Automated Code Repair

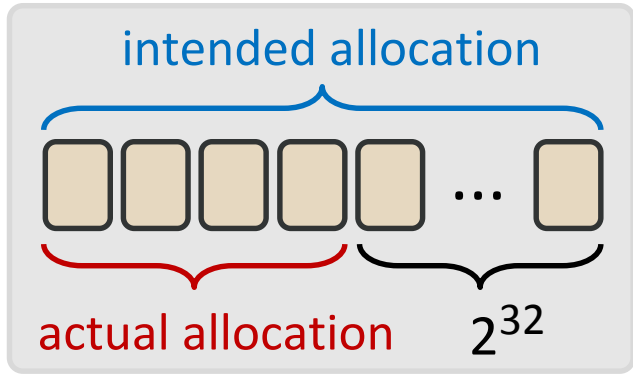
Will Klieber

December 2018

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-1391

Premises for Automated Code Repair (ACR)



1. Many security bugs follow common patterns.
   - E.g., "`p = malloc(n * sizeof(T))`" where *n* is attacker-controlled.
   - Integer overflow ⟹ too little memory gets allocated.
2. By recognizing such a pattern, it is possible to confidently guess the developer's intention (the ***inferred specification***).
   - E.g., "Try to allocate enough memory to hold *n* objects of type *T*".
3. It is possible to repair the code to satisfy this inferred specification.
   - Check if overflow occurs; if so, simulate `malloc` failing (by returning NULL).

What about false positives? Okay to 'repair' them, if we don't break code. Performance-critical parts might require manual review.

Why not repair as a compiler pass? Requires permanent change to build process, and it is very hard to audit the results.

Repair of integer overflow in array bounds check

Repair: **UADD(start, n)** /* UADD is defined below */

```
if (start + n <= dest_size) {
  memcpy(&dest[start], src, n);
} else {
  return -EINVAL;
}
```

Example: copy *n* bytes from *src* to *dest*, starting at index *start* of *dest*.

```
1.  inline static size_t UADD(size_t lop, size_t rop) {
2.      size_t result; bool overflow;
3.      overflow = __builtin_add_overflow(lop, rop, &result);
4.      if (overflow) {result = SIZE_MAX;}
5.      return result;
6.  }
```

Inferred specification:
inequality comparisons involving array indices or bounds should behave as in normal arithmetic (not modular arithmetic).

Normal arithmetic can be emulated using **saturation arithmetic**.

Experimental results for repair of integer overflows

Xi Wang et al., OSDI 2012 -- uses SMT solver Boolector

	Overflows reported by Kint	Overflows that are *sensitive*	Overflows fully repaired
OpenSSL (1.0.2g)	969	233	180 (77%)
Jasper (1.900.1)	481	101	53 (52%)

An overflow is ***sensitive*** if it involves variables associated with array indices or bounds.

Some of the 'repairs' are actually false positives (i.e., operation never overflows). Then our 'repair' just adds a little overhead. **It never breaks working code.**

Other repairs fix known vulnerabilities.

Inserting missing bounds checks

Ultimate goal: Repair program to enable proof of memory safety.

When the result of pointer arithmetic is used to access memory, ensure that it is still within the bounds of the original allocated region.

```
v = calloc(n, sizeof(int));
...
scanf("%i", &k);
for (i=0; i < k; i++) {
   if (v[i] < 0) break;
   v[i] += 1;
}
```

Repair: Insert bounds check

```
if (i >= n) abort();
```