

Enhancing Web Security using Client-Side Web Assembly based Web Application Firewall (WAF)



MCS

by

Umar Mahboob

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

(Nov 2023)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS thesis written by Mr. **Umar Mahboob**, Registration No. **00000397964**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/ Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.


Signature:  _____

Name of Supervisor Dr. Waleed Bin Shahid

Date: 22 Nov 23 _____

Signature (HoD):  _____

Date: 24/11/23 _____
HoD
Information Security
Military College of Sigs

Signature (Dean/ Principal):  _____

Date: 27/11/23 _____
Brig
Dean, MCS (NUST)
(Asif Masood, Phd)

Declaration

I hereby declare that no portion of the work presented in this thesis has been submitted in support of another ward or qualification either at this institution or elsewhere.

Signature: _____

Umar Mahboob

MS Student

Dedication

To my parents, whose love, encouragement, and sacrifices have made all my achievements possible. Your unwavering belief in me has been a constant source of inspiration. This thesis is dedicated to the people who have been my guiding lights and unwavering support throughout my academic pursuit.

To my dear wife and child, for their understanding, patience, and unending encouragement. Their love and support have been my rock during the challenging times of this journey.

To my mentor and advisor, Dr. Waleed Bin Shahid, for his wisdom, guidance, and faith in my abilities. His mentorship has shaped me into a better researcher.

To my friends, who have supported me and brought me joy during my academic journey, thank you. Your friendship has made this experience truly unforgettable.

Acknowledgements

Above all, I am extremely grateful to Almighty Allah, the Most Gracious and the Most Merciful, for granting me health, knowledge, wisdom, and the ability to communicate naturally. I owe my supervisor a great deal for his invaluable supervision, encouragement, and direction, which allowed me to work on this project and finish my research in this field. I sincerely appreciate my peers' help, notably that of Awais Bin Riaz, whose assistance was invaluable to my research. I also want to express my gratitude to my parents, wife, and child for their support, encouragement, love, and devotion in getting me through my challenges and finishing my research work.

Abstract

JavaScript has been a popular and the most widely used language for web applications. However, it has some limitations especially related to performance while running computationally intensive tasks which hinders usage of such applications through the web. To resolve the issue of performance, a new low-level assembly-like language Web Assembly “also referred to as WASM” has been developed to run in the browser and to complement the usage of JavaScript rather than replacing it. WASM is designed keeping the security features in mind. However, being a new technology, it still has some security flaws which can be exploited to compromise different applications. Mining cryptocurrency is a lucrative opportunity due to its increased usage. One of the illegitimate ways of mining is through deploying cryptojacking malware within web browsers. Web Assembly has provided malicious actors with a new avenue for utilizing cryptojacking malware given its performance gains. This resulted in development of different systems for detection of Wasm-based cryptojacking, using both static and dynamic analysis. In this paper, we provide an overview of Web Assembly (WASM) and a comprehensive review of different cryptojacking detection techniques. Furthermore, we propose a novel framework which is based on AI-driven WebAssembly analysis engine designed to detect WebAssembly-based cryptojacking attacks. Our evaluation of the framework shows an accuracy rate of 98.5% with only 0.78% FN rate in detecting cryptojacking WASM applications. In the end, we carried out a comparative analysis of our proposed framework with two malware detection tools: VirusTotal and Malwarebytes.

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation	6
1.3	Problem Statement	6
1.4	Research Objectives	7
1.5	Contributions	7
1.6	Thesis Outline	8
2	Literature Review	9
2.1	JavaScript	9
2.1.1	Usage of JavaScript	9
2.1.2	Advantages of JavaScript	10
2.1.3	Dis-Advantages of JavaScript	10
2.2	WebAssembly	11
2.3	Cryptocurrency	12
2.4	Cryptojacking	13
2.4.1	Coinhive and Monero	14
2.4.2	CryptoNight Algorithm	15
2.4.3	Cryptojacking through Web Applications	16
2.5	Analysis (Static vs Dynamic analysis)	17
2.5.1	Static Analysis	17

2.5.2	Dynamic Analysis	17
2.6	Review of WebAssembly	17
2.7	Work Related to Cryptojacking	19
3	Methodology	22
3.1	Intrinsic Characteristics of Cryptojacking Malware	22
3.1.1	Semantic Signature Matching	22
3.1.2	Strings & Pattern Matching	24
3.1.3	Call Flow Graph	25
3.1.4	Cryptographic Primitives	25
3.2	Architecture of Proposed Framework	29
3.2.1	Wasm Binary Collector	29
3.2.2	Wasm to WAT conversion	30
3.2.3	Processing and Static Analysis	30
3.2.4	Detection Classifier	31
3.2.5	Alertify	33
4	Result And Performance Evaluation	35
4.1	Preparation of Dataset	35
4.2	Selection of Machine Learning Classifier	36
4.2.1	Linear Support Vector Machine (SVM)	36
4.2.2	Random Forest (RF)	36
4.3	Results	37
4.3.1	Application of Linear Support Vector Machine (SVM)	40
4.3.2	Application of Random Forest (RF)	42
4.4	Comparative Analysis	47
4.4.1	Comparison with Other Tools	48
4.4.2	Comparison with VirusTotal & Malware Bytes	49

CONTENTS

5	Discussion and Future Work	53
5.1	Discussion	53
5.1.1	Strengths	53
5.1.2	Limitations	54
5.2	Future Work	54
6	Conclusion	55

List of Figures

1.1	Web Assembly Working	2
1.2	How Browser-based Cryotojacking Works	3
2.1	Working of Blockchain Technology	13
3.1	Comparison of Instructions between Crypto and Benign WASM	23
3.2	Word Cloud of Strings and Keywords	24
3.3	Call Flow Graph - Standard CryptoNight	26
3.4	Call Flow Graph - Obfuscated CryptoNight Implementation	26
3.5	Call Flow Graph - Benign Application	27
3.6	Crypto Primitives Fingerprinting in WASM Binaries	28
3.7	Overview of Framework	29
3.8	Detailed Work Flow of Framework	34
4.1	Performance Metrics of Linear SVM (80-20 Split)	41
4.2	Confusion Matrix – Linear SVM (80-20 Split)	42
4.3	Performance Metrics of Linear SVM (70-30 Split)	42
4.4	Confusion Matrix – Linear SVM (70-30 Split)	43
4.5	Performance Metrics of Random Forest (80-20 Split)	44
4.6	Confusion Matrix – Random Forest (80-20 Split)	44
4.7	Performance Metrics of Random Forest (70-30 Split)	45
4.8	Confusion Matrix – Random Forest (70-30 Split)	46

LIST OF FIGURES

4.9 Accuracy Achieved in Applying Random Forest Model	46
4.10 Loss Observed in Applying Random Forest Model	47
4.11 Result of VirusTotal (False Negatives)	51
4.12 Result of MalwareBytes-1 (False Negatives)	51
4.13 Result of MalwareBytes-2 (False Negatives)	52

List of Tables

2.1	Comparison of Static and Dynamic Analysis	18
2.2	Summary of WebAssembly-Based Cryptojacking Detection Tools	20
4.1	Comparison of Characteristics	48
4.2	Comparison of Results	49

CHAPTER 1

Introduction

1.1 Background

JavaScript is the most widely adopted web language (98.3% of all Web) since its creation in 1995. It is an interpreted, high-level language used to create dynamic and interactive web applications. It is preferred by developers because of its compatibility, versatility, large ecosystem and ease of use. While being popular, JavaScript has some limitations [1]:-

- Being an interpreted language, it is slower in performing computationally intensive tasks such as Virtual and Augmented Reality, 3D games, image/video editing and other domains requiring high performance.
- JavaScript can have compatibility issues as it can behave differently in different browsers.
- Debugging complex JavaScript applications is challenging.
- It can be vulnerable to security threats. Since the code is visible to users, it can be used for malicious purposes.
- Codes written in other programming languages such as C, C++, or Rust can provide better performance and more control over system resources than JavaScript but cannot be executed directly in a web browser. Also, integration of JavaScript with other programming languages is difficult.

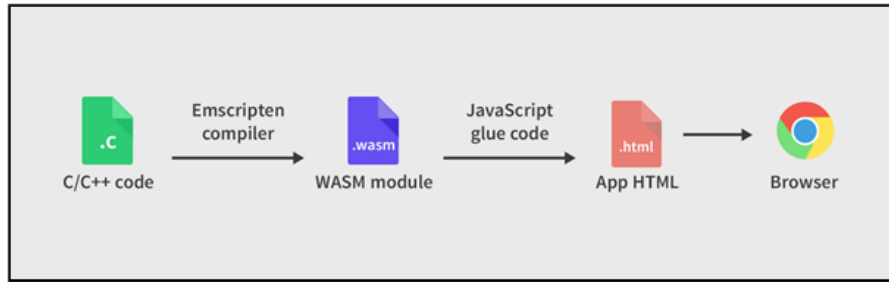


Figure 1.1: Web Assembly Working

Web Assembly, also known as "WASM" provides a solution to all these problems. WASM is the fourth and only precompiled language to be run in web browsers after HTML, JavaScript, and CSS. It is like an assembly language with a compact binary format providing many new features and high performance gains by running at near native speed [2]. All of the main web browsers, including Microsoft Edge, Mozilla Firefox, Google Chrome, Apple Safari, and Chrome; mobile browsers, such as Chrome, iOS Safari, and Firefox for Android, support this open standard, which was created by the World Wide Web Consortium (W3C) [3] [4] [5]. Web Assembly is becoming increasingly popular and is used by many websites and web applications such as Figma, Google Earth, AutoCAD Web, WhatsApp, Dropbox and many more.

WASM is not meant to be written directly rather it is as a compilation target for high-level languages like C/C++, C# [6], and Rust; thus, developers can work with their preferred language without having to learn a new one. It is also not created to replace JavaScript but runs alongside allowing both to work together. Web Assembly can also run outside the browser environment such as mobile agents, server-side applications and IoT devices using Web Assembly System Interface (WASI), a WASM code interpreter outside the browser, such as WASmtime and WAMR (Web Assembly Micro Runtime) [7]. Some of the features of WASM include [6]:-

- A portable language that is fast, efficient, and compact.
- Human-readable and debuggable so that the code can be written, viewed, and debugged.
- Keeping the security in mind (running in a safe, sandboxed execution environment).
- Works well with other existing web technologies.

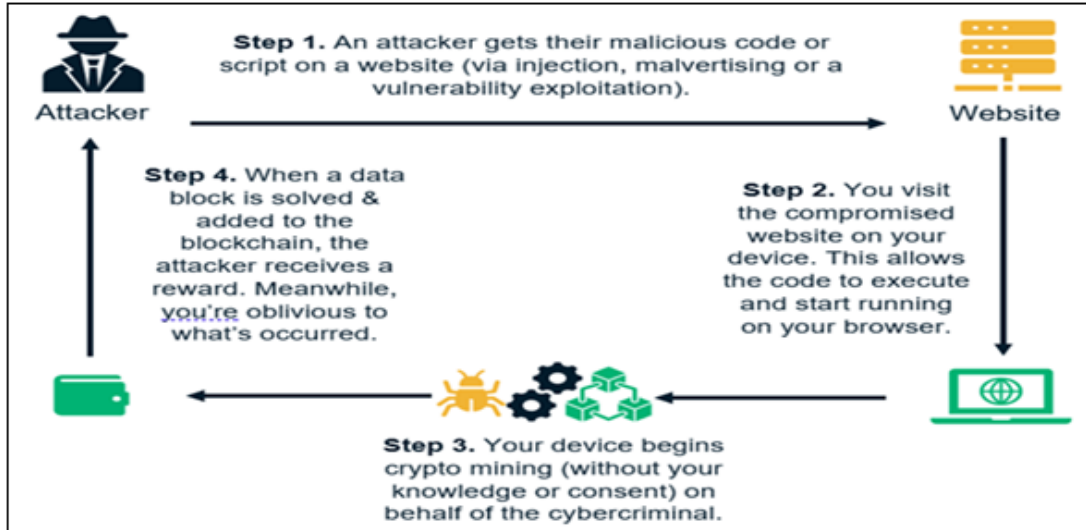


Figure 1.2: How Browser-based Cryptojacking Works

WASM has been developed with the particular focus of making it secure. The security model of WASM is centred around (1) protecting users from applications which are malicious or have bugs (2) equipping developers with useful tools to develop safe applications and mitigating such issues. WASM provides the following features [8] [9] which comply with the security model:-

- WASM applications run in a secure and sandboxed environment, which means that they cannot directly access other system resources. This protects against security vulnerabilities and prevents malicious code from accessing sensitive information.
- The WASM compiler creates a Control Flow Graph (CFG) during compilation which guarantees the Control-Flow integrity at runtime for all types of function calls. Therefore, the application will fail if it calls a function that is not expected.
- Developers can define the resources including memory, network and files that are required to be accessed by a module. This restricts the capabilities of a module preventing it from accessing unauthorized resources.
- Developers can define the resources including memory, network and files that are required to be accessed by a module. This restricts the capabilities of a module preventing it from accessing unauthorized resources.
- Indirect function calls are checked at runtime to make sure the correct function is

called. It is not possible to overwrite a return pointer, so the call stack is protected and invulnerable to buffer overflow which ensures safe function returns.

- WASM only accesses a linear memory implemented in managed buffers in a limited area where all reading and writing of data is done. Bounds of accessing the linear memory are checked at the regional level.
- Traps are generated in case of any abnormal behaviour which are used to end the application execution and intimate execution environment about the issue.
- Being a statically typed language, any type of errors are caught at compile time rather than runtime reducing the risk of vulnerabilities such as buffer overflows and memory leaks.

WASM also has some potential security issues, given its early days of inception:-

- Even though WASM runs in a sandboxed environment and has access to its linear memory, still supports certain type-unsafe language i.e. WASM binaries suffer from different memory bugs like buffer overflows due to C/C++ [10].
- Malicious WASM code/ binaries generated from malicious high-level code in the first place to steal resources i.e. cryptojackers (cryptomining malware) [10] [11], which is further complicated by the use of WASM code obfuscation techniques.
- There are no inherent ways to check the integrity of WASM binaries [12] which could result in the execution of tampered or corrupted WASM applications if remain unchecked.

