Toward Pointer Leasing for Use-After-Free Prevention

Wataru Hashimoto
The University of Tokyo
hashimoto@os.ecc.u-tokyo.ac.jp

Takahiro Shinagawa
The University of Tokyo
shina@ecc.u-tokyo.ac.jp

Extended Abstract

Low-level languages such as C and C++ are still widely used in various system software such as web servers and browsers. In these languages, memory management is the responsibility of programmers, leading to various memory-related bugs. One such bug is Use-After-Free (UAF), which occurs when a *dangling pointer*, a pointer that points to a freed object, is dereferenced. The number of UAF bugs in real-world software is increasing and its impact on system security is severe; UAF is ranked 7th in the 2022 Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses.

The challenge in UAF prevention is how to guarantee that freed and reallocated objects are not accessed through dangling pointers without compromising performance, comprehensiveness, and compatibility. Static approaches address this by detecting dangling pointers through source code analysis, but have difficulty in comprehensiveness due to the characteristics of low-level languages. Dynamic approaches include pointer invalidation [1], which detects dangling pointers with periodic memory sweeps, one-time allocation (OTA) [3], which allocates memory objects only once to avoid using dangling pointers, and new hardware [2], which exploits new processor's memory capabilities. However, they have problems with sweeping overhead, additional memory space overhead, probabilistic detection, or compatibility with existing systems.

We present LeaseMalloc, a combination of a compile-time instrumentation and a runtime library to address the challenge. The key idea of LeaseMalloc is the time-limited "leasing" of pointers to heap objects. To allow leased pointers to be used without checking until the lease expires, LeaseMalloc guarantees that heap objects pointed to by leased pointers will not be reallocated during the lease period even if they are freed. When the lease period has expired, LeaseMalloc checks the validity of the heap object; if the object has not been freed, the lease is renewed, otherwise it is considered a UAF bug. Leasing reduces performance overhead by removing the need of checking until the lease expires, while checking on lease renewal allows for comprehensive, rather than probabilistic, dangling pointers detection.

To achieve efficient pointer leasing while maintaining compatibility with existing systems, we introduce two techniques. The first is to embed lease expiration time information in the upper bits of pointers. Since the upper bits of pointers are not used in modern 64-bit processors, they can be utilized to embed new information without increasing the number of bits in pointers. In addition, Intel, AMD, and Arm are introducing processor extensions that ignore the upper bits of the pointer, allowing the expectation of efficient implementation by hardware.

The second is to introduce the notion of *Round* as information corresponding to the lease expiration time. Round is a monotonically increasing counter that is assigned to both pointers and objects. When allocating an object, the current Round value is set to both the pointer and the object, When the object is freed, the Round value of the object is incremented so that the Round value of the dangling pointer becomes smaller than the object. To compress the number of bits by slowing the pace of Round increase, freed objects are accumulated until a certain threshold is exceeded, as in the OTA technique, and the actual free is batch processed and the current Round value is incremented when the threshold is exceeded. This suppresses memory consumption in OTA while avoiding sweep costs in pointer invalidation.

We use LLVM's Pass Framework to implement the instrumentation of the target application. Since LeaseMalloc needs to interpose on almost all pointer dereferences in the target application, we will make several implementation optimizations. First, to reduce the access time to the current Round value, which is accessed each time during the pointer lease expiration check, we store the Round value in an unused register such as a debug register or an old floating-point register. Second, since UAF bugs are more likely to be introduced into applications than into libraries with stable implementations, we limit our instrumentation to applications and exclude libraries. However, there is an option to instrument libraries as well.

In summary, LeaseMalloc reduces performance overhead by leasing pointers using the pointer upper bits and the Round notion, provides comprehensive UAF prevention through pointer checking, and enables compatibility through a combination of compile-time instrumentation and run-time libraries. As a future work, we seek for more effective optimizations to reduce the overhead and compare the performance with previous works.

Acknowledgement: This work was supported by JST, CREST Grant Number JPMJCR22M3, Japan.

References

[1] Márton Erdős et al. MineSweeper: A "Clean Sweep" for Drop-In Use-after-Free Prevention. In *Proc. ASPLOS 2022*, 2022.

[2] Wesley Filardo et al. Cornucopia: Temporal safety for CHERI heaps. In *Proc. 2020 IEEE Symposium on Security and Privacy*, 2020.

[3] Brian Wickman et al. Preventing Use-After-Free Attacks with Fast Forward Allocation. In *Proc. 30th USENIX Security Symposium*, 2022.