By Gene Novark, Emery D. Berger and Benjamin G. Zorn
Presented by Matthew Kent

Exterminator: Automatically Correcting Memory Errors with High Probability

Contents

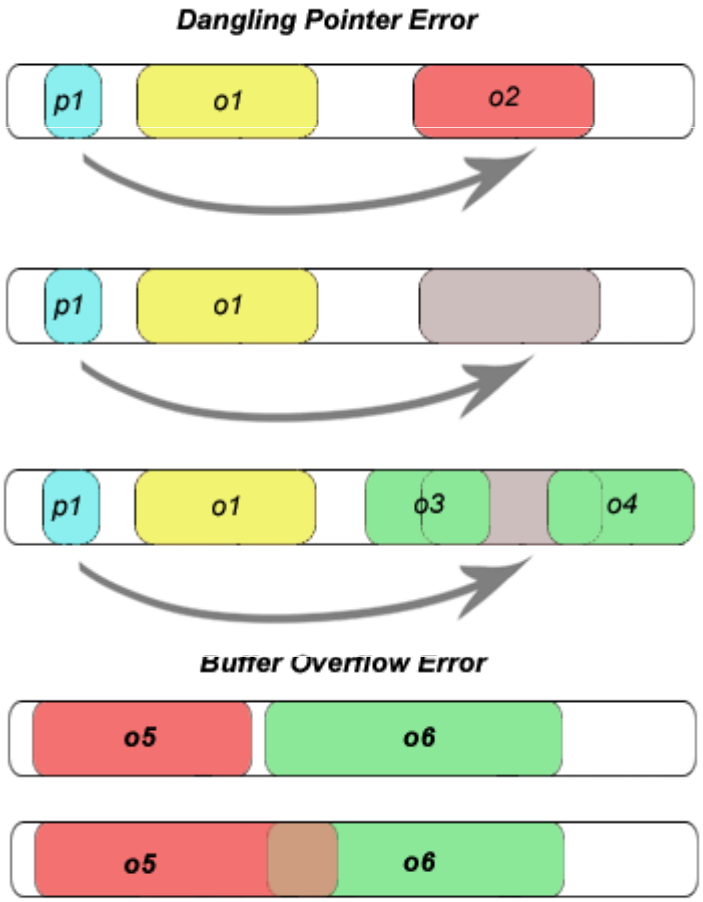
- Overview
- Target memory errors
- Modes of operation
- Components of Exterminator
- Heap architecture
- Debugging allocator
- Error isolation
- Error correction
- Results found
- Conclusion

Overview

- Manual memory management in C and C++ results in vulnerability to various memory errors.
- In particular Dangling-Pointers, Buffer Overflows.
- Traditional debugging methods are costly and time consuming.
- Symantec: avg time between discovery and patch deployment for an exploitable bug is 28 days.
- The author presents Exterminator, a runtime system that detects, isolates & corrects two classes of memory error.

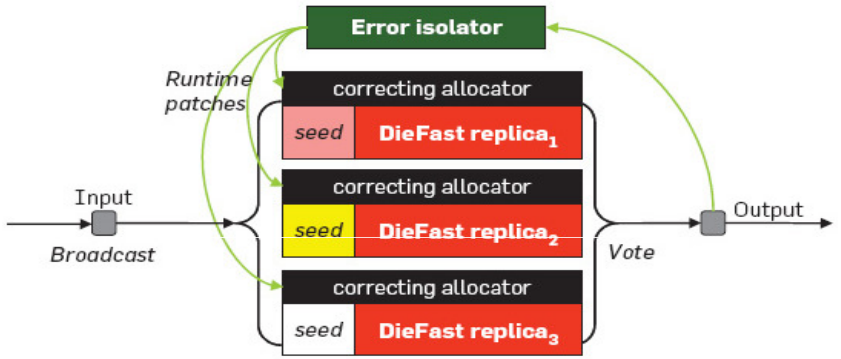
Target Memory Errors

- Dangling pointers and Buffer-overflow errors are the most commonly exploited memory errors.
- Previous works could detect and tolerate these. Exterminator aims to correct them.



Modes of Operation

- Iterative mode:
  - Sequentially replays the program, each time with the same input but using different random-seeds.
- Replicated mode :
  - Monitors concurrent executions with different random-seeds.



- Cumulative mode:
  - Gradually builds up statistical data over normal runs of the program.

Components of Exterminator

- Exterminator features 3 core components:
  - Probabilistic debugging allocator.
  - Probabilistic error isolation algorithm.
  - Correcting memory allocator.
- The Probabilistic Debugging allocator:
  - Ensures the heap is always M times larger than max 'needed' (1/M ratio).
  - Does this using 'mini-heaps' .
  - Randomised Bitmaps tests allow for $O(1)$ allocation/de-allocation.
  - Ensures:
    - Probable that overflows go into free space.
    - Probable that freed objects aren't immediately reused.

