INSTRUMENTATION TO PREVENT PROGRAMS FROM BUFFER-OVERFLOW ATTACKS

VISHAL CHEBROLU
AGENDA

- Buffer-Overflow Attack
- Outline of AddressSanitizer (ASan)
- Instrumentation for Read and Write accesses
- Pointer Aliasing problem
- Optimization
- Conclusion
BUFFER-OVERFLOW ATTACK

- Shell code execution
- Reordering execution of functions
- Application DoS
ADDRESS SANITIZER

- Tool to **REPORT** memory corruption errors
- Uses shadow memory
- Checks if memory is addressable in 8 byte chunks (Sane Check)
- **STOPS** the execution of program in times of error
INSTRUMENTATION FOR READ ACCESS

- Proxy pointers for primitive data types
- \( i1/i8/i16/iN : 0 \)
- Float/Double : 0.0
- Can be given as command line argument
INSTRUMENTATION FOR READ ACCESS

ENTRY

SANE CHECK

YES

NEXT

EXIT

NO

REPORT ERROR

ENTRY

SANE CHECK

YES

REL_BLOCK

ϕ((%A, SANE_CHECK), (%PROX_I8, REL_BLOCK))

NEXT

EXIT

NO

EXIT
INSTRUMENTATION FOR WRITE ACCESS

- Static buffers are moved to the heap by dynamic allocation
- Bounds are inferred for every dynamically allocated buffer through static analysis

```c
int *A = malloc(n*sizeof(int));
unsigned int A_size = n;
```
INSTRUMENTATION FOR WRITE ACCESS

- Static buffers are moved to the heap by dynamic allocation
- Bounds are inferred for every dynamically allocated buffer through static analysis

```c
int *A = malloc(n*sizeof(int));
unsigned int A_size = n;

%1 = load i64, 64* %n
%mul = mul i64 %1, 4
%call = call i8* @malloc(i64 %mul)
%2 = bitcast i8* %call to i32*
store i32* %2, i32** %A

%1 = load i64, 64* %n
%mul = mul i64 %1, 4
%call = call i8* @malloc(i64 %mul)
%2 = bitcast i8* %call to i32*
store i32* %2, i32** %A
store i64 %1, i64* A.size
```
INSTRUMENTATION FOR WRITE ACCESS

- Write A[i]. What if “i” is not initialized?
- A check which let us decide whether to expand the bounds or not
- SCALE_OF_RELOCATION (default value is 2)

A[i] = x;
INSTRUMENTATION FOR WRITE ACCESS

- Write A[i]. What if “i” is not initialized?
- A check which let us decide whether to expand the bounds or not
- SCALE_OF_RELOCATION (default value is 2)

```
A[i] = x;
if(i <= SCALE_OF_RELOCATION*A_size) {
    A_size = A_size*SCALE_OF_RELOCATION;
    A = realloc(A, A_size);
}
A[i] = x;
```
INSTRUMENTATION FOR WRITE ACCESS

- Split a store statement into a separate basic block

```
store i32 0, i32* p
%1 = load i32, i32* p
....

store i32 0, i32* p
br %next
```

```
next:
%1 = load i32, i32* p
....
```
INSTRUMENTATION FOR WRITE ACCESS

ENTRY

SANE CHECK
- NO: REPORT ERROR
- YES: NEXT

NEXT

EXIT

ENTRY

SANE CHECK
- NO: REL CHECK
- YES: RELOC BLOCK

RELOC BLOCK
- NO: WRITE
- YES: NEXT

WRITE

EXIT
In case of pointer aliasing and a relocation occurs, all the aliasing pointers need to be updated.
If case of pointer aliasing and a relocation occurs, all the aliasing pointers need to be updated.
POINTER ALIASING PROBLEM

int *p,*q,*r,*s;
Allocation of memory

int *p = malloc(8*sizeof(int));
POINTER ALIASING PROBLEM

- Pointer Assignment

```c
int* q = p;
```
POINTER ALIASING PROBLEM

- Pointer Assignment

```c
int* r = q;
```
POINTER ALIASING PROBLEM

- Pointer Assignment

```c
int* r = q;
```
POINTER ALIASING PROBLEM

- Pointer Assignment

Diagram showing pointer assignment with nodes labeled B, A, C, D, R, S, P, Q, and T.
Write to memory

\[ p[i]/q[i]/r[i]/s[i] = x; \]
Write to memory

\[ p[i]/q[i]/r[i]/s[i] = x; \]
Write to memory

p[i]/q[i]/r[i]/s[i] = x;
Write to memory

\[ p[i]/q[i]/r[i]/s[i] = x; \]
Free Memory

The call to free() function can just be skipped
POINTER ALIASING PROBLEM

- Free memory

```c
free(p/q/r/s);
```
Free memory

```c
free(p/q/r/s);
```
POINTER ALIASING PROBLEM

- Free memory

```c
free(p/q/r/s);
```
Free memory

```c
free(p/q/r/s);
```
POINTER ALIASING PROBLEM

- Free memory

```c
free(p/q/r/s);
```
Live Variable Analysis

- Calculates variables which are live at each point
- Aids to reduce the number of pointers to deal with
- Substantial improvements in large monolithic function programs
Legacy code can be safely reused

Buffer-overflow attacks can be eradicated by dynamically expanding memory during run-time

Can devise a mechanism for inter function memory communication

Use-after-free, Invalid free, Double free errors can also be mitigated
Thank You!