Summary & Conclusion

- Memory addresses can impact program execution
Memory addresses can impact program execution

**KDAALLOC**: Cross-run and cross-path deterministic allocation

Visit me tomorrow for more examples and everything not covered today!

Try it yourself!
Memory addresses can impact program execution

**KDAlloc**: Cross-run and cross-path deterministic allocation

Not presented *today*: Spatial and temporal distancing, memory and time efficiency, asan integration demo, …
Memory addresses can impact program execution

**KDAlloc**: Cross-run and cross-path deterministic allocation

Not presented *today*: Spatial and temporal distancing, memory and time efficiency, asan integration demo, ...

Visit me tomorrow for more examples and everything not covered today!
A Treap with Concrete Inputs

```
#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

struct node {
    struct node *lhs, *rhs;
    char const *key;
} *root = NULL;

static uint64_t priority(void *p) {
    uint64_t h = 0xcbf29ce484222325;
    for (size_t i = 0; i < sizeof(p); ++i)
        h = (h ^ (*((unsigned char *)p)[i])) * 0x100000001b3;
    return h;
}

void insert(struct node **n, char const *str) {
    if (!*n) {
        *n = calloc(1, sizeof(struct node));
        (*n)->key = str;
    } else {
        int cmp = strcmp(str, (*n)->key);
        if (cmp < 0) {
            insert(&(*n)->lhs, str);
            if (priority((*n)->lhs) < priority(*n)) {
                struct node *lhs = (*n)->lhs;
                (*n)->lhs = lhs->rhs;
                lhs->rhs = (*n);
                *n = lhs;
            } else if (cmp > 0) {
                insert(&(*n)->rhs, str);
                if (priority((*n)->rhs) < priority(*n)) {
                    struct node *rhs = (*n)->rhs;
                    (*n)->rhs = rhs->lhs;
                    rhs->lhs = (*n);
                    *n = rhs;
                }
        } else if (cmp == 0) {
            assert(strcmp(root->key, "1") == 0);
        }
    }
}

int main() {
    insert(&root, strdup("1"));
    insert(&root, strdup("2"));
    insert(&root, strdup("3"));
    assert(strcmp(root->key, "1") == 0);
    return 0;
}
```
Running Concretely

```
$ clang treap-conc.c -g3 -o treap-conc.exe
$ ./treap-conc.exe
$ echo $? 
0
```
Running Concretely

$ clang treap-conc.c -g3 -o treap-conc.exe
$ ./treap-conc.exe
$ echo $? 
0  
$ ./treap-conc.exe
treap-conc.exe: treap-conc.c:49: int main(): Assertion `strcmp(root->key, "1") == 0' failed. (core dumped) ./treap.exe
$ echo $? 
134
Running in KLEE

$ clang treap-conc.c -g3 -c -emit-llvm
$ klee --posix-runtime --libc=uclibc treap-conc.bc

[...]
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled.
Using alignment of 8.

KLEE: done: total instructions = 14801
KLEE: done: completed paths = 1
KLEE: done: partially completed paths = 0
KLEE: done: generated tests = 1
Running in KLEE

$ clang treap-conc.c -g3 -c -emit-llvm
$ klee --posix-runtime --libc=uclibc treap-conc.bc

[...]
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled. Using alignment of 8.

KLEE: done: total instructions = 14801
KLEE: done: completed paths = 1
KLEE: done: partially completed paths = 0
KLEE: done: generated tests = 1

$ klee --posix-runtime --libc=uclibc treap-conc.bc

[...]
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled. Using alignment of 8.
KLEE: ERROR: treap-conc.c:49: ASSERTION FAIL: strcmp(root->key, "1") == 0
KLEE: NOTE: now ignoring this error at this location

KLEE: done: total instructions = 14340
KLEE: done: completed paths = 0
KLEE: done: partially completed paths = 1
KLEE: done: generated tests = 1
Running in KLEE with KDAlloc 1/2

