

**MANIPAL SCHOOL OF INFORMATION SCIENCES**

**(A Constituent unit of MAHE, Manipal)**

**Secure Code Chatbot For C**

|  |  |  |
| --- | --- | --- |
| **Reg. Number** | **Name** | **Branch** |
| **241059042** | **Meghana D Duttargi** | **CYS** |
| **241059044** | **Sanjith R** | **CYS** |
| **241059047** | **Prajwal K V** | **CYS** |

**Under the guidance of**

**Keerthana B**

Associate Professor,

Manipal School of Information Sciences,

MAHE, MANIPAL

**27/09/2024**

****

Table of Contents

[1. Introduction 4](#_Toc179248466)

[2. Objectives: 5](#_Toc179248467)

[3. Literature Survey 6](#_Toc179248468)

[4. Specification 8](#_Toc179248469)

[5. Block Diagram 9](#_Toc179248470)

[6. Work progress 10](#_Toc179248471)

[6.1 Week 1 10](#_Toc179248472)

[6.2 Week 2 10](#_Toc179248473)

[6.3 Week 3 12](#_Toc179248474)

[6.4 Week 4 12](#_Toc179248475)

[7. Partial Results 14](#_Toc179248476)

[8. Conclusion 14](#_Toc179248477)

**LIST OF FIGURES**

[Figure 5.1: Block Diagram](#_Toc54001391) 8

# Introduction

In the realm of software development, ensuring code security is of paramount importance, particularly when working with languages like C. While C is renowned for its performance and versatility, it is also known for its susceptibility to various security vulnerabilities if not written with care. Issues such as buffer overflows, memory management errors, and improper use of pointers can lead to severe security risks and system vulnerabilities.

To address these challenges, this project introduces the Secure Code Chatbot, an intelligent tool designed to help developers write secure C code. The chatbot provides real-time analysis and feedback on C code snippets, guiding developers through best practices for secure coding and identifying potential vulnerabilities. Its user-friendly graphical interface makes it accessible and easy to use, even for those new to secure coding practices.

This project aims to bridge the gap between developers and secure coding practices, empowering them to write safer C programs and reducing the likelihood of security vulnerabilities in production code. Whether you are a seasoned professional or just starting out in C programming, the Secure Code Chatbot is designed to enhance your development workflow, providing peace of mind that your code is both functional and secure.

# Objectives:

* Develop an Interactive Chatbot: Create a chatbot that assists developers in identifying and addressing security vulnerabilities in C code.
* Implement Static Analysis Techniques: Integrate data flow and control flow analysis to detect potential vulnerabilities in C code.
* Incorporate Static Analysis Tools: Utilize features from popular tools like GCC, Clang, Flawfinder, and Splint to enhance the chatbot’s analysis capabilities.
* Address Analysis Challenges: Manage issues such as false positives and scalability to improve the chatbot’s reliability and performance.
* Facilitate Best Practices: Provide feedback and recommendations based on secure coding best practices.
* Explore Emerging Trends: Investigate and incorporate machine learning-based approaches to enhance analysis accuracy.
* Ensure Smooth Integration: Design the chatbot for seamless integration into the software development lifecycle.
* Enhance User Experience: Develop a user-friendly GUI to make interaction with the chatbot easy and accessible for developers.

# Literature Survey

1. A Comparative Study of Static Code Analysis Tools for Vulnerability Detection in C/C++ and JAVA Source Code by Prof. Arvinder Kaur and Ruchikaa Nayyar

This study examines the effectiveness of various static code analysis tools for detecting vulnerabilities in C/C++ and Java source code. The authors highlight that static code analysis tools are crucial for identifying vulnerabilities early in the development process, thereby reducing both the cost and time needed for fixing these issues. Tools such as Flawfinder, RATS, and CPPCheck are effective in C/C++ vulnerability detection, with Flawfinder being particularly noted for identifying the most categories of vulnerabilities. Java-based tools such as SpotBugs and PMD are also discussed for their efficacy in detecting vulnerabilities in Java code. The study points out that each tool has its strengths and weaknesses, particularly in terms of the number of vulnerabilities detected and the rate of false positives, which can vary significantly.