Since the Bitcoin white paper published by Satoshi Nakamoto [13], cryptocurrencies have revolutionized the modern economy by offering alternate ways for financial transactions and investments. Major cryptocurrencies like Bitcoin require special hardware for efficiently solving hash puzzles and minting cryptocurrency. Monero (XMR) cryptocurrency, launched in 2014, was designed to provide enhanced privacy and anonymity for its users with the promise of untraceable transactions [14]. Monero is based on the CryptoNight hashing algorithm. The algorithm allows cryptomining to run on limited resources of CPU and GPU instead of using specially designed ASICs. Coinhive is a cryptocurrency mining service that relies on a small chunk of code designed to be installed on websites [15]. It provided Web Assembly/JavaScript-based code that websites

could incorporate to make visitors' machines mine Monero. Like with any other emerging technology, the increase in usage and value of cryptocurrencies has also attracted malicious actors to illicit gains through cryptojacking. Cryptojacking, also sometimes referred to as drive-by or malicious cryptomining, involves unauthorized and stealthy use of computing resources to mine cryptocurrencies, often without the permission or knowledge of the users as most cryptojacking software is designed to stay hidden from users. WebAssembly (Wasm) has provided a new platform for cryptojacking attacks, enabling adversaries to deploy stealthy and resource-efficient use of victim resources [16] for mining operations within web browsers especially due to its performance gains over traditional JavaScript as it allows execution of code at near-native speed. Therefore, Wasm-based mining is the most used technique for cryptojacking attacks. These malicious activities consume substantial processing resources of users' devices leading to reduced system performance, increased electrical consumption and potential financial losses for victims. Fig.2 explains a working mechanism of browser-based cryptojacking. Similarly, Coinhive's code was quickly abused: mining script is frequently used by malware authors to mine Monero using the user's CPU resources without their knowledge and consent [17] [18]. This misuse of Coinhive scripts by malicious entities led to its closure in March 2019 [19]. Even though it is not being maintained, cryptojacking malware is still in use.

The potential for exploiting Wasm for illegal crypto mining continues to grow with the increase in its adoption for web application development. Detecting and mitigating these attacks is essential for preserving user trust and confidence in web browsing experiences. This has led to the development of various tools and mechanisms which apply different techniques for the detection of cryptojacking. In this research we have carried out a comprehensive review of existing cryptojacking detection techniques that particularly focus on WebAssembly, duly highlighting the gaps and challenges posed by them. Details of these mechanisms are discussed in Section 2. Furthermore, we introduce a novel framework; an AI-driven WebAssembly analysis engine designed to detect WebAssembly-based cryptojacking. The core of the proposed framework is based on static analysis of Wasm binaries that distinguish malicious cryptojacking from benign ones.

1.2 Motivation

Research is being carried out to find innovative and accurate techniques for detection of malicious cryptojacking WASM binaries. Some tools exist for the detection of same however, they face certain challenges.

- Many techniques are prone to code obfuscation techniques.
- Tools do not cover all intrinsic characteristics of Wasm binaries which may result in false negative results.
- Many of the proposed solutions tend to suffer from more computational overhead and require administrator privileges to run.
- Keeping in view the challenges posed by previous techniques, there exists a considerable margin for improvement in Web Assembly based cryptojacking detection.

1.3 Problem Statement

Cryptojacking malware is an ever-evolving field. The adversary can take different actions such as code obfuscation, changing strings and function names [16], using encrypted communication [20] etc to evade any detection mechanisms which makes their detection even more difficult. Static code analysis depending on a single detection mechanism has been prone to code obfuscation techniques [16]. Two more research works have investigated the possibility of evading detection techniques by cryptojacking WebAssembly. Bhansali et al. [11], studied different source code obfuscation techniques for both benign and malicious (cryptojacking malware) WASM applications, compiled from C language only, which were found to be very effective. They fed both un-obfuscated and obfuscated versions of applications to “MINOS” which showed the least promising results in detecting obfuscated applications, having an 18.75% success rate as well as producing 7.7% false positives. J. Arteaga et al. [21], carried out automatic binary diversification and byte code transformations at Wasm level using WASM-mutate of 33 WebAssembly cryptojacking binaries to evade cryptojacking detectors. The evasion technique was evaluated against two malware detectors: VirusTotal and MINOS. The variants of WASM cryptojacking were able to evade in 90% of cases for VirusTotal and 100% for MINOS. Both the research works showed promising results and that detection evasion is very

much possible through source code obfuscation or byte code level diversification. Dynamic analysis-based detection systems usually tend to suffer from more computational overhead, require administrator privileges and more time to run [20]. They also have less accuracy due to interference from other running processes and comparatively higher false positive rates. Also, dynamic analysis running may affect user experience. Similarly, a lot of other mechanisms used only a single technique for the detection of malicious Wasm binaries which can sometimes be evaded and may generate false negative results.

1.4 Research Objectives

The main objectives unfolding our research comprise of following objectives:-

- Performing thorough survey of Web Assembly (WASM) by analyzing its advantages, weaknesses, and security issues.
- A framework for detection of Web Assembly based cryptojacking using static analysis.
- Comparative analysis of the proposed framework in relation to other tools to measure its efficacy.

1.5 Contributions

The principal contributions of the research work are enumerated below:-

- A novel unified framework has been proposed which combines static analysis and AI for detecting Wasm-based cryptojacking.
- Preparation of own dataset using WASM binaries collected from open source which is subsequently used to train and evaluate the framework for detection accuracy.
- Address the gaps in existing tools as well as improve the accuracy of detecting malicious Wasm binaries through the proposed framework.
- The proposed framework gives a detection accuracy of 98.5% with only 0.78% FN rate.

- A comparative analysis of the proposed framework with other tools and most specifically two malware detection tools: VirusTotal and Malwarebytes.

1.6 Thesis Outline

The following chapters comprise the distribution and organization of the research work:-

- **Chapter 1:** A brief introduction is mentioned, followed by the problem statement. Then motivation/reason for doing this research is mentioned. Research objectives are highlighted and in the end, contributions we intend to make through this research are duly listed.
- **Chapter 2:** Presents a brief literature review of web assembly and related work on Wasm-based cryptojacking detection tools.
- **Chapter 3:** Outlines the proposed framework's implementation methodology.
- **Chapter 4:** Presents the results and performance metrics for evaluation of the proposed framework along with its comparative analysis with other tools and two general-purpose malware detection platforms:VirusTotal and MalwareBytes.
- **Chapter 5:** Mentions the strengths and limitation of the framework as well as offers future research works and avenues for further improvements in the area of research.
- **Chapter 6:** This chapter summarizes the research with the conclusion drawn.

Literature Review

2.1 JavaScript

JavaScript, often referred to as the "language of the web," is a versatile programming language that plays an important role in the development of web. JavaScript, initially developed by Netscape, has become a widely used and robust scripting language. It is primarily known for its compatibility with web browsers to create dynamic as well as interactive web pages. JavaScript is a lightweight language which is interpreted during runtime and based on object orientation. It is compatible with imperative, functional, and event-driven programming paradigms. JavaScript is often employed for client-side scripting, enhancing user experience by allowing developers to create responsive interfaces. Over the years, the language has expanded in addition to the web browser environment as it can also be used for development of server-side applications through Node.js, mobile apps through React Native, Ionic, and even in Internet of Things (IoT) related projects.

2.1.1 Usage of JavaScript

JavaScript is used to create dynamic as well as interactive web pages. It is commonly used to perform tasks such as form validation, DOM manipulation, asynchronous communication (AJAX), and animation. JavaScript frameworks and libraries, such as jQuery, Angular, and React, have gained popularity, simplified the development process, and fostered the creation of sophisticated web applications. Moreover, developers can also use JavaScript for both client and server-side development using Node.js, which is JavaScript's server-side counterpart. Node.js, allows developers to use the same language

for both client and server-side development. This unification streamlines the development process, enhancing code re-usability and facilitating the creation of scalable and efficient applications.

2.1.2 Advantages of JavaScript

- **Versatility.** JavaScript's versatility is a key strength. It is not limited to web development and can be used across various platforms and environments.
- **Ease of Learning.** JavaScript's syntax is relatively straightforward, making it accessible to both novice and experienced developers. Its ubiquity in web development also means a vast pool of resources for learning and problem-solving.
- **Interactivity.** JavaScript enables the creation of dynamic and interactive user interfaces, providing a more engaging user experience.
- **Community Support.** JavaScript boasts a large and active community. This fosters collaboration, knowledge sharing, and the continuous improvement of the language through the development of new frameworks, libraries, and tools.

2.1.3 Dis-Advantages of JavaScript

- **Browser Compatibility.** JavaScript behaviour may vary across different browsers, requiring developers to implement browser-specific solutions to ensure consistent performance.
- **Security Concerns.** As a client-side scripting language, JavaScript is susceptible to security vulnerabilities, such as cross-site scripting (XSS) attacks. Developers must implement proper security measures to mitigate these risks.
- **Single-threaded Execution.** JavaScript's single-threaded nature can lead to performance bottlenecks in computationally intensive tasks. This limitation can be addressed through techniques like asynchronous programming and the use of web workers.

2.2 WebAssembly

WebAssembly (Wasm) is a binary instruction format that aims to complement JavaScript by providing a low-level, efficient programming language for the web. While JavaScript excels in certain areas, it has inherent performance limitations, especially in tasks requiring extensive computational power. WebAssembly addresses this gap by allowing developers to write performance-critical code in languages like C, C++, and Rust and then compile it to run in the browser. The need for WebAssembly (Wasm) arises from the evolving landscape of web development, where the demand for more complex and performance-intensive applications has outpaced the capabilities of traditional web technologies, including JavaScript. Below are several key reasons that highlight the growing need for WebAssembly:

- **Performance Optimization.** Modern web applications often involve resource-intensive tasks such as complex calculations, data processing, and graphics rendering. As an interpreted language, it can sometimes struggle to deliver optimal performance in these scenarios. WebAssembly allows developers to write performance-critical code in other programming languages like Rust, C or C++, which can then be compiled to a binary format that runs at near-native speed. This is particularly beneficial for applications requiring real-time processing, simulations, games, and other computationally intensive tasks.
- **Language Diversity and Code Reusability.** Multiple programming languages are supported by WebAssembly, giving developers the freedom to select the language that best suits their level of experience or the demands of a particular task. This enables the integration of existing code bases, allowing developers to leverage legacy code or utilize languages that are better suited for certain tasks. By supporting a variety of languages, WebAssembly facilitates the reuse of code across different platforms, fostering a more efficient development process. Developers can use the same codebase for both web and non-web applications, reducing redundancy and maintenance efforts.
- **Improved Loading Times and Efficiency.** WebAssembly binaries are typically smaller and load faster than equivalent JavaScript files. This is particularly advantageous in scenarios where quick loading times are critical for user experience, such

as in online games, multimedia applications, and interactive simulations. Smaller file sizes mean reduced bandwidth requirements, which is beneficial for users with slower internet connections or those accessing applications on mobile devices.

- **Security Enhancements.** WebAssembly operates in a sandboxed environment, isolating its execution from the rest of the web page. This isolation enhances security by preventing unauthorized access to sensitive resources, mitigating certain types of security vulnerabilities such as those associated with traditional JavaScript code execution. WebAssembly is designed to work seamlessly with JavaScript, allowing developers to integrate Wasm modules into existing web applications. This interoperability ensures a smooth transition for developers who want to enhance specific parts of their applications with WebAssembly without abandoning their existing code base. WebAssembly is supported by all major web browsers, ensuring cross-browser consistency. This broad support enables developers to deploy WebAssembly-powered applications without concerns about compatibility issues.
- **Platform-Independent.** WebAssembly is designed to be platform-independent, enabling developers to write code that runs consistently across different operating systems and architectures. This platform neutrality is crucial for ensuring a uniform experience for users accessing web applications from various devices and environments.

2.3 Cryptocurrency

Cryptocurrency is a type of virtual or digital money that runs on decentralised networks built on blockchain technology and employs encryption for security. Cryptocurrencies rely on a distributed ledger system for transaction recording and verification in contrast to conventional currencies issued by governments and central banks. The creation of Bitcoin in 2009 by an unidentified person known by the alias “Satoshi Nakamoto” signalled the beginning of cryptocurrencies and opened the door for a wide range of digital assets. The underlying technology of cryptocurrencies is "Blockchain" which is an immutable, decentralised ledger that keeps track of every transaction made via a computer network. Transparency, security, and fraud resistance are therefore guaranteed. Each transaction is grouped into a block, and these blocks are linked sequentially, forming a

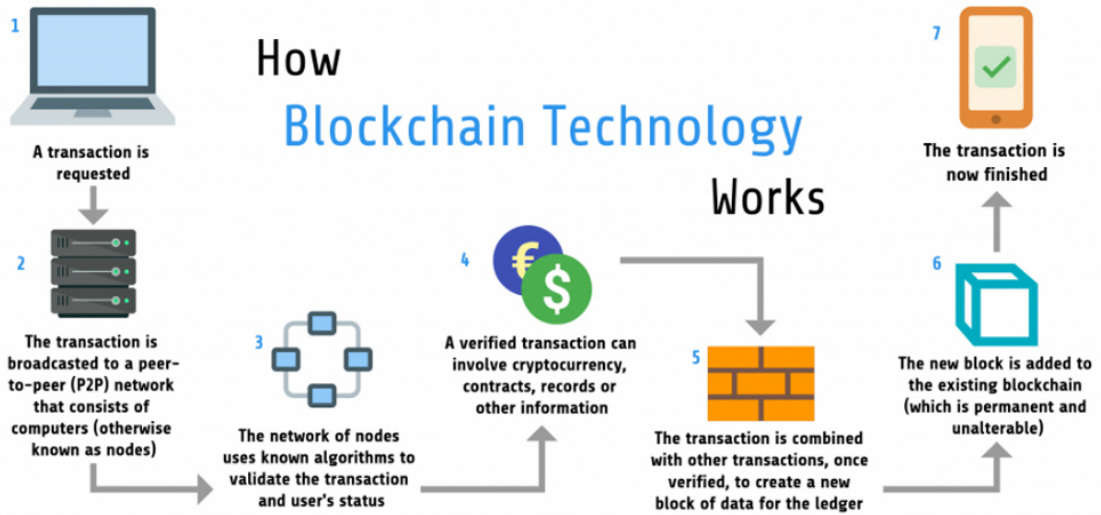


Figure 2.1: Working of Blockchain Technology

chain.