$ klee --posix-runtime --libc=uclibc --kdalloc treap-conc.bc
[...]
KLEE: Deterministic allocator: Using quarantine queue size 8
KLEE: Deterministic allocator: globals (start-address=0x7f992f600000 size=10 GiB)
KLEE: Deterministic allocator: constants (start-address=0x7f96af600000 size=10 GiB)
KLEE: Deterministic allocator: heap (start-address=0x7e96af600000 size=1024 GiB)
KLEE: Deterministic allocator: stack (start-address=0x7e76af600000 size=128 GiB)
KLEE: WARNING ONCE: calling external: syscall(16, 0, 21505, 139093310701568) at /klee-src/runtime/POSIX/fd.c:997 10
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignment of 8.
KLEE: WARNING ONCE: calling __klee_posix_wrapped_main with extra arguments.
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled. Using alignment of 8.
KLEE: ERROR: treap-conc.c:49: ASSERTION FAIL: strcmp(root->key, "1") == 0
KLEE: NOTE: now ignoring this error at this location

KLEE: done: total instructions = 14393
KLEE: done: completed paths = 0
KLEE: done: partially completed paths = 1
KLEE: done: generated tests = 1
Running in KLEE with KDAlloc 2/2

$ klee --posix-runtime --libc=uclibc --kdalloc treap-conc.bc
[...]
KLEE: Deterministic allocator: Using quarantine queue size 8
KLEE: Deterministic allocator: globals (start-address=0x7fccc56e00000 size=10 GiB)
KLEE: Deterministic allocator: constants (start-address=0x7fcd96e00000 size=10 GiB)
KLEE: Deterministic allocator: heap (start-address=0x7ece96e00000 size=1024 GiB)
KLEE: Deterministic allocator: stack (start-address=0x7ee96e00000 size=128 GiB)
KLEE: WARNING ONCE: calling external: syscall(16, 0, 21505, 139313016733696) at /klee-src/runtime/POSIX/fd.c:997 10
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignment of 8.
KLEE: WARNING ONCE: calling __klee_posix_wrapped_main with extra arguments.
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled. Using alignment of 8.

KLEE: done: total instructions = 14854
KLEE: done: completed paths = 1
KLEE: done: partially completed paths = 0
KLEE: done: generated tests = 1
Running in KLEE with KDAlloc and Fixed Base-Addresses

$ klee --posix-runtime --libc=uclibc --kdalloc  
   --kdalloc-constants-start-address=0x610000000000  
   --kdalloc-globals-start-address=0x620000000000  
   --kdalloc-heap-start-address=0x640000000000  
   --kdalloc-stack-start-address=0x630000000000  
treap-conc.bc

[...]
KLEE: Deterministic allocator: Using quarantine queue size 8
KLEE: Deterministic allocator: globals (start-address=0x620000000000 size=10 GiB)
KLEE: Deterministic allocator: constants (start-address=0x610000000000 size=10 GiB)
KLEE: Deterministic allocator: heap (start-address=0x640000000000 size=1024 GiB)
KLEE: Deterministic allocator: stack (start-address=0x630000000000 size=128 GiB)
KLEE: WARNING ONCE: calling external: syscall(16, 0, 21505, 108896748306432) at /klee-src/runtime/POSIX/fd.c:997
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignment of 8.
KLEE: WARNING ONCE: calling __klee_posix_wrapped_main with extra arguments.
KLEE: WARNING ONCE: Alignment of memory from call "calloc" is not modelled. Using alignment of 8.
KLEE: ERROR: treap-conc.c:49: ASSERTION FAIL: strcmp(root->key, "1") == 0
KLEE: NOTE: now ignoring this error at this location

