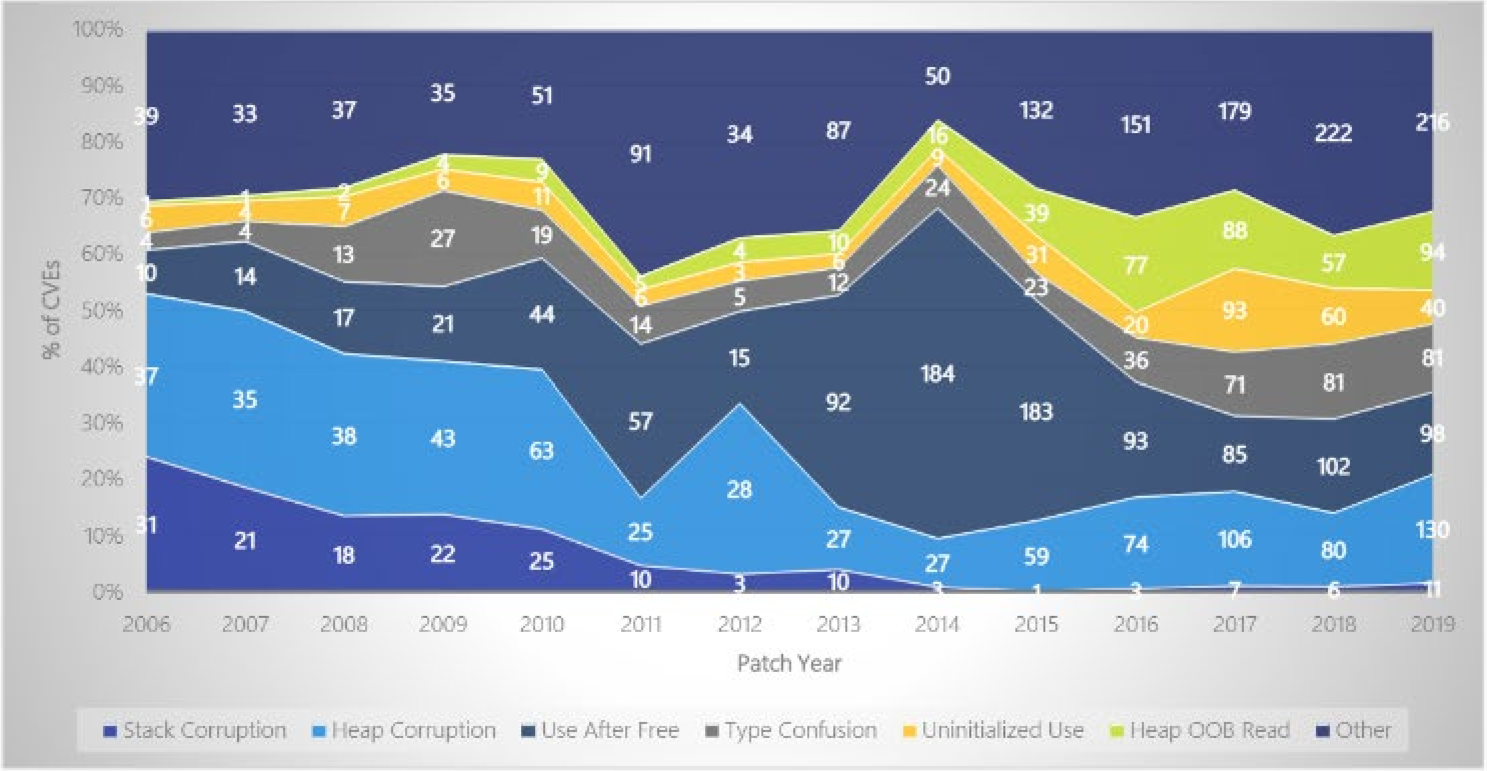
HardsHeap: A Universal and Extensible Framework for Evaluating Secure Allocators

Insu Yun, Woosun Song, Seunggi Min(KAIST),

Taesoo Kim (Georgia Institute of Technology)

Heap vulnerabilities are serious



Top vulnerability classes in systems software[1] at Microsoft (2016 through 2019)

