Extreme LUD explainer

Yuki Shiino <yukishiino@chromium.org>

Overview

The Extreme LUD (Lightweight Use-after-Free Detector) is another variant of LUD, originally proposed in this doc.

As introduced in the design doc of LUD, we already have many UaF (Use-after-Free) detectors such as GWP-Asan, BRP, and LUD. Among them, the Extreme LUD aims to be extremely lightweight while giving up a lot of useful features. Maximizing the coverage (i.e. minimizing the runtime cost) is the primary focus for the Extreme LUD. The Extreme LUD is unique on this point. Other than that, the Extreme LUD is similar to the existing UaF detectors.

Basic mechanism

The basic mechanism is quite the same as LUD.

1. [Intercept] When a program deallocates an object, the Extreme LUD intercepts the deallocations on a sampling basis (e.g. 1% sampling) so that it will not regress the program performance much.
2. [Zap] The Extreme LUD zaps the memory chunk used for the object so that reading data from the (already-deallocated) object will likely crash.
3. [Quarantine] The Extreme LUD quarantines the object for some time so that the memory chunk will not be re-used for another object.
4. [Dequarantine] When the quarantine capacity is full, the Extreme LUD de-quarantines some of the quarantined objects (i.e. actually deallocates the memory and returns it to the underlying memory allocator, allowing it to be allocated again).

The original LUD does collect a stack trace in addition when it intercepts a deallocation so that debugging will be easier. The Extreme LUD focuses on the runtime performance and doesn't collect any additional data in the first version. (There is a future plan to collect the type information of the being-deallocated object if we can implement it without regressing the performance much.)

The crash reports caused by the Extreme LUD will contain the crashing address specific to the zapping pattern of the Extreme LUD. So, we can tell whether a crash comes from the Extreme LUD or not. When it's from the Extreme LUD, it's likely to be a UaF bug or OOB (Out-Of-Bounds) bug.

Security considerations

The Extreme LUD should have no negative impact on security, and the positive impact is minimal.

Even without the Extreme LUD or any sort of UaF detectors, a program with a UaF bug has a chance to crash when a deallocated memory chunk is re-used for a new object and the memory is overwritten with a new value and then the program reads the overwritten memory as UaF. The Extreme LUD just increases the chance to crash by zapping the memory immediately.

The quarantine doesn't change the game either. A deallocated memory chunk may or may not be re-used immediately. It highly depends on a memory usage pattern of the program and an algorithm of its memory allocator. The quarantine just makes the time until a memory chunk gets re-used longer so that the program has more chances to crash. This is meant to catch incidental accesses, but can be easily worked around by an attacker.

The goal of the Extreme LUD is not to mitigate any security issue, but to detect UaFs. It doesn't really affect security.

Privacy considerations

The Extreme LUD does not collect any user data except for UMA histograms (how many objects are quarantined, how long time objects are quarantined, etc.). The Extreme LUD has no impact on privacy.

List of UMA metrics (source)

Name[1]	What
Count	How many objects the quarantine holds currently.
SizeInBytes	How many bytes in total the quarantine holds currently. (The total byte size of the all quarantined objects.)
CumulativeCount	How many objects in total have been quarantined in the process lifetime. (Some objects have already been dequarantined.)
CumulativeSizeInBytes	How many bytes in total have been quarantined in the process lifetime.
QuarantineMissCount	How many times we gave up quaranting objects due to insufficient capacity of the quarantine in the process lifetime.
BytesPerMinute	How many bytes of objects are quarantined in the last minute on average.
CountPerMinute	How many objects are quarantined in the last minute on average.

[1] "Malloc.ExtremeLUD." prefix is omitted. "Count" is actually "Malloc.ExtremeLUD.Count".

MissCountPerMinute	How many objects are *not* quarantined due to the capacity constraint in the last minute on average.
QuarantinedTime	How long an object will be quarantined until it gets dequarantined due to the capacity constraint (estimation based on BytesPerMinute).