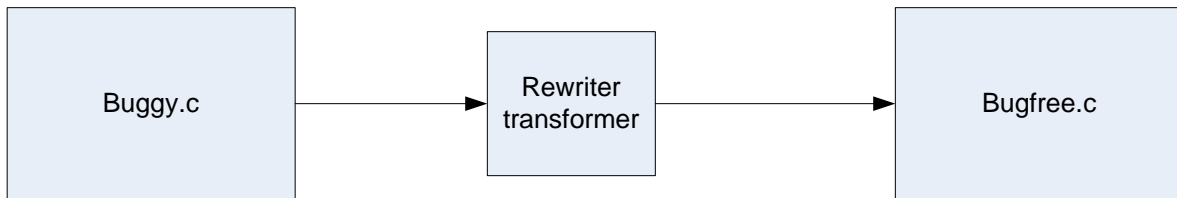


Program transformation



Desired Properties

- Bug-free code should continue working
- Bug-free code should not affect performance
- Output should be free of bugs
- Least manual effort/require little or no changes to source code
- Compatibility between transformed and untransformed code
- Require little or no annotation

Ccure-memory safety

Design as 3 subsets of C language:

SAFE – SEQUENCE (SEQ) - DYNAMIC

Pointer (ptr) operation in C:

- Can point anywhere
- Arithmetic ✓
- Dereference
- Casting between pointer types ✓
- Pointer to function
- ptr-to-int (cast/implicit conversion)
- int-to-ptr (cast/implicit conversion) ✓
- Assignment/parameter parsing
- Reference operator (&)
- Allocation/free
- NULL pointer
- void pointer

✓ : Dangerous operation

Temporal memory bugs

Example 1:

```
p = malloc(10);
free(p);
*p = 0;
```

Example 2:

```
int *foo(void)
{
    int x;
    return &x;
}

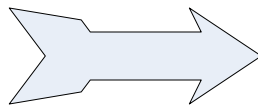
void bar(void)
{
    int *p = foo();
    *p = 0;
}
```

SAFE

- pointer to function
- dereference
- ptr-to-int cast
- assignment/parameter parsing
- reference operator (&), alloc
- null

```
void * memcpy(void *dst, void src, int size)
```

```
int *p;
int *q;
.
.
.
memcpy(void *p, void *q, sizeof(*q));
int foo(int &x) { x= 5};
```



Save because of implicit conversion (between pointers)

Runtime check:

- none
- maybe: check for null

SEQ = SAFE + arithmetic + int-to-ptr casts

- need bounds checking
- need to keep track of bounds

```
struct s {int x;}
```

```
struct p {int *p;}
```

```
struct s *s;
```

```
struct p *p;
```

```
s.x = <some codes>;
```

```
p = (struct p *)s;
```

```
*(p->p) = 0;
```

- No type_inconsistent aliasing
i.e: always know statically the type of data pointed to by every pointer

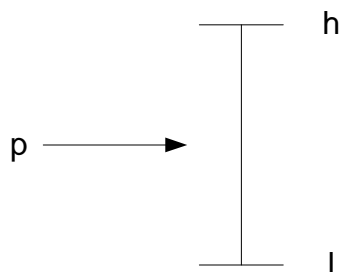
Need: Un-dereferable pointer

- cast from int-to-ptr yield an un-dereferable pointer

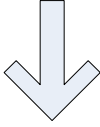
Pointer representation

<l, h, p>

l: low
bound
h: high
bound
p: pointer



*p = 0



```
If (p < l || p > h)
  abort();
```

```
*p = 0;
```

<l, h, 0> : problem when casting int-to-ptr
<0, 0, p>

SAFE

```
void foo(int *p)
{
    *p = 0;
    *p = 1;
    *p = 2;
}
```

SEQ

```
void bar(void)
{
    int A[5];
    int *p;

    p = A;
    p = p + 3;
    foo(p);
}
```

Conversion from SEQ to SAFE

```
p = <&A, &A[4], p>
bound check(<&A, &A[4], p>)
foo(p);
```