A Project report on

Code Guardian Pro

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

GOPU CHITRA BHANU REDDY
20H51A05C4

JAKKIDI SANTHOSH REDDY
20H51A05E1

TWINKLE SHARMA
20H51A05Q3

Under the esteemed guidance of

Mr. V. NARASIMHA
Assistant Professor

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A^+ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2020- 2024

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Major Project Phase I report entitled **"Code Guardian Pro"** being submitted by Gopu Chitra Bhanu Reddy(20H51A05C4), Jakkidi Santhosh Reddy (20H51A05E1), Twinkle Sharma (20H51A05Q3) in partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out his/her under my guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

Mr. V. Narasimha
Assistant Professor
Dept. of CSE

Dr. Siva Skandha Sanagala
Associate Professor and HOD
Dept. of CSE

ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. V. Narasimha, Assistant Professor**, Department of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. Siva Skandha Sanagala,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Vijaya Kumar Koppula**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, We extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

Gopu Chitra Bhanu Reddy	20H51A05C4
Jakkidi Santhosh Reddy	20H51A05E1
Twinkle Sharma	20H51A05Q3

TABLE OF CONTENTS

List of Figures

ABSTRACT

The Code Bug Finder project is a sophisticated tool designed to automatically identify and locate bugs and errors within source code, enhancing software quality and development efficiency. Using state-of-the-art programming language parsing techniques, static code analysis, and pattern recognition algorithms, the tool efficiently scans codebases to detect both syntax and logical errors. The Code Bug Finder goes beyond conventional bug detection by providing clear and precise guidance on how to fix the identified issues. It also performs in-depth vulnerability analysis, highlighting potential security risks in the code. Not only it checks the malware in the source code but it can also check for the possible threats in websites, emails and identify the potential errors in the images. By employing the Code Bug Finder, developers can streamline the debugging process, reduce development time, and produce robust, secure, and reliable software applications with fewer defects, ultimately delivering an improved user experience.

CHAPTER 1
INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1. Problem Statement

Developing high-quality software is essential in today's technology-driven world. However, the presence of bugs, errors, and vulnerabilities in source code can significantly hinder the software development process. Traditional bug detection methods are often time-consuming and may not identify all potential issues, leading to subpar software quality and security risks. To address this challenge, there is a need for an advanced tool that automates the process of identifying and locating bugs, syntax errors, logical issues, and security vulnerabilities within source code, websites, emails, and images.

1.2. Research Objective

The research objective of the Code Bug Finder project is to revolutionize software engineering practices and cybersecurity protocols by pioneering innovative techniques for automated bug detection, error localization, and security vulnerability analysis. The research aims to advance the state of the art in programming language parsing algorithms, pattern recognition techniques, and static code analysis methods. By delving into the complexities of diverse digital content, including source code, websites, emails, and images, the research endeavors to develop algorithms capable of identifying intricate syntax errors, logical bugs, and potential security threats. A key focus lies in creating a tool that not only detects bugs but also provides developers with clear, context-specific guidance, enabling them to swiftly and accurately rectify identified issues. This research also delves into the realm of security vulnerability analysis, aiming to develop deep assessment methods that categorize vulnerabilities based on severity, ensuring the creation of software applications resistant to malicious attacks and cyber threats.

Furthermore, the research objective encompasses the design and implementation of an intuitive and interactive user interface, enhancing user experience and

facilitating seamless interpretation of scan results. Scalability and real-time analysis capabilities are critical areas of exploration, ensuring that the tool can handle vast codebases and dynamic digital environments. Ethical considerations form an integral part of this research, with the objective of developing a tool that adheres to a robust ethical framework, addressing privacy concerns, data security, and unbiased bug detection. By achieving these objectives, the research aims to significantly contribute to the development of high-quality, secure, and efficient software applications, ultimately elevating the standards of software engineering practices and bolstering cybersecurity measures on a global scale.

1.3. Project Scope and Limitations

The project is a multifaceted software development tool with a primary focus on advanced bug detection and code quality enhancement. It employs static code analysis and pattern recognition algorithms to identify syntax and logical errors, although it may not catch all runtime issues. Moreover, the project conducts in-depth vulnerability analysis to uncover potential security risks in the codebase. Its versatility extends to source code, websites, emails, and images, ensuring comprehensive error detection and security validation. The tool's core objective is to reduce development time by providing actionable insights and streamlined debugging, resulting in more efficient software development and delivering improved quality and security. It stands as an indispensable asset in the developer's toolkit.

