

ANTI-CRYPTOJACKING SYSTEM

BY

ASHER ASHLEY GOH GHIN SHI

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS

AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

© 2025 Asher Ashley Goh Ghin Shi. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of **Bachelor of Information Technology (Honours) Communications and Networking** at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Puan Nor' Afifah Binti Sabri, who has given me this bright opportunity to engage in a cybersecurity project. It is my first step to establish a career in cybersecurity. A million thanks to you. Finally, I must say thanks to my parents, family and friends for their love, support, and continuous encouragement throughout the course.

ABSTRACT

This final year project is about creating a browser-based Anti-Cryptojacking System to protect users from hidden mining attacks. Cryptojacking happens when malicious scripts secretly use a person's computer resources to mine cryptocurrency without permission. The system is developed as a browser extension that can block harmful websites using an updated blacklist, scan webpages for suspicious keywords, functions or scripts, and monitor CPU usage for unusual spikes. It works quietly in the background and alerts users in real time if a threat is detected, while making sure normal browsing is not affected. The extension is built to be lightweight, user-friendly, and easy to install, so even non-technical users can benefit from it. Research for this project involved reviewing current detection tools, studying different prevention techniques, and applying them into a practical extension design. Testing results have shown that the extension can successfully detect and prevent cryptojacking activities, making it a reliable solution for everyday browsing. This project contributes to the field of cybersecurity by providing a simple and effective tool that raises awareness and protects users against cryptojacking attacks.

Area of Study: Cybersecurity

Keywords: Cryptojacking, Browser Extension, CPU Monitoring, Domain Blocking, Web Security

TABLE OF CONTENTS

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
LIST OF ABBREVIATIONS	xi
CHAPTER 1: INTRODUCTION	1
1.1 Problem Statement and Motivation	1
1.2 Objectives	2
1.3 Project Scope and Direction	3
1.4 Contributions	4
1.5 Report Organization	5
CHAPTER 2: LITERATURE REVIEW	6
2.1 Cryptojacking and its Attack Vectors	6
2.2 Detection and Prevention Mechanisms	7
2.3 Previous Works	8
2.3.1 CoinPolice: Detecting Hidden Cryptojacking Attacks with Neural Networks	8
2.3.1.1 Strengths and Weaknesses of CoinPolice	10
2.3.2 Cryptomining Makes Noise: Detecting Cryptojacking via Machine Learning	11
2.3.2.1 Strengths and Weaknesses of the Crypto-Aegis Framework	14
2.3.3 MinerAlert: A Hybrid Approach for Web Mining Detection	15
2.3.3.1 Strengths and Weaknesses of MinerAlert	18
2.3.4 DNS Based In-Browser Cryptojacking Detection	19

2.3.4.1 Strengths and Weaknesses of the System	21
2.3.5 TrustSign: Trusted Malware Signature Generation in Private Clouds Using Deep Feature Transfer Learning	22
2.3.5.1 Strengths and Weaknesses of TrustSign	24
2.4 Summary	25
CHAPTER 3: SYSTEM METHODOLOGY/APPROACH	27
3.1 System Design Diagram/Equation	28
3.1.1 System Architecture Diagram	28
3.1.2 Use Case Diagram and Description	29
3.1.3 Activity Diagram	35
CHAPTER 4: SYSTEM DESIGN	39
4.1 System Block Diagram	39
4.2 System Components Specifications	39
4.3 Circuits and Components Design	41
4.4 System Components Interaction Operations	43
CHAPTER 5: SYSTEM IMPLEMENTATION	45
5.1 Hardware Setup	45
5.2 Software Setup	46
5.3 Setting and Configuration	47
5.4 System Operation (with Screenshot)	51
5.5 Implementation Issues and Challenges	69
5.6 Concluding Remark	70
CHAPTER 6: SYSTEM EVALUATION AND DISCUSSION	72
6.1 System Testing and Performance Metrics	72
6.2 Testing Setup and Result	74
6.3 Project Challenges	79

6.4	Objectives Evaluation	80
6.5	Concluding Remark	82
CHAPTER 7: CONCLUSION AND RECOMMENDATION		83
7.1	Conclusion	83
7.2	Recommendation	84
REFERENCES		86
POSTER		89

LIST OF FIGURES

Figure Number	Title	Page
Figure 1	Graph of battery life over throttling level	8
Figure 2	Confusion Matrix Graph	11
Figure 3	Results for Incoming Traffic	12
Figure 4	Results for Outgoing Traffic	13
Figure 5	AUC Graph	13
Figure 6	Benchmarking Executions from Different Activities	15
Figure 7	Benchmarking Performances in Different States	16
Figure 8	TrustSign's Design Model	22
Figure 9	System Design Diagram	28
Figure 10	System Architecture Diagram	28
Figure 11	Use Case Diagram	29
Figure 12	Activity Diagram (Main Monitoring Loop)	35
Figure 13	Activity Diagram (Checks Outgoing Requests – Domain Blocking)	36
Figure 14	Activity Diagram (Scan Webpage Content)	37
Figure 15	Activity Diagram (Monitor CPU Usage)	38
Figure 16	System Block Diagram	39
Figure 17	System Components	39
Figure 18	Circuit and Components Design	41
Figure 19	System Interaction Operations	43
Figure 20	1 st and 2 nd step	47
Figure 21	3 rd step	48
Figure 22	4 th step	48
Figure 23	5 th step	49
Figure 24	Extension Files	49
Figure 25	Extension's Outlook	50
Figure 26	6 th and 7 th step	50
Figure 27	8 th step	51

Figure 28	<code>fetchAndApplyUrlhausList()</code>	52
Figure 29	<code>parseUrlhausList()</code>	52
Figure 30	<code>updateDynamicRules()</code>	53
Figure 31	<code>calculateCpuUsage()</code>	54
Figure 32	<code>chrome.runtime.onMessage.addListener</code>	55
Figure 33	<code>scanInlineScriptContent()</code>	56
Figure 34	<code>MutationObserver</code>	57
Figure 35	<code>scanScriptsInElement()</code>	58
Figure 36	<code>scanUrlsInElement()</code>	59
Figure 37	Popup Alert for Script Detection	60
Figure 38	Popup Alert for CPU Monitoring	61
Figure 39	Protection Status Section	63
Figure 40	Blocked Domain List Section	64
Figure 41	Current Tab Activity Section (CPU Monitoring)	64
Figure 42	<code>chrome.runtime.sendMessage()</code>	65
Figure 43	<code>updateCpuDisplay()</code>	66
Figure 44	Parts of the code for styles.css	67
Figure 45	Parts of the code for script_test.html	68
Figure 46	Formula for Detection Accuracy	72
Figure 47	Formula for CPU Monitoring Accuracy	73
Figure 48	Formula for Blocking Success Rate	73
Figure 49	Script Detection Test	74
Figure 50	CPU Monitoring Test	75
Figure 51	Domain Blocking Test	76
Figure 52	Popup Interface Test	77

LIST OF TABLES

Table Number	Title	Page
Table 1	Experiment Results on CoinPolice's Performance	9
Table 2	Results on the Accuracy Levels	17
Table 3	Summary of Related Studies	19
Table 4	TrustSign's Dataset	23
Table 5	Comparison Table	25
Table 6	Hardware Setup	45
Table 7	Software Setup	46
Table 8	Testing Results	78

LIST OF ABBREVIATIONS

<i>CPU</i>	Central Processing Unit
<i>TPR</i>	True Positive Rate
<i>FPR</i>	False Positive Rate
<i>DDoS</i>	Distributed Denial-of-Service
<i>VPN</i>	Virtual Private Network
<i>AUC</i>	Area Under the Curve
<i>DNS</i>	Domain Name System
<i>ISP</i>	Internet Service Provider
<i>RAM</i>	Random Access Memory
<i>VM</i>	Virtual Machine
<i>AI</i>	Artificial Intelligence
<i>GPU</i>	Graphics Processing Unit
<i>SSD</i>	Solid State Drive
<i>GB</i>	Gigabyte
<i>TB</i>	Terabyte
<i>API</i>	Application Programming Interface
<i>HTML</i>	HyperText Markup Language
<i>CSS</i>	Cascading Style Sheets
<i>JSON</i>	JavaScript Object Notation
<i>URL</i>	Uniform Resource Locator
<i>N/A</i>	Not Available
<i>ID</i>	Identification

CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Motivation

In this era of globalization, the rapid growth of cryptocurrencies in the current market has attracted many investors, politicians and tech enthusiasts alike due to its decentralized nature and high profitability. Although these two characteristics have benefited a lot of investors and organizations to wealth and success, it has also led to the rise of various cyber threats. One of these cyber threats is known as cryptojacking. Cryptojacking is a type of attack that uses malicious programs to take control over a user's device and its capabilities to mine cryptocurrency illegally without asking for the user's consent [1]. Normally, using traditional malware will require the cyber attacker to install a file or script on the target device in order for the malware to perform its intended function. However, with cryptojacking, these attackers will just need to run the malicious script in the target user's browser while he or she visits a compromised website. This makes it difficult for users to detect any cryptojacking occurrence.

The cryptojacking impacts can be divided into both short term and long term. For short term impacts, cryptojacking can lead to system slowdown, increased fan speed as well as poor responsiveness due to the excessive CPU computational power usage [2]. On the other hand, for long term impacts, it will cause the target's hardware performance to decrease gradually, and this can result in the hardware to have a shorter lifespan [3]. These impacts have not only put us individuals at risk, but also to all the current organizations out there as well. This is because cryptojacking practices often remain undetected by traditional antivirus software [4].

Not only that, but a lot of people do also not have sufficient knowledge on how to detect and prevent such threats in their daily lives [5]. Users might not even notice that their devices have been used for cryptojacking purposes until their devices either breaks down or having severe performance issues. That is why this ever-growing cyber threat has inspired cyber detectives and ethical hackers alike to create a browser-based Anti-Cryptojacking System that not only helps to block known cryptojacking domains but also helps to detect them through abnormal usages of hardware power.

By creating a lightweight browser extension that remains user-friendly, this final year project aims to provide users with the ability to examine and protect their devices and systems against cryptojacking attempts without the need to understand advanced technical coding. The motivation of this project is derived from the increasing number of cryptojacking cases as well as the minimal developments of user-friendly tools for cryptojacking detection and prevention.

1.2 Objectives

The overall goal of this final year project is to create a browser-based Anti-Cryptojacking System that have the necessary functions to detect, alert and prevent any sort of cryptojacking attempts while still ensuring that the system remains user-friendly for all users. The first one is to develop a lightweight and user-friendly browser extension for web browsers. This browser extension will be able to run in the background efficiently in order to defend against cryptojacking activities by identifying certain cryptojacking domains or malicious scripts. It can also help to block these domains before they can exploit the user's hardware resources. The second objective of this final year project is to maintain an up-to-date blacklist of malicious domains that may be related to cryptojacking. This list can either be maintained and updated manually during the development process, or it can be implemented together with any open-source intelligence feeds. The browser extension will then refer back to this blacklist in real time to block any access to these domains.

The third objective of this final year project is to monitor the hardware's CPU usage for suspicious activities. This objective is driven by the undiscovered sources of new, emerging cryptojacking scripts. To address this issue, the browser extension will include a feature that can help to monitor the browser's overall CPU usage. If a certain website consumes too much CPU power for a certain period of time, the browser extension will immediately alert the user and suggest him or her to block the current website being visited. Last but not least, the fourth objective of this final year project is to ensure easy usability and deployment of the browser extension system. One of the goals for this project is to create a system that is also accessible even to non-technical users. The browser extension will have a simple installation and configuration process so that users can use it without needing to know any advanced technical knowledge.

1.3 Project Scope and Direction

This final year project aims to develop a browser-based Anti-Cryptojacking System with a goal to protect end users against cryptojacking attacks. This project involves the creation of a browser extension that is compatible together with Google Chrome. The browser extension will be capable of detecting and blocking cryptojacking scripts or blacklisted domains through an up-to-date blacklist. Not only that, but it will also have the feature to monitor in-browser CPU usage and alert users for potential cryptojacking sites in real-time by displaying user-friendly popup alerts. Moreover, it can run in the background efficiently without using additional resources.

The overall scope of the final year project will not include the feature to detect cryptojacking software as it is way beyond the capabilities of a regular browser extension system. Other than that, the project does not aim to implement any functions related to analysing web traffic as this may violate the privacy and legal aspects.

The direction that has been made for this final year project will include future expansions to be implemented in the browser extension system. Such expansions can be the integration of multiple types of well-known browsers such as Mozilla Firefox or Microsoft Edge, or more advanced analysis methods that are applicable to the browser extension system. However, for now, the current version will only focus on in-browser cryptojacking protection using lightweight and legal methods.

1.4 Contributions

This project is designed to make a meaningful contribution to the field of cybersecurity, with a strong focus on protecting users from cryptojacking attacks. One of the technical contributions is the creation of a fully working browser extension that can detect and block cryptojacking activities. The extension includes several functions such as checking websites or domains against an up-to-date blacklist of malicious sources that may be related to cryptojacking, monitoring CPU usage for unusual spikes and performing scans on the website's source code for any cryptojacking keywords, functions or scripts. By combining these methods, the extension aims to offer stronger and more reliable protection against hidden mining scripts.

Another important contribution is the focus on usability. Many cybersecurity tools can be complicated for non-technical users, which prevents them from being used widely. To address this, the browser extension will have a simple, user-friendly interface that anyone can easily understand and use. Even users without any background in cybersecurity will be able to install the extension and start protecting themselves from these cryptojacking threats. This design choice lowers the barrier for users to take basic steps toward securing their online activities.

In addition to providing protection, the extension will also work to raise awareness about cryptojacking threats. The system will include pop-up alerts that warn users when they visit websites suspected of running cryptojacking scripts. These pop-ups will also help to block the threats that have been detected.

Finally, the extension is being developed with a modular and scalable design. This means it will be easy to add new features in the future without having to rebuild the entire system. For example, later versions could include options for users to create their own lists of trusted websites (whitelists) or to schedule automatic security scans. By building the extension in a flexible way from the start, the project ensures that it can grow and adapt as new cybersecurity needs arise.

1.5 Report Organization

There are seven chapters that can be found in this report. The first chapter will be the introduction and it provides an overall insight of the final year project and explaining the problem statements, objectives, scopes, contributions as well as how the entire report is organized. The second chapter will be the literature review and it reviews existing technologies related to cryptojacking detection systems, comparing each of their strengths and weaknesses. The third chapter will be the system methodology and approach, that includes the system architecture, use case diagrams, and activity diagrams, providing a clear view of how the system is structured and planned. The fourth chapter will be the system design, detailing the block diagrams, component specifications, and the interactions between modules, explaining how each part of the system functions. The fifth chapter will be the system implementation, that includes hardware and software setup, configuration, system operation with screenshots, challenges faced, and concluding remarks. The sixth chapter will be the system evaluation and discussion, that includes testing procedures, performance metrics, challenges, and assessment of how the system meets its objectives. Finally, the seventh chapter will be the conclusion and recommendations, that summarizes the project outcomes together with suggestions on possible improvements for the current system.

CHAPTER 2: LITERATURE REVIEW

2.1 Cryptojacking and its Attack Vectors

Cryptojacking is known to be a malicious activity that secretly uses the target's user hardware to mine cryptocurrency illegally without the user's consent. It uses their hardware's resources illegally to mine for digital currencies, and this can help the attackers to earn money easily without having to pay for additional equipment or electricity [6]. There are two main types of attacks, which are the browser-based and file-based attacks. The browser-based attacks infect the target websites with malicious mining scripts whereas the file-based attacks are performed by installing malware into the hardware's file system, allowing it to mine cryptocurrency even when the browser is closed [7].

Browser-based attacks usually occur through websites with hidden scripts or online advertisements. When a user visits an affected site, the mining script runs automatically in the background, using the device's CPU and memory. These attacks do not require the user to download any files, making them difficult to detect with normal antivirus software [8].

File-based cryptojacking is more persistent. Attackers install malware directly onto the device through phishing emails, malicious downloads, or software vulnerabilities. Once installed, the malware continues mining cryptocurrency even after the browser is closed or the system is restarted. This type of attack can target personal computers, servers, and cloud systems, where attackers can exploit a large amount of computing power for higher profit [9].

A key feature of cryptojacking is its stealth. Attackers often limit CPU usage to avoid slowing down the system, run scripts only when the device is idle, use fileless malware that exists in memory, or hide the code to make it hard to detect. Some attacks even combine browser-based and file-based methods to remain active for longer periods [10].

Because of these attack methods, cryptojacking can reduce system performance and affect the user experience. Devices may become slower, overheat, or consume more electricity than usual, while attackers continue mining cryptocurrency without permission. Understanding these attack vectors is important for developing effective detection and prevention strategies [11].

2.2 Detection and Prevention Mechanisms

The presence of cryptojacking threats has remained consistent over the past few years. The steady increase in such cryptojacking cases have driven numerous researchers and ethical hackers alike to explore more about the mechanisms that can be used for cryptojacking detection and prevention. Some of these include keeping an eye on the system's behaviour, like unusual CPU usage, network activity, or memory changes [12], and machine learning that uses models like decision trees and neural networks to help spot cryptojacking based on system data [13]. Existing browser extensions such as AntiMiner and NoScripts are also used to block mining activities during web browsing [14].

Besides browser extensions like AntiMiner and NoScripts, there are other ways to detect and prevent cryptojacking. One method is **signature-based detection**, which looks for known mining scripts or malware by comparing them with a database of known threats. This works well for threats that are already known, but it cannot detect new or changed scripts that attackers create [15].

Another method is **behavioral or heuristic analysis**. This approach watches how a computer behaves to find unusual activity that could mean cryptojacking is happening. For example, if the CPU or GPU suddenly starts running very fast for a long time, the battery drains quickly, the device gets unusually hot, or the fan runs more than normal, these could all be signs of mining. This method can catch new threats that signature-based detection might miss. Many modern antivirus programs use this type of monitoring [16].

Prevention tools are also important. Extensions like No Coin and MinerBlock block known mining scripts and domains, while ad blockers can help because many attacks come from malicious advertisements. Antivirus programs now often include anti-cryptojacking features that combine signature detection and behavioural monitoring. For cloud-based systems, security tools watch for unusual CPU usage or strange activity patterns to stop unauthorized mining [17].

Even with these protections, cryptojacking can still be hard to stop because attackers keep improving their methods. They may run mining scripts only when the device is idle, limit CPU usage to avoid detection, or use fileless malware that only runs in memory. Because of

this, it is important to use multiple methods together such as monitoring, detection, and prevention to keep devices safe. Security tools also need to be updated regularly to handle new types of cryptojacking attacks.

2.3 Previous Works

This literature review will assess five articles that can help to provide useful information for existing systems of previous work, explaining each of their techniques, strengths and weaknesses.

2.3.1 CoinPolice: Detecting Hidden Cryptojacking Attacks with Neural Networks

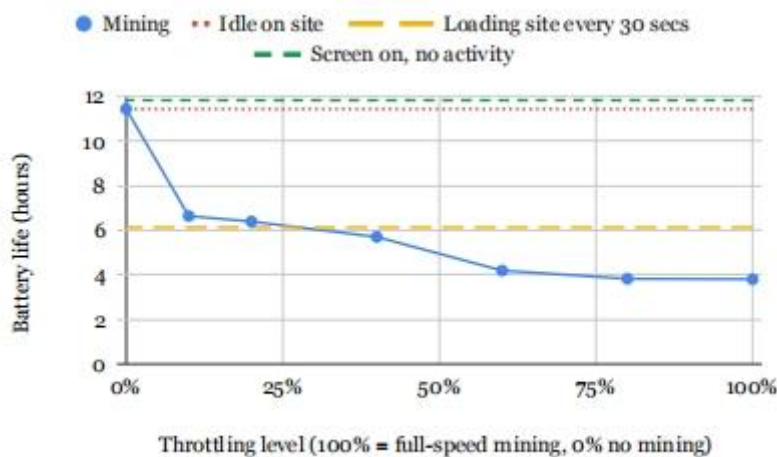


Figure 1: Graph of battery life over throttling level

This article focuses on the problem of cryptojacking, that occurs when hackers will secretly take over the target user's device to mine cryptocurrency illegally without the user's permission. Figure 1 above shows how cryptojacking threats can have a negative impact on a user's device, which in this case, is the battery life of the device [18]. This kind of attack is becoming more and more common these days because cryptocurrencies are known to be valuable, and hackers do not want to pay for the electricity and expensive hardware required to mine these digital currencies.

