How to Grow a TREE from CBASS

Interactive Binary Analysis for Security Professionals

Lixin (Nathan) Li, Xing Li, Loc Nguyen, James E. Just
Outline

• Background
• Interactive Binary Analysis with TREE and CBASS
• Demonstrations
• Conclusions
Interactive Binary Analysis

• Automated binary analyses useful for certain tasks (e.g., finding crashes)

• Many binary analyses can’t be automated

• Expert experience and heuristics are still key to binary analyses
Benefits of Interactive Binary Analysis

• Applicable to many security problems

• Our tools increase productivity in:
  – Finding vulnerabilities
  – Analyzing root causes
  – Exploitability and risk assessment
Interactive Analysis Like Connecting Dots

What’s in the dots?
Our Tools are Designed to Help

- Fix the Dots
- Connect the Dots
- Explore New Dots
What Do Our Tools Do?

Fix the Dots

Connect the Dots

Explore New Dots

TREE
Replay & Taint Analysis
*Tainted-enabled Reverse Engineering Environment*

CBASS
Symbolic Execution
*Cross-platform Binary Automated Symbolic execution System*
Gaps between Research and Interactive Binary Analysis

• Existing research does not support interactive binary analysis
  – No practical tools
  – No uniform trace collection tools
  – No unified Instruction Set Architecture (ISA) - independent analysis tools
Bringing Proven Research Techniques to Interactive Binary Analysis

• Our tools use dynamic, trace-based, offline analysis approach
  – Interactive binary analysis [1]
  – Dynamic taint analysis ([2][3][4])
  – Symbolic execution/ SMT solver ([2][5])
  – Trace replay ([6])
Making It Practical

• TREE integrates with IDA Pro now and other mainstream binary analysis environments (later)

• TREE leverages debugging infrastructure to support tracing on multiple platforms

• CBASS uses Intermediate Representation (REIL [6][7])-based approach to support ISA-independent analysis
CBASS Supports Both Automated & Interactive Analysis

TREE
Interactive Analysis

Automated Fuzzer
Automated Analysis

CBASS
IR-based Symbolic Execution Engine

TREE fills gaps for interactive analysis
Tools Support Interactive Binary Analyses

Fix the Dots

Connect the Dots

Explore New Dots

**TREE**
- **Replay**
  - Don’t chase a moving target
- **Taint Analysis**
  - Focus only on data and code that are relevant

**CBASS**
- **Symbolic Execution**
  - Explore the unexplored path and code
Illustrative Dots in Vulnerability Analysis: A Running Example

//INPUT
ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);

//INPUT TRANSFORMATIONS
……

//PATH CONDITIONS
if(sBigBuf[0]=='b') iCount++;
if(sBigBuf[1]=='a') iCount++;
if(sBigBuf[2]=='d') iCount++;
if(sBigBuf[3]=='!') iCount++;
if(iCount==4) // bad!
    StackOVflow(sBigBuf,dwBytesRead)
else // Good
    printf("Good!");

//Vulnerable Function
void StackOVflow(char *sBig,int num)
{
    char sBuf[8]= {0};
    ......
    ......
for(int i=0;i<num;i++)
    //Overflow when num>8
    {
        sBuf[i] = sBig[i];
    }
    ......
return;
}
Our Tools Support

Fixing the Dots (TREE)
Fix the Dots

• Reverse engineers don’t like moving dots

• Why do the dots move?
  – Concurrency (multi-thread/multi-core) brings non-deterministic behavior
  – ASLR guarantees nothing will be the same
Fix the Dots

• How does TREE work?
  – Generates the trace at runtime
  – Replays it offline

• TREE trace
  – Captures program state = \{Instruction, Thread, Register, Memory\}
  – Fully automated generation

• TREE can collect traces from multiple platforms
  – Windows/Linux/Mac OS User/Kernel and real devices
    (Android/ARM, Cisco routers/MIPS, PowePC)
TREE Taint-based Replay vs. Debug-based Replay