1. A Comparison of Static Analysis Tools for Vulnerability Detection in C/C++ Code by Andrei Arusoaie, Ciobac, and Vlad Craciun

This paper focuses on the challenges and limitations faced by static analysis tools when detecting vulnerabilities in C/C++ code. The authors note the issues related to false positives and the inconsistent vulnerability taxonomies used by different tools. The study benchmarks several popular tools, including CodeSonar, CppCheck, and Facebook Infer, using the Toyota ITC test suite. The evaluation criteria include precision, recall, and running time, all of which are key metrics for determining the effectiveness of these tools. The paper emphasizes the need for more tailored and focused frameworks for comparing these tools, particularly in the context of C/C++ vulnerabilities, where existing surveys are often insufficient.

1. (Partial) Program Dependence Learning by Aashish Yadavally and Tien N. Nguyen

This research explores the role of static analysis tools in enhancing the security of C code by providing real-time feedback to developers. The authors demonstrate how these tools are useful in detecting security issues such as buffer overflows and memory management errors, contributing to the overall reliability of the code. By automating code reviews, these tools make it easier for developers to identify and fix vulnerabilities during the development process. The integration of static analysis into development environments with customizable rules helps make security checks an integral part of the coding workflow, further enhancing secure coding practices.

1. BegBunch – Benchmarking for C Bug Detection Tools by Cristina Cifuentes and Christian Hoermann

Cifuentes and Hoermann introduce the BegBunch benchmarking suite as a solution to the deficiencies observed in current C bug detection tool benchmarks. They note that existing benchmarks often lack comprehensive evaluations and automated validation, making it difficult to assess tool effectiveness accurately. BegBunch addresses these issues by offering an Accuracy Suite for evaluating precision and recall, as well as a Scalability Suite for testing the performance of bug detection tools on large codebases. This automated benchmarking framework provides consistency and reproducibility in evaluations, making it an important resource for researchers and developers looking to assess the effectiveness of C bug detection tools.

1. An Empirical Evaluation of GitHub Copilot’s Code Suggestions by Nhan Nguyen and Sarah Nadi

This study evaluates GitHub Copilot, an AI-driven tool that suggests code snippets to developers, with a focus on the security implications of its code suggestions. While Copilot can significantly enhance productivity by speeding up the coding process, the study warns that it may introduce insecure code patterns if the suggestions are not carefully reviewed. The authors highlight the need for developers to balance the efficiency gained through the use of AI tools like Copilot with the need for manual oversight to ensure secure and high-quality code. The research underscores the importance of vigilance when using AI-driven code generation tools in security-critical environments.

# Specification

**Software Requirements**

* **Operating System:** Windows 10
* **Programming Languages:** Python 3.8 or higher**,** C Compiler (GCC, Clang)
* **Development Environment:**

Integrated Development Environment (IDE) such as Visual Studio Code, PyCharm Command Line Interface (CLI) for running and compiling C code (e.g., Terminal, Command Prompt)

* **Libraries and Frameworks:**

Python libraries: tkinter (for GUI), (for static analysis)

