Fill out the bottom of this page with your computing ID.
Write your computing ID at the top of each page in case pages get separated.

X86-64 calling convention reminder:

- first argument: `%rdi`
- second argument: `%rsi`
- return value: `%rax`
- return address: on stack

X86-64 registers reminder:

- `%rax` (64-bit), `%eax` (lower 32 bits), `%ax` (lower 16 bits), `%al` (lower 8 bits)
- (and similar for `%rbx`, `%rcx`, `%rdx`)
- `%rsi` (64-bit), `%esi` (lower 32 bits), `%si` (lower 16 bits), `%sil` (lower 8 bits)
- (and similar for `%rbp`, `%rsp`, `%rdi`)
- `%r8` (64-bit), `%r8d` (lower 32 bits), `%r8w` (lower 16 bits), `%r8b` (lower 8 bits)
- (and similar for `%r9` through `%r15`)

AT&T syntax reminder:

- `0x1234(%r9,%r10,4)` = memory at `0x1234` + `%r9` + `%r10` × 4
- `$0x12345678` = constant
- `0x12345678` = memory at `0x12345678`
- source, destination

1. Consider the following C function:

```
void foo(char *array) {
    char buffer[64]; int i;
    for (i = 0; array[i] != '\0'; ++i) {
        buffer[i] = array[i] ^ array[i-1];
    }
    for (i = 0; array[i] != '\0'; ++i) {
        array[i] = buffer[i];
    }
}
```

With one compiler and set of optimization flags it compiles to the following assembly (shown using `objdump` output from a generated executable):

```
00000000004004e6 <foo>:
  4004e6:       48 83 ec 40             sub    $0x40,%rsp
  4004ea:       31 c0                   xor    %eax,%eax
  4004ec:       8a 14 07                mov    (%rdi,%rax,1),%dl
  4004ef:       84 d2                   test   %dl,%dl
  4004f1:       74 0c                   je     4004ff <foo+0x19>
  4004f3:       32 54 07 ff             xor    -0x1(%rdi,%rax,1),%dl
  4004f7:       88 14 04                mov    %dl,(%rsp,%rax,1)
  4004fa:       48 ff c0                inc    %rax
  4004fd:       eb ed                   jmp    4004ec <foo+0x6>
  4004ff:       31 c0                   xor    %eax,%eax
  400501:       80 3c 07 00             cmpb   $0x0,(%rdi,%rax,1)
  400505:       74 0b                   je     400512 <foo+0x2c>
  400507:       8a 14 04                mov    (%rsp,%rax,1),%dl
  40050a:       88 14 07                mov    %dl,(%rdi,%rax,1)
  40050d:       48 ff c0                inc    %rax
  400510:       eb ef                   jmp    400501 <foo+0x1b>
  400512:       48 83 c4 40             add    $0x40,%rsp
  400516:       c3                      retq
```

(a) What instruction implements the read from `array[i-1]`?
   ◯ `mov (%rdi,%rax,1),%dl` √ **`xor -0x1(%rdi,%rax,1),%dl`**
   ◯ `mov (%rsp,%rax,1),%dl` ◯ none of these

(b) In what location is `i` stored during the first loop?
   ◯ `%edx/%rdx` ◯ `%esi/%rdi` √ **`%eax/%rax`** ◯ `%esp/%rsp`

(c) In what location is `array` stored during the first loop?
   ◯ `%rdx` √ **`%rdi`** ◯ `%rax` ◯ `%rcx`

(d) In what location is the return address stored during the first loop?
   ◯ `(%rsp)` ◯ `%rip` √ **`0x40(%rsp)`** ◯ `0x48(%rsp)` ◯ none of these

(e) In what location is `buffer[0]` stored during the first loop?
   √ **`(%rsp)`** ◯ `%rip` ◯ `0x40(%rsp)` ◯ `0x48(%rsp)` ◯ none of these

(f) The `jmp 0x400501` at address `0x400510` is encoded with the one-byte opcode `0xEB` followed by the one-byte signed offset `0xEF` (two's complement: -17). What would be the encoding of that jump if it jumped to address `0x4004FF` instead?
   ◯ `eb ff` ◯ `eb e1` √ `eb ed` ◯ `eb f1` ◯ not possible in two bytes

2. For each of the following malware detection techniques that might be used by antivirus software, identify which of the listed malware countermeasures may be effective against them. **Select all that apply.** *Grading: 5 points base; -1 for disagreeing answer; minimum zero*

   (a) (5 points) Looking for fixed strings that indicate virus code in executable files on disk.