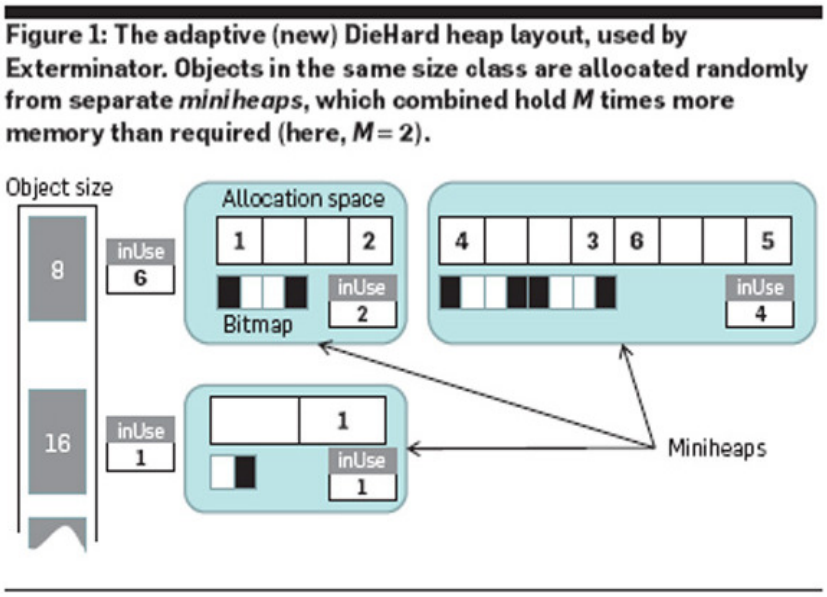
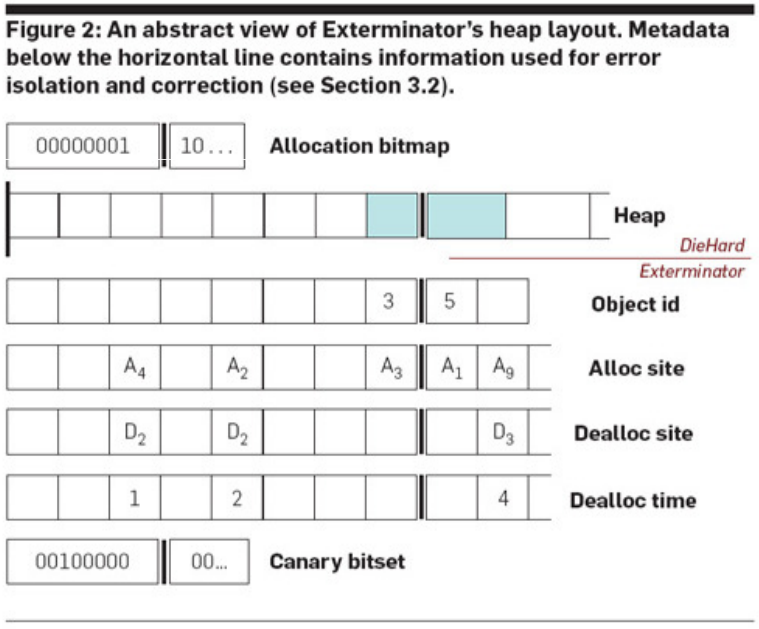


Figure 1: The adaptive (new) DieHard heap layout, used by Exterminator. Objects in the same size class are allocated randomly from separate *miniheaps*, which combined hold *M* times more memory than required (here, *M* = 2).

Heap Architecture

- Object ID's are sequential (an ID of *n* indicates the object as the *n*th object allocated).
- 'Site' information fields capture the calling contexts.
- Canary bit indicates if the object location was filled with canaries.
- This data persists from allocation, until an object is allocated in its place.
- This adds a memory overhead of 16 bytes and 2 bits per allocated object.



Debugging Allocator

- 32bit randomised canary padding:
  - **Buffer overflows** detected when canary ‘fence-posts’ are overrun.
  - **Dangling pointers** detected when canaries are read, and the program throws an error.
    - May match true data values; in '*cumulative mode*' Exterminator only sets canaries with probability *P*.
  - When Exterminator allocates memory it checks canary integrity (based on bit-set).
    - If not intact then it signals an error and isolates this memory.
  - When Exterminator de-allocates memory, it also checks neighbours. If free it performs the same canary-check.

