td>
<td>FlsSetValue</td>
</tr>
<tr>
<td>00000008</td>
<td>0002C870</td>
<td>0007</td>
<td>003A0482</td>
<td>GetIpcClientData</td>
</tr>
<tr>
<td>00000009</td>
<td>001D6DB0</td>
<td>0008</td>
<td>003A0493</td>
<td>GetModuleHandleA</td>
</tr>
<tr>
<td>0000000A</td>
<td>001D6E60</td>
<td>0009</td>
<td>003A04A4</td>
<td>GetModuleHandleW</td>
</tr>
<tr>
<td>0000000B</td>
<td>001D6F10</td>
<td>000A</td>
<td>003A04B5</td>
<td>GetProcAddress</td>
</tr>
<tr>
<td>0000000C</td>
<td>001D72F0</td>
<td>000B</td>
<td>003A04C4</td>
<td>InitializeCriticalSection</td>
</tr>
<tr>
<td>0000000D</td>
<td>001D7260</td>
<td>000C</td>
<td>003A04DE</td>
<td>InitializeCriticalSectionAndSpinCount</td>
</tr>
<tr>
<td>0000000E</td>
<td>003A0513</td>
<td>000D</td>
<td>003A0504</td>
<td>NativeTlsAlloc</td>
</tr>
</tbody>
</table>
Detect by PE characteristics – Detect by pintools exported functions

- Detect by pintools exported functions
  - CharmVersionC
  - ClientIntC
Detect by PE characteristics – Detect by sections names

• Detect by sections names
  
  • Pintools sections
    • .pinclie
    • .charmve
  
  • Pinvm sections
    • .charmve

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Size</th>
<th>Virtual Address</th>
<th>Raw Size</th>
<th>Raw Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte[8]</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
</tr>
<tr>
<td>.text</td>
<td>002791CC</td>
<td>00001000</td>
<td>00279200</td>
<td>00000400</td>
</tr>
<tr>
<td>.rdata</td>
<td>00085DF7</td>
<td>00278000</td>
<td>00085E00</td>
<td>00279600</td>
</tr>
<tr>
<td>.data</td>
<td>0002541C</td>
<td>00301000</td>
<td>00002400</td>
<td>002FF400</td>
</tr>
<tr>
<td>.pinclie</td>
<td>00000380</td>
<td>00327000</td>
<td>00000400</td>
<td>00301800</td>
</tr>
<tr>
<td>.charmve</td>
<td>00000043</td>
<td>00328000</td>
<td>00000200</td>
<td>00301C00</td>
</tr>
<tr>
<td>.reloc</td>
<td>00019878</td>
<td>00329000</td>
<td>00019A00</td>
<td>00301E00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Size</th>
<th>Virtual Address</th>
<th>Raw Size</th>
<th>Raw Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte[8]</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
</tr>
<tr>
<td>.text</td>
<td>002E1B3E</td>
<td>00001000</td>
<td>002E1C00</td>
<td>00000400</td>
</tr>
<tr>
<td>.rdata</td>
<td>000BD5F7</td>
<td>002E3000</td>
<td>000BD600</td>
<td>002E2000</td>
</tr>
<tr>
<td>.data</td>
<td>000E7EE4</td>
<td>003A1000</td>
<td>00002E00</td>
<td>0039F600</td>
</tr>
<tr>
<td>.charmve</td>
<td>00000043</td>
<td>00489000</td>
<td>00000200</td>
<td>003A2400</td>
</tr>
<tr>
<td>.reloc</td>
<td>0002A498</td>
<td>0048A000</td>
<td>0002A600</td>
<td>003A2600</td>
</tr>
</tbody>
</table>
Anti-instrumentation techniques – Handles Inspection

- Handles inspection
  - Detect by Event handles
  - Detect by Section handles
  - Detect by Process handles
Handles inspection – Detect Event handles

- These objects are used by Pin for IPC (Inter Process Communication)

<table>
<thead>
<tr>
<th>Event</th>
<th>\Sessions\1\BaseNamedObjects\PIN_IPC_EventAckSetByClient_0x958_0x1484_0x3f587d5766fa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>\Sessions\1\BaseNamedObjects\PIN_IPC_EventSetByServer_0x958_0x1484_0x3f587d5766fa</td>
</tr>
<tr>
<td>Event</td>
<td>\Sessions\1\BaseNamedObjects\PIN_IPC_EventSetByClient_0x958_0x1484_0x3f587d5766fa</td>
</tr>
<tr>
<td>Event</td>
<td>\Sessions\1\BaseNamedObjects\PIN_IPC_EventAckSetByServer_0x958_0x1484_0x3f587d5766fa</td>
</tr>
</tbody>
</table>
Handles inspection – Detect by Section handles

