Bug class genocide

Applying science to eliminate 100% of buffer overflows

Hackito Ergo Sum, 2014
Andreas Bogk, @andreasdotorg

The problem

```
void foo (char* arg) {
  char some[16];
  char* p = some;
  while (*p++ = *arg++);
}
```

The problem: spatial memory safety

```
void foo (char* arg) {
  char some[16];
  char* p = some;
  while (*p++ = *arg++);
}
```

The other problem

```
void bar (char* arg) {
  char* p = malloc(16);
  free(p);
  while (*p++ = *arg++);
  free(p);
}
```

The other problem: temporal memory safety

```
void bar (char* arg) {
  char* p = malloc(16);
  free(p);
  while (*p++ = *arg++);
  free(p);
}
```

Existing approaches

- Use a safe language!
- Mitigations: ASLR, DEP, stack canaries
- Memory debugging tools:
  - Valgrind
  - gcc and llvm memory sanitizer
  - SAFEcode
  - Ccured
  - SafeC
  - Cyclone
  - Etc...
  -
  -

A word on notation

```
red.is->compiler_generated(code);
```

```
black.is->user_written(code);
```

Object-based approach

```
if (IsPoisoned(address)) {
  ReportError(address);
}
```

```
*address = ...;  // or: ... = *address;
```

Intrastructural safety

```
struct {
  char id[8];
  int account_balance;
} bank_account;
char* ptr = &(bank_account.id);
strcpy(ptr, "overflow...");
```

Pointer-based approach

- Every pointer represented by three words: pointer value, base, bound
- We call that a „fat pointer“

Meet SoftBoundCETS

- Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, Steve Zdancewic; UPenn
- SoftBound: spatial safety
- CETS: temporal safety
- Uses disjoint fat pointers
- Proof of correctness (but caveat emptor)
- Implemented as LLVM optimizer pass

Meet SoftBoundCETS

- Source compatibility
- Complete coverage
- Separate compilation
- Low overhead

Spatial: instrumenting dereference

```
check(ptr, ptr_base, ptr_bound,
      sizeof(*ptr));

value = *ptr;
```

Spatial: implementation of check

```
void check(ptr, base, bound, size)
{
  if ((ptr < base)
      || (ptr + size > bound)) {
    abort();
  }
}
```

„Beware of bugs in the above code; I have only proved it correct, not tried it.“

– Donald E. Knuth

Spatial: correct implementation of check

```
void check(ptr, base, bound, size) {
  if ((ptr < base)
      || (ptr + size > bound)
      || (ptr + size < ptr)) {
    abort();
  }
}
```

Spatial: memory allocation

```
ptr = malloc(size);
ptr_base = ptr;
ptr_bound = ptr + size;
if (ptr == NULL) ptr_bound = NULL;
```

Spatial: stack allocation

```
int array[100];
ptr = &array;
ptr_base = &array[0];
ptr_bound = ptr_base +
                sizeof(array);
```

Spatial: Pointer arithmetic

```
newptr = ptr + index;
// or &ptr[index]

newptr_base = ptr_base;
newptr_bound = ptr_bound;
```

Spatial: narrowing

```
struct { ... int num; ... } *n;

...

p = &(n->num);


p_base = max(&(n->num), n_base);
p_bound = min(p_base + sizeof(n->num),
                    n_bound);
```

Spatial: more narrowing

```
struct { ... int arr[5]; ... } *n;
...
p = &(n->arr[2]);

p_base = max(&(n->arr), n_base);
p_bound = min(p_base + sizeof(n->arr),
              n_bound);
```

Spatial: loading metadata

```
int** ptr;
int* new_ptr;
...
check(ptr, ptr_base, ptr_bound,sizeof(*ptr));

newptr = *ptr;

newptr_base = table_lookup(ptr)->base;
newptr_bound = table_lookup(ptr)->bound;
```

Spatial: storing metadata

```
int** ptr;
int* new_ptr;
...
check(ptr, ptr_base, ptr_bound, sizeof(*ptr));

(*ptr) = new_ptr;

table_lookup(ptr)->base = newptr_base;
table_lookup(ptr)->bound = newptr_bound;
```

•

Spatial: augmenting function calls

```
int func(char* s)
{ ... }
int value = func(ptr);
```

```
int func(char* s,void* s_base,void* s_bound)
{ ... }
int value = func(ptr, ptr_base, ptr_bound);
```

Spatial: loose ends

- Global variables
- Separate compilation and library code
- Memcpy()
- Function pointers
- Creating pointers from integers
- Arbitrary casts and unions
- Variable argument functions

Temporal: allocation

```
ptr = malloc(size);
ptr_key = next_key++;
ptr_lock_addr = allocate_lock();
*(ptr_lock_addr) = ptr_key;
freeable_ptrs_map.insert(ptr_key,
ptr);
```

Temporal: dereference check

```
if (ptr_key != *ptr_lock_addr)
{ abort(); }

value = *ptr;
```

Temporal: pointer arithmetic

```
newptr = ptr + offset;
// or &ptr[index]

newptr_key = ptr_key;
newptr_lock_addr = ptr_lock_addr;
```

Temporal: free

```
if (freeable_ptrs_map.lookup(ptr_key) != ptr) {
  abort();
}
freeable_ptrs_map.remove(ptr_key);

free(ptr);

*(ptr_lock_addr) = INVALID_KEY;
deallocate_lock(ptr_lock_addr);
```

Temporal: propagating metadata

```
int** ptr;
int* newptr;

if (ptr_key != *ptr_lock_addr) { abort(); }

newptr = *ptr;

newptr_key = table_lookup(ptr)->key;
newptr_lock_addr = table_lookup(ptr)->lock_addr;
```

Temporal: propagating metadata

```
int** ptr;
int* newptr;

if (ptr_key != *ptr_lock_addr) { abort(); }

(*ptr) = newptr;

table_lookup(ptr)->key = newptr_key;
table_lookup(ptr)->lock_addr = newptr_lock_addr;
```

Temporal: globals

```
int var; // global variable
ptr = &var;

ptr_key = GLOBAL_KEY;
ptr_lock_addr = GLOBAL_LOCK_ADDR;
```

Temporal: loose ends

- Threads. Shared state is evil. Zalgo, etc.
- Shared memory. Shared state... see above.

Own contribution

- Introduced two function attributes to control instrumentation process
- Ported SoftBoundCETS to FreeBSD
- Instrumented FreeBSD libc and executable startup code
- Deleted ton of now useless wrappers
- Goal: build all of FreeBSD world with safe memory access
- Status: PoC works!

Demo Time!

- Everybody loves a good demo!
- Sufficiently advanced technology is indistinguishable from a propery rigged demo.

Thanks and references

- SoftBoundCETS: http://www.cs.rutgers.edu/~santosh.nagarakatte/softbound/
- SoftBoundCETS on FreeBSD: https://github.com/andreas23/freebsd/tree/softbounds

- Many thanks to Hannes Mehnert for joint work on FreeBSD port
- Many thanks to Santosh Nagarakatte, Milo M. K. Martin, Steve Zdancewic for answering questions and providing access to beta versions of code