In addition to Bitcoin, a plethora of other cryptocurrencies have surfaced, each with distinct characteristics and objectives. For example, Ethereum introduced smart contracts, which are self-executing contracts with the conditions of the agreement clearly encoded into code. Whereas “Litecoin” prioritises faster transaction confirmation times and “Ripple” concentrates on enabling speedy and affordable cross-border payments.

Cryptocurrencies are based on P2P (peer-to-peer) network therefore intermediaries like banks are not required. Transactions are verified through a process called mining or consensus mechanisms, depending on the cryptocurrency. Mining involves solving complex mathematical problems, while consensus mechanisms like proof-of-stake rely on validators who hold a stake in the currency.

The market demand, changes in regulations, and breakthroughs in technology can all have an impact on the value of cryptocurrencies. Despite obstacles, cryptocurrencies have become widely accepted, leading to financial technology innovation, and eliminating the dependency on conventional system of money and banking structures.

2.4 Cryptojacking

Cryptojacking, short for cryptocurrency mining hijacking, refers to the unauthorized use of computing resources to mine cryptocurrencies. In this malicious practice, perpe-

trators exploit the processing power of computers, servers, or other devices without the knowledge or consent of the owners. The primary goal is to mine cryptocurrencies like Bitcoin, Monero, or Ethereum, utilizing the computational power of victims' machines to perform the complex calculations required for validating and adding transactions to the blockchain. As cryptocurrencies gained popularity, so did the methods employed by cybercriminals to acquire them illicitly. Cryptojacking emerged as a stealthy and financially motivated form of cyber attack that targets both individuals and organizations. Unlike traditional cyber threats such as ransomware, which directly demand payment from victims, cryptojacking operates covertly by harnessing the computational resources of compromised devices.

The typical scenario involves injecting malicious code, often in the form of JavaScript, into websites, online ads, or email attachments. Users unknowingly execute these scripts when visiting compromised websites or interacting with infected content. Once activated, the scripts initiate the mining process in the background, utilizing the victims' computing power to solve cryptographic puzzles and mine cryptocurrency. The appeal of cryptojacking for attackers lies in its subtlety; it can go unnoticed for extended periods, allowing perpetrators to accumulate cryptocurrency without raising immediate alarms. The impact on victims includes degraded system performance, increased energy consumption, and potential hardware wear and tear.

2.4.1 Coinhive and Monero

Coinhive was a JavaScript-based cryptocurrency mining service that gained notoriety for its controversial use in browser-based mining of Monero (XMR) cryptocurrency. Launched in 2017, Coinhive provided a simple way for website owners to monetize their platforms by utilizing the processing power of visitors' computers to mine Monero.

Monero, known for its privacy-focused features, became the preferred choice for browser mining due to its efficient mining algorithm, CryptoNight. Unlike Bitcoin, Monero's privacy-centric design obscures transaction details, making it well-suited for discreet browser-based mining. Coinhive's script, when embedded in a website, would execute on visitors' browsers, harnessing their CPU power for cryptocurrency mining without explicit consent. However, the practice of in-browser cryptocurrency mining without user consent garnered significant criticism. Many users considered it a violation of their

computer resources and privacy, leading to negative perceptions of both Coinhive and Monero.

In response to growing backlash, major web browsers implemented measures to block or restrict Coinhive's scripts, and antivirus software flagged them as potentially unwanted applications. Coinhive eventually shut down its services in early 2019, citing economic infeasibility and the negative impact on Monero's reputation. While Monero itself is a legitimate privacy-focused cryptocurrency with robust security features, Coinhive's association with unauthorized and intrusive mining practices underscored the ethical challenges surrounding cryptocurrency usage. The incident highlighted the importance of responsible and transparent implementation of cryptocurrency-related services to maintain user trust and uphold ethical standards in the evolving landscape of digital currencies.

2.4.2 CryptoNight Algorithm

CryptoNight is the hashing algorithm employed by Monero (XMR), a privacy-focused cryptocurrency. Introduced in 2013 by an anonymous user known as "Bytecoin," CryptoNight is designed to be ASIC-resistant, meaning it resists efficient mining by specialized hardware (Application-Specific Integrated Circuits), promoting a more decentralized network and inclusive mining opportunities for individuals. The significance of the CryptoNight algorithm in Monero lies in its focus on privacy and security. Unlike Bitcoin, where transactions are transparent and traceable on the blockchain, Monero utilizes features like ring signatures, confidential transactions, and stealth addresses. CryptoNight supports these privacy features by obfuscating transaction details, making it difficult to trace the sender, recipient, and transaction amount.

CryptoNight's ASIC resistance also aligns with Monero's commitment to decentralization. By preventing the dominance of specialized mining hardware, it encourages a broader range of individuals to participate in the mining process, promoting a more democratic distribution of mining power. It is essential to note that the landscape of cryptocurrency mining is dynamic, and efforts to maintain ASIC resistance have led to periodic updates and adjustments to the CryptoNight algorithm. Overall, CryptoNight's significance in Monero is rooted in its contribution to privacy, security, and the ethos of decentralized mining within the cryptocurrency ecosystem.

2.4.3 Cryptojacking through Web Applications

Cryptojacking through web applications has emerged as a significant cybersecurity threat, exploiting unsuspecting users' computing resources to mine cryptocurrencies without their knowledge or consent. This malicious activity involves embedding cryptocurrency mining scripts into websites, which are then executed on visitors' browsers, leveraging their processing power to mine digital currencies. The perpetrators of cryptojacking seek to capitalize on the increasing popularity of cryptocurrencies and the rising value of coins like Bitcoin and Monero. They discreetly inject JavaScript or other scripting languages into compromised websites or through malicious ads, exploiting vulnerabilities in poorly secured web applications. When users visit these compromised sites, the embedded scripts activate, utilizing the visitors' device resources to perform the complex mathematical calculations required for cryptocurrency mining.

The impact of cryptojacking on web application users can be profound. It can lead to sluggish performance, increased energy consumption, and potential hardware damage, as the mining process strains the CPU and other components. Additionally, users may experience reduced battery life on mobile devices. Preventing and mitigating cryptojacking involves implementing robust security measures for web applications. Regular security audits, patching known vulnerabilities, and employing content security policies are crucial steps. Website owners can also use specialized tools to detect and block cryptojacking scripts. Furthermore, end-users can protect themselves by using browser extensions that block known mining scripts and keep their software and security tools updated.

As the cryptocurrency landscape continues to evolve, vigilance against cryptojacking remains essential. In addition, individuals and organizations must implement robust cybersecurity measures, including regular software updates, network monitoring, and the use of security tools that can detect and block malicious mining scripts. As the threat landscape continues to evolve, awareness and proactive defenses are crucial to safeguarding against the stealthy menace of cryptojacking.

2.5 Analysis (Static vs Dynamic analysis)

Analysis is essential to gain insightful knowledge and making calculated informed decisions in many fields of work. Two fundamental types of analysis, static and dynamic analysis, differ in their approaches and objectives.

2.5.1 Static Analysis

Static analysis is a method that examines software, code, or data without executing it. It is a process of inspecting and evaluating a program's source code or artifacts to identify potential issues, vulnerabilities, or compliance violations. Common static analysis techniques include code reviews, syntax checking, and automated tools that analyze the code structure. Static analysis is valuable for identifying issues early in the development process, enhancing code quality, and ensuring compliance with coding standards. It is particularly useful for security assessments, as it can uncover vulnerabilities without running the code.

2.5.2 Dynamic Analysis

Dynamic analysis, in contrast, involves the evaluation of software or systems while they are running or in operation. It includes techniques such as testing, monitoring, and profiling to assess the program's behaviour, performance, and interactions with its environment. Dynamic analysis is well-suited for detecting runtime issues, memory leaks, and performance bottlenecks. Common dynamic analysis methods include penetration testing, runtime code instrumentation, and behaviour monitoring. This type of analysis provides insights into the actual execution of the system, making it valuable for assessing real-world performance and functionality.

In the succeeding paras we give a comprehensive review of WebAssembly as well the cryptojacking detection.

2.6 Review of WebAssembly

Even though relatively new technology, a fair amount of research has been carried out to date on Web Assembly. Vili-Petteri Niemela [7] carried out a basic study on WASM pro-

Table 2.1: Comparison of Static and Dynamic Analysis

Aspect	Static Analysis	Dynamic Analysis
Timing	Performed before runtime.	Performed during runtime.
Execution	Carry out analysis without executing.	Evaluates code while it is running or in operation.
Scope	Focuses on code/ binary structure and artifacts.	Assesses behaviour, performance, and interactions.
Purpose	Identifies potential issues early in development.	Detects runtime issues, performance bottlenecks.
Examples	Code reviews, syntax checking, automated tools.	Testing, monitoring, profiling, runtime instrumentation.
Use Cases	Security assessments, compliance checks.	Performance analysis, debugging, real-world behaviour.
Advantages	Early issue detection, compliance assurance.	Real-world insights, performance evaluation.
Limitations	Limited to static code analysis.	May miss issues not encountered during runtime.
Complementarity	Provides a foundation for early issue detection.	Offers insights into runtime behavior and performance.
Common Techniques	Code review, syntax analysis, automated tools.	Testing, profiling, runtime code instrumentation.

viding a general introduction of WASM along with its applications and security features. It also gives examples of converting a couple of high-level language (RUST) programs to WASM: one was a simple Hello World and the other was a basic QR code reader application. M. Kim et al. [10] surveyed security enhancement techniques of WASM binaries with a quantitative and qualitative analysis as well as limitations of each technique. They focused more on the deployment of WASM in a cloud-native environment. The survey divided the security techniques into 2 types (1) detection of malicious WASM binaries and (2) protecting WASM binaries including their vulnerability analysis.

Tushar and Biju R Mohan [22] carried out a comparative analysis of JavaScript and WASM. They used 2 types of programs, a recursive Fibonacci function and an iterative function to check a prime number, in JS and WASM (compiled from C, Rust & GO). They measured the resources (CPU and memory usage) utilized and time taken by running the programs in 2 different types of browsers i.e., Mozilla and Firefox. The results show that while JavaScript outperforms WASM when computations are easy but WASM is far superior in handling large calculations. According to the results, WASM is better than JavaScript in handling large calculations but JavaScript outperformed WASM when the computations are easy.

Y. Yan et al. [23], conducted a study regarding the performance of WebAssembly applications and compared it with JavaScript. They performed measurements on three different sets of programs with diverse settings. Findings include:-

- The performance of Wasm is dependent on compiler and runtime environment.
- The limited effect of JIT and compiler optimizations on Wasm.

- More memory utilization by Wasm as compared to JavaScript.

Ghafari et al. [24], investigated whether porting C programs to WASM affects the security of programs or not. They compiled 17802 programs, deterministic and terminating, having common vulnerabilities to 64-bit x86 native code and WASM binaries. They concluded that the running 4911 binaries produced varied results and that was due to 3 different root causes (1) different standards of libraries implementations (2) low security measures in Web Assembly (3) different semantics of the execution environments. B. Litzer [25] suggested an in-place interpreter for WASM rather than using optimized compilers or baseline compilers for creating WASM binaries. He carried out a comparative analysis of binaries created using optimizing compilers, baseline compilers, rewriting interpreter and using his interpreter which produced better results in terms of processing time and space overhead utilized.

2.7 Work Related to Cryptojacking

There has been a huge amount of work undertaken for the detection of web-based/in-browser cryptojacking malware. However, here we mention only those related to Wasm-based cryptojacking, further categorized as per type of analysis:

- **Static Analysis.** Naseem et al. [20], proposed a lightweight and fast static analysis-based mechanism, MINOS, that uses deep learning (an image-based classification technique) to detect cryptojacking Wasm with a detection accuracy of 98.97%. Similarly, Konoth et al. [26] suggested MineSweeper, which is based on the knowledge of intrinsic components, hashing function execution, of cryptomining algorithm (CryptoNight). It also deals with evasion techniques by utilizing an obfuscation-resilient approach of monitoring CPU cache events at run time to identify crypto miners based on suspicious memory access patterns.
- **Dynamic Analysis.** Wang et al. [16] proposed SEISMIC, which carries out dynamic analyses of Wasm binaries by instrumenting them to profile specific cryptomining instructions i.e., XOR instructions during runtime. It achieved 98% accuracy in detecting crptomining Wasm with negligible false positives. Bian et al. [27] introduced MineThrottle, which also used a similar approach of instrumenting Wasm modules for counting specific instructions, achieving 0% false positive and

1.83% false negative. Both SEISMIC and MineThrottle also use machine learning models for the classification of Wasm modules as benign or malicious. Romano et al. [28] proposed MinerRay, which identifies Wasm-based cryptojacking by dynamically analyzing their control flow graph during execution and detecting specific cryptographic patterns usually associated with cryptojacking.

- **Hybrid Analysis.** Kharraz et al. [29] proposed a method which used seven distinct features relevant to cryptojacking to train a Support Vector Machine (SVM) model along with searching for cryptomining function names to detect cryptojacking, achieving 97.9% TPR and 1.1% FPR. Lehmann et al. [30] proposed a general-purpose framework, Wasabi, for dynamically analyzing WebAssembly modules. It is based on binary instrumentation of Wasm modules and provides high-level API, written in JavaScript, to carry out different types of dynamic analysis i.e., log analysis, call graph extraction, taint analysis etc.

Table 1 provides a summary of all detection tools for cryptojacking based on Wasm.

