14 – 10/23/14 – Machine Programming: Attacks and Defenses

Madhu Vijay, Marwan Kallal, Rebecca Chen, Jordan Canedy

- f62
  - return a + b
- f65
  - return a[i]+a[j], for an int or unsigned array
- f64
  - return *a+*b
  - With less optimization, we would see "movl 8(%ebp), %eax" instead of "movl 4(%esp), %eax"
    - The optimizer can get rid of this in this case (but not in all cases) because the function isn't calling another function, so it doesn't have to save the value of %ebp
- f66
  - Moves through structs of size 8
  - Have an array of structs (which is the first argument, %edx)
  - The indices are the second argument (which is 8(%esp)) and the 3rd argument (12(%esp))
  - int f(struct pair* array, int ai, int bi) {return array[ai].b + array[bi].b;}
- f67
  - Returns a pointer to the ai-th element of the array
    - The instruction lea loads the address rather than dereferencing the value
  - int* f(struct pair* array, int ai) {return &array[ai].a;}
- f77

  -
    ```
    int f(const int* v, int n) {
        int sum = 0;
        for (int i = 0; i < n; ++i)
            sum += v[i];
        return sum;
    }
    ```
- stacksmash.c
  - Uses "gets()", which doesn't check for the end of the buffer – could lead to buffer overflow
    - Man page for "gets" says "never use gets"
  - E.g. you can overwrite the return address for another function
    - %esp (beginning of the main function) is at some address – at that address, there's a return address
    - Before %esp, there's some space to store temporary variables
    - Before that, there's 100 characters of a buffer
    - Since the buffer is at the same region of memory (and is before the other things), we can intentionally modify the temporaries and the return address
  - We can figure out that the difference between the start of the buffer and the return address is 112 – so, we need a 112-character string, followed by a misleading return address (we'll use "\0\0\0\0")
    - We expect this to take our input, say something like "Read 116 characters," try to return to the address 0, and then segfault as a result (on return)

        - However, instead, it detects a buffer overflow attack, and promptly destroys the function
            - Note that we crash <u>before</u> we get past the "gets" line (not just on the return)
    - We can figure out what's happening by backtracing
        - It turns out that there's another function, __gets_chk, that takes the input and the size of the buffer as inputs – if it sees an input that is larger than the buffer, it ends
        - How does "gets" know the size?
            - gcc takes unsafe code and makes it safe, by calling another function ("__bos") which checks the size of the string and calls either __gets_chk (with the string and the size) or the regular function __gets_warn (with just the string), which just does what "gets" does normally
- How can we fool gcc? See stacksmashf.c
    - We create a separate function "read_line" that just takes a char* as an argument, rather than an array with definite size – this hides the size from gcc
        - If we disassemble read_line, we see that read_line calls the real "gets," not "__gets_chk"
    - This still detects a stack smash and exits
    - Stack diagram:
        - "canary"
        - Old %edi
        - Old %ebp
        - Return address at %esp
    - "thread local storage area" – stores the canary, and various other things that differ among threads
        - The canary is like our off-by-one detector from pset 1 – it checks for overwrites, using "xor"
        - The canary, at %eax, has a different value each time you run it
        - The canary is placed on the stack, <u>in between</u> the buffer and the return address – so, a typical buffer overrun that tries to modify the return address will have to modify the canary, so the program can recognize the canary overwrite and quit
        - Compiler puts in a canary every time it sees the possibility of a buffer overflow attack (e.g. in this case it sees a character array)
- See GNUmakefile – UNSAFE_CFLAGS basically returns you to the 1990s and excludes major safety protections
    - So, if you run ./stacksmash.unsafe with the right input, you can force a segfault
    - See ./runevil – can use this to control what the program does