Quiz 3 (Nov 11)

Questions 1-3. Side channel attack: Pages 4-10.
See page 4's code. A timing-based side-channel attack is possible if any input results in a different running time.

```c
// ./t PasSwOrD
int main(int argc, char** argv) {
    char* input = argv[1];
    char buf[1024] = { 0, };
    char buf_concat[1024] = { 0, };
    unsigned char md[MD5_DIGEST_LENGTH];  // MD5 digest is 16 bytes

    input[8] = 0;
    for( int i = 0; i < strlen(input); i++ ) {
        buf[0] = input[i]; buf[1] = 0;

        for ( int j = 0; j < 2000; j++ ) {
            compute_md5(buf, md); get_md5(md, buf);
        }
        strcat(buf_concat, buf);
        if( answers[i][0] != buf[0] || answers[i][30] != buf[30] ) {
            printf("Incorrect.\n");
            return 0;
        }
    }

    compute_md5(buf_concat, md);
    get_md5(md, buf);
    if( strcmp(buf, "3c85a6013778f3325c8035860a22a82e") == 0 )
        printf("Correct password.\n");
    else
        printf("Incorrect.\n");

    return 0;
}
```

In this example, the code below exits the "for" loop early if it finds out the input is not an answer. This would be a good optimization, saving computational resources that it does not need to do, but the shortened execution time can be exploited to know how many characters were correct from the input.

```c
if( answers[i][0] != buf[0] || answers[i][30] != buf[30] ) {
    printf("Incorrect.\n");
    return 0;
}
```

Such a functionality is typically called "**early exit,**" and this is also a typical side channel source.

Remember that many crypto algorithms do not have predicates that have different computational times depending on the result.
The predicate (if-then-else) itself is not a problem, but if the logic in the true and false branches take different times, that could be a source code side channel attack.

Questions 4-6. Speculative Execution and Spectre Attack: Pages 41-53.
When there is a predicate where knowing whether it takes a true or false branch takes a long time, the speculative execution is an optimization method that runs both true/false branches secretly, and when the branch outcome is available, it will simply use one of the executions it ran secretly, while discarding the branch's execution that is not taken.

During the speculative execution, (1) no errors will be reported because it is not an actual execution yet. If it is not chosen, errors should also be discarded, (2) privileged memory accessible during the speculative execution, (3) caches are used during the speculative execution, (4) when a speculative execution is discarded, related caches are not discarded.

Quiz 3 (Nov 11)

See the code below.
It is important to have the privileged buffer (or kernel buffer, buf_to_victim in this code) to be an index of the buf[].
This is because, after the speculative execution, we need to test the caches to know the secret.
If we directly use buf_to_victim without buf, to test the cache, we need to access buf_to_victim directly, which will raise the invalid memory access error.
By using the buf_to_victim[] as an index, we affect the cache of buf[] based on the values of buf_to_victim[].
Then, we can test buf[] to know the secret.

```
void process_a()
{
    unsigned byte buf_to_victim[1024]; // @ 0xBFFFE000
    unsigned byte buf[255]; // uncached buffer
    // secret @ kernel memory: 0xC0000044
    // (0xC0000044 == &buf_to_victim[0x2044])
    print("%d", buf[ buf_to_victim[0x2044] ]);
}
```

In this code, the data type of buf, buf_to_victim does not matter.
It is important to make sure buf_to_victim and buf are not cached before the execution.
Whether the “buf” is on the stack or in global memory does not matter.

Question 7. API Hooking – Detours: Pages 14-17
Detours overwrites the first few instructions of a function with “jmp TARGET”, which takes 5 bytes in a 32-bit system (9 bytes in a 64-bit system).
If the first few instructions perfectly fit into 5 bytes, it just needs to copy them and use them as the original function. However, if the first few instructions do not fit into the 5 bytes, meaning that at the 5 byte, it is in the middle of some instruction, it needs to identify how long that instruction is, and copy the instruction as well.
Depending on the instruction, it can be 6, 7, or 8 bytes (or even longer) to copy.
This is why the Detours have its own disassembler.
However, even if it has a good disassembler, running those at runtime is not desirable (for the performance).
So, Microsoft changed its library functions’ code by inserting “mov edi, edi” into the beginning of their functions, to make the first three instructions fit perfectly in 5 bytes. This is a special preparation for hot patching, which uses detours.
“mov edi, edi” does nothing (equivalent to “nop”). (do not confuse with “xor edi, edi,” which sets 0 to edi).

Questions 8~10: ShadowWalker: Pages
A page table maintains a mapping between the page number (of a virtual memory) and the frame number (of a physical memory). Changing the mapping can immediately change the content of the virtual memory.
The ShadowWalker technique aims to change the content while maintaining the permissions (i.e., flags in the slide).
A key method in the ShadowWalker technique is that it can distinguish whether a page fault happens because of a read operation or instruction execution. Specifically, a page fault handler receives two key addresses: the faulty memory address (CR2) and the faulty instruction address (EIP).
Consider if an instruction is reading a memory and fails, the instruction address and the faulty memory address will be different. If it fails to execute an instruction as the memory containing the instruction is not accessible, the CR2 and EIP addresses will be identical.

Quiz 3 (Nov 11)

Regarding the permission, if a page is pointing to the frame with fake values (e.g., FF FF FF FF ... or 00 00 00 00 ...), it can be read/exec or read. And “read” is more desirable.

This is because, when we set up the page table to give fake values, any programs reading it would be fine. However, if the current program executes it, we need to stop the execution and replace the page with the correct instruction.
Giving the “read” permission to the page, it will cause a page fault if it tries to execute.
On that page fault, we can replace it with a correct frame containing the correct code (with “read/exec” permission).
After it executes an instruction, we need to vacate the page again. Otherwise, other programs can read the correct instruction without any errors.