#1 – heap out-of-bounds

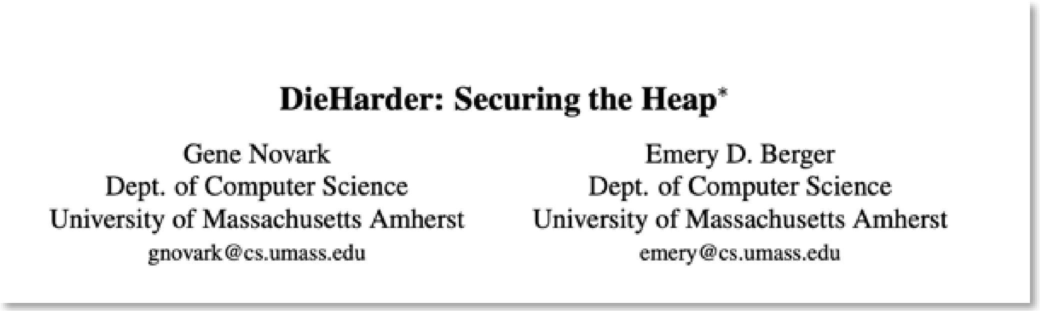
#2 – use after free

#3 – type confusion

#4 – uninitialized use

From "Pursuing Durably Safe Systems Software", Matt Miller, SSTIC 2020

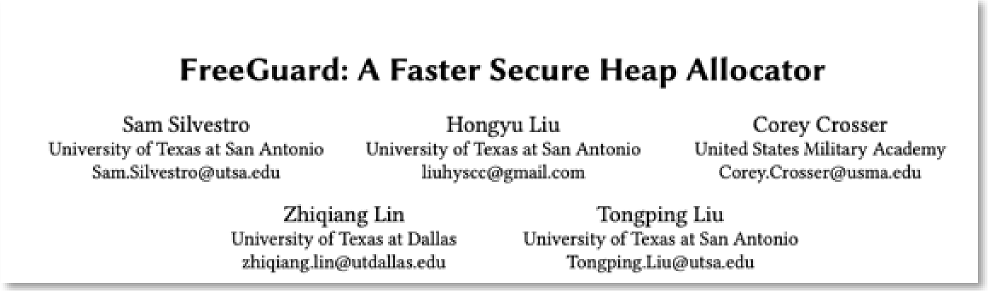
Many secure allocators are proposed



DieHarder: Securing the Heap*

Gene Novark
Dept. of Computer Science
University of Massachusetts Amherst
gnovark@cs.umass.edu

Emery D. Berger
Dept. of Computer Science
University of Massachusetts Amherst
emery@cs.umass.edu



FreeGuard: A Faster Secure Heap Allocator

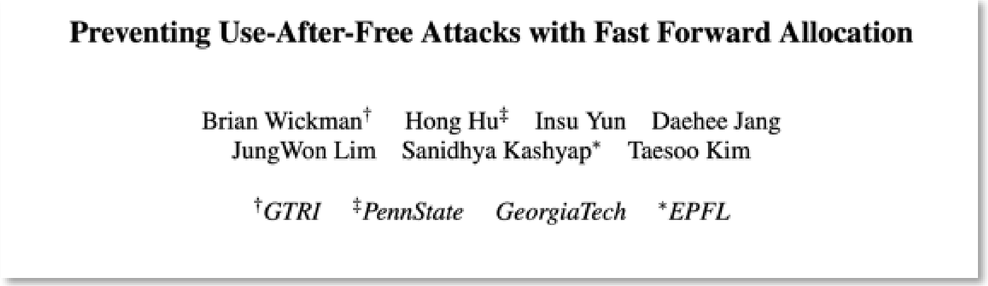
Sam Silvestro
University of Texas at San Antonio
Sam.Silvestro@utsa.edu

Hongyu Liu
University of Texas at San Antonio
liuhyscc@gmail.com

Corey Crosser
United States Military Academy
Corey.Crosser@usma.edu

Zhiqiang Lin
University of Texas at Dallas
zhiqiang.lin@utdallas.edu

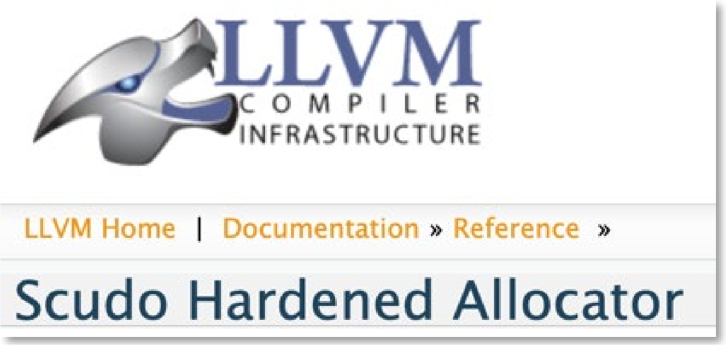
Tongping Liu
University of Texas at San Antonio
Tongping.Liu@utsa.edu



Preventing Use-After-Free Attacks with Fast Forward Allocation

Brian Wickman[†] Hong Hu[‡] Insu Yun Daehee Jang
JungWon Lim Sanidhya Kashyap[*] Taesoo Kim

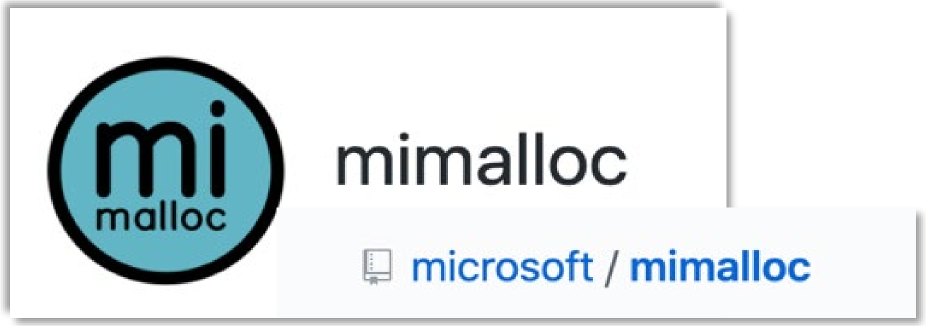
[†]GTRI [‡]PennState GeorgiaTech [*]EPFL



LLVM
COMPILER
INFRASTRUCTURE

LLVM Home | Documentation » Reference »

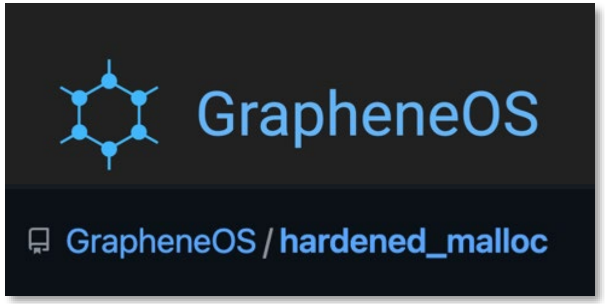
Scudo Hardened Allocator



mi
malloc

mimalloc

microsoft / **mimalloc**



GrapheneOS

GrapheneOS / **hardened_malloc**

Secure allocators support many security properties

- Prevent adjacent chunks
  - e.g., randomization
- Detect buffer overflow
  - e.g., heap canary
- Prohibit reusing memory
  - e.g., randomization
- Stop heap spray
  - e.g., randomization
- Prevent information leakage
  - e.g., separated heap metadata