- These objects are used by Pin for IPC (Inter Process Communication)

<table>
<thead>
<tr>
<th>Section</th>
<th>\Sessions\1\BaseNamedObjects\PIN\IPC\FileSentByServer 0x958 0x1484 0x3f587d5766fa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section</td>
<td>\Sessions\1\BaseNamedObjects\PIN\IPC\FileSentByClient 0x958 0x1484 0x3f587d5766fa</td>
</tr>
</tbody>
</table>
**Handles inspection – Detect by Process handles**

<table>
<thead>
<tr>
<th>Process</th>
<th>Handles</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd.exe</td>
<td>4864</td>
</tr>
<tr>
<td>pin.exe</td>
<td>3708</td>
</tr>
<tr>
<td>calc.exe</td>
<td>2392</td>
</tr>
<tr>
<td>pin.exe</td>
<td>6108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process</th>
<th>Handles</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin.exe(6108)</td>
<td></td>
</tr>
</tbody>
</table>
Anti-instrumentation techniques – Detect by execution delay

- Detect time variations
  - Detect Pin’s overhead
Detect by execution delay – Time variations

- Detect execution delay introduced by Pin

```c
printf("HMODULE: %x\n", LoadLibrary("user32.dll"));
printf("HMODULE: %x\n", LoadLibrary("ntmarta.dll"));
printf("HMODULE: %x\n", LoadLibrary("gdi32.dll"));
printf("HMODULE: %x\n", LoadLibrary("advapi32.dll"));
printf("HMODULE: %x\n", LoadLibrary("comctl32.dll"));
printf("HMODULE: %x\n", LoadLibrary("comdlg32.dll"));
printf("HMODULE: %x\n", LoadLibrary("crypt32.dll"));
printf("HMODULE: %x\n", LoadLibrary("dbghelp.dll"));
printf("HMODULE: %x\n", LoadLibrary("ole32.dll"));
printf("HMODULE: %x\n", LoadLibrary("urlmon.dll"));
```

- Non-instrumented execution ≈ 15 to 30 milliseconds.
- Instrumented execution ≈ 1200 to 1500 milliseconds.
- Depends on your machine’s power.
Anti-instrumentation techniques – JIT compiler detection

• Detect the JIT compiler
  • Detect ntdll.dll hooks
  • Detect by page permissions
  • Detect by common API calls
JIT compiler detection – Detect by common API calls

- Detect by ntdll.dll hooks

- **KiUserApcDispatcher**
  - $E9 C367BBDC
  - JMP pinvm.541C6800

- **KiUserCallbackDispatcher**
  - $E9 FB66BBDC
  - JMP pinvm.541C67EC

- **KiUserExceptionDispatcher**
  - $E9 EF66BBDC
  - JMP pinvm.541C6828

- **LdrInitializeThunk**
  - $E9 C6C9B8DC
  - JMP pinvm.541C6814
JIT compiler detection – Detect by page permissions

- Detect by page permissions

- This technique may not work with programs which already have a JIT compiler.
JIT compiler detection – Detect common API calls

- Detect by common API calls
  - ZwAllocateVirtualMemory
    - AllocationType = MEM_COMMIT | MEM_RESERVE
    - Protect = PAGE_EXECUTE_READWRITE
- This technique may not work with programs which already have a JIT compiler.
Anti-instrumentation techniques – Real EIP value

• Real EIP value

(Remember that: the original code remains in memory just as a reference but it is never executed)

• Detect by FSTENV
• Detect by FSAVE
• Detect by FXSAVE

• Detect by Interruptions
Real EIP value – Detect by FSTENV

__asm
{
   fldz;
    fstenv [esp-0x1c];
    mov eax, [esp-0x10];
    mov RealEIP, eax;
}

- FSTENV saves the FPU environment, which includes the instruction pointer.
- Alternative: FNSTENV
Real EIP value – Detect by FSTENV

Non-instrumented

0x00401000 fldz
0x00401002 fstenv [esp-0x1c]
0x00401006 mov eax, [esp-0x10]
0x0040100a ...

Instrumented

0x00401000 fldz
0x00401002 fstenv [esp-0x1c]
0x00401006 mov eax, [esp-0x10]
0x0040100a ...

