UMassAmherst

Exterminator: Automatically Correcting Memory Errors

Gene Novark, Emery Berger — UMass Amherst

Ben Zorn — Microsoft Research

Debugging Memory Errors

- Billions of lines of deployed C/C++ code
- Apps contain memory errors
  - Heap overflows
  - Dangling pointers
- Notoriously hard to debug
  - Must reproduce bug, pinpoint cause
  - Average 28 days from discovery of *remotely exploitable* memory error and patch [Symantec 2006]

Coping with memory errors

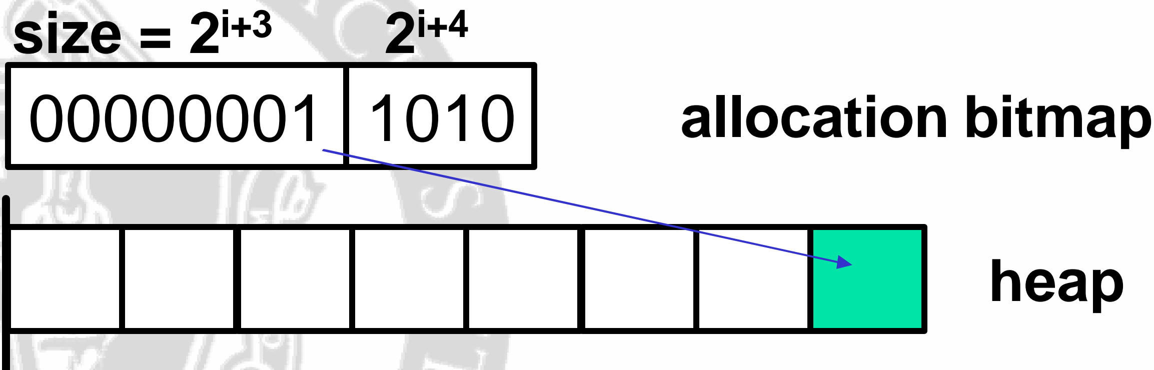
- **Unsound, *may* detect errors**
  - Windows, GNU libc, Rx
- **Sound, *always* finds dynamic errors**
  - *CCured, CRED, SAFECode*
    - Requires source modification
  - Valgrind, Purify
    - Order of magnitude slowdown
- Probabilistically **avoid** errors
  - DieHard [Berger 2006]

- **Exterminator**: automatically **isolate** and **fix** detected errors

DieHard Overview

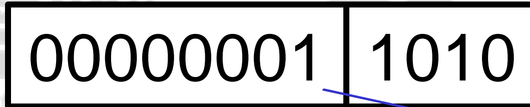
- **Fully-randomized** memory manager
  - Bitmap-based with random probing
  - Increases odds of **benign** memory errors
  - Different heap layouts across runs
- **Replication**
  - Run multiple **replicas** simultaneously, vote on results
- **Increases reliability (hides bugs) by using more space**

DieHard Heap Layout

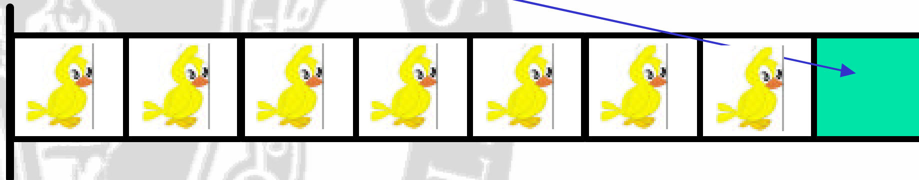


- Bitmap-based, **segregated** size classes
  - Bit represents one **object** of given size
    - i.e., one bit = 2^{i+3} bytes, etc.
- **malloc()**: randomly probe bitmap for free space
- **free()**: just reset bit

