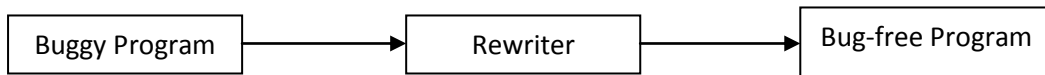


CSE 409/509: System Security- Lecture Notes Lecture on CCured

Date: 03/06/09

Submitted By: Gaurav Naigaonkar

Program Transformation:



Desired Properties:

- Bug free code (should continue working)
 - Shouldn't affect performance
 - Output should be free of bugs
 - Least manual effort/ Require little or no changes to source code
 - Compatibility: transformed code should be compatible with the untransformed code.
 - Require little or no annotation.
-

CCured

Design as 3 subsets of C language

- SAFE
- SEQ
- DYNAMIC

Pointer operations in C

- Arithmetic - (*unsafe*)
- Dereference
- Cast between pointer types - (*unsafe*)
- Cast pointer to int/ implicit conversion
- Cast int to pointer/ implicit conversion - (*unsafe*)
- Assignment / Parameter passing
- &
- Allocation/ Free
- NULL
- void *

Temporal Memory Bugs:

1. Using a pointer after freeing its memory:

```
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;
}
```

The paper on CCure focuses on spatial memory bugs and does not talk about temporal memory bugs. So such bugs can be ignored as of now.

The language with following operations can be considered SAFE:

- Dereferencing
- Pointer to integer cast
- Assignment / Parameter passing
- &
- Allocation / Free
- NULL
- Pointer to function

Runtime checks:

- None
- Maybe check for NULL

References on C++:

- Pointer arithmetic not allowed
- NULL can't be passed in place of an int ref.

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

SEQ = SAFE + arithmetic operations + int to ptr casts

- Needs bound checking. Therefore, need to keep track of bounds.
- Whenever two pointers point to same memory location, they agree on type. That's why casts between pointers still excluded in SEQ.
- 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;
```

- Integer to pointer cast can still be a problem
So we need: Pointer that cannot be dereferenced
(int-to-ptr casts yield undereferencable pointer)

Pointer Representation:

To perform bound checking, along with pointer store upper and lower bound information

< l, h, p >

l – lower bound

h – higher (upper) bound

p – pointer

Thus,

*p would be re-written as follows:

```
if(p < l || p > h)
```

```
    abort();
```

```
*p=0;
```

Undereferencable pointers can be represented as:

< 0, 0, p >

With these lower and upper bounds, such a pointer would not be able to be dereferenced

Conversion of pointers from SEQ to SAFE:

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);  
}
```

Here,

$p \approx \langle \&A, \&A[4], p \rangle$

Thus,

foo(p)

will be replaced by:

```
boundscheck(<&A, &A[4],p>)
```

```
foo(p);
```