Eliminating Memory Errors in C

Thames Software

Karsten Pedersen

10/06/2020

Existing solutions

- Valgrind
- ElectricFence
- Debug runtimes (I.e MSVC)
- AddressSanitizer (Clang)
  - Older Mudflap (GCC)

Thames
Software

Introducing libstent

- A "weak pointer" for C
  - Only enabled at debug time
- A "templated vector" for C
- Header-only library
  - C89+ compatible

Thames Software

Getting started (current)

```
#include <stdlib.h>

struct Employee
{
  int id;
};

int main()
{
  struct Employee *emp = calloc(1, sizeof(struct Employee));
  free(emp);

  return 0;
}
```

Thames Software

Getting started (libstent)

```
#define STENT_IMPLEMENTATION
#include <stent.h>

struct Employee
{
  int id;
};

int main()
{
  ref(Employee) emp = allocate(Employee);
  release(emp);

  return 0;
}
```

Thames Software

Accessing members

```
ref(Employee) emp = allocate(Employee);

_(emp).id = 2 * 2;

printf("ID: %i\n", _(emp).id);
```

Result:

```
ID: 4
```

Thames Software

Detecting leaks

```
int main()
{
  ref(Employee) emp = allocate(Employee);

  return 0;
}
```

Result:

```
Warning: Allocated memory [main.c:16] persisted after application exit [Employee]
Aborted
```

Thames Software

Use after free (simple)

```
int main()
{
  ref(Employee) emp = allocate(Employee);
  release(emp);

  printf("ID: %i\n", _(emp).id);

  return 0;
}
```

Result:

```
Error: Employee pointer no longer valid in main.c:17
Aborted
```

Thames Software

Use after free (continued)

```
ref(Employee) curr;

int main()
{
  ref(Employee) emp = allocate(Employee);
  curr = emp;
  release(emp);

  printf("ID: %i\n", _(curr).id);

  return 0;
}
```

Result:

```
Error: Employee pointer no longer valid in main.c:19
Aborted
```

Thames Software

Use ref for members

```
struct Employee
{
  int id;
  ref(Department) dept;
  ref(Employee) mgr;
};

ref(Employee) EmployeeCreate(ref(Department) dept, ref(Employee) mgr)
{
  ref(Employee) rtn = allocate(Employee);
  _(rtn).dept = dept;
  _(rtn).mgr = mgr;

  return rtn;
}
```

Thames Software

Assume raw to refer to stack

```
struct Employee
{
  int id;
  ref(Department) dept;
  ref(Employee) mgr;
  struct Work *work; /* Avoid raw pointers in structures */
};

void EmployeeWork(ref(Employee) ctx, struct Work *work)
{
  /* work is stack memory so guaranteed be valid in this function */

  if(work->type == WORK_JUMP) { ... }
}
```

Thames Software

Using vectors (simple)

```
int main()
{
  vector(int) ids = vector_new(int);

  vector_push(ids, 9);
  vector_push(ids, 5);
  vector_push(ids, 3);

  printf("ID: %i\n", vector_at(ids, 1));

  vector_delete(ids);

  return 0;
}
```

Thames Software

Using vectors (continued)

```
int main()
{
  vector(ref(Employee)) emps = vector_new(ref(Employee));

  vector_push(emps, EmployeeCreate(NULL, NULL));
  vector_push(emps, EmployeeCreate(NULL, NULL));
  vector_push(emps, EmployeeCreate(NULL, NULL));

  printf("ID: %i\n", EmployeeId(vector_at(emps, 1)));

  vector_delete(emps);

  return 0;
}
```

Thames Software

We just leaked memory!

Result:

```
Warning: Allocated memory persisted after application exit [Employee]
Warning: Allocated memory persisted after application exit [Employee]
Warning: Allocated memory persisted after application exit [Employee]
Aborted
```

Note: If you forget to call vector_delete, you would also see:

```
Warning: Allocated memory persisted after application exit [vector(ref(Employee))]
```

Thames Software

The quick fix (using foreach)

```
/*
 * Free employees before deleting the vector
 */
foreach(ref(Employee) emp, emps,
  release(emp);
)

vector_delete(emps);
```

Thames
Software

Standard iteration through vectors

```
size_t i = 0;

/*
 * Free employees before deleting the vector
 */
for(; i < vector_size(emps); ++i)
{
  release(vector_at(emps, i));
}

vector_delete(emps);
```

Thames Software

Detecting vector out of bounds

```
vector(int) ids = vector_new(int);
vector_push(ids, 0);
vector_push(ids, 1);
vector_push(ids, 2);

printf("ID: %i\n", vector_at(ids, 3));
```

Result:

```
Error: Index [index=3] out of bounds [size=3]
Aborted
```

Thames Software

Additional vector functionality

- vector_resize
- vector_clear
- vector_insert
- vector_erase

Thames
Software

Additional libstent features

- Thread-safe
- Some additional facilities
  - sstream_new()
  - ifstream_open(...)
  - dir_open(...)

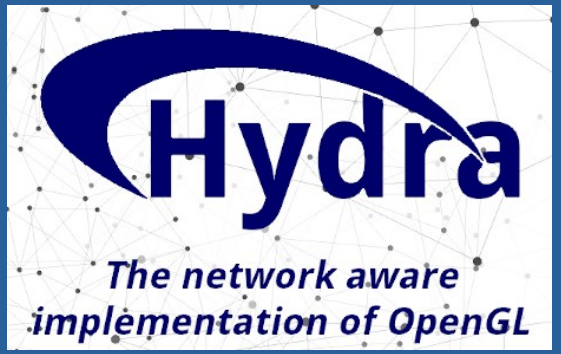
Thames
Software

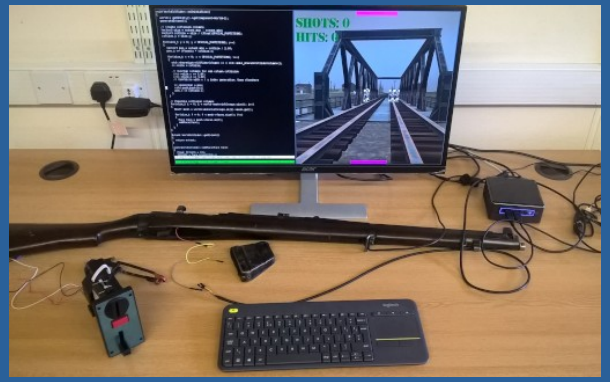
Overhead free release builds

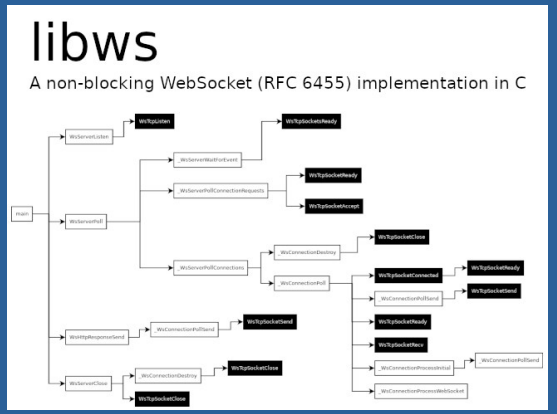
- Simply undefine STENT_ENABLE
- Pointers aren’t checked
  - ref(Employee) --> struct Employee *
- Vector bounds aren’t checked

Thames
Software

Some of our projects using libstent







Thames Software

Questions?

- kpedersen@thamessoftware.co.uk
- https://github.com/osen/stent

Thames
Software