
International Journal of Control Theory and Applications

ISSN : 0974–5572



Volume 9 • Number 40 • 2016

# Enhancing the Security of C Programs with the Standard GCC Compiler and a Static Analyser

**R. Subburaj[a] Anuj Singh[a] Deepak Singal and Aditya Shah[a]**

[a]*Department of Information Technology, SRM University, Kattankulathur-603203, Chennai area*

*E-mail: anuj.singh111995@gmail.com, @corresponding author*

***Abstract:*** A significant amount of application and systems programming is carried out in C language. Popular languages like Python, Java, etc. also make use of C libraries. Code written in C is susceptible to many security vulnerabilities which can be used by an attacker to halt system operations or intrude in the system database. Hence, it is essential that all code in C is free of any vulnerable construct found in C libraries. In order to address this issue, we developed a tool which combines a standard GNU GCC compiler with a static analyser developed by us. The analyser uses an expandable repository of insecure coding constructs to check for their presence in any C code. The static analyser and the tool interface were both designed in C#. The analyser displays the insecure constructs along with their line numbers and also provides suggestions for replacing the well-known screened vulnerable constructs. Alongside these functions, a programmer can also compile C programs with the help of the integrated GCC compiler.

***Keywords:*** *C programming, static analysis, secure coding, compiler optimization*

## 1. INTRODUCTION

C and C++ are widely used as programming languages for systems development due to their performance and efficiency. Source code of security critical systems is heavily audited, tested and often verified [1]. This is due to the fact that code written in C, due to the language's age, is susceptible to various security vulnerabilities. These vulnerabilities can allow an attacker to wreak havoc in a system which was otherwise thought to be secure. Nowadays, even hardware companies are designing hardware with inbuilt security countermeasures.

Cyber-attacks are becoming increasingly common place and with their rise, the security of all system programs and applications is at risk. According to a survey conducted in China, applications and software developed by newer (start-up) companies include more security vulnerabilities and as a result are targeted more frequently by attackers [2].

Usually, in a system of source code, compiler and hardware, the compiler acts as the weak link as a compiler does not add to the security of the code in any manner. Such concerns regarding compilers and compiler optimizations are fairly common as nearly 50 years of research has been carried out to prove the correctness of compilers[3,4,5,6].

To find the security vulnerabilities in developed code, two types of analysis can be carried out: – Static Analysis and Dynamic Analysis. Static analysis is carried out before the execution of the program while dynamic analysis is carried out during the execution of the program. It is worth noting that according to findings, static analysis is more appropriate for finding security vulnerabilities [7].

In this paper, we discuss about the Optimizing Compilers in Section 2 and Security Vulnerabilities and the Need for Static Analysis in Section 3. After the proposed solution in section 4, summary and conclusions are given in Section 5.

## 2. OPTIMIZING COMPILERS

GCC provides various levels of optimization which can be controlled by the developer [8]. An optimizing compiler is a compiler that tries to minimize or maximize some attributes of a program.

Usually compiler optimization is implemented using a sequence of optimizing transformations which are algorithms that take a program and transform it to produce a semantically equivalent output program that uses less system resources. The various protocols which are needed for transformations about source code and other representations are based on thelanguage standard. According to this bug report[9], GCC 3.2 only preserves the connotations of the C language as specified by the ISO 9899 standard. These standards do not specify the memory footprint, size of activation records, stack usage, time and power consumption. This results in a situation where even if the source code has been designed while keeping in mind the security of the code, an optimizing compiler may contribute to the loss of those defence mechanisms.

The correctness-security gap arises when a compiler optimization preserves the functionality but violates a security guarantee made by source code.Compiler optimizations and the correctness-security gap is discussed in great detail in [1].

Since most compilers are optimized, it is imperative that security vulnerabilities are detected and eliminated in a careful manner before compilation.

## 3. SECURITY VULNERABILITIES AND THE NEED FOR STATIC ANALYSIS

Security vulnerabilities are a result of software design and implementation practises which do not put an emphasis on protecting systems. Software vulnerabilities are being detected at the rate of over 4,000 per year[7].