The main problem that this paper discusses is that cryptojacking has a stealthy nature that enables it to run quietly in the background, making it hard to be noticed by traditional

security tools [18]. Normally, regular antivirus software will try to scan for any known patterns related to malicious activities, but cryptojacking can avoid being detected by using new and hidden scripts that does not match any of the regular known patterns. As a result, a lot of these cryptojacking attacks remain unnoticed for a long period of time, which may cause serious damage to the target user's hardware due to its high draining power.

To overcome this problem, the authors have created a system known as CoinPolice that can counter these cryptojacking attacks based on the usage of neural networks [18]. It is a type of artificial intelligence that can learn to analyse and detect unusual patterns in data. The system works by collecting data such as CPU usage, memory usage and internet activities from a particular device. It will then teach the neural network to detect the differences between normal usage and malicious activities that could be related to cryptojacking. Unlike regular antivirus software that scans data based on known patterns, this method can detect new, unknown cryptojacking patterns by recognizing the data's overall behaviour instead of just its code.

Table 1: Experiment Results on CoinPolice's Performance

Paper	Features	Classifier	Performance
CMTracker [11]	JS signatures	Hand-crafted rules	Not reported
SEISMIC [32]	WASM opcode counts	Hand-crafted rules	98% accuracy, calculated on 12 samples
MineSweeper [18]	Human-assisted static analysis	Hand-crafted rules	Not reported
Saad et al. [28]	Clustering code-complexity features	Fuzzy C-means	96% accuracy and 3.3% false positives, calculated on only 8 samples
OUTGUARD [16]	Signatures and behavioral features of aggressive miners	Support-Vector Machine	97.9% TPR and 1.1% FPR, trained and tested on 36k samples (10% test split) with no throttling randomization.
Coinpolice (this work)	Throttling-independent timeseries behavior	Convolutional neural network	97.8% TPR and 0.74% FPR, trained and tested on 47k samples (10% test split) with randomized throttling levels .

Table 1 above shows the overall experimental results on CoinPolice's performance compared to other models, with a TPR of 97.9% and a FPR of 0.74% with randomized throttling levels [18]. It was able to find out hidden cryptojacking activities that other tools had already missed [18]. This neural network could identify small changes in a system's behaviour, such as high CPU usage or unusual network traffic patterns [18]. This makes it possible for users and organizations alike to protect their devices from these attacks without any high power consumption from the system.

2.3.1.1 Strengths and Weaknesses of CoinPolice

One of the strengths of CoinPolice is that it consists of **an advanced detection method** based on the usage of neural networks. Compared to traditional security methods that rely on known attack patterns, CoinPolice implements machine learning into its system to detect for any unusual behavioural patterns such as abnormal network traffic or high CPU spikes. This enables it to identify unknown or hidden cryptojacking attacks that always remain undetected when using traditional antivirus software. As shown in Table 1 previously, CoinPolice has overtaken other detection models in terms of accuracy and False Positive Rates [18]. Not only that, but it also has a **lightweight and responsive design**, which makes it suitable for a wide range of devices. From Section IV in the article, we can see that several tests have been performed on different types of platforms and CoinPolice managed to maintain a high accuracy level on these platforms without sacrificing its performance [18]. This feature is essential as some of the devices have limited processing power.

However, despite all of its strengths, CoinPolice does come with certain weaknesses. The first weakness is that CoinPolice **relies on large amounts of data to train the neural networks**. These neural networks will need to acquire important datasets that consists of both beneficial and malicious behaviours to learn effectively, and these types of data are not always available in certain environments. This problem is discussed at the methodology section of the article, whereby the authors have talked about the efforts needed to develop a reliable training dataset for the neural networks [18]. The second weakness is that CoinPolice **only focuses on detecting these cryptojacking attacks**, but it **does not have a function to block these attacks in real time**. This problem is highlighted in the discussion section of the article, whereby the authors have agreed that although CoinPolice can send alerts to users of an incoming cryptojacking attack, but it cannot take immediate actions to prevent the attack from happening [18]. This makes the system to be at risk in urgent environments.

2.3.2 Cryptomining Makes Noise: Detecting Cryptojacking via Machine Learning

This article introduces a system called Crypto-Aegis, which uses machine learning to detect unauthorized cryptocurrency mining activities like solo mining, pool mining, and the use of full nodes, which can lead to other attacks like DDoS or Eclipse attacks [19]. The main problem the paper addresses is that cryptojacking attacks often don't leave any obvious signs on the affected device, making them hard to detect. Regular malware detection tools that look for files or changes in the system fail to catch these attacks, especially when traffic is hidden behind encryption or VPNs.

To solve this, Crypto-Aegis focuses on network traffic analysis, as even encrypted communication can still leave patterns or "noise" that can be detected [19]. The system starts by comparing the traffic of known cryptocurrency clients with normal software traffic. Features like packet size which is the size of the data packets sent and interarrival time which is the time between packets are used to train a Random Forest model, and the results are tested using a method called a 10-fold cross-validation [19].

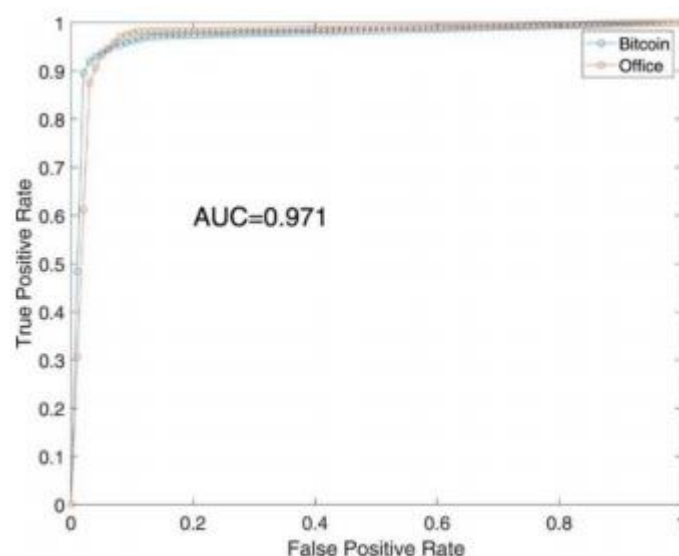


Figure 2: Confusion Matrix Graph

As shown from the graph in Figure 2, the model had a high TPR of 0.941 and a low FPR of 0.059, meaning it could accurately identify Bitcoin traffic without confusing it with normal software traffic [19]. The system also detects full-node activity across multiple

cryptocurrencies like Bitcoin, Monero, and Bytecoin by looking at both incoming and outgoing traffic.

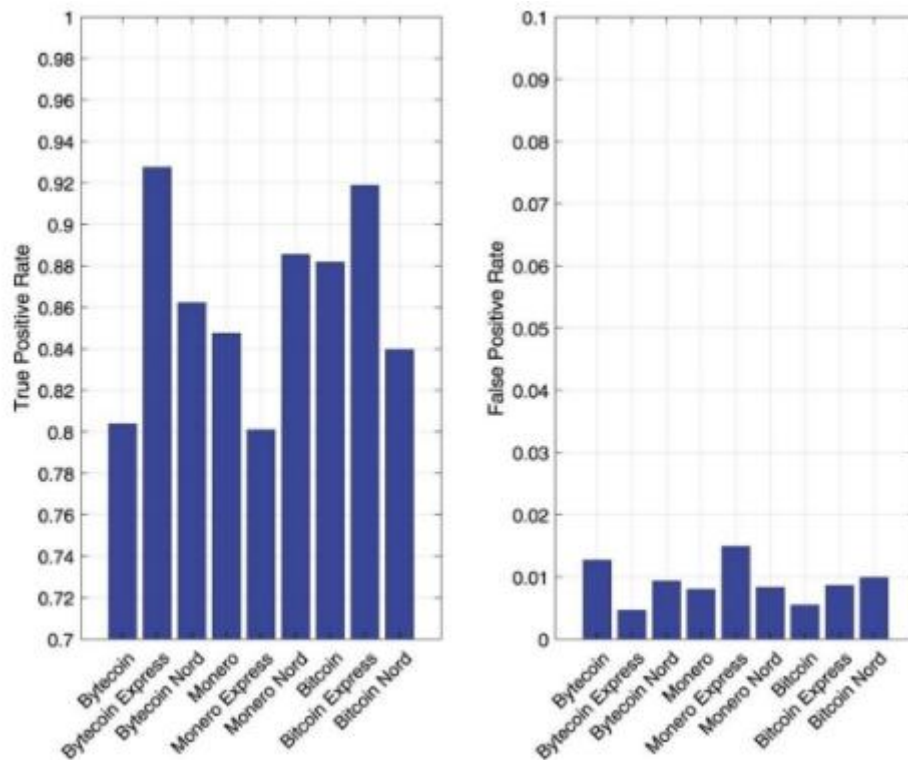


Figure 3: Results for Incoming Traffic

Figure 3 shows the results for incoming traffic, with a TPR of 0.86 and an FPR of 0.0088, with Bitcoin Express and Bytecoin Express scenarios showing the best detection [19].

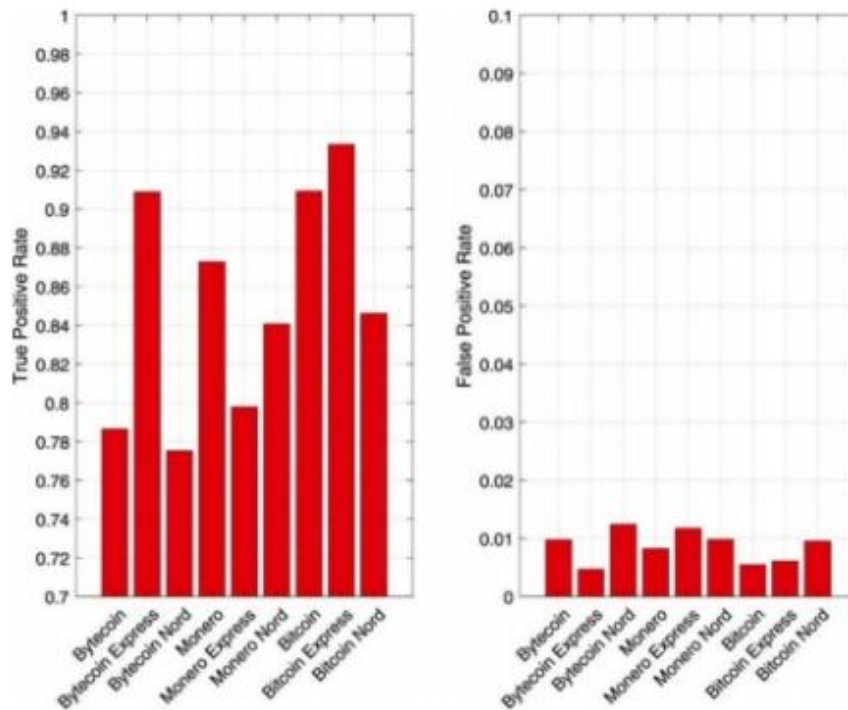


Figure 4: Results for Outgoing Traffic

Similarly, Figure 4 shows that outgoing traffic had a TPR of 0.85 and an FPR of 0.008, indicating good performance in detecting cryptojacking in both traffic directions [19].

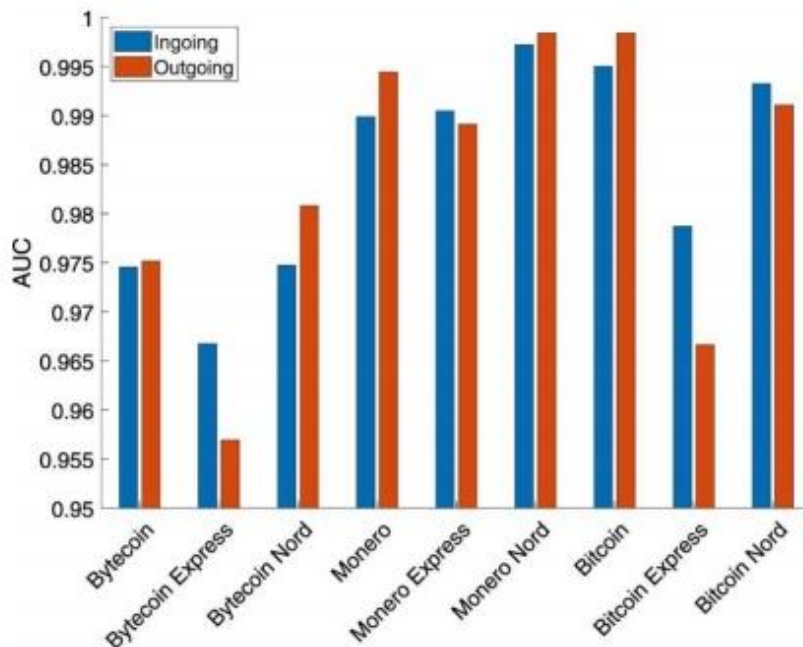


Figure 5: AUC Graph

The system remains effective even when attackers use VPNs to hide their activities. As shown in Figure 5, the model's AUC score stays above 0.955, meaning the system can still detect cryptojacking even if the traffic is encrypted or hidden by a VPN [19]. Key features like average packet size and time between packets that can be calculated over a sliding window are crucial for detection, with packet size being the most important factor.

2.3.2.1 Strengths and Weaknesses of the Crypto-Aegis Framework

One of the main strengths of this framework is that it **doesn't rely on looking at harmful code or scanning websites to detect cryptojacking**. Instead, it watches how the device behaves, such as how high the CPU usage gets and how hot the device becomes during normal use. Because it **focuses on system behaviour**, it can catch both known and unknown cryptojacking attacks, including new ones that have not been discovered, showcasing the overall versatility of the system. Another benefit is that it **doesn't slow down the device**, since it doesn't need to scan files or internet traffic all the time. It can also **work in places where the device isn't always connected to the internet**, which is helpful for offline systems or secure environments.

However, one weakness of this method is the **chance of false positives** whereby it incorrectly thinks something is a cryptojacking attack. For example, if users are using their devices for something that needs a lot of processing power, like playing high-end games, editing videos, or running complex software, the CPU and temperature will naturally rise. This can look like a cryptojacking attack to the system, even though it's safe. As a result, users might receive false alerts. Another issue is that this method has mostly been tested on desktop and laptop computers. It **has not been widely tested on mobile devices** like smartphones or tablets, which are also being targeted by cryptojackers more often. Because of this, the approach may not perform as well or give reliable results on those types of devices in real-world situations.

2.3.3 MinerAlert: A Hybrid Approach for Web Mining Detection

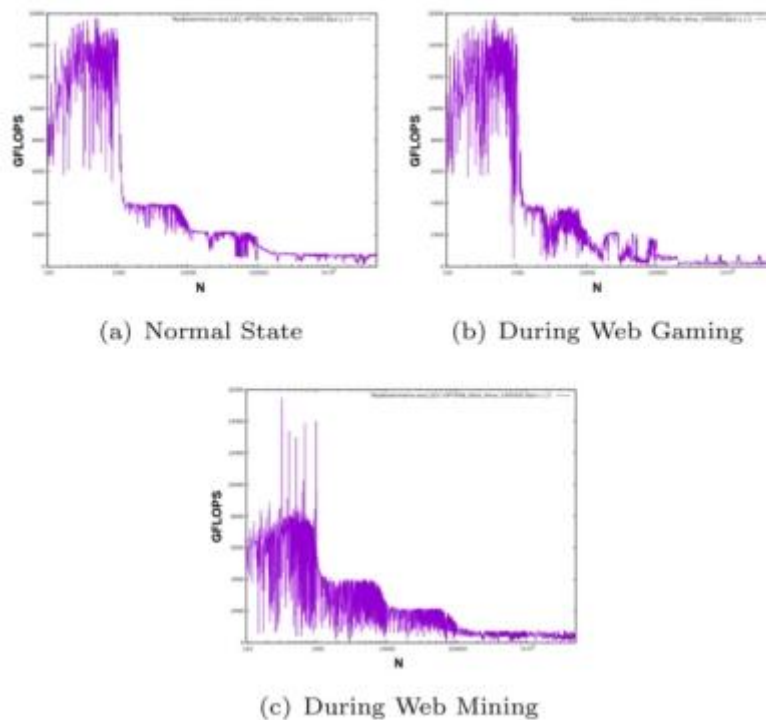


Figure 6: Benchmarking Executions from Different Activities

This article talks about a new way to detect secret cryptocurrency mining on websites, also known as web cryptojacking. This happens when websites secretly run scripts that use a visitor's CPU power to mine cryptocurrency, which can slow down the computer and increase power usage without the user knowing [20]. Figure 6 above shows the differences of the benchmark executions from different activities, highlighting the performance impact caused by high CPU usage. Some websites use these scripts as a way to earn money without ads, but many do it secretly, which causes a big problem in terms of performance and user privacy [20].

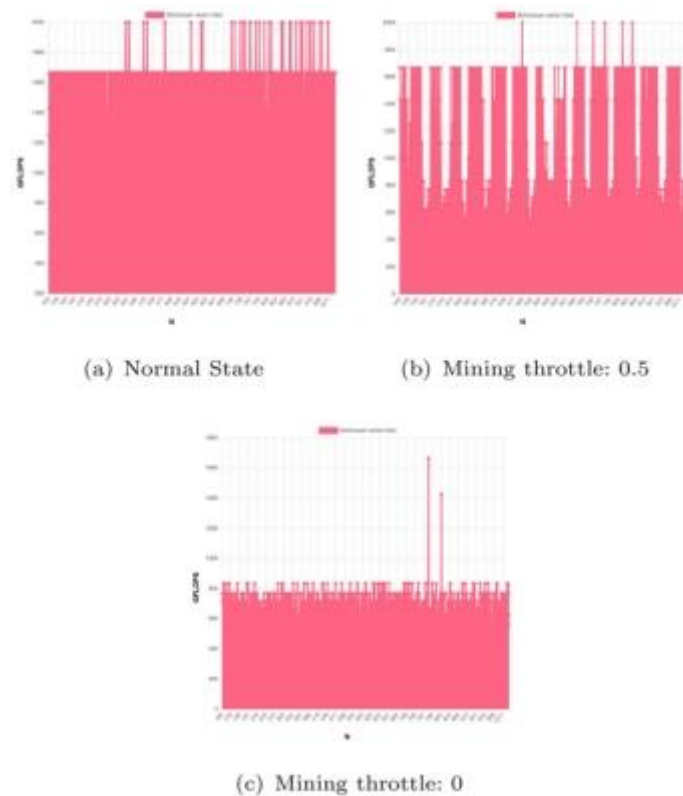


Figure 7: Benchmarking Performances in Different States

The problem the authors are solving is that existing tools like antivirus software and blacklists often fail to catch these mining scripts. That is because these scripts are hidden in complicated code or are constantly changing to avoid detection [20]. This problem is shown in Figure 7 above that compares the overall benchmarking performances between the normal state and the one that involving mining threads, showing how these hidden mining scripts can harm the system's performance. These methods don't always check how scripts behave when they are running, which means some mining scripts can go unnoticed. So, a more advanced and flexible solution is needed to properly catch both known and new types of mining scripts.