Code GuardianPro, like any automated tool, can yield false positives and false negatives, necessitating careful developer scrutiny. It primarily focuses on static code analysis, potentially missing runtime errors and struggling with complex code or unconventional techniques. While it offers guidance, human expertise remains vital for nuanced decisions. Compatibility limitations, resource intensity, privacy concerns, and learning curves are factors to consider. Additionally, costs and licensing may challenge smaller development teams, and comprehensive security validation may require supplementary testing.

CHAPTER 2
BACKGROUND WORK

CHAPTER 2

BACKGROUND WORK

2.1. Detecting Malicious Files with YARA Rules

2.1.1. Introduction

YARA, the versatile pattern-matching tool cherished by malware researchers, has proven immensely valuable in detecting suspicious files on individual endpoints. However, there is a notable scarcity of publicly available information on harnessing this powerful tool for network-based file scanning. In this paper, we illuminate a novel approach that combines the open-source Zeek IDS (formerly known as Bro) with a custom script to extract files from network traffic and pinpoint potential threats at an early stage, before they can inflict substantial damage.

The advantage of network-level YARA scanning is evident in its efficiency, markedly contrasting with the laborious task of scanning numerous gigabytes or terabytes of data on endpoint devices. Moreover, this approach enables precise targeting of specific MIME types frequently exploited for malware delivery, significantly enhancing the network's security posture.

The synergy between YARA and Zeek IDS introduces a substantial boon to network defense. When a YARA rule is triggered, Zeek can provide crucial contextual information, empowering defenders to respond more swiftly and effectively. This context includes details about the network traffic, the source, destination, and associated metadata, all of which equip security teams with the insight needed to make well-informed decisions in real-time.

In essence, the integration of YARA with Zeek IDS offers an advanced layer of defense against evolving cyber threats, facilitating early threat detection, network-wide file inspection, and rapid, context-rich incident response. This synergy is an indispensable asset for organizations seeking to bolster their cybersecurity efforts in an increasingly complex and dynamic digital landscape.

2.1.2. Merits, Demerits and Challenges

Merits:

- Integrating YARA rules with Zeek provides enhanced visibility into network traffic. Zeek's network analysis capabilities combined with YARA's file pattern recognition allow for a comprehensive view of the network, helping to identify potentially malicious files.

- Zeek provides contextual information about network behavior. Integrating YARA rules with Zeek allows security professionals to analyze files in the context of network sessions, aiding in understanding the behavior of potentially malicious files within the network.

- By integrating YARA rules with Zeek, the accuracy of detection is improved. Zeek's metadata about network traffic can be used to create more precise and context-aware YARA rules, reducing false positives and enhancing the efficiency of the detection process.

- Zeek operates at the network level, allowing YARA rules to scan files across the entire network. This network-wide approach ensures that potentially malicious files are detected regardless of which specific endpoint or device they are targeting.

- The integration of YARA rules with Zeek enables real-time analysis of files as they traverse the network. This real-time analysis is crucial for quickly identifying and responding to threats, preventing potential damage before it occurs.

- Zeek's integration with YARA rules facilitates effective incident response. Security teams can swiftly identify and isolate affected systems, analyze the scope of the incident, and respond promptly to mitigate the threat.

Demerits and Challenges:

- Integrating YARA rules with Zeek requires a complex configuration process. Writing and optimizing YARA rules to work seamlessly with Zeek's output can be challenging and may require a significant investment of time and expertise.
- The integration of YARA rules with Zeek can be resource-intensive, especially in high-traffic networks. Proper hardware resources and system optimization are required to ensure that the integration does not significantly impact network performance.
- Similar to standalone YARA implementations, the integration with Zeek is limited to detecting known malware signatures. It may struggle with detecting new, previously unseen threats or sophisticated, polymorphic malware that constantly changes its code to evade detection.
- Encrypted network traffic continues to pose a challenge for YARA-based detection, even when integrated with Zeek. As the content of encrypted files is not readily visible, YARA rules might not be effective in detecting malicious payloads within encrypted connections.
- Like any signature-based system, the integrated solution requires constant maintenance. YARA rules must be updated and refined to adapt to evolving threats, which demands ongoing effort and attention from security professionals.
- Despite contextual analysis provided by Zeek, false positives remain a challenge. Overly aggressive YARA rules might trigger false alarms, leading to unnecessary investigations and potentially diverting resources from more critical security tasks.