C and C++ programming languages have the following types of security vulnerabilities[10]:

1. Integer Errors
2. Code Injection attacks
3. Buffer overflows
4. Format String Vulnerabilities
5. Pointer Subterfuge
6. Dynamic Memory Management
7. Race Conditions

Even if the developer is aware of the presence of security vulnerabilities in a language, there always is a possibility of including these vulnerabilities in end user products unless the developer is completely focussed on security during all phases of development [11].

Also, presence of security vulnerabilities may cause failure of mission critical systems developed in C or C++. One simple software error or a coding mistake can cause the failure of expensive missions, as a NASA mission to Mars could cost at least 250 million USD) [16].

According to a survey on security perspectives, insecure software can cause about sixty percent of higher business threat than secured software. Currently, it is not a compiler's function to check for security vulnerabilities in C programs. Due to this, static analysers are needed which capture such security vulnerabilities before run-time[12]. Furthermore, studies show that secure software can diverge widely based upon the user requirements, updated designs and their implementations [13,14,15].

The advantage of spending more time on analysing the software decreases the amount of time spent in testing. [16]

However, static analysers will never outperform a good manual audit of the code. This is due to the fact that there are too many variables in this analysis and it is quite impossible to include them all in a completely automated scan. Static Analysers can only detect vulnerabilities which are pre-programmed[17,18].

[19] discusses the effectiveness of various types of static analysis approaches including the string and pattern matching approach.

## 4. PROPOSED SOLUTION

We have developed a tool which combines the standard GNU GCC compiler with a self-developed static analyser named "TraC++". The complete software tool is named "Secure Compilation Tool".

The static analyser uses string and pattern matching approach to identify vulnerable constructs in a given C as well as C++ code. For this, it uses a modifiable repository of vulnerable constructs which is saved on disk and is accessible by the user of the tool. If vulnerable constructs are found, they are displayed in the output field along with the line numbers on which these constructs are present. The analyser also suggests secure constructs which can be used to replace well known vulnerable functions. The tool ensures that the code is following secure coding standards. CERT (Computer Emergency Response Team) is involved in the development of coding standards for programming languages such as C, C++, Java, Pearl, etc. [20]

The analyser automatically ignores vulnerabilities present in the comment sections in order to remove false positives and generate more accurate results.

The analyser is integrated with the compiler with the help of a GUI developed in C#. Upon starting the tool, the user first enters his login credentials for an added layer of security. After this step the user can check their code with the help of the analyser.

The user then has the option of securing his code or leaving it as is because there can be certain scenarios where the vulnerable construct is desirable. The user can then click on thecompile button which will convert the C source code into object code and then into an executable file.

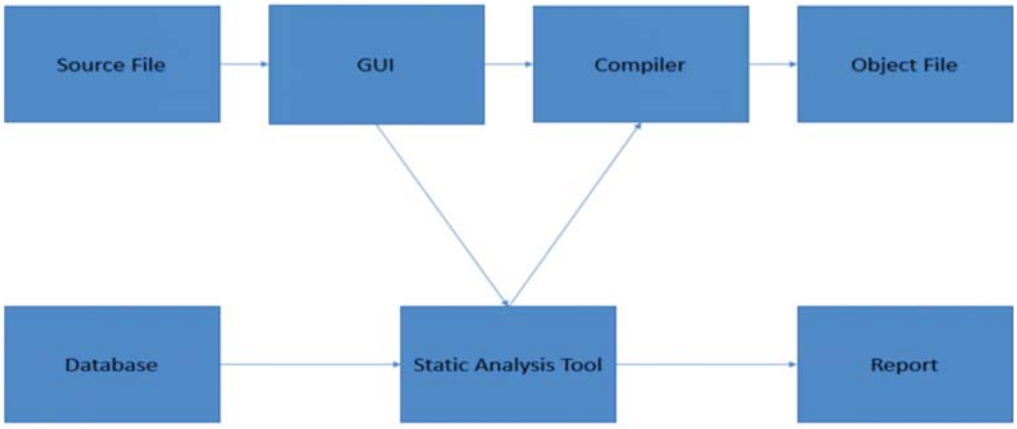


**Figure 1: System Architecture of the tool**

## 5. SUMMARY AND CONCLUSIONS

In this paper, we discussed static analysis and the need for static analysisbefore compilation in order to make C programs more secure and efficient. We also discussed the limitations of optimizing compilers and the correctness-security gap which arises. A tool which combines a static analyser as well as the GCC compiler was developed and is described here. Extensive testing was carried out with the tool to make sure the tool operates as per the requirements. In any given piece of C code, the Secure Compilation tool finds the vulnerable constructs which have already been identified and are stored in the vulnerable construct repository. The tool is able to create executable files from source code using the GCC compiler in a Windows environment. With the help of this tool, more secure software can be delivered to users faster than before. The overall level of security in C programs will rise substantially with the increased usage of this tool. Furthermore, the process of systems development will be more efficient and reliable. This tool can prevent a lot of insecure programs from entering the consumer space. We are looking forward to further research and development in this field.