To fix this, the researchers created a tool called MinerAlert, which uses both static and dynamic analysis. Static analysis means checking the script's code before it runs to look for patterns that suggest mining [20]. Dynamic analysis means watching how the code behaves when it runs, for example, if it causes high CPU usage or sends data over WebSockets, which are often used in cryptojacking [20]. The tool combines both approaches using machine learning, so it learns from past examples and can detect mining more accurately [20].

Table 2: Results on the Accuracy Levels

Selected features	Linear Kernel	Radial Kernel	Radial Kernel (with parameters scaling)	ROC curves
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message	Best parameter: C=8 CV Accuracy: 97.73%	Best parameter: C=512, gamma=3.0517578125e- 05 CV Accuracy: 95.45%	Best parameter: C=8, gamma=0.5 CV Accuracy: 98.14%	Fig. 4a
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule	Best parameter: C=8 CV Accuracy: 98.55%	Best parameter: C=128, gamma=3.0517578125e- 05 CV Accuracy: 97.30%	Best parameter: C=2, gamma=0.5 CV Accuracy: 99.59%	Fig. 4b
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio	Best parameter: C=2 CV Accuracy: 98.55%	Best parameter: C=128, gamma=3.0517578125e- 05 CV Accuracy: 97.30%	Best parameter: C=2, gamma=0.5 CV Accuracy: 99.59%	Fig. 4c
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio LongestStringLength	Best parameter: C=32 CV Accuracy: 97.10%	Best parameter: C=2, gamma=3.0517578125e- 05 CV Accuracy: 88.21%	Best parameter: C=2, gamma=0.5 CV Accuracy: 99.59%	Fig. 4d
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule defToUseRatio LongestStringLength iframeNumber	Best parameter: C=2 CV Accuracy: 97.10%	Best parameter: C=2, gamma=3.0517578125e- 05 CV Accuracy: 88.21%	Best parameter: C=2, gamma=0.5 CV Accuracy: 99.59%	Fig. 4e
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule LongestStringLength	Best parameter: C=0.5 CV Accuracy: 85.93%	Best parameter: C=8, gamma=3.0517578125e- 05 CV Accuracy: 95.45%	Best parameter: C=2, gamma=0.5 CV Accuracy: 99.59%	Fig. 4f
Standard deviation average mflops dist mean min values duplicates worker web socket web socket message cpuUsage WasmModule LongestStringLength iframeNumber	Best parameter: C=1 CV Accuracy: 91.72%	Best parameter: C=8, gamma=3.0517578125e- 05 CV Accuracy: 95.45%	Best parameter: C=32, gamma=0.125 CV Accuracy: 99.59%	Fig. 4g

The team tested MinerAlert using a mix of normal websites and websites that had mining scripts. The results showed that MinerAlert could detect mining scripts with high accuracy, even when the scripts were hidden or unknown as can be seen from Table 2 above [20]. It also had a low number of false positives, which means that it did not wrongly label many good websites as bad. This shows that using both static and dynamic methods together is better than using just one.

2.3.3.1 Strengths and Weaknesses of MinerAlert

One of the main strengths of MinerAlert is that it was **tested using a real-world dataset** that includes both regular websites and websites that secretly run cryptojacking scripts [20]. This means the system was not only tested in a lab setting, but in situations that reflect how users actually browse the internet. Because of this, the results are more trustworthy and show that MinerAlert can really work in the real world, not just in theory. Another strong point of MinerAlert is that it is **able to detect disguised mining scripts**. Many attackers try to hide or scramble their code so that regular antivirus programs or simple code checkers cannot recognize it. With MinerAlert, it combines two methods which are the static analysis that checks the code itself before it runs and dynamic analysis that watches what the code does when it runs [20]. This means that even if the code is cleverly hidden, if it behaves like a mining script such as causing high CPU usage, MinerAlert will still notice and flag it. This makes the tool much more effective than other tools that only look at the code and not its behaviour.

Despite its many benefits, MinerAlert has a few weaknesses that are important to understand. First, the tool **may not work the same way on every browser**. During the research, the system was tested using a fixed browser environment, most likely one type of browser with a specific configuration [20]. But in the real world, people use many different browsers, like Chrome, Firefox, Edge, and Safari, and mining scripts might behave differently in each one. If MinerAlert relies too much on how a script behaves in just one browser, it may miss out on detecting threats in others. Another weakness is that MinerAlert is **not designed to detect mining in real time**. The dynamic part of the analysis takes time because the system has to run the code, monitor the behaviour, and then decide if it is malicious. This means that if someone visits a website with a mining script, MinerAlert might detect it only after some delay. This makes it less useful for tools like browser extensions that need to block cryptojacking the moment it happens [20]. The researchers even mentioned that real-time detection is something they might work on in the future, showing that it is not currently a feature of the system.

2.3.4 DNS Based In-Browser Cryptojacking Detection

This article proposes a new and effective approach to identify in-browser cryptojacking activities by leveraging DNS traffic data. The authors present a lightweight, scalable detection system that focuses on analysing DNS metadata rather than relying on traditional methods like system-level CPU monitoring, browser fingerprinting, or static code analysis. These traditional methods can be evaded through techniques such as making the malicious codes or scripts hidden, and often require more access to the user environment, which makes them less practical for large-scale deployment [21].

Table 3: Summary of Related Studies

Technique	Ref.	Based On										Method	Datasets		Performance / Results	Limitation
		S	P	M	D	N	C	O	H	DNS	Others		Source	Size		
Static	[16]	x	x	x	x	x	x	✓	✓	x	x	Crawling	Alexa	1.2M	901 TLDs	Unable to handle obfuscation techniques and Memory overhead
	[17]	x	x	x	x	x	x	x	✓	x	✓	Threshold-based	Alexa	853K	2770 TLDs	Detects only hash modeled signatures
	[12]	x	x	x	x	x	x	✓	x	x	x	RF	VirusShare OpenDNS	1K	Acc=>99.0% Recall=99.2% Precision=99.2% TPR=99.2% FPR=0.9%	Performance validated on limited data
	[19]	x	✓	✓	x	✓	x	x	x	x	✓	Crawling	Alexa BlackLists, Public WWW, CoinHive, CryptoLoot, JSEcoin, CoinHive	200K	Profit≈5.5× ↓ CPU≈59× ↑ Temp≈52.8× ↑ Power≈2.0× ↑	Performance and Time overhead
	[11]	x	✓	x	x	x	✓	x	x	x	✓	CNN	Alexa	47K	Acc=98.7% TPR=97.87% FPR=0.74%	Address exclusively browser-based mining
	[20]	x	✓	x	x	x	x	x	x	x	x	TLC, SMO, MISVM, Random SubSpace	Alexa	1.2K	1837 TLDs Precision=1.0% Recall=1.0%	Performance validated on limited data
	[21]	x	✓	✓	x	✓	x	x	x	x	x	K-Means DBSCAN Agglomerative	Hybrid dataset, CIC-IDS2018	-	Precision, Recall, F1-Score = >92.0	Limited mining samples
	[22]	x	x	x	x	✓	x	x	x	x	✓	RF	Self Generated	-	F1-Score=96.0% AUC=99.0% Acc=98.97%	Solely relying on the network traffic
	[13]	x	x	x	x	x	x	✓	x	x	x	CNN	Public WWW	-	Precision=93.07% F1-Score=95.04%	Considers only WASM modules and does not support JS modules
Hybrid	[8]	✓	✓	✓	x	✓	✓	x	x	x	x	Crawling	Alexa	1M	-	Detect only CryptoNight miners, Do not support JS miners
	[14]	✓	✓	x	x	✓	✓	x	x	x	✓	FCM SVM RF	Pixalate Netlab360	5.7K	Acc=96.4% FPR=3.3% FNR=3.7%	Scalability issue, Code obfuscation and WASM are not considered
	[10]	✓	✓	✓	✓	✓	x	x	x	x	x	CNN	Self Generated	1.8K	DR=87.0% DR=99.0% (after 11 sec.)	Address exclusively browser-based mining
	[9]	✓	x	x	x	✓	x	x	x	x	x	Crawling	Alexa, Majestic, Public WWW, [23]	1.8M 48.9M	204 Campaigns 1136 TLDs	Exclusively depends on vulnerabilities of CMS providers such as WordPress

• Based on: ^S Signature, ^P Processor / CPU, ^M Memory, ^D Disk, ^N Network Analysis, ^C Code Analysis, ^O Opcode, ^H Hashing Algorithm, ^{DNS} Domain Name System, ^{Others} Others, • Method: ^{RF} Random Forest, ^{CNN} Convolutional Neural Network, ^{TLC} Two-Level Classification, ^{FCM} Fuzzy C-Means, ^{MISVM} Multiple-Instance Support Vector Machine, ^{SMO} Sequential Minimal Optimization, ^{RandomSubSpace} Random Subspace Method, • x not used, ✓ used, - no specific mention, x times

The problem stated is the difficulty in detecting in-browser mining behaviours in real time without impacting performance or requiring harmful methods. The authors suggest that DNS traffic offers a passive, yet informative, source of behaviour analysis. They observe that domains used for mining purposes often show distinct access patterns and naming features that can be extracted and analysed for detection purposes. Table 3 above supports this by showcasing the summary of the related studies that have analysed such patterns, showing how previous studies have used these features for identifying malicious domains.

To build their detection model, the authors divide DNS-based features into three categories. **Time-based features** include query frequency, time intervals between queries, and overall duration of domain activity in the session. These characteristics help to capture the repeating and consistent access patterns often shown by mining domains. **Graph-based features** are extracted by modelling the DNS traffic as partition graphs, measuring metrics such as the number of nodes which are the domains or clients, edge count, and diameter to highlight irregular structural patterns. **Fixed features** focus on the composition of domain names, such as their length, use of digits, or randomness, which can suggest whether they were auto generated or are part of known mining infrastructure [21].

The system was trained and evaluated using real-world DNS logs. For the supervised learning models, several classifiers were tested, and the Decision Tree model produced the best results. It achieved a recall rate of 59.5%, meaning it correctly identified nearly 60% of the known cryptojacking domains. Precision and F1-score were also evaluated to confirm the model's practical usefulness. However, the results also suggest room for improvement in detecting lesser-known or newly emerging domains [21].

An unsupervised learning approach was also tested, where the model attempted to group domains based on feature similarity without prior labelling. The best performing method here was K-means clustering, which worked effectively even when configured with just two clusters. The ability to separate suspicious domains from regular ones indicates the potential of this method in flagging previously unknown or stealthy mining domains. However, some false positives and overlaps still remained [21].

This article also includes a **case study focusing on Indian government websites**, where DNS logs were analysed from October to December 2021 to check for signs of in-browser cryptojacking. While no direct evidence was found using known mining domain signatures, a group of ten domains with unusual behaviour was identified. These domains showed distinctive query patterns and resource access trends, suggesting they should be monitored for future risk. This shows how effective the approach is for proactive threat monitoring, even in environments that have no indication of any breaches [21].

2.3.4.1 Strengths and Weaknesses of the System

One of the main strengths of this system is its **use of DNS traffic to detect cryptojacking**. Unlike traditional methods that require accessing and monitoring a user's system, this approach only looks at the data sent through the network, specifically DNS queries. This means it doesn't disrupt the user experience or require a lot of system resources, which is a big advantage. Since DNS traffic is available to network administrators without needing extra software on individual machines, it is a **scalable solution** that can be used by large organizations or ISPs to monitor many devices at once. The detection system can run in the background without causing significant slowdowns, and it can be easily added to existing systems that already log DNS data. This makes it a practical and efficient way to identify cryptojacking activities on a large scale without requiring major changes to the current setup.

Despite its strengths, the system also has some limitations in its approach. One of the weaknesses that can be found is **the moderate accuracy of the detection system**. The supervised learning model used in the study has a recall rate of only 59.5%, which means it misses more than 40% of cryptojacking domains. This could be a problem, especially in high-risk environments where it is important to detect every possible threat. The system might also **have trouble identifying new or sophisticated cryptojacking methods**, such as techniques that hide or change the domains used for mining. These methods could allow attackers to avoid being detected by frequently changing domains or mimicking regular traffic patterns. As a result, the system might not be able to detect all forms of cryptojacking, especially if they are well-hidden or newly developed, making it less reliable in certain situations.

2.3.5 TrustSign: Trusted Malware Signature Generation in Private Clouds Using Deep Feature Transfer Learning

This article introduces TrustSign, a smart system made to detect malware, especially a type called fileless malware that hides in a computer's memory rather than being stored as a file [22]. This kind of malware includes cryptojacking, where hackers secretly use someone's computer to mine cryptocurrency. Traditional antivirus systems don't work well with these threats because they usually check files saved on the hard drive. TrustSign solves this by focusing on what's happening inside the computer's RAM instead of files [22].

To do this, TrustSign uses VMs that act as fake computers inside real computers. These virtual machines are then executed by something known as a hypervisor, which controls resources like memory and CPU [22]. The system takes snapshots of the RAM of the VM while malware is running. These snapshots include everything happening in memory, such as active programs and hidden code [22]. Then, the Volatility Framework is used to go through these memory snapshots and find suspicious processes [22].

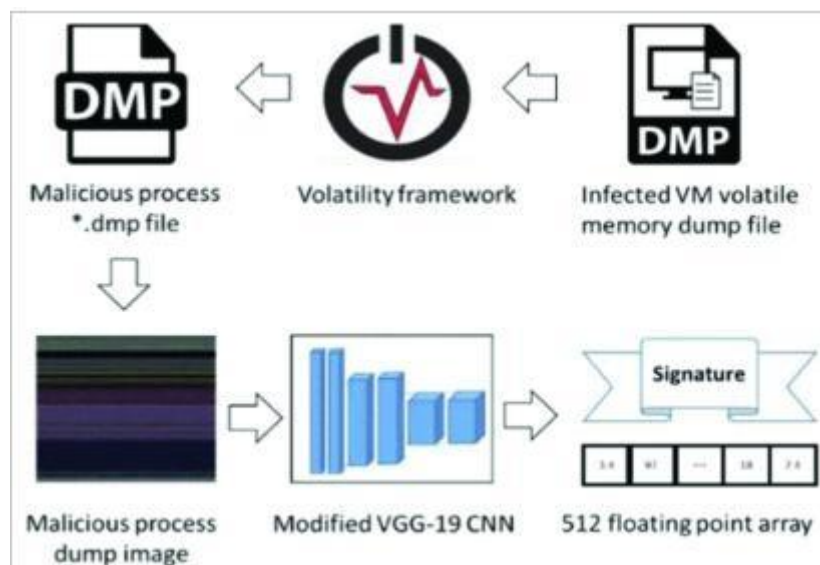


Figure 8: TrustSign's Design Model

After that, TrustSign takes the memory dump files and changes them into 224x224 pixel grayscale images using tools called Dump2Picture and the Pillow Python library [22]. These images basically represent what the malware looks like in memory. Instead of needing a

human to study these images, TrustSign uses a deep learning model called VGG-19, which was originally trained to recognize objects like animals and cars as can be seen from the system's model design in Figure 8. This model is changed slightly by adding a global max pooling layer, so it can better find hidden malware patterns in the images.

The deep learning model then creates a 512-dimensional feature vector which is like a digital fingerprint that represents each type of malware [22]. This process is called transfer learning because it uses a model trained on one task for image recognition and applies it to another for malware detection.

Table 4: TrustSign's Dataset

Type	Name	Instances
Miner	XMRig	100
Miner	XMR-Stak	100
Miner	Coinhive	100
Web Miner	Mineralt	100
Web Miner	Minero	100
Web Miner	Coinhave	100
Web Miner	CoinIMP	100
Ransomware	Vipasana	100
Ransomware	Cerber	100
Ransomware	CryptoLocker	100
Ransomware	ZeroLocker	100
Ransomware	Dircrypt	100
Ransomware	WannaCry	100
Ransomware	Jigsaw	100

To make sure TrustSign works well, researchers tested it using a dataset as shown in Table 4 above that included web-based and desktop cryptojacking scripts like Coinhive, Minero, and CoinIMP. They also tested it on ransomware, such as WannaCry and CryptoLocker, to check if the system could detect more than just cryptojacking [22]. The results showed that TrustSign could successfully identify different kinds of malware by studying memory, even when it couldn't be found using normal antivirus methods [22].

2.3.5.1 Strengths and Weaknesses of TrustSign

One of the strengths that can be found from TrustSign is that it **can catch fileless malware**, like cryptojacking scripts, which many other systems miss. These types of malwares don't store files on the hard drive and instead run directly in the RAM, making them hard to detect. TrustSign solves this by using RAM snapshots, which record everything happening in the memory while the malware is active. This way, even if the malware leaves no trace on the disk, TrustSign can still catch it. Another strength is that TrustSign **uses deep learning** to find malware automatically. It doesn't need human experts to decide what malware looks like. Instead, the system uses a pre-trained deep learning model called VGG-19, which was originally used to identify things like cars and animals but has been trained to look at memory images and find patterns of malware on its own. This makes TrustSign more powerful and faster at identifying new and unknown threats.

However, TrustSign also has some weaknesses. One problem is that **it depends a lot on VMs and taking memory snapshots**. Setting up a VM, running malware inside it, and capturing the RAM content can take time and require a lot of computer power. This makes it less practical for real-time use, especially in systems that need fast and constant protection. Another issue is that TrustSign **only focuses on what's inside the computer's memory**. That means it could miss malware that doesn't run in memory, like malware that attacks through networks or hides in files on the hard drive. Although it's excellent at catching threats that live in RAM, it might not be enough to protect a system completely on its own.

2.4 Summary

Cryptojacking, which secretly uses someone's computer to mine cryptocurrency, has become a serious cybersecurity issue. Many researchers have come up with new ways to find and stop these attacks using machine learning and deep learning methods. The article **CoinPolice** uses neural networks to find hidden cryptojacking by looking at how systems behave over time. In a similar way, the article **Cryptomining Makes Noise** uses machine learning to notice strange system activity that might mean someone is mining coins in the background.

Other researchers have also tried different methods. The article **MinerAlert** mixes two types of code analysis, namely both static and dynamic, to detect if a website is mining in the background. The article **DNS-Based In-Browser Cryptojacking Detection** talks about watching DNS traffic to catch mining scripts running in browsers. Lastly, the article **TrustSign** explains the usage of deep learning and shared features to create trusted malware signatures in private cloud networks. Together, these five studies offer a wide range of ideas and tools to fight cryptojacking, showing many different ways that can be used to protect systems and networks. The comparison table below shows all of the details from the five articles.

Table 5: Comparison Table

Article Title	What the Paper is About	The Problem They Are Solving	How They Solve It	Results & Findings
CoinPolice: Detecting Hidden Cryptojacking Attacks with Neural Networks	A tool called CoinPolice that detects hidden cryptojacking scripts on websites using AI.	Cryptojackers use tricks to hide their scripts, making them difficult to detect with normal security tools.	Uses a deep learning AI model trained on thousands of cryptojacking samples to recognize hidden mining scripts.	CoinPolice detects 97.87% of cryptojacking scripts with very few false positives, making it highly effective.
Cryptomining Makes Noise: Detecting Cryptojacking	A detection method that looks for cryptojacking	Most tools rely only on scanning scripts, but	Uses machine learning to find patterns in network traffic	The method detects cryptojacking with 96%

via Machine Learning	by analyzing unusual network traffic.	cryptojackers find ways to hide, making them invisible to script-based detection.	linked to cryptojacking by analyzing data flow and CPU usage.	accuracy and works even when mining scripts are obfuscated.
MinerAlert: A Hybrid Approach for Web Mining Detection	A browser extension called MinerAlert that detects cryptojacking by checking CPU usage and scanning for mining scripts.	Many cryptojackers frequently change their scripts, making traditional detection methods less effective.	Combines two techniques: scanning scripts for mining code and monitoring CPU usage spikes.	The hybrid approach improves accuracy and reduces false positives compared to single-method detection tools.
DNS-Based In-Browser Cryptojacking Detection	A method that detects cryptojacking websites by analyzing DNS.	Cryptojackers constantly change their website domains, making it hard for traditional detection tools to keep up.	Uses AI to analyze website domain patterns and flag suspicious sites that may host cryptojacking scripts.	Detected many cryptojacking sites, but requires improvement to reduce false positives.
TrustSign: Trusted Malware Signature Generation in Private Clouds Using Deep Feature Transfer Learning	Uses deep feature transfer learning to generate trusted malware signatures for cloud environments.	Difficulty detecting novel malware in cloud systems with traditional methods.	Uses transfer learning to create reliable malware signatures.	Achieved a 30% improvement in detection accuracy and a 10% reduction in false positives.

