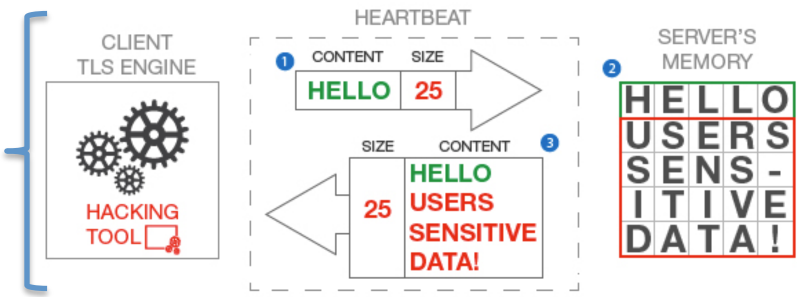
GUARDER: A Tunable Secure Allocator

Sam Silvestro, Hongyu Liu, Tianyi Liu,
Zhiqiang Lin*, Tongping Liu

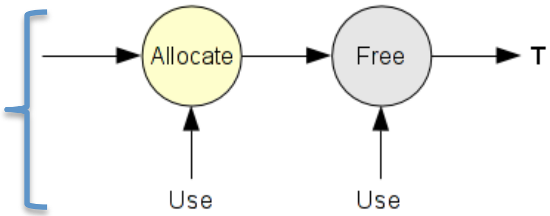
University of Texas at San Antonio
* The Ohio State University

Common Heap Vulnerabilities

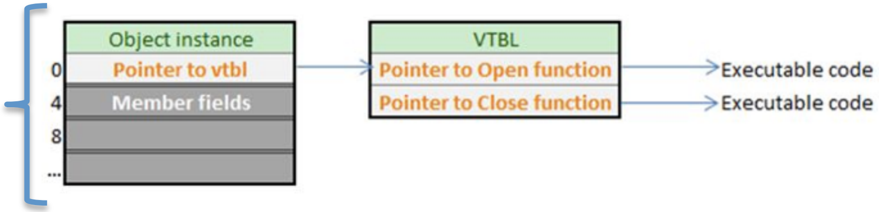
- Buffer over-read
  - Information leakage
    - e.g., Heartbleed



- Use-after-free



- Buffer overflow



- Double / invalid free
  - Unexpected results, program crash, hijacked control flow

Heap Vulnerabilities Reported in NIST Database

Heap Vulnerabilities	Occurrences (#)
Heap Over-reads	125
Heap Over-writes	673
Use-after-frees	264
Invalid-frees	35
Double-frees	33

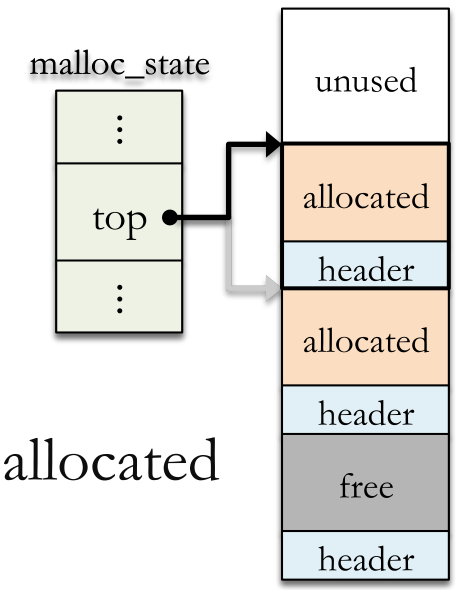
Many vulnerabilities were reported just in the past year!

