Command-Line Arguments in C

Bryan Parno

As a reminder, the standard function signature for the `main` function of a C program:

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

The program's command-line options are in `argv`. Specifically, `argv` is an array of length `argc`. Each entry in that array is a pointer to a null-terminated C string (i.e., each element of `argv` is a `char*` pointer).

By convention, `argv[0]` is the name of the program that is being executed, although like any convention, this is not officially guaranteed. See this article for what happens when this convention is violated.

How do arguments get from the command line into `argv`? Your shell (e.g., `bash` or `zsh`), is itself just an ordinary program that reads from its standard input to process the command you type in. It parses the command into a program name and the subsequent arguments, possibly transforming some of them in the process (e.g., expanding a `*.pdf` argument into the actual matching filenames in the current directory). It assembles the arguments into the `argv` array, and then calls (on Linux) some flavor of the fork system call to create a copy of the current process. The new child process then calls some flavor of the exec system call to start running the program you asked for. The `argv` array is passed as an argument to the exec call. If the target program is a C program, then execution will begin with the C runtime, which will perform various initialization tasks, before eventually calling your `main` function.

At least on Linux-flavored systems, the convention is that the arguments themselves are laid out sequentially in memory. In other words, there's a chunk of memory where you will find:

```
Argument0 ; Argument1 ; ... ; ArgumentN
```

where each `Argument` is a null terminated string. Put another way, the following relation holds for all `i > 0`:

```
argv[i] = argv[i-1] + strlen(argv[i-1]) + 1 // Note the +1 accounts for the null terminator
```

Note 1. *From a security perspective, it's worth thinking about the implications of passing empty arguments (i.e., `""`) as command-line arguments. As an attacker, this may help you get certain kinds of values into the program's memory that would otherwise be difficult to arrange.*

Another aspect to keep in mind is that when writing a C program, it's not your responsibility to get command-line arguments into memory. Your shell has already done that for you. Hence, you can potentially make some useful assumptions about those arguments. For example, the size of any given argument, as well as the total size of all the arguments must be smaller than main memory (yes, technically the OS could page things out to disk, but many systems disable that these days, because the performance degradation of swapping is so high). Similarly, as long as your program is launched from a trusted process, like the shell, you can assume that all arguments are properly null-terminated. As a side note, this implies that you can't pass a command-line argument to a program if the argument has a null value in the middle.