CHAPTER 3: SYSTEM METHODOLOGY/APPROACH

The browser extension system is developed carefully to make sure it can detect, and block cryptojacking attempts while being easy to use and not slowing down the browser. The main goal of this project is to create a working browser extension that anyone could use, even if they have no technical knowledge. This is important because cryptojacking usually happens quietly in the background, and most users do not notice that their computer is being used for mining cryptocurrencies. Therefore, the system has to work automatically in the background while keeping the user informed in a simple and clear way.

The development follows a modular approach, which means the system is divided into smaller parts with clear responsibilities. This makes it easier to build, test, and maintain. Each part can be worked on and tested individually before combining them into the final system. This approach also makes it easier to fix problems, because errors can be identified in one part without affecting the entire system. By using this method, the system becomes more reliable and manageable during development.

Another important part of the methodology is following Google Chrome's Manifest V3 standards. These rules guide how Chrome extensions work and they also help to make sure that the system can run smoothly on modern browsers. The methodology also included using live threat data from sources like **URLhaus**. This ensures the extension can detect new cryptojacking threats without needing constant manual updates. By following these standards and using live threat data, the system becomes both effective and up to date.

Testing is also key step in the methodology. The system is tested in a safe environment where sample cryptojacking scripts were used to see if the extension can detect them correctly. The CPU monitoring feature is also tested with stress simulations to check if unusual spikes can be detected, even if the threat is not already known. Testing each feature step by step makes the system more stable and trustworthy.

Overall, the methodology focused on creating a system that is **easy to use, reliable, and effective**. By dividing the system into modules, following browser standards, integrating live threat information, and testing thoroughly, the development process ensures that the browser extension can protect users against cryptojacking in a practical and user-friendly way.

3.1 System Design Diagram

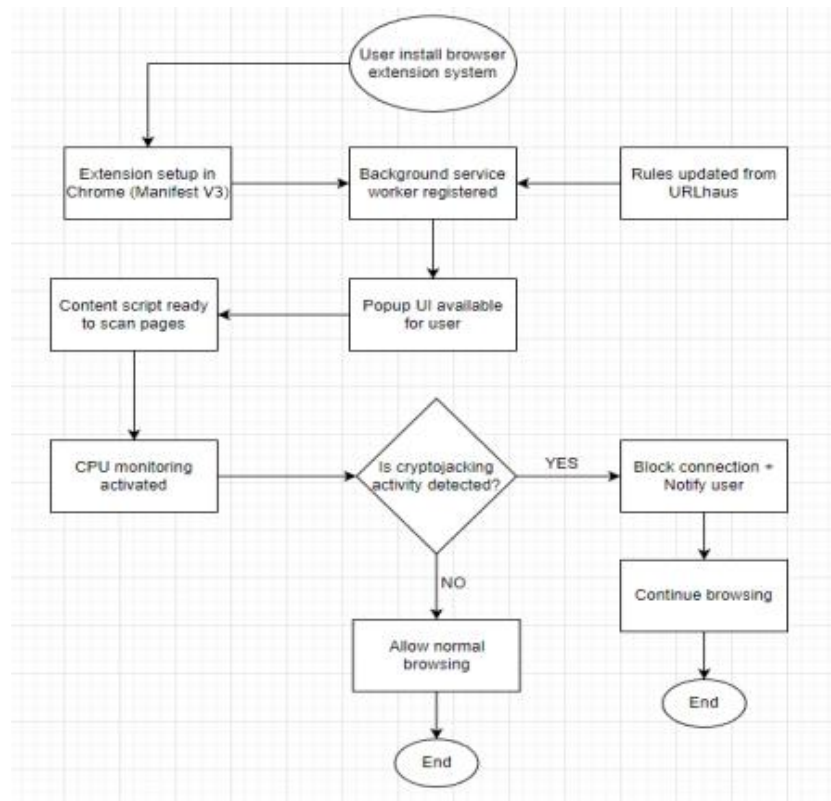


Figure 9: System Design Diagram

3.1.1 System Architecture Diagram

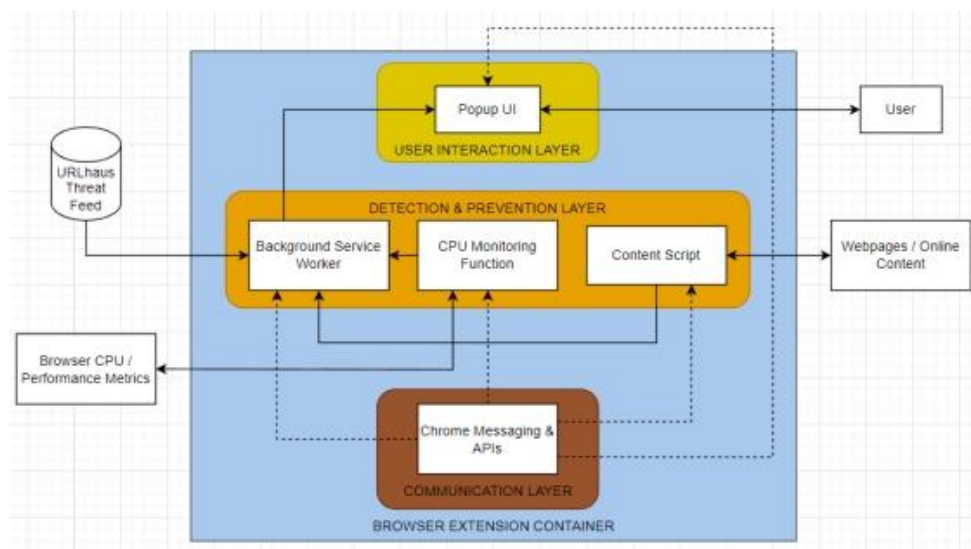


Figure 10: System Architecture Diagram

3.1.2 Use Case Diagram and Description

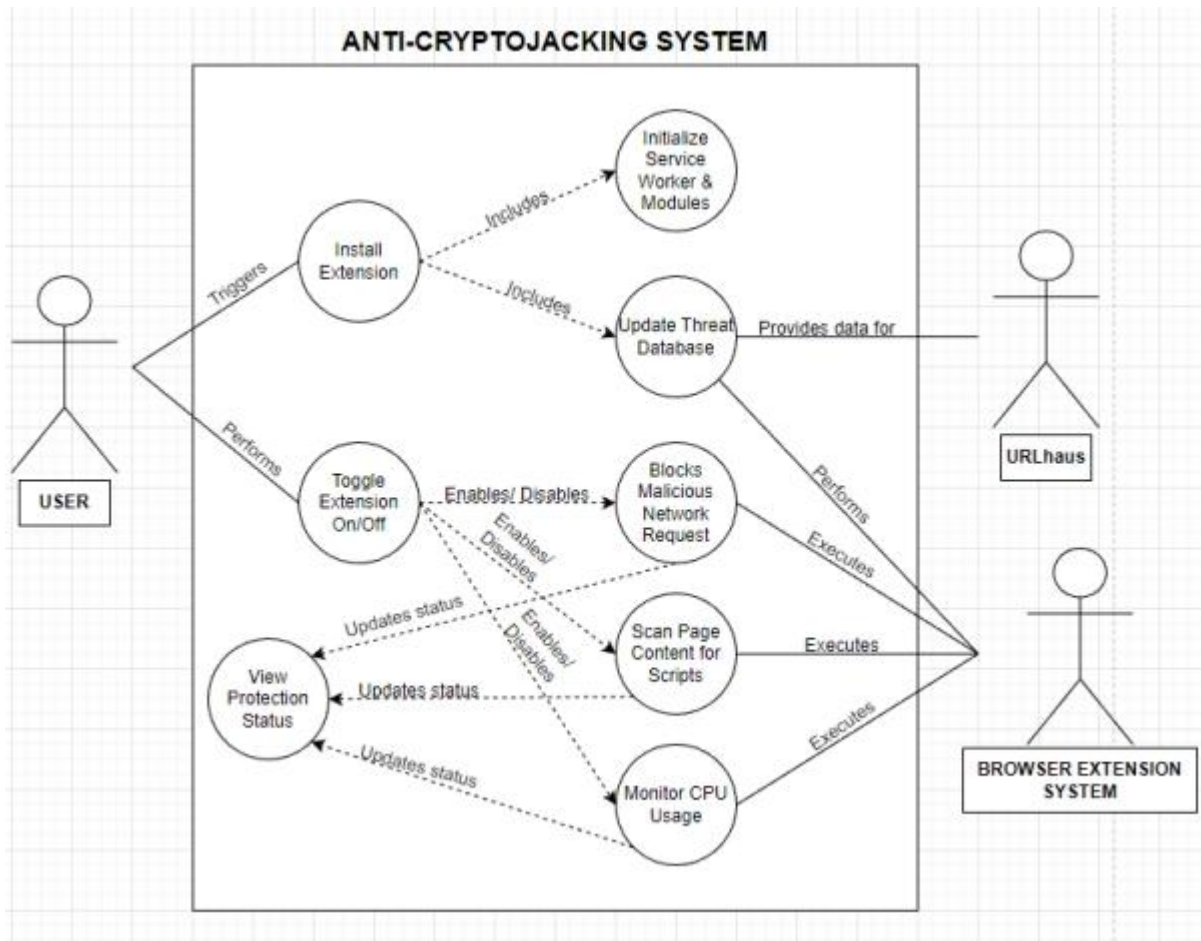


Figure 11: Use Case Diagram

Name	Install Extension
Actor	User
Description	The user installs the browser extension system. This triggers the initial setup process.
Preconditions	The user is using a Chromium-based browser (e.g. Google Chrome).
Postconditions	The extension is installed, configured and actively protecting the user.
Flow of Events	<ol style="list-style-type: none"> 1. User clicks on the “Extension” icon located on the top bar of the Chromium-based browser 2. User clicks on “Manage Extensions” to go to the Extension Settings page and enables the “Developer Mode”.

	<p>3. User clicks on “Load Unpacked” and selects the path to the browser extension system’s coding files for installation.</p> <p>4. Include: Update Threat Database</p> <p>5. Include: Initialize Service Worker & Modules</p> <p>6. Extension icon appears in the browser’s toolbar (from the “Extensions” icon located on the top bar of the Chromium-based browser), indicating successful installation.</p> <p>7. User pins the installed extension for easier navigation.</p>
--	---

Name	Initialize Service Worker & Modules
Actor	Browser Extension System (Browser Runtime)
Description	After installation, the browser initializes the extension. This involves registering the background service worker, which helps to coordinate the activation of all other modules (content script, CPU monitoring).
Preconditions	Extension has been successfully installed.
Flow of Events	<ol style="list-style-type: none"> 1. Browser loads the extension manifest (Manifest V3). 2. Browser registers the background service worker defined in the manifest. 3. The service worker executes its initialization routine. 4. Triggers: Update Threat Database 5. It also sets up listeners for network requests, runtime messages, and CPU monitoring.

Name	Update Threat Database
Actor	Browser Extension System (Background Service Worker)
Supporting Actor	URLhaus (External System)
Description	The system connects to the specific URLhaus hostlist endpoint (https://urlhaus.abuse.ch/downloads/hostfile/) to fetch a blocklist in the

	“ hosts ” file format. It processes this plaintext data to extract domain names and converts them into a blocking rule format compatible with the Chrome’s declarativeNetRequest API.
Preconditions	Extension is installed and has an internet connection.
Postconditions	The extension’s internal rule set is dynamically updated with the latest known malicious domains without requiring a browser restart. The “Blocked Domains (From URLhaus)” section in the popup interface is also updated to show the latest blocklist, together with the number of active blocking rules from the blocklist and the last updated time.
Trigger	Extension installation and periodic update checks (e.g. once every 24 hours).
Flow of Events	<ol style="list-style-type: none"> 1. Fetch list: The system downloads the plaintext blocklist from https://urlhaus.abuse.ch/downloads/hostfile/. 2. Clean data: The system removes all comment lines and empty lines from the plaintext blocklist. 3. Extract Domains: For each remaining line, the system extracts just the domain name. 4. Convert Rules: The system converts the list of domains into browser-readable blocking instructions. 5. Apply Rules: The system instantly updates its active blocklist with the new rules.

Name	Toggle Extension On/Off
Actor	User
Description	The user can enable or disable the entire extension’s functionalities with toggle switches for each function in the popup interface.
Preconditions	The extension is installed, and the popup is open.
Postconditions	The scanning, blocking and monitoring functions are either activated or deactivated based on the user’s choice.
Flow of	<ol style="list-style-type: none"> 1. User opens the extension’s popup interface in the browser’s

Events	<p>toolbar.</p> <ol style="list-style-type: none"> 2. User clicks on the toggle switches for all 3 functionalities. 3. System changes its state and displays the new status (e.g. “Domain Blocking: INACTIVE”, “Domain Blocking: ACTIVE”) 4. All monitoring, scanning and blocking modules are paused or resumed accordingly.
---------------	--

Name	Blocks Malicious Network Request
Actor	Browser Extension System (Background Service Worker)
Description	The system automatically checks all outgoing network requests against the updated blocklist. If a match is found, the request is blocked immediately.
Preconditions	The extension is installed and the “Domain Blocking” function is active (not toggled “INACTIVE” by user).
Postconditions	The searched domain is blocked by the browser extension system.
Flow of Events	<ol style="list-style-type: none"> 1. User visits one of the domains in the updated blocklist from the “Blocked Domains (From URLhaus)” section in the system’s popup interface. 2. The system then checks the requested URL against the blocklist. 3. A match is found. 4. The system blocks the request. 5. The user will not be able to access the searched domain.

Name	Scan Page Content for Scripts
Actor	Browser Extension System (Content Script)
Description	The system scans the HTML and JavaScript source code of the webpage for patterns, keywords and functions strongly associated with cryptojacking scripts (e.g. coinimp, cryptoloot).
Preconditions	The extension is installed and the “Script Detection” function is active

	(not toggled “INACTIVE” by user).
Postconditions	An inline keyword, function or hidden cryptojacking script is identified.
Flow of Events	<ol style="list-style-type: none"> 1. User opens the script_test.html file from the browser extension system’s coding files as a webpage in Google Chrome to test out the “Script Detection” function. 2. The system will then inject the content script into the webpage, 3. Content script executes and scans the DOM and script contents. 4. Suspicious keywords, functions or scripts are detected. 5. The content script sends a message to the background service worker through the runtime messaging API. 6. The background service worker logs the event and takes action by creating a popup alert at the bottom right corner of the webpage, displaying the things that have been detected from the webpage and gives the user an option to either close the current webpage or dismiss the popup alert.

Name	Monitor CPU Usage
Actor	Browser Extension System (CPU Monitoring)
Description	The system continuously monitors the browser’s CPU usage percentage. If it detects a sudden spike in CPU consumption, it triggers a popup alert, saying that the current webpage might be consuming the user’s resources for background mining activity.
Preconditions	The extension is installed and the “CPU Monitoring” function is active (not toggled “INACTIVE” by user).
Postconditions	A popup alert will be triggered, depending on the current browser’s CPU usage percentage.
Flow of Events	<ol style="list-style-type: none"> 1. User goes to a CPU Stress Test Simulation webpage in Google Chrome to simulate the high CPU usage percentage. This will trigger the background service worker to take action. 2. User opens the popup interface to check and observe the increase

	<p>in the CPU usage percentage.</p> <ol style="list-style-type: none"> 3. If the CPU usage percentage reaches 70% or higher, a popup alert will be created at the bottom left of the current webpage by the background service worker, stating that the webpage might be consuming significant resources, together with the recorded CPU usage percentage. 4. The popup alert also gives the user an option to either close the current webpage or dismiss the popup alert.
--	---

Name	View Protection Status
Actor	User
Description	The user clicks on the extension icon in the browser's toolbar to open the popup interface. The popup will display the current status of all the 3 functions, including whether it is active, the updated blocklist in the "Blocked Domains (From URLhaus)" section and the current CPU level in the "Current Tab Activity" section.
Preconditions	The extension is installed.
Postconditions	The user is informed of the system's operational status, as well as the updated blocklist and the current CPU level.
Flow of Events	<ol style="list-style-type: none"> 1. User clicks on the extension icon. 2. The system will then display the popup UI. 3. The popup shows the current protection status of each function (e.g. "Script Detection: ACTIVE"), the updated blocklist and a CPU usage indicator. 4. User observes the information in the popup UI and closes the popup.

