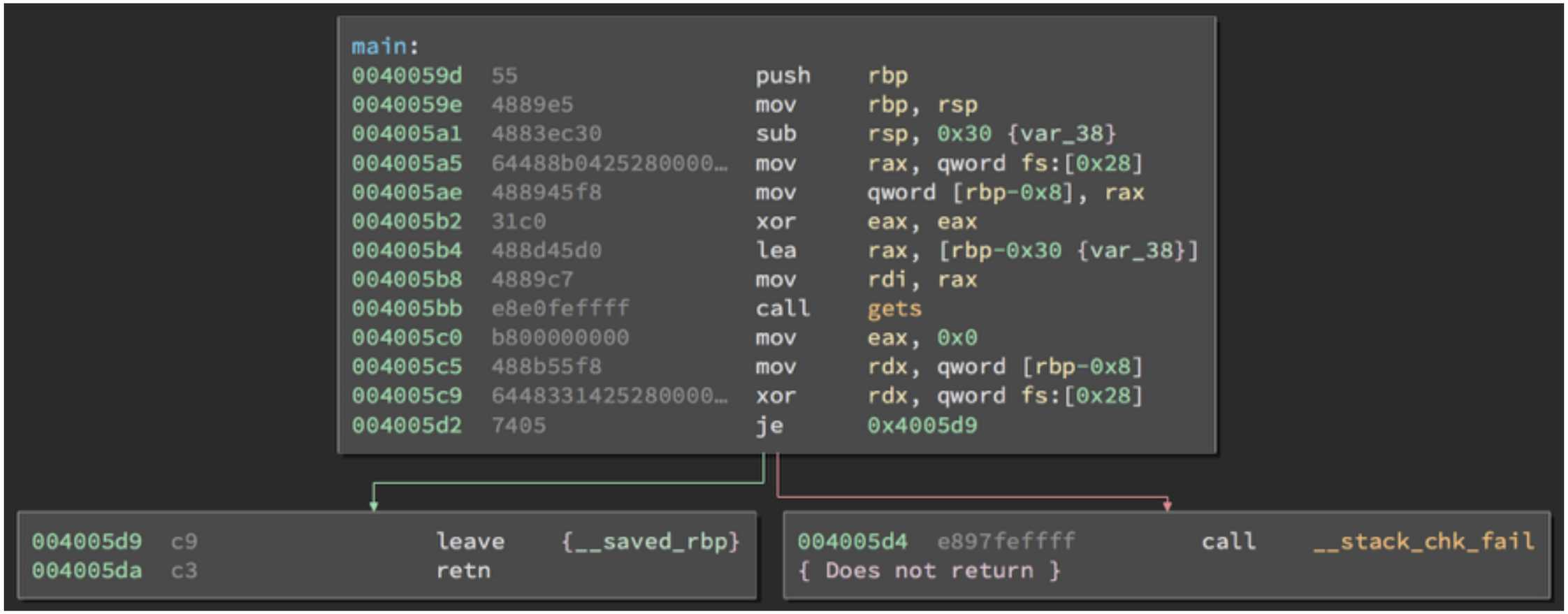
Stack Canaries

CS-UY 3943-G / CS-GY 9223-H

Stack Canaries



Stack Canaries

- We put a secret value on the stack
  - This value changes every time we start the program

```
main:
0040059d  55                    push    rbp
0040059e  4889e5                mov     rbp, rsp
004005a1  4883ec30              sub     rsp, 0x30 {var_38}
004005a5  64488b0425280000…     mov     rax, qword fs:[0x28]
004005ae  488945f8              mov     qword [rbp-0x8], rax
```

Stack Canaries

- Before the program exits, it checks if the stack canary has changed, and exits immediately if it does

Stack Canaries

- This is a quick and easy mitigation to the stack smashing we've been doing!
- We can't really guess a 64-bit value out of thin air…
- We can still work around this

Stack Canaries

- A new item gets added to every stack frame
- This 8-byte value is added right after the saved rbp

Stack Canary Leaking

- If we can read the data in the stack canary, we can send it back to the program later
    - Remember, it's the same throughout the execution of the program
- Linux tries to make this slightly tricky:
    - The first byte of the stack canary is a NULL, so string functions will stop when they hit it
    - You can partially overwrite and then put the NULL back, or you can find a way to leak bytes at an arbitrary stack offset

Stack Canary Leaking

- Situations we might be able to leak a canary:
  - User-controlled format string
  - User-controlled length of an output
    - “Hey, can you send me 1000000 bytes? thx!”

