The Detection of Software Security Vulnerabilities Through the Usage of Unsafe Functions

Mohammad Barrawi, Alexander VanBeekum, Oliver Sue, and Zaid Altahat*
Computer Science Department
University of Wisconsin-Parkside
Kenosha, WI 53144
{barra002, vanbe015, sue00002}
@rangers.uwp.edu, *altahat@uwp.edu

Saleh Alnaeli
Mathematics, Statistics and Computer Science Department
University of Wisconsin-Stout
Menomonie, WI 54751
alnaelis@uwstout.edu

Abstract

The premise behind this research project is to aid programmers in writing more secure software systems. It does that by examining the distribution and the usage of some unsafe functions that are known to introduce security vulnerabilities to the software systems written in C/C++. This work extends prior research led by two of the current authors and aims at improving the accuracy and the practicality of the prior work (e.g., improving the vulnerability mechanism and identifying the potential false positives). With the proposed approach, software developers can integrate their software repositories with our solution through GitHub webhooks. Using this technique, our analysis tool will be automatically invoked each time a change is committed to the repository with instant and more accurate feedback. Additionally, integrating GitHub eliminates the need to create an account to conduct the system analysis making the tool easy to use. Future improvements are also discussed.

1. Introduction

The Proliferation of Cloud Services, distributed computing resources, and today's internet speed and quality, a substantial number of software systems are being designed and developed as Open Source Systems (OSS). This open development approach has gained widespread acceptance in the computing communities from both academia and industry. As those systems are present in most of the today's computers and devices (including mobile and small-scale devices) one of the cores aspects that will be used to determine the reliability and the quality, as well as the success of those systems is security. That is, the security standards used when developing the open source software is a core concept and real concern for the end users and institutions who plan to use these open source systems, fully or partially, as one of their business components that plays a major role in their business operations and processes [1-3].

This project concerns some of the aspects and techniques that can help the community build better software systems, especially the open source systems that are typically built and developed by programmers from different computing backgrounds and with a diverse level of skill and proficiency when it comes to software engineering practices and security standards. The objective of this research project is to better the world of software engineering by aiding programmers in the development cycle. Our team is in the process of solidifying an algorithm that helps define what constitutes the safety of a function. By bringing to light unsafe function calls within a program we can help developers create code that is both efficient and secure. We have chosen to tackle the behemoth amidst the unsafe functions which is the unsafe functions that are due to memory issues. Memory allocation is an integral part for both software and hardware optimization.

The premise behind this research project is to aid programmers in writing and developing code that is more secure. It does that by examining the distribution and the usage of some unsafe functions, and their alternatives, that are known to introduce security vulnerabilities in software systems written in C/C++. This project has been worked on previously by another group of software engineering students who have published initial findings in the IEEE EIT [1]. But the scope was broad and did not focus on one aspect of the research. Our team has determined the process that needs to be taken and narrowed the scope in a way that makes the objective clear and feasible. Our effort is a work in progress that will lay the ground for the phase 2 of the project that will be worked on during the summer of 2019. The project is now setup to expands on the previous work by identifying false positives. A percentage of the functions that are flagged as unsafe are categorized so due to the lack of context checking.

The algorithm will be enhanced to search the context by looking for tags of unsafe functions and then once they are found they are stored into an object. After the object variables have been allocated, they are then sized checked if they are memory-dependent functions. We then check the unsafe function objects and see if the user allocates memory in an appropriate manor. Once the test for unsafe functions is done with one segment it deletes the variables so that it may proceed to another segment and implement the

algorithm on it. The project will also require better integration the solution with GitHub. Now a repository owner on GitHub can integrate the repository with our solution through GitHub webhooks. Using this approach, the algorithm will be automatically invoked each time a change is committed to the repository. This way a feedback is immediately provided to the developer about the usage of unsafe functions.

The current framework of the project requires the users to create an account to run the analysis on their code, but with our GitHub integration that step will be eliminated. Our future plans revolve around eradicating the need of users creating accounts and allowing them to run the algorithm on cloud-based source for ease of access.