Exterminator Extensions



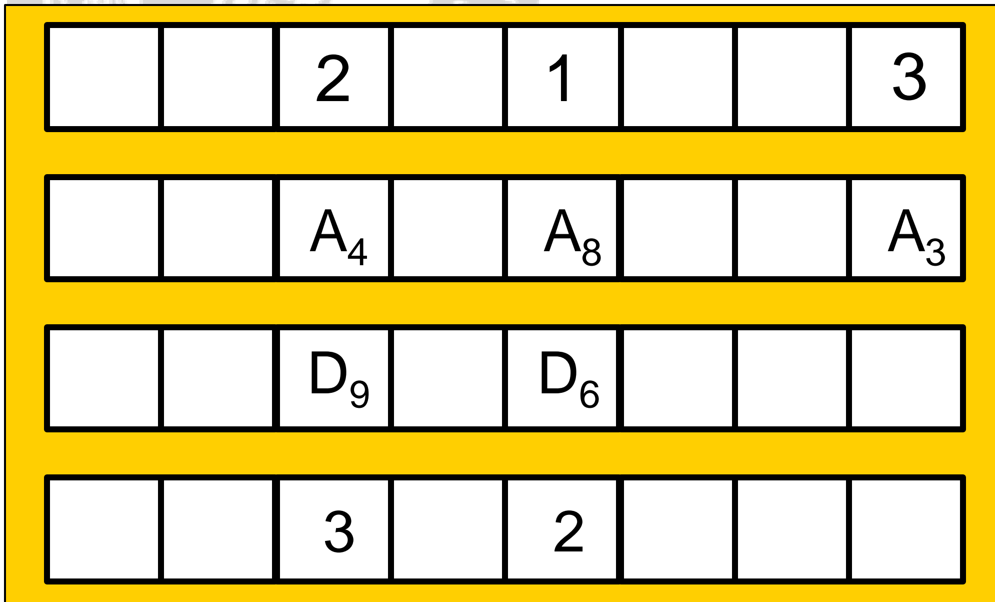
allocation bitmap



heap

DieHard

Exterminator



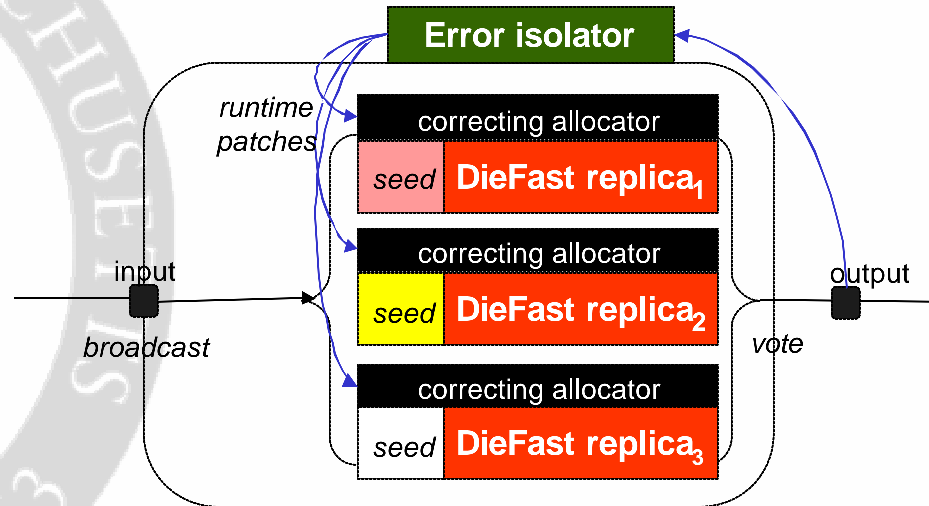
object id (serial number)

alloc site

dealloc site

dealloc time

The Exterminator System



- On failure, create **heap images** (core dump)
- **Isolator** analyzes images, creates **runtime patch**
- **Correcting allocator** corrects isolated errors:
  - pad allocations
  - extend object lifetimes

Exterminator Isolation Algorithm

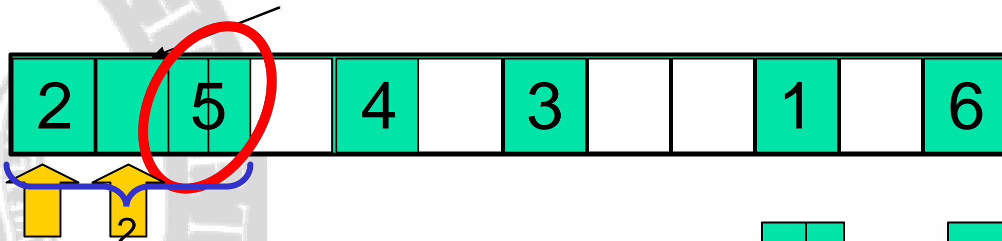
- Identify **“discrepancies”**
  - Compare valid object data
    - Find equivalent objects (same ID) with **different contents**
  - Find **corrupted canaries** (free space)
- Check for possible **buffer overflows**
- Check for **dangling pointer error**

Comparing Object Data

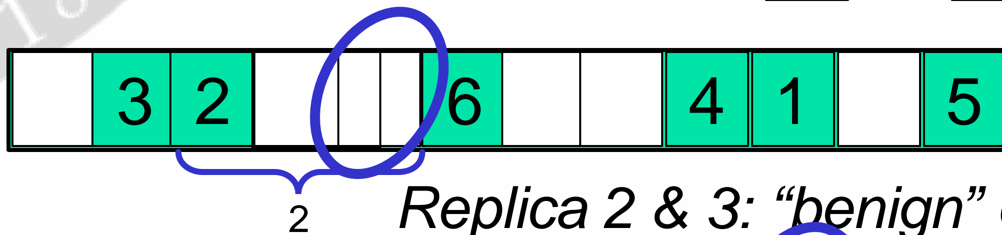
- Lots of valid reasons for data to differ
  - Pointers (random target locations)
  - File descriptors
  - Non-transparent use of pointers
    - e.g. Red-Black tree keyed on pointer value
  - Etc.
- Exterminator identifies and ignores:
  - Values which differ across **all** replicas
  - **Valid pointers** referring to same target ID

Error Isolation: Buffer Overflows

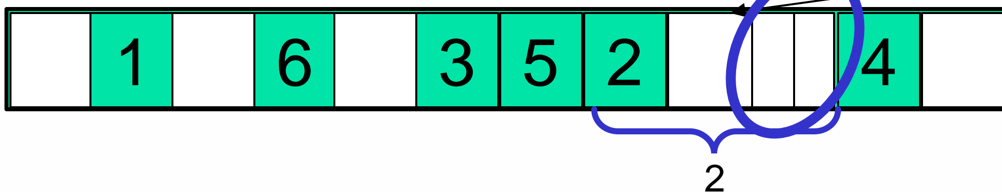
Replica 1: "malignant" overflow



1. Identify corrupt object (5 vs. 5 & 5)
2. Search for source (δ = 2: 2 candidate!)
3. Compare data at same δ (5 vs. & Match!)



