Thesis

jbaramidze

June 2018

1 Introduction

What is memory safety, and why is it so important? Amorim et al. [1] tries to give a formal characterization of what it means for a programming language to be memory safe, while other papers give more informal, straightforward definition, stating that memory safety is a characteristic of a program or a programming language, which guarantees that memory access errors never occur. Memory access errors can be split in two different kinds: spacial and temporal errors. Spacial memory errors are violations caused by dereferencing pointer that is out of bounds. This can be caused by indexing array outside bounds, more generally invalid pointer arithmetic, or by dereferencing uninitialized or NULL pointers. Temporal errors, on the other hand, are violations caused by dereferencing pointer that is not valid anymore, because it was freed (dangling pointers). Freeing already deallocated memory is also a temporal violation. The reason why memory safety is so important is that it's violations lead to undefined behaviour, that can often be exploited. To illustrate how those vulnerabilities can be exploied, we give following example:

```
struct my_struct
{
  char buffer[4];
  int is_admin;
};

void func1()
{
  struct my_struct s;
  s.is_admin = check_privileges();
  scanf("%s", s.buffer);
}
```

In this example we have a spatial memory vulnerability, or more concretely a buffer overflow vulnerability. In func1, if scanf reads more then 4 bytes, it will overwrite is_admin variable, which can lead to gaining some privileges.

References

[1] Catalin Hritcu Arthur Azevedo de Amorim and Benjamin C. Pierce. The meaning of memory safety. 2018.