The security properties are *claimed* individually but attested with *limited* test cases

Problem 1: Hard to compare them with each other

DieHarder: Securing the Heap*

Gene Novark
Dept. of Computer Science
University of Massachusetts Amherst
gnovark@cs.umass.edu

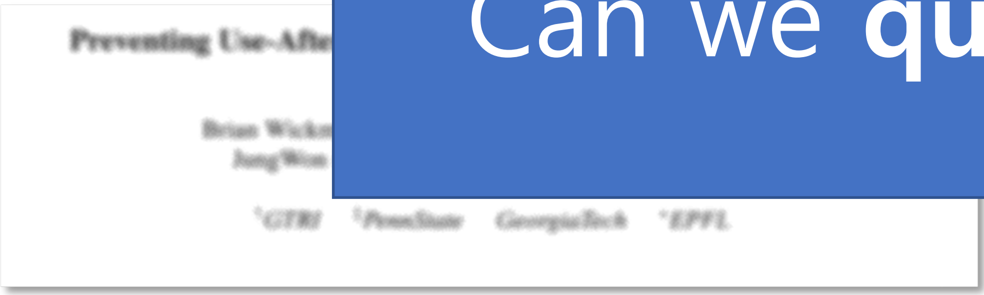
Emery D. Be
Dept. of Compute
University of Massachu
emery@cs.umass

FreeGuard: A Faster S Heap Allocator

Sam Silvestro
University of Texas at San Antonio
Sam.Silvestro@utsa.edu

Hongyu L
University of Texas at

Corey Crosser
Academy

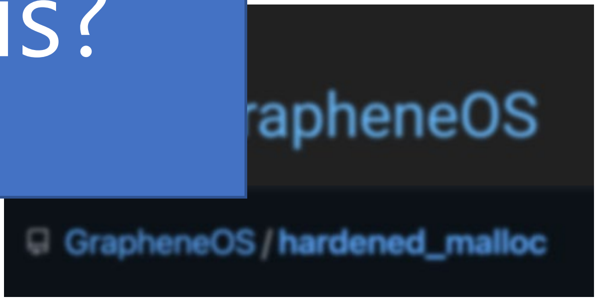


Preventing Use-Afte

Brian Wick
Jang Won

*GTRI †PennState GeorgiaTech *EPFL

microsoft / mimalloc



GrapheneOS / hardened_malloc

Does it support **all** security properties?

Can we **quantify** this?

Problem 1: Hard to compare them with each other

Does it secure in **every** case?

large allocation,
negative allocation,
even more …

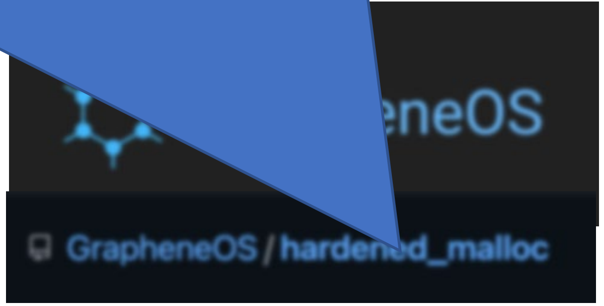
Dep
Universit

Universit

Preventing Use-After-Free Attacks with Fast Forward Allocation

Brian Wickman[1] Hong Hu[2] Insu Yun Daehee Jang
JungWon Lim Sanidhya Kashyap[3] Taesoo Kim

