1) Make it hard for the attacker to exploit buffer overflow
   • ASR
   • Model Checking to catch abnormal execution

2) Maintain bounds information for each buffer / pointer & instrument the program so that overflows are caught early.
   • FAT Pointers
     int sum (int *p, n)
     {
       int sum = 0;
       for(int i = 0; i < n; ++i) {
         sum += *(p+i);
       }
       return sum;
     }
     
     safe (compiler changes code to do this:)
     int sum (int *p, n, int *p_lo, int *p_hi)
     {
       int sum = 0;
       for(int i = 0; i < n; ++i) {
         assert(p_lo <= p + i < p_hi)
         assert (p + i != null))
         sum += *(p + i);
       }
       return sum;
     }
     
     int *p;
     int *p_lo;
     int *p_hi;
     int *p_lo = p;
     int *p_hi = p + 2;
     int foo{
       int *p = malloc(2 * sizeof(int));
       int *p_lo = p;
       int *p_hi = p + 2;
     }
     
     free(p);
     p=0;
p_hi = 0;
p_lo = 0;

int p[3];
int *p_lo = p;
int *p_hi = p + 3;

struct foo{
    int *a;
    int n;
    int *a_lo;
    int *a_hi;
};

struct foo * p = malloc(sizeof(struct(foo)));
struct foo * p_lo = p;
struct foo * p_hi = p+1;
p->a = malloc(2 * sizeof(int));
p->a_lo = p->a;
p->a_hi = p->a + 2;

int foo (struct foo *p,
    struct foo *p_lo,
    struct foo *p_hi)
{
    assert (p_lo <= p < p_hi);
    assert (p != null);
    assert (p->a_lo <= p->a < p->a_hi);
    assert (p->a_hi!= null);
    ...
}

int sum (int *p, n)
{
    int sum = 0;
    for(int i = 0; i < n; ++i){
        sum += *(p+i);
    }
    return sum;
}

int sum (int *p, n, int *p_lo, int *p_hi)
```c
{ int sum = 0;
 for(int i = 0; i < n; ++i){
    assert(p_lo <= p + i < p_hi);
    assert (p + i != null));
    sum += *(p + i);
 }
 return sum;
}

int sum (int p, n)
{
    int sum = 0;
    for(int i = 0; i < n; ++i){
        assert(p <= p + i < p + n);
        assert (p + i != null));
        sum += *(p+i);
    }
    return sum;
}

int *p = malloc(2 * sizeof(int));
BND (p, p+2);
p_lo = p;
p_hi = p+2;
p_lo' = p;
p_hi' = p+1;
sum (p,1);

if p is a pointer such that BND(p_lo, p_hi) and we pass it to a context which expect the bounds to
be BND(p_lo', p_hi') then it must be the case that:
p_lo <= p_lo' <= p <= p_hi' <= p_hi  (check at call site)

struct foo{
    int *a; //BND(a,a+n0 where n is another field in the same
    struct.
    int n;
};

int bar(struct foo* p){ //write
    BND(p, p+1);
    assert (p <= p < p+1);
   assert (p != null));
    int *tmp = p->a; //BND(p->a, p->a+1);
    assert (p->a <= tmp < p->a+1);
    assert (tmp != null);
    int n = *(tmp);
}

- Advantages
  - Reverse compatibility
- Disadvantages
  - Overhead on assertions
```
Using static analysis you can remove assertions

- Programmer overhead
  - Type Inference (like OCAML & SML) for local variables