



CS107, Lecture 7

C Strings

Reading: K&R (1.9, 5.5, Appendix B3) or Essential C section 3
[Ed Discussion](#)

Recall: Buffer Overflows

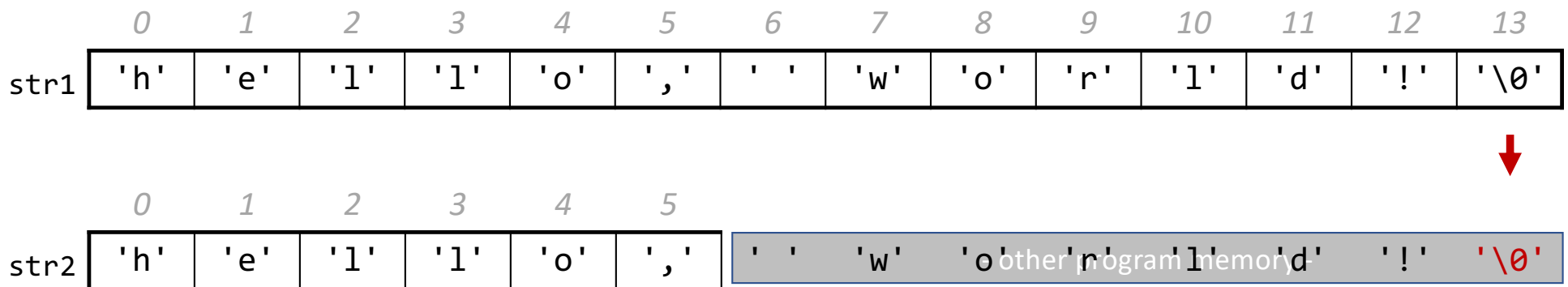
We must make sure there is enough space in the destination to hold the entire copy, *including the null-terminating character*.

```
char str2[6];           // not enough space!  
strcpy(str2, "hello, world!"); // overwrites other memory!
```

Writing past memory bounds is called a "buffer overflow". It can allow for security vulnerabilities!

Recall: Buffer Overflows

```
char str1[14];  
strcpy(str1, "hello, world!");  
char str2[6];  
strcpy(str2, str1); // not enough space - overwrites other memory!
```



Buffer Overflow Impacts

Buffer overflows can be serious, as they can lead to runtime errors and even introduce security vulnerabilities into an executable. Examples include:

- accessing memory you shouldn't be able to access
- modifying memory you shouldn't be modifying
- changing the value of a variable used later in the program
- changing the program to execute your assembly code instructions instead of its own

It's our job as programmers to find and fix buffer overflows and other bugs, not just for the functional correctness of our programs, but to protect people who use and interact with our code.

Buffer Overflow Impacts

- AOL instant messenger buffer overflow: allowed remote attackers to execute code:

<https://www.cvedetails.com/cve/CVE-2002-0362/>

<https://www.computerworld.com/article/2586310/aol-instant-messenger-vulnerable-to-hackers.html>

- Morris Worm: first internet worm to gain widespread attention; exploited buffer overflow in Unix command called "finger":

https://en.wikipedia.org/wiki/Morris_worm

How can we identify buffer overflows?

There's no single solution that works for everything. Finding and repairing overflow vulnerabilities require a combination of software development techniques:

- vigilance while programming (scrutinizing array reads and writes, pointer arithmetic)
- carefully reading documentation
- thoroughly testing to identify issues before shipping product
- thoroughly documenting assumptions in your code
- using software tools to methodically examine code for suspicious function calls

How can we identify buffer overflow?

MAN page for `gets()`:

```
char *gets(char *s);
```

Never use `gets()`. Because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. **Use `fgets()` instead.**

How can we identify buffer overflows?

- **Valgrind**: your best friend for this
- Write your own tests
- Consider writing tests *before* writing the main program

✨ cs107.stanford.edu/testing.html ✨

How Can We Fix Overflows?

Documentation & MAN Pages (Written by Others)

The `strcpy()` function copies the string pointed to by `src`, including the terminating null byte (`'\0'`), to the buffer pointed to by `dest`. The strings may not overlap, and the destination string `dest` must be large enough to receive the copy. **Beware of buffer overruns!**

If the destination string of a `strcpy()` is not large enough, then **anything might happen**. Overflowing fixed-length string buffers is a favorite cracker technique for taking complete control of the machine. Any time a program reads or copies data into a buffer, the program first needs to check that there's enough space. This may be unnecessary if you can show that overflow is impossible, but be careful: programs can get changed over time, in ways that may make the impossible possible.

Memory Safe Systems Programming

Choose your Tools & Languages Carefully

Existing code bases or requirements for a project may dictate what tools you use. Knowing C is crucial – it is and will remain widely used.

When you are choosing tools for systems programming, consider languages that can help guard against programmer error.

- Rust (Mozilla)
- Go (Google)
- [Project Verona](#) (Microsoft)

Association for Computing Machinery (ACM) Code of Ethics

ACM Code of Ethics and Professional Conduct

ACM Code of Ethics and Professional Conduct

Preamble

Computing professionals' actions change the world. To act responsibly, they should reflect upon the wider impacts of their work, consistently supporting the public good. The ACM Code of Ethics and Professional Conduct ("the Code") expresses the conscience of the profession.

The Code is designed to inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way. Additionally, the Code serves as a basis for remediation when violations occur. The Code includes principles formulated as statements of responsibility, based on the understanding that the public good is always the primary consideration. Each principle is supplemented by guidelines, which provide explanations to assist computing professionals in understanding and

On This Page

[Preamble](#)

[1. GENERAL ET](#)

[1.1 Contribute
well-being, ack
are stakeholder](#)

[1.2 Avoid harm](#)

[1.3 Be honest a](#)

[1.4 Be fair and
discriminate.](#)

ACM Code of Ethics on Security

2.9 Design and implement systems that are robustly and usably secure.

Breaches of computer security cause harm. Robust security should be a primary consideration when designing and implementing systems. Computing professionals should perform due diligence to ensure the system functions as intended, and take appropriate action to secure resources against accidental and intentional misuse, modification, and denial of service. As threats can arise and change after a system is deployed, computing professionals should integrate mitigation techniques and policies, such as monitoring, patching, and vulnerability reporting. Computing professionals should also take steps to ensure parties affected by data breaches are notified in a timely and clear manner, providing appropriate guidance and remediation.

To ensure the system achieves its intended purpose, security features should be designed to be as intuitive and easy to use as possible. Computing professionals should discourage security precautions that are too confusing, are situationally inappropriate, or otherwise inhibit legitimate use.

In cases where misuse or harm are predictable or unavoidable, the best option may be to not implement the system.

Demo: Memory Errors



memory_errors.c