2.1.3. Implementation

The Zeek sensor is the first part; as previously said, it will extract data to a designated location, generate network logs, and receive traffic for review. The second part is a

corn job that will execute a custom Python script I wrote called zeekYaraAlert.py. It will receive as input all of the enabled YARA rules. In essence, it will combine every YARA rule in the YARA rule folder into a single file, which will be utilized to scan every file Zeek extracts and stores in the extracted files folder. YARA regulations shouldn't be repeated. The file that triggered any YARA rule will be copied to an Alerted files folder. If there is a file that has been flagged, the file id will be used to search other Zeek log files for additional context, including the involved IP address and application level details like HTTP URL, user agent, and HTTP method. If the file is smaller than the 10 MB file limit that the script sets, it will send an email alert with the file sample that triggered the YARA rule zipped with the password "infected." Lastly, the script will execute continuously for the duration of its default one-minute runtime, erasing every file in the extracted files folder. Compared to doing a thorough scan of a YARA rule on an endpoint hard disk, the scan will be comparatively quick because only a small number of files are extracted each minute.

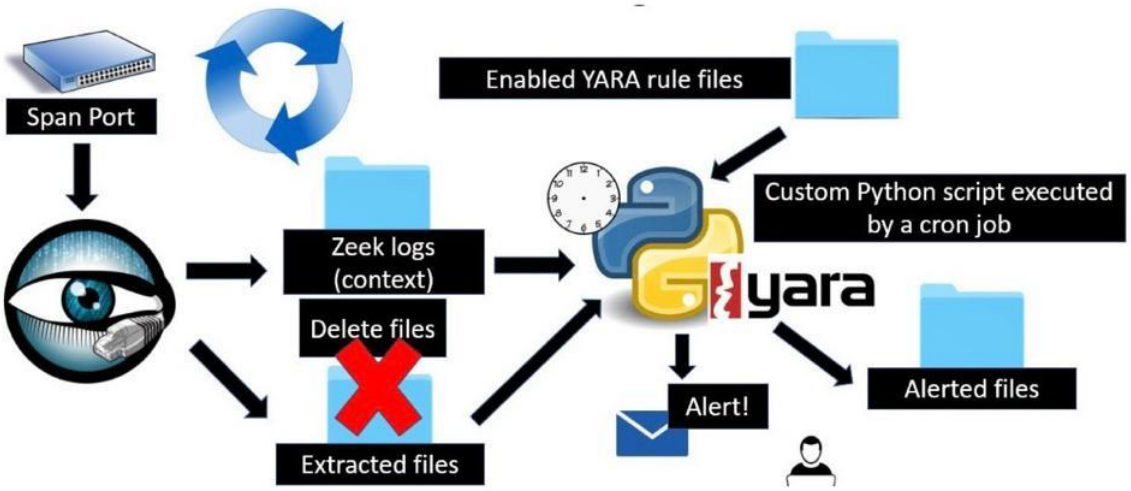

Figure 2.1.3: Implementation of Detecting Malicious Files with YARA Rules.

2.2. The Ghost in The Browser Analysis of Web-based Malware

2.2.1. Introduction

Web-based malware infections have become increasingly prevalent due to the ease of setting up and deploying websites. However, a critical challenge remains in keeping the required software up to date with patches, a task that typically demands human intervention. The growing number of applications necessary for running modern web portals, coupled with the frequency of patch releases, has made maintaining website security a daunting and often overlooked task.

In response to this problem and with the aim of safeguarding users from web-based infections, an initiative has been launched to identify potentially malicious web pages across the vast expanse of the internet. Notably, Google's web crawling capabilities already encompass billions of web pages. To efficiently sift through this extensive repository, simple heuristics are applied to identify pages attempting to exploit web browsers. These heuristics significantly reduce the number of URLs subjected to further analysis. Pages classified as potentially malicious serve as inputs for instrumented browser instances running in virtual environments.

The overarching objective is to observe the behavior of malware when users visit these identified malicious URLs and determine if malware binaries are being downloaded as a result. Websites verified as malicious through this rigorous process are marked as potentially harmful when returned as search results. This labeling empowers users to avoid exposure to such sites, leading to a reduction in the number of individuals falling victim to web-based infections, thereby enhancing web security and user safety.

