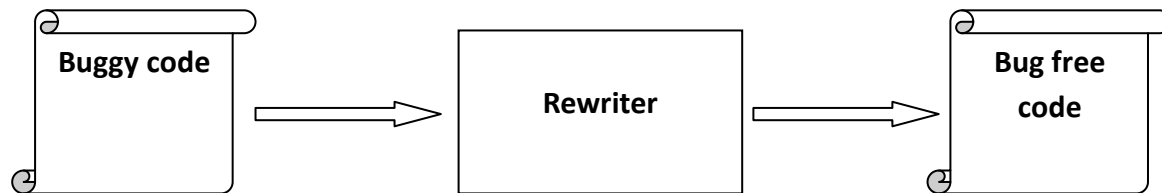


CSE 409/509: System Security- Lecture Notes

Lecture: **Static analysis: CCured**

Date: **3/6/09**

Program transformation:-



Desired properties:-

- Bug free code generated by the rewriter should continue working.
 - The rewriter should not impact performance (or less strictly have as little impact on performance as possible).
 - The output should be free of bugs.
 - There should be least manual effort involved i.e., the rewriter should need us to perform little or no changes to source.
 - Compatibility between transformed and untransformed code should be preserved.
 - The source should require little or no annotation.
-

CCured:-

Attempts to avoid memory errors.

Design as 3 subsets of C language:-

- SAFE
- SEQ
- DYNAMIC

Pointer operations in C:-

- Pointer arithmetic. ×
- Dereferencing.
- Cast between pointer types. ×
- Pointer to integer cast (including implicit conversions).
- Integer to pointer cast (including implicit conversions). ×
- Assignment/parameter passing.
- & (address of operator).
- Allocation/Free.
- NULL.
- void *

- Pointer to function.

x: Unsafe operations.

Temporal memory bugs:-

1. Using a pointer after it has been freed:-

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

2. Pointer escaping:-

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

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

CCure focuses on spatial memory bugs.

SAFE

Only the following pointer operations are allowed:-

- Dereferencing.
- Pointer to integer cast (including implicit conversions).
- Assignment/parameter passing.
- & (address of operator).
- Allocation/Free.
- NULL.
- Pointer to function.

Are we sure that with given set of operations, we need no static checks on dereference (i.e., everything remains within bounds)?

What about memcpy, if we pass incorrect 'sz'?

```
Using: void * memcpy(void * dst, void * src, int sz)
```

```
int * p;
int * q;
memcpy(p, q, sizeof(* q));
```

Since this involves implicit conversion between pointer types(int * to void *), which are not allowed, we are safe.

References in C++

- We can't pass NULL.
 - We can't do pointer arithmetic.
- Thus, these are like pointers in our SAFE language.

```
int foo(int &x){  
x=5;  
}
```

Thus, for SAFE, runtime checks:-

- None.
- Maybe check for NULL.

SEQ

= SAFE + Pointer arithmetic + Integer to pointer cast

- Needs bound checking, thus we need to keep track of bounds.
- No type inconsistent aliasing i.e., always know statically the type of data pointed to by every pointer.

```
struct S{  
int x;  
};
```

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

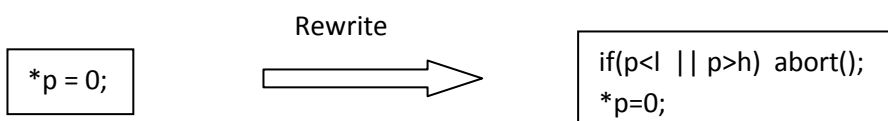
```
struct S s;  
struct P *p;  
s.x=<any value>;  
p=(struct P*)&s;  
*(p->p)=0;
```

- We need undereferencable pointers – casts from integer to pointer yield undereferencable pointers.

Pointer representation:-

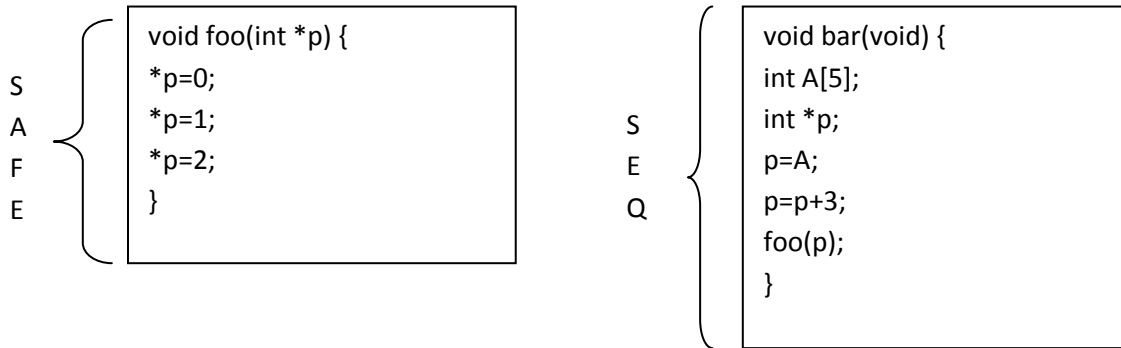
A pointer is just an unsigned int (stores a memory address), but to perform bounds checking we need some information about allowed bounds of the pointer.

Thus, pointers represented as: <l, h, p> i.e. <low bound, high bound, pointer>



Now, for undereferencable pointers, set l=h=0.

Conversion of pointers from SEQ to SAFE:-



`p ≈ <&A, &A[4], p>`

`foo(p);`

will be replaced by

`if(p<&A || p>&A[4]) abort(); foo(p);`

Conversion from SAFE to SEQ:-

SAFE pointers point to just one memory location, thus just keep that bound when converting to SEQ.