• Debug-replay lets you connect the dots
  – Single step, stop at function boundary, Breakpoint

• TREE replay connects dots for you
  – Deterministic replay with taint-point break
Our Tools Support

Connecting the Dots (TREE)
Connecting Dots is Hard

- Basic elements complex in real programs
  - Code size can be thousands (++) of lines
  - Inputs can come from many places
  - Transformations can be lengthy
  - Paths grow exponentially
- Basic elements likely separated by millions of instructions, spatially and temporally
- Multiple protections built in
Techniques Help Connect the Dots

• Dynamic Taint Analysis
  – Basic Definitions
    o Taint source
    o Taint Sink:
    o Taint Policy:

• Taint-based Dynamic Slicing
  – Taint focused on data
  – Slicing focused on relevant instructions and sequences
Connect the Dots

- TREE connects dots -- using taint analysis

Taint Source:
Connect the Dots

• TREE connects dots -- using taint analysis
Connect the Dots

- TREE connects dots -- using taint analysis

Taint Source:

Taint policy

Taint Sink:
Connect the Dots

• TREE connects dots -- using taint analysis

Taint Source: Taint policy Taint Sink:

- Dynamic Slicing
Find the Dots and Slice that Matter

In practice, most dots don’t matter – eliminate them quickly to focus on what matters
Connecting Dots in Running Example

Taint Source: (Input)

Taint policy (Data)

Taint Sink: eip

---

The Slice

```
call ds:ReadFile
movb (%eax), %dl
movb %dl, -0x8(%ebp,%ecx,1)
retl
```

The Taint Graph

[Diagram showing taint flow from input to eip]
What You Connect is What You Get

• Dots can be connected in different ways
  – Data dependency
  – Address dependency
  – Branch conditions
  – Loop counter

• Connect dots in different taint policies
TREE
TAINT-ENABLED
REVERSE ENGINEERING ENVIRONMENT
TREE Key Components

**Execution Tracer**
- Cross-platform Debugging

**Taint Analyzer & Slicing**
- Taint Visualizer & Slice Navigator
  (IDA Native/Qt)

**TREE Replay**
- Execution Trace

**Taint Graph**
- IDA Plug-in
TREE: The Front-end of Our Interactive Analysis System

Taint Graph
**TREE: The Front-end of Our Interactive Analysis System**

<table>
<thead>
<tr>
<th>UUID</th>
<th>Type</th>
<th>Name</th>
<th>Start Sequence</th>
<th>End Sequence</th>
<th>Instruction</th>
<th>Child C</th>
<th>Child D</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>register</td>
<td>eip_3_14876</td>
<td>0x11c</td>
<td></td>
<td>retl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>register</td>
<td>eip_2_14876</td>
<td>0x11c</td>
<td></td>
<td>retl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>register</td>
<td>eip_1_14876</td>
<td>0x11c</td>
<td></td>
<td>retl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>register</td>
<td>eip_0_14876</td>
<td>0x11c</td>
<td></td>
<td>retl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>memory</td>
<td>0x38f79b</td>
<td>0x112</td>
<td></td>
<td>movb %dl, -...</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>51</td>
<td>register</td>
<td>edx_0_14876</td>
<td>0x111</td>
<td>0x117</td>
<td>movb (%eax), %dl</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>50</td>
<td>memory</td>
<td>0x38f79a</td>
<td>0x106</td>
<td></td>
<td>movb %dl, -...</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>49</td>
<td>register</td>
<td>edx_0_14876</td>
<td>0x105</td>
<td>0x10b</td>
<td>movb (%eax), ...</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>48</td>
<td>memory</td>
<td>0x38f799</td>
<td>0xfa</td>
<td></td>
<td>movb %dl, -...</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>47</td>
<td>register</td>
<td>edx_0_14876</td>
<td>0xf9</td>
<td>0xff</td>
<td>movb (%eax), ...</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td>memory</td>
<td>0x38f798</td>
<td>0xee</td>
<td></td>
<td>movb %dl, -0x8(%ebp,</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>register</td>
<td>edx_0_14876</td>
<td>0xed</td>
<td>0xf3</td>
<td>movb (%eax), %dl</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>input</td>
<td>0x38f7bb</td>
<td>0x0</td>
<td></td>
<td>0x12f106a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**TREE: The Front-end of Our Interactive Analysis System**