Defending Heap Vulnerabilities

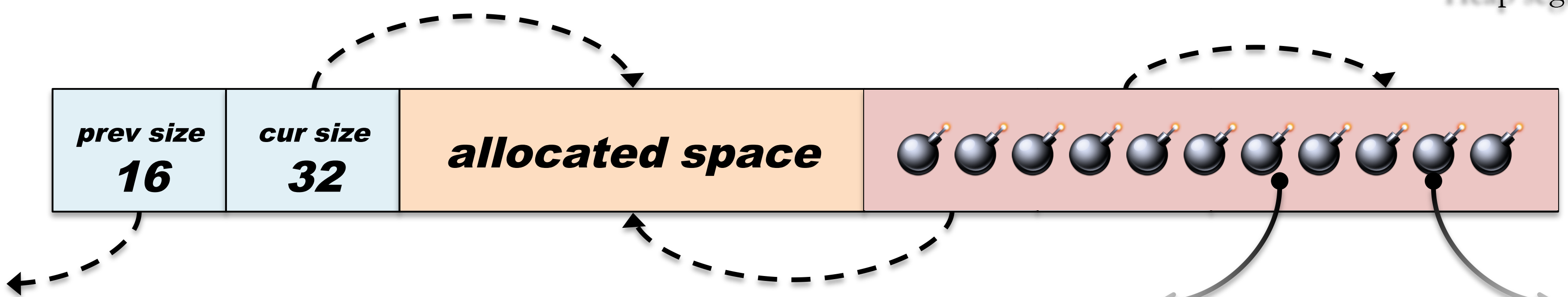
- Detect bugs with automated tools, e.g. Coverity, ASan
  – Overhead issue, completeness, false positives
- Rewrite code using a safer language, e.g. Java
  – Huge amount of effort and performance issue
- Prevent code execution
  – Cannot handle return-to-libc or ROP attack
- Sanity check on execution flow, e.g. CFI
- Secure heap allocator, e.g. randomization

Default Linux Allocator

- Designed to perform well
  - Bump pointers + freelists
  - Not designed for security purposes
- Prepends metadata before actual objects
- Provides no randomization
  - Result: easy to determine when an object will be allocated
- Poor handling of double/invalid frees



Heap segment



An example of Linux allocator's object metadata placement

Existing Secure Allocators:
OpenBSD, DieHarder, and FreeGuard

- Each are BIBOP-style secure allocators
  - "Big Bag of Pages"
  - Each "bag" of pages holds objects of a specific size class
  - Metadata are separated from the actual heap

- All feature randomization
  - DieHarder = log *n* bits of entropy
  - OpenBSD = 2 ~ 10 bits
  - FreeGuard = 2 bits

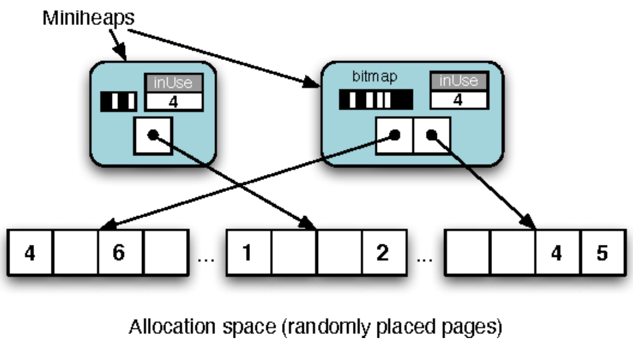
- Some impose high performance overhead
  - OpenBSD ≈ 31%
  - DieHarder ≈ 74%, up to 9.2X

Entropy of Existing Secure Allocators

1. Exhibit either low or unstable entropy
2. Unstable entropy dependent on size class, execution phase, inputs, and applications

DieHarder's Security Issue

- Always selects one object randomly, among all available objects
  - May take extended period before search is successful
  - Not reliable ➔ unstable entropy
- Worse: security is bound to its specific design, which is not flexible



