Paper Title *TODO edit*

1st Given Valentin Brandl
Faculty of Computer Science and Mathematics
OTH Regensburg
Regensburg, Germany
valentin.brandl@st.oth-regensburg.de
MatrNr. 3220018

***Abstract*—TODO**
***Index Terms*—Buffer Overflow, Software Security**

I. Motivation

When the first programming languages were designed, memory had to be managed manually to make the best use of slow hardware. This opened the door for many kinds of programming errors. Memory can be deallocated more than once (double-free), the programm could read or write out of bounds of a buffer (information leaks, buffer overflows). Languages that are affected by this are e.g. C, C++ and Fortran. These languages are still used in critical parts of the worlds infrastructure, either because they allow to implement really performant programms, because they power legacy systems or for portability reasons. Scientists and software engineers have proposed lots of solutions to this problem over the years and this paper aims to compare and give an overview about those.

Reading out of bounds can result in an information leak and is less critical than buffer overflows in most cases, but there are exceptions, e.g. the Heartbleed bug in OpenSSL which allowed dumping secret keys from memory. Out of bounds writes are almost always critical and result in code execution vulnerabilities or at least application crashes.

II. Sources

- RAD: A Compile-Time Solution to Buffer Overflow Attacks [1] (might not protect against e.g. vtable overrides, PLT address changes, ...)

III. Main Part, TODO

A. Background

text

B. Concept and Methods

- Runtime bounds checks
- Prevent overriding return address
- Restricting language features to a secure subset
- Static analysis
- Dependent types (only allow indexing with values that are provably in bounds)

C. Discussion

text

IV. Concusion and Outlook

text

References

[1] T.-c. Chiueh and F.-H. Hsu, "RAD: A Compile-Time Solution to Buffer Overflow Attacks," in *Proceedings 21st International Conference on Distributed Computing Systems*, 2001.