Identifying and Analyzing Pointer Misuses for Sophisticated Memory-corruption Exploit Diagnosis

Mingwei Zhang (†)
Aravind Prakash (§)
Xiaolei Li (†)
Zhenkai Liang (†)
Heng Yin (§)

(†) School of Computing, National University of Singapore
(§) Department of Electrical Engineering and Computer Science, Syracuse University
Simple Stack Buffer Overflow

- An attacker overwrites vulnerable function return address, which points to shellcode on stack.

- These single step attacks don’t work anymore thanks to:
  - ASLR, DEP, NX, etc.
Exploiting SEH Mechanism

- SEH Record
  - exception handler
  - prev
  - return address
  - security cookie
  - buffer
    - DispatcherContext
    - ContextRecord
    - EstablisherFrame
    - ExceptionRecord
    - return address

Stack growth

- step 1: pop
- step 2: pop
- step 3: ret

arguments prepared by OS for exception handler

return address of exception handler function
Insights

- Recent attacks employ **multiple steps**.
- **Pointer misuse** is very prominent in sophisticated attacks.
- Key steps constitute pointer misuses.

Our Goal:
Diagnosing pointer misuses in a multi-step attack.
Pointerscope – Attack Diagnosis Engine

- **Type System** tailored to diagnose pointer misuse.
- **Eager type inference** system to detect pointer misuses.
- Provide **big picture** of the misuse through *key steps* graph.
Overview

Exploit (Malicious Server) → Guest OS → Vulnerable Program (Browser) → Execution Monitor (TEMU) → Execution Trace → Type Inference Engine → Diagnosis Engine → Diagnosis Report
Variable And Variable Type

- A variable is a memory location or a register.
- Simple primitive variable types:
  - Integer
  - Control Pointer (or code pointer)
  - Data Pointer
  - Other. (The rest of the types)
Type Lattice

any

non-pointer

other

integer

control pointer

data pointer

conflict
Eager Dynamic Type Inference

- **Type Propagation:**
  - `mov %eax, %ebx`
    - Inference: eax and ebx have same type

- **Type Constraints:**
  - `call %eax`
    - Inference: %eax contains Control Pointer
Example – Type Inference

{eax, ebx} : ANY ------------------------------ mov %eax, %ebx
{eax, ebx, ecx} : ANY ------------------------- mov %ebx, %ecx
ecx is an INT --------------------------------- imul $0x05, %ecx, %ecx
{eax, ebx, ecx} : Integer --------------------- mov %ecx, %edx

Used as a pointer. **Conflict** ------------------------------ call *%ecx

Harder than it seems!
Challenges

- X86 supports base-index with displacement – Problem: Compilers don’t follow convention.

Solution: Register closest to result is the base.
Challenges… contd.

- Individual instructions not always lead to accurate type inferences.
  - Eg:

    ```
    not %ebp
    or $0x3,%ebp
    not %ebp
    ```

    ```
    and $0xffffffffc,%ebp
    ```

- Solution: recognizing the common patterns and treat them as special cases
Challenges… contd.

- LEA designed to load effective address, but often used in arithmetic.

\[
\text{lea } $0x8(\%eax,\%edx,4), \%ecx \quad \Rightarrow \quad \%ecx = \%eax + \%edx \times 4 + $0x8
\]

Solution: Treat lea as an arithmetic operation.
Challenges…

- More challenges discussed in the paper!
Key Steps Graph – Example

Infer: Top of stack, (0xbfff0000) is an INT

Where the variable was initialized

Where the variable was misused

Returning to 0xbfff0000

Mem: 0xbfff0000 [4]

0x42050000: pushfd

Type Origin (INT)

Type Usage (CTR)

0x08048000: ret
Evaluation

- Implementation
  - Execution monitor on TEMU.
  - 3.6K lines of C code.

- Experimental setup
  - Evaluated against real world exploits from Metasploit framework.
## Summary of Effectiveness

