International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-3S, February 2019

Compiler Architecture for Detection of Suspicious or Malicious Strings in a Program

Rajeshwari K Gundla

ABSTRACT: ***Malware (i.e. malicious software) is one of the most serious security threat on the internet today. Malwares can be of various types such as virus, worms, Trojan horse, ransomware, adware, rootkit, spyware etc. Programmer write code to get desired results while doing this he/she can involve vulnerabilities in code, Attacker can exploit these vulnerabilities in the code in order to gain confidential information or gain unauthorized access to the user system. Programs need to be compiled and run to create an executable file for which compiler is needed. Compiler is a system program that translates a code written in source language into target language. In this paper we propose a model where we redesign a compiler in such a way that compiler identifies suspicious/malicious string at compile time itself. Compiler gives warning of suspicious code. Compiler comprises many phases. Lexical analysis is the first phase of the compiler which does many tasks such as removing whitespaces and comments, making entry to the symbol table, correlating line number with error message. Here we are adding additional task that is suspicious/malicious string detection to lexical analysis phase of the compiler. Static malware analysis can be done at compilation phase itself.***

Keywords – Compiler Architecture, Lexical Analysis, Static Malware Analysis, Suspicious/Malicious String Detection

I. INTRODUCTION

Some of the most concerning and pressing issues in the security of the Internet is Malicious code or malware. Several studies significantly focused on developing techniques to collect, study and mitigate malicious code. Without a second thought, it is important to collect, study and mitigate malware found on the Internet but there is very little research focused on preventing program vulnerabilities and malicious code getting compiled.

Programmer writes a code which needs to be compiled to create executable file. Compiler is a system program that translates a source language into target language. As shown in Fig. 1 the structure of a compiler is composed of several phases which are divided into two parts Front end and Back end. Front end consists of lexical analyzer, syntax analyzer, semantic analyzer and intermediate code generator. Back end consists of code optimizer and code generator [5].

The first phase is lexical analysis or scanning. This is the only phase which interacts with original source code written by the programmer and perform the various tasks which includes, read the input characters and produce as output sequence of tokens that the parser uses for syntax analysis, and make entry to symbol table, Stripping out from the source program comments and white space in the form of blank, tab, new line characters. Another task is correlating error messages from the compiler with the source program and keeping the track of error by line number [3].

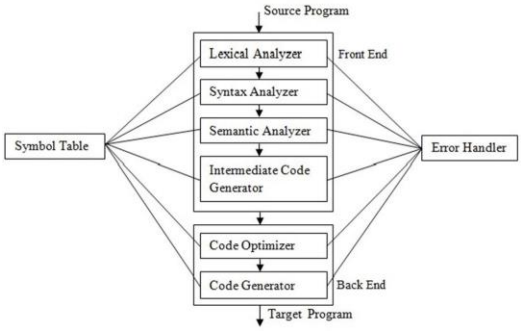

Fig. 1: Structure of Compiler

In this paper we propose a model where compiler itself detect suspicious/malicious code by detecting suspicious strings. Suspicious/malicious string detection is also called signature based detection. This signature based detection can be added to first phase of compiler that is lexical analysis phase. Thus along with basic tasks of lexical analyzer, suspicious/malicious string detection task can be added to the tasks of lexical analyzer in order to perform static malware analysis in the compilation phase itself so that it becomes easy to detect suspicious/malicious code in compilation phase itself.

The goal of our system is to include static malware analysis in one of the phases of the compiler. Programmer writes a code which needs to be compiled to create executable file. If executable file is malicious it can spread into the network in order to perform malicious activities. In this paper we design a method where static malware analysis is done in compilation phase itself. Here compiler itself gives error/warning that program code can be malicious or suspicious.

II. RELATED WORK

As described by Abhishek Nayyar, Umang Saxena and Arun Kumar, program vulnerabilities may be unwarranted for any organization and may lead to severe system failure. Due to the advancement of technology there has been increase in the area of vulnerability attacks which are exploited by hackers for getting access to the system or insertion of their malicious code.

Manuscript published on 28 February 2019.
* Correspondence Author (s)
Rajeshwari K Gundla, School of Information Technology Ajeenkya D Y Patil University, Pune, India. (E-mail: radhikagundla22@gmail.com)

Authors presented a proposal for compiler design which prevents some common vulnerability. The output result for compiler would be compile time warning stating the possible vulnerability in the code.

As described by Luke Jones, Andrew Sellers, Martin Carlisle, authors of malicious software, or malware, have a plethora of options when deciding how to protect their code from network defenders and malware analysts [1]. For many static analyses, malware authors do not even need sophisticated obfuscation techniques to bypass detection, simply compiling with different flags or with a different compiler will suffice. Authors proposed a new static analysis called CARDINAL that is tolerant of the differences in binaries introduced by compiling the same source code with different flags or with different compilers [2].

Asaf Shabtai et al. author presents a method for detecting unknown malicious code by applying classification techniques on OpCode patterns [6].

The exponential growth of the Internet interconnections has led to a significant growth of cyber- attack incidents often with disastrous and grievous consequences. Malware is the primary choice of weapon to carry out malicious intents in the cyberspace, either by exploitation into existing vulnerabilities or utilization of unique characteristics of emerging technologies. Authors discussed about new attack patterns in emerging technologies such as social media, cloud computing, smartphone technology, and critical infrastructure [7].