2.2.2. Merits, Demerits and Challenges

Merits:

- By identifying potentially malicious web pages, users are provided with a safer browsing experience. They can avoid these marked sites, reducing the risk of malware infections.

- This effort reflects a proactive stance in dealing with web-based malware. By identifying and labeling potentially harmful sites, preventative measures can be taken before users unknowingly stumble upon these sites.

- Leveraging Google's extensive web crawling capabilities allows for a comprehensive analysis of a vast number of web pages. This large-scale analysis can provide valuable insights into the patterns and techniques used by malicious websites.

- Keeping detailed statistics and tracking identified malware binaries enables data-driven decision-making processes. This data can be used to analyze trends, identify common vulnerabilities, and improve overall web security strategies.

- By marking malicious sites in search results, the wider internet community benefits. Fewer users being infected means less malware propagation, ultimately making the internet a safer environment for all users.

Demerits and Challenges:

- Despite Google's extensive crawling efforts, it's nearly impossible to cover every corner of the internet. There will always be new, undiscovered malicious websites that might evade this detection system.

- The heuristics used for identifying potentially malicious sites might produce false positives (marking safe sites as harmful) or false negatives (missing actual malicious sites). These errors can lead to either unnecessary warnings or undetected threats.

- Keeping software up to date with patches still relies on human intervention. As long as this remains a manual task, there is always a risk of outdated software being exploited, regardless of efforts to identify malicious sites.

- Malicious actors continuously evolve their techniques. As new methods of obfuscation and exploitation emerge, the detection mechanisms need to adapt. This creates an ongoing challenge in keeping up with the evolving landscape of web-based malware.

- The detailed tracking and analysis of web pages, even for security purposes, raise concerns about user privacy. Striking a balance between enhancing security and respecting user privacy is a constant challenge.

- The process of instrumenting browser instances, running virtual machines, and analyzing malware behavior is resource-intensive. It requires significant computing power and storage capacity, which might be a limitation for smaller organizations with limited resources.

- Determining the legality and ethics of labeling websites as potentially harmful requires careful consideration. False accusations could lead to legal challenges, and ethical concerns arise when deciding which sites should be labeled, raising questions about censorship and freedom of expression.

2.2.3. Implementation

In the initial phase of our approach, we leverage the power of MapReduce to process the vast array of web pages we've crawled, focusing on identifying properties that could indicate potential exploits. The MapReduce model operates in two main stages: first, the Map stage takes input as key-value pairs and produces intermediate key-value pairs as output. Here, we extract the URLs of analyzed web pages as keys and all links pointing to potential exploit URLs as values. This involves parsing HTML to identify elements known to be malicious, like iframes pointing to hosts distributing malware, allowing us to identify a significant portion of malicious web pages. For cases not covered by these known indicators, we scrutinize the interpreted JavaScript code embedded in each page. We detect malicious content based on abnormalities such as heavy obfuscation, common in exploit attempts. In the Reduce stage, we retain only the first intermediate value, significantly narrowing down the number of URLs from several billion to a few million. Further refinement is achieved by sampling on a per-site basis, implemented as another MapReduce process.

To confirm whether a URL is genuinely the source of a web browser exploit, we instrument Internet Explorer within a virtual machine. We instruct the browser to navigate to each potential URL and meticulously record all HTTP fetches, alongside virtual machine state changes, including newly initiated processes, registry modifications, and file system alterations. Each component of the analysis is assigned an individual score; for instance, we assess each HTTP fetch using various anti-virus engines. The overall score for an analysis run is the sum of these individual scores. If we detect the initiation of new processes on the machine after visiting a web page, it strongly suggests a drive-by download occurrence. To gather additional cues for detecting drive-by downloads, we closely monitor changes in the file system and registry.

To identify which search results should be flagged as potentially harmful, we aggregate the URL analysis on a per-site basis. If a significant portion of URLs from a particular site exhibits malicious traits, the entire site or specific path components of the site may be marked as harmful

when displayed as search results.as harmful when displayed as search results.