<table>
<thead>
<tr>
<th>Instruction Address</th>
<th>Disassembly</th>
<th>Registers</th>
<th>Memory Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>270 0x12f1130</td>
<td>mov   eax, [ebp+arg_0]</td>
<td>eax=0x38f7ba ebp=0x38f794</td>
<td>R 4 0x38f79c</td>
</tr>
<tr>
<td>271 0x12f1133</td>
<td>add   eax, [ebp+var_C]</td>
<td>eax=0x38f7ac ebp=0x38f794 eflags=0x287</td>
<td>R 4 0x38f788</td>
</tr>
<tr>
<td>272 0x12f1136</td>
<td>mov   ecx, [ebp+var_C]</td>
<td>ebp=0x38f794 ecx=0xf</td>
<td>R 4 0x38f788</td>
</tr>
<tr>
<td>273 0x12f1139</td>
<td>mov   dl, [eax]</td>
<td>eax=0x38f7bb dl=0xf</td>
<td>R 1 0x38f7bb</td>
</tr>
<tr>
<td>274 0x12f113b</td>
<td>mov   [ebp+ecx+var_8], dl</td>
<td>dl=0x68 ebp=0x38f794 ecx=0xf</td>
<td>W 1 0x38f79b</td>
</tr>
<tr>
<td>275 0x12f113f</td>
<td>jmp   short loc_12F111F</td>
<td>eip=0x12f113f</td>
<td></td>
</tr>
<tr>
<td>276 0x12f111f</td>
<td>mov   ecx, [ebp+var_C]</td>
<td>ebp=0x38f794 ecx=0xf</td>
<td>R 4 0x38f788</td>
</tr>
<tr>
<td>277 0x12f1122</td>
<td>add   ecx, 1</td>
<td>eflags=0x216 ecx=0xf</td>
<td></td>
</tr>
<tr>
<td>278 0x12f1125</td>
<td>mov   [ebp+var_C], ecx</td>
<td>ebp=0x38f794 ecx=0x10</td>
<td>W 4 0x38f788</td>
</tr>
</tbody>
</table>
**TREE: The Front-end of Our Interactive Analysis System**

<table>
<thead>
<tr>
<th>Stack View</th>
<th>Register View</th>
<th>Memory View</th>
</tr>
</thead>
<tbody>
<tr>
<td>0038F758</td>
<td>0C 3F 7D 77 2C 85 4A 74</td>
<td>00 00 2F 0</td>
</tr>
<tr>
<td>0038F768</td>
<td>00 00 00 00 F0 53 7D 77</td>
<td>5C F7 38 0</td>
</tr>
<tr>
<td>0038F778</td>
<td>F0 F7 38 00 23 41 87 77</td>
<td>B4 4D OF 0</td>
</tr>
<tr>
<td>0038F788</td>
<td>0C 3F 7D 77 70 10 2F 01</td>
<td>5C 00 00 0</td>
</tr>
<tr>
<td>0038F798</td>
<td>10 00 00 00 A8 F7 38 00</td>
<td>00 00 00 0</td>
</tr>
<tr>
<td>0038F7A8</td>
<td>10 00 00 00 62 61 64 21</td>
<td>62 65 74 7</td>
</tr>
<tr>
<td>0038F7B8</td>
<td>73 74 74 68 00 F8 38 00</td>
<td>E1 12 2F 0</td>
</tr>
<tr>
<td>0038F7C8</td>
<td>A0 1B 45 00 38 20 45 00</td>
<td>40 EA 7A 7</td>
</tr>
<tr>
<td>0038F7D8</td>
<td>00 00 00 00 00 E0 FD 7E</td>
<td>00 00 00 0</td>
</tr>
<tr>
<td>0038F7E8</td>
<td>D0 F7 38 00 B2 3E 7E 4A</td>
<td>3C F8 38 0</td>
</tr>
</tbody>
</table>

Register/stack/memory Views
TREE: The Front-end of Our Interactive Analysis System

Replay is focal point of user interaction
Tree Demo

Using TREE to Analyze a Crash
Our Tools Support

Exploring New Dots
A Key Branch Point for a Duck

Connects 16 -> 17
The Path for a ...

