Kruiser: Semi-synchronized Non-blocking Concurrent Kernel Heap Buffer Overflow Monitoring

Donghai Tian¹,², Qiang Zeng², Dinghao Wu², Peng Liu² and Changzhen Hu¹

¹ Beijing Institute of Technology
² The Pennsylvania State University
Kernel Heap Buffer Overflow
Motivation

• There are more and more kernel buffer overflow exploits.

• To our knowledge, there are no practical mechanisms that have been widely deployed detecting kernel heap buffer overflows.
Current Methods: Limitations 1 & 2

• Some approaches perform detection before each buffer write operation.
  [PLDI '04], [USENIX ATC '02], [NDSS '04]

  High overhead!

• Some approaches do not check heap buffer overflows until a buffer is de-allocated.
  [LISA '03], [BLACKHAT '11]

  Large detection delay!
Our Idea

Inlined Checking

Concurrent checking
Basic Method

• Canary-based Concurrent Monitoring
Challenges

• Self-protection.
  • Monitor and the metadata

• Synchronization.
  • Races between hooks and monitor

• Compatibility.
  • OS and hardware
Out-of-the-VM Architecture

(Our previous CCS submission - rejected)
Hybrid VM monitoring Architecture

*(NDSS submission - accepted)*
Now, Kernel Cruising

• How to gather canary location info?

• How to deal with the races between hooks and monitor?
Kernel Cruising

• Page Identity Array (PIA)
  • Heap buffer canary location information
  • Other information

• Race conditions
  • Concurrent updates by two hooks
  • Inconsistent reads by monitor
  • Time of check to time of use (TOCTTOU)
Semi-synchronized Non-blocking Cruising Algorithm

• Avoid Concurrent Entry Updates.
  • Put the PIA entry update operations into the critical section.
Resolve TOCTTOU

**Hook:**

\[
\text{if the page is moved to the heap page pool} \\
\quad \text{flag} = \text{true}; \\
\text{else if the page is removed from the heap} \\
\quad \text{flag} = \text{false};
\]

**Monitor:**

\[
\text{if (the canary is tempered)} \\
\quad \text{if (flag == true)} \{ \text{// the page is still in heap} \\
\quad \quad \text{report overflow!} \}
\]
ABA Hazard Solution

if the page is moved to the heap page pool
    version++;
else if the page is removed from the heap
    version++;

...

if (the canary is tampered) {
    if (version == original version) {
        report overflow!
    }
}
Secure Canary Generation

- R1) The canaries are not predictable.

- R2) The canary generation and verification algorithms should be efficient.

- Generate unpredictable canaries using RC4 from a per-virtual-page random value.
Outline

• Idea
• Architecture
• Kernel Cruising
• Evaluation
• Related Work
• Summary
Effectiveness

• We exploited five heap buffer overflow vulnerabilities in Linux, including three synthetic bugs and two real world vulnerabilities.

• All the overflows are successfully detected by Kruiser.
SPEC CPU2006 performance (normalized to the execution time of original Linux).
Scalability

Throughput of the Apache web server for varying numbers of concurrent requests.
Detection Latency

Different cruising cycle for different applications in the SPEC CPU2006 benchmark

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Maximum cruising number</th>
<th>Minimum cruising number</th>
<th>Average cruising number</th>
<th>Average cruising cycle(μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>107,824</td>
<td>105,145</td>
<td>106,378</td>
<td>39,259</td>
</tr>
<tr>
<td>bzip2</td>
<td>79,085</td>
<td>76,325</td>
<td>76,682</td>
<td>27,662</td>
</tr>
<tr>
<td>gcc</td>
<td>78,460</td>
<td>76,810</td>
<td>77,413</td>
<td>27,774</td>
</tr>
<tr>
<td>mcf</td>
<td>82,885</td>
<td>79,328</td>
<td>79,540</td>
<td>28,156</td>
</tr>
<tr>
<td>gobmk</td>
<td>80,761</td>
<td>80,345</td>
<td>80,519</td>
<td>28,606</td>
</tr>
<tr>
<td>hmmer</td>
<td>81,278</td>
<td>80,435</td>
<td>80,591</td>
<td>28,635</td>
</tr>
<tr>
<td>sjeng</td>
<td>81,437</td>
<td>80,259</td>
<td>80,535</td>
<td>28,610</td>
</tr>
<tr>
<td>libquantum</td>
<td>80,911</td>
<td>80,317</td>
<td>80,407</td>
<td>28,493</td>
</tr>
<tr>
<td>h264ref</td>
<td>80,756</td>
<td>80,337</td>
<td>80,480</td>
<td>28,572</td>
</tr>
<tr>
<td>omnetpp</td>
<td>82,109</td>
<td>80,796</td>
<td>81,088</td>
<td>28,836</td>
</tr>
<tr>
<td>astar</td>
<td>81,592</td>
<td>81,022</td>
<td>81,097</td>
<td>28,897</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>99,436</td>
<td>82,747</td>
<td>88,454</td>
<td>30,190</td>
</tr>
</tbody>
</table>

10 of 12 applications have less than 29ms (for scanning the kernel heap).
Outline

• Idea
• Architecture
• Kernel Cruising
• Evaluation
• Related work
• Summary
Related Work

• Countermeasures Against Buffer Overflows
  • StackGuard [USENIX Security '98]
  • Heap Integrity Detection [LISA '03]
  • Cruiser [PLDI '11]
  • DieHard [PLDI '06] and DieHarder [CCS '10]

• VM-based Methods
  • SIM [CCS '09]
  • OSck [ASPLOS '11]
Summary

• *Kruiser* can achieve *concurrent monitoring* against kernel heap buffer overflows.
  • Non-blocking
  • Semi-synchronized
  • NO false positive

• The *hybrid VM monitoring* scheme provides high efficiency without sacrificing the security guarantees.
Thank you!

Questions?
Outline

• Background and Idea
• Architecture
• Kernel Cruising
• Evaluation
• Related Work
• Summary
Non-blocking Cruising Algorithm

Monitor()
uint ver1, ver2;
for (int page = 0; page < ENTRY NUMBER; page++){
    ver1 = PIA[page].version;
    if (The page is non-heap page)
        continue; // Bypass non-heap page
    Read the metadata stored in PIA[page];
    ver2 = PIA[page].version;
    if (ver1 != ver2)
        continue; // Metadata was updated
    for (each canary within the page){
        if (the canary is tampered){
            DoubleCheckOnTamper(page, ver1);
        }
    }
}