0x00521000 fldz
0x00521002 fstenv [esp-0x1c]
0x00521006 mov eax, [esp-0x10]
0x0052100a ...
Real EIP value – Detect by FSTENV

VirtualQuery((LPCVOID)RealEIP, &mbi, sizeof(mbi));

if((DWORD)hGlobalModule == (DWORD)mbi.AllocationBase)
    return NOTDETECTED;
else
    return DETECTED;
Real EIP value – Detect by FSAVE

```asm
__asm
{
    FLDZ
    FSAVE (108-BYTE) PTR SS:[ESP-6C]
    MOV EAX,DWORD PTR SS:[ESP-60]
}
```

- FSAVE stores the FPU state (FPU environment + register stack).
- Alternative: FNSAVE
Real EIP value – Detect by FXSAVE

```asm
__asm
{
    LEA EAX, [ESP-0x20C];
    AND EAX, 0xFFFFFFFF0;
    FLDZ;
    FXSAVE [EAX];
    MOV EAX, [EAX+8];
}

• FXSAVE writes the state of the x87 FPU + MMX registers + SSE registers.
Real EIP value – Detect by Interruptions

```asm
__asm {
    xor eax, eax;
    xor edx, edx;
    int 0x2e;
    nop;
    mov RealEIP, edx;
}
```

- This technique was documented by the corkami project ([http://code.google.com/p/corkami/](http://code.google.com/p/corkami/)).
- This technique only works on 32 bits systems (Windows XP/Vista/Seven).
- Does not work on WoW64 (it raises an exception).
Anti-instrumentation techniques - Misc techniques

• Misc techniques

  • Detect by Argv
  • Detect by parent process
  • Detect by SYSENTER emulation
Misc techniques – Detect by argv

- Detect by argv

We get the argv array of our parent process by searching within the memory of our process.
Misc techniques – Detect by argv

- Detect by argv

```
000305C8 000305F0 ASCII "C:\pin\ia32\bin\pin.exe"
000305CC 00030610 ASCII "-p32"
000305D0 00030618 ASCII "C:\pin\ia32\bin\pin.exe"
000305D4 00030638 ASCII "-p64"
000305D8 00030640 ASCII "C:\pin\intel64\bin\pin.exe"
000305DC 00030660 ASCII "-t"
000305E0 00030668 ASCII "tools\SimpleExamples\obj-ia32\opcodemix.dll"
000305E4 000306A0 ASCII "--"
000305E8 000306A8 ASCII "C:\dummy.exe"
000305EC FEEFEFEE
```
Misc techniques – Detect by parent process

- Detect by parent process

<table>
<thead>
<tr>
<th>Program</th>
<th>PID</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd.exe</td>
<td>4864</td>
<td>TRAVESTI\nriva</td>
</tr>
<tr>
<td>pin.exe</td>
<td>3708</td>
<td>TRAVESTI\nriva</td>
</tr>
<tr>
<td>calc.exe</td>
<td>2392</td>
<td>TRAVESTI\nriva</td>
</tr>
<tr>
<td>pin.exe</td>
<td>6108</td>
<td>TRAVESTI\nriva</td>
</tr>
</tbody>
</table>

- Will not work when instrumenting a process by attaching it.
Detect by SYSENTER emulation

Eloi Vanderbeken in 2011 found a bug in the way Pin emulates the SYSENTER instruction.

- Normal execution ring0 – ring3: the execution continues in ntdll!KiFastSystemCallRet
- Instrumented execution ring0 – ring3: continues in the instruction following the SYSENTER
- The last affected version of Pin is build 39599, Feb 28, 2011
- Discussion of this bug can be found here:
  http://tech.groups.yahoo.com/group/pinheads/message/6363
Misc techniques – Detect by SYSENTER emulation

