CAMP: **C**ompiler and **A**llocator-based heap **M**emory **P**rotection

*Zhenpeng Lin, **Zheng Yu**, Ziyi Guo, Simone Campanoni, Peter Dinda, and Xinyu Xing*

Memory Corruption Errors

- Google Project Zero discovered that heap errors accounted for **69% of zero-day vulnerabilities** observed in the wild.
- **65% vulnerabilities** are confirmed as heap-based zero-day in Linux.

CAMP

- ✅ A defense mechanism designed to render OOB and UAF vulnerabilities unexploitable.
- ✅ Low performance overhead.

Temporal Security

- Constructs the point-to relation by instrumenting the program.

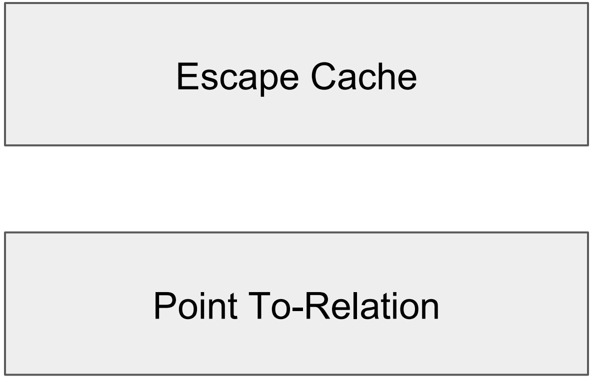
```
char *buf = malloc(16);
__escape(&buf, buf);
```

- When a memory object is freed, CAMP could identify the dangling pointers and nulitify them.

```
free(buf);
```

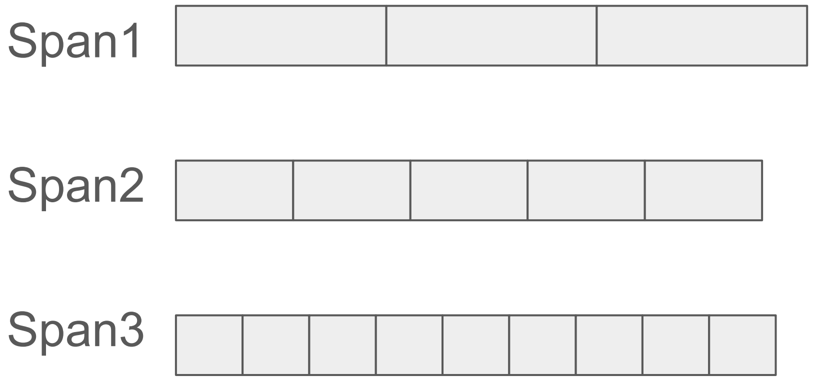
Temporal Security Escape Cache

- New point-to relations are temporarily stored in the cache until it becomes full, at which point the records are transferred to the allocator in a batch, while skipping any duplicates. This cache design boosts runtime speed and reduces memory overhead, particularly in scenarios where the program operates repeatedly in the same block and creates similar point-to relations.



Spatial Security SegList Allocator

- Uses span as unit. Each span manages a size class of memory objects on several contiguous memory pages.
- Leverage the base address and size of span, we can calculate the boundary of each pointer.



Spatial Security

Insert checking at pointer arithmetic site

```
struct obj *o = malloc(sizeof(struct obj));
__check_range(o, o, sizeof(struct obj));

__check_range(ptr, &ptr->a, sizeof(ptr->a));
ptr->a = 1;
__check_range(ptr, &ptr->b, sizeof(ptr->b));
ptr->b = 2;
```

Optimization Type

```
struct obj {
    int a;
    int b;
};
struct obj* bar() {
    // type-casting from void* to obj*
    struct obj *o = malloc(sizeof(struct obj));
    __check_range(o, o, sizeof(struct obj));
    ...
}
int foo(struct obj *ptr) {
    __check_range(ptr, &ptr->a, sizeof(ptr->a));
    ptr->a = 1;
    __check_range(ptr, &ptr->b, sizeof(ptr->b));
    ptr->b = 2;
}
```

Ensure the memory space referenced by a typed pointer is adequate for its corresponding type

Northwestern University

Optimization Type

```c
struct obj {
    int a;
    int b;
};
struct obj* bar() {
    // type-casting from void* to obj*
    struct obj *o = malloc(sizeof(struct obj));
    __check_range(o, o, sizeof(struct obj));
    ...
}
int foo(struct obj *ptr) {
    __check_range(ptr, &ptr->a, sizeof(ptr->a));
    ptr->a = 1;
    __check_range(ptr, &ptr->b, sizeof(ptr->b));
    ptr->b = 2;
}
```