The first few steps of tackling the logic in this projects is to solve the false positives for the unsafe functions such as memcpy(), wcsncpy(), and various other functions that can cause a security leak in the program caused by an over flow of a value. These methods often through many false positives because although the use of them can lead to memory overflow, logic is often used in the code to prevent this from happening. The purpose of working with the GitHub API is to find a solution to how to automatically notify the Safe Functions program that changes were made to a GitHub repository. This would allow GitHub users to get automatic updates on how optimized their code is to prevent unsafe procedures.

2. Related Works

In both general and special purpose computing, especially open source versions, security has become an integral part of every stage of software design and development. The quality and reliability of software systems depend on many important and core aspects paired with the design and development process. For example, developers' expertise and dedication to write a robust, high quality, and secure software that meets the end user expectations, plays a major role in the success of the software in the market [6, 7, 9]. However, not all programmers of open source software systems have the aforementioned experience and dedication. Consequently, software systems that are developed by inexperienced programmers (from software engineering, quality, and security perspectives) need to be always evaluated and tested against any potential risks. One form of those expected risks is the usage of insecure and vulnerable functions and code constructs. These functions are known to increase the threat of malicious attacks against the software, if exploited by hackers.

Most of the previous research on this topic has focused on identifying some of the security concerns related to the detection and definition of the vulnerable functions. Some other studies have been conducted on the data privacy within different levels. However, with the best of our knowledge, no study has addressed the usage or the prevalence and the evolution of vulnerable code in open source software systems that are developed in C/C++,

from software engineering perspective on the source code level the way it is approached by our team with the methodologies used in our prior and current studies.

One relevant study and research we like to share in this paper is conducted by Jovanovic et al. in [10]. In their project, they have proposed a new vulnerability prediction approach based on the CERTC Secure Coding Standard. Their proposed approach has focused on the prediction, accuracy, and cost-effectiveness, and showed that their technique was able to potentially improve the accuracy of vulnerability prediction. However, the study did not reveal any historical trends or the recent status of the tested systems. It also did not show how the existing systems in the market do regarding using vulnerable code, and the trend of the usage of risky vulnerable overtime.

There are other projects in development similar to this one, however, these projects often times provide raw statistics or do not have a clean interface to interact with. That is where our project separates itself from the rest. Our project consists of a nice, clean, easy-to-use interface with no command-line arguments to remember. The application has been divided into separate tabs, or windows, to give clear lines of separation between its abilities and the data provided. Any data the user wishes to view is presented in clean, easy to understand graphs. This limits our ability to convey all information at once, but we believe that the easier to use format will be more beneficial.

3. Methodology

The approach that our research team has taken to pinpoint unsafe functions is a three-step process.

1) Convert the code to an XML representation that is in a readable format for the algorithm.
2) Runs the analyzer on the XML representation of the code so it pinpoints all the unsafe functions.
3) Determine whether the called unsafe function needs to be replaced or if there is a way to implement it safely.

The list of unsafe functions comes from different sources such as Microsoft. We have based our judgement of a function's safety on its existence on the list of unsafe function.

Upon examining the scope of how many unsafe functions are present our team decided to pinpoint the functions that were related with memory issues due to the amount of damage that these methods can inflict. Our team utilizes the SDL (Security Development Lifecycle) banned functions to become aware to the threats that a programmer can face when programming functions unsafely. The largest threat amidst all of the functions are the ones that are memory related due to the fact that they not only can allow software to misbehave, but they also can cause hardware to malfunction due to memory leaks. We are deciding to convert the C++ code into XML using srcML [2] to make it easier to work with. With the code converted for us we can use the XML tags to find when a variable is created and track every variable. Not only can we use the XML tags to track the variables created, but also

how much space the program allots for each variable and what methods the program uses each variable in, and what methods are used on each variable. These uses of XML tags will allow us to see when the program uses an unsafe function and what variables the program runs through the function in order for us to determine if the unsafe function is used in safe way.

A second component of the app is the full integration with GitHub repositories. The GitHub component of the app acts on its own, via GitHub's API. By using the API we are able to monitor when commits happen on a repository. By using the our application we can receive webhooks [3]. Webhooks allows us to set up an integration between GitHub and our test application. When one of the events that we monitor using the API it sends a HTTP POST payload via the webhook to the preconfigured URL. We use Node.js [4] for the test application to receive the webhooks. Inside the test application the code needed to have event handling to address what happens when the webhooks are sent to the url.