C libraries: Standard C Library (for compiling and testing C code snippets

# Block Diagram

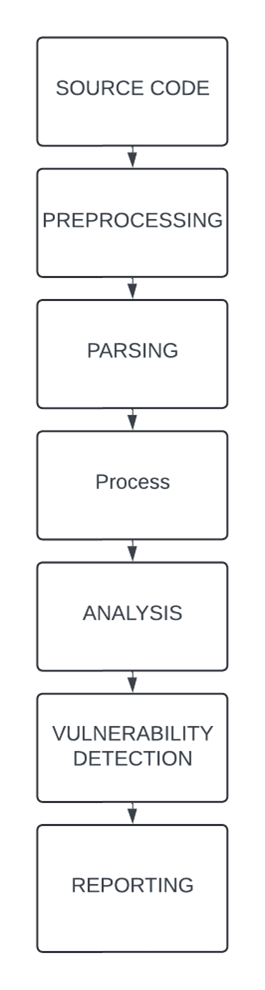
****

Fig 5.1. Block Diagram

# Work progress

6.1 Week 1**:**

* Gone through the IEEE paper to find the vulnerability in c

[1] Detailed list of the key vulnerabilities discussed:

* Buffer Overflow:  Happens when more data is written to a buffer than it can hold, causing data to overwrite adjacent memory, potentially leading to system crashes or code execution.
* Divide-by-Zero: This vulnerability occurs when the program performs division by zero, causing unpredictable behaviour or system crashes.
* Memory Leak: Memory leaks happen when allocated memory is not freed, leading to resource exhaustion over time, especially in long-running applications.
* Integer Overflow: This occurs when an arithmetic operation results in a value that exceeds the maximum limit of the data type, potentially causing buffer overflows or other unexpected behaviour.

[2] Focuses on vulnerability detection using static analysis tools:

* The document lists a variety of common vulnerabilities that static analysis tools aim to detect, such as
* Buffer Overflows,Integer Overflows, NULL Pointer Dereferencing and Uninitialized Variables.

6.2 Week 2**:**

* Researched about the different GUI their advantages and disadvantages: -
* Python offers several GUI frameworks, each with its strengths and weaknesses. Here are some of the most commonly used Python GUI libraries, followed by why tkinter stands out as one of the best options:
  + - * 1. **Tkinter:**

Tkinter is the standard GUI library for Python, included with most Python distributions:

**Advantages**:

**Built-in**: No need for additional installations; it's part of the standard Python library.

**Cross-platform**: Works on Windows, macOS, and Linux without significant changes in the code.

**Simplicity**: It's relatively easy to use for beginners and has a straightforward syntax

**Large Community:** Being the standard Python GUI, there is extensive documentation and a large community to help with development.

**Lightweight**: It is not resource-heavy compared to other frameworks

* + - * 1. **PyQt / PySide (Qt for Python):**

**Overview**: PyQt and PySide are Python bindings for the Qt framework, which is a popular C++ library for cross-platform GUI development.

**Advantages**:

**Rich Features**: Includes support for multimedia, network communication, databases, and more.

**Beautiful Widgets**: Offers advanced widgets and controls for more polished UIs.

**Cross-platform**: Windows

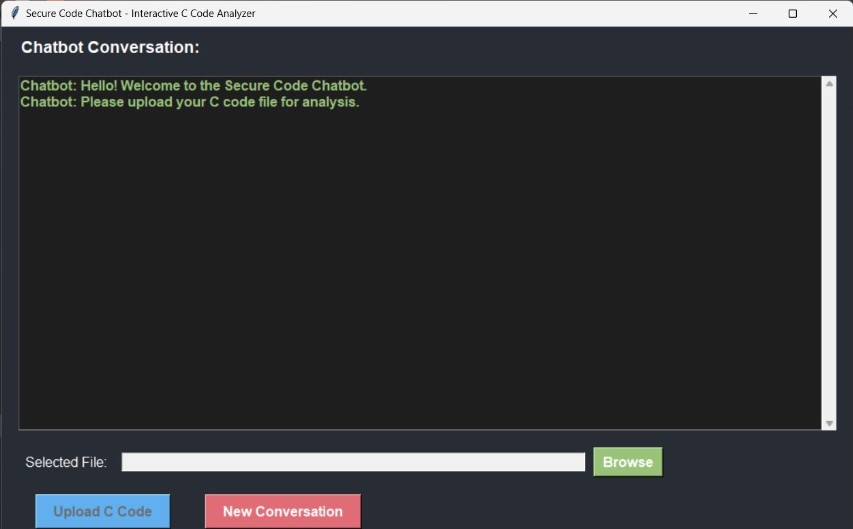
**Disadvantages**:

**Steeper Learning Curve**: More complex than Tkinter, with a larger API and more features.

**Licensing**: PyQt comes with GPL licensing (unless you purchase a commercial license), whereas PySide is LGPL.

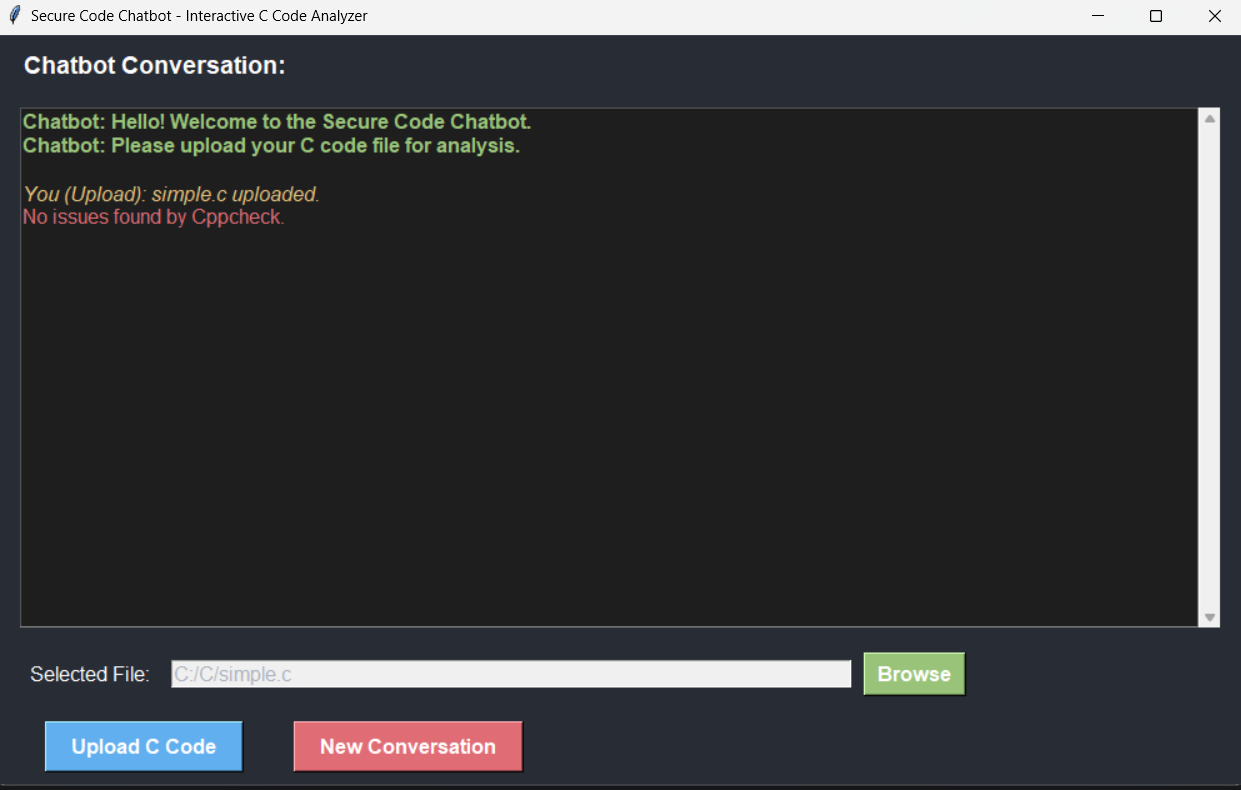
6.3 Week 3**:**

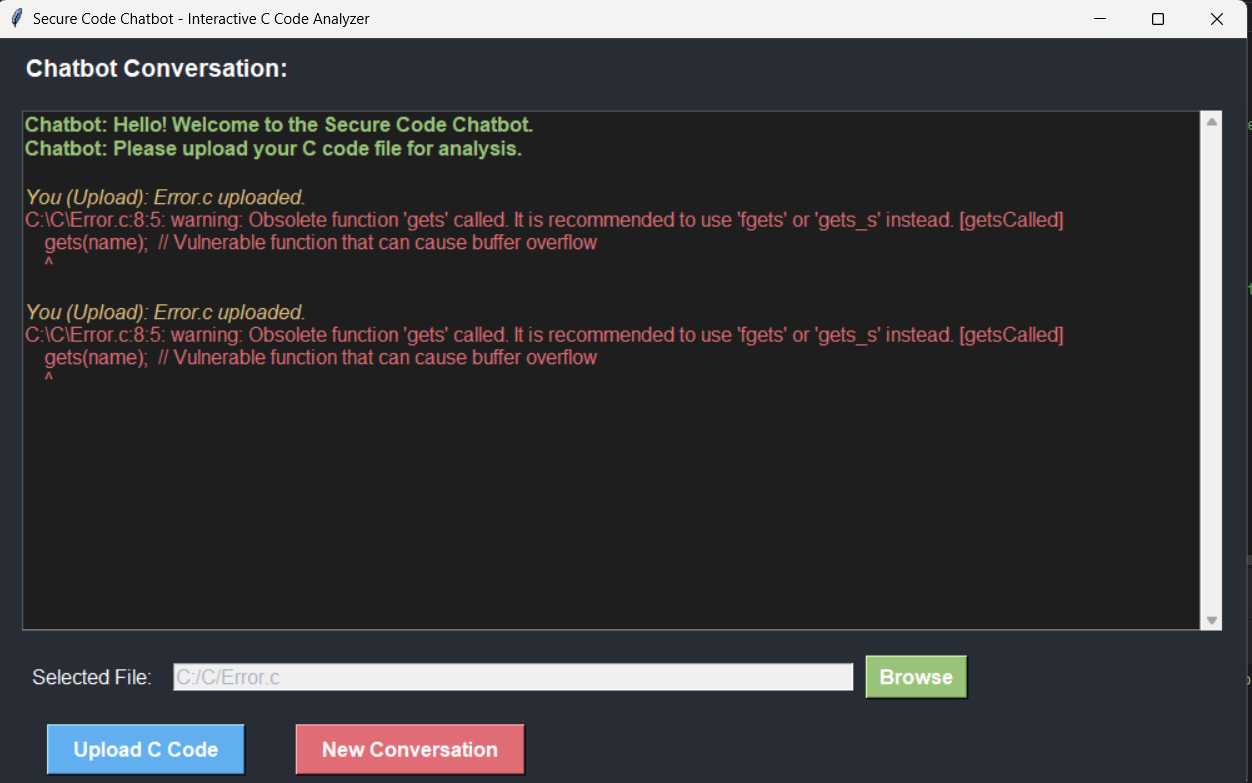
Designed the GUI using tkinter

****

6.4 Week 4**:**

Integrating static analysis tool with the GUI created





# Partial Results

Implemented the GUI using the Tkinter and working towards the execution of buffer overflow.

# Conclusion

Completed with the GUI (frontend) and gone through the vulnerabilities in the C programming, implementing static analysis tool.

**References**

1. A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code (Arvinder Kaura & Ruchikaa Nayyar)
2. A Comparison of Static Analysis Tools for Vulnerability Detection in C/C++ Code (Andrei Arusoaie, Ciobac, and Vlad Craciun.)
3. Program Dependence Learning (Aashish Yadavally and Tien N. Nguyen)
4. BegBunch – Benchmarking for C Bug Detection Tools (Cristina Cifuentes, Christian Hoermann et al.)
5. An Empirical Evaluation of GitHub Copilot’s Code Suggestion (Nhan Nguyen & Sarah Nadi ,2022)