

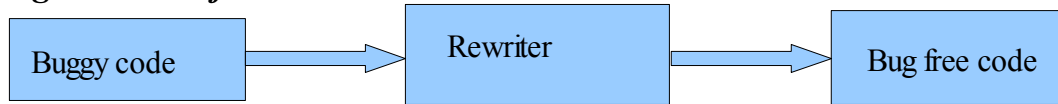
# CSE 409/ 509: Spring 2009, System Security

## Discussion on CCured

Ajay Venkateshan Krishnaprasad

### Class on 3/6

#### **Program Transformation:**



Desired properties of the Rewriter:

- -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.
- -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)
- -Assignment
- -Allocation/ free
- -Can point anywhere
- -Cast between pointer types (unsafe)
- -Cast pointer to int/ implicit conversion
- -Cast int to pointer/ implicit conversion (unsafe)
- -Dereference
- -&
- -NULL
- -void \*

#### **Temporal Memory Bugs**

1. Use of pointer after free

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

## 2. Pointer Escape

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

### i) *SAFE*

- -Pointer to function
- -Dereference
- -Pointer to int
- -Assignment
- -&,allocation
- -NULL

Runtime checks

-None

-May be, check for NULLs.

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

```
int *p;
```

```
int *q;
```

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

### \* *References in C++*

- -Does not allow pointer arithmetic
- -NULL can't be passed in place of an int ref.

```
int foo(int &x)
```

```
{
```

```
  x=5;
```

```
}
```

### ii) *SEQ*

- -SAFE+arithmetic+ int to pointer casts
- -Needs bound checking.
- -No type inconsistent aliases: always know statically the type of data pointed to by the pointer.

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

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

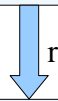
```
struct S s;
struct P *p;
s.x=<some value>
p=(struct *P) &s;
*(p->p)=0;
```

**Need:** a pointer that can't be dereferenced  
-Casts from integer to pointer yield a pointer that can't be dereferenced.

**Pointer representation**

<l,h,p>

```
*p=0;
```



rewrite

```
if(p<l||p>h)
    abort();
*p=0;
```

To get a pointer that can't be dereferenced - set 'l' and 'h' to 0  
<0,0,p>

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

void bar( void)
{
    int A, *p;
    p=A;
    p+=3;
    foo(p);
}

Fig.1
```

**Conversion from SEQ to SAFE (eg Fig.1**

```
p ~ <&A, &A+3,p>
boundscheck(<&A, &A[4],p>)
free(p);
```