Why is memory safety still a concern?

PhD Candidacy Exam

Mohamed (Tarek Ibn Ziad) Hassan
mtarek@cs.columbia.edu

Department of Computer Science
Columbia University

April 9th, 2020
10:30 A.M.
Location: TBD

1 Candidate Research Area Statement

Languages like C and C++ are the gold standard for implementing a wide range of software systems such as safety critical firmware, operating system kernels, and network protocol stacks for performance and flexibility reasons. As those languages do not guarantee the validity of memory accesses (i.e., enforce memory safety), seemingly benign program bugs can lead to silent memory corruption, difficult-to-diagnose crashes, and most importantly; security exploitation. Attackers can compromise the security of the whole computing ecosystem by exploiting a memory safety error with a suitably crafted input. Since the spread of Morris worm in 1988 and despite massive advances in memory safety error mitigations, memory safety errors have risen to be the most exploited vulnerabilities.

As part of my research studies in Columbia, I have worked on designing and implementing hardware primitives [1, 2] that provide fine-grained memory safety protection with low performance overheads compared to existing work. My ongoing work in this research include extending the above primitives to system level protection in addition to addressing their current limitations.

2 Faculty Committee Members

The candidate respectfully solicits the guidance and expertise of the following faculty members and welcomes suggestions for other important papers and publications in the exam research area.

- Prof. Simha Sethumadhavan
- Prof. Steven M. Bellovin
- Prof. Suman Jana
3 Exam Syllabus

The papers have broad coverage in the space of memory safety vulnerabilities and mitigations, which are needed to (1) make a fair assessment of current defensive and offensive techniques and (2) explore new threats and defensive opportunities.

- I begin with a brief overview of memory safety [3–5] showing why it is still a concern.

- To motivate the need for complete memory safety solutions, I go over a timeline of prior exploitation techniques and mitigations, as shown in Figure 1. I start with the earliest documented memory corruption attack from 1972. Then, I describe various memory attacks ranging from code injection (e.g., Morris Worm [6]) and code reuse [7–9] to data-oriented attacks [10]. I also cover immediate mitigations, such as ASLR [11], CFI [12], DFI [13], Data Randomization [14, 15], and the recent ARM pointer authentication (PAC) [16].

- Next, I provide an overview of defensive techniques that aim at enforcing spatial and temporal memory safety. First, I group prior techniques, which offer spatial memory safety guarantees, into three main categories (Whitelisting, Blacklisting, and Randomized Allocators). Each group is further divided by the way it manages its own metadata, as shown in Table 1. Second, I explain the different approaches, which are used to guarantee temporal memory safety in Table 2. My goal is to show the strengths and limitations of each technique.

- Finally, I conclude by highlighting opportunities for future work, given the restrictions of current solutions.

Figure 1: Timeline for memory safety exploitation techniques (marked with demons) and mitigations (marked with shields).
Table 1: Spatial memory protection techniques. Tools with no separate citations are covered in the Systematization of Knowledge papers [3–5] and/or Background section of the PhD dissertations [17, 18].

<table>
<thead>
<tr>
<th>Metadata Type</th>
<th>Whitelisting</th>
<th>Blacklisting</th>
<th>Randomized Allocators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Per-Object</td>
<td>Per-Pointer</td>
<td></td>
</tr>
<tr>
<td>Disjoint</td>
<td>Compatible C [19]</td>
<td>Mondrian <a href="Ch2">18</a></td>
<td>Purify</td>
</tr>
<tr>
<td></td>
<td>Baggy Bounds [22]</td>
<td>M-machine <a href="Ch2">18</a></td>
<td>Valgrind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Softbound <a href="Ch4">17</a></td>
<td>Dr. Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hardbound [23]</td>
<td>Electric Fence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Watchdog <a href="Ch5">17</a></td>
<td>ASan [24]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CUP [25]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intel MPX [26]</td>
<td></td>
</tr>
<tr>
<td>Inlined</td>
<td>EffectiveSan [27]</td>
<td>CHERI [18, 28]</td>
<td>SafeMem [29]</td>
</tr>
<tr>
<td>Metadata</td>
<td></td>
<td>Cyclone [30]</td>
<td>REST [31]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CheckedC [32]</td>
<td>Califoms [1]</td>
</tr>
<tr>
<td>Co-joined</td>
<td>ARM Memory Tagging [33]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata</td>
<td>SPARC ADI [34]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Metadata</td>
<td>Lowfat S/W [35]</td>
<td>Lowfat H/W [36]</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Temporal memory protection techniques. Tools with no separate citations are covered in the Systematization of Knowledge papers [3–5] and/or Background section of the PhD dissertations [17, 18].

<table>
<thead>
<tr>
<th>Solution Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garbage Collection (GC)</td>
<td>Regular Hardware Accelerated GC [37]</td>
</tr>
<tr>
<td></td>
<td>Conservative MarkUs [38]</td>
</tr>
<tr>
<td>Memory Quarantining</td>
<td>Valgrind, ASan [24], REST [31],</td>
</tr>
<tr>
<td></td>
<td>CHERIinvoke, and Califoms [1]</td>
</tr>
<tr>
<td>Lock &amp; Key</td>
<td>Explicit CETS <a href="Ch4">17</a>, CUP [25]</td>
</tr>
<tr>
<td></td>
<td>Implicit Electric Fence, Oscar [39]</td>
</tr>
<tr>
<td></td>
<td>Revoke key DangNull, DangSan, and BOGO [40]</td>
</tr>
</tbody>
</table>

References


Why is memory safety still a concern?


Why is memory safety still a concern?


Why is memory safety still a concern? PhD Candidacy Exam


