Widthness Overflow

A bad type conversion can cause widthness overflows

```
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {

    unsigned int l = 0xdeabeef;
    printf("l = 0x%u\n", l);

    unsigned short s = l;
    printf("s = 0x%u\n", s);

    unsigned char c = l;
    printf("c = 0x%u\n", c);

}
```

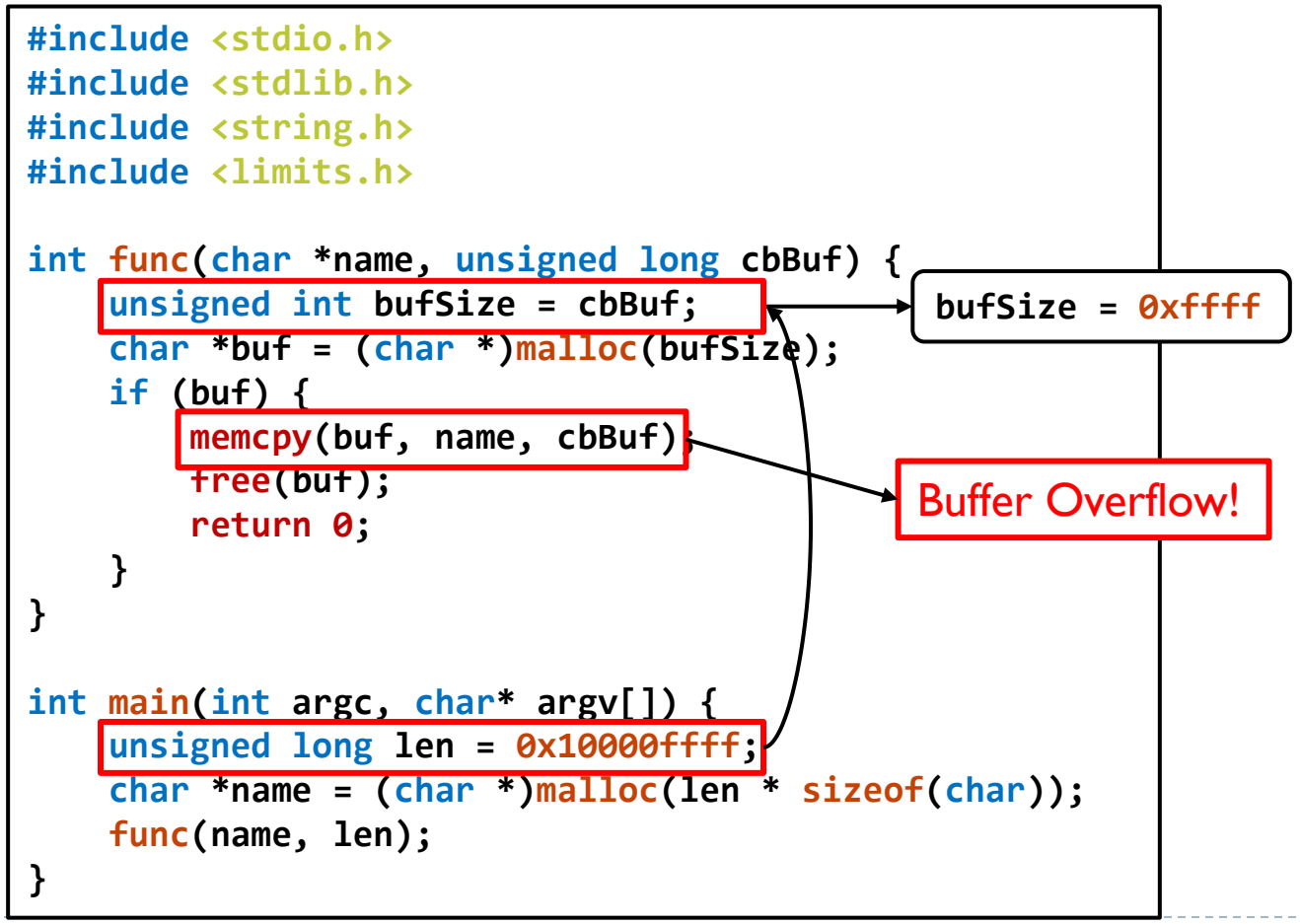
➡ 0xdeadbeef

➡ 0xbeef

➡ 0xef

Example 2: Truncation Errors

Incorrect type conversion could lead to integer overflows, and then buffer overflow.

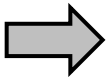


How to Fix Integer Overflow Vulnerability

Be more careful about all the possible consequences of vulnerable operations.

Better length checking

```
if (len1 + len2 + 1 <= sizeof(buf))
```



```
if (len1 <= sizeof(buf) &&
    len2 <= sizeof(buf) &&
    (len1 + len2 + 1 <= sizeof(buf)))
```

Safe type conversion:

- Widening conversion: convert from a type of smaller size to that of larger size.

Outline

- Format String Vulnerabilities
- Integer Overflow Vulnerabilities
- **Scripting Vulnerabilities**

Scripting Vulnerabilities

Scripting languages

- Construct commands (scripts) from predefined code fragments and user input at runtime
- Script is then passed to another software component where it is executed.
- It is viewed as a domain-specific language for a particular environment.
- It is referred to as very high-level programming languages
- Example:
  - Bash, PowerShell, Perl, PHP, Python, Tcl, Safe-Tcl, JavaScript

Vulnerabilities

- An attacker can hide additional commands in the user input.
- The system will execute the malicious command without any awareness

Example 1: Command Injection

Consider a server running the following command

- `system`: takes a string as input, spawns shell, and executes the string as command in the shell.

```
void display_file(char* filename) {
  char cmd[512];
  snprintf(cmd, sizeof(cmd), "cat %s", filename);
  system(cmd);
}
```

Normal case:

- A client sets `filename`=hello.txt

```
cat hello.txt
```

Compromised Input:

- The attacker sets `filename` = hello.txt; rm –rf /
- The command becomes:

```
cat hello.txt; rm -rf /
```

- After displaying file, all files the script has permission to delete are deleted!

Defenses against Command Injection

Avoid shell commands

Use more secure APIs

- Python: subprocess.run()
- C: `execve()`

Input inspection

- Sanitization: escape dangerous characters
- Validate and reject malformed input.
- Whitelist: only choose from allowed values

Drop privileges

- Run processes as non-root users.