• Reverse engineers don’t just connect dots; they want to explore new dots:

Connects 16 -> 26
Explore New Dots

• How do you force the program to take a different path to lead to “bad!”?

    //INPUT
    ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);
    ....
    //PATH CONDITION
    if(sBigBuf[0]=='b') iCount++;
    if(sBigBuf[1]=='a') iCount++;
    if(sBigBuf[2]=='d') iCount++;
    if(sBigBuf[3]=='!') iCount++;
    if(iCount==4) // “bad!” path
        StackOVflow(sBigBuf,dwBytesRead) ?
    Else // “Good” path
        printf(“Good!”);
Explore New Dots

- User wants execution to take different path at a branch point Y – what input will make that happen?

**User:**
How to execute different path at branch Y?

**TREE:** Input [0] must be ‘b’

**TREE:** Can we negate path condition at Y?

**CBASS:** This byte must be ‘b’

**CBASS** (symbolic execution)
Explore New Dots Demo

IDA Plugin (Front End)

Execution Tracer (Cross-platform Debugging)

Execution Trace

Taint Analyzer & Slicing

Taint Visualizer & Slice Navigator (IDA Native Qt)

TREE Replay

On-demand Symbolic Execution & Constraint Generation

SMT Solver

New input

Path Selection

Path constraints

Satisfiable input

CBASS (BACK End)
Task 1: Force the Program to Take "bad!" Path

//INPUT
ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);

//INPUT TRANSFORMATION

......

//PATH CONDITION
if(sBigBuf[0]=='b') iCount++;
if(sBigBuf[1]=='a') iCount++;
if(sBigBuf[2]=='d') iCount++;
if(sBigBuf[3]=='!') iCount++;
if(iCount==4) // "bad!" path
    //Vulnerable Function
StackOVflow(sBigBuf,dwBytesRead)
else
    printf("Good!");
PIN: A popular Dynamic Binary Instrumentation (DBI) Framework
2 TREE Console: Trace Generation

PINAgent: Connects TREE with PIN tracer
TREE: Taint Analysis Configuration

- **Taint Propagation Policy**
  - TAINT_DATA
  - TAINT_BRANCH
  - TAINT_COUNTER
  - TAINT_ADDRESS

- **Instruction Set Architecture**
  - x86
  - x86_64
  - ARM
  - PPC
  - MIPS

- **Misc**
  - PIN
  - Verbose

- **Image Load Table**
  - Name: ['badcon...', ntdll.dll, kernel32.dll, kernel32.dll, KernelBase...]
  - Address: [0x120000, 0x7760000, 0x777c0000, 0x777c0000, 0x75ff0000]
  - Size: [0x5000, 0x180000, 0x110000, 0x110000, 0x47000]

- **Taint Source Table**
  - Input Address: 0x38f7ac
  - Size: 16
  - Size: 62616421

- **Taint Graph Output**
  - Path Conditions:
    - [19] bc_0x3a[0x3a:0x0] < jnz 0x9
    - [18] reg_eflags_0[0x39:0x0] < cmp $0x62, %edx
    - [17] reg edx_0_0[0x38:0x0] < movsxb -0x10(%ebp), %edx
    - [1] ln_0x12ff6c[0x0:0x0] < -0xffff:ReadFile
    - [22] bc_0x3d[0x3d:0x0] < jnz 0x9
TREE: Branch Taint Graph

