DieHard: Probabilistic Memory Safety for Unsafe Languages

Emery D. Berger and Benjamin G. Zorn

PLDI 2006

Presented by Uri Kanonov
23.02.2014

The article presented a new and intriguing approach titled "Probabilistic Memory Safety". The notion represents the idea of providing runtime safety for programs probabilistically rather than deterministically. Such an approach has many advantages, primarily its unpredictability when faced with an attacker trying to break into a system utilizing it.

The implementation, namely DieHard, is a custom heap allocator that overrides **malloc** and **free** (at runtime using LD_PRELOAD) and thus provides the application a more secure heap. In practice, DieHard allocates a heap ***M*** times larger than required, partitions the memory into regions by object sizes and allocates objects at random inside each region up to ***1/M*** capacity. Such an approach probabilistically helps defend against **buffer overflows** and **dangling pointers**. DieHard can also run in **"replicated"** mode to detect **uninitialized reads**.

DieHard was analyzed theoretically and evaluated practically and shown to be able to contain multiple kinds of memory errors. Although it incurs an overhead both runtime and memory wise, it is suitable for use with general purpose applications.

From my perspective, DieHard is a fine example of how randomness can be utilized in memory management, although one must take into account its impact on performance. On the other hand, one must remember that simply randomizing allocations will not deter a determined attacker from breaking into a system.
As an alternative solution I presented the notion of **heap cookies** (a similar solution to stack canaries) and its implementation in iOS 6. Such a solution provides relatively good protection against heap-based buffer overflows at a low runtime cost.

During the discussion after the presentation the following points came up:

- Regarding DieHard
    - It can be used as an additional tool for tests
    - Seems to address the symptom rather than the problem
    - The algorithm's randomness becomes less effective as the heap gets fuller
    - One should consider on-demand allocation of heap memory to save on memory costs
    - Due to its performance impact it is not suitable for soft/hard RealTime systems
    - Although Valgrind is better suited for testing, DieHard can utilized during product deployment

- Regarding Heap Cookies
    - The Heap Cookie algorithm (unlike DieHard) is mostly deterministic and thus more vulnerable
    - Conversely, it requires the attacker to utilize an additional vulnerability to learn the cookies