Ensure the memory space referenced by a typed pointer is adequate for its corresponding type

Optimization Merge

```
void foo(char *ptr, int i, int j) {
    unsigned int start, end;
    __get_range(ptr, &start, &end);
    assert(&ptr[i]>=start && &ptr[i]+1<end);
    ptr[i] = 'x';
    assert(&ptr[j]>=start && &ptr[j]+1<end);
    ptr[j] = 'y';
}
```

Fetch boundary at first

Optimization Merge

```
void foo(char *ptr, int i, int j) {
    unsigned int start, end;
    __get_range(ptr, &start, &end);
    assert(&ptr[i]>=start && &ptr[i]+1<end);
    ptr[i] = 'x';
    assert(&ptr[j]>=start && &ptr[j]+1<end);
    ptr[j] = 'y';
}
```

Fetch boundary at first

Simplified Checking

Security Evaluation Real World Vulnerabilities

- The evaluation compares CAMP' performance with other defense solutions offering similar heap protection levels.

CVE/Issue ID	Application	Bug Type	CAMP	ASAN--	Memcheck	DangNull	MarkUs	Delta pointer
CVE-2015-3205	libmimedir	Use-After-Free	✔	✔	✔	✔	✔	/
CVE-2015-2787	PHP 5.6.5	Use-After-Free	✔	✔	✔	✖	✖	/
CVE-2015-6835	PHP 5.4.44	Use-After-Free	✔	✔	✔	✔	✖	/
CVE-2016-5773	PHP 7.0.7	Use-After-Free	✔	✔	✔	✔	✖	/
Issue-3515 [50]	mruby	Use-After-Free	✔	✔	✔	Build Fail	✖	/
CVE-2020-6838	mruby	Use-After-Free	✔	✔	✔	Build Fail	✖	/
CVE-2021-44964	Lua	Use-After-Free	✔	✔	✔	Build Fail	✔	/
CVE-2020-21688	FFmpeg	Use-After-Free	✔	✔	✔	✖	✔	/
CVE-2021-33468	yasm	Use-After-Free	✔	✔	✔	✔	✔	/
CVE-2020-24978	nasm	Use-After-Free	✔	✔	✔	✖	✔	/
Issue-1325664 [6]	Chrome	Use-After-Free	✔	✔	✔	Build Fail	✖	/
CVE-2022-43286	Nginx	Use-After-Free	✔	✔	✔	✖	✔	/
CVE-2019-16165	cflow	Use-After-Free	✔	✔	✔	✖	✔	/
CVE-2021-4187	vim	Use-After-Free	✔	✔	✔	✖	✔	/
CVE-2022-0891	libtiff	Heap Overflow	✔	✔	✔	/	/	✔
CVE-2022-0924	libtiff	Heap Overflow	✔	✔	✔	/	/	✔
CVE-2020-19131	libtiff	Heap Overflow	✔	✔	✔	/	/	✔
CVE-2020-19144	libtiff	Heap Overflow	✔	✔	✔	/	/	✔
CVE-2021-4214	libpng	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2021-3156	sudo	Heap Overflow	Run Well	✔	✔	/	/	Build Fail
CVE-2018-20330	libjpeg-turbo	Heap Overflow	✔	✔	✔	/	/	✔
CVE-2020-21595	libde265	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2020-21598	libde265	Heap Overflow	✔	✔	✔	/	/	Build Fail
Issue-5551 [4]	mruby	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2022-0080	mruby	Heap Overflow	Run Well	✔	✔	/	/	Build Fail
CVE-2019-9021	PHP	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2022-31627	PHP	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2021-32281	gravity	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2020-15888	Lua	Heap Overflow	✔	✔	✔	/	/	Build Fail
CVE-2021-26259	htmldoc	Heap Overflow	✔	✖	✔	/	/	Build Fail
CVE-2022-28966	Wasm3	Heap Overflow	✔	✔	✔	/	/	Build Fail