Design Purpose

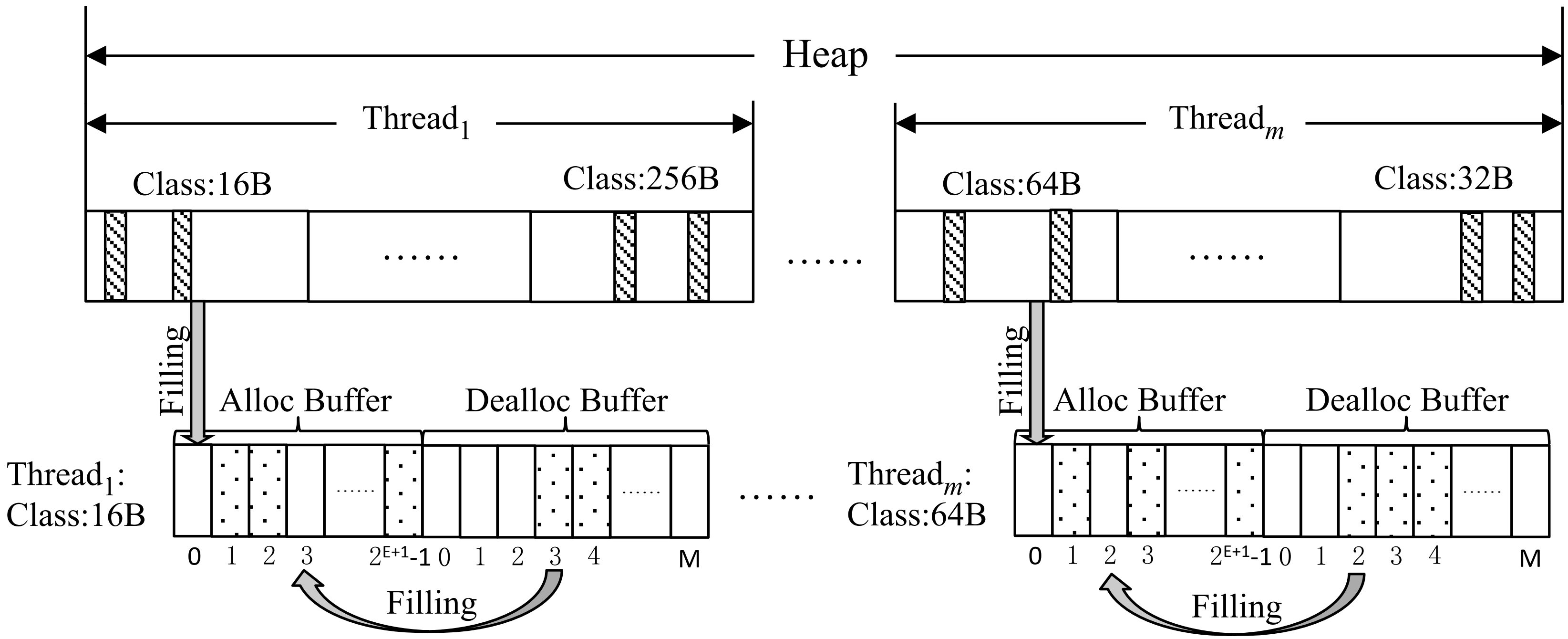
- Reliable Security
  - Stable allocation entropy across:
    - Size classes
    - Inputs
    - Execution phases
    - Applications
- Tunable security
  - User may specify the bits of entropy
  - Balances performance budget with security needs

Supplying the Specified Entropy

- We could use a simple array as the object buffer
  - 1 out of 256 objects = 8 bits of entropy
- Challenges with this approach:
  - How to handle deallocations?
    - How to efficiently find space to reinsert freed objects?
  - How to avoid repopulating array after every allocation?
    - < 256 objects ➔ < 8 bits of entropy
  - How to avoid excessive checking cycles?
    - Upon allocations, probability of choosing empty slot

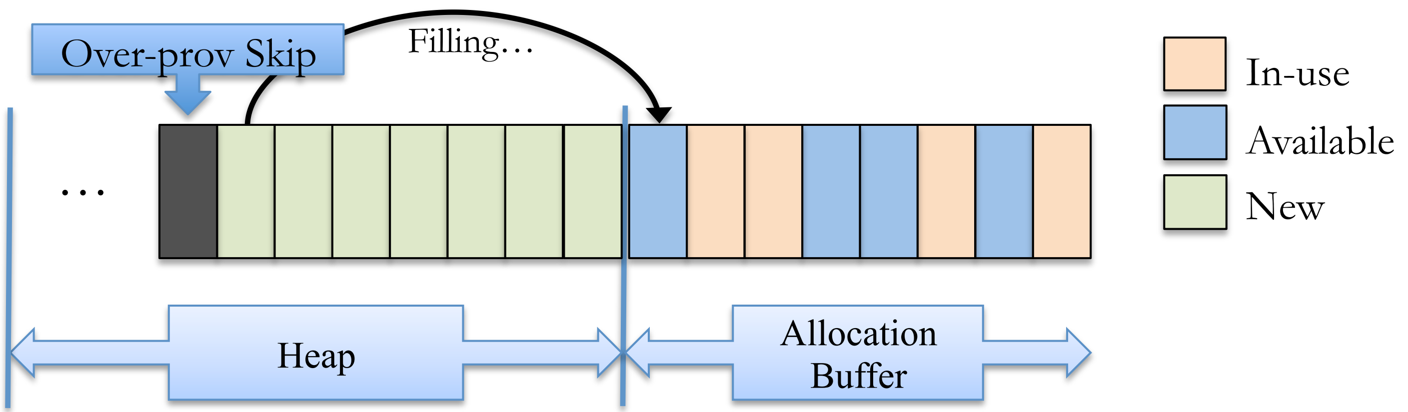
Combining Allocation and Deallocation Buffers