Table 2.2: Summary of WebAssembly-Based Cryptojacking Detection Tools

Year	Tool Name	Type	Brief Overview of Working	Dataset		Results
				Training	Evaluation	
2018	Mine-Sweeper [26]	Static	<ul style="list-style-type: none"> · Based on the knowledge of intrinsic components of cryptomining algorithm (CryptoNight). · Can also deal with evasion techniques (obfuscation) by observing events caches in CPU during running. 		Crawled Alexa TOP 1M websites to collect 40 unique Wasm Binaries	No False positive

2021	MINOS [20]		<ul style="list-style-type: none"> · Converts Wasm binaries to greyscale images. · Implements a Convolutional Neural Network (CNN) classifier to distinguish between cryptojacking or benign images. 	150 malicious and benign binaries each.	Compiled from PublicWWW and Tranco	<ul style="list-style-type: none"> · 98.97% success rate · Low TNR and FPR
2018	SEIS-MIC [16]	Dy-namic	<ul style="list-style-type: none"> · Instrumenting Wasm binaries profile-specific cryptographic instructions i.e., XOR. · Use machine learning models for classification 		12 applications (5 mining and 7 non-mining)	<ul style="list-style-type: none"> · 98% · Negligible FPR
2020	MineThrottle [27]		<ul style="list-style-type: none"> · Block-level instrumentation of Wasm · Use machine learning models for classification 		Alexa TOP 1M websites	<ul style="list-style-type: none"> · 0% FPR · 1.83% FNR
2020	MinerRay [28]		<ul style="list-style-type: none"> · Analyses Cryptojacking Wasm modules' control flow graph during execution and detecting specific cryptographic patterns 		Alexa TOP 1M websites	N/A
2019	Out-guard [29]	Hy-brid	<ul style="list-style-type: none"> · Used seven distinct features relevant to cryptojacking to train a Support Vector Machine (SVM) model 		Alexa Top 1M websites	<ul style="list-style-type: none"> · 97.9% TPR · 1.1% FPR

Methodology

3.1 Intrinsic Characteristics of Cryptojacking Malware

All types of cryptojacking malware have some peculiar features which make them distinctive from other benign applications. The authors of cryptojacking malware have to implement optimized proof-of-work (PoW) schemes that use less computational resources. As depicted by previous studies, almost all malicious cryptojacking implementations have similar intrinsic structures based on some encryption algorithms and the core features remain the same which are essential for cryptomining operations [17, 25]. i.e., hash calculations, utilizing high computational resources, opening communication channels with the mining pool through web sockets, generating web workers threads etc. Furthermore, the size of cryptojacking wasm binaries is usually small as compared to benign applications (games and graphics) which usually have iterative functions. In the proposed framework, we have focused on the following important features of cryptojacking malware, especially used in the CryptoNight algorithm, which are essential to perform any detection through static analysis:

3.1.1 Semantic Signature Matching

Semantics refers to the meaning or interpretation of language, focusing on how words, phrases, and symbols convey significance within a given context. In computing and information technology, semantics is crucial for understanding the intent and function of data or programming languages. Semantic Signature Matching is a method used in cybersecurity to identify and detect patterns associated with malicious code or threats.

It involves creating unique semantic signatures that capture the behavioural aspects and characteristics of specific malware or attack patterns. These signatures go beyond traditional static signatures and instead focus on understanding the intent and actions of the code. By employing Semantic Signature Matching, cybersecurity systems can proactively recognize and thwart sophisticated threats, enhancing the overall resilience and responsiveness of security measures in the ever-evolving landscape of digital threats. Cryptojacking Malware have distinctive computation semantics whereby they have a higher percentage of typical cryptographic instructions as compared to benign applications. Cryptocurrency mining algorithms usually have iterative or repetitive instructions. Moreover, only a few blocks of code use the majority of CPU time and hashing code also has certain specific cryptographic operations (rotate, shift, and XOR operations). Moreover, the semantics are somewhat difficult, not impossible, to obfuscate as they are intrinsic to cryptojacking functionality but at the cost of performance. The five most executed specific operations in applications are identified [20] i.e. `i32.shl`, `i32.add`, `i32.shr_u`, `i32.and`, and `i32.xor`. In semantic signature matching, profiling of the above-mentioned instructions is done which shows a distinctive pattern of cryptomining malware. Fig 3.1 clearly shows a higher percentage of rotate, shift, and XOR operations in crypto-based applications as compared to the benign ones (first three are crypto applications and next three are benign).

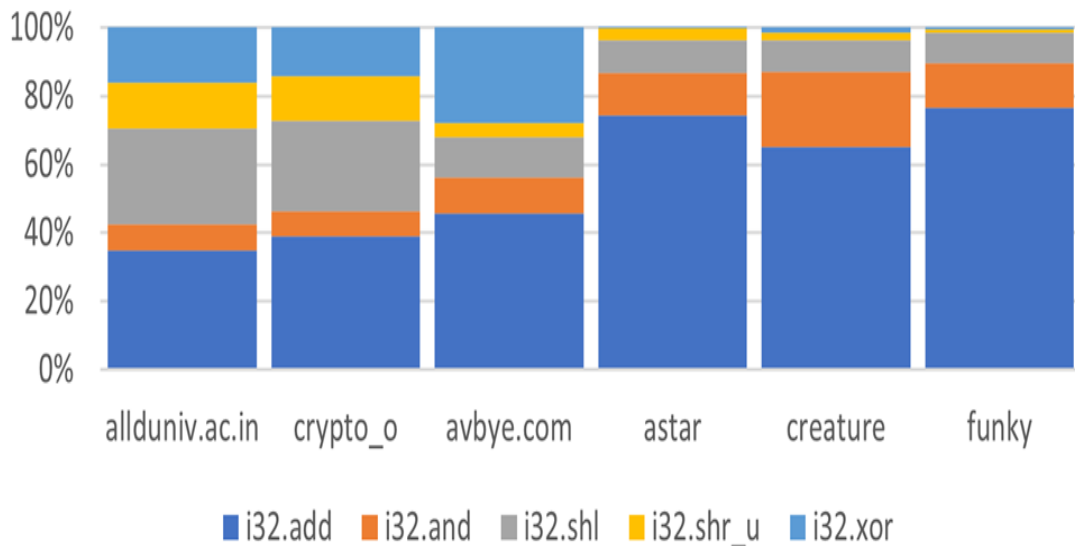


Figure 3.1: Comparison of Instructions between Crypto and Benign WASM



Figure 3.2: Word Cloud of Strings and Keywords

3.1.2 Strings & Pattern Matching

In computer science, strings are sequences of characters, typically used to represent text or data. String manipulation is a fundamental operation in programming, involving tasks such as searching, modifying, and extracting information from these character sequences. Pattern matching is a powerful concept within string manipulation, enabling the identification of specific patterns or sequences of characters within a larger text. Pattern matching involves searching for a defined sequence or structure within a string, making it essential for tasks like text search algorithms, data parsing, and regular expressions. Algorithms like the Knuth-Morris-Pratt and Boyer-Moore are commonly employed for efficient string pattern matching. This capability is crucial in various applications, including text processing, data validation, and information retrieval, contributing to the efficiency and functionality of algorithms and software systems that deal with textual data.

Cryptomining codes contain certain function names and strings, if not obfuscated, which are only specific to them i.e. `cryptonight_hash`, `cryptonight_create`, `cryptonight_destroy`, etc. A list of relevant strings and keywords is curated after analyzing different cryptomining applications. Even though strings and names can be easily obfuscated they do provide some insight into distinguishing malicious Web Assembly Binaries.

3.1.3 Call Flow Graph

A Call Flow Graph (CFG) is a visual representation that illustrates the sequence of function calls and the flow of control within a software program. It is a valuable tool in software analysis and debugging, providing insights into the program's structure and the interactions between different functions or procedures. In a Call Flow Graph, nodes represent functions or methods, and directed edges depict the flow of control between them. The graph reflects the order in which functions are called during the execution of the program, creating a visual map of the program's execution paths. This visualization aids developers in understanding the program's architecture, identifying potential issues, and optimizing code.

CFGs are particularly useful in security analysis, software testing, and debugging. They assist in identifying vulnerabilities, analyzing code coverage, and comprehending the impact of code changes. Security analysts often use CFGs to trace the flow of sensitive information or identify potential entry points for security exploits. By providing a clear and concise representation of a program's execution, Call Flow Graphs enhance developers' ability to comprehend complex software systems, facilitating effective problem-solving, debugging, and optimization processes. They serve as a valuable visual aid in software engineering and play a crucial role in enhancing the overall understanding of program behaviour.

As mentioned earlier that a lot of cryptojacking malware is based on Coinhive's CryptoNight algorithm as this is the most optimized and lightweight algorithm which can run on reduced resources. Any application of the CryptoNight algorithm or its variant will have a similar call graph structure whereas benign applications will be the least similar. A visual depiction of similarity and difference between CryptoNight and benign applications is shown in subsequent diagrams.

3.1.4 Cryptographic Primitives

Cryptographic primitives are fundamental building blocks in the field of cryptography, serving as the basic components for constructing secure communication and information protection systems. These primitives are mathematical algorithms or protocols designed to provide essential security services, such as confidentiality, integrity, authentication, and non-repudiation. Some common cryptographic primitives include:-


```

crypto_functions = {
  "ref": {"i32.shr_u": 0, "i32.shl": 0, "i32.xor": 0, "i64.shl": 0, "i64.xor": 0, "i64.shr_u": 0, "loop": 0, "if": 0},
  "aes1": {"i32.shr_u": 12, "i32.shl": 16, "i32.xor": 16},
  "aes2": {"i32.shr_u": 120, "i32.shl": 160, "i32.xor": 160},
  "keccak": {"i32.shl": 4, "i64.shl": 6, "i64.xor": 101, "i64.shr_u": 6},
  "keccak2": {"i32.shl": 10, "i64.xor": 61, "i64.shl": 6, "i64.shr_u": 6, "loop": 3, "if": 2},
  "skein1": {"i64.shl": 288, "i64.xor": 305, "i64.shr_u": 288},
  "skein2": {"i64.shl": 144, "i64.xor": 153, "i64.shr_u": 144},
  "skein3": {"i32.shl": 24, "i64.shl": 64, "i64.xor": 97, "i64.shr_u": 64},
  "blake3": {"i32.shr_u": 32, "i32.shl": 128, "i32.xor": 80},
  "blake4": {"loop": 2, "if": 1, "i32.xor": 80, "i32.shl": 85, "i32.shr_u": 32},
  "blake5": {"loop": 2, "if": 1, "i32.xor": 80, "i32.shl": 69, "i32.shr_u": 32},
  "groestl1": {"i32.shr_u": 99, "i32.shl": 286, "i32.xor": 127},
  "groestl2": {"i32.shr_u": 99, "i32.shl": 286, "i32.xor": 136}
}

```

Figure 3.6: Crypto Primitives Fingerprinting in WASM Binaries

- **Symmetric Encryption.** Involves using only one secret key for both encryption and decryption. Popular algorithms include DES (Data Encryption Standard) and AES (Advanced Encryption Standard).
- **Asymmetric Encryption.** Carries out encryption and decryption using a pair of keys, known as private and public keys. Examples of asymmetric algorithms include ECC (Elliptic Curve Cryptography) and RSA (Rivest-Shamir-Adleman) and which are most popular.
- **Hash Functions.** Transform data into fixed-length strings of characters, known as message digest or hash values. Hash functions such as SHA-256, SHA-512 (Secure Hash Algorithm) are crucial for ensuring data integrity. Other examples of hash functions include Keccak, Groestl, Blake, Skein etc.
- **Digital Signatures.** Integrity and authenticity of digital messages are provided by digital signatures. Algorithms like RSA and DSA (Digital Signature Algorithm) are commonly used for generating such signatures.
- **Key Exchange Protocols.** Facilitate secure communication by allowing parties to establish shared secret keys over insecure channels i.e. Diffie-Hellman and Elliptic Curve Diffie-Hellman (ECDH).

Cryptographic primitives form the foundation of secure communication, data protection, and authentication in various applications, including secure messaging, online banking, and e-commerce. Robust and well-established cryptographic primitives are essential for building secure systems that can fend off sophisticated attacks and guarantee the confidentiality and integrity of sensitive data.

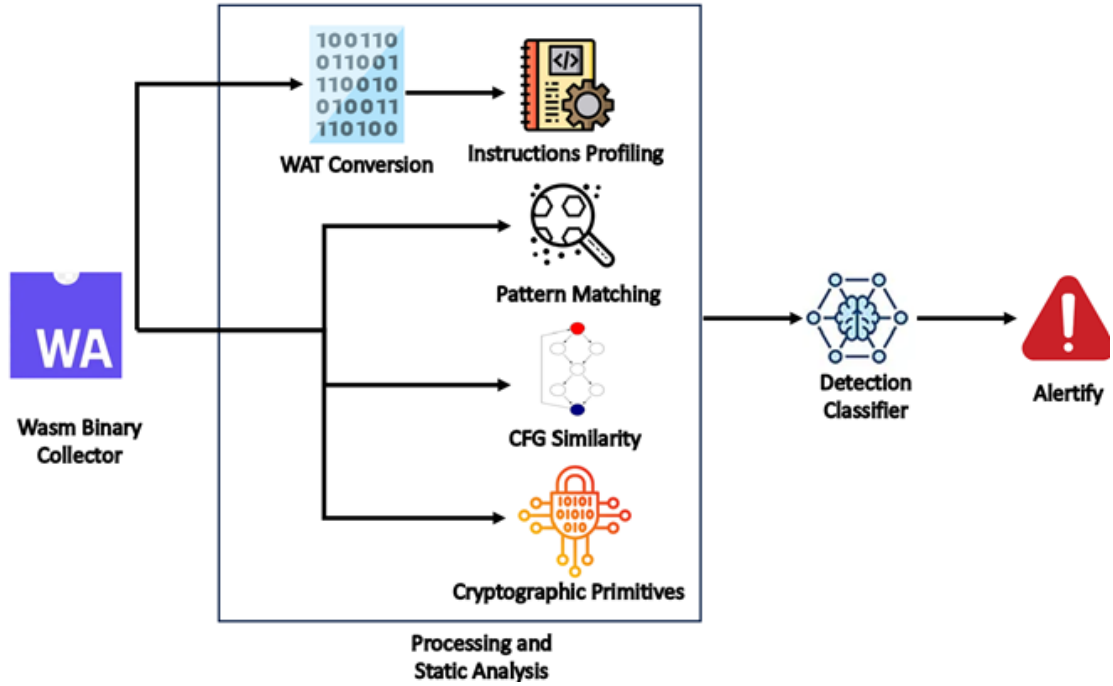


Figure 3.7: Overview of Framework

The CryptoNight algorithm employs certain cryptographic functions which are essential for its working. Algorithms such as AES, Keccak, Groestl, BLAKE, Skein are predominantly found in crypto-based applications and all variants of CryptoNight use these functions for required calculations.

