CS431

Computer and Network Security

Buffer Overflows - Mitigations

Abhishek Bichhawat

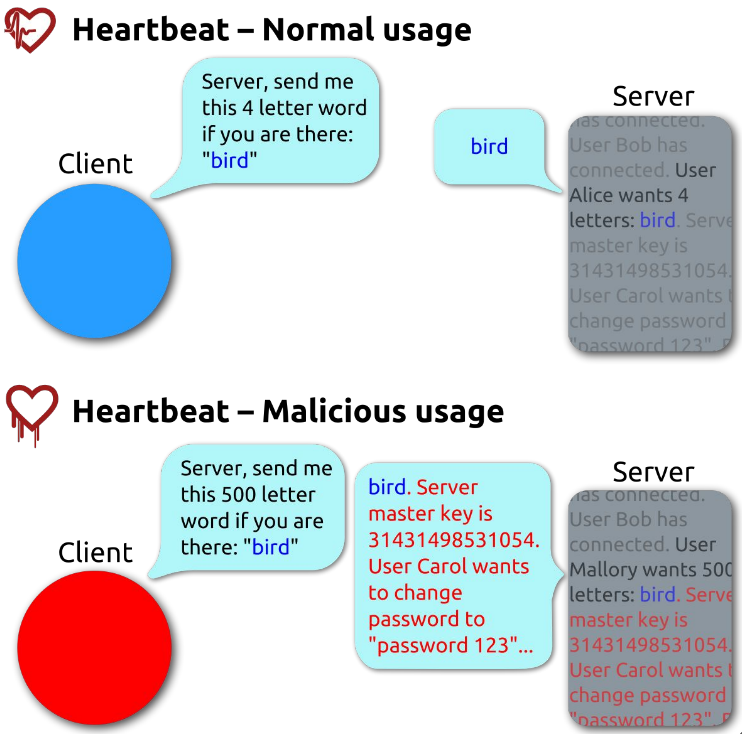
01/02/2024

Heartbleed

- **Heartbeats** are used to check if the server and clients are alive
  - echo the message as a keep-alive
- OpenSSL used

```
memcpy(bp, pl, payload);
```

- Can send more data than was requested
- Impacted many systems
- Check this



Why do such vulnerabilities exist?

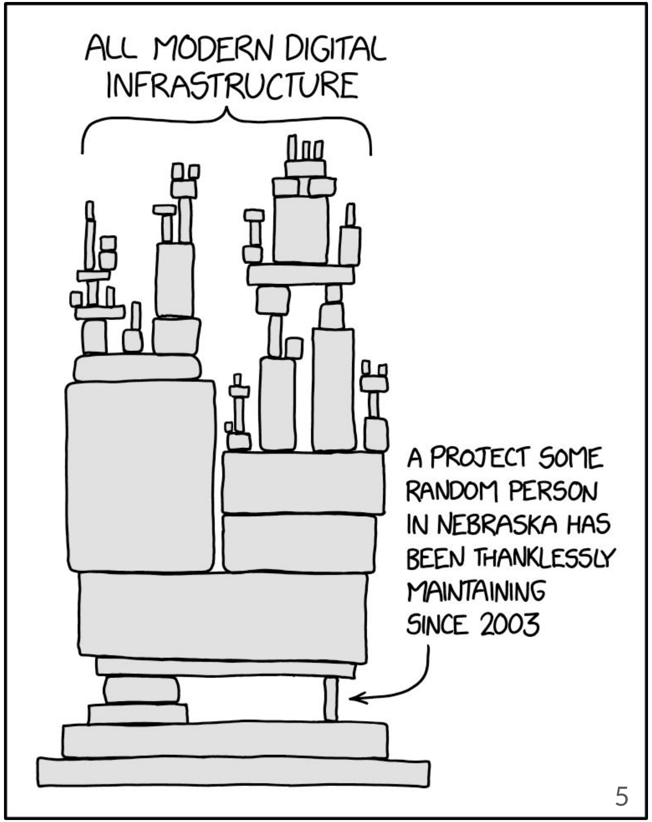
- Programming languages aren’t designed with security in mind
- Programmers aren’t security aware
- Programs are not implemented in a secure-by-design fashion
- Programming errors that cause bugs

Defenses

- DO NOT use programming languages that aren't designed with security in mind
  - Use memory-safe languages
  - Prevent undefined memory accesses and vulnerabilities
  - Performance might be worse due to garbage collection
    - This may, however, be insignificant in most applications
    - Today's languages have comparable performances
- Programmers should AVOID risky constructs
  - Use APIs that specify bounds instead of the simpler APIs
  - DO NOT trust the inputs - add all checks
  - Write memory-safe code
  - Reason about the code (check e.g., F*)