- Provides minimum of *E*-bits of user-specified entropy
  - Every thread has pair of allocation and deallocation buffers per size class
  - Allocation buffer holds 2^{E+1} objects
    - To never fall below half full ensures minimum E bits entropy
  - Allocation buffer is filled from top of heap if no freed objects are available



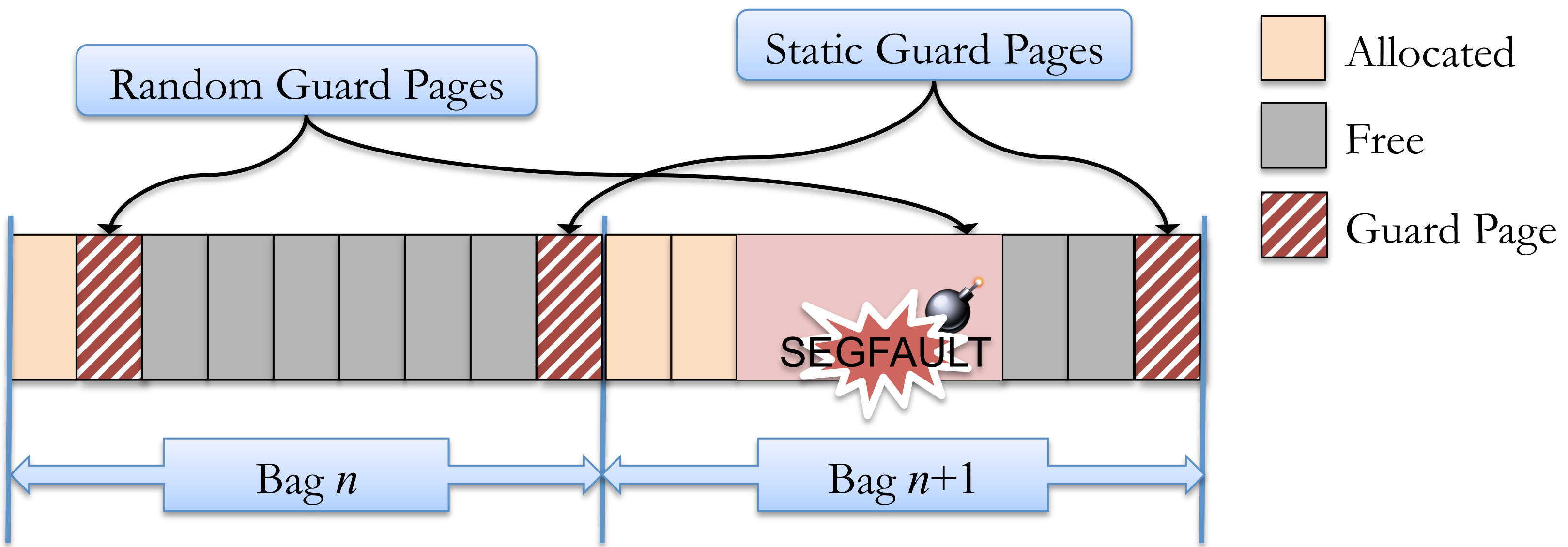
Tunable Security – Overprovisioning

- Dedicates a portion of heap objects as "never use"
  - Guarder's default factor = 1/8
  - Thus, each object has 1/8 probability of being excluded from future use
- Helps thwart buffer overflow
- During allocation buffer filling step:
  - 1/8 of objects will be selected randomly for exclusion
  - Corresponding slot marked empty
  - Remaining 7/8 of non-empty slots will be pulled into allocation buffer