[1]GTRI [2]PennState GeorgiaTech [3]EPFL



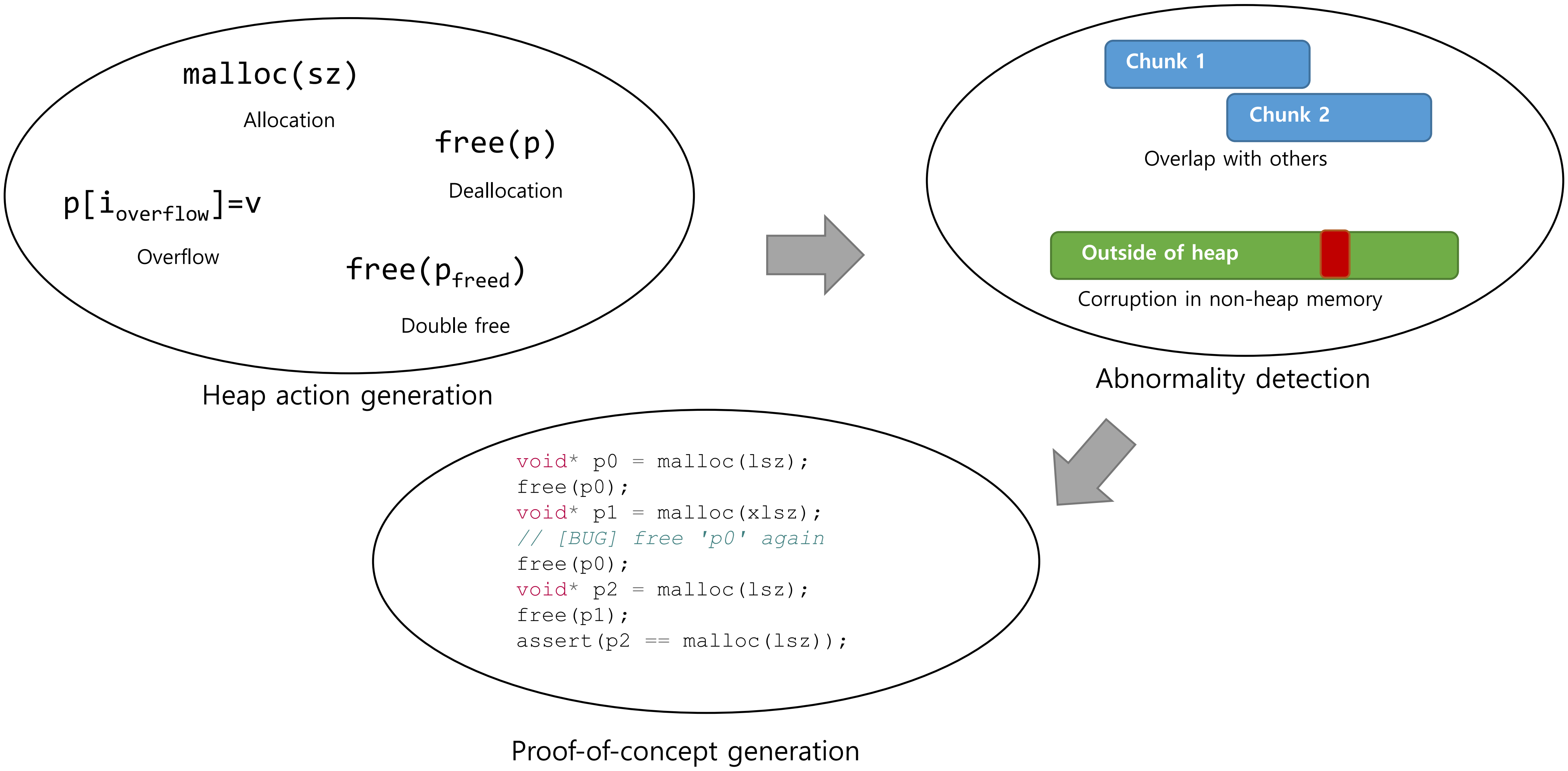
GrapheneOS / harden d_malloc

Example: Double free in DieHarder

```
void* p0 = malloc(80KB);
free(p0);

void* tmp = malloc(100KB);

free(p0); // free 'p0' again

void* p2 = malloc(80KB);

free(tmp);

void* p3 = malloc(80KB);

assert(p2 == p3);
```

Double free a large chunk
→
Overlapping chunks
(Because DieHarder has
no protection on large chunks)

Recall: ArcHeap (Usenix Security '20)



malloc(sz)

Allocation

free(p)

Deallocation

$p[i_{overflow}]=v$

Overflow

free(p_{freed})