Defenses

- Implement programs in a secure-by-design fashion
  - Include security as a feature of the program
  - Do not depend on untrusted libraries
- Test the programs extensively for programming errors
  - Design tools for analyzing code
  - Bug-finding tools
  - Penetration testing
  - Fuzz testing
- Run in a virtual environment or sandbox to contain the effect



Defenses

- Non-executable buffers
  - Prevent injected code from executing
  - Can affect optimizations
- Array bounds checking
  - Compiler warnings
  - Run-time memory access checks (e.g., Purify)
  - Static analysis (e.g., model checking tools)

Defenses - StackGuard

- Mitigate ways to exploit a vulnerability
    - Stop executing before the exploit “exploits”
    - StackGuard (for stack smashing)
        - Also known as stack canary
        - Place a canary immediately after the return address
            - The canary is a value that we do not care about
        - If the canary does not change, there was no overwrite
        - If the canary dies (changes), there was an overwrite

StackGuard Attacks

- Fixed canary
  - Fixed value for canary
  - Overwrite with the same value or bypass the canary
- Random canary
  - Format string vulnerability
  - Brute-force (works with poorly generated random numbers)
- Terminator canary
  - Use NULL or \r\n or -1 to indicate end of line
  - String functions terminate here (so cannot replace canary value)

StackShield

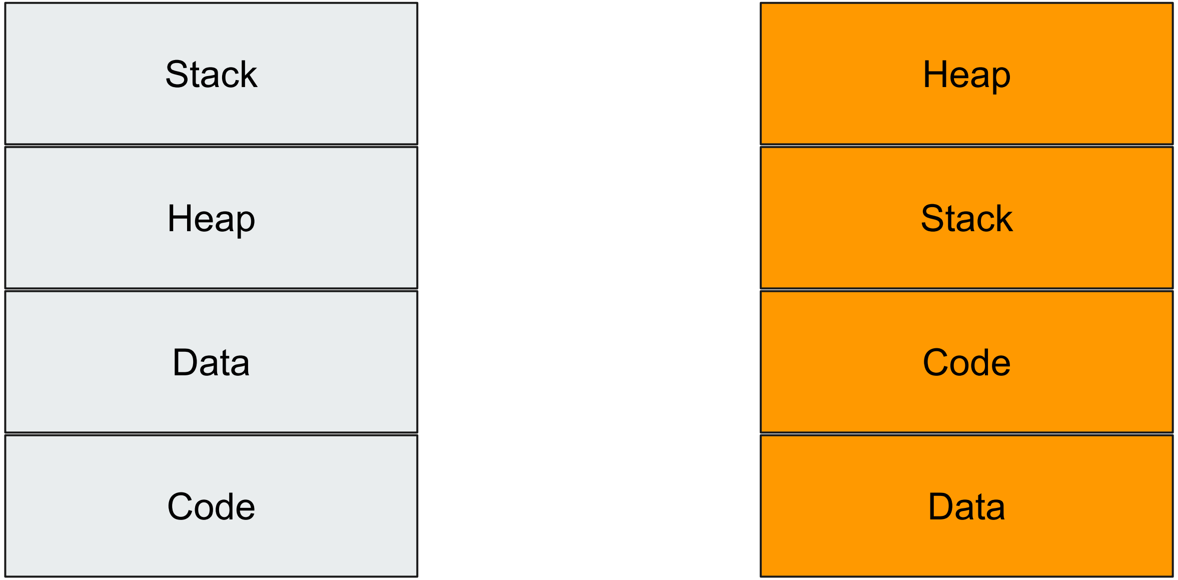
- Duplicate the stack
- Use return address from the duplicate stack
- Three ways
  - Global return stack
    - Separate stack for return addresses (256 entries)
  - Return range check
    - Global variable that stores the base of the stack
    - Compare before returning; can detect attacks
  - Function pointer protection
    - Function pointers should only point to the text segment
- https://www.angelfire.com/sk/stackshield/

Pointer Authentication Code

- Actual addresses use lesser than 64 bits
- Use the remaining bits to store a pointer authentication code
- Check the code before accessing the address (pointer) value
  - Can do it for different pointers
  - Abort, if invalid
- Uses a secret value (key) to generate the PAC (just like MACs)
- Stored in the CPU
- Different secrets may be used for different types of pointers

Address Space Layout Randomization

- Randomly arrange the various sections of a process' address space in different parts of the memory



Address Space Layout Randomization

- Relative addresses (like the address of Return Instruction Pointer with respect to Stack Frame Pointer) are fixed

```
void vulnerable(char *dest) {
    // Format string vulnerability
    printf(dest);
}

int main(void) {
    int secret = 42;
    char buf[20];
    fgets(buf, 20, stdin);
    vulnerable(buf);
}
```

Input: '%x'

Output: 0x0408

What is the address of `secret`?