<table>
<thead>
<tr>
<th>CVE</th>
<th>Attack Technique</th>
<th>Runtime*</th>
<th>Pointer Misuses</th>
<th>Trace Size</th>
<th>Slice Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2010-0249</td>
<td>Uninitialized memory; heap spray</td>
<td>18m23s, 8m30s</td>
<td>11</td>
<td>307,987,560</td>
<td>48,404,242</td>
</tr>
<tr>
<td>CVE-2009-3672</td>
<td>Incorrect variable initialization; heap-spray</td>
<td>3m10s, 31s</td>
<td>2</td>
<td>22,759,299</td>
<td>955,325</td>
</tr>
<tr>
<td>CVE-2009-0075</td>
<td>Uninitialized memory; heap spray</td>
<td>25m, 21m16s</td>
<td>6</td>
<td>411,323,083</td>
<td>44,792,770</td>
</tr>
<tr>
<td>CVE-2006-0295</td>
<td>Heap overflow; heap spray</td>
<td>3m5s, 1s</td>
<td>3</td>
<td>808,392</td>
<td>34,883</td>
</tr>
<tr>
<td>CVE-2006-1016</td>
<td>Stack overflow; SEH exploit</td>
<td>4m59s, 1m33s</td>
<td>3</td>
<td>64,355,691</td>
<td>1,334,253</td>
</tr>
<tr>
<td>CVE-2006-4777</td>
<td>Integer overflow; heap spray</td>
<td>1m45s, 40s</td>
<td>3</td>
<td>2,632,241</td>
<td>1,669,751</td>
</tr>
<tr>
<td>CVE-2006-1359</td>
<td>Incorrect variable initialization; heap spray</td>
<td>11m58s, 13s</td>
<td>2</td>
<td>8,336,193</td>
<td>29,520</td>
</tr>
<tr>
<td>CVE-2010-3333</td>
<td>Stack overflow vulnerability; SEH exploit</td>
<td>18m53s, 7m24s</td>
<td>1</td>
<td>236,331,307</td>
<td>814,305</td>
</tr>
<tr>
<td>CVE-2010-3962</td>
<td>Incorrect variable initialization; heap spray</td>
<td>10m36, 15s</td>
<td>2</td>
<td>9,281,019</td>
<td>78,704</td>
</tr>
</tbody>
</table>

*Time taken to generate trace, time taken to generate key steps*
Case Study: CVE-2009-3672

- This is a real world exploit for vulnerable version of IE Browser
- This attack is caused by a vulnerability in the class `CDispNode`'s member function `SetExpandedClipRect`
The First Type Conflict

0x749120f2: \textbf{or $0x2$, \%eax}
\texttt{I}@0x00000000[1](R) \texttt{R}@eax[4](RW)
0x102098@mshtml.dll@CLayout::SizeDispNode

Type Origin (INT)

\texttt{eax (4 bytes)}

Type Usage (CTR)

0x7490e854: \textbf{call *0x2c(\%eax)}
\texttt{M}@0x74831546[4] \texttt{M}@0x0013e0d4[4]
0xfe838@mshtml.dll@CLayout::GetFirstContentDispNode

Infer: Integer

\textbf{or $0x2$, \%eax}

Used as Control Ptr Violation

19
The Second Type Conflict

0x74912000: \texttt{call *0x2c(\%eax)}
\texttt{M@0x74000004\[4\]} \texttt{M@0x0013e068\[4\]}
0x102098@mshtml.dll@CLayout::SizeDispNode

0x74943a14: \texttt{call *0x30(\%eax)}
\texttt{M@0x74000008\[4\]} \texttt{M@0x0013dde4\[4\]}
0x13393f@mshtml.dll@CDispNode::GetNodeClipTransform

Type Origin (CTR) \hspace{5cm} Type Origin (CTR)

Type Usage (CTR) \hspace{5cm} Type Usage (CTR)

\small{\texttt{[16940584] 0x7490e584 call *0x2c(\%eax)}
\texttt{M@0x74000006\[4\]} \texttt{M@0x0013e0d4\[4\]}
0xfe838@mshtml.dll@CLayout::GetFirstContentDispNode}
Final Result

Vulnerable Location: mshtml.dll@CLayout::SizeDispNode
LOGIC ERROR
Reducing False Positives

- What makes it hard?
  - Compiler optimizations
  - Code obfuscation – even by proprietary code.

- Note: Our goal is NOT to eliminate False Positives.
Related Work

- Attack Diagnosis Techniques
  - BackTracker [King, et. al, SOSP’03], Dynamic Taint Analysis [Newsome, et. al, NDSS’05]
- Type and Data Structure recovery from binary
  - Rewards [Zin, et.al, NDSS’10], Howard [Slowinska, NDSS’11], Tie [Lee, et.al, NDSS’11]
- Defense and evasion techniques
  - CFI [Abadi, et.al, CCS’05], DFI [Castro, et.al, OSDI’06], WIT [Akritidis, et.al, IEEE S&P’08]
Conclusion

- We define a **pointer centric type system** to track pointers.
- We design a **type inference system** to detect pointer misuses.
- We **generate the key steps graph** to identify key steps.
- We evaluate our work by testing our system on real-world exploits from metasploit.
Questions?
Challenges... contd.

- Handling memory copy operations
  - Memory copy operations may break the integrity of variables

Solution: Aggregation.
Case Study: SEH Attack