```assembly
__asm {
    //invalid syscall
    mov eax, 0x42424242;
    push retaddress;
    mov edx, esp;
    //Sysenter
    _emit 0x0F;
    _emit 0x34;
    //if execution reaches here, it means that it's being instrumented
    mov detected, DETECTED;
    jmp endasm;
    retaddress:
    //normal execution should continue here after the sysenter
    mov detected, NOTDETECTED;
    endasm:
}
```
Keep in mind that ...

• All the presented techniques have different levels of reliability.
• So, you may combine them to be more accurate when detecting Pin.

Don’t forget!
eXait – eXtensible Anti-Instrumentation Tester
eXait – eXtensible Anti-Instrumentation Tester

- There are benchmark-like tools to test:
  - Anti-Virtualization techniques (ScoopyNG - Trapkit)
eXait – eXtensible Anti-Instrumentation Tester

- There are benchmark-like tools to test:
  - Anti-Debugging techniques (xADT- Shub Nigurrath)
eXait – eXtensible Anti-Instrumentation Tester

- eXait is the eXtensible Anti-Instrumentation Tester tool.
- It was written in C using Visual C++ Express 2008.
- It has a plugin architecture.
- It is open-source code (BSD license).
- It has more than 15 plugins to test all the techniques presented in this talk.
### eXait – eXtensible Anti-Instrumentation Tester

![eXait v1.0 interface](image)

<table>
<thead>
<tr>
<th>Enable</th>
<th>Plugin name</th>
<th>Result</th>
<th>Status</th>
<th>Plugin description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Detect Pin by time</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin tries to detect Pin by checking execution time</td>
</tr>
<tr>
<td></td>
<td>Detect Pin by common API calls</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin hooks ZwAllocateVirtualMemory to check for Pin</td>
</tr>
<tr>
<td></td>
<td>Detects Pin argv</td>
<td>NaN</td>
<td>NaN</td>
<td>Detects Pin by searching for the original argv variable</td>
</tr>
<tr>
<td></td>
<td>Detect pin by searching code patterns</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin implements a search function to search for code patterns</td>
</tr>
<tr>
<td></td>
<td>Detect int 2c</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin detects Pin by executing the INT 0x2E</td>
</tr>
<tr>
<td></td>
<td>Detect Pin ntdll.dll pointers</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin looks for four pointers to ntdll.dll functions</td>
</tr>
<tr>
<td></td>
<td>Detect pin by page permissions</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin looks for memory pages with EXECUT Permission</td>
</tr>
<tr>
<td></td>
<td>Detect pin by searching a code pattern</td>
<td>. NaN</td>
<td>NaN</td>
<td>This plugin searches for a code pattern usually located in the process map</td>
</tr>
<tr>
<td></td>
<td>Detect pin by searching patterns</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin implements a search function to search for patterns</td>
</tr>
<tr>
<td></td>
<td>Detect pin by searching PE section names</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin detects Pin by searching PE section names</td>
</tr>
<tr>
<td></td>
<td>Detect pin by syseinter</td>
<td>NaN</td>
<td>NaN</td>
<td>Detects Pin by executing a syseinter instruction</td>
</tr>
<tr>
<td></td>
<td>Detect pin by EIP</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin determines the address in which its code starts</td>
</tr>
<tr>
<td></td>
<td>Detect NTDLL hooks</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin looks for hooks that Pin usually sets in ntdll.dll</td>
</tr>
<tr>
<td></td>
<td>Detect parent process</td>
<td>NaN</td>
<td>NaN</td>
<td>This plugin checks the name of the parent process</td>
</tr>
<tr>
<td></td>
<td>Detect pintoos Exports</td>
<td>NaN</td>
<td>NaN</td>
<td>Looks for functions exported by the pintoos</td>
</tr>
<tr>
<td></td>
<td>Detect pinvm Dll</td>
<td>NaN</td>
<td>NaN</td>
<td>Looks for the pinvm.dll into the list of loaded modules</td>
</tr>
</tbody>
</table>

**Buttons:**
- Select All
- Start Test
- Clear
- Refresh
- About
eXait – eXtensible Anti-Instrumentation Tester

- eXait comes in two flavors: console and GUI.

- You can write your own plugins for eXait, check the project wiki.

- We are waiting for your contribution.
eXait – eXtensible Anti-Instrumentation Tester

• eXait can be downloaded from:

http://corelabs.coresecurity.com
Applications of our research
Applications of our research

• Each one of the discussed techniques can be included in any software that wants to protect itself against dynamic binary analysis:
  
  • Packers
  • Malware
  • Shellcodes?
Future work
Future work

• Extend our research to other DBI frameworks (DynamoRIO, Valgrind, DynInst, ERESI, Fjalar).

• Further our research to other platforms and architectures.

• Find new anti-instrumentation techniques (obvious!!!).
Future work

• Create a library for pintools to bypass anti-instrumentation techniques.

• Things to discuss in this field:
  • How to implement it as generic as possible?
  • Is this a never ending story? Who wins, if anyone?
It’s show time!. Demo.
Acknowledgments & Greetings
Acknowledgments & Greetings

• Fernando Russ
  • for coordinating our research and feedback

• Gal Diskin
  • for his feedback about the presentation

• Ariel Futoransky
  • for his ideas for further research

• RECon Organizers
Contact info
Contact info

Francisco Falcón

@fdfalcon
ffalcon@coresecurity.com

Nahuel Riva

@crackinglandia
nriva@coresecurity.com
Questions?
Thank you.