3.2 Architecture of Proposed Framework

Figure 3.3 gives an overview of the proposed framework components. The technical details of the proposed framework are outlined in this section. The framework is implemented in Python3 with PowerShell scripts for automation. The core components and functionalities of the framework are mentioned below:-

3.2.1 Wasm Binary Collector

The WebAssembly (Wasm) Binary Collector is a feature in our proposed framework that facilitates the efficient handling and management of WebAssembly binary code. It acts as a collector for these binary modules and optimizing their retrieval. This module automatically collects Wasm binaries from the sites visited by the user and saves the

Wasm binaries to local directory.

3.2.2 Wasm to WAT conversion

The `wat2wasm` tool is part of the WebAssembly Binary Toolkit (`wabt`), a suite of tools for working with WebAssembly. Specifically, `wat2wasm` converts WebAssembly Text Format (WAT) to the binary format (Wasm). WAT is a human-readable representation of WebAssembly, whereas Wasm is the compact, binary format that browsers and runtimes execute. WebAssembly (Wasm) to WebAssembly Text Format (WAT) conversion is the reverse process, allowing developers to convert the binary Wasm code back to the human-readable WAT format. This conversion can be useful for debugging, understanding code structure, and making manual modifications before recompiling and deploying the WebAssembly code. The collected binaries are then converted to WAT using the `wasm2wat` tool of The WebAssembly Binary Toolkit (WABT) [31] for further evaluation.

3.2.3 Processing and Static Analysis

Moser et al. [32] explained that combining multiple independent techniques can effectively detect a wide range of malware and have greater detection efficacy as compared to using a single one. Therefore, our framework will mainly carry out static analysis based on the following techniques:-

- **Individual Crypto Instructions Profiling.** This module counts the overall number of specific cryptographic operations executed in the application. Cryptomining applications have a higher count of rotate, shift and XOR operations as compared to benign applications.
- **Pattern Matching using Yara Rule.** YARA rules are a set of syntax-based patterns used in the YARA tool for identifying and classifying malware or specific patterns within files or data streams. YARA is an open-source pattern or string-matching tool that enables cybersecurity professionals, and analysts to create customized unique rules for detecting and categorizing malicious code based on features such as strings, regular expressions, and binary patterns. The significance of YARA rules lies in their ability to enhance threat detection and response

capabilities. Security experts can create YARA rules tailored to the unique signatures and behaviours of known malware or suspicious patterns. When applied to a system or network, YARA rules can efficiently identify and flag potential threats, aiding in the early detection of malicious activity. This makes YARA rules a valuable asset in the arsenal of cybersecurity tools, empowering analysts to proactively identify and mitigate security risks through effective pattern matching and signature-based detection. The Yara rule is used to find the presence of crypto-related keywords. A carefully curated list of patterns and strings is prepared which is used in the rule to carry out pattern/ string matching and find the sum of several matches in the Wasm binaries.

- **Block Function Crypto Instructions Profiling.** In this module, a profile of functions used in a reference CryptoNight Wasm is prepared based on the ratio of specific cryptographic operations. The same is calculated for other Wasm binaries and then compared with the reference profile to find the presence of matching cryptographic functions.
- **Call Flow Graph Similarity.** Dot files are generated from the Wasm binaries which are compared with a standard CryptoNight dot file to calculate a similarity score (1-0) based on nodes, edges, and overall graph structure. A score of 1 means a perfect match whereas 0 shows the least similar result.
- **Cryptographic Primitives Detection.** The presence of these cryptographic primitives is calculated in any Wasm binary using fingerprinting which is defined for each primitive based on a certain cryptographic operations count in functions. The total count of primitives is then noted distinguishing a cryptomining Wasm from a benign one. A higher count relates to a crypto application.

3.2.4 Detection Classifier

Machine learning is a field of artificial intelligence that empowers computers to discern patterns and render decisions on their own without explicit programming. In the context of machine learning, binary classification is a prevalent task wherein the objective is to categorize input data into one of two classes, commonly denoted as positive (1) or negative (0). Various classifiers are employed for binary classification tasks, each with its strengths and applications. Some of the types of classifiers are mentioned below:-

- **Logistic Regression.** It is a linear model used for binary classification which simulates the chances that a sample belongs to a particular class.
- **Support Vector Machines (SVM).** SVM aims to find a hyperplane that best separates data into two classes. It is effective in high-dimensional spaces.
- **Decision Trees.** Decision trees recursively split data based on feature values, forming a tree-like structure. They can capture very complex relationships and are interpretable.
- **Random Forest.** The Random Forest is a collective technique that constructs numerous decision trees and amalgamates their predictions to improve accuracy and resilience.
- **Naive Bayes.** Naive Bayes are based on Bayes' theorem and assume independence among characteristics making them effective for tasks like text classification and spam filtering.
- **K-Nearest Neighbours (KNN).** KNN categorizes data by considering the predominant class among its k-nearest neighbours, making it a simple and effective algorithm.
- **Neural Networks.** Deep learning models, particularly neural networks, can be employed for binary classification tasks, leveraging multiple layers to learn complex representations.

It is essential to comprehend each classifier's details to choose the best model for a given situation. Factors such as dataset size, feature space, interoperability, and computational efficiency influence the choice of classifier in machine learning applications. Experimentation, understanding the strengths and weaknesses of each algorithm and iterative refinement are common practices in achieving optimal model performance in binary classification scenarios.

In this module, the output of all static analysis techniques is combined and fed into the AI-driven Detection engine, a Random Forest (RF) based classifier. The model is pre-trained on a dataset of 191 benign and 163 malicious binaries. The result of the processor is input to the trained model which classifies the binary as either malicious or benign.

3.2.5 Alertify

In case of detection of cryptojacking Wasm binary based on the results of the classifier, it will provide alerts and notify the user to terminate the program. Users can continue performing the task as usual if the application is termed as a benign one.

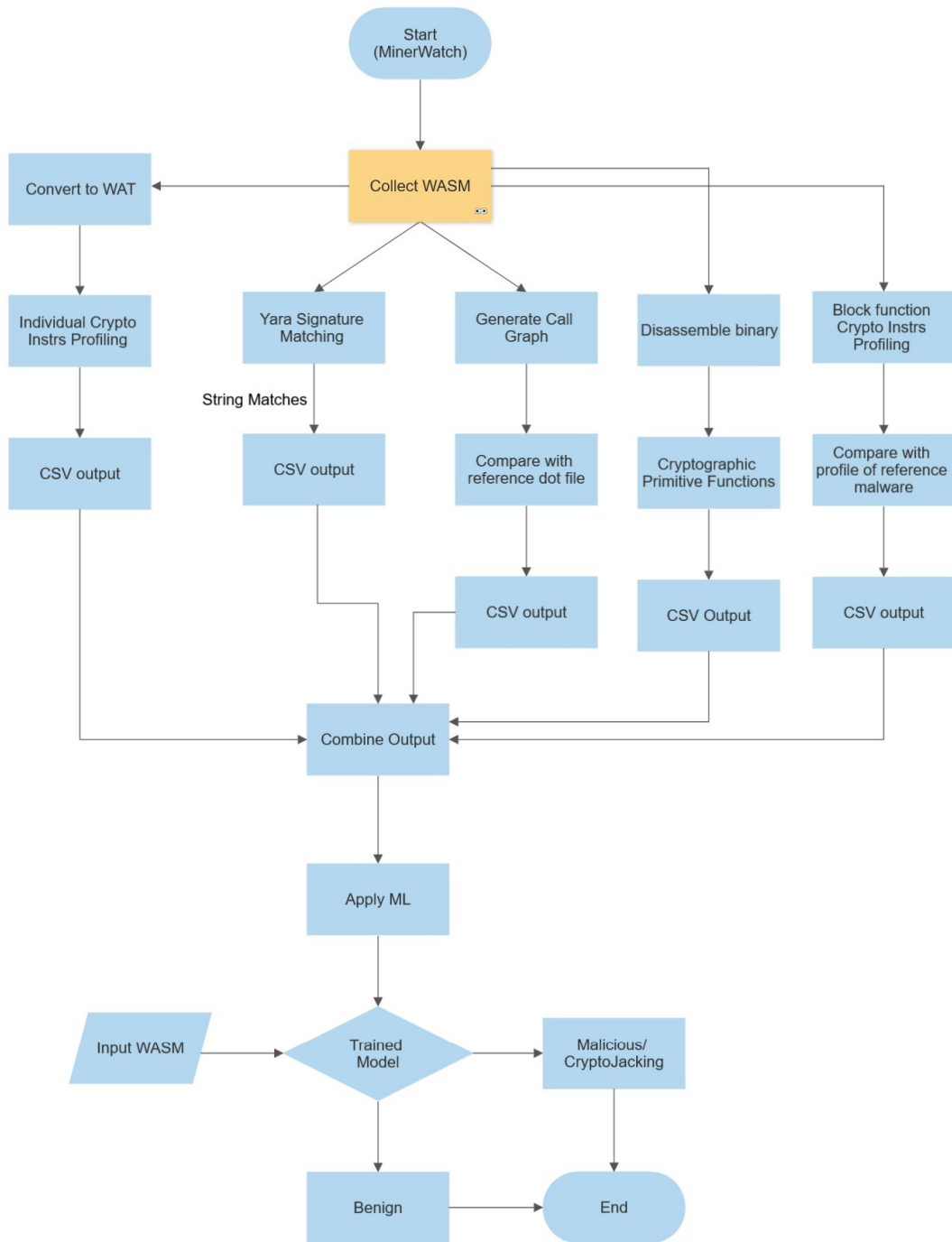


Figure 3.8: Detailed Work Flow of Framework

Result And Performance Evaluation

This section provides details about the dataset gathered to train the framework and the selection of a machine learning classifier. In addition, the performance of the framework is also evaluated in detail along with a comparative analysis of own results gathered with all the tools and specifically two general-purpose malware detection tools: VirusTotal and Malwarebytes.

4.1 Preparation of Dataset

One of the main challenges of this research was to create a meticulously curated dataset for the machine learning task to get the maximum performance out of the model. First, the Web Assembly binaries of both benign and crypto applications were collected from different sources i.e. research papers, GitHub repositories and visiting Web pages. A portion of the binaries was collected from MineSweeper [26], SEISMIC [16], Yara [33] and Musch et al. [34] who have already classified the binaries as malicious and benign. For the remaining, MinerRay [28] and WasmBench were used, both of which contained 162 and 8448 wasm binaries respectively. WasmBench is one of the largest datasets of Wasm binaries to date. The wasm binaries of both MinerRay and WasmBench were evaluated through VirusTotal to further distinguish between benign and cryptojacking binaries (marked as malicious by either 2 or more engines of VirusTotal). After the collection of binaries, they were then processed to extract the desired features as already discussed

previously. The final dataset consists of 427 binaries with a binary label of benign and crypto. The resulting dataset, stored in a CSV file, is imbalanced (255 benign and 172 crypto entries), well-structured, and comparatively small. The first column of the CSV is the label, and the remaining columns contain a numerical representation of different extracted features. This dataset is then used to train the framework's Classifier which is split into a ratio of 70:30 for training the model and subsequently evaluation.

4.2 Selection of Machine Learning Classifier

Given the nature of the compiled dataset, 2x machine learning classifiers, Linear Support Vector Machine (SVM) and Random Forest (RF), were shortlisted for training on the dataset. The main reasons for selecting these classifiers are being lightweight, fast, and best suited for our dataset:-

4.2.1 Linear Support Vector Machine (SVM)

Support Vector Machine is a type of supervised learning algorithm used for classification and regression tasks. SVM tried to locate n hyperplane in an n-dimensional space and distinctly classifies dataset into classes. SVM can also deal with both linear and non-linear data. Linear Support Vector Machine (Linear SVM) is a special case of SVM where the decision boundary is a linear combination of input features and is especially used for linearly separable data. Some of its advantages are listed below:-

- A powerful machine learning model commonly used for binary classification tasks.
- Simpler and faster as it is memory efficient, providing a balance between performance, computational efficiency and interoperability.
- Good for datasets with a low number of features.
- Is accurate and robust.

4.2.2 Random Forest (RF)

Random Forest is an ensemble learning technique which means combining the predictions of several models to improve the accuracy and robustness of the results. It uses multiple

decision trees to perform classification tasks and each tree is constructed by selecting a random subset of the training data, which is called bootstrapping or bagging and a random subset of features, which is called feature bagging or the random subspace method. The final prediction is typically the mode (classification) or mean or median (regression) of the outputs of individual trees. Some of its advantages are listed below:-

- Provides high accuracy in both training and testing datasets.
- Optimally handle categorical and numerical data as well as imbalanced datasets effectively.
- Less sensitive to outliers or anomalies in the dataset due to aggregation of predictions.
- Less prone to overfitting compared to individual decision trees.