3.1.3 Activity Diagram

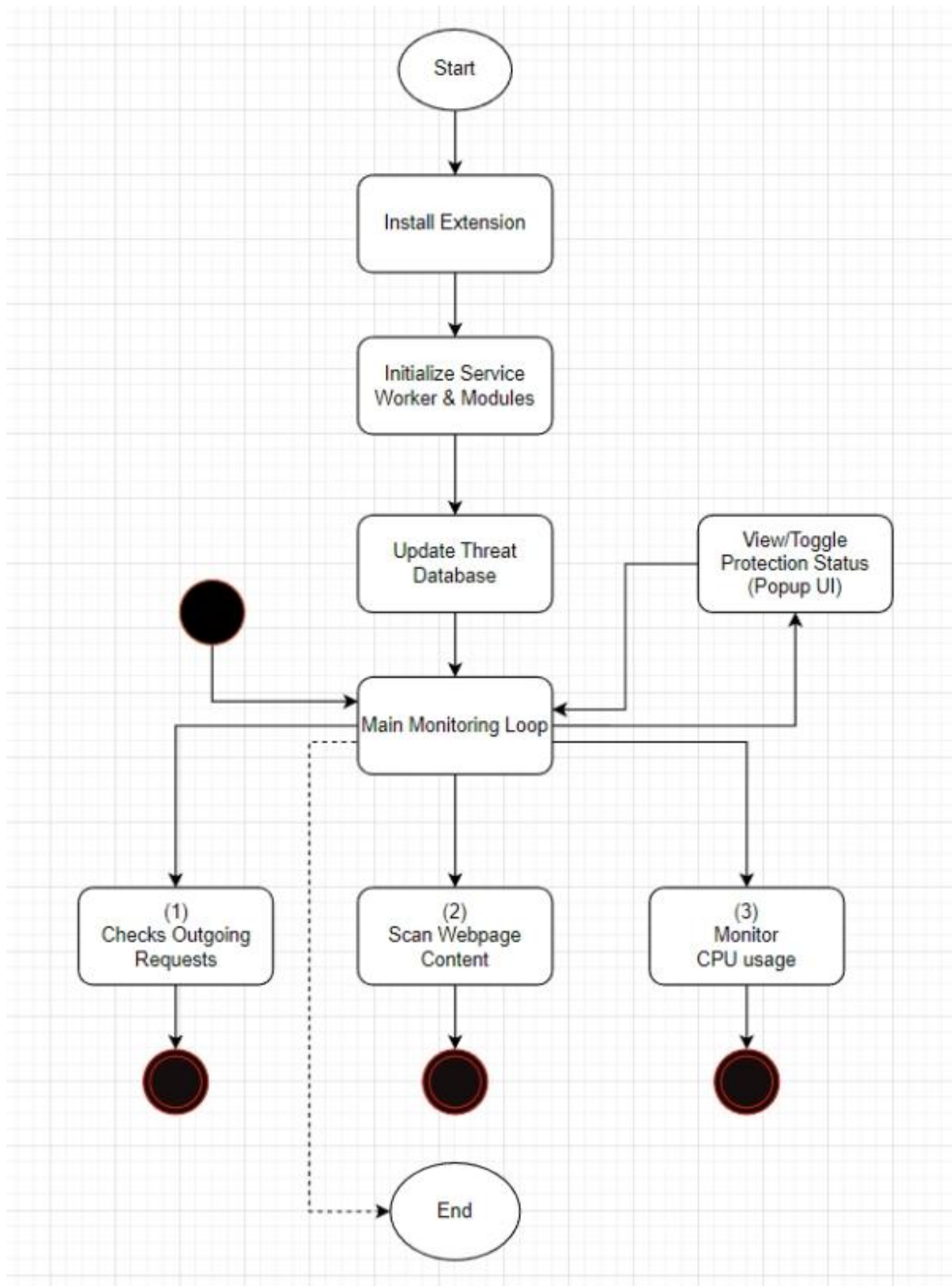


Figure 12: Activity Diagram (Main Monitoring Loop)

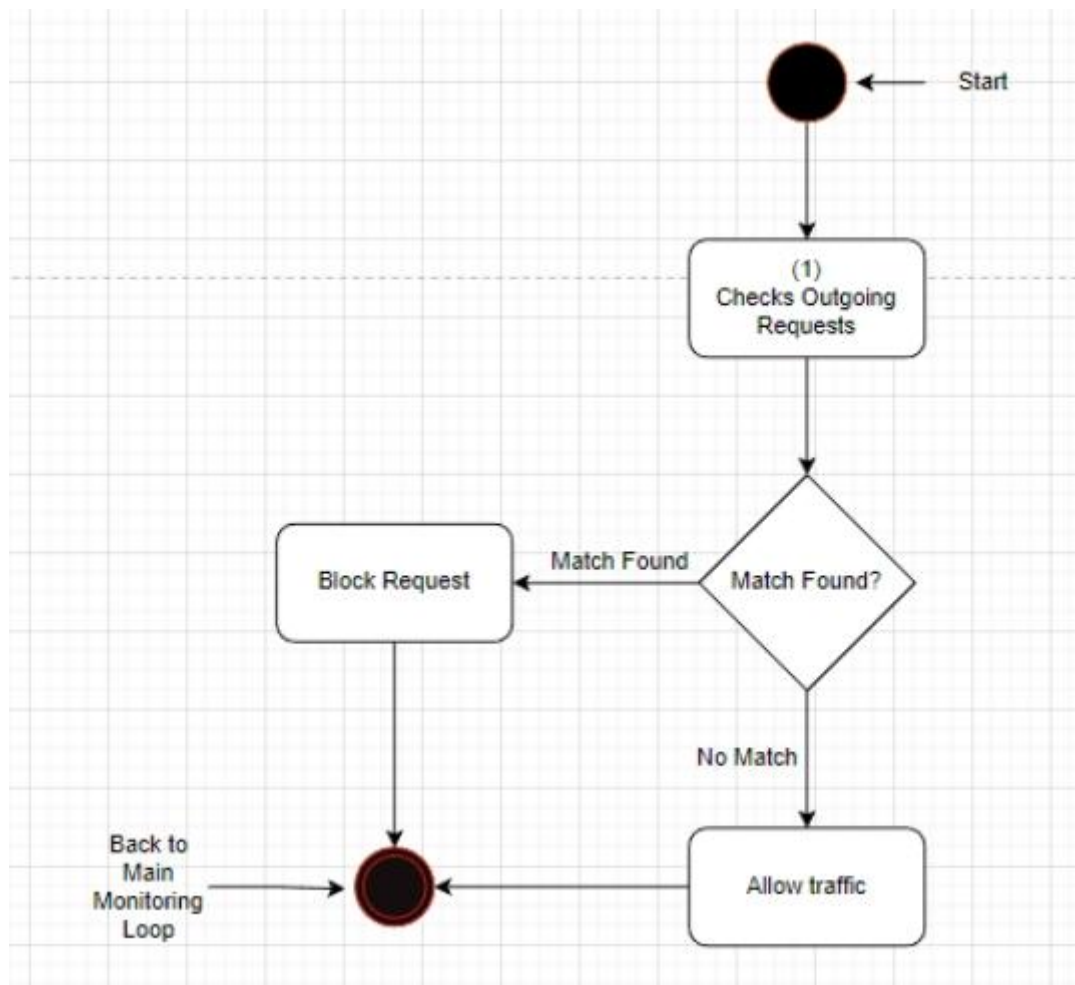


Figure 13: Activity Diagram (Checks Outgoing Requests – Domain Blocking)

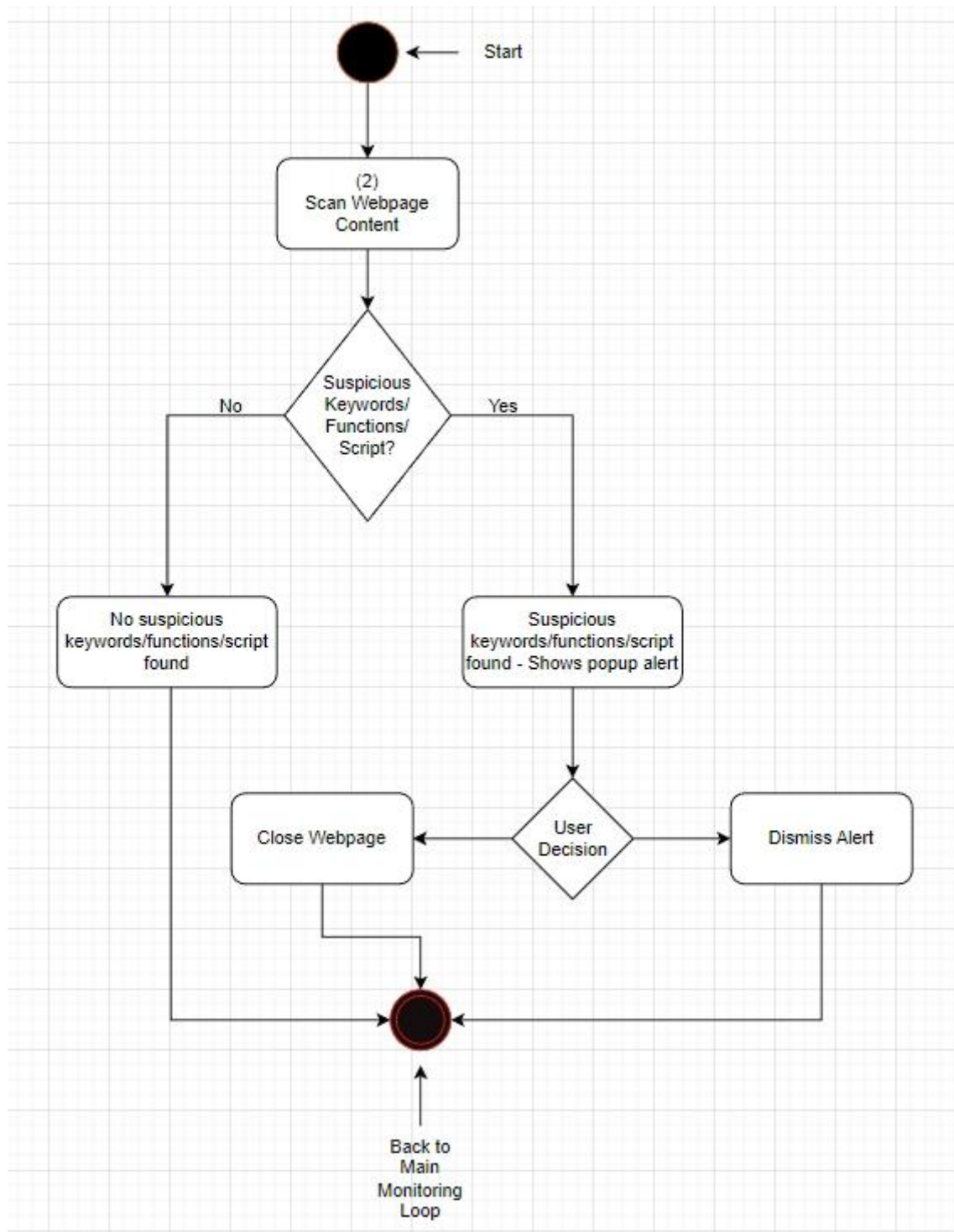


Figure 14: Activity Diagram (Scan Webpage Content)

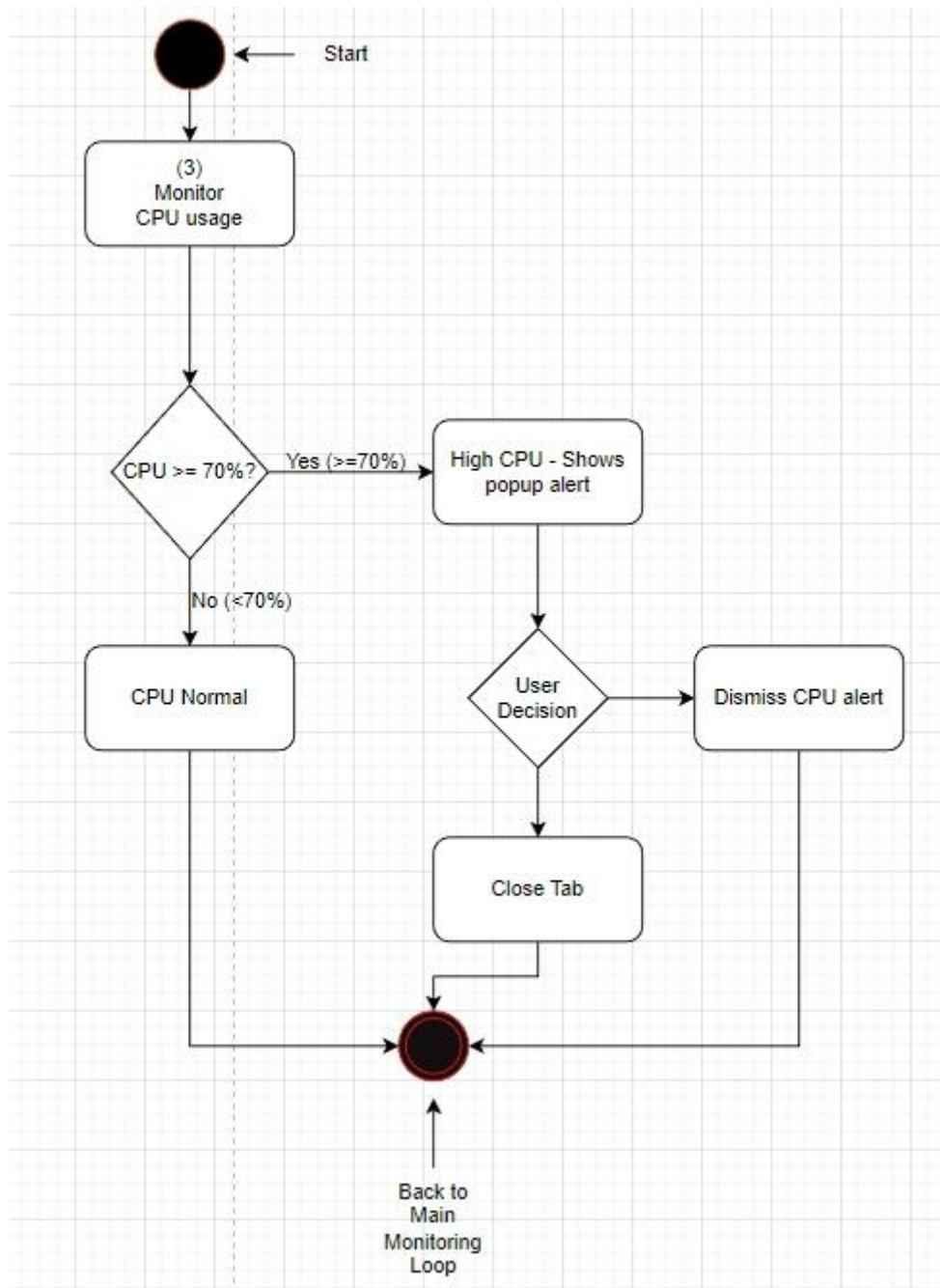


Figure 15: Activity Diagram (Monitor CPU Usage)

CHAPTER 4: SYSTEM DESIGN

4.1 System Block Diagram

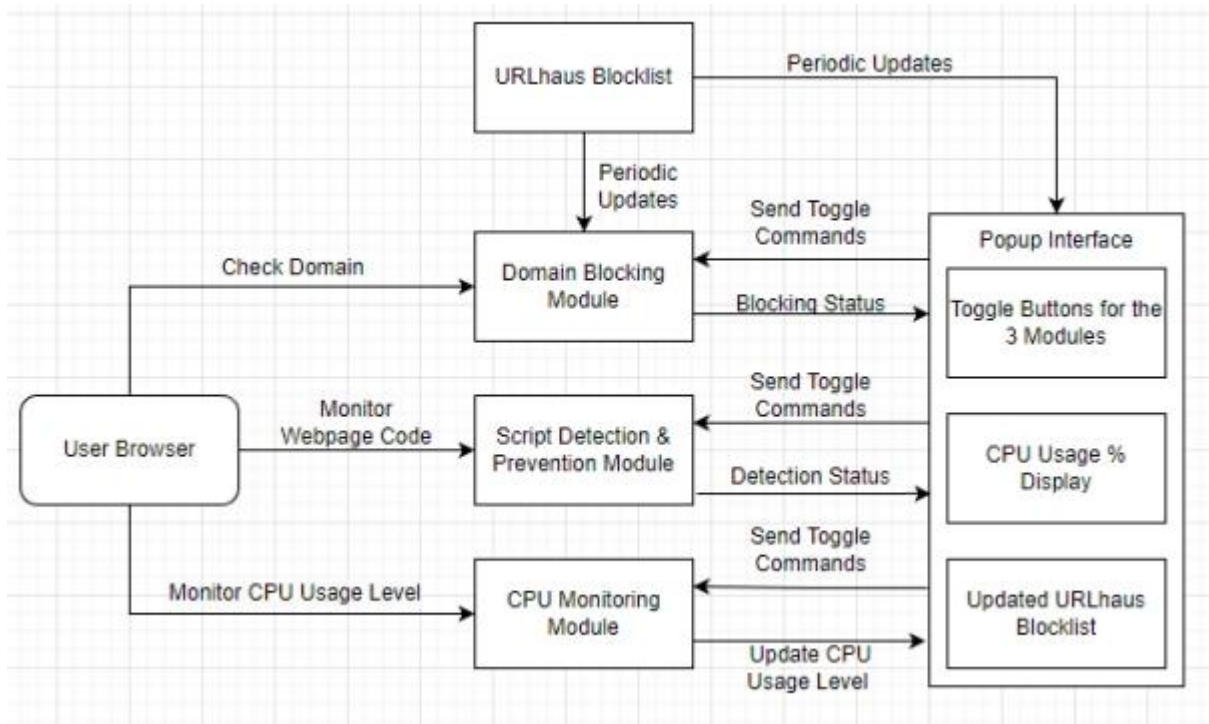


Figure 16: System Block Diagram

4.2 System Components Specifications

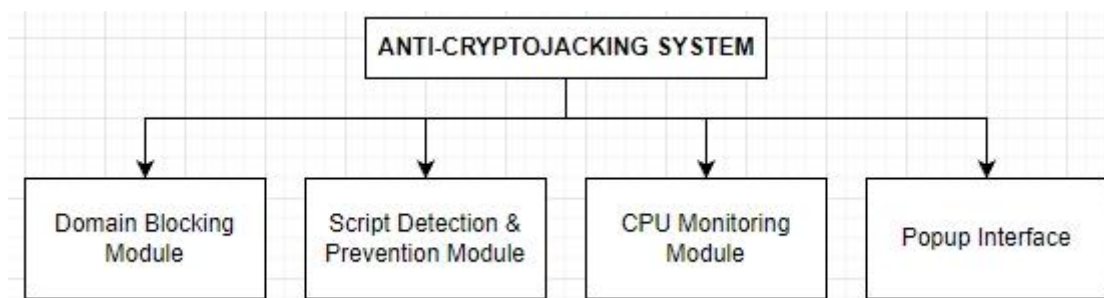


Figure 17: System Components

The Anti-Cryptojacking System is designed to provide users with a secure and smooth browsing experience without requiring any technical expertise. The main goal of the system

is to allow users to surf the internet safely, preventing hidden cryptocurrency mining scripts or other malicious activities from affecting the performance of their computers. The system achieves this by offering a combination of automated protection and clear, user-friendly feedback through a small popup interface. This interface acts as the central point where users can see all alerts, monitor system performance, and control the modules of the extension.

The **Domain Blocking Module** is a critical part of the system from the user's perspective. Its main function is to automatically evaluate the safety of every website a user attempts to visit. When a user enters a URL or clicks a link, this module checks the domain against an up-to-date list of known malicious sites. If a website is considered unsafe, the popup interface immediately notifies the user, explaining the potential risks in simple language. This allows the user to make an informed choice about whether to leave the website or proceed at their own risk. The module works silently in the background, so users are protected without needing to monitor or manage it actively.

Another essential feature visible to users is the **Script Detection and Prevention Module**. Many websites include scripts that run automatically without the user's knowledge. While most scripts are harmless, some may secretly consume system resources or perform malicious actions, such as cryptocurrency mining. This module continuously monitors the content of web pages and detects any unusual behaviour. Alerts are displayed in a clear format, showing the user what kind of activity was detected. By providing this transparency, users can understand what is happening on their system and act if necessary.

The **CPU Monitoring Module** allows users to see how their computer's resources are being used in real time. It tracks CPU usage and immediately notifies the user if a sudden spike occurs. High CPU usage may indicate hidden scripts running in the background, such as unauthorized cryptocurrency miners. Through the popup interface, users receive a warning and can close problematic websites before significant harm occurs. This module reassures users that the system is actively monitoring their computer without requiring them to check system performance manually.

The **Popup Interface** serves as the central control and observation point. Users can view the status of all active modules, enable or disable specific protections, and receive regular

updates on blocklists and CPU activity. The interface is deliberately simple to ensure that users of any technical background can understand it. Through these components, the Anti-Cryptojacking Browser Extension offers continuous protection while keeping users in control, making the browsing experience safe and easy to manage.