Double free

Heap action generation

Chunk 1

Chunk 2

Overlap with others

Outside of heap

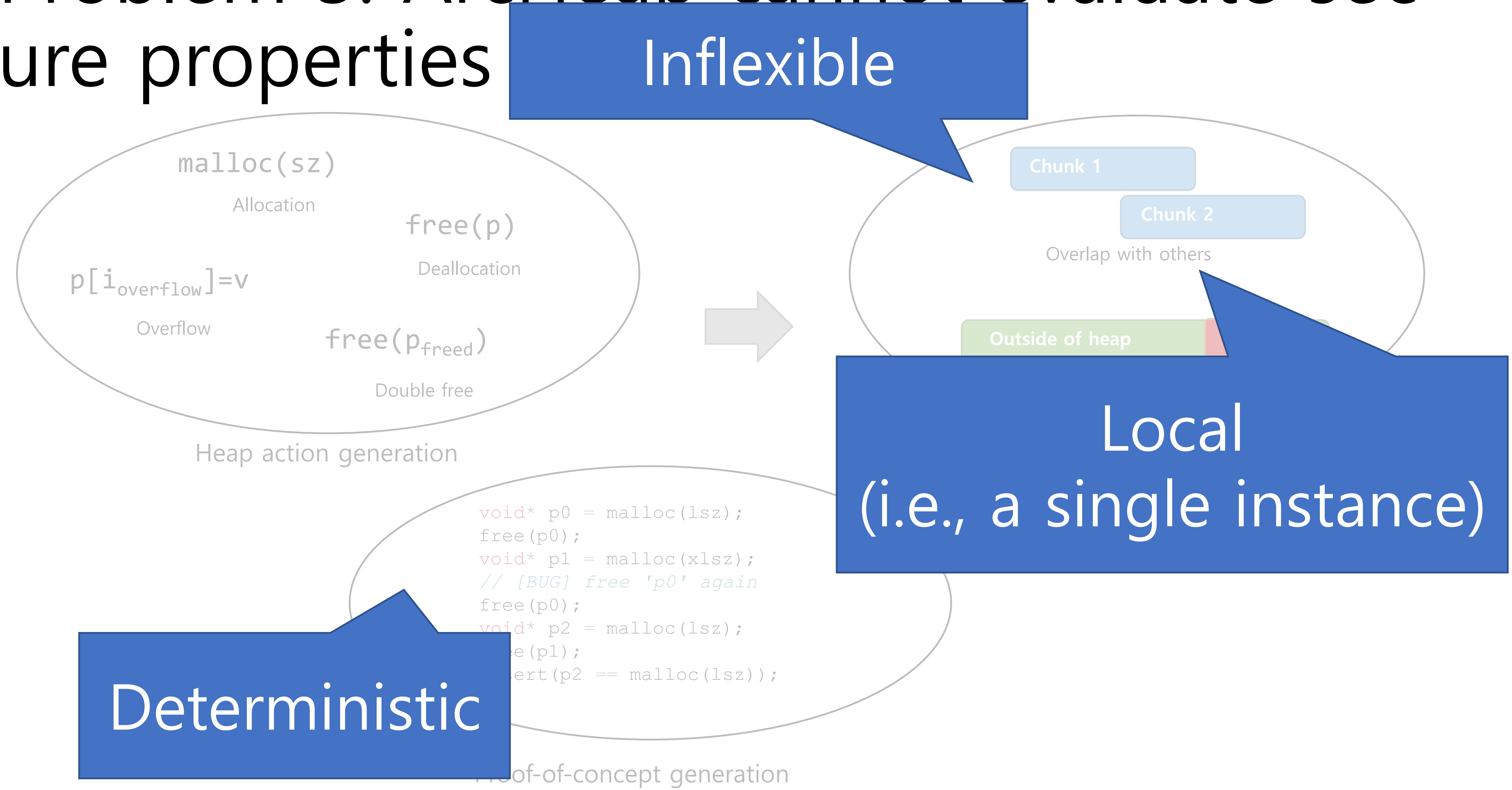
Corruption in non-heap memory

Abnormality detection

```
void* p0 = malloc(lsz);
free(p0);
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```

Proof-of-concept generation

Problem 3: ArcHeap cannot evaluate secure properties



```
void* p0 = malloc(lsz);
free(p0);
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
 e(p1);
 ert(p2 == malloc(lsz));
```

Inflexible

Local
(i.e., a single instance)

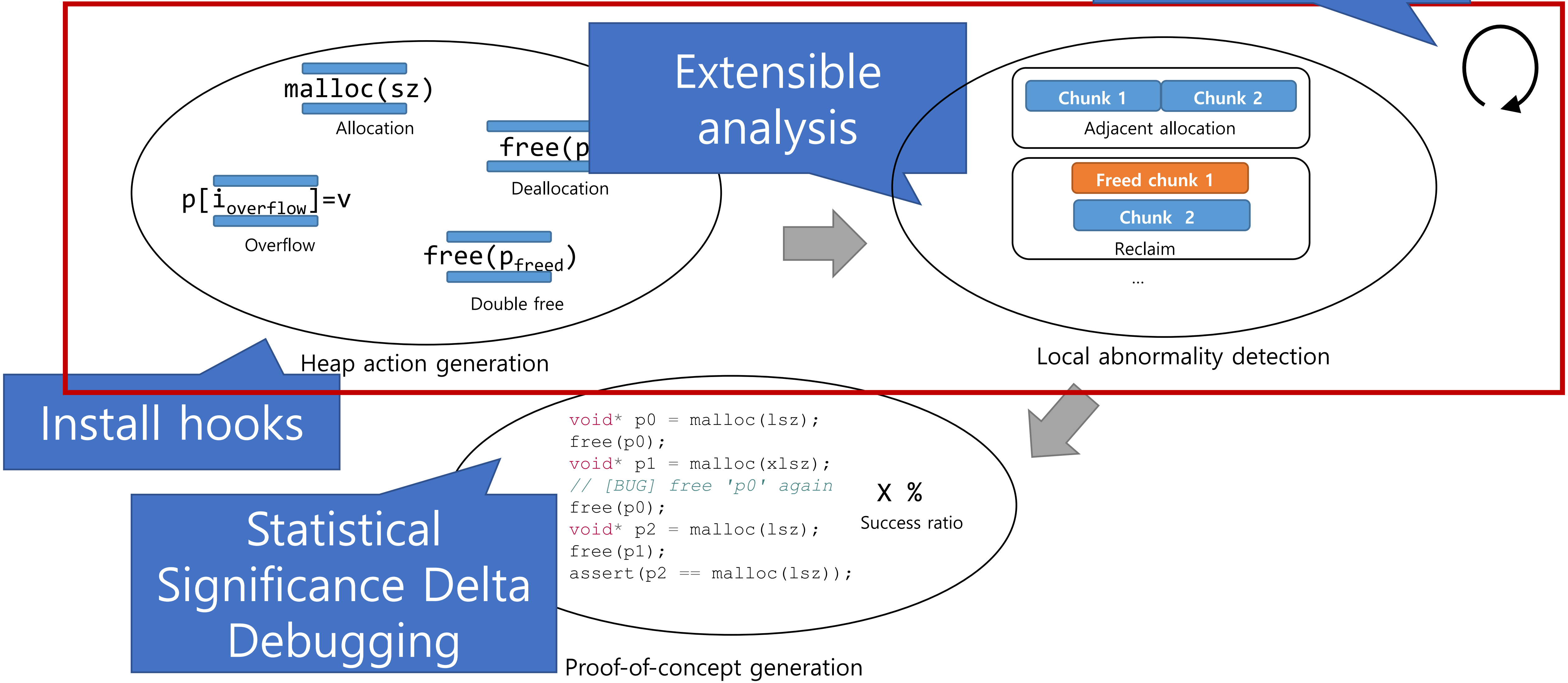
Deterministic

Recall: secure allocators support many security properties

- Prevent adjacent chunks
  - e.g., randomization
- Detect buffer overflow
  - e.g., heap canary
- Prohibit reusing memory
  - e.g., randomization
- Stop heap spray
  - e.g., randomization
- Prevent information leakage
  - e.g., separated heap metadata

HardsHeap: A Universal and Exte
Framework for Evaluating Secure

Sampling-based Testing



```
void* p0 = malloc(lsz);
free(p0);
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```

Extensible analysis

Install hooks

Statistical Significance Delta Debugging

Examples: adjacent chunks

- **Goal:** Check whether the secure allocator can avoid adjacent chunks

- **Analysis:**
    - Local: Check whether adjacent chunks happen by hooking allocations
    - Global: Calculate the probability of adjacent chunks

- **PoC:** Programs with a high chance to get adjacent chunks (e.g., > 25%)