4.3 Results

This section of the study aims to present the outcome of the research and implementation of the framework conducted to evaluate the effectiveness of detecting malicious cryptojacking wasm binaries. It will provide details of the performance accuracy achieved to demonstrate the efficacy and usability of the proposed framework. Both Linear SVM and Random Forrest classifiers were then applied to our compiled dataset. A 70:30 ratio split was used, 70% of the data (298 values) was utilized for training the model and the remaining 30% (129 values) for testing the performance. The dataset used for training and evaluation consisted numerical representation of different extracted features of Web Assembly binaries. Performance of a classification model in machine learning is evaluated using several metrics, each with a specific purpose. Accuracy, recall, precision, and F1 score are commonly used measures to provide insights into different aspects of a model's effectiveness which are defined and calculated as follows:-

- **Accuracy.** Accuracy is the main metric that measures the overall correctness and efficiency of any model. It is calculated by dividing the number of correctly predicted instances, that includes both true positives and true negatives, by the total number of instances. While accuracy offers a general overview of a model's performance, it can sometimes be misleading in the presence of imbalanced datasets

in which one class is significantly more the other.

$$\textit{Accuracy} = (TP + TN) / TP + TN + FP + FN \quad (4.3.1)$$

- **Precision.** Precision evaluates the precision of positive predictions which is calculated as the ratio of true positives to the aggregation of false positives and true positives. Precision becomes paramount when the expense of false positives is significant, as seen in detection of fraud or diagnoses of medical issues. A high precision value signifies the model's precision in identifying positive instances.

$$\textit{Precision} = TP / TP + FP \quad (4.3.2)$$

- **Recall.** Recall, also sometimes called as sensitivity, is defined as the ability of a model to capture all pertinent instances of a positive class. Calculated by dividing true positives by the sum of false negatives and true positives, recall is important in scenarios where the expense of false negatives is considerable, like in disease detection. A high recall value indicates the model's efficacy in capturing a substantial large portion of positive instances.

$$\textit{Recall} = TP / TP + FN \quad (4.3.3)$$

- **F1 Score.** The F1 score offers a fair evaluation of model's performance and is measured as harmonic mean of recall and precision. It is significantly important in scenarios with imbalanced class distributions. The F1 score spans from 0 to 1, with 1 denoting flawless recall and precision. It helps strike a balance between recall and precision, providing a comprehensive assessment of a model's effectiveness across both aspects.

$$\textit{F1 - Score} = 2 \times \textit{Precision} \times \textit{Recall} / \textit{Precision} + \textit{Recall} \quad (4.3.4)$$

- Precision and recall often have an inverse relationship; improving one may lead to a decline in the other. The F1 score becomes especially important when there is a need to find a balance between precision and recall, ensuring a well-rounded evaluation of the model's performance, particularly in situations where false positives or false negatives carry far more significant impacts. These metrics collectively contribute to a nuanced understanding of a model's strengths and weaknesses, guiding practitioners in refining and optimizing their machine learning models.

In addition, results are also depicted in terms of following graphs:

- **Confusion Matrix.** A confusion matrix is a fundamental tool in machine learning for assessing the performance of classification models. It provides a comprehensive summary of the model's predictions by comparing them against the actual outcomes. The matrix is a square table with rows representing the actual classes and columns representing the predicted classes. In the confusion matrix, the main diagonal contains the true positive (TP) and true negative (TN) values, representing the instances correctly classified. Off-diagonal elements include false positives (FP) and false negatives (FN), indicating instances misclassified by the model. These metrics form the basis for calculating performance metrics such as precision, recall, and the F1 score. The confusion matrix is invaluable for gaining insights into a model's strengths and weaknesses, particularly in identifying specific types of errors. Analyzing its components aids in refining models and optimizing their performance by addressing the challenges posed by misclassifications.
- **Training and Validation Accuracy .** Training and validation accuracy play pivotal roles in assessing the performance of models. Training accuracy is a measure of how well a model performs on the dataset it was trained on. It reflects the proportion of correctly classified instances within the training set, showcasing the model's ability to learn and memorize patterns present in the training data. On the other hand, validation accuracy gauges the model's generalization capability to new, unseen data. This metric is crucial in evaluating whether the model can effectively apply its learned knowledge to previously unseen instances. During the training process, a separate portion of the dataset, not used for training, is reserved for validation. The model's performance on this validation set provides insights into its ability to make accurate predictions on real-world, unfamiliar data. The ideal scenario is to achieve high training accuracy along with high validation accuracy, signifying a model that not only learns well from the training data but also generalizes effectively to new situations. Discrepancies between training and validation accuracy may indicate overfitting, where the model becomes too specialized in the training data and struggles to generalize to new data. Balancing and optimizing both training and validation accuracy are essential steps in developing robust machine learning models.

- **Training and Validation Loss** . Training and validation loss are critical metrics used to evaluate the performance and generalization capabilities of a model. During the training process, the model learns to minimize its training loss, which measures the difference between the predicted outcomes and the actual labels in the training dataset. Lower training loss indicates that the model is effectively capturing patterns in the training data. Validation loss assesses the model's ability to generalize to new, unseen data. It measures the disparity between predicted and actual outcomes in a separate dataset that the model has not encountered during training. The goal is to achieve low validation loss, indicating that the model can make accurate predictions on novel instances. Monitoring both training and validation loss is crucial for preventing overfitting, a situation where the model becomes excessively tailored to the training data but performs poorly on new data. A widening gap between training and validation loss may signify overfitting, highlighting the need for adjustments in the model's complexity or regularization techniques. Balancing the minimization of training loss and the optimization of validation loss is a delicate task. Striking this balance ensures that the model learns from the training data while maintaining the ability to generalize effectively to diverse and unseen instances, fostering the development of robust and reliable machine learning models.

The results of both techniques are discussed in the subsequent paras:-

4.3.1 Application of Linear Support Vector Machine (SVM)

Linear SVM model was applied to the same dataset with different ratios of training and testing: one for 80-20 split and other for 70-30 split.

80-20 Training/ Testing Split

Linear SVM achieved an overall accuracy of **97.7%** with a **1.16%** FN rate (1x malicious samples wrongly identified as benign). Details of results are mentioned below:-

- Total number of binaries in Test Dataset: 86
- Benign binaries: 55

```

Accuracy: 0.9767441860465116
Classification Report:
              precision    recall  f1-score   support

   benign       0.98         0.98         0.98         55
   crypto       0.97         0.97         0.97         31

 accuracy                   0.98         86
 macro avg       0.97         0.97         0.97         86
 weighted avg    0.98         0.98         0.98         86

```

Figure 4.1: Performance Metrics of Linear SVM (80-20 Split)

- Crypto binaries: 31
- Correctly Identified benign binaries (TN) : **54/55 (98.18%)**
- Correctly Identified crypto binaries (TP): **30/31 (96.77%)**
- Binaries identified incorrectly as crypto (FP): **1/86 (1.16%)**
- Binaries identified incorrectly as benign (FN): **1/86 (1.16%)**

Fig 4.1 depicts the overall details of the performance metrics whereas Fig 4.2 shows the Confusion Matrix.

70-30 Training/ Testing Split

Linear SVM achieved an overall accuracy of **97.7%** with a **1.5%** FN rate (2x malicious samples wrongly identified as benign). Details of results are mentioned below:-

- Total number of binaries in Test Dataset: 129
- Benign binaries: 80
- Crypto binaries: 49
- Correctly Identified benign binaries (TN) : **79/80 (98.75%)**
- Correctly Identified crypto binaries (TP): **47/49 (95.92%)**
- Binaries identified incorrectly as crypto (FP): **1/129 (0.78%)**
- Binaries identified incorrectly as benign (FN): **2/129 (1.55%)**

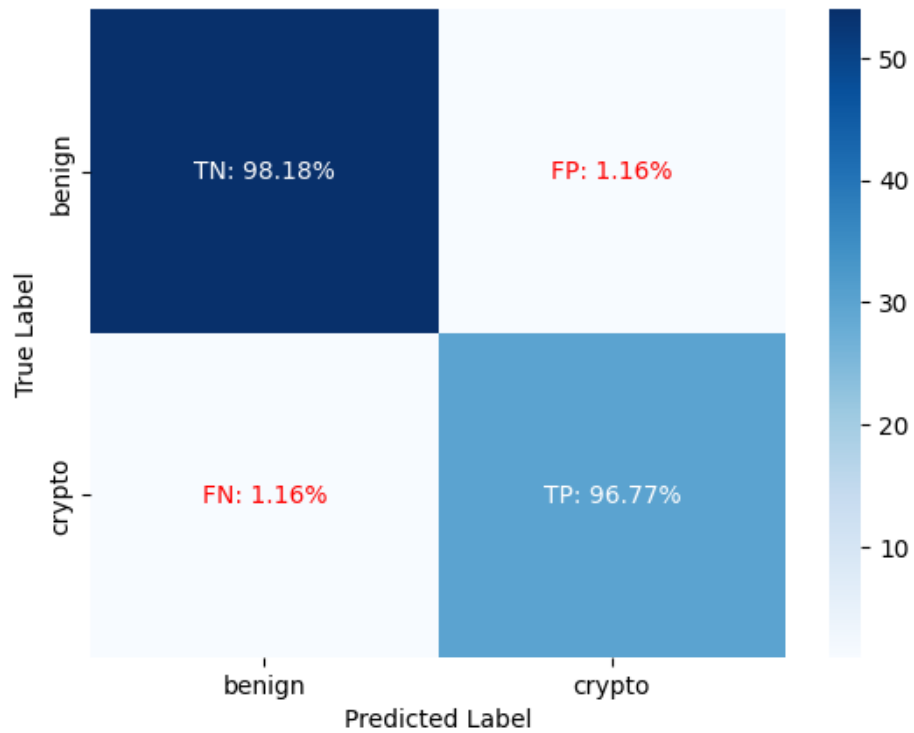


Figure 4.2: Confusion Matrix – Linear SVM (80-20 Split)

Fig 4.3 depicts the overall details of the performance metrics whereas Fig 4.4 shows the Confusion Matrix.

4.3.2 Application of Random Forest (RF)

Similarly, Random Forest was applied to the same dataset with different ratios of training and testing: one for 80-20 split and other for 70-30 split.

```

Accuracy: 0.9767441860465116
Classification Report:

```

	precision	recall	f1-score	support
benign	0.98	0.99	0.98	80
crypto	0.98	0.96	0.97	49
accuracy			0.98	129
macro avg	0.98	0.97	0.98	129
weighted avg	0.98	0.98	0.98	129

Figure 4.3: Performance Metrics of Linear SVM (70-30 Split)

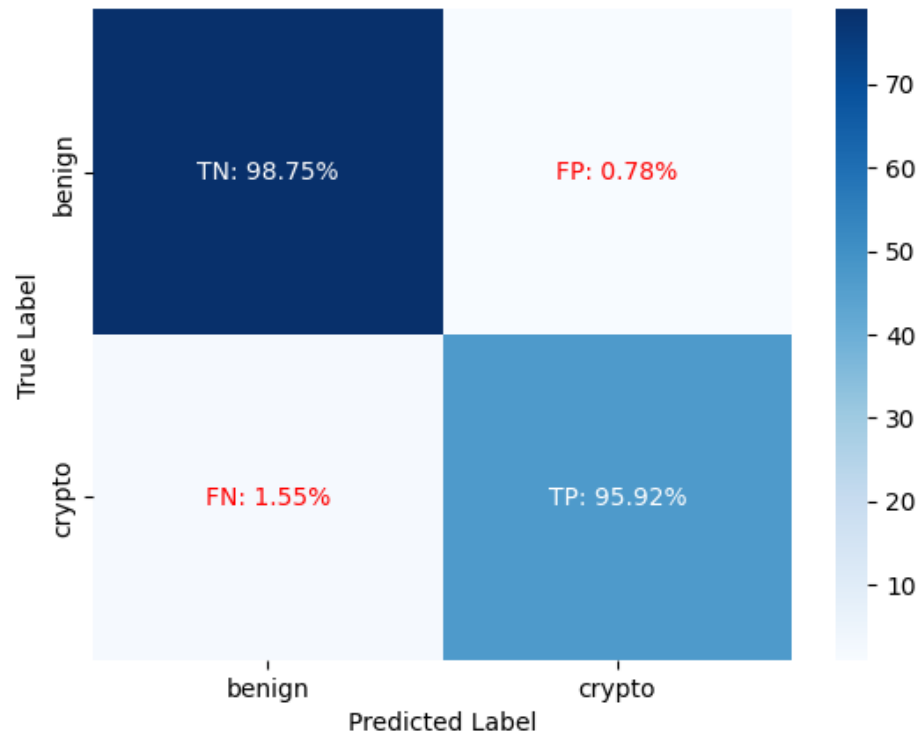


Figure 4.4: Confusion Matrix – Linear SVM (70-30 Split)

80-20 Training/ Testing Split

Random Forest achieved an overall accuracy of **97.7%** with a **1.16%** FN rate (1x malicious samples wrongly identified as benign). Details of results are mentioned below:-

- Total number of binaries in Test Dataset: 86
- Benign binaries: 55
- Crypto binaries: 31
- Correctly Identified benign binaries (TN) : **54/55 (98.18%)**
- Correctly Identified crypto binaries (TP): **30/31 (96.77%)**
- Binaries identified incorrectly as crypto (FP): **1/86 (1.16%)**
- Binaries identified incorrectly as benign (FN): **1/86 (1.16%)**

So the results were almost similar to Linear SVM with 8-20 split. Fig 4.5 depicts the overall details of the performance metrics whereas Fig 4.6 shows the Confusion Matrix.