## REFERENCES


[1] V. D'Silva, M. Payer, and D. Song,"TheCorrectness-SecurityGapinCompilerOptimization", IEEE CS Security and Privacy Workshops, 2015.

[2] C. Huang, J.Y. Liu, Y. Fang, and Z.Zuo,"A study on Web security incidents in China by analyzing vulnerability disclosure platforms", Computers and Security. 2016 May; Vol 58. pp.47–62.

[3] M. A. Dave, "Compiler verification: a bibliography," SIGSOFT Software Engineering Notes, vol. 28, no. 6, pp. 2–2, Nov. 2003.

[4] J. McCarthy and J. A. Painter, "Correctness of a compiler for arithmetic expressions," in Proc. of the Symposium in Applied Mathematics,vol.19. AmericanMathematicalSociety,1967, pp. 33–41.

[5] J. A. Painter, "Semantic correctness of a compiler for an Algol-like language," Stanford University, DTIC Document ADA003619, 1967.

[6] K. Thompson, "Reflections on trusting trust," Communications of the ACM, vol. 27, no. 8, pp. 761–763, Aug. 1984.

[7] R.C. Seacord, Secure coding in C and C++. 2nd Ed. USA: Addison-Wesley Professional; 2005.

[8] M.T. Jones, "Optimizations in GCC", http://www.linuxjournal.com/article/7269, Nov 2005.

[9] J. D. Wagner, "Bug 8537 - optimizer removes code necessary for security," https://gcc.gnu.org/bugzilla/showbug.cgi?id=8537, Nov. 2011.

[10] R.Subburaj, P.U. Raikar, and S.P. Shruthi,"Static Analysis of Security Vulnerabilities in C/C++ Applications", Indian Journal of Science and Technology.2016 May; 9 (20);1-4.

[11] J. Viega, J.T. Bloch, Y. Kohno, and G. McGraw. "ITS4: A Static Vulnerability Scanner for C and C++ Code", 16th Annual Conference, IEEE; New Orleans, LA. 2000 Dec. p. 257–67.

[12] R. Subburaj, A. Singh, and D. Singal; "Enhancing the Security of C/C++ Programs using Static Analysis",Indian Journal of Science and Technology, Vol 9(44), DOI: 10.17485/ijst/2016/v9i44/104031, November 2016

[13] P. Nikhat, N.S. Kumar, andM.H. Khan,"Model to quantify integrity at requirement phase", Indian Journal of Science and Technology. 2016 Aug; 9(29):1–5.

[14] F. Flechais, M. Sasse, and S.M.V. Hailes, "Bringing security home: A process for developing secure and usable systems", NSPW'03. ACM. USA; 2003 Aug. p. 18–21

[15] B.B. Madan, K.G. Popstojanova, K. Vaidyanathan, and K.S. Trivedi,"A method for modelling and quantifying the security attributes of intrusion tolerant system", An International Journal of Performance Evaluation. 2004; 56(1):167–86.

[16] G. Brat and A. Venet,"Precise and scalable static program analysis of NASA flight software",Proceedings of the 2005 IEEE Aerospace Conference (2005).

[17] J. Nazario,"Source code scanners for better code" http://www.linuxjournal.com/article/5673?page=0,2. Jan, 2002

[18] A.V. Revnivykh andA.M. Fedotov,"Root Causes of Information Systems Vulnerabilities", Indian Journal of Science and Technology, 2015 December; 8 (36).

[19] P. Emanuelsson and U. Nilsson,"A comparative study of industrial static analysis tools", Electronic Notes in Theoretical Computer Science. 2008 Jul; 217:5–21.

[20] SEI CERT Coding Standards [Internet]. 2016 [accessed 2016 Oct 06]. Available from: https:// www.securecoding.cert. org.