Introduction to Cryptography and Software Security
Problem Set 3: Low-Level Software Vulnerabilities

Due date: 16/5/2019

Please provide your solution by completing the on-line quiz.

Part I

Answer questions 1–5 by choosing the correct answer for each question.

1. The stack is used for storing
   - Local variables.
   - Global variables.
   - Dynamically-allocated variables.
   - Program code.
2. How does a buffer overflow on the stack facilitate running injected code?
   - By writing directly to the instruction pointer register the address of the injected code.
   - By overwriting the return address to point to the address of the injected code.
   - By overwriting the stored stack frame pointer.
3. What is a nop sled?
   - A sequence of nops preceding injected code, useful when the return address is unknown.
   - A method for removing zero bytes from shellcode.
   - A branch instruction at the end of a sequence of nops.
4. Exploitation of the Heartbleed bug permits
   - A format string attack.
   - A read outside the bounds of a buffer.
   - Overwriting the server's memory.
5. An integer overflow occurs when
   - There is no more space to hold integers in the program.
   - An integer's expression result "wraps around".
   - An integer is used as a pointer.
   - An integer is used to access a buffer outside its bounds.

We thank Prof. Michael Hicks (UMD) and Coursera for generously sharing some of their teaching resources with us.

Part II

Consider the following program and answer questions 6–9 by choosing the correct answer for each question.

```
int main(int argc, char *argv[]) {
    char out[100];
    char buf[1000];
    char msg[] = "Welcome to the argument echoing program\n";
    int len = 0;
    buf[0] = '\0';
    printf(msg);
    while (argc) {
        sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);
        argc--;
        strncat(buf,out,sizeof(buf)-len-1);
        len = strlen(buf);
    }
    printf("%s",buf);
    return 0;
}
```

6. Which one of the following variables is vulnerable to a buffer overflow attack?
   - `len`.
   - `out`.
   - `msg`.
   - `buf`.

7. What line of code can overflow the vulnerable buffer?
   - `printf(msg)`.
   - `sprintf(out,"argument %d is %s\n",argc-1,argv[argc-1])`.
   - `strncat(buf,out,sizeof(buf)-len-1)`.
   - `len=strlen(buf)`.
   - `printf("%s",buf)`.

8. Which of the following one-line changes, on its own, will eliminate the vulnerability?
   - Change `printf("%s",buf)` to `printf(buf)`.
   - Change `strncat(buf,out,sizeof(buf)-len-1)` to `strlcat(buf,out,sizeof(buf))`.
   - Change `printf(msg)` to `printf("%s",msg)`.
   - Change `sprintf(out,"argument %d is %s\n",argc-1,argv[argc-1])` to `snprintf(out, 100,"argument %d is %s\n",argc-1,argv[argc-1])`.

9. If we change `printf("%s",buf)` to `printf(buf)` then
   - The program would be vulnerable to a format string attack.
   - The program would be vulnerable to a heap overflow attack.
   - The program would be vulnerable to a use-after-free attack.
   - All of the above.

Part III

Answer questions 10–14 by choosing the correct answer for each question.

10. When a program indexes a buffer after a pointer to that buffer has been used as a parameter to the `free()` function, this is
    - Correct behavior.
    - An information flow violation.
    - A violation of temporal memory safety.
    - A violation of spatial memory safety.

11. An engineer proposes that in addition to making the stack non-executable, your system should also make the heap non-executable. Doing so would
    - Not make programs more secure, since the heap cannot contain attacker-controlled data.
    - Ensure that dynamically-allocated memory is always deallocated.
    - Ensure that only the correct amount of data was written to a heap-allocated block, preventing heap overflows.
    - Make programs more secure by disallowing another location for an attacker to place executable code.

12. Consider the following code:

```
char *foo(char *buf) {
    char *x = buf+strlen(buf);
    char *y = buf;
    while (y != x) {
        if (*y == 'a')
            break;
        y++;
    }
    return y;
}

void bar() {
    char input[10] = "leonard";
    foo(input);
}
```

    The definition of spatial safety models pointers as triples (p, b, e) where p is the actual pointer, b is the base of the memory region the pointer is allowed to access, and e is the extent of that region. Assuming characters are one byte in size, what is the triple (p, b, e) for the variable `y` when it is returned at the end of the code?

    - $(\texttt{\&input} + 4, \texttt{\&input}, \texttt{\&input} + 7)$.
    - $(\texttt{\&input} + 4, 0, \texttt{sizeof(input)})$.
    - $(\texttt{\&input} + 4, \texttt{\&input}, \texttt{\&input} + 10)$.
    - $(\texttt{y}, \texttt{\&input}, \texttt{buf})$.

13. Recall that classic enforcement of CFI requires adding labels prior to branch targets, and adding code prior to the branch that checks the label to see if it's the one that is expected. Consider the following program:

```
int cmp1(char *a, char *b) {
    return strcmp(a,b);
}

int cmp2(char *a, char *b) {
    return strcmp(b,a);
}

typedef int (*cmpp)(char*,char*);

int bar(char *buf) {
    cmpp p;
    char tmpbuff[512] = { 0 };
    int l;

    if(buf[0] == 'a') {
        p = cmp1;
    } else {
        p = cmp2;
    }
    printf("%p\n", p);
    strcpy(tmpbuff, buf);
    for(l = 0; l < sizeof(tmpbuff); l++) {
        if(tmpbuff[l] == 0) {
            break;
        } else {
            if(tmpbuff[l] > 97) {
                tmpbuff[l] -= 32;
            }
        }
    }
    return p(tmpbuff,buf);
}
```

To ensure that the instrumented program runs correctly when not being attacked, which of the following functions must always be given the same label? Choose at least two functions.

- `printf`.
- `cmp1`.
- `cmp2`.
- `strcpy`.
- `bar`.

14. When enforcing Control Flow Integrity (CFI), there is no need to check that direct calls are compatible with the control flow graph because

- Programs that use CFI do not have any direct calls.
- Attackers are not interested in corrupting direct calls.
- CFI should be deployed on systems that ensure the code is immutable.
- Direct calls always share the same label.