   - √ **metamorphic malware code**
   - √ **polymorphic malware code**
   - ◯ checking whether loaded machine code has changed in memory
   - ◯ using cavities instead of appending virus code to a file
   - ◯ tunneling via examining antivirus library or OS "hooks"
   - √ **stealth via hooking OS filesystem functions**
   - ◯ randomly deciding whether or not to run the malware code

   (b) (5 points) Looking for fixed strings that indicate virus code in program memory after an executable has run for some time.

   - √ **metamorphic malware code**
   - ◯ polymorphic malware code
   - ◯ checking whether loaded machine code has changed in memory
   - ◯ using cavities instead of appending virus code to a file
   - ◯ tunneling via examining antivirus library or OS "hooks"
   - ◯ stealth via hooking OS filesystem functions
   - √ **randomly deciding whether or not to run the malware code**

   (c) (5 points) Detecting **attempts** to modify a "sacrificial goat" executable file. *no deduction for also choosing "randomly deciding..."*

   - ◯ metamorphic malware code
   - ◯ polymorphic malware code
   - ◯ checking whether loaded machine code has changed in memory
   - ◯ using cavities instead of appending virus code to a file
   - √ **tunneling via examining antivirus library or OS "hooks"**
   - ◯ stealth via hooking OS filesystem functions
   - ◯ randomly deciding whether or not to run the malware code

   (d) (5 points) Periodically scanning for **changes** in the contents of a "sacrificial goat" executable file. *no deduction for also choosing "randomly deciding..."*

   - ◯ metamorphic malware code
   - ◯ polymorphic malware code
   - ◯ checking whether loaded machine code has changed in memory
   - ◯ using cavities instead of appending virus code to a file
   - ◯ tunneling via examining antivirus library or OS "hooks"
   - √ **stealth via hooking OS filesystem functions**
   - ◯ randomly deciding whether or not to run the malware code

(e) (5 points) Periodically scanning for **changes** in executable file metadata.

- ◯ metamorphic malware code
- ◯ polymorphic malware code
- ◯ checking whether loaded machine code has changed in memory
- √ **using cavities instead of appending virus code to a file**
- ◯ tunneling via examining antivirus library or OS "hooks"
- √ **stealth via hooking OS filesystem functions**
- ◯ randomly deciding whether or not to run the malware code

(f) (5 points) Before any executable is run, checking for the appearance of API function names (like `GetFileAttributesA`) in an executable file's code instead of in it's linking information? accepted not selecting stealth (assumption: can't setup hooks yet)

- √ **metamorphic malware code**
- √ **polymorphic malware code**
- ◯ checking whether loaded machine code has changed in memory
- ◯ using cavities instead of appending virus code to a file
- ◯ tunneling via examining antivirus library or OS "hooks"
- √ **stealth via hooking OS filesystem functions**
- ◯ randomly deciding whether or not to run the malware code

(g) (5 points) Before any executable is run, checking whether its entry-point is in the last segment of an executable (on systems where this is not typical). accepted not selecting stealth (assumption: can't setup hooks yet)

- ◯ metamorphic malware code
- ◯ polymorphic malware code
- ◯ checking whether loaded machine code has changed in memory
- √ **using cavities instead of appending virus code to a file**
- ◯ tunneling via examining antivirus library or OS "hooks"
- √ **stealth via hooking OS filesystem functions**
- ◯ randomly deciding whether or not to run the malware code

3. (6 points) Which of the following statements about a program running in a system virtual machine executing a system call are true? Assume the virtual machine is implemented by privileged operations executed from user mode triggering exceptions (a "native" or "trap-and-emulate" implementation), **not** with emulation or binary translation. **Select all that apply.**

- √ **Control reaches the host OS or virtual machine monitor before the system call implementation in the guest OS is run.**
- ◯ The implementation of the system call in the **guest OS** is executed in kernel mode.
- ◯ The system call in the program must be replaced by a normal function call.

