# Contents

# 1 Memory Bugs

Memory-related bugs are, frankly, unpleasant–especially when they're well hidden. When they do rear their ugly heads, they can be deceptively difficult to reproduce and hunt down. Memory corruption is a critical issue in programming, particularly in C and C++, which provide direct access to memory. It occurs when a program accesses or modifies memory in unintended ways, often leading to data corruption, security vulnerabilities, or crashes. This section covers some of the most common types of memory-related bugs in C++, how they arise, and how to avoid them.

## 1.1 Buffer Overflows

Buffer overflows occur when a program writes past the end of an array (a buffer), corrupting adjacent memory. Theyre a common source of security vulnerabilities, as attackers can exploit them to inject or execute arbitrary code.

### 1.1.1 Buffer Overflow Attack

Below is an example of a very simple password authenticator that is susceptible to a buffer overflow:

*password.cpp*

```cpp
#include <iostream>
#include <cstring>

int main()
{
    char input[8] {};
    char passwd[8] { "rbaker" };

    std::cin >> input;

    if (std::strncmp(input, passwd) == 0)
    {
        std::cout << "password accepted" << std::endl;
    }
```

```
15      else
16      {
17          std::cout << "password rejected" << std::endl;
18      }
19  }
```

This code prompts the user to enter a password, then compares it against the stored password and prints a corresponding message. Of course, storing passwords this way is inherently insecure–they're kept in plain text and left unprotected in memory. Nonetheless, for the sake of demonstration, well continue with this simplified implementation. Running the program:

Terminal

```
$ clang++ -std=c++23 password.cpp
$ ./a.out
enter password: passwd
password rejected
$ ./a.out
enter password: 123
password rejected
$ ./a.out
enter password: rbaker
password accepted
```

At first glance, everything appears to be working correctly. The application denies access when the input does not match the stored password and grants access when it does. However, there is a subtle but serious issue with this implementation:

Terminal

```
$ ./a.out
enter password: qwertyuiqwertyui
password accepted
```

This is unexpected–we entered a string that is clearly not the correct password, yet the program still granted access.