<table>
<thead>
<tr>
<th>UUID</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>input</td>
<td>0x12ff6c</td>
</tr>
<tr>
<td>17</td>
<td>register</td>
<td>edx_0_0</td>
</tr>
<tr>
<td>18</td>
<td>register</td>
<td>eflags_0</td>
</tr>
<tr>
<td>19</td>
<td>branch</td>
<td>0x3a</td>
</tr>
<tr>
<td>2</td>
<td>input</td>
<td>0x12ff6d</td>
</tr>
<tr>
<td>20</td>
<td>register</td>
<td>ecx_0_0</td>
</tr>
<tr>
<td>21</td>
<td>register</td>
<td>eflags_0</td>
</tr>
<tr>
<td>22</td>
<td>branch</td>
<td>0x3d</td>
</tr>
<tr>
<td>23</td>
<td>register</td>
<td>eax_0_0</td>
</tr>
<tr>
<td>24</td>
<td>register</td>
<td>eflags_0</td>
</tr>
<tr>
<td>25</td>
<td>branch</td>
<td>0x40</td>
</tr>
<tr>
<td>26</td>
<td>register</td>
<td>edx_0_0</td>
</tr>
<tr>
<td>27</td>
<td>register</td>
<td>eflags_0</td>
</tr>
<tr>
<td>28</td>
<td>branch</td>
<td>0x43</td>
</tr>
<tr>
<td>3</td>
<td>input</td>
<td>0x12ff6e</td>
</tr>
<tr>
<td>4</td>
<td>input</td>
<td>0x12ff6f</td>
</tr>
</tbody>
</table>
Negate Tainted Path Condition to Exercise a New ("Bad") Path

CBASS (Cross-platform Symbolic Execution)

Connecting to CBASS server at 127.0.0.1:8888
Query on branch condition of: [19]bc_0x3a [jnz 0x9]
Result: Offset=0, Value=98

Connecting to CBASS server at 127.0.0.1:8888
Query on branch condition of: [22]bc_0x3d [jnz 0x9]
Result: Offset=1, Value=97

Connecting to CBASS server at 127.0.0.1:8888
Query on branch condition of: [25]bc_0x40 [jnz 0x9]
Result: Offset=2, Value=100

$b$

$a$

$d$
On-demand Symbolic Execution (What Happens Behind the Scene)

(set-logic QF_AUFBV)

(declare-fun _IN_0x12ff6c_0x0_SEQ0 () (_ BitVec 8))
(declare-fun EXPR_0 () (_ BitVec 32))
(assert (= EXPR_0 (bvsub ((_ sign_extend 24) (bvxor _IN_0x12ff6c_0x0_SEQ0 (_ bv128 8))) (_ bv4294967168 32))))

(assert (= (ite (not (= (ite (not (= (bvand ((_ extract 63 0) (bvsub ((_ sign_extend 32) (bvand ((_ extract 31 0) EXPR_0) (_ bv4294967295 32))) (_ bv98 64))) (_ bv4294967295 64))) (_ bv0 64))) (_ bv1 32) (_ bv0 32)) (_ bv0 32)) (_ bv1 8) (_ bv0 8)))

(check-sat)
(get-value (_IN_0x12ff6c_0x0_SEQ0))
TREE: Re-execute with “Satisfiable” Input

7 Satisfiable Input

8
Task 2: Own the Execution
Assume Payload at 0x401150

```
.align 10h
push ebp
mov ebp, esp
push 1010h
push offset aYouHaveBeenHacked ; "*** Yo
push offset aCassandraCrossPlatform ; "CBASS<
push 0
call ds:MessageBoxA
push 0xFFFFFFFFh
call ds:exit
```
TREE Constraint Dialogue
Task 2: Own the Execution: From Crash to Exploit

Symbolic eip =
(= expr_0 (concat (bvand (bvor
_IN_0x12ff6c_0xd_SEQ0 (_ bv0 8)) (_ bv255 8)))
(bvand (bvor _IN_0x12ff6c_0xc_SEQ0 (_ bv0 8))
(_ bv255 8))))

Query:
get-value (_IN_0x12ff6c_0xd_SEQ0
_IN_0x12ff6c_0xc_SEQ0
_IN_0x12ff6c_0xe_SEQ0
_IN_0x12ff6c_0xf_SEQ0)

Sat:
(_IN_0x12ff6c_0xd_SEQ0 #x11
_IN_0x12ff6c_0xc_SEQ0 #x50
_IN_0x12ff6c_0xe_SEQ0 #x40
_IN_0x12ff6c_0xf_SEQ0 #x00

Symbolize Input and perform concrete-symbolic execution

SMT Solver
Using CBASS/TREE to Explore Bad Paths and Refine Exploits
# Real World Case Studies

<table>
<thead>
<tr>
<th>Target Vulnerability</th>
<th>Vulnerability Name</th>
<th>Target Application Mode</th>
<th>Target OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2005-4560</td>
<td>Windows WMF</td>
<td>User Mode</td>
<td>Windows</td>
</tr>
<tr>
<td>CVE-2207-0038</td>
<td>ANI Vulnerability</td>
<td>User Mode</td>
<td>Windows</td>
</tr>
<tr>
<td>OSVDB-2939</td>
<td>AudioCoder Vulnerability</td>
<td>User Mode</td>
<td>Windows</td>
</tr>
<tr>
<td>CVE-2011-1985</td>
<td>Win32k Kernel Null Pointer Dereference</td>
<td>Kernel Mode</td>
<td>Windows</td>
</tr>
<tr>
<td>CVE-2004-0557</td>
<td>Sound eXchange (SoX) WAV Multiple Buffer Overflow</td>
<td>User Mode</td>
<td>Linux</td>
</tr>
<tr>
<td>Compression/Decompression</td>
<td>Zip on Android</td>
<td>User Mode</td>
<td>Real Device Trace Generation (In Progress)</td>
</tr>
</tbody>
</table>
Highlights from Real World Case Study: Windows WMF Vulnerability (CVE-2005-4560)

• WMF SETABORTPROC Escape Vulnerability
  – http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4560
  – The Windows Graphical Device Interface library (GDI32.DLL) in Microsoft Windows allows remote attackers to execute arbitrary code via a Windows Metafile (WMF) format image with a crafted SETABORTPROC GDI Escape function call, related to the Windows Picture and Fax Viewer (SHIMGVW.DLL).
**WMF Format**

- **[MS-WMF]: Windows Metafile Format**

- **A Simplified One:**

- **Overall WMF File Structure:**

<table>
<thead>
<tr>
<th>Meta Header</th>
<th>Meta Record 1</th>
<th>Meta Record 2</th>
<th>Meta Record 3</th>
<th>...</th>
</tr>
</thead>
</table>

- One type of record is “escape” record
- `SETABORTPROC` escape allow an application to register a hook function to handle spooler errors
WMF Crash

The WMF SETABORTPROC Vulnerability

`rundll32.exe c:\windows\system32\shimgvw.dll,ImageView_Fullscreen C:\escape\escape.wmf`

Dynamic Facts: 229,679 instructions executed just to cause the crash
WMF Taint Graph

Partial TREE Taint Graph Visualization

(Taint Sources)
Input Bytes 7th-10th
(FileSize)

Register taint node

Memory taint node

(Taint Sources)
Initial Input Bytes 29th(0x1d)-32th(0x20)
(SHELLCODE)

Data Dependency
(Default)

Security-Sensitive Locations
(Taint Sinks)

ALL Nodes are Clickable to Take User to IDA CFG

LOOP Counter Dependency
(C)
WMF File: The Fields & The Vulnerability

• Key Structures:

```c
typedef struct _WindowsMetaHeader
{
    WORD FileType; /* Type of metafile (0=memory, 1=disk) */
    WORD HeaderSize; /* Size of header in WORDS (always 9) */
    WORD Version; /* Version of Microsoft Windows used */
    DWORD FileSize; /* Total size of the metafile in WORDs */
    WORD NumOfObjects; /* Number of objects in the file */
    DWORD MaxRecordSize; /* The size of largest record in WORDs */
    WORD NumOfParams; /* Not Used (always 0) */
} WMFHEAD;
```

```c
typedef struct _StandardMetaRecord
{
    DWORD Size; /* Total size of the record in WORDs */
    WORD Function; /* Function number (defined in WINDOWS.H) */
    WORD Parameters[]; /* Parameter values passed to function */
} WMFRECORD;
```
WMF Slicing (1)