Examples: heap spray

- **Goal:** Check whether the allocator is resilient from heap spray attacks

- **Analysis:**
  - Local: Record chunks' start and size by hooking allocations
  - Global: Calculate the highest probability of the common address among multiple executions

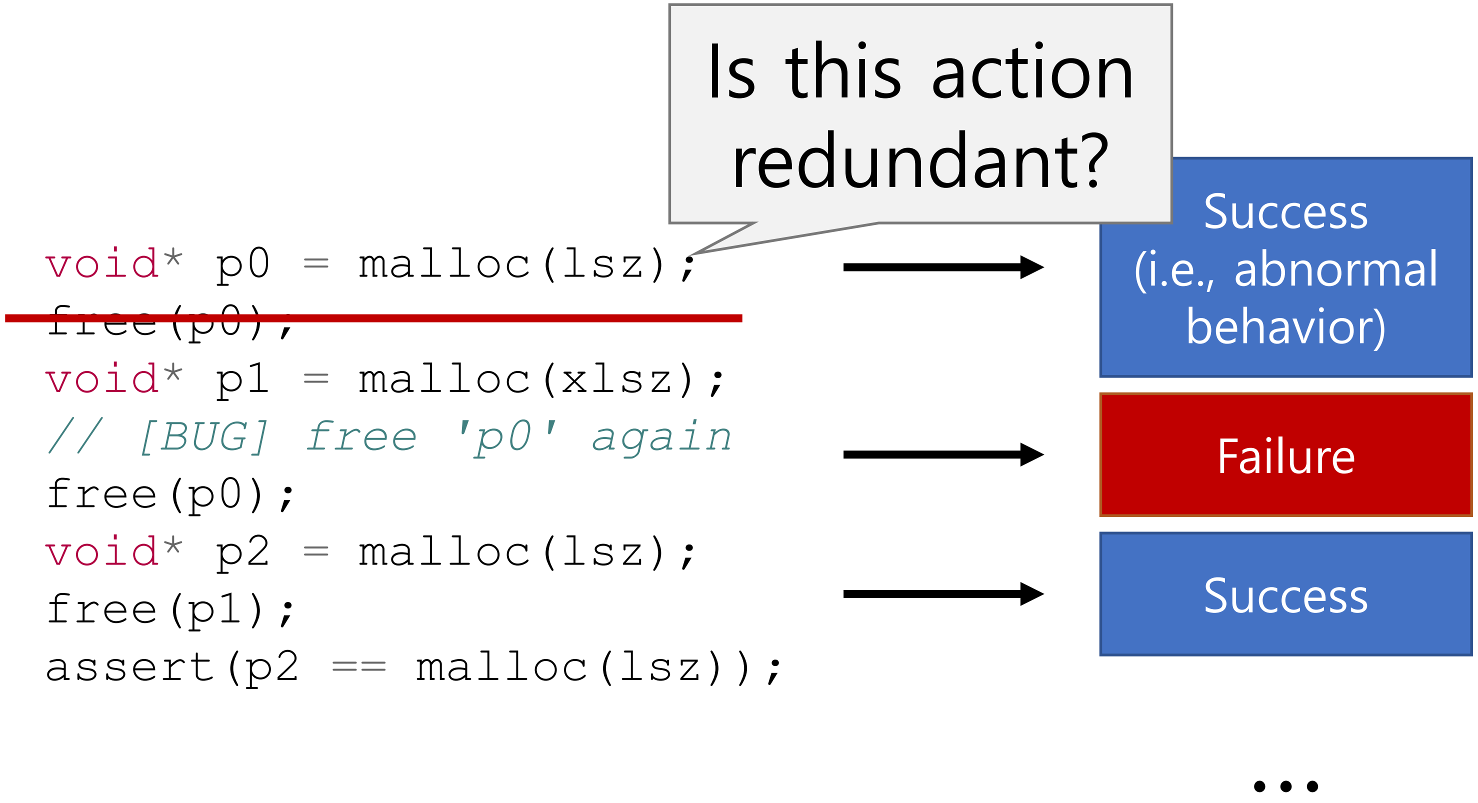
- **PoC:** Programs with a high chance to get the common address

HardsHeap is extensible to cover various security properties

Modules	LoC	Description
Adjacent	135	Check if chunks can be adjacent
Reclaim	119	Check if a dangling chunk is reclaimable
CheckOnFree	89	Check if an allocator can detect a corrupted chunk at free
Uninitialized	78	Check if we get metadata of allocators
Heap spray	64	Check if we can guess a fixed address for every execution
SizeCheck	61	Check if a chunk can be smaller than its request
ArcHeap	574	Other heap vulnerabilities

- **Usable**: ~100 lines of code
- **Extensible**: Various security properties

Due to randomized mechanisms, some test cases are non-deterministic

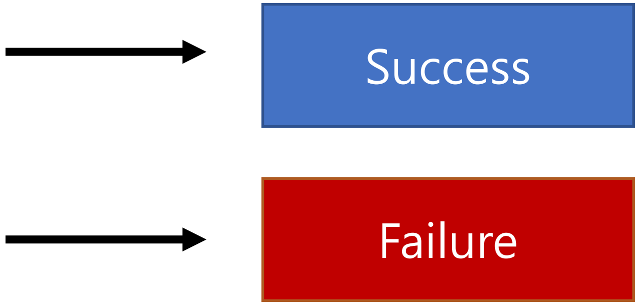


Is this action redundant?

```
void* p0 = malloc(lsz);
~~free(p0);~~
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```

Recall: Delta Debugging

```
void* p0 = malloc(lsz);
~~free(p0);~~
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```



Success

Failure

This action is redundant!

No, this action is not redundant!