4.3 Circuits and Components Design

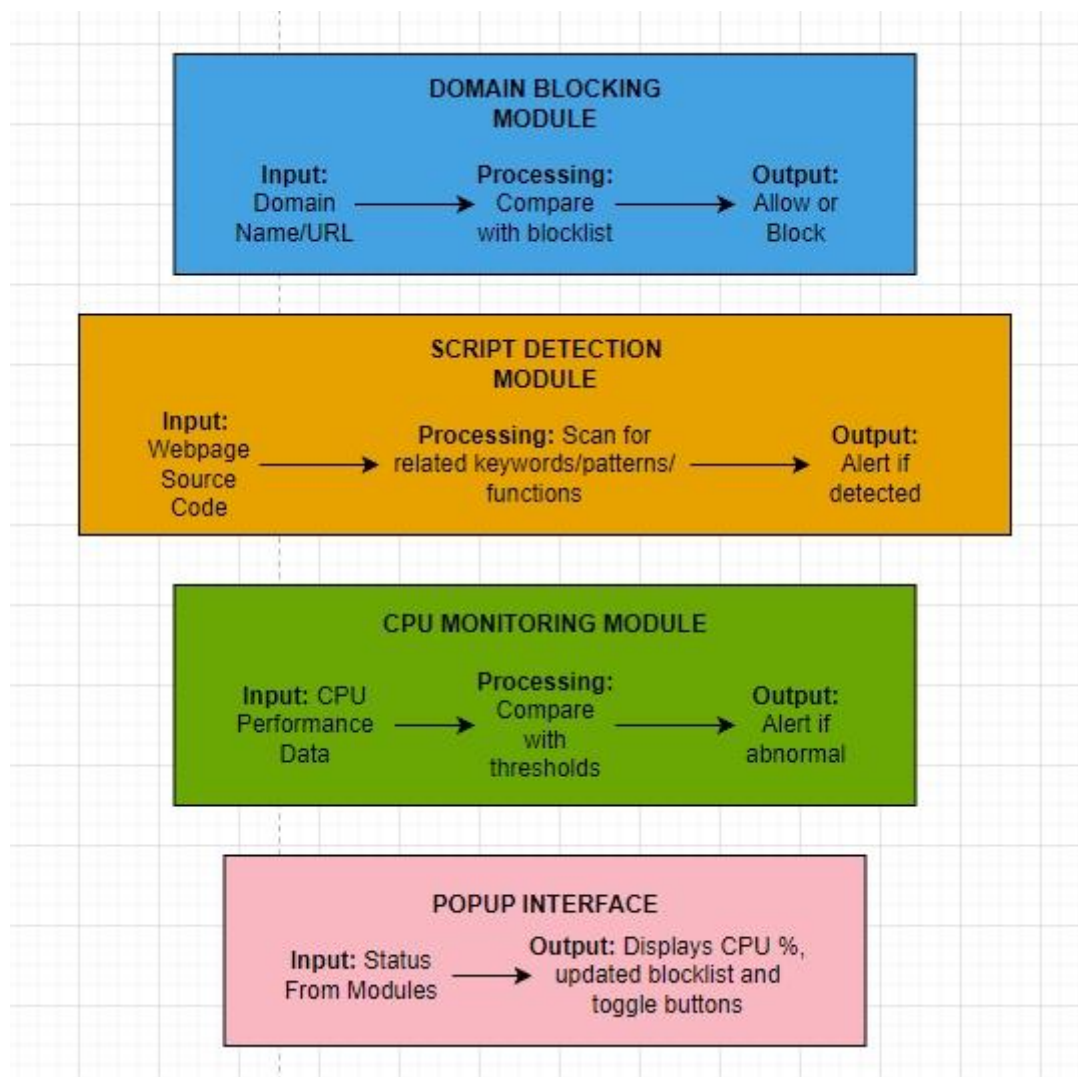


Figure 18: Circuit and Components Design

From a technical perspective, the Anti-Cryptojacking System can be viewed as a system of interconnected processing modules, each performing specific tasks to maintain security and efficiency. Unlike the previous section, this part focuses on the internal workings of the system rather than the user experience. Each module can be considered a separate processing

block that receives inputs, performs analysis or evaluation, and generates outputs that contribute to the system's overall operation as shown in Figure 18.

The **Domain Blocking Module** operates as a filtering block. It receives input in the form of website addresses that the user tries to access. The module compares these inputs against a reference list of known malicious domains. If a match is detected, the module generates an output signal to indicate a potential threat. If no match is found, the input passes through without interruption. The module only performs its function when it is activated via the popup interface. If deactivated, no processing occurs, ensuring efficient use of system resources.

The **Script Detection Module** functions as a continuous scanning block. It receives webpage code as input and systematically examines it for patterns or functions commonly associated with hidden mining scripts or malicious actions. When suspicious activity is found, the module produces output signals detailing the findings. These outputs are sent to the popup interface, which records and organizes the results. If the module is deactivated, scanning is paused, and no output is generated, maintaining efficiency while keeping the system modular.

The **CPU Monitoring Module** continuously evaluates system performance. Inputs in this module consist of real-time CPU usage data. The module compares this data against predefined thresholds to detect abnormal spikes that may indicate hidden scripts running in the background. When such activity is identified, output signals are sent to the popup interface to update the user on potential issues. If deactivated, the module suspends monitoring, preventing unnecessary processing while maintaining the last known performance values.

Finally, the **Popup Interface** serves as the central coordination point for all outputs from active modules. It does not perform any security analysis itself but organizes and displays information in a clear and precise way. It shows which modules are active, updates CPU usage information, and displays blocklist versions from the Domain Blocking Module. By focusing on outputs from active modules only, the interface ensures that the system remains modular, efficient, and simple to understand. This internal design allows each module to operate independently while contributing to the overall functionality of the extension.

4.4 System Components Interaction Operations

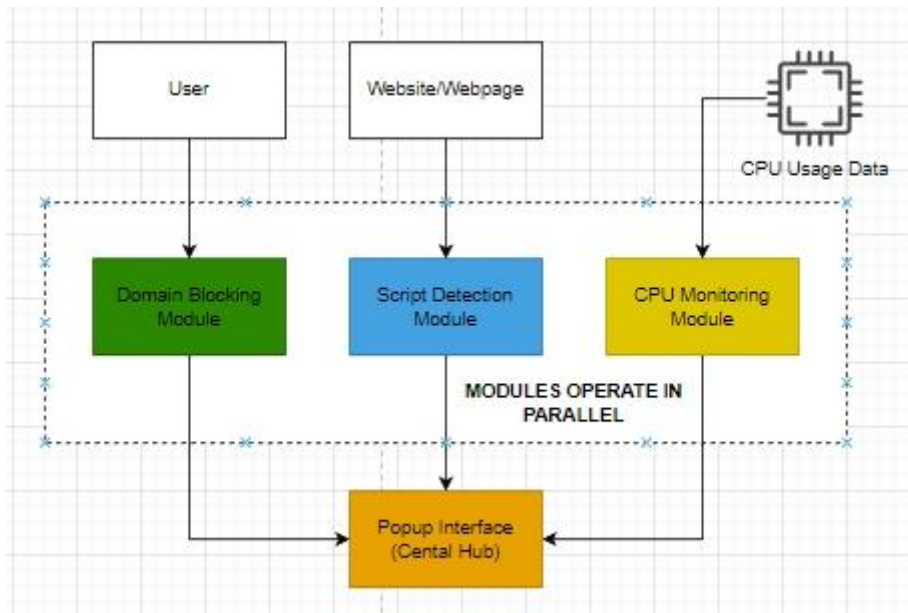


Figure 19: System Interaction Operations

The Anti-Cryptojacking System relies on smooth interaction between its modules to provide a complete security solution. This section describes how the modules work together to process data, detect threats, and communicate results efficiently. Unlike the previous sections, the emphasis here is on the **flow of information and coordination**, rather than the user interface or internal block design as shown in Figure 19.

When a user visits a website, the system begins processing the browsing activity in parallel across multiple modules. The Domain Blocking Module checks the URL against its blocklist, while the Script Detection Module scans the webpage for hidden scripts. The CPU Monitoring Module observes the system's performance to detect unusual activity. All three modules operate simultaneously and independently, generating outputs that represent their findings.

The outputs from each module are then communicated to the Popup Interface, which acts as a hub to consolidate information. The interface organizes alerts from multiple sources so the system can provide the user with a clear summary of any potential threats. By presenting combined security alerts alongside performance data, the system ensures users are aware of

both external and internal risks, such as malicious domains or hidden scripts consuming resources.

Additionally, module interactions allow the system to maintain flexibility and efficiency. Each module can be activated or deactivated independently, and only active modules contribute to the system's outputs. This modular interaction prevents unnecessary processing while keeping the system responsive. By coordinating operations in this way, the Anti-Cryptojacking System maintains continuous monitoring, provides timely alerts, and ensures that the user receives an accurate and up-to-date overview of their system's security at all times.

CHAPTER 5: SYSTEM IMPLEMENTATION

5.1 Hardware Setup

The only hardware that has been used for my final year project is my own personal laptop and it has been utilized for various processes that includes the creation of the browser extension system, containing background scripts, blocking and monitoring functions, as well as testing procedures for the extension that has been created. Although the browser extension is made to be lightweight for maintaining normal browsing performance, a minimal hardware setup is necessary for testing and performance. Since the system only relies on browser APIs and JavaScript code to perform all its intended functions, the hardware requirements are low. Table 6 below shows the overall specifications of my hardware setup.

Table 6: Hardware Setup

Description	Specifications
Model	Acer Predator PH315-53
CPU	Intel Core i7-10870H
GPU	NVIDIA GeForce RTX 2060
Memory	16 GB DDR4 RAM
Storage	1.4 TB SATA SSD
Internet Connection	Required for browsing and testing procedures

5.2 Software Setup

Visual Studio Code is used as the main code editor to create the browser extension system because of its extensive support for web development languages such as HTML, CSS, JSON and JavaScript. Not only that, but it also has useful extensions that have helped during the development process under Manifest V3, which is the latest framework required for modern Chrome extensions. Google Chrome has been used as the primary testing platform because the web browser is commonly used by everyday users, and it also supports Manifest V3-based extensions. JavaScript libraries and browser APIs, such as **chrome.declarativeNetRequest**, are also used to update the blocking list and CPU usage levels, as well as improving the extension's responsiveness and efficiency for real-time monitoring. Table 7 below shows the overall summary of my software setup.

Table 7: Software Setup

Description	Tools Used
Operating System	Windows 11
Web Browser	Google Chrome
Coding Editor	Visual Studio Code
Programming Languages Used	JavaScript – Handles the functionalities of the extension such as domain blocking, script detection & prevention as well as CPU monitoring HTML & CSS – Used for the popup interface design JSON – Helps to store retrieved blacklist from URLhaus and configuration data.

5.3 Setting and Configuration

The Anti-Cryptojacking System requires a few steps of setting and configuration before it can be used. Since the system is developed as a browser extension, the user would need to install it into the web browser. In this project, Google Chrome was used as the main platform for deployment.

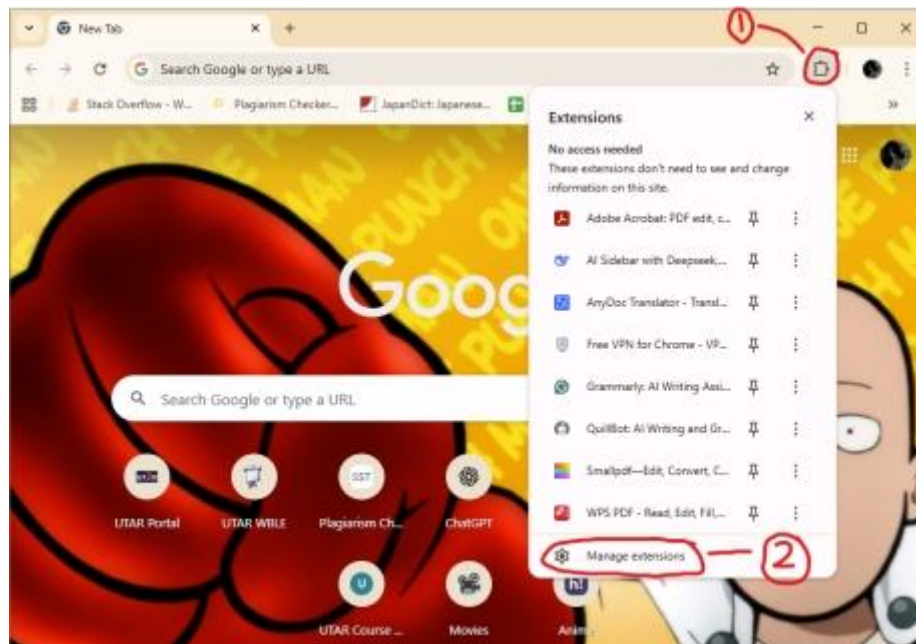


Figure 20: 1st and 2nd step

The process begins by opening the extension management page in Chrome. This page can be accessed by following Steps 1 & 2 as can be seen from Figure 20, and it is responsible for listing out all installed extensions and providing options for managing them.

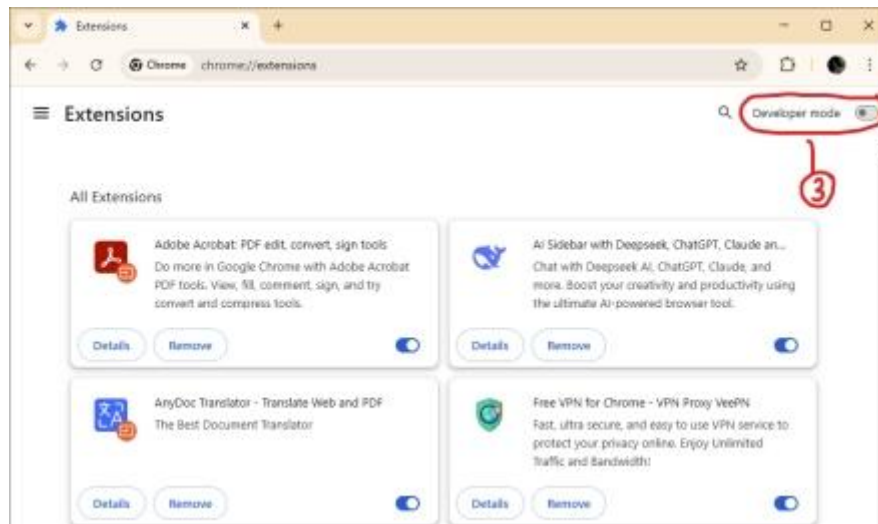


Figure 21: 3rd step

On the extension management page as can be seen from Figure 21, the user needs to enable the “Developer Mode” option by following Step 3, which allows local extensions to be added without uploading them to the Chrome Web Store.

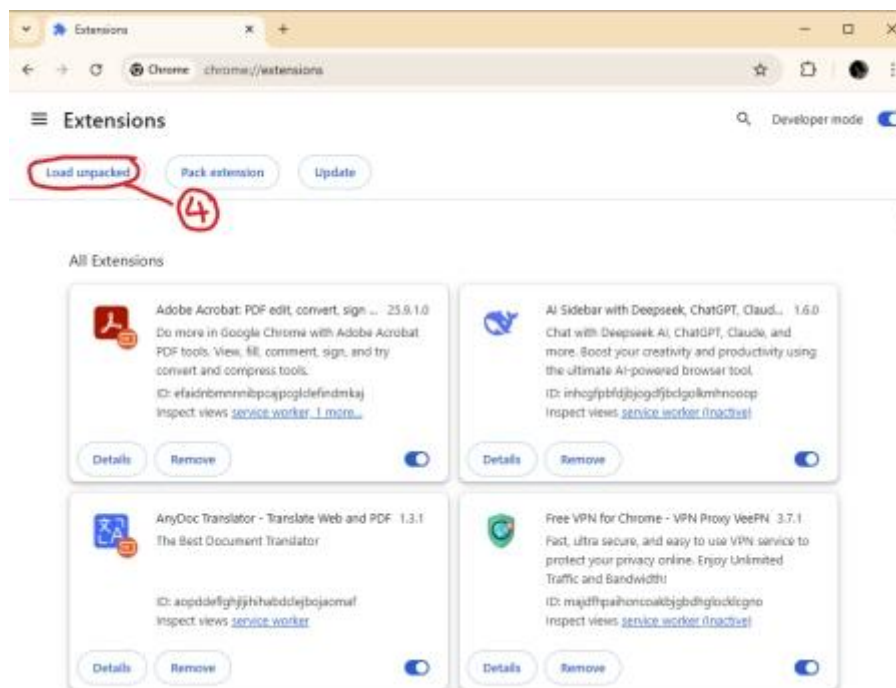


Figure 22: 4th step

Once this option is enabled, the “Load Unpacked” button becomes available. By following Step 4 as shown in Figure 22, the user can browse the computer and select the project folder (**CryptoGuardv4**) that contains the extension files as shown in Step 5 (Figure 23).

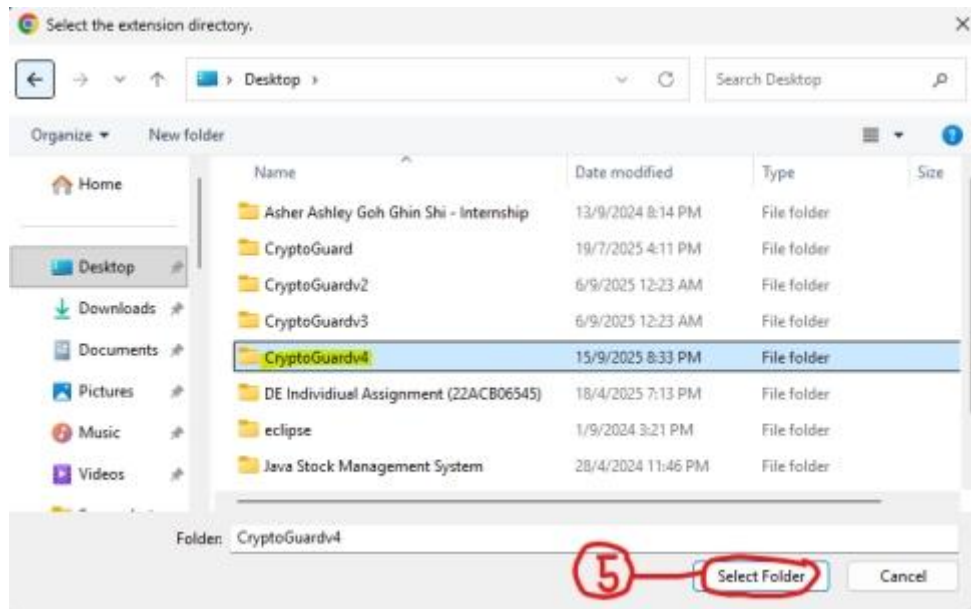


Figure 23: 5th step

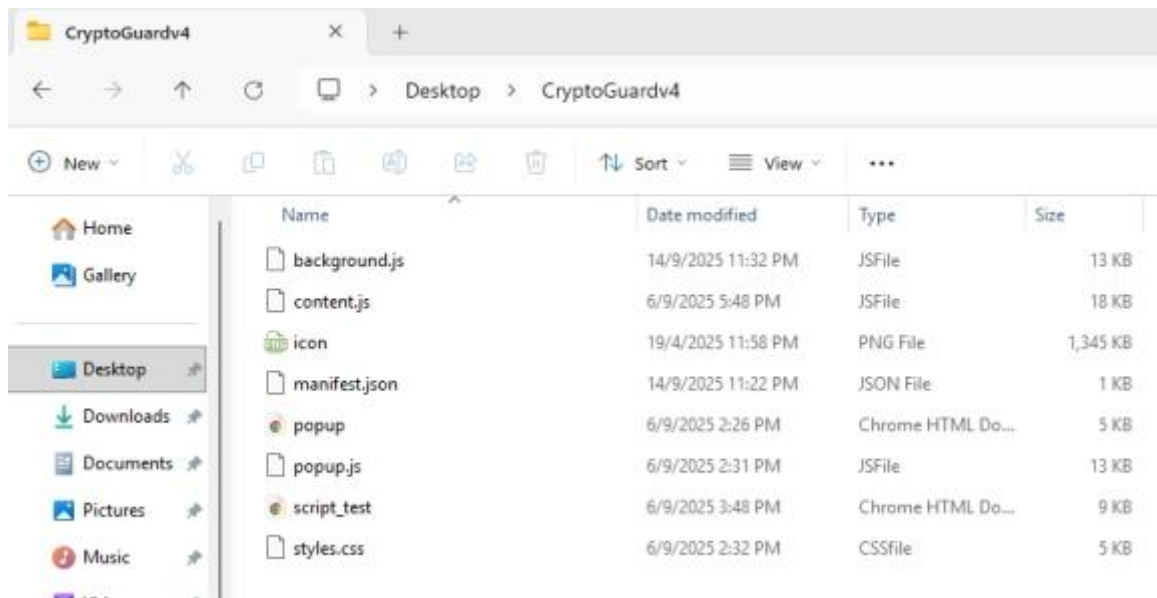


Figure 24: Extension Files

Before uploading the project folder to install the browser extension, the user must make sure that it contains all the necessary extension files which are the **manifest.json**, **background.js**, **content.js**, **popup.html**, **popup.js**, **styles.css**, **script_test.html** and the extension icon (**icon.png**) as shown in Figure 24. After the folder is loaded, the extension will appear in the list of installed extensions as shown in Figure 25 (**CryptoGuard 1.0**).