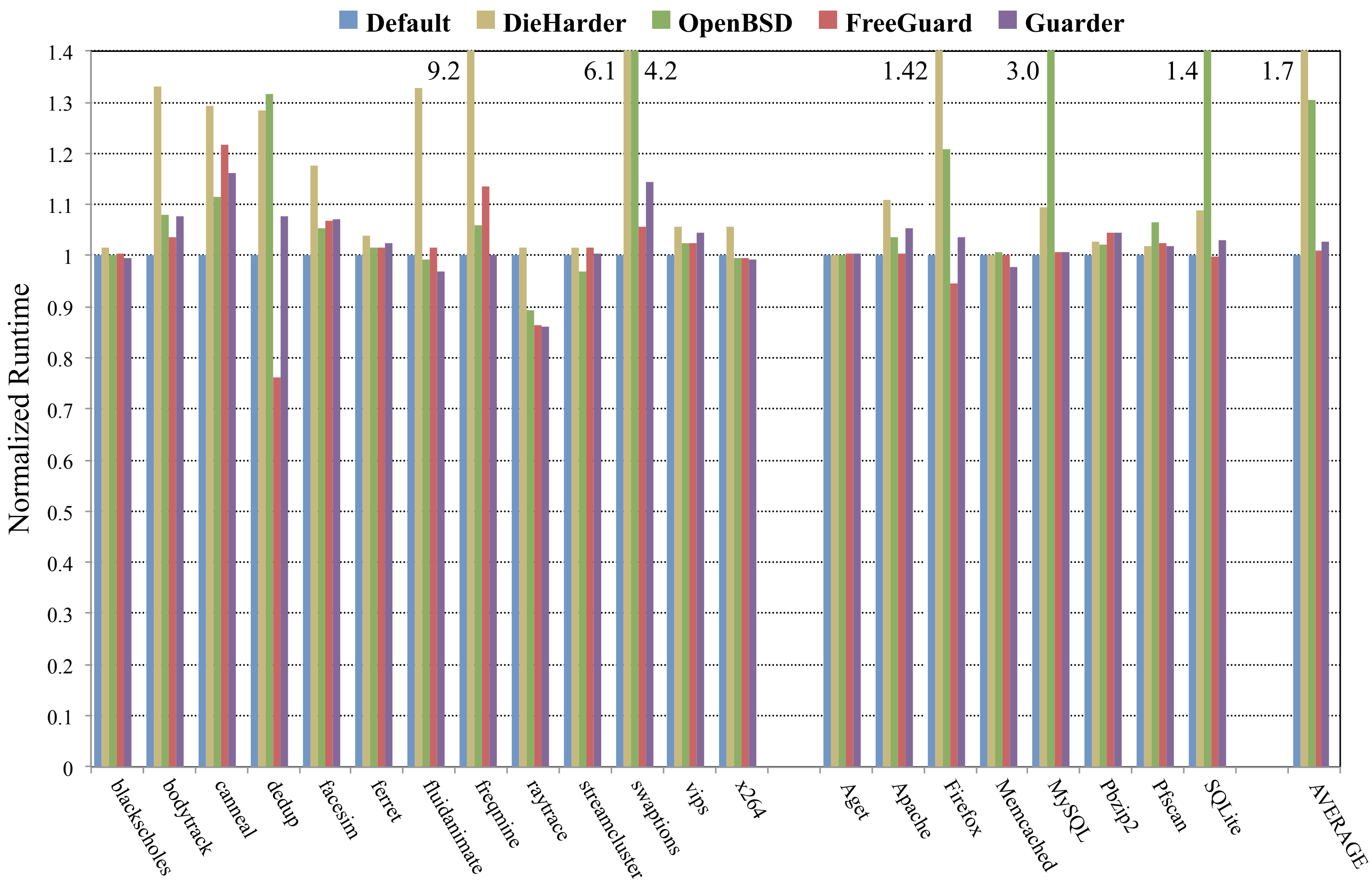
Tunable Security – Custom Guard Pages

- Guard pages: cannot be accessed
  - Helps prevent heap spraying, buffer overflow attacks
  - Guarder's default proportion = 10%
  - During buffer filling, the given proportion of pages are marked as guard pages