3.1 Finding the unsafe functions that are a priority

Memory allocation is crucial for software to function as intended, for this very reason we chose to prioritize the function search of the next phase of the project solely to the memory related functions. In the future we do intend on pursuing the resolution of other unsafe functions, but at the meantime we would like to ensure that the memory related functions are caught and dealt with accordingly due to how much of a liability they can be to programmers.

3.2 Methodology of pinpointing false positives within code

We are planning on converting the C++ code into xml using srcML to get the code into a usable format. Once the code has been converted into XML using srcML we can use java's DocumentBuilderFactory class to parse the XML into a node tree. From there we will be able to traverse through the tree seeing what logic the developer has performed on the arguments for the unsafe functions needed in order to use the functions safely. This check for logic in their code is necessary to remove false positives from being recorded where a normal unsafe function is used with logic that makes it safe.

3.3 Creation of GitHub app and relevance to project unsafe functions

To get the GitHub app working there are some prerequisites that needed to be completed. Smee [5] needed to be created, the cloud based Smee service receives payloads and then sends them to the smee client running locally on our application application. This allows us to have a place to send the webhook data. After the Smee service is set up correctly then the GitHub is registered with the URL that is created from Smee so that the webhooks have a URL to send the webhooks. From there the runtime environment needed

to be setup, getting the GitHub API keys connected to the test application. The test application was originally coded in Ruby, but will be changed to Node.js to make a more streamlined process with the rest of the project. We had to adjust settings on the GitHub app permissions. This is where we allowed the app to scan for issues which allowed us to see commits made to the repository where the app was installed. Once the code and the all the software were working together, we installed the GitHub Safe Functions app into a test repository. Whenever a commit was made to the test repository it sent a webhook to our test application running on Smee and the application display a message saying that the webhook was received.

4. Future Work

Project Safe Functions intends on expanding the scope to pursue other software vulnerabilities and unsafe functions. Our team currently wants to make sure that all memory vulnerabilities are dealt with accordingly. The first phase was identifying the usage of the unsafe functions in public source code repositories. Next is to identify the safe usage of unsafe functions. Such cases will be removed from the list of unsafe functions usage. The teams also is planning on integrating the GitHub app solution into this project of detecting unsafe functions so users can be notified of the presence of unsafe functions within their code via the embedded webhooks.

5. Conclusion

In conclusion project Safe Functions is evolving into the project we want it to be, and that is the pursuit for better and more efficient programming. Our team will expand and evolve the project to put in a good place for future additions and optimizations.

6. References

[1] 1Crifasi, Pike, Stuedemann, et. al; “Cloud-Based Source Code Security and Vulnerabilities Analysis Tool for C/C++ Software Systems”,” in the 8th Annual IEEE International Conference On Electro Information Technology; 2018; Rochester, Michigan, USA

[2] M. L. Collard, M. J. Decker, and J. I. Maletic, "Lightweight Transformation and Fact Extraction with the srcML Toolkit," presented at the SCAM'11, Williamsburg, VA, USA, 2011. Tool: https://www.srcml.org

[3] https://developer.github.com/webhooks/

[4] https://nodejs.org/en/

[5] https://smee.io

[6] E. Erturk, "A case study in open source software security and privacy: Android adware," in Internet Security (WorldCIS), 2012 World Congress on, 2012, pp. 189-191.

[7] J. R. Tubilleja, "Security issues surrounding the use of open source software by online students," in Emerging Technologies for a Smarter World (CEWIT), 2015 12th International Conference & Expo on, 2015, pp. 1-6.
[8] Alnaeli, S.; Sarnowski, M.; Aman, S.; Abdelgawad, A.; Yelamarthi, K., “On the Evolution of Mobile Computing Software Systems and C/C++ Vulnerable Code,” The 7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, 20 - 22 October 2016.
[9] S. R. Vadalasetty. Security Concerns in Using Open Source Software for Enterprise Requirements [Online]. Available: https://www.sans.org/readingroom/whitepapers/awareness/security-concerns-open-sourcesoftware-enterprise-requirements-1305
[10] Y. Joonseok, R. Duksan, and B. Jongmoon, "Improving vulnerability prediction accuracy with Secure Coding Standard violation measures," in 2016 International Conference on Big Data and Smart Computing (BigComp), 2016, pp. 115-122.