Accuracy: 0.9767441860465116

Classification Report:

	precision	recall	f1-score	support
benign	0.98	0.98	0.98	55
crypto	0.97	0.97	0.97	31
accuracy			0.98	86
macro avg	0.97	0.97	0.97	86
weighted avg	0.98	0.98	0.98	86

Figure 4.5: Performance Metrics of Random Forest (80-20 Split)

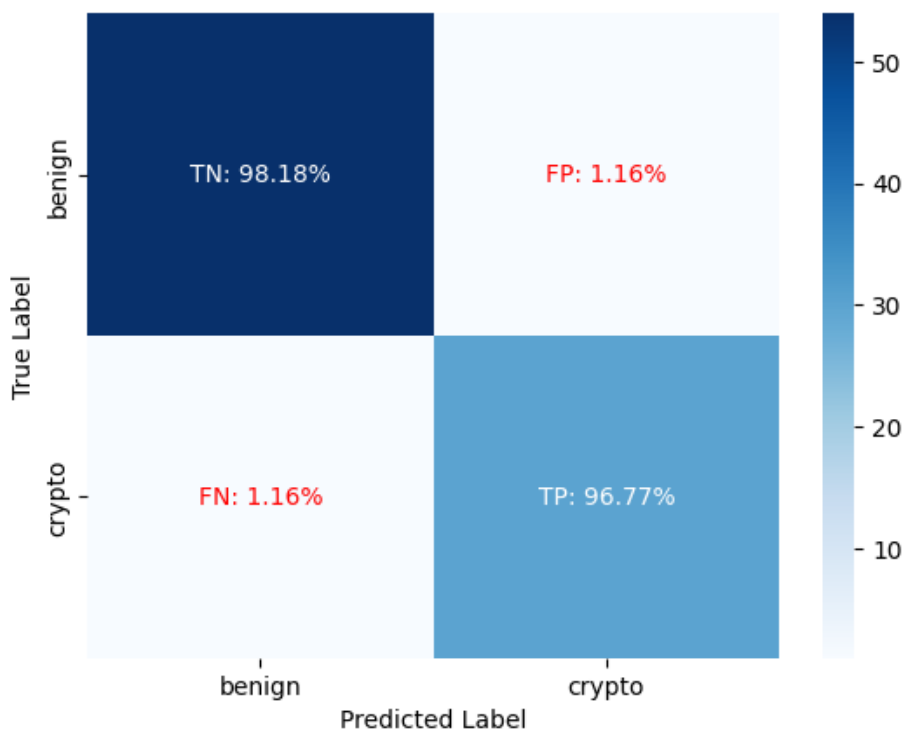


Figure 4.6: Confusion Matrix – Random Forest (80-20 Split)


```

Accuracy: 0.9844961240310077
Classification Report:
              precision    recall  f1-score   support

   benign      0.99      0.99      0.99         80
   crypto      0.98      0.98      0.98         49

 accuracy              0.98         129
 macro avg              0.98         129
 weighted avg           0.98         129

```

Figure 4.7: Performance Metrics of Random Forest (70-30 Split)

70-30 Training/ Testing Split

Random Forest achieved an accuracy of **98.5%** with only a **0.78%** FN rate (1x malicious sample wrongly identified as benign) which is very negligible. Details of results are mentioned below:-

- Total number of binaries in Test Dataset: 129
- Benign binaries: 80
- Crypto binaries: 49
- Correctly Identified benign binaries (TN) : **79/80 (98.75%)**
- Correctly Identified crypto binaries (TP): **48/49 (97.96%)**
- Binaries identified incorrectly as crypto (FP): **1/129 (0.78%)**
- Binaries identified incorrectly as benign (FN): **1/129 (0.78%)**

Overall details of the performance metrics are depicted in Fig 4.7. Fig 4.8 shows the Confusion Matrix. Moreover, Fig 4.9 depicts the overall accuracy achieved while using the RF algorithm, whereas Fig 4.10 elaborates on the loss observed while achieving the accuracy.

The Random Forest Algorithm yielded much better results and accuracy as compared to Linear SVM and therefore was selected as the classifier for our framework. As per our belief, the following factors contributed to the better results achieved by Random Forest:-

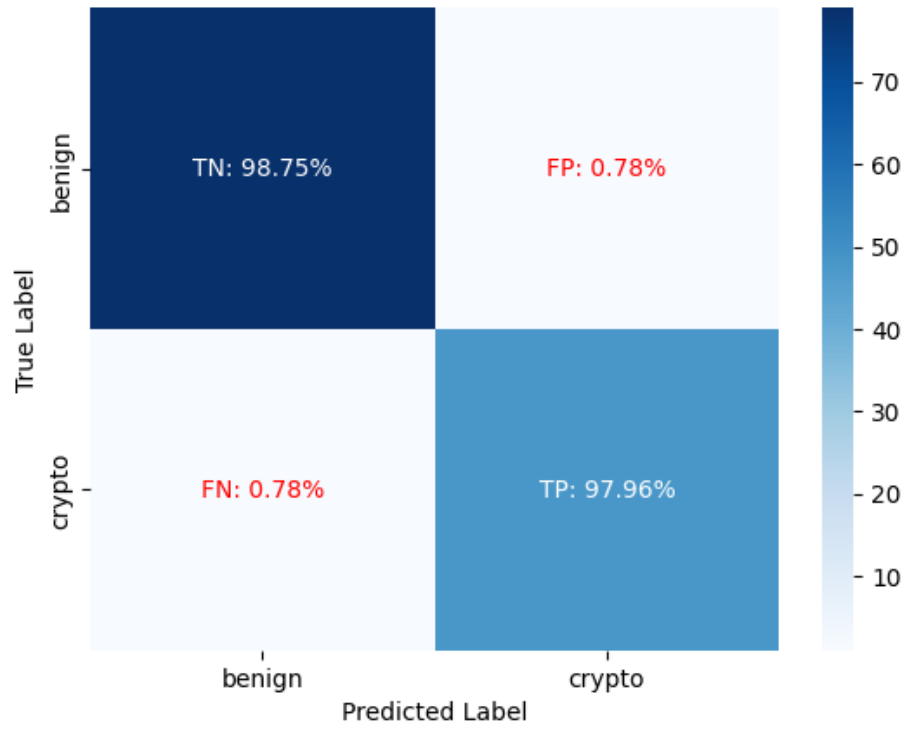


Figure 4.8: Confusion Matrix – Random Forest (70-30 Split)

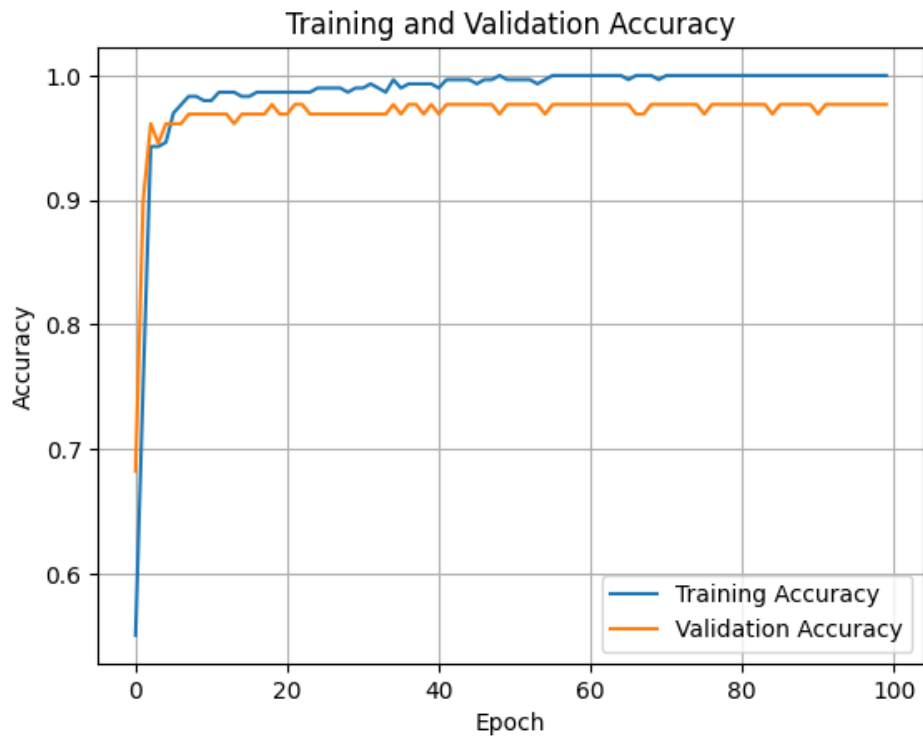


Figure 4.9: Accuracy Achieved in Applying Random Forest Model

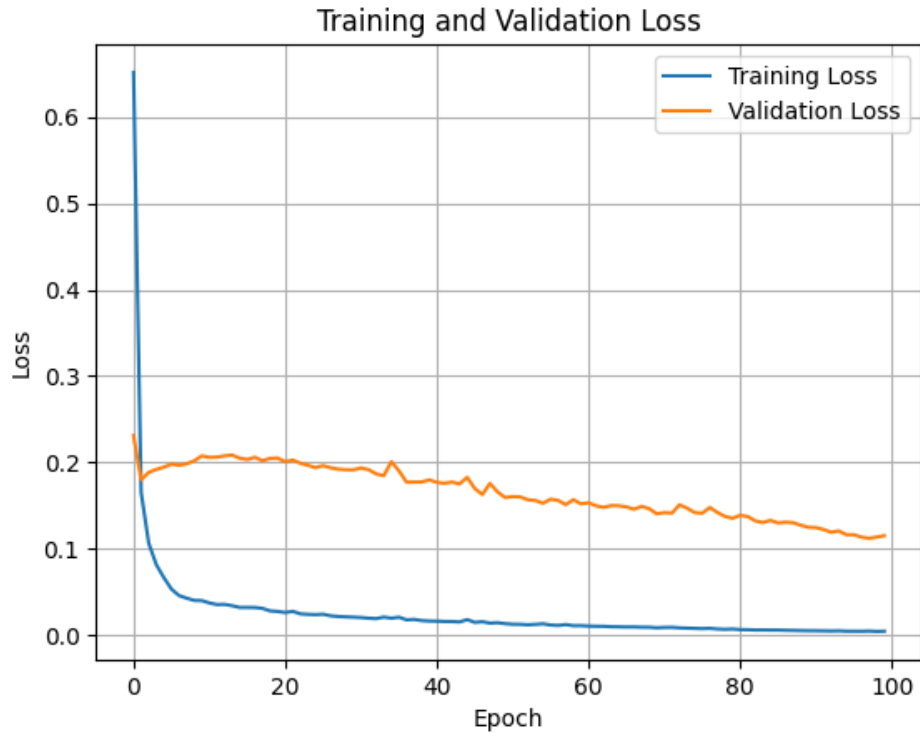


Figure 4.10: Loss Observed in Applying Random Forest Model

- The Random Forest model is better equipped to optimally handle categorical and numerical data.
- It also deals with imbalanced datasets effectively. Since our dataset is imbalanced, having more entries for benign as compared to crypto, so RF model was more suited.
- The Random Forest algorithm is also less sensitive to outliers or anomalies in the dataset due to aggregation of predictions. We also had some very peculiar outlier features result in our dataset which would have been overlooked by Linear SVM Model. Provides high accuracy in both training and testing datasets.

4.4 Comparative Analysis

Comparative analysis involves evaluating and contrasting different elements, such as products, systems, or strategies, to discern similarities, differences, and performance. This method focuses on strengths, weaknesses, opportunities, and threats, facilitating informed decision-making. In business, comparative analysis is pivotal for market re-

Table 4.1: Comparison of Characteristics

Tool Name	Semantics		Strings/ Pattern Matching	Call Flow Graph	Cryptographic Primitives
	Individual Instructions Profiling	Block Level Instrumentation			
MineSweeper	Y	N	N	N	N
MINOS	Y	Y	N	N	N
SEISMIC	Y	N	N	N	N
MineThrottle	N	Y	N	N	N
MinerRay	N	N	N	Y	N
Proposed Framework	Y	Y	Y	Y	Y

search, competitive intelligence, and strategic planning. By examining various options side by side, organizations can identify optimal solutions, improve efficiency, and gain a comprehensive understanding of the factors influencing success in a given context.

This portion of the study aims to present the outcome of the research and evaluate the effectiveness of the framework by comparing the performance metrics, such as detection accuracy and false positive rates. A comparative analysis of our framework is done with other previous tools results as well as with two general-purpose malware detection tools i.e. VirusTotal and Malware Bytes.

4.4.1 Comparison with Other Tools

Feature comparison is a method of assessing and contrasting the characteristics and functionalities of different products, services, or solutions. This analysis helps users or decision-makers make informed choices by highlighting the strengths and weaknesses of each option. Whether evaluating software, gadgets, or services, a feature comparison allows for a side-by-side examination, aiding in the identification of specific attributes that align with individual needs or preferences. This approach streamlines decision-making processes, ensuring that users can prioritize and select offerings that best meet their requirements based on a comprehensive understanding of available features.

- **Comparison of Features.** Most of the tools and research have relied on single or at max two features for detection of malicious cryptojacking binaries. As mentioned earlier, depending on single feature can lead to false negative results and even prone to obfuscation techniques. In MinerWatch we have covered maximum possible features to carry out the static analysis. Comparison is reflected in table 4.1.

Table 4.2: Comparison of Results

Tool Name	Accuracy	FPR	FNR
MineSweeper	N/A	None	N/A
MINOS	98.97%	Low	Low
SEISMIC	98%	Negligible	N/A
MineThrottle	N/A	0%	1.83%
MinerRay	N/A	N/A	N/A
Outguard	97.9%	1.1%	N/A
Proposed Framework	98.5%	0.78%	0.78%

- **Comparison of Results.** Detailed results are not shown by all research authors however we compared our result with whatsoever data was available. Comparison is reflected in table 4.2.

4.4.2 Comparison with VirusTotal & Malware Bytes

- **VirusTotal.** VirusTotal is a widely used and popular online tool that provides a thorough and free of cost analysis of different URLs and files to detect malware as well as other security threats. Developed by Hispasec Sistemas, VirusTotal aggregates antivirus scan engines, threat intelligence services, and various security tools to deliver a holistic assessment of submitted files or links. Users can submit files or URLs to the VirusTotal platform, and it scans the content using over 70 antivirus engines and multiple threat detection tools. The results provide a detailed report on the potential threats detected, including information on the specific antivirus engines flagging the file and additional context such as behavioural analysis. VirusTotal is a valuable resource for individuals, security researchers, and organizations seeking to assess the safety of files before downloading or executing them. The platform facilitates threat intelligence sharing and collaboration within the cybersecurity community. Additionally, VirusTotal offers premium services for enterprises, providing advanced features such as private analysis and custom threat feeds. While VirusTotal enhances cybersecurity efforts, its results should not be totally relied on as false positives or negatives may occur. It is best used as part of a comprehensive security strategy alongside other tools and practices. Over-

all, VirusTotal contributes significantly to the collective defence against evolving cyber threats.