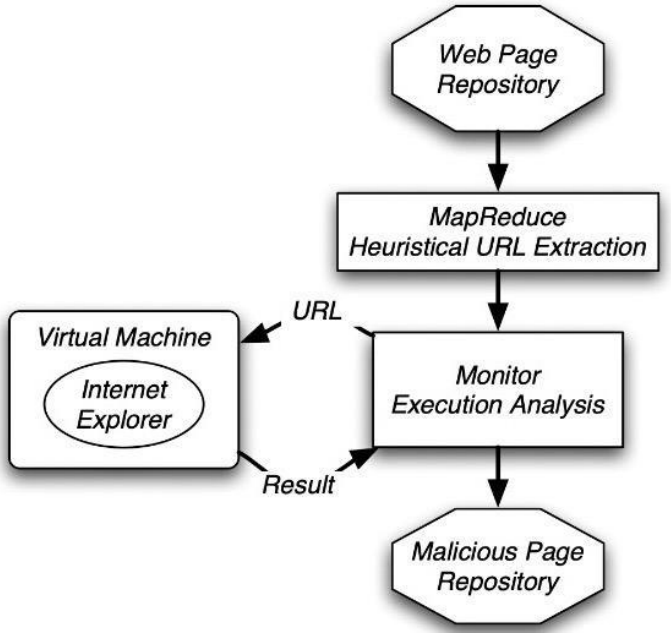

Figure 2.2.3.1: Implementation of The Ghost In The Browser Analysis of Web-based Malware

2.3 File print analysis for Malware Detection:

2.3.1 Introduction

The limitations of commercial off-the-shelf (COTS) antivirus systems in detecting malware that is embedded within otherwise normal files. While traditional antivirus solutions rely on signature-based analysis, attackers have found various ways to embed malicious code in files, making it difficult for these systems to detect such hidden threats. Even with access to the content, COTS antivirus scanners often do not perform deep scans of all files, leaving room for malware to remain undetected. The paper highlights the ineffectiveness of signature-based scanners in detecting new and hidden "zero-day" malicious exploit code, emphasizing the need for alternative detection methods.

The use of statistical content analysis to detect anomalous file segments indicating potential malware infection. By exploring this approach, they aim to develop an efficient method for identifying infected files, especially in the context of online network communication, file sharing, media streaming, and scanning large data warehouses. The research's significant contribution lies in the revelation that antivirus systems can be easily deceived even when provided with malware signatures. The experiments conducted involve inserting known malware into normal PDF or DOC files. Surprisingly, these manipulated files, containing hidden malcode, managed to evade detection by popular COTS antivirus scanners and were successfully opened by applications like Adobe and Word. This finding highlights the inherent vulnerabilities in file formats and application logic, providing attackers with a means to infect hosts with seemingly innocent-looking files, rendering traditional detection techniques ineffective in many cases.

2.3.2. Merits, Demerits and Challenges

Merits:

- The experiments simulate real-world scenarios, providing valuable insights into how malware can be stealthily embedded in different file types, including PDFs and Windows executables. This realistic approach helps researchers and security professionals understand potential vulnerabilities.
- The experiments identify specific weaknesses in popular software, such as Adobe Acrobat and WINWORD.EXE, shedding light on how these applications handle files with embedded malware. This knowledge is crucial for developers and security experts aiming to improve software security.
- The experiments are conducted methodically, systematically testing different insertion points within files and analyzing the behavior of commercial antivirus systems. This systematic approach ensures a comprehensive understanding of the vulnerabilities and

of existing security measures.

- The experiments employ innovative techniques, such as leveraging zeroed portions of executables, to hide malcode. By exploring unconventional methods, researchers gain a deeper understanding of potential avenues for malware insertion, which can inform the development of more robust security solutions.

Demerits and Challenges:

- Conducting experiments involving malware raises ethical and legal concerns, especially if not done in a controlled environment. Malware handling and dissemination must adhere to strict ethical guidelines and legal regulations to avoid unintended consequences.
- The experiments focus on specific file types (PDFs, DOC files, and Windows executables) and specific applications (Adobe Acrobat, WINWORD.EXE). While these findings are valuable, they might not cover the entire spectrum of potential vulnerabilities present in other file formats or software applications.
- The effectiveness of malware insertion techniques might be dependent on specific versions of software applications. Updates or patches released by software vendors could potentially mitigate the vulnerabilities identified in the experiments, making the findings time-sensitive and context-specific.
- While the experiments aim to enhance security awareness, the detailed information provided could potentially be misused by malicious actors to refine their techniques for evading detection. Striking a balance between knowledge dissemination and security concerns is crucial.
- The experiments highlight vulnerabilities but might not provide comprehensive mitigation strategies. Understanding the problem is essential, but the absence of robust solutions in the discussions could leave readers without clear guidance on how to address these issues effectively.