Replica 2 & 3: "benign" overflows



Error Isolation: Dangling Pointer

Freed, Canary value

2	4			5		3			1		6

Dangled ptr

?

	1		6		3		2		5	4	

- Corrupted canary values for object 5
- Same object, same corruption
- Buffer overflow?
- Assume dangling pointer
  - Source object would be at same δ in all replicas
  - Extend lifetime of object
    - Unlikely, $\Pr = (\frac{1}{H})^k$

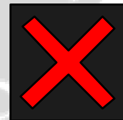
Error Isolation: Dangling ptr read

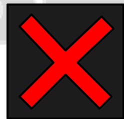
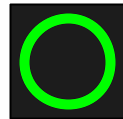
- What if the program doesn't write to the dangled pointer?
  - DieFast overwrites freed objects
  - Canaries produce invalid reads, crashes
  - How to identify prematurely freed objects?
    - Common case 1: read something that was a pointer, dereference it
    - Common case 2: read numeric value, error propagates through computation
    - No information: previous contents destroyed!

Error Isolation: Dangling ptr read

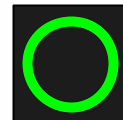
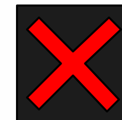
- Solution: Write canaries **randomly** (half the time)
  - Equivalent to extending object lifetime (until overwritten)

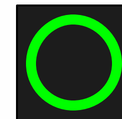
Legal free:

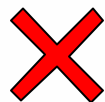
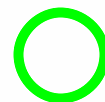
OK  OK

 OK  OK

Illegal free:
(later read + deref ptr)

 OK  CRASH!

CRASH!  OK

 : overwritten with canaries  : data intact

Error Isolation: Dangling ptr read

- Correct frees uncorrelated with crash

$$\Pr(crash \mid canaried[i]) = \Pr(crash)$$
$$\therefore \Pr(crash \equiv canaried[i]) = 0.5$$

- For each object *i*, compute estimator:

$$P = \frac{\sum (crash \equiv canaried[i])}{replicas}$$

- *P > 0.5*: dangling pointer error
  - Create patch when confidence reaches threshold

Runtime Patches

- Overflow patches
  - Allocation callsite
  - Overflow amount
- Dangling pointer patches
  - Allocation & Deallocation callsites
  - Lifetime extension

Correcting Allocator

- Extended DieHard allocator
- Reads runtime patches
- Stores **pad table** & **deferral table**
- On free:
  - Check for life extension for current object
    - Place ptr, time on **deferral priority queue**
- On allocation:
  - Check for overflow fix for current callsite
  - Check deferral queue for pending frees

Results

- Analytical results
- Empirical results
  - Runtime overhead
  - Error detection
    - Injected faults
    - Real application (Squid)

Analytic results summary

- Buffer overflows
  - False negative & positive rate decrease exponentially with # of replicas
- Dangling pointers
  - Write: exponentially low false +/- rate
  - Read-only:
    - Confidence threshold controls false positive rate, # replicas needed to identify culprit

Empirical Results: Runtime

Exterminator Overhead

GNU libc | Exterminator

Normalized Execution Time

2.5
2
1.5
1
0.5
0

allocation-intensive

SPECint2000

cfrac
espresso
lindsay
p2c
roboop
164.gzip
175.vpr
176.gcc
181.mcf
186.crafty
197.parser
253.perlbmk
254.gap
255.vortex
256.bzip2
300.twolf
Geometric mean

Empirical Results: Overflows

Buffer Overflow Isolation

3 images
4 images
5 or more

100%
80%
60%
40%
20%
0%

4
8
16

Overflow Size

Empirical Results: Dang. Ptrs.

Dangling Pointer Isolation

Number of Images	Value
3	30%
19	60%
failed	10%

Empirical Results: Squid

- Squid web cache heap overflow
  - Remotely exploitable
  - Crashes glibc 2.8.0 and BDW collector
- DieFast detects error immediately
  - Corrupted canary past overflowed object
- Exterminator's isolator generates an object pad of 6 bytes, fixing the overflow

Conclusion

- Randomization + Replication = Information
  - **Randomization Þ** bugs have *different* effects
  - Exterminator exploits different effects across heaps to isolate cause
- Low overhead
  - **Automatically fix bugs** in **deployed** programs
  - Breaks crash-debug-patch cycle
    - Create 0-day patches for 0-day bugs

Questions?

- http://www.cs.umass.edu/~gnovark/