HardsHeap addresses this issue by using Statistical Significance Delta Debugging (SSDD)

```
void* p0 = malloc(lsz);
free(p0);
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```

X_1 %
Success ratio

X_2 %
Success ratio

X_3 %
Success ratio

This action is redundant if 1) **Y** is not significantly worse or 2) **Y** is significantly better than **X**

```
void* p0 = malloc(lsz);
~~free(p0);~~
void* p1 = malloc(xlsz);
// [BUG] free 'p0' again
free(p0);
void* p2 = malloc(lsz);
free(p1);
assert(p2 == malloc(lsz));
```

Y_1 %
Success ratio

Y_2 %
Success ratio

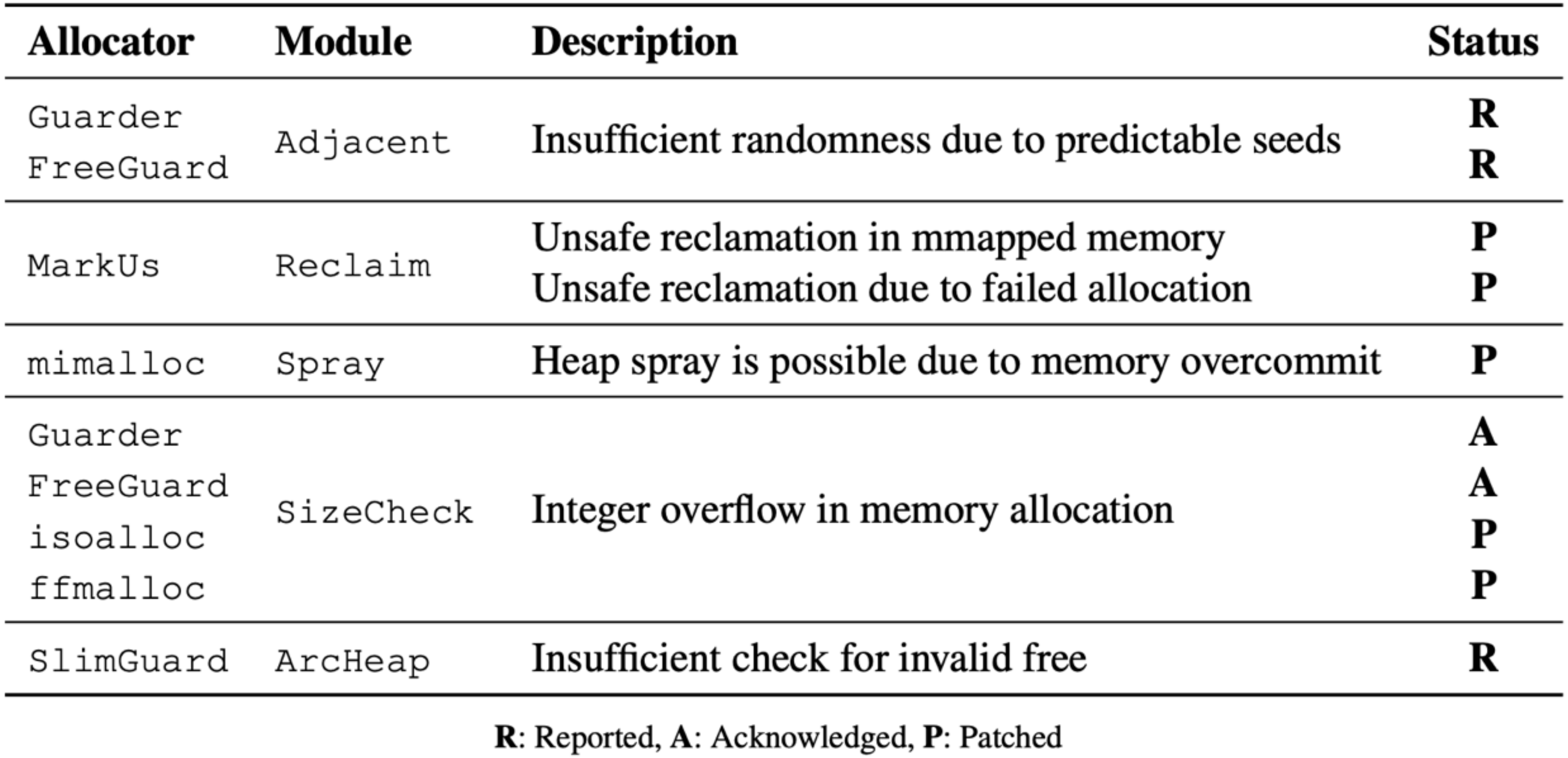
Y_3 %
Success ratio

Evaluation on real-world secure allocators

- Apply to **10** open-source *secure* allocators
  - 6 from academic works
    - DieHarder (CCS '10), FreeGuard (CCS '17),
    - Guarder (Security '18), SlimGuard (Middleware '19),
    - MarkUS (Oakland '20), ffmalloc (Security '21)
  - 4 from non-academic works
    - scudo (Android)
    - mimalloc (Microsoft)
    - hardened_malloc (GrapheneOS)
    - isoalloc (partially inspired by Chrome's PartitionAlloc)

Bugs found by HardsHeap

- **10 bugs** are discovered, **5** are fixed



Allocator	Module	Description	Status
Guarder	Adjacent	Insufficient randomness due to predictable seeds	**R**
FreeGuard			**R**
MarkUs	Reclaim	Unsafe reclamation in mmapped memory	**P**
		Unsafe reclamation due to failed allocation	**P**
mimalloc	Spray	Heap spray is possible due to memory overcommit	**P**
Guarder	SizeCheck	Integer overflow in memory allocation	**A**
FreeGuard			**A**
isoalloc			**P**
ffmalloc			**P**
SlimGuard	ArcHeap	Insufficient check for invalid free	**R**

R: Reported, **A**: Acknowledged, **P**: Patched

Example: adjacent objects in Guarder/FreeGuard

- **Claim:** malloc() return random chunks