III. PROPOSED SYSTEM & RESULTS

From the literature studies, it is found that if there are vulnerabilities in the program it can be exploited from the attackers. Some researchers also worked on detecting program vulnerabilities at compilation phase [1]. From the literature studies it has been seen that very little work has been done on compiler design to detect malicious/suspicious code in compilation phase itself.

Here we are proposing a compiler architecture to detect suspicious/malicious strings in a program at compilation phase. Compiler has different phases such as lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization and code generation. Here we are focusing on first phase of compiler that is lexical analysis.

Fig. 2 shows existing system of lexical analyzer. As shown in Fig. 2 lexical analyzer takes the input source program into the input buffer. It reads source program character by character until it finds a meaningful lexeme, whitespace, tabs, newline and comment. Based on lexeme found, Lexical analyzer performs following tasks:

1. If extracted lexeme is a token then it generates <token id, value> and if token is identifier or constant then it makes entry to symbol table.
2. If extracted lexeme is comment, it removes comment lines and checks for next lexeme.
3. If it finds whitespace then it removes it and checks for next lexeme.
4. If it finds newline, it keeps track of line number which is used by error handler [4].

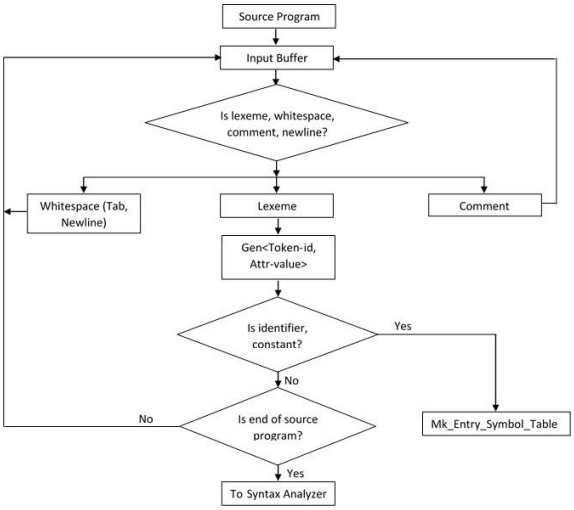

Fig. 2: Existing System of Lexical Analyzer

As shown in Fig. 3 (Proposed System of Lexical Analyzer), lexical analyzer reads source program character by character until it finds a meaningful lexeme, whitespace, tabs, newline and comment.

1. If whitespace, tab is found, lexical analyzer simply removes whitespaces and tabs.
2. If comment is found, lexical analyzer removes the comment.
3. If newline is found, lexical analyzer keeps track of line number to associate error message with line number.
4. If lexeme is found, lexical analyzer generates token for lexeme.
    a) If lexeme is variable/constant, it makes entry to the symbol table.
    b) If lexeme is string literal, it checks if that string is suspicious/malicious. If string is malicious then it gives warning to the user. If string is not suspicious/malicious then it reads the next character.

Fig. 3: Proposed System of Lexical Analyzer

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

IV. CONCLUSIONS

A method to detect suspicious/malicious string which can be added as a task to the lexical analysis phase of the compiler is presented. It is clear that substantial amount of time can be saved by detecting suspicious/malicious string in the compilation phase itself. This speedup is expected to increase further if source code (which is input to lexical analysis phase) is divided on some criteria and processed parallel.

Along with suspicious/malicious string detection, detection of suspicious/malicious code can be added to compilation phase. And to improve efficiency and effectiveness of compiler we can execute tasks of lexical analyzer parallel/concurrently.

Authors of malicious software, or malware try to protect their code from network defenders and malware analysts. So compiler should be designed in such a way that it should detect a code which tries to bypass network defenders and malware analysts.

Also Programmer while writing program should consider security measures. Nowadays programmers write program code without considering security concerns. This creates more number of program vulnerabilities which invites attackers to get into the system and do malicious activities. So programmer should consider program vulnerabilities while writing programs for security reason.

We are currently working on further improvements of compiler design so that along with suspicious/malicious string detection, detection of suspicious/malicious code can be added to compilation phase. We are also working on parallel/concurrent execution of tasks of lexical analyzer to accelerate the speed of compiler with this additional task of suspicious/malicious string detection.

REFERENCES

1. Nayyar, Umang Saxena and Arun Kumar. Article: “Compiler for Detection of Program Vulnerabilities”. International Journal of Computer Applications 104(6):25-31, October 2014.
2. Luke Jones, Andrew Sellers, Martin Carlisle, “CARDINAL: similarity analysis to defeat malware compiler variations: 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, ISBN: 978-1-5090-4542-6.
3. Alfred Aho and Jeffrey Ullman, “Principles of Compiler Design” Second Edition.
4. Rohitkumar Rudrappa Wagdarikar and Rajeshwari Krishnahari Gundla “Parallel Execution of Tasks of Lexical Analyzer using OpenMP on Multi-core Machine”, Eighth International Conference on Advances in Computer Engineering ACE 2018, McGraw Hill Education, Jan 20 2018.
5. Lexical analyzer, https://www.tutorialspoint.com.
6. Asaf Shabtai et al.”Detecting unknown malicious code by applying classification techniques on OpCode patterns” licensee Springer. 2012.
7. Navnita Nandakumar et al, “EMERGING AND UPCOMING THREATS IN CYBER SECURITY IN 21 CENTURY” International Journal of Computer Science and Mobile Computing, Vol.6 Issue.2, February- 2017, pg. 107-118.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)