CS 6335 – Language-Based Security

Dr. Kevin Hamlen

A Functional Correctness Proof for *memcpy*

Adem Odza, Avery Arnold, Dagmawet Zemedkun, Omar Khan, Fatema Tuj Johora

Author
François Rozet

December 16, 2025

Contents

1 Introduction

2 Technical Approach

For our proof, we took a lot of inspiration from the proof of the *memset* function. There is one key difference in the behavior of the two function: *memset* sets all modified bytes to the same value (Which is provided by the user) while *memcpy* will set bytes to the corresponding value at the provided source pointer. This similarity allowed for us to create modified versions of the *memset* definitions and apply them to our proof.

The first step in our proof was to create some definitions for certain properties of *memcpy*. We created the *filled* definition to represent the modified memory:

```
Definition filled m dest src len :=
    N.recursion m (fun i m' => m'[(B) dest + i := m (B)[src + i]]) len.
```

This definition states that a memory filled with *len* bytes means that the *len* bytes starting at *dest* and *src* have the same values.

In the invariants section, we defined a few variables:

```
Section Invariants.
    Variable mem : memory    (* initial memory state *).

    Variable dest : N
    Variable src : N
    Variable len : N
```

The first variable *mem* represents the memory state and the rest represent the parameters to the *memcpy* function. The *dest* variable holds the address to the section of memory that will be written to, while *src* holds the address to the section of memory that bytes will be copied from. The *len* variable specifies the amount of bytes to copy.

We also defined a function in the invariant section that returned the intended memory and register state of the function:

```
(* Registers state after copying k bytes *)
Definition memcpy_regs (s : store) k :=
```

```
    s V_MEM64 = filled mem dest src k /\
    s R_X0 = dest /\
    s R_X1 = src /\
    s R_X2 = (len - k).
```

This definition allows us to create an invariant at any point in the function, including during any loop iteration. The value k represents the current progress of the function. If k out of *len* bytes have been filled, then we expect the memory to match the k filled state. We also expect the R_X2 register to have k less than its initial state as the function will subtract from this value to track progress.

Next, we created the invariants so that we could begin our proof.

```
Definition memcpy_invset' (t : trace) : option Prop :=
    match t with
    | (Addr a, s) :: _ =>
        match a with

        (* Entry point *)
        | 0x100000 => Some (memcpy_regs s 0)

        (* Loop 1 (1-byte writes to word boundary) *)
        | 0x100130 => Some (exists mem k, s V_MEM64 = filled mem dest src k)

        (* post-condition: *)
        | 0x100188
        | 0x1000f8
        | 0x100088
        | 0x100064
        | 0x100048
        | 0x100030
        | 0x1000b8 => Some (exists mem, s V_MEM64 = filled mem dest src len)

        | _ => None
        end
    | _ => None
    end.
```

The entry point invariant states that the memory should be unmodified as it should match a $filled\ mem\ dest\ src\ 0$ memory state.

We place the loop invariant where we enter the loop at address $0x100130$. This loop invariant states that for every iteration of the loop, there exists a memory state *mem* and a value k where the current memory state matches $filled\ mem\ dest\ src\ k$. That is, every loop iteration should match the previous one with an extra byte added from the *src* to the *dest* pointer.

The final invariant states that the memory state should equal a fully filled memory state

(*filled mem dest src len*). All *len* bytes at the *src* addresss should have been copied to the memory at *dest*.

3 Team Organization

4 Evaluation

5 Future Work

There are many additions to the proof that could be worked on.

More properties of *memcpy* could also be proved.

Safety properties

6 Related Work

memset