Error Isolation

In Iterative and Replicated modes

- Buffer overflows:
  - Compare objects across heaps (by Object ID). They should be *identical*.
  - If they are not identical, they are considered a candidate **'victim'** of an overflow.
    - Some exceptions (pointers, handles).
  - For each candidate-victim found Exterminator then examines heap images to find a matching '**culprit**' object.
    - Must be the same distance δ, from the victim within each heap-image.
  - Once a culprit has been deduced, overflow size is recorded as max-victim-δ.
- Dangling pointers:
  - Under these modes, only read & write dangling pointers are found.
    - Would require >20 replicas for read only dangling pointers.
  - When a **freed** object is overwritten with the same values across all heap-images, classified as a dangling pointer error.

Error Isolation

In Cumulative mode

- Approach is to consider exceptional program events (crashes) and generate statistics.
- **Dangling pointers:**
  - To force different runs to respond differently, de-allocated objects are only filled with canaries with a probability *P*.
  - The choice of *P* reflects a trade-off between dangling pointer isolation, and buffer overflow isolation (converse effects).
- **Buffer overflows:**
  - Identifies corruption by looking for overwritten canary values.
  - For each allocation site, computes probability that the 'culprit' is within.
  - Combines these independent statistics from each run to deduce sites that consistently appear as candidates for housing the culprit.
    - Deduces the culprit from these sites.

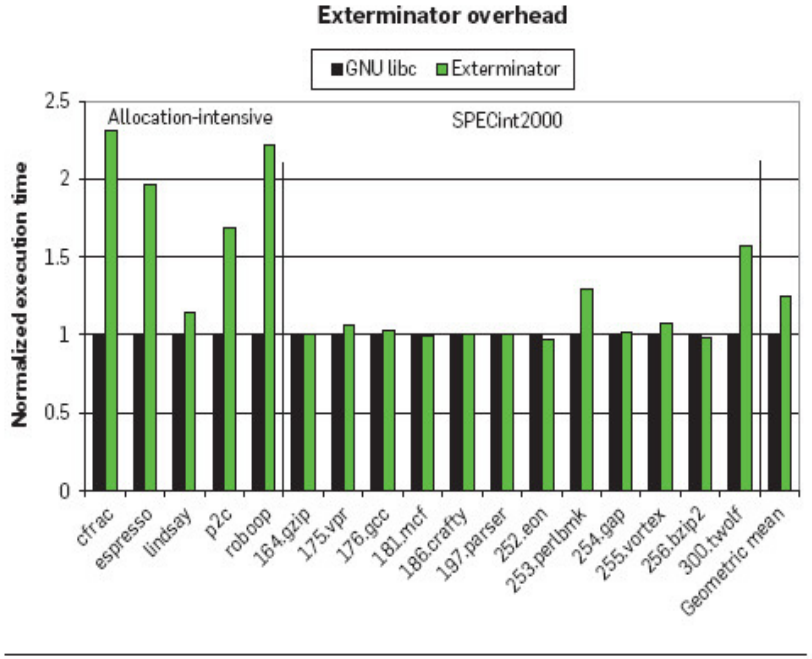
Error Correction

- Buffer overflow correction
  - Runtime patch contains allocation-site hash and amount of padding needed to contain the overflow (δ + size of the overflow). If a patch for the allocation site already exists, the Max is used.
- Dangling Pointer Correction
  - Runtime patch contains allocation/de-allocation-site hash and amount of time by which to delay de-allocation:
    - Let τ be the recorded de-allocation time of this object. T be the time at which the program crashed or corruption was detected.
    - Delay will be: $2 \times (T - \tau) + 1$.

Results Found

- Exterminator degrades performance by from 0% to 132%.
- Overall geometric mean of 25.1%.
- Most extensive across allocation-intensive suite (81.2% geometric mean).
- Less pronounced across SPECint2000 suite (7.2% g.m.).

Figure 4: Runtime overhead for Exterminator across a suite of benchmarks, normalized to the performance of GNU libc (Linux) allocator.



Results Found

Injected Errors

Test Type	Iterations Taken	False-negative rate
Buffer-overflows	3	0%

Test Type	Successful Runs	Iterations Taken
Dangling Ptr. (iterative-mode)	4/10	1
Dangling Ptr. (cumulative-mode)	N/A	22-30

Real Errors

Software	Problem	Result
Squid Web Cache	Input-dependant overflow Leads to crash	6-byte pad generated Error corrected
Mozilla Firefox	Overflow Cumulative mode only	23 runs with quick-trigger 34 runs with prolonged trigger Error corrected

Conclusion

- Exterminator automatically detects and corrects runtime heap-based memory errors in C/C++ programs, to a high probability.
- Consists of:
  - Probabilistic debugging allocator.
  - Probabilistic error isolation algorithm.
  - Correcting memory allocator.
- Provably low false-positive and false-negative rates.
- Runtime patches are generated that the correcting-memory-allocator corrects memory errors with.
- Exterminator is useful during testing **and** in deployed applications.