4. (6 points) Which of the following are techniques to detect or break virtual machines like those that might be used by antivirus software? **Select all that apply.**

   ◯ metamorphic malware code

   √ **using exotic system calls**

   ◯ checking whether loaded machine code has changed in memory

   √ **timing operations like system calls**

   ◯ attempting to use a pseudo-random number generator

   √ **checking the names of devices on the system**

   ◯ using the stack pointer for something other than a stack

   ◯ corrupting information in executables that is not used at runtime

   ◯ using cavities instead of appending virus code to a file

5. (6 points) Which of the following are techniques to detect or break debuggers? **Select all that apply.**

   ◯ metamorphic malware code no deduction for selecting (breakpoints in to-be-decrypted code won't work)

   ◯ using exotic system calls

   √ **checking whether loaded machine code has changed in memory**

   ◯ timing operations like system calls no deduction for selecting (could detect single-stepping)

   ◯ attempting to use a pseudo-random number generator

   ◯ checking the names of devices on the system

   √ **using the stack pointer for something other than a stack**

   √ **corrupting information in executables that is not used at runtime**

   ◯ using cavities instead of appending virus code to a file

6. (10 points) Some malware includes code that transforms machine code programmatically to new machine code. Which of the following are true about such transformations in **metamorphic** malware? **Select all that apply.**

   ◯ The transformation code must handle all instructions that exist in the instruction set architecture for the new machine code to operate properly.

   √ **If the malware changes the lengths of the machine code, then the analysis (meant transformation) code needs to change relative jumps.**

   ◯ The transformation code needs to be written without using absolute addresses.

   √ **The transformation code will be run on itself.**

   ◯ The transformed code will include do-nothing instructions that antivirus software can use to detect this technique.

7. Consider the following excerpt from running `objdump -d` on an executable:

```
0000000000400581 <foo>:
  400581:       55                      push   %rbp
  400582:       53                      push   %rbx
  400583:       48 89 f5                mov    %rsi,%rbp
  400586:       48 89 fb                mov    %rdi,%rbx
  400589:       48 89 fe                mov    %rdi,%rsi
  40058c:       48 81 ec 08 04 00 00    sub    $0x408,%rsp
  400593:       48 89 e7                mov    %rsp,%rdi
  400596:       e8 95 fe ff ff          callq  400430 <strcpy@plt>
  40059b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  4005a0:       48 89 df                mov    %rbx,%rdi
  4005a3:       e8 98 fe ff ff          callq  400440 <strlen@plt>
  4005a8:       48 8d 3c 04             lea    (%rsp,%rax,1),%rdi
  4005ac:       48 89 ee                mov    %rbp,%rsi
  4005af:       e8 7c fe ff ff          callq  400430 <strcpy@plt>
  4005b4:       48 89 e7                mov    %rsp,%rdi
  4005b7:       e8 c4 ff ff ff          callq  400580 <do_something_with>
  4005bc:       48 81 c4 08 04 00 00    add    $0x408,%rsp
  4005c3:       5b                      pop    %rbx
  4005c4:       5d                      pop    %rbp
  4005c5:       c3                      retq
  4005c6:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  4005cd:       00 00 00
```

Suppose we wanted to insert a jump to some virus code in the middle of this function. Assume that:

- we can encode the jump using 5 bytes;
- our virus code does not modify any registers;
- besides the virus code and the jump itself, we don't add any other code to the program or rely on code not implied by the above disassembly being present

(a) (10 points) Suppose the virus code ends by **returning like a normal function**. Where can we insert this jump so it will be reached but will not disrupt the program's behavior? **Select all that apply.** special case: 8/10 for interpretation consistent with jump being call

- ◯ in place of the `subq` at `0x40058c`
- ◯ in place of the `call` at `0x400596`
- ◯ in place of the `nop` at `0x40059b`
- √ **in place of the `ret` at `0x4005c5` virus returns to foo's caller**
- ◯ in place of the `nop` at `0x4005c6`

(b) (10 points) Suppose we end the virus code with a jump to a **fixed address of our choice** instead of by returning. Where can we insert ~~this~~ jump *to the virus code (correction during exam)* so it will be reached but will not disrupt the program's behavior?

- ◯ in place of the `subq` at `0x40058c`
- ◯ in place of the `call` at `0x400596`if inserted `call`, jump to 0x400430
- √ **in place of the `nop` at `0x40059b`** ***end of virus jump to `0x4005a0`***
- ◯ in place of the `ret` at `0x4005c5`
- ◯ in place of the `nop` at `0x4005c6`