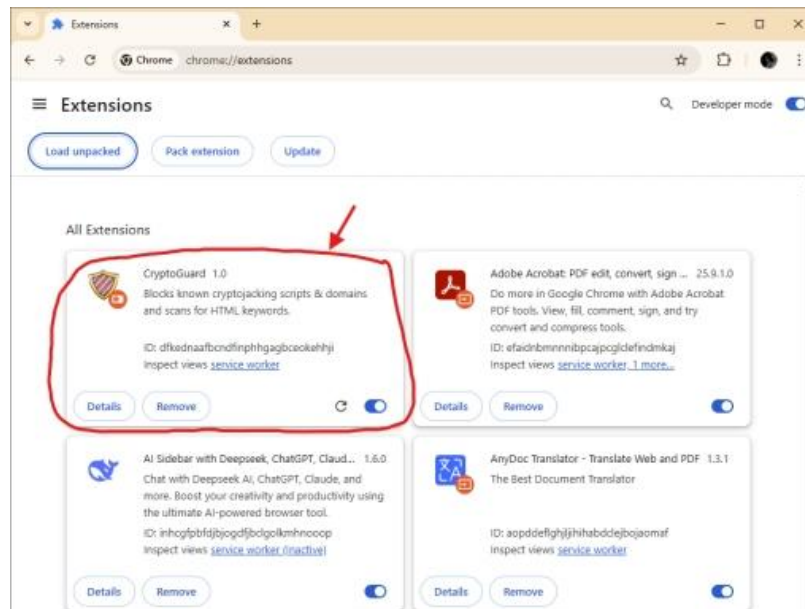


Figure 25: Extension's Outlook

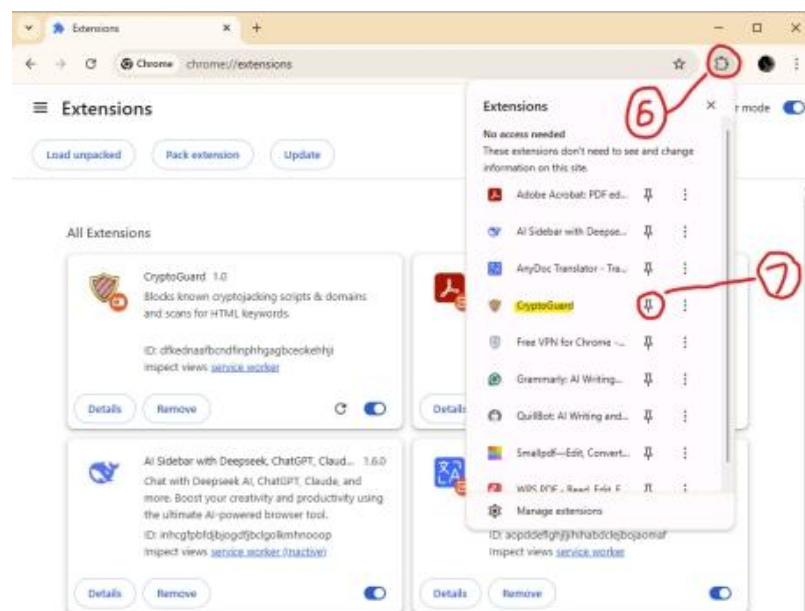


Figure 26: 6th and 7th step

Next, the user would have to pin the browser extension by following Steps 6 and 7 as shown in Figure 26. This is done so that the user would have easier navigation to the browser extension's popup interface. After that, the extension icon will appear as can be seen in Figure 27. By following Step 8 to click on the icon, the user is now able to access the popup interface.

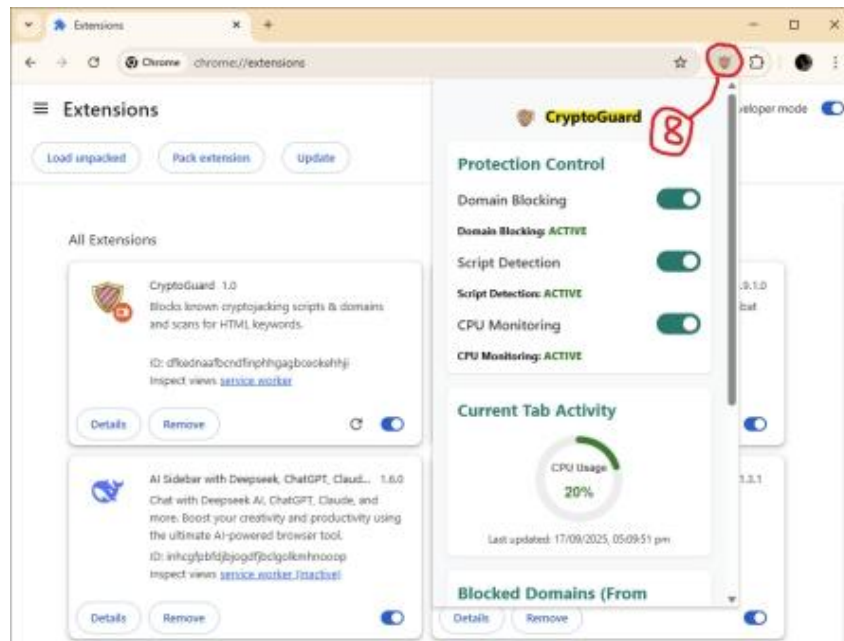


Figure 27: 8th step

5.4 System Operation (with Screenshot)

The Anti-Cryptojacking System is designed to work as a Chrome browser extension that quietly protects users from cryptojacking while they are browsing the internet. Instead of being a single program, it is made up of several different files that work together to form a complete security system. Each file has its own purpose, and they all communicate with one another. Some files are responsible for blocking harmful domains, others scan for suspicious scripts, some check CPU usage, and others show alerts or give users controls through an interface. In this section, I will explain in detail how these files operate, how each function inside them works, and how everything is combined to create the system.

Background Script (background.js)

The background script is the most important part of the extension because it controls the major functions in the background of the browser. Unlike scripts that run only when a webpage is open, the background script keeps running all the time while the extension is active. This means it can handle important tasks like downloading new blocklists, applying domain blocking rules, checking CPU usage, and sending or receiving messages between other parts of the extension.

```
// Function to fetch and apply URLhaus list
async function fetchAndApplyUrlhausList() {
  console.log("Fetching URLhaus list...");
  try {
    const response = await fetch(URLHAUS_LIST_URL);
    if (!response.ok) {
      throw new Error("HTTP error! status: ${response.status}");
    }
    const text = await response.text();
    const domains = parseUrlhausList(text);
    console.log(`Fetched ${domains.length} domains from URLhaus.`);

    const options = {
      day: '2-digit',
      month: '2-digit',
      year: 'numeric',
      hour: '2-digit',
      minute: '2-digit',
      second: '2-digit',
      hour12: true
    };

    await chrome.storage.local.set({
      blockedDomains: domains,
      lastUrlhausUpdate: new Date().toLocaleString('en-GB', options)
    });

    await updateDynamicRules(domains);
    console.log("URLhaus list fetched and applied successfully.");
  } catch (error) {
    console.error("Failed to fetch or apply URLhaus list:", error);
  }
}
```

Figure 28: fetchAndApplyUrlhausList()

```
// Parse the URLhaus hostfile content
function parseUrlhausList(text) {
  const domains = new Set();
  const lines = text.split('\n');
  for (const line of lines) {
    if (line.startsWith('#') || line.trim() === '') {
      continue;
    }
    const parts = line.split(/\s+/);
    if (parts.length > 1) {
      const domain = parts[1].trim();
      if (domain && !domain.startsWith('#')) {
        domains.add(domain);
      }
    }
  }
  return Array.from(domains);
}
```

Figure 29: parseUrlhausList()

One of the main functions in this file is **fetchAndApplyUrlhausList()** as shown in Figure 28. This function connects to the **URLhaus** hostlist endpoint, which is a source that provides updated lists of domains known to host cryptojacking or malware scripts. The function first downloads the latest data and then passes the text to **parseUrlhausList()**. This second function goes through the file line by line, skips comments or empty lines, and only keeps the actual domain names of harmful sites. These domains are collected inside a **Set** to avoid duplicates and then stored in an array.

After the parsing is complete, the **fetchAndApplyUrlhausList()** function in Figure 29 saves the domains and the exact time of the last update into **chrome.storage.local**, so the extension remembers the data even if the browser is closed. It then calls **updateDynamicRules()**, which pushes these domains into Chrome's built-in blocking system so that dangerous sites are blocked immediately.

```
// Update declarativeNetRequest rules
async function updateDynamicRules(domains) {
  const currentDynamicRules = await chrome.declarativeNetRequest.getDynamicRules();
  const removeRuleIds = currentDynamicRules.map(rule => rule.id);

  let newRules = [];
  domains.forEach((domain, index) => {
    const commonCondition = {
      resourceTypes: ["script", "main_frame", "sub_frame", "xmlhttprequest", "websocket"]
    };

    newRules.push({
      id: DYNAMIC_RULE_START_ID + (index * 2),
      priority: 1,
      action: { type: "block" },
      condition: {
        urlFilter: `*://${domain}/*`,
        ...commonCondition
      }
    });
  });

  try {
    await chrome.declarativeNetRequest.updateDynamicRules({
      removeRuleIds: removeRuleIds,
      addRules: newRules
    });
    console.log(`Updated declarativeNetRequest: Removed ${removeRuleIds.length} old rules, Added ${newRules.length} new rules.`);
  } catch (error) {
    console.error("Error updating dynamic rules:", error);
  }
}
```

Figure 30: updateDynamicRules()

Once the domains are ready, the system uses another function called **updateDynamicRules()** as shown in Figure 30. This function works with Chrome's built-in **declarativeNetRequest API**, which allows the extension to block harmful requests before they load in the browser. At the start, the function checks if there are already rules in place and clears them by taking their IDs. After that, it creates new rules for each domain from the blocklist. These rules are set to block things like scripts, page frames, and network requests that try to connect to the harmful sites.

When the rules are finished, the function updates Chrome by removing the old ones and applying the new ones. This makes sure that the extension is always using the most recent blocklist from **URLhaus**. Together with the earlier functions, this process makes a complete cycle where the system fetches the list, filters it, saves it, and finally applies the blocking rules. This way, dangerous websites are stopped before the user even has the chance to open them.

```
// Function to calculate CPU usage percentage from CPU info
function calculateCpuUsage(cpuInfo) {
  let totalUsage = 0;
  let totalIdle = 0;

  cpuInfo.processors.forEach(processor => {
    totalUsage += processor.usage.kernel + processor.usage.user;
    totalIdle += processor.usage.idle;
  });

  const totalTime = totalUsage + totalIdle;
  if (totalTime === 0) return 0; // Avoid division by zero

  return Math.round((totalUsage / totalTime) * 100);
}
```

Figure 31: calculateCpuUsage()

Another important function is **calculateCpuUsage()**, which helps to calculate how much CPU power the browser is consuming as shown in Figure 31. It works by taking CPU readings at different intervals and comparing the results. From this, the function calculates the percentage of CPU that is active instead of idle. Normally, browsing the web does not use very high CPU continuously, so if the system notices that the CPU usage is unusually high for a long period, it can be a strong signal that a cryptojacking script is secretly running. If

this situation is detected, the background script immediately sends a message to the content script so that the user can be warned through a visible alert.

```
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.action === "toggleDomainBlocking") {
    chrome.storage.local.set({ domainBlockingEnabled: message.enabled });
    (async () => {
      try {
        if (message.enabled) {
          // Directly fetch and apply URLhaus rules
          await fetchAndApplyUrlhausList();
        } else {
          const currentDynamicRules = await chrome.declarativeNetRequest.getDynamicRules();
          const removeRuleIds = currentDynamicRules.map(rule => rule.id);
          await chrome.declarativeNetRequest.updateDynamicRules({
            removeRuleIds: removeRuleIds
          });
        }
        sendResponse({ success: true });
      } catch (error) {
        console.error("Error during toggleDomainBlocking:", error);
        sendResponse({ success: false, message: "Error toggling domain blocking: ${error.message}" });
      }
    })();
    return true;
  }
});
```

Figure 32: chrome.runtime.onMessage.addListener

Finally, the background script contains a message listener, which is the **chrome.runtime.onMessage.addListener** function as shown in Figure 32. This listener works like a communication hub between the popup, content script, and the background script. One of the actions it listens for is **"toggleDomainBlocking"**. When this message is received, the listener checks whether domain blocking is being turned on or off. If it is enabled, the background script immediately calls **fetchAndApplyUrlhausList()** to download and apply the latest rules. If it is disabled, it clears all the current blocking rules from Chrome so that no domains are being blocked.

The listener then sends a response back to confirm if the action was successful or if an error occurred. This setup allows the popup interface to control whether the domain blocking feature is active or not, while still keeping the process automatic and secure. It also ensures that communication between the different parts of the extension is smooth and reliable.

Content Script (content.js)

The content script runs inside every webpage that the user opens, and its job is to carefully inspect the page for hidden cryptojacking attempts. If the background script can be seen as the “brain” of the extension, the content script is like the “eyes,” since it directly looks at what is happening on the webpage.

```
/**
 * Scans inline script content for cryptojacking indicators.
 * @param {HTMLElement} scriptElement The script element to scan.
 */
function scanInlineScriptContent(scriptElement) {
  try {
    if (scriptElement.innerHTML) {
      const foundKeyword = checkForKeywords(scriptElement.innerHTML);
      if (foundKeyword) {
        let detectedContent = scriptElement.innerHTML.substring(0, 500);
        if (scriptElement.innerHTML.length > 500) {
          detectedContent += '... (truncated)';
        }
        reportDetection(detectedContent, foundKeyword, "Inline Script");
      }
    }
  } catch (error) {
    console.error("CryptoGuard Content: Error during inline script scan:", error);
  }
}
```

Figure 33: scanInlineScriptContent()

The function **scanInlineScriptContent()**, as shown in Figure 33, is responsible for checking scripts that are directly written inside the webpage’s HTML. These are called **inline scripts**, and sometimes cryptojacking code can be hidden there. The function looks for certain keywords such as “**coinhive**,” “**cryptoloot**,” “**webmine**,” or “**monero**,” because these names are linked to popular cryptomining programs. If one of these keywords is found, the system immediately knows the script is suspicious.

```
// --- MutationObserver for dynamic content injection ---
const observer = new MutationObserver((mutationsList) => {
  console.log("CryptoGuard Content: MutationObserver triggered. Checking for new content...");

  chrome.storage.local.get("scriptDetectionEnabled", (data) => {
    if (!data.scriptDetectionEnabled) {
      console.log("CryptoGuard Content: Script detection is disabled. Skipping mutation-observer scan.");
      return;
    }

    for (const mutation of mutationsList) {
      if (mutation.type === 'childList' && mutation.addedNodes.length > 0) {
        console.log(`CryptoGuard Content: MutationObserver - ${mutation.addedNodes.length} nodes added.`);
        for (const node of mutation.addedNodes) {
          if (node.nodeType === Node.ELEMENT_NODE) {
            if (node.hasAttribute('href') || node.hasAttribute('src')) {
              const urlToCheck = node.hasAttribute('href') ? node.getAttribute('href') : node.getAttribute('src');
              const foundKeyword = checkForKeywords(urlToCheck);
              if (foundKeyword) {
                const detectionType = (node.tagName === 'SCRIPT' && node.hasAttribute('src')) ? "Script SRC" : "URL";
                reportDetection(urlToCheck, foundKeyword, detectionType);
              }
            }
          }
          scanUrlsInElement(node);
          scanScriptsInElement(node);
        }
      }
    }
  });
});
```

Figure 34: MutationObserver

Websites can also add new scripts after the page has already loaded. To handle this, the content script uses something called a **MutationObserver** as shown in Figure 34. This observer works like a security guard that keeps watching the page for any changes. If a new script is added, the observer quickly calls functions like **scanScriptsInElement()** and **scanUrlsInElement()** to scan it for harmful content. This is very important because some cryptojacking attacks only start after the main page has finished loading.

```

/**
 * Scans script elements (both external and inline) within a given root.
 * @param {HTMLElement} [root=document] The root element to start the scan from. Defaults to document.
 */
function scanScriptsInElement(root = document) {
  console.log("CryptoGuard Content: Starting scanScriptsInElement...");
  try {
    const scripts = root.querySelectorAll('script');
    console.log(`CryptoGuard Content: Found ${scripts.length} script elements in current scan scope.`);
    for (const script of scripts) {
      if (script.src) {
        const foundKeyword = checkForKeywords(script.src);
        if (foundKeyword) {
          reportDetection(script.src, foundKeyword, "Script SRC");
        }
      } else {
        scanInlineScriptContent(script);
      }
    }
  } catch (error) {
    console.error("CryptoGuard Content: Error during script scan:", error);
  }
  console.log("CryptoGuard Content: Finished scanScriptsInElement.");
}

```

Figure 35: scanScriptsInElement()

The role of the **scanScriptsInElement()** function as shown in Figure 35 is to go through all the script tags that exist in a webpage. It checks both external scripts that are loaded from another source using the **src** attribute, and inline scripts that are written directly inside the HTML. If an external script is found, the function looks at the script URL and checks if it contains any cryptojacking keywords using the **checkForKeywords** function. If a keyword is detected, the system immediately calls **reportDetection()** to log the suspicious script. On the other hand, if the script is inline, the function passes it to **scanInlineScriptContent()**, which checks the actual code written inside the tag. This way, both types of scripts are covered during the scan.

```

/**
 * Scans all URL attributes (href, src) in the DOM for cryptojacking keywords.
 * @param {HTMLElement} [root=document] The root element to start the scan from. Defaults to document.
 */
function scanUrlsInElement(root = document) {
  console.log("CryptoGuard Content: Starting scanUrlsInElement...");
  try {
    const elementsWithUrls = root.querySelectorAll('[href], [src]');
    console.log(`CryptoGuard Content: Found ${elementsWithUrls.length} elements with URL attributes in current scan scope.`);

    for (const element of elementsWithUrls) {
      let urlToCheck = '';
      let attributeName = '';

      if (element.hasAttribute('href')) {
        urlToCheck = element.getAttribute('href');
        attributeName = 'href';
      } else if (element.hasAttribute('src')) {
        urlToCheck = element.getAttribute('src');
        attributeName = 'src';
      }

      if (urlToCheck) {
        const foundKeyword = checkForKeywords(urlToCheck);
        if (foundKeyword) {
          const detectionType = (element.tagName === 'SCRIPT' && attributeName === 'src') ? "Script SRC" : "URL";
          reportDetection(urlToCheck, foundKeyword, detectionType);
        }
      }
    }
  } catch (error) {
    console.error("CryptoGuard Content: Error during URL scan:", error);
  }
  console.log("CryptoGuard Content: Finished scanUrlsInElement.");
}

```

Figure 36: scanUrlsInElement()