KLEE: done: total instructions = 14415
KLEE: done: completed paths = 0
KLEE: done: partially completed paths = 1
KLEE: done: generated tests = 1
A Treap with Symbolic Inputs

```c
#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>

struct node {
    struct node *lhs, *rhs;
    char const *key;
} *root = NULL;

static uint64_t priority(void *p) {
    uint64_t h = 0xcbf29ce484222325;
    for (size_t i = 0; i < sizeof(p); ++i)
        h = (h ^ (*((unsigned char *)p) + i)) * 0x100000001B3;
    return h;
}

void insert(struct node **n, char const *str) {
    if (!*n) {
        *n = calloc(1, sizeof(struct node));
        (*n)->key = str;
    } else {
        int cmp = strcmp(str, (*n)->key);
        if (cmp < 0) {
            insert(&(*n)->lhs, str);
            if (priority((*n)->lhs) < priority(*n)) {
                struct node *lhs = (*n)->lhs;
                (*n)->lhs = lhs->rhs;
                lhs->rhs = (*n);
                *n = lhs;
            }
        } else if (cmp > 0) {
            insert(&(*n)->rhs, str);
            if (priority((*n)->rhs) < priority(*n)) {
                struct node *rhs = (*n)->rhs;
                (*n)->rhs = rhs->lhs;
                rhs->lhs = (*n);
                *n = rhs;
            }
        }
    }
}

int main() {
    char sym2[2], sym3[2];
    sym2[1] = '\0';
    sym3[1] = '\0';
    read(0, sym2, 1);
    read(0, sym3, 1);
    insert(&root, strdup("1"));
    insert(&root, sym2);
    insert(&root, sym3);
    assert(strcmp(root->key, "1") == 0);
}
```
Replaying Symbolic Execution with KDAloc

$ clang treap-sym.c -g3 -c -emit-llvm
$ klee --posix-runtime --libc=uclibc --emit-all-errors \klee-replay klee-last/test000002.ktest ./treap-sym.exe

KLEE-REPLAY: NOTE: EXIT STATUS: NORMAL (0 seconds)
Replaying Symbolic Execution with KDAlloc

```
$ clang treap-sym.c -g3 -c -emit-llvm
$ klee --posix-runtime --libc=uclibc --emit-all-errors \
   treap-sym.bc --sym-stdin 2
[...]
$ clang treap-sym.c -g3 -o treap-sym.exe
$ klee-replay klee-last/test000002.ktest ./treap-sym.exe
[...]
KLEE-REPLAY: NOTE: EXIT STATUS: NORMAL (0 seconds)
[...]
$ klee-replay klee-last/test000002.ktest ./treap-sym.exe
[...]
treap-sym.exe: treap-sym.c:59: int main(): Assertion `strcmp(root->key, "1") == 0' failed.
KLEE-REPLAY: NOTE: EXIT STATUS: CRASHED signal 6 (0 seconds)
[...]```
Replaying Symbolic Execution with KDAlloc

```bash
$ klee --posix-runtime --libc=uclibc --emit-all-errors \
   --kdalloc \
   --kdalloc-constants-start-address=0x610000000000 \
   --kdalloc-globals-start-address=0x620000000000 \
   --kdalloc-heap-start-address=0x640000000000 \
   --kdalloc-stack-start-address=0x630000000000 \
   treap-sym.bc --sym-stdin 2

[...]

$ klee-replay klee-last/test000002.ktest ./treap-sym.exe

[...]
KDAlloc initialized at 0x640000000000 with 1024GiB and quarantine 8
treap-sym.exe: treap-sym.c:59: int main(): Assertion `strcmp(root->key, "1") == 0' failed.
KLEE-REPLAY: NOTE: EXIT STATUS: CRASHED signal 6 (0 seconds)

[...]
```
Summary & Conclusion

- Memory addresses can impact program execution
Summary & Conclusion

- Memory addresses can impact program execution
- **KDAALLOC**: Cross-run and cross-path deterministic allocation

Visit me tomorrow for more examples and everything not covered today!
Summary & Conclusion

- Memory addresses can impact program execution
- **KDAlloc**: Cross-run and cross-path deterministic allocation
- Not presented *today*: Spatial and temporal distancing, memory and time efficiency, asan integration demo, …

Try it yourself!
Memory addresses can impact program execution

**KDAlloc**: Cross-run and cross-path deterministic allocation

Not presented *today*: Spatial and temporal distancing, memory and time efficiency, asan integration demo, ...

Visit me tomorrow for more examples and everything not covered today!