Bound Checking to avoid buffer overflow

Goals: Space, Time Efficient, Backward Compatibility

Components
- **Buddy Allocator**
  - Make sure allocated size = $2^k$
  - Minimum size = slot size
  - No two objs share the same slot

Bound Table
- Keep $\log$ of the "allocated size" *NOT obj. size*

Virtual mem address $\rightarrow$ Bound Table

Ex: $\text{buff}[28] >> 4 \rightarrow \text{table}[A1]$
  = $\text{buff} + 0 \times 16$
  = $0 \times 1ff...A16$

Checking
Bound checking on p' pointer derived from pointer p

$p' = p + 30$

Arithmetic
- In bound OK
- Over bound $p' > slot \_ size \rightarrow$ set 00B bit = most significant bit
- O.W. $\rightarrow$ raise an ERROR = terminate program

Dereference
- Some example
  - $\text{buff}[65] = 'A' \rightarrow$ ERROR
  - $(\text{buff} + 30) = 'A'$
  - $\text{buff}[65] = 'A'$ work fine

Baggy Bound
- User can use this much space for this obj.
- Initialize table slot with $31$ for backward compatibility.
- Pointer to obj allocated by external uninstrumented library will not raise issue when baggy bound code try to access.
- Deallocate also set the slot to $31$ - mark available.
- $\text{size} = \text{table}[p >> 4]$
- Base = $p \& \sim (\text{size} - 1)$

$\log(\text{slot size})$ clear low bits of p

$\#$ $p = \text{ERROR}$ bc. trying to access an inaccessible page (High memory start with)

out of bound
Buffer Overflow + Lab 1

(doesn't have much time to cover)

Please make sure you understand what you are doing in Lab 1

```
high
    input-addr
    arg1
    return
    old ebp
    num
    buff

low
    overflow
```

**Attack** to change control flow

- overwrite return address
- overwrite function pointer
- change data value

```
Ex. void func(int arg1, char* input) {
    int num;
    char buff[40];
    strcpy(buff, input);
}
```

Some solutions

1. Stack Canaries
   - compiler mechanism inserting canary before return address on the stack
   - Canary Property: random
   - contain NULL termination
   - to avoid attack writing further if he can guess canary
   - bc. most function handling string will terminate
   - gets()

2. ASLR
   - random the address space so that attacker can't easily hardcode the address.

3. NX
   - Non executable → can't execute code on the stack.
7. [6 points]:

The program prints `0x1001005a`. What was N? Hint 1: there is only one correct answer. Hint 2: recall that Bagg is manipulating certain bits in pointers for bookkeeping.

```
0x1001005a = p + 0xN + N
```

16 x = slot address = 0x--- 0

```
0x1001005a = 0x1001006b + N
```

\[ N = -18 \]

This question wasn't covered during exam review but many students want detailed solution.

2017

IV. Bagg bounds checking

Consider a system that runs the following code under the Bagg bounds-checking system, as described in the text paper by Shridhar et al., with this change:

```
struct var {
    char data;
    void *ptr;
} 

void handle void;

void bagg(int a, void * v) {
    if (a = handling)
        get石油化工;
    else {
        void team1(void);
        struct v a;
        bagg(a, buf[f1, v]);
    }

    void team2(void);
    struct ab a;
    bagg(a, buf[b, v]);
}
```

```
\[ \text{Bagg Bound} \]
```

Assume the compiler performs no optimizations and places variables on the stack in the order declared. The stack grows down (from high address to low address), that this is a 32-bit system, and that the address of handle is known.

9. [4 points]:

A. True / False: If function test1 is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

B. True / False: If function test2 is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

C. True / False: If function test3 is called, an adversary can construct an input that will cause the program to jump to an arbitrary address.

For the next four questions, determine what is the minimum number of bytes that an adversary has to provide as input to cause the program to likely crash. When defining different test functions, do not count the newline character that the adversary has to type in to signal the end of the file to test a Bagg fault in the memory denoting shifts in access bytes.

10. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running test2 to crash?

11. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running test3 to crash?

12. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running test4 to crash?

13. [4 points]: What is the minimum number of bytes that an adversary has to provide as input to likely cause a program running test5 to crash?