Stack Canary Bruting

- Remember, the canary is determined when the program starts for the first time

- If the program forks, it keeps the *same* stack cookie in the child process

    - That means that if our input that can overwrite the canary is sent to the child, we can use whether it crashes as an oracle and brute-force 1 byte at a time!

Accept-and-Fork Servers

- So far, all the programs we've exploited have used stdin/stdout to talk to you
    - In the real world, most talk over a network socket directly
- A common way this is done is with an accept-and-fork

Accept-and-Fork Servers

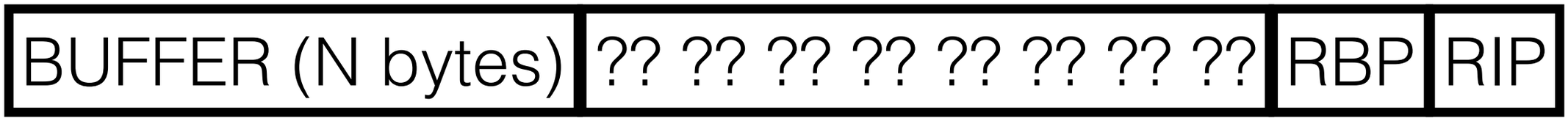
- The server waits for a connection to come in, and then forks off a child process to handle the connection
    - The forked-off child process gets a copy of the memory of the parent process
    - Including the stack canary!
    - This means we can use whether a child process crashes or not as an *oracle* for our guess of a byte of the canary

Canary Bruting

- This is actually a lot like the Blind SQLi from very early in the semester!

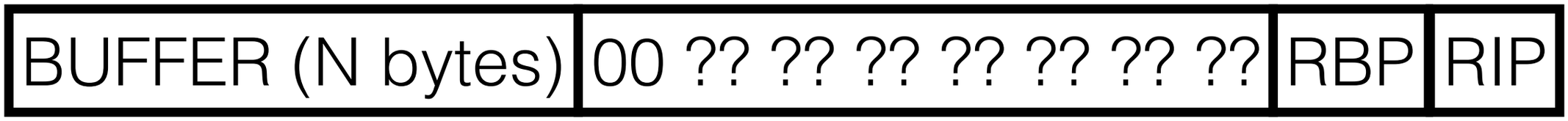
- This also only works on functions that don't append a NULL byte to our input
  - We need exact control of the last byte!
  - Functions like *read*, and *recv* have this property

Canary Bruting



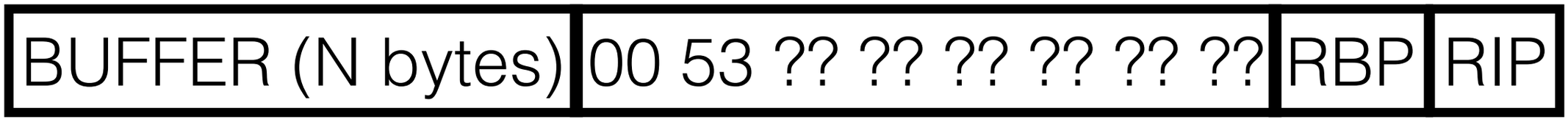
00 -> No crash!

Canary Bruting



00 -> crash :(
01 -> crash :(
...
53 -> No crash!

Canary Bruting



00 -> crash :(
01 -> crash :(
...
FE -> No crash!

5 Bytes Later…

Canary Bruting

BUFFER (N bytes)	00 53 FE D0 A5 BC 74 11	RBP	RIP

^ Found the cookie!

And now we can do our normal buffer overflow exploitation!