2.3.3 Implementation

In the study, a collection of malware was tested to verify if they could be detected by a commercial off-the-shelf (COTS) antivirus system. Each malware sample was embedded both at the head and tail of normal PDF files. Surprisingly, the COTS antivirus system had a low detection rate for these infected files, especially when malware was attached at the tail. Even more concerning was the fact that many of these infected files, which went undetected, could be opened without error using Adobe Acrobat. This suggests that malware can be easily hidden inside PDF files without being noticed. The reason behind this is that Adobe Reader scans the head of a file for specific "magic numbers" indicating the beginning header metadata needed to interpret the rest of the content. Consequently, the portions of the file passed over by the reader during this search provide a convenient hiding place for malcode.

In a separate experiment, malware was inserted into random positions within PDF files. However, due to PDF's specific encoding, blind insertion in the middle often disrupted the encoding, making it easily detectable by Acrobat Reader upon opening. This approach was unsuccessful because the altered encoding triggered errors. In contrast, appending malware to the head or tail of the PDF file proved more effective as it did not disrupt the encoding, allowing the file to appear normal and evade detection by the reader. The experiment was also repeated with DOC files using selected malwares, yielding similar results.

Researchers focused on Windows executables, such as WINWORD.EXE. They observed that these executables had dominant zero byte values, a result of standard block alignment and padding (at addresses n*4096) for efficient disk loading. Leveraging these zeroed portions, the researchers inserted hidden malcode into a continuous block of zeros large enough to accommodate the entire malware. This manipulated executable was then stored back on disk. Surprisingly, a commercial off-the-shelf (COTS) antivirus scanner failed to detect these poisoned applications. The experiment involved replacing the padded segments of WINWORD.EXE, specifically from byte positions 2079784-2079848. Despite the alteration, both versions of the application - the normal executable and the infected one - were able to open DOC files without any issues.

CHAPTER 3
RESULTS AND DISCUSSION

CHAPTER 3

RESULTS AND DISCUSSION

3.1 Results & Discussion

The Code GuardianPro project represents a significant leap in the field of bug detection and software quality assurance. Its multifaceted approach, utilizing cutting-edge programming language parsing techniques, static code analysis, and pattern recognition algorithms, distinguishes it from existing tools. While traditional bug detection tools primarily focus on identifying syntax errors, Code GuardianPro delves deeper, detecting not only syntax but also complex logical errors. This comprehensive methodology ensures a thorough analysis of the codebase, leading to more robust and error-free software.

A key feature of Code GuardianPro lies in its proactive bug resolution approach, offering precise guidance to developers on fixing identified issues. This sets it apart from many existing tools that require developers to invest additional time diagnosing problems and finding solutions. By providing actionable insights, the project significantly reduces debugging time, enhancing development efficiency. Additionally, the tool's in-depth vulnerability analysis, covering both bugs and security risks, adds an essential layer of protection to software applications. Its versatility in analyzing various digital entities, including websites, emails, and images, gives developers a holistic view of potential risks, enabling them to address vulnerabilities effectively across multiple mediums. Code GuardianPro stands as a powerful tool in ensuring software reliability, security, and efficiency.

CHAPTER 4
CONCLUSION

CHAPTER 4
CONCLUSION

In conclusion, Code GuardianPro signifies a significant advancement in software development. It tackles bug detection and code quality improvement using innovative techniques like language parsing, static analysis, and pattern recognition. What sets it apart is its adaptability, spanning source code, websites, emails, and images for comprehensive error detection and security validation. This versatility accelerates development, empowering developers with actionable insights. In an era of digital vulnerabilities, it safeguards software integrity by detecting syntax, logic errors, and security threats. Code GuardianPro enables a shift from troubleshooting to innovation, resulting in defect-free, secure software that delivers exceptional user experiences. This project ushers in a new era where software development embodies quality, security, and user satisfaction.

CHAPTER 5
REFERENCES

REFERENCES

References

- **For Static Code Analysis and Code Quality Enhancement:**
- SonarQube: https://www.sonarqube.org/
- Checkmarx: https://www.checkmarx.com/
- **For Web Security and Malware Detection:**
- OWASP (Open Web Application Security Project): https://owasp.org/
- Malwarebytes: https://www.malwarebytes.com/
- https://ieeexplore.ieee.org/xpl/conhome/7166062/proceeding