Performance Evaluation

- 21 applications evaluated
  - PARSEC
  - 8 real-world
- < 3% overhead, on average (arithmetic mean)



All values normalized to performance of default Linux allocator

Performance Evaluation

– Two reasons why Guarder performs faster
  - Avoids use of central lock
  - Due to the following design

Trials		DieHarder	OpenBSD	FreeGuard	Guarder
Allocation	Average	1.99	3.79	1	1.77
	Maximum	93	45	1	131
Deallocation	Average	12.40	1	1	1
	Maximum	141	1	1	1

Data collected with Guarder's default tunable parameter of 9 bits of entropy.

Security Feature Comparison

Security Feature	Linux	DieHarder	OpenBSD	FreeGuard	Guarder
Fully-segregated metadata		✓	✓	✓	✓
Randomized allocation		✓	✓	⊖	✓
Guard pages		⊖	✓	✓	✓
Check overflows on free			⊖	✓	✓
Over-provisioned allocation		✓			✓
Detects double/invalid frees	⊖	✓	⊖	✓	✓

✓ indicates the allocator has this feature
⊖ indicates the implementation has some weakness

- Guarder provides the most complete feature-set as compared to existing works
- Provides the strongest guarantee with respect to randomization entropy

Entropy of Secure Allocators

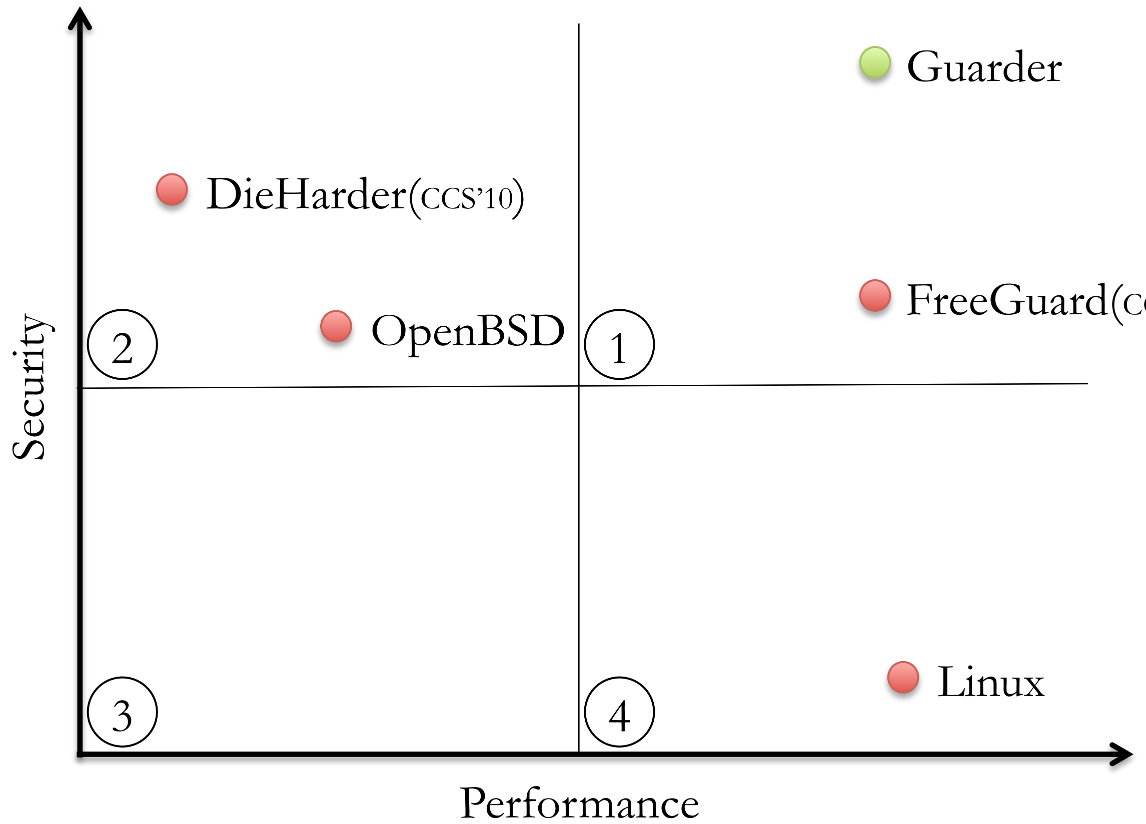
- Guarder exhibits reliable entropy
- Allows users to specify the entropy (e.g., 9 bits here)