An Instruction Slice Traced Back from Crash Site to Input
Each node uniquely trace back to one execution event through its sequence number

0x77f330a3 call eax 2 ffd0 0x0 0x3812f Reg( EAX=0xa8b94 ESP=0xb4fb88 EIP=0x77f330a3 ) W 4 b4fb88

0x77c472e3 rep movsd 2 f3a5 0x0 0xb142 Reg( EDI=0xa8804 eflags=0x10216 ESI=0xa9f8c ECX=0xa ) R 4 a9f8c cc_cc_cc_cc W 4 a8804

0x77f2e997 mov ecx, [ebp+arg_8] 3 8b4d10 0x0 0xc5c3 Reg( EBP=0xb4fbf8 ECX=0x7c809a20 ) R 4 b4fc08 44_0_0_0

0x77f2e983 mov [ebp+arg_8], eax 3 894510 0x0 0xbd8c Reg( EAX=0x44 EBP=0xb4fbf8 ) W 4 b4fc08

0x77f2e97f add eax, eax 2 03c0 0x0 0xbd89 Reg( EAX=0x22 eflags=0x246 )

0x77f2e949 mov eax, [edi+6] 3 8b4706 0x0 0xbd7d Reg( EAX=0xa8920 EDI=0xa87e8 ) R 4 a87ee 22_0_0_0

0x77c472e3 rep movsd 2 f3a5 0x0 0xb13c Reg( EDI=0xa87ec eflags=0x10216 ESI=0xa9f74 ECX=0x10 ) R 4 a9f74 0_3_22_0 W 4 a87ec
WMF Slicing (2)

An Instruction Slice with Text Helps

Put Instruction In Its Context Helps More
Module: gdi32.dll
Function: CommonEnumMetaFile

text:77F330A3 call eax
WMF Slicing (3)

An Instruction Slice with Text Helps a Little

text:77F330A3 call eax

More Context Helps More
Module: gdi32.dll
Function: CommonEnumMetaFile
Call Graph: caller PlayMetaFile
WMF -- The Relevant Parts

The WMF SETABORTPROC Vulnerability

```
rundll32.exe c:\windows\system32\shimgvw.dll,ImageView_Fullscreen
C:\escape\escape.wmf
```

Dynamic Facts:
Out of 229,679 instructions executed just to cause the crash
ONLY 12 Unique Instructions Are Relevant to the CRASH
Conclusions

• Our tools support *interactive binary analysis*, with *Replay, Dynamic Taint Analysis, and Symbolic Execution*.

• **TREE** runs on top of IDA Pro and supports cross-platform trace collection, taint analysis and replay.

• **CBASS** (based on REIL) enables IR-based architecture-independent symbolic execution and can support both automated and interactive analysis.

• **YOU drive the tools!**
Where You Can Get TREE

• TREE is open source at: 
  http://code.google.com/p/tree-cbass/
  • First version of TREE (Taint Analysis) is released
  • Replay is in Progress
  • CBASS is Following

— Contacts:
  • Li.L.Lixin@gmail.com, Project Lead
  • xingzli@gmail.com, Developer
  • locvnguy@gmail.com, Developer
  • james.just@gmail.com, Program Manager
Acknowledgements

• Thanks to Ilfak Guilfanov and the IDA team for promptly fixing the bugs that we have reported to them and for their suggestions on the GUI integration.

• Thanks to Thomas Dullien and Tim Kornau of the Google Zynamics team for making their latest version of REIL available to us.

• Thanks to numerous reviewers at Battelle Memorial Institute for their feedback
References


