

### Part 1:

1) The value is at the 9th argument slot. %9\$ld can leak the value.

2) Even if there are no secrets on the stack: the vulnerability still provides read/write access to program memory. That is enough for an attacker to create an attack.

For example, a format-string bug can let an attacker write memory, not just read it. Say a program does printf(user\_input).

It also does if (auth == 10) {grant\_access()}. An attacker can place the auth address in their input so it goes on the stack where printf will treat it as an argument. They can use certain specifiers to write the number of bytes printed so far into that address – by printing the right number (or using multiple small writes) they can set auth = 10 and bypass the check.

### Part 2:

1) It is 72 bytes away. We can override it with the address of print\_flag to actually print the flag.

2) Yes. The libc base can be leaked to find the address of /bin/bash or /bin/sh. A chain of gadgets can be used.

If the system is not available, syscalls can still be found in libc or in the vDSO. Then ROP can be used for the attack.

### Part 3:

1)

```
@Itisalex2 → /workspaces/25Fall-UCLA-ECE-117-CS138/assignment-2/3-killing-the-canary (main) $ checksec killing-the-canary
[*] '/workspaces/25Fall-UCLA-ECE-117-CS138/assignment-2/3-killing-the-canary/killing-the-canary'
    Arch:      amd64-64-little
    RELRO:     Partial RELRO
    Stack:     Canary found
    NX:        NX enabled
    PIE:       No PIE (0x400000)
    SHSTK:    Enabled
    IBT:       Enabled
    Stripped: No
    Debuginfo: Yes
```

RELRO: the binary's GOT/PLT are only partially protected. It is still writable until dynamic relocations finish. An attacker may be able to overwrite some GOT entries

Stack: a canary is found to mitigate buffer overflows. it prevents simple return-address overwrites unless the canary value is known or bypassed.

NX: You cannot directly execute injected shellcode on the stack; attacks need to use ROP.

PIE: the binary is loaded with a fixed base address, so code addresses are predictable and exploitable

SHSTK: enabled so return address are stored on an isolated shadow stack, making ROP harder

IBT: restricts where indirect branches can jump

Stripped (No) and Debuginfo (Yes): reversing and finding function addresses like `print_flag` is much easier because symbols and debug info aren't removed.

The binary is still vulnerable to buffer-flow attacks.

2) Saved return addresses, saved frame pointers, function arguments etc

Part 4:

1) Attackers can reuse gadgets in that already exist in mapped code (e.g. `libc`). By chaining gadgets that end in `ret`, they can emulate arbitrary computation and call sequences without injecting code

2) There should be a new arrow from `game` to `print_flag`. The diagram is a call graph / control-flow graph.

CFI enforces that indirect transfers only go to a small, precomputed set of legitimate targets. `Print flag` is not legitimate so the CFI is violated.

3) Pure symbolic execution suffers path-explosion i.e. constraints that cannot be solved. Concolic executes concrete inputs and symbolically records constraints along that run to produce solvable path constraints.

4) Yes. Given enough time, someone can brute force the attack. The success rate can be improved by using a large NOP-sled, so many randomized return addresses land in the sled