Why Tunable Matters?

Entropy (bits)				*GPR=10%, OPF=1/8*
8	9	10	11	12
1.003	1.000	1.016	1.031	1.047
Guard Page Ratio				*EB=9, OPF=1/8*
2%	5%	10%	20%	50%
0.987	0.990	1.000	1.016	1.046
Over-provisioning Factor				*EB=9, GPR=10%*
1/32	1/16	1/8	1/4	1/2
0.998	0.995	1.000	1.001	1.011

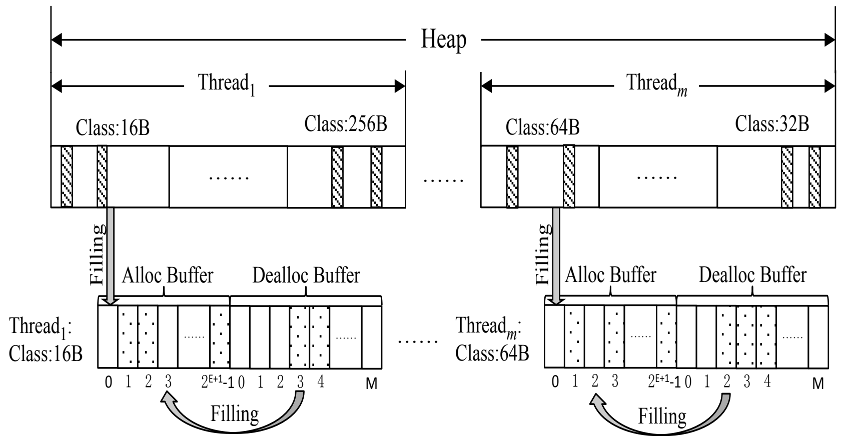
- Values normalized to default settings
- Higher security indicates higher overhead

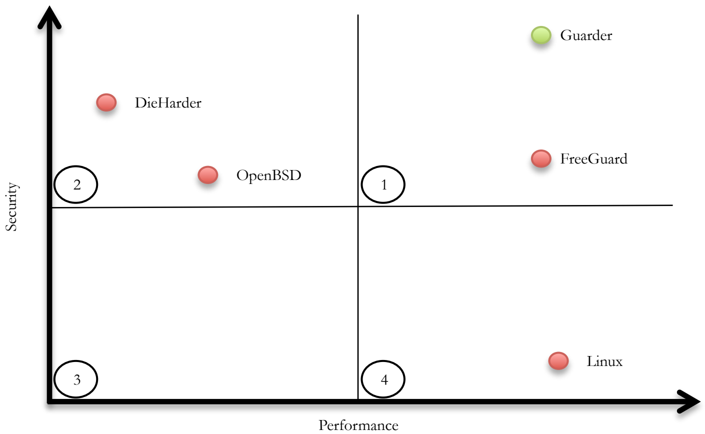
Comparison of Existing Security Allocators



Conclusion

- GUARDER is a tunable secure heap allocator
  - Tunable security allows users to choose their security based on their performance budget
  - Reliable security provides a stable entropy level across size classes, inputs, execution phases, and applications
  - The allocation buffer design facilitates other tunable security features, heap over-provisioning and random guard pages
  - Implements greatest feature set compared to other evaluated allocators
- GUARDER provides reliable, tunable security with < 3% performance overhead





Guarder can be downloaded at https://github.com/UTSASRG/Guarder

The work is also supported by Mozilla