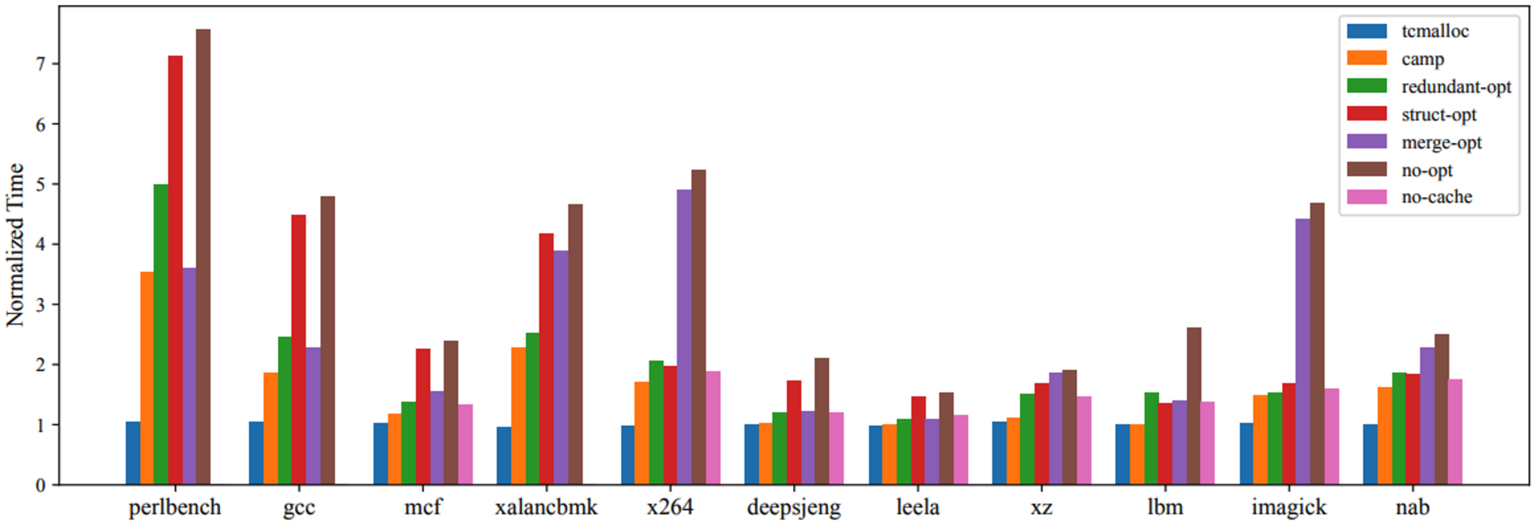
Performance Evaluation SPEC CPU 2017

- CAMP provide better performance on SPEC CPU 2017.

Benchmark	Time and Memory Overhead				
	CAMP	ASAN−−	ASAN	ESAN	Memcheck
600.perlbench_s	237.95% / 2241.12%	76.95% / 366.92%	143.59% / 358.20%	644.00% / 4.15%	3496.46% / 138.97%
602.gcc_s	78.56% / 135.52%	83.61% / 63.42%	99.47% / 62.77%	-	2888.13% / 30.42%
605.mcf_s	14.62% / 31.55%	24.45% / 3.61%	27.88% / 3.61%	109.33% / -4.24%	601.05% / 22.68%
623.xalancbmk_s	138.94% / 1220.66%	107.86% / 428.07%	109.41% / 433.51%	81.67% / 8.60%	4962.60% / 98.81%
625.x264_s	75.07% / 12.68%	62.26% / 13.52%	75.92% / 13.26%	90.94% / -3.55%	2070.57% / 56.96%
631.deepsjeng_s	1.58% / 0.00%	44.23% / -0.23%	64.08% / -0.23%	18.85% / -0.25%	3251.34% / 25.34%
641.leela_s	3.02% / 514.19%	13.97% / 2832.83%	17.33% / 2833.72%	6.65% / -17.52%	4163.69% / 262.82%
657.xz_s	7.79% / 0.00%	17.45% / 2.98%	13.40% / 2.98%	14.61% / -0.70%	718.87% / 24.45%
619.lbm_s	1.34% / 0.01%	37.32% / 5.94%	29.38% / 5.94%	34.14% / -0.36%	2907.53% / 25.98%
638.imagick_s	45.47% / 0.07%	17.23% / 4.46%	28.56% / 4.47%	21.70% / -2.00%	4452.66% / 22.93%
644.nab_s	62.55% / 26.13%	35.18% / 67.52%	35.14% / 66.63%	1988.66% / -1.34%	3722.35% / 31.80%
Geomean	21.27% / 127.47%	38.27% / 104.72%	44.78% / 104.35%	65.31% / -1.94%	2546.88% / 56.49%

Ablation Study

- Evaluation result of CAMP breakdown. The bars show the normalized time of tcmalloc replacement, CAMP, CAMP with each optimization disabled, and CAMP without optimization.



Conclusion

- **Introduction of CAMP**: Employs a customized allocator and a compiler to safeguard against heap memory corruption.

- **Implementation**: Customizes tcmalloc and builds on LLVM 12.0 compiler framework, with the prototype open-sourced.

- **Evaluation**: Tested on Nginx and SPEC CPU benchmarks, assessed for security and runtime overhead, and compared with other defense solutions.

Thank You

HomePage

- Zheng Yu
- zheng.yu@northwestern.edu