Investigating the Security of the Smaller C Compiler

Anonymous Author(s)

ABSTRACT

The abstract is a fancy way to saying **Summary.** It should preferably be two paragraphs that summarize your report. Abstracts are read independently from the rest of the report so you must not cite your report or any other papers. Study other abstracts in the papers you have been reading to understand what an abstract should mean, although many are poorly written.

The abstract is not an introduction or overview of your report! It is a summary of your report, which should include the background, context, content, and contributions/results of your report. Make sure you only take credit for what you did (which is the comparison and your learning) and mention all work (research ideas, software, etc.) done by others.

The first paragraph should provide an overview of the topics being covered in the report. The second paragraph should describe what you learned and and how it would be meaningful to your reader, but do not this in the first person.

Write the entire abstract in the third person and in past tense. It should typically be around 200-250 words.

KEYWORDS

Come up with your own descriptive keywords to make possible for a potential reader to find your report.

1 OVERVIEW

We are investigating how the "Smaller C" compiler addresses various security vulnerabilities common in compilers, examining the security impact of any bugs or compiler restrictions, and documenting any vulnerabilities that we find.

Some compilers are also able to catch and address many of the most common software vulnerabilities described in the Common Weakness Enumeration [2]. We intend to see if Smaller C has any checks in place to address Out-of-bounds Read and Writes, Use after free, null pointer dereferencing, or use of shared resources with improper synchronization. These issues may be possible to catch in a compiler, or prevent from happening all together in a language specification. Since this compiler is based on C, it would be surprising if any of these issues are addressed. Additionally, due to the limitations of Smaller C being a single pass compiler, it may not even be possible to catch some of these issues all of the time, but it should be possible to add safety features to catch some of the issues sometimes. We will take time during this project to find possible solutions for common vulnerabilities that work in single pass compilers.

Since this is a compiler for C, we will be investigating how the compiler handles memory allocation and access, looking for measures put in place to safeguard against buffer overflows, invalid memory, or other potentially insecure behaviors. These issues are most dangerous when the data being accessed is on the stack, as this could allow the function return address to be overwritten, allowing malicious actors to execute arbitrary code. It will be important to investigate how, if at all, Smaller C attempts to prevent these types of attacks. David Monniaux details one method for catching these attacks, in which a randomly generated "canary" value is inserted before the location that the return address is stored. [4] This value is checked at the end of function execution. If it was changed, the return address was likely changed too, so the program stops execution to prevent any undesired behavior.

Code Optimization in compilers can often unintentionally remove security checks or make code less secure. This can be especially true when compiling on hardware different than the intended program location. We will explore how Smaller C handles common optimization techniques, such as dead store elimination. D'Silva et al. laid out how dead store elimination in GCC removed a necessary security feature from the function crypt(). [3] We will see if Smaller C has similar erroneous optimization. Another common error with optimization is how the compiler handles undefined behavior within the C standard. There have been noted cases, by Wang et al. where various security checks are removed due to it being undefined by the C standard. [5]

Smaller C has some unique limitations that may, if improperly implemented, lead to security vulnerabilities. First, there is no stack overflow check, and extremely few compile-time checks. This might cause a program to be able to be frozen by putting it into an infinite loop. Secondly, large integers constant are ambiguously assigned to types of different sizes unless they are explicitly marked. This could cause different amounts of space to be allocated, which could cause issues in extreme cases.

2 RELATED WORK

Review the related work in the literature, both research and practice, to place your project in perspective, and what other people have been doing to address this problem. Make sure your literature survey is fairly complete and recent, which means you should references that are as new as possible, i.e., some (not necessarily all) should be in the current or last year.

You must cite your sources correctly per ACM style guidelines (and of course, you need to use LaTeXand BibTeXcorrectly).

3 DESIGN AND IMPLEMENTATION

Use this section to describe the basic design, architecture, and implementation of your project

4 ANALYSIS

Use this section to describe the analysis of your project that you conducted, and whether the results are meaningful or not.

Also, discuss what all you learned from the project, especially what mistakes to avoid in the future.

5 LEGAL CONSIDERATIONS

Use this section to discuss legal issues relevant to your project, especially relating aspects of data that are relevant to your project.

Use the textbook and your readings to guide the legal aspects of your discussion. Look at the laws that have been passed in recent years, and look at legislation that is being proposed in the space covered by your project.

6 ETHICAL CONSIDERATIONS

Use this section to discuss ethical issues relevant to your project, especially relating aspects of data that are relevant to your project.

Use the ACM Code to guide the ethical aspects of your discussion [1].

7 CONCLUSIONS

Use this section to describe the current status of your work and what else needs to be done.

Also, discuss what further directions your work can be taken by others.

Finally, present some final words to place your project in perspective.

REFERENCES

[1] Association for Computing Machinery. 2018. ACM Code of Ethics and Professional Conduct. ACM, New York. https://www.acm.org/code-of-ethics.
[2] CWE. 2023. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html
[3] Vijay D'Silva, Mathias Payer, and Dawn Song. 2015. The Correctness-Security Gap in Compiler Optimization. In *2015 IEEE Security and Privacy Workshops*. IEEE, IEEE, New York, NY, 73–87.
[4] David Monniaux. 2024. Memory Simulations, Security and Optimization in a Verified Compiler. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. Association for Computing Machinery, Association for Computing Machinery, London, UK, 103–117.
[5] Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama. 2013. Towards optimization-safe systems: analyzing the impact of undefined behavior. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. Association for Computing Machinery, Association for Computing Machinery, New York, NY, 260–275.