Memory Safety

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

04/07/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Why is this an issue?

- C and C++ are low-level languages, which means they allow access and manipulation to memory directly as the developer
- This is a double-edged sword
  - Allows you to do some efficient and fast operations
  - But also requires you, as the developer, to be very cautious

Some stats

- Google reported that 70% of severe security bugs are actually memory issues [1]
- There is a recent push for memory-safe languages from the government (last several year)
  - Safecpp.org

Example 1: raw pointers

- What is a “dangling pointer”?
    - int* ptr = new int(20);
    - delete ptr;
    - ptr = nullptr;
    - std::cout << *ptr; // dereferencing the ptr leads to undefined behaviour

Example 1: Consequences

- Memory corruption
  - If the memory is reallocated elsewhere, then it can lead to memory corruption or overwrite the data
- Information leaks
  - Can lead to access to sensitive data if it is put there, or to privilege escalation (if that memory is used in security checks)
- Malicious code execution
  - If the pointer is used to make a virtual function call, then a different address (pointing to exploit code) can be called due to vtable pointer being overwritten

Use after free

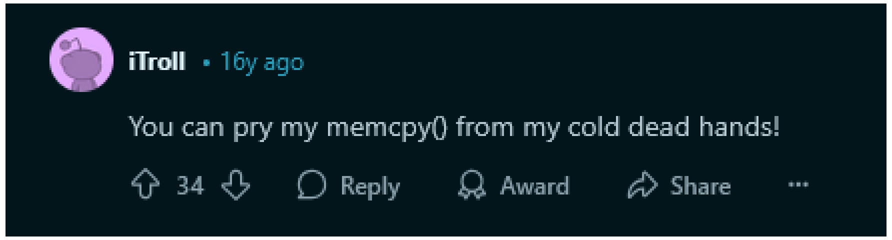
- When a dangling pointer is used after it has been freed without allocating a chunk of memory, it is known as a “use after free” vulnerability
- CVE-2014-1776 [2]: In Microsoft Internet Explorer version 6-11, this vulnerability allowed attackers to execute arbitrary code or cause DoS attack using memory corruption
- Exploited in the wild!

Example 1: fixes

- Use smart pointers
- Use audited pointers
- Be sure to always set the ptr back to null

Example 2: Copying memory

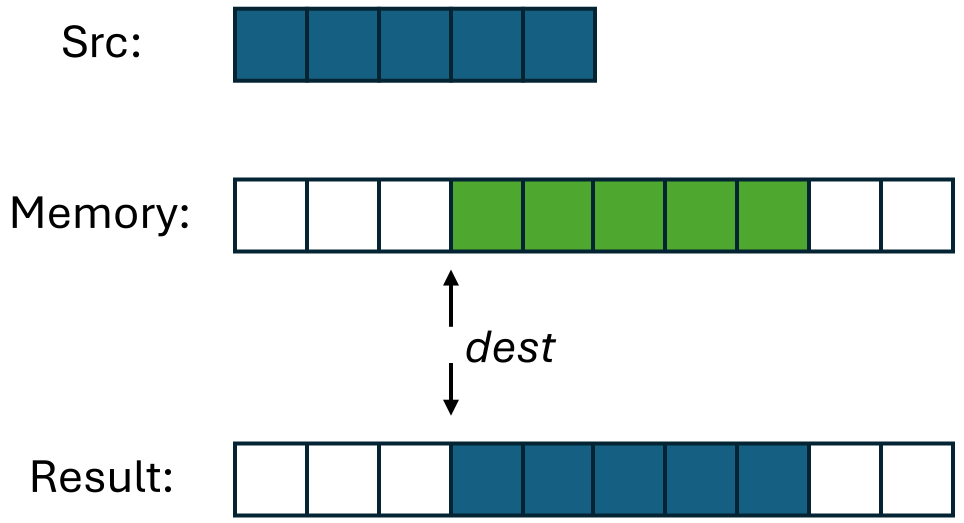
- The C native functions memcpy() and strcpy() are some of the most dangerous functions
- Yet, they are very popular in the C community
- In fact, Microsoft banned their use almost 20 years ago in 2009 in their secure development cycle


iTroll • 16y ago

You can pry my memcpy() from my cold dead hands!

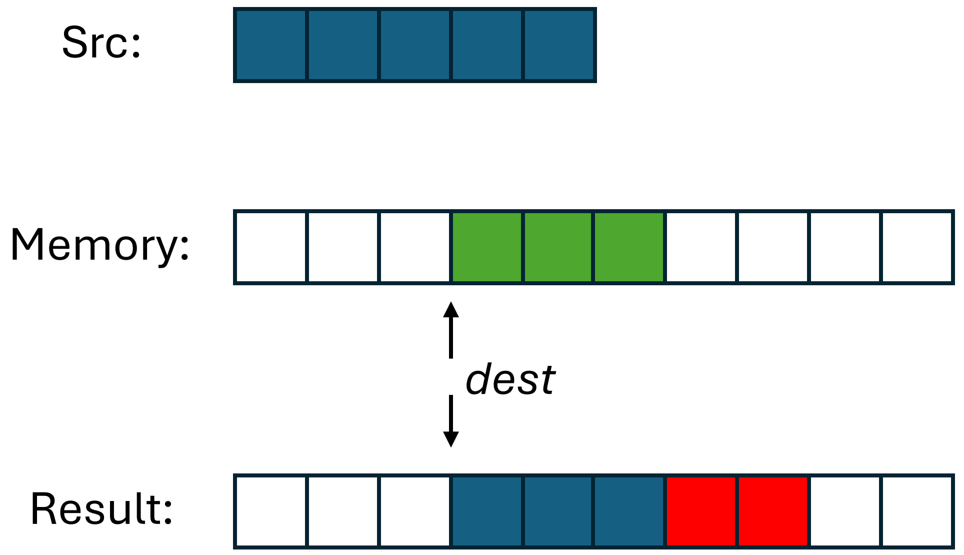
Memcpy() and strcpy()

- As the name suggest, it copies a buffer over from a *src* to a *dest*
- Assume *dest* is a buffer of char*[5]
- Ex: memcpy(*dest, src, 5)*



Buffer Overflow Attack

- If *dest* is a buffer of char*[3]
- Ex: memcpy(*dest, src, 5)*



- This becomes a really big problem if red contains code, or if sensitive information
- Imagine if red contains a bit that is True if user is authenticated

Underlying issue

- Memcpy() and strcpy() does not do bounds checking
    - It is assumed that the developer will always use it correctly
    - A contract you sign when you use C or C++

If you can write perfect code…

- Let’s assume that you have gained the ability to write perfectly efficient and safe code
- Is your code always safe?

- What kind of packages do you import?

Log4J incident

- A widely used logging tool for Java
- A bug allowed for attackers to do Remote Code Execution
- A zero-day attack was first reported in November 2021 (CVE-2021-44228)[3]
- This vulnerability has been in the tool since 2013
- An estimated 93% of enterprise cloud environment was affected [4]
- Some services include: AWS, Cloudflare, iCloud, Minecraft servers, Steam, Tencent, etc.

Summary

- Low-level languages allow developers to directly manipulate memory
    - This allows for flexibility and efficiency, but also puts the burden on the developer to write safe and good code
- Vulnerabilities are not limited to code that you write, but code that you use and import as well
- There is no silver bullet to solve security, but require diligence from us (the devs) to think about types of attacks that can affect our code

References

- [1] https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html
- [2] https://nvd.nist.gov/vuln/detail/CVE-2014-1776
- [3] https://nvd.nist.gov/vuln/detail/cve-2021-44228
- [4] https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell