CCured Lecture Notes

Jared Verdi – CSE 509, 2/22/2007

INSECURE CODE (buffer overflows)	→	CCURED REWRITE (inserts runtime checks)	→	SECURE CODE (buffer overflows eliminated)

Motivation

- Bounds checking is a common mistake among C programmers
- Too many bugs to fix manually
- Sometimes it's just too hard to write a good static analysis

Memory Safety

CCured aims to achieve even stricter - type safety

Arguably changes code infection vulnerability to a DoS vulnerability.

Divide C language into three sublanguages:

1. SAFE – just like C except:

   - No casts (except pointer → int)
   - No pointer arithmetic
   - No arrays
   - No unions

   ### Making SAFE safe

   - NULL pointers → insert NULL checks
   - Uninitialized pointers → always init to NULL
   - Dangling pointers (already been freed): Never free() memory, can add garbage collector to reclaim some memory

```
int x;
int *p = &x;
char *q = malloc(1);
int y = (int) q;
free(q);
*q = 5;
```

```
int x;
int *p = &x;
char *q = malloc(1);
int y = (int) q;
// no more free
// free(q);
if ( q == NULL )
     abort();
*q = 5;
```

2. SEQ – SAFE plus the addition of the following:

- pointer arithmetic
- casts from int to pointer
- arrays

Making SEQ safe

- still need to init all pointers, no free, NULL check
- bounds check pointers:
    - pointer = < base, size, p >
    - int = < i >
- cast of int to pointer results in un-dereferenceable pointer: < 0, 0, p >
    - (may break some correct programs)

```
int *p = malloc(10);
int x;
int *q = &x;
p += 5;

*p = 0;
```

```
int *t = malloc(10);
int *p = <t, t==NULL?0:10, t>;
int x;
int *q = <&x, sizeof(x), &x>;
p = <t, 10, t+5>;

if( p == NULL || p < t ||
    p > (t + 10 - sizeof(*p))
{
    abort();
}

*p = 0;
```

Converting between SEQ pointers and SAFE pointers

SEQ to SAFE:

```
if( P_SEQ < base ||
    P_SEQ >= (base + size - sizeof(*p))
{
    abort();
}
P_SAFE = P_SEQ;
```

SAFE to SEQ:

```
P_SEQ = < P_SAFE, sizeof(P_SAFE), P_SAFE >;
```

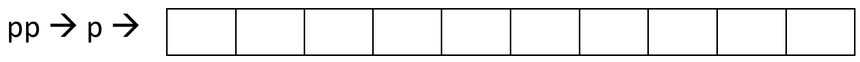
3. DYNAMIC (aka WILD) – SEQ plus the addition of the following:

- casts
- unions

  Need a way to differentiate ints from pointers:
  LSB of every word indicates whether it is a pointer or an int

  Runtime check for type of word

```
int **pp;
int *p;
p = malloc(10);
pp = (int **) p;
```



RESULTS

- programs is free of buffer overflows
- overhead - runs about 50% slower on average
- may require substantial modification to program
- incompatibility with shared libraries not also compiled with CCured due to “fat pointers”