

1. True/False
 - a. By definition, a threat model describes all possible attacks against a system.
(t) True
(f) False
 - b. Return-oriented Programming is a technique whereby the attacker overwrites the return address to point to code the attacker has previously injected onto the stack.
(t) True
(f) False
 - c. Use-after-free attacks can be used to read confidential data, but cannot be used to manipulate control flow.
(t) True
(f) False
 - d. Control Flow Integrity (CFI) is a technique for dynamically checking if indirect control transfers during program execution are consistent with those described by the program source code.
(t) True
(f) False
2. In Aleph One's "Smashing the Stack for Fun and Profit", the author writes a candidate shellcode but then transforms it. We discussed this in class. What changes did he make and why? **(select all that apply)**
 - (a) To add NOP instructions (a so-called "NOP sled") to make it easier to jump to shellcode**
 - (b) To remove any existing NOP instructions that would interfere with NOP sleds
 - (c) To remove any zero bytes that might terminate a string function**
 - (d) Using pointer subterfuge to skip the stack canary value and avoid detection
3. Control Flow Integrity **(select all that apply)**
 - (a) Prevents the overflow of heap buffers
 - (b) Prevents the overflow of stack buffers
 - (c) Requires that the attacker cannot modify the code segment**
 - (d) Checks the target of each indirect jump/call to ensure it is reachable from that call site**
 - (e) Requires the heap pointers only point to "live" memory (i.e., currently allocated)
 - (f) Is only applicable for programs written in type-safe languages
 - (g) Validates that the target of each direct jump/call to ensure it is reachable from that call site

4. The operating system kernel protects itself from user programs using which of the following techniques? **(fill in all that apply)**
- (a) **Only allowing user programs to execute in the processor's unprivileged execution mode**
 - (b) **Carefully validating the arguments passed in system calls from user programs**
 - (c) Encrypting the kernel heap to protect it from being read or altered by user programs
 - (d) **Protecting the memory holding kernel data structures from being read/written by user programs**
 - (e) Randomizing the order in which user programs execute
 - (f) Flushing the processor cache after each system call