```
void* p0 = malloc(…);
void* p1 = malloc(…);
void* p2 = malloc(…);
void* p3 = malloc(…);
…
```

Two malloc **100%** return adjacent objects in a short time period

use `time()` as random source:

- seconds since 1/1/1970
- the same within 1 second

Example: reclaim objects in MarkUs (1/2, Fixed)

- **Claim:** Do not reallocate an object if any reference exists

```
void* p0 = malloc(-1);
void* p1 = malloc(0x80000);
free(p1);
void* p2 = malloc(0x40000);
assert(p1 <= p2 && p2 < p1 + 0x80000);
```

Reallocate the object even if
p2 points to it

After the very large malloc fails (e.g., -1), MarkUs switches to unsafe reallocation

Example: heap spray in mimalloc (Fixed)

- **Claim:** heap address is randomized within 64-bit address space

```
void* p0 = malloc(4TB);
// p0 is always like 0x7FFFFFFFxxx for any runs
```

Low entropy

mimalloc uses MAP_NORESERVE to overcommit memory, which is harmful for randomization

[Fix: return NULL for large allocation > 1GB]

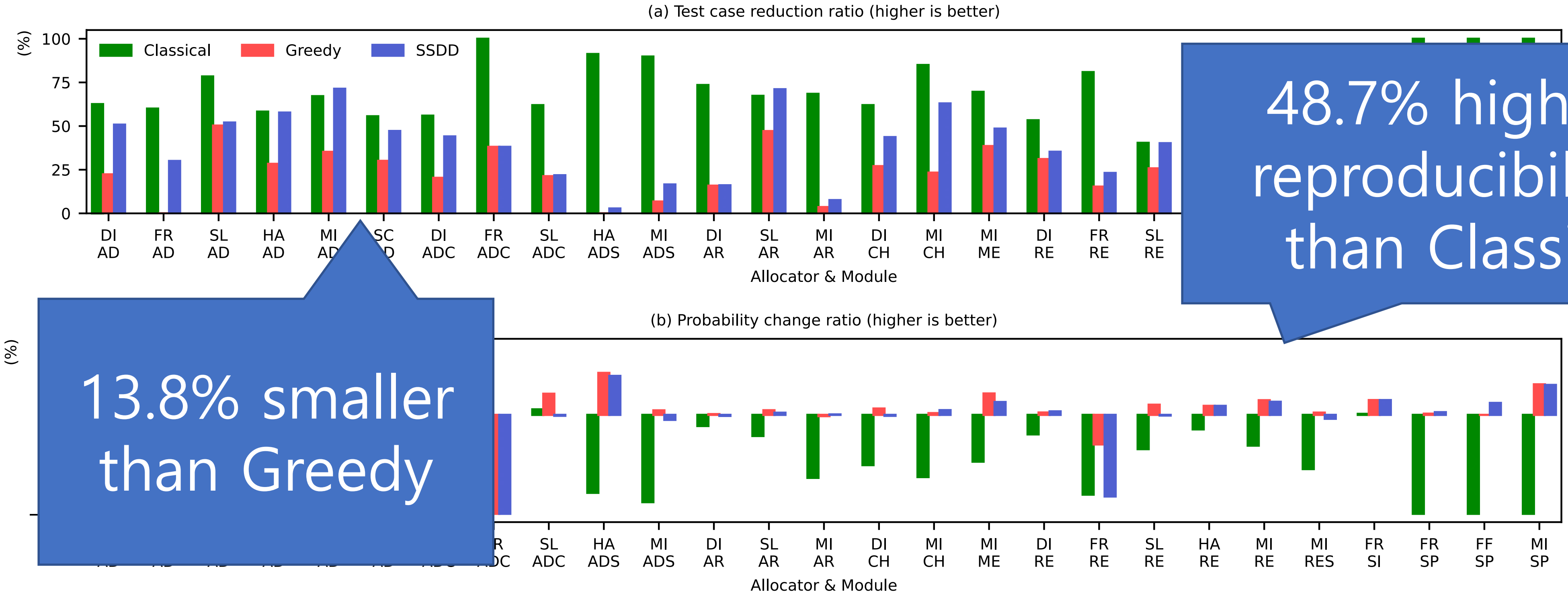
HardsHeap also shows limitations of secure allocators (e.g., Large allocation)

- Known: DieHarder's entropy is inversely proportional to size
  - HardsHeap found reliable adjacent chunks on very large allocation
- **Unknown**: Scudo's entropy is similar to DieHarder's
- **Unknown:** Guarder's entropy becomes zero if we allocate very large chunks (> 512KB)

HardsHeap can discover these behaviors **automatically**!

SSDD is better than other minimization mechanisms

- Classic: Classical Delta Debugging
- Greedy: Only consider average probability without statistical significance



(a) Test case reduction ratio (higher is better)

(b) Probability change ratio (higher is better)

48.7% higher reproducibility than Classic

13.8% smaller than Greedy

Limitations & Discussion

- Limitations
  - Incompleteness
  - Lack of reasoning
  - Only Linux support

Q: HardsHeap results imply that secure allocators are **useless**?

A: **No!** They are not silver bullet but are very useful (See our paper). **Please use them!**

Conclusion

- HardsHeap: Automatic ways to evaluate secure allocators
  - Extensible framework
  - Sampling-based testing
  - Statistical Significance Delta Debugging (Please see our paper)
- 10 implementation bugs and many limitations of various secure allocators
- Open source: https://github.com/kaist-hacking/HardsHeap

Thank you