- **MalwareBytes.** Malwarebytes is a prominent cybersecurity company specializing in developing anti-malware and endpoint protection solutions. Founded in 2008, Malwarebytes has become a key player in the fight against malware, ransomware, and other cybersecurity threats. The company's flagship product, Malwarebytes Anti-Malware, employs advanced heuristic detection and behaviour-based analysis to identify and remove malicious software. Malwarebytes provides both free and premium versions of its software, catering to individual users, businesses, and enterprises. The premium version offers real-time protection, scheduled scanning, and a broader range of threat detection capabilities. The company has expanded its product line to include solutions for endpoint security, incident response, and threat intelligence. Known for its effectiveness in detecting and eliminating a wide range of cyber threats, Malwarebytes is trusted by millions of users globally. Its commitment to continuous innovation and adaptation to emerging threats reinforces its position as a reliable cybersecurity solution for users and organizations seeking robust protection against evolving malware landscapes.

Several Web Assemble binaries collected from different sources were scanned using VirusTotal and MalwareBytes for the detection of cryptojacking. Both tools returned considerable FN results, 7x cryptojacking wasm binaries were wrongly identified as Benign by these renowned platforms. The same binaries were also made part of our dataset and then evaluated using our framework, which only failed to identify only 1 binary as malicious leading to negligible False Negative Rate (FNR) in our results.

The above comparison clearly shows the superiority of our proposed framework and that it will further improve the detection of malicious Web Assembly-based cryptojacking applications and prevent such attacks.



Figure 4.11: Result of VirusTotal (False Negatives)

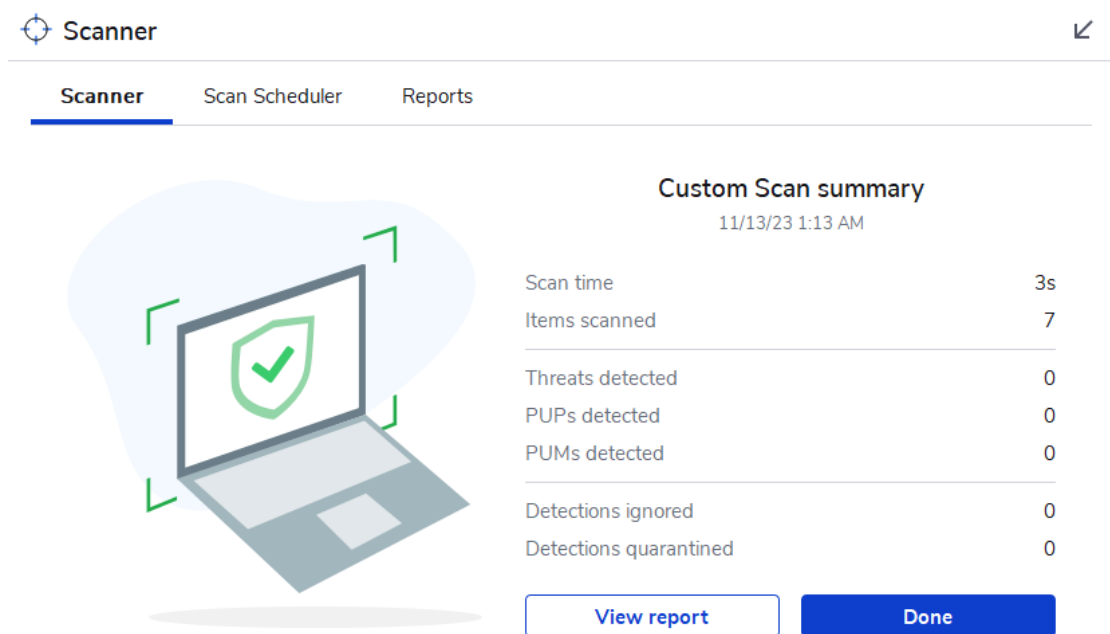


Figure 4.12: Result of MalwareBytes-1 (False Negatives)

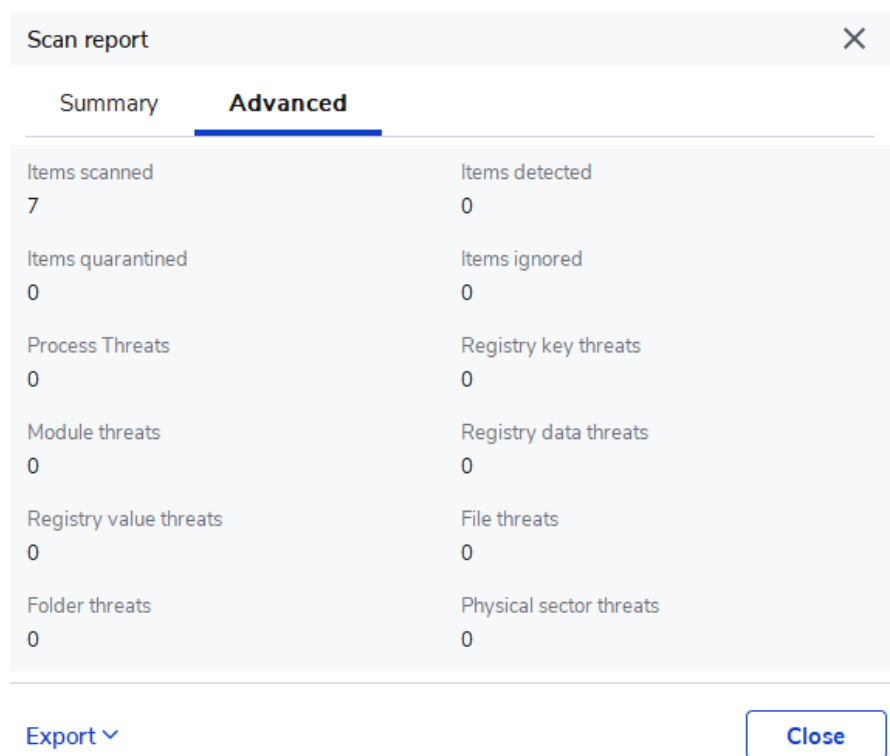


Figure 4.13: Result of MalwareBytes-2 (False Negatives)

Discussion and Future Work

5.1 Discussion

In this section, we throw some light on the strengths and the limitations of our framework in comparison to related works. Furthermore, we also discuss some future works that could improve the efficiency of the framework.

5.1.1 Strengths

Some of the strong points of our framework which make it more efficient are mentioned below:-

- Our framework takes into consideration the intrinsic semantics of Web Assembly binaries which are essential to cryptojacking malware and are difficult to obfuscate, therefore resulting in better results.
- The detection engine is based on multiple static analysis techniques, not only one. Hence, it should cover all possible scenarios and work better against any evasion mechanisms.
- The framework relies on simple calculations/ resources and is not required to be run for longer times as in the case of any dynamic analysis hence incurs little computational overhead.
- A Wasm binary can be detected as either malicious or benign and can be done in a fairly short time.

- It does not require any administrative rights to run and therefore a user with any privilege level can perform the operations.
- It is platform agnostic and therefore can be implemented across any platform.

5.1.2 Limitations

The framework also has some limitations:-

- The framework is purely based on static analysis and does not involve any run-time or dynamic properties of binaries which may result in false negatives.
- For calculating block instructions profiling and call graph similarity, the framework compares the input Wasm binaries against the standard CryptoNight algorithm which is used as a reference. This approach will cover all variants of CryptoNight but might struggle to detect completely different and new cryptojacking implementations.

5.2 Future Work

As future work, following avenues can be explored for further advancements in cryptojacking detection:-

- A combination of both static and dynamic analysis may offer a more effective approach. Therefore, dynamic analysis to cater for high resource (memory/ CPU) utilization, the opening of Web Sockets for communication with the mining pool and the creation of Web Workers threads can be used in conjunction with static analysis techniques for increased efficacy.
- Instead of depending on a single algorithm for detection, different existing algorithms used for cryptomining can be incorporated into the framework to detect all possible variants of cryptomining.

Conclusion

With the increase in usage of web applications and the value of cryptocurrency, cryptojacking has become a prominent threat to be used by malicious actors. The development of Web Assembly has further increased the potency of this threat. In our research, we have tried to propose a novel AI-driven Web Assembly analysis framework to effectively detect cryptojacking attacks. The core of the framework is based on static analysis of Web Assembly binaries that distinguish malicious cryptojacking from benign ones after taking into consideration multiple inherent characteristics of cryptojacking malware. We started our research by carrying out a comprehensive survey of Web Assembly and different tools developed to detect cryptojacking. We then identified key features that would distinguish crypto-based Wasm binaries from benign ones. We also created our own dataset having a numerical representation of extracted features. Two different Machine Learning classifiers were applied to the dataset and the Random Forest algorithm was selected based on the results achieved. Furthermore, in addition to presenting the framework's strengths and limitations, we have also discussed potential future enhancements and research directions for further advancements in cryptojacking detection. The framework showed more promising results when compared with other cryptojacking analysis tools. The results establish that the detection of cryptojacking Web Assembly binaries can be significantly improved to ensure secure web usage and protect against unauthorized cryptocurrency mining. In the end, web security can be enhanced by deploying our proposed framework in a Client-Side Web Application Firewall (WAF) to filter and block cryptojacking Wasm.

Bibliography

- [1] May 2022. URL: <https://www.becomebetterprogrammer.com/why-people-hate-javascript-and-why-you-might-hate-it-too/>.
- [2] Serdar Yegulalp. *What is webassembly? the next-generation web platform explained*. May 2023. URL: <https://www.infoworld.com/article/3291780/what-is-webassembly-the-next-generation-web-platform-explained.html>.
- [3] Feb. 2023. URL: <https://www.lambdatest.com/web-technologies/wasm>.
- [4] URL: <https://caniuse.com/wasm>.
- [5] Uno Platform Team. *The state of webassembly - 2020 and 2021*. June 2023. URL: <https://platform.uno/blog/the-state-of-webassembly-2020-and-2021/>.
- [6] URL: <https://webassembly.org/>.
- [7] Vili-Petteri Niemelä. “WebAssembly, Fourth Language in the Web”. In: (2021).
- [8] URL: <https://webassembly.org/docs/security>.
- [9] Oct. 2023. URL: <https://www.forcepoint.com/blog/x-labs/webassembly-potentials-and-pitfalls>.
- [10] Minseo Kim, Hyerean Jang, and Youngjoo Shin. “Avengers, assemble! Survey of WebAssembly security solutions”. In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE. 2022, pp. 543–553.
- [11] Shrenik Bhansali et al. “A first look at code obfuscation for webassembly”. In: *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 2022, pp. 140–145.
- [12] Shuki Levy. *6 security risks to consider with WebAssembly*. May 2023. URL: <https://thenewstack.io/6-security-risks-to-consider-with-webassembly/>.
- [13] Nakamoto S Bitcoin. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

BIBLIOGRAPHY

- [14] Oct. 2023. URL: <https://en.wikipedia.org/wiki/Monero>.
- [15] Mar. 2018. URL: <https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/>.
- [16] Wenhao Wang et al. “Seismic: Secure in-lined script monitors for interrupting cryptojacks”. In: *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II 23*. Springer. 2018, pp. 122–142.
- [17] Kaspersky. *What is cryptojacking and how does it work?* May 2023. URL: <https://usa.kaspersky.com/resource-center/definitions/what-is-cryptojacking>.
- [18] Catalin Cimpanu. *Coinhive is rapidly becoming a favorite tool among malware DEVS*. Oct. 2017. URL: <https://www.bleepingcomputer.com/news/security/coinhive-is-rapidly-becoming-a-favorite-tool-among-malware-devs/>.
- [19] Said Varlioglu et al. “Is cryptojacking dead after coinhive shutdown?” In: *2020 3rd International Conference on Information and Computer Technologies (ICICT)*. IEEE. 2020, pp. 385–389.
- [20] Faraz Naseem Naseem et al. “MINOS: A Lightweight Real-Time Cryptojacking Detection System.” In: *NDSS*. 2021.
- [21] Javier Cabrera-Arteaga et al. “WebAssembly diversification for malware evasion”. In: *Computers & Security* 131 (2023), p. 103296.
- [22] Biju R Mohan et al. “Comparative Analysis Of JavaScript And WebAssembly In The Browser Environment”. In: *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*. IEEE. 2022, pp. 232–237.
- [23] Yutian Yan et al. “Understanding the performance of webassembly applications”. In: *Proceedings of the 21st ACM Internet Measurement Conference*. 2021, pp. 533–549.
- [24] Quentin Stiévenart, Coen De Roover, and Mohammad Ghafari. “Security risks of porting c programs to WebAssembly”. In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 2022, pp. 1713–1722.
- [25] Ben L Titzer. “A fast in-place interpreter for WebAssembly”. In: *Proceedings of the ACM on Programming Languages* 6.OOPSLA2 (2022), pp. 646–672.

BIBLIOGRAPHY

- [26] Radhesh Krishnan Konoth et al. “Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1714–1730.
- [27] Weikang Bian, Wei Meng, and Mingxue Zhang. “Minethrottle: Defending against wasm in-browser cryptojacking”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 3112–3118.
- [28] Alan Romano, Yunhui Zheng, and Weihang Wang. “Minerray: Semantics-aware analysis for ever-evolving cryptojacking detection”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 2020, pp. 1129–1140.
- [29] Amin Kharraz et al. “Outguard: Detecting in-browser covert cryptocurrency mining in the wild”. In: *The World Wide Web Conference*. 2019, pp. 840–852.
- [30] Daniel Lehmann and Michael Pradel. “Wasabi: A framework for dynamically analyzing webassembly”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 1045–1058.
- [31] *WABT: The WebAssembly Binary ToolKit*. URL: <https://github.com/WebAssembly/wabt>.
- [32] Andreas Moser, Christopher Kruegel, and Engin Kirda. “Limits of static analysis for malware detection”. In: *Twenty-third annual computer security applications conference (ACSAC 2007)*. IEEE. 2007, pp. 421–430.
- [33] *YARA - Pattern Matching*. URL: <https://github.com/davbo/yara-rs/tree/master/sample-miners>.
- [34] Marius Musch et al. “New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*. Springer. 2019, pp. 23–42.