On the other hand, the function, **scanUrlsInElement()**, as shown in Figure 36, focuses on scanning all elements in the page that contain links or sources, such as those with **href** or **src** attributes. It collects every element with these attributes and examines the URL values for suspicious keywords. If a keyword is detected in any of these links or sources, the function again reports it as a possible cryptojacking attempt. This scanning is not limited to script tags only, but can also include other elements like images, links, or iframes that might load harmful resources. By running both of these functions together, the extension is able to carefully inspect all scripts and URLs in the webpage to ensure no hidden cryptojacking code goes unnoticed.

```

if (message.action === "displayCryptojackWarning") {
  if (document.getElementById('cryptojackWarningBanner')) {
    console.log("CryptoGuard Content: Warning banner already exists. Skipping display.");
    return;
  }

  console.log("CryptoGuard Content: Displaying cryptojacking warning banner.");
  const warningDiv = document.createElement('div');
  warningDiv.id = 'cryptojackWarningBanner';
  warningDiv.style.cssText = `
    position: fixed;
    bottom: 20px;
    right: 20px;
    width: auto;
    max-width: 400px;
    background-color: #ffeb3b; /* Yellow */
    color: #333;
    padding: 15px;
    border-radius: 8px;
    text-align: center;
    font-size: 16px;
    font-weight: bold;
    z-index: 99999; /* Ensure it's on top */
    box-shadow: 0 4px 10px #0000000.25;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 10px;
    font-family: 'Segoe UI', Arial, sans-serif; /* Consistent font */
  `;

  const snippetsHtml = detectedSnippets.size > 0 ? Array.from(detectedSnippets).map((snippet, index) => `
    <div style="margin: 5px 0; text-align: left; width: 100%;">
      <strong>Snippet ${index + 1}</strong>
      <code style="display: block; background-color: #f0f0f0; padding: 8px; border-radius: 4px; overflow-x: auto;
        white-space: pre-wrap; word-break: break-all; font-size: 0.85em;">${snippet}</code>
    </div>
  `).join('') : `<p style="font-weight: normal; font-size: 0.9em;">No specific code snippet available for display, but a threat was detected.</p>`;

  warningDiv.innerHTML = `
    <span style="font-size: 1.1em;">⚠️ CryptoGuard: Potential cryptojacking script detected!</span>
    <div style="font-size: 14px; font-weight: normal; text-align: left; width: 100%; max-height: 200px;
      overflow-y: auto; border: 1px solid #ccc; padding: 5px; border-radius: 4px;">
      <p style="margin: 0 0 5px 0;">Detected in:</p>
      ${snippetsHtml}
    </div>
    <div style="display: flex; gap: 10px;">
      <button id="closePageButton" style="background-color: #d32f2f; color: white; border: none; padding:
        8px 15px; border-radius: 5px; cursor: pointer; font-size: 1em; font-weight: bold; transition: background-color 0.2s ease;">Close Page</button>
      <button id="closePopupButton" style="background-color: #4CAF50; color: white; border: none; padding:
        8px 15px; border-radius: 5px; cursor: pointer; font-size: 1em; font-weight: bold; transition: background-color 0.2s ease;">Dismiss</button>
    </div>
  `;

  try {
    document.body.appendChild(warningDiv);
  } catch (error) {
    console.error("CryptoGuard Content: Could not append warning banner to body:", error);
    document.documentElement.appendChild(warningDiv);
  }

  const closePageButton = document.getElementById('closePageButton');
  if (closePageButton) {
    closePageButton.addEventListener('click', () => {
      console.log("CryptoGuard Content: 'Close Page' button clicked. Sending closeTab message.");
      chrome.runtime.sendMessage({ action: "closeTab" });
    });
  } else {
    console.error("CryptoGuard Content: 'Close Page' button not found after creation.");
  }

  const closePopupButton = document.getElementById('closePopupButton');
  if (closePopupButton) {
    closePopupButton.addEventListener('click', () => {
      console.log("CryptoGuard Content: 'Close Popup' button clicked. Removing warning banner.");
      const banner = document.getElementById('cryptojackWarningBanner');
      if (banner) {
        banner.remove();
      }
    });
  } else {
    console.error("CryptoGuard Content: 'Close Popup' button not found after creation.");
  }
}

```

Figure 37: Popup Alert for Script Detection

```
// New: Display CPU warning
if (message.action === "displayCpuWarning") {
  if (document.getElementById('cpuWarningBanner')) {
    console.log("CryptoGuard Content: CPU Warning banner already exists. Skipping display.");
    return;
  }

  console.log("CryptoGuard Content: Displaying CPU warning banner.");
  const cpuWarningDiv = document.createElement('div');
  cpuWarningDiv.id = 'cpuWarningBanner';
  cpuWarningDiv.style.cssText = `
    position: fixed;
    bottom: 20px;
    left: 20px; /* Position on the left to avoid conflict with cryptojack warning */
    width: auto;
    max-width: 400px;
    background-color: #ff9800; /* Orange */
    color: white;
    padding: 15px;
    border-radius: 8px;
    text-align: center;
    font-size: 16px;
    font-weight: bold;
    z-index: 99998; /* Slightly lower z-index than cryptojack warning */
    box-shadow: 0 4px 10px rgba(0,0,0,0.25);
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 10px;
    font-family: 'Segoe UI', Arial, sans-serif;
  `;
};
```

```
cpuWarningDiv.innerHTML = `
<span style="font-size: 1.1em;">▲ CryptoGuard: High CPU Activity Detected!</span>
<p style="font-size: 14px; font-weight: normal; margin: 0;">This page might be consuming significant resources
(estimated system CPU: ${message.activityLevel}%).</p>
<div style="display: flex; gap: 10px;">
  <button id="closeCpuPageButton" style="background-color: #d32f2f; color: white; border: none; padding: 8px 15px;
border-radius: 5px; cursor: pointer; font-size: 1em; font-weight: bold; transition: background-color 0.2s ease;">Close Page</button>
  <button id="dismissCpuWarningButton" style="background-color: #4CAF50; color: white; border: none; padding: 8px 15px;
border-radius: 5px; cursor: pointer; font-size: 1em; font-weight: bold; transition: background-color 0.2s ease;">Dismiss</button>
</div>
`;

try {
  document.body.appendChild(cpuWarningDiv);
} catch (error) {
  console.error("CryptoGuard Content: Could not append CPU warning banner to body:", error);
  document.documentElement.appendChild(cpuWarningDiv);
}

const closeCpuPageButton = document.getElementById('closeCpuPageButton');
if (closeCpuPageButton) {
  closeCpuPageButton.addEventListener('click', () => {
    console.log("CryptoGuard Content: 'Close Page' button clicked on CPU warning. Sending closeTab message.");
    chrome.runtime.sendMessage({ action: "closeTab" });
  });
}

const dismissCpuWarningButton = document.getElementById('dismissCpuWarningButton');
if (dismissCpuWarningButton) {
  dismissCpuWarningButton.addEventListener('click', () => {
    console.log("CryptoGuard Content: 'Dismiss' button clicked on CPU warning. Removing banner.");
    const banner = document.getElementById('cpuWarningBanner');
    if (banner) {
      banner.remove();
    }
  });
}
}
```

Figure 38: Popup Alert for CPU Monitoring

Whenever a keyword, function or script related to cryptojacking is detected in the webpage's source code, the content script creates a popup alert on the webpage itself as shown in Figure 37. This is done by adding new HTML elements and placing them at the bottom right of the page. The popup displays a message such as "CryptoGuard: Potential cryptojacking script detected!" and shows the detected code snippets inside a small scrollable box. Two buttons are added, which are **Close Page**, that closes the current tab, and **Dismiss**, that removes the warning from the current webpage. This way, the user is clearly informed about the threat and can quickly choose how to respond.

The same method is used for CPU warnings as shown in Figure 38. When the background script detects that the CPU is being used too much, it tells the content script to show another popup alert. This popup is placed at the bottom left of the page and it shows a message like "CryptoGuard: High CPU Activity Detected!" along with the actual CPU usage. Just like the first one, it comes with **Close Page** and **Dismiss** buttons so the user can either exit the webpage immediately or continue browsing.

Popup Script and Interface (popup.html, popup.js, styles.css)

The popup is the part of the extension that users interact with directly. When the user clicks on the system's icon in the Chrome toolbar, a small window opens, showing the popup interface. This interface is built using three files: **popup.html**, **popup.js**, and **styles.css**.

```
<!-- Protection Control Section -->
<div class="info-box">
  <h3>Protection Control</h3>
  <div class="toggle-section">
    <label for="toggleDomainBlocking" class="toggle-label">Domain Blocking</label>
    <div class="switch-container">
      <label class="switch">
        <input type="checkbox" id="toggleDomainBlocking">
        <span class="slider"></span>
      </label>
    </div>
  </div>
  <div id="domainBlockingStatus" class="status inactive">Domain Blocking: OFF</div>

  <div class="toggle-section" style="margin-top: 15px;">
    <label for="toggleScriptDetection" class="toggle-label">Script Detection</label>
    <div class="switch-container">
      <label class="switch">
        <input type="checkbox" id="toggleScriptDetection">
        <span class="slider"></span>
      </label>
    </div>
  </div>
  <div id="scriptDetectionStatus" class="status inactive">Script Detection: OFF</div>

  <!-- New: CPU Monitoring Toggle -->
  <div class="toggle-section" style="margin-top: 15px;">
    <label for="toggleCpuMonitoring" class="toggle-label">CPU Monitoring</label>
    <div class="switch-container">
      <label class="switch">
        <input type="checkbox" id="toggleCpuMonitoring">
        <span class="slider"></span>
      </label>
    </div>
  </div>
  <div id="cpuMonitoringStatus" class="status inactive">CPU Monitoring: OFF</div>
</div>
```

Figure 39: Protection Status Section

```

<!-- Blocked Domains List -->
<div class="info-box">
  <h3>Blocked Domains (From URLhaus)</h3>
  <p style="color: #333;">
    <span id="lastUpdate">Last updated: Loading...</span>
  </p>
  <p>
    <button id="updateListButton" style="margin-left: 10px; padding: 5px 10px; border: none;
    border-radius: 4px; background-color: #00796b; color: white; cursor: pointer;">Update List Now</button>
  </p>
  <div id="blockedList" style="max-height: 150px; overflow-y: auto; border: 1px solid #eee;
  padding: 5px; border-radius: 5px;">
    <!-- List of blocked domains will be populated here -->
  </div>
</div>

```

Figure 40: Blocked Domain List Section

```

<!-- Current Tab Activity Section -->
<div class="info-box">
  <h3>Current Tab Activity</h3>
  <div id="cpuActivityDisplay" style="display: flex; flex-direction: column;
  align-items: center; justify-content: center; gap: 10px;">
    <div class="cpu-circle-container" style="width: 100px; margin: auto;">
      <svg class="cpu-circle" width="100" height="100" viewBox="0 0 36 36" style="font-family:
      Arial, sans-serif;">
        <path class="cpu-circle-bg" d="M18 2.0845
        a 15.9155 15.9155 0 0 1 0 31.831
        a 15.9155 15.9155 0 0 1 0 -31.831"
        fill="none" stroke="#e0e0e0" stroke-width="2" />
        <path class="cpu-circle-progress" d="M18 2.0845
        a 15.9155 15.9155 0 0 1 0 31.831
        a 15.9155 15.9155 0 0 1 0 -31.831"
        fill="none" stroke="#4caf50" stroke-width="2" stroke-dasharray="0, 100" />
        <!-- Combined label, centered horizontally -->
        <text x="18" y="14" class="cpu-circle-label" text-anchor="middle" dominant-baseline="middle">
          CPU Usage
        </text>
        <!-- Percentage, larger and bold -->
        <text x="18" y="24" class="cpu-circle-text" text-anchor="middle" dominant-baseline="middle">
          0%
        </text>
      </svg>
    </div>
    <p style="color: #333; margin: 0;">
      <span id="cpuActivityTimestamp">Last updated: N/A</span>
    </p>
  </div>
</div>

```

Figure 41: Current Tab Activity Section (CPU Monitoring)

The **popup.html** file is used to build the layout of the extension's interface. Inside this file, there are toggle switches that let the user control key features such as domain blocking, script detection, and CPU monitoring as shown in Figure 39. These switches make it easy for the user to decide which protections should be active at any moment. The file also has a section

that shows the list of blocked domains and another part that displays the date and time when the blocklist was last updated as shown in Figure 40, so the user always knows if the system is working with the latest data.

Another part of the interface is the circular display as shown in Figure 41, which represents the CPU activity level in real time. This allows the user to quickly see if the system is using too much CPU, which might mean a cryptojacking attempt is happening. Together, these elements make the popup simple but effective, giving the user both control and clear information about the extension's current status.

```
// Initialize toggles and statuses from runtime or local storage
chrome.runtime.sendMessage({ action: "getProtectionStatus" }, (response) => {
  const domainBlockingEnabled = response?.domainBlockingEnabled ?? false;
  const scriptDetectionEnabled = response?.scriptDetectionEnabled ?? false;
  const cpuMonitoringEnabled = response?.cpuMonitoringEnabled ?? false;
  allDomains = response?.blockedDomains || [];
  const lastUrlhausUpdate = response?.lastUrlhausUpdate;
  const currentTabCpuActivity = response?.currentTabCpuActivity;

  toggleDomainBlocking.checked = domainBlockingEnabled;
  toggleScriptDetection.checked = scriptDetectionEnabled;
  toggleCpuMonitoring.checked = cpuMonitoringEnabled;

  updateStatus(domainBlockingStatus, domainBlockingEnabled, "Domain Blocking");
  updateStatus(scriptDetectionStatus, scriptDetectionEnabled, "Script Detection");
  updateStatus(cpuMonitoringStatus, cpuMonitoringEnabled, "CPU Monitoring");
  updateCpuDisplay(currentTabCpuActivity, cpuMonitoringEnabled); // Pass cpuMonitoringEnabled

  displayBlockedDomains(domainBlockingEnabled);
  if (lastUrlhausUpdate) {
    lastUpdateSpan.textContent = `Last updated: ${lastUrlhausUpdate}`;
  } else {
    lastUpdateSpan.textContent = `Last updated: Never`;
  }
});
```

Figure 42: chrome.runtime.sendMessage()

The **popup.js** file makes the popup interface interactive and functional. It not only reacts to user actions but also loads the current protection status each time the popup is opened. To do this, it sends a message to the background script through **chrome.runtime.sendMessage()** as shown in Figure 42 to find out if domain blocking, script detection, and CPU monitoring are active. It also receives the list of blocked domains, the last update time of the blocklist, and the CPU activity of the current tab. With this information, popup.js updates the toggle switches, status messages, and CPU display so the user can instantly see what features are active. This way, the popup always shows accurate and up-to-date information.

```

// Function to update CPU usage display
function updateCpuDisplay(activityData, isMonitoringEnabled) {
    const cpuActivityContainer = document.querySelector('.info-box:nth-of-type(3)'); // Adjust if needed

    if (isMonitoringEnabled) {
        cpuActivityContainer.style.display = 'none'; // Hide the entire section
        if (cpuCircleProgress) cpuCircleProgress.setAttribute('stroke-dasharray', '0, 100');
        if (cpuCircleText) cpuCircleText.textContent = 'N/A';
        if (cpuCircleProgress) cpuCircleProgress.className.baseVal = 'cpu-circle-progress';
        if (cpuCircleText) cpuCircleText.className.baseVal = 'cpu-circle-text';
        document.getElementById('cpuActivityTimestamp').textContent = "Last updated: N/A";
        return;
    } else {
        cpuActivityContainer.style.display = 'block'; // Show the section
    }

    if (activityData && activityData.activityLevel !== undefined) {
        const percentage = activityData.activityLevel;
        const circumference = 100; // normalized circumference for stroke-dasharray

        // Calculate stroke-dasharray for progress
        const offset = circumference - (percentage / 100) * circumference;
        if (cpuCircleProgress) {
            cpuCircleProgress.setAttribute('stroke-dasharray', `${circumference - offset}, ${offset}`);

            // Update color based on percentage
            cpuCircleProgress.className.baseVal = 'cpu-circle-progress'; // reset classes
            if (percentage > 70) {
                cpuCircleProgress.classList.add('high');
            } else if (percentage > 40) {
                cpuCircleProgress.classList.add('medium');
            } else {
                cpuCircleProgress.classList.add('low');
            }
        }
    }
}

```

```

// Update the text inside the circle
if (cpuCircleText) {
    cpuCircleText.textContent = `${percentage}%`;
    cpuCircleText.className.baseVal = 'cpu-circle-text'; // reset classes
    if (percentage > 70) {
        cpuCircleText.classList.add('high');
    } else if (percentage > 40) {
        cpuCircleText.classList.add('medium');
    } else {
        cpuCircleText.classList.add('low');
    }
}

const options = {
    day: '2-digit',
    month: '2-digit',
    year: 'numeric',
    hour: '2-digit',
    minute: '2-digit',
    second: '2-digit',
    hour12: true
};
document.getElementById('cpuActivityTimestamp').textContent = `Last updated:
${new Date(activityData.timestamp).toLocaleString('en-GB', options)}`;
} else {
    if (cpuCircleProgress) cpuCircleProgress.setAttribute('stroke-dasharray', '0, 100');
    if (cpuCircleText) cpuCircleText.textContent = 'N/A';
    if (cpuCircleProgress) cpuCircleProgress.className.baseVal = 'cpu-circle-progress';
    if (cpuCircleText) cpuCircleText.className.baseVal = 'cpu-circle-text';
    document.getElementById('cpuActivityTimestamp').textContent = "Last updated: N/A";
}
}

```

Figure 43: updateCpuDisplay()

In addition to handling toggles, popup.js is also responsible for keeping the CPU display updated in real time. It uses the **updateCpuDisplay()** function as shown in Figure 43 to show the current CPU usage inside the circular meter. This helps the user quickly see if the CPU is working normally or if there is unusual activity that could mean cryptojacking.

The **updateCpuDisplay()** function controls how the CPU activity looks on the screen. When monitoring is off, it hides the CPU section and shows “N/A.” When it is on, the function updates the circle with the correct percentage and changes its color depending on the usage level. It also records the time of the latest update. This gives the user a clear and simple view of CPU performance at all times.

```
/* Container for blocked domains */
#blockedlist {
  display: flex;
  flex-direction: column;
  gap: 8px; /* Reduced gap */
  margin-top: 12px;
}

/* Style for each blocked domain item */
.blocked-domain-item {
  background-color: #f1f1f1;
  color: #333;
  padding: 8px 12px; /* Reduced padding */
  border-radius: 6px; /* Slightly smaller border radius */
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  font-size: 0.9rem; /* Smaller font size */
  font-weight: 500; /* Slightly lighter weight */
  display: flex;
  justify-content: space-between;
  align-items: center;
  transition: background-color 0.3s, transform 0.2s;
}

/* Hover effect for domain item */
.blocked-domain-item:hover {
  background-color: #e0e0e0;
  color: #000;
  transform: scale(1.03); /* Slightly smaller scale effect */
}

.footer-gradient-text {
  text-align: center;
  font-size: 0.85rem;
  font-weight: 600;
  background: linear-gradient(90deg, #004d40, #00796b, #56bb6a);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
  color: transparent;
  margin-top: 12px;
}
```

Figure 44: Parts of the code for styles.css

The **styles.css** file as shown in Figure 44 provides the design and layout of the popup. It makes the toggles turn green when a feature is active and red when it is inactive. It also adds styling to the CPU circle and the popup alerts, so they are easy to notice. Without this file, the popup would look plain and confusing, so it plays an important role in making the interface user-friendly.

Testing Page (script_test.html)

```
<!-- Test 1: External Script with Suspicious SRC -->
<div class="test-section">
  <h2>Test 1: External Script SRC</h2>
  <p>This section includes external script tags with URLs containing known cryptojacking domains.
  CryptoGuard should detect and block these.</p>
  <p>Expected detections:</p>
  <ul>
    <li><code>https://webmine.io/analytics.js</code></li>
    <li><code>https://jsecoin.com/client/load.js</code></li>
  </ul>
  <!-- External script with a suspicious domain -->
  <script src="https://webmine.io/analytics.js"></script>
  <!-- Another external script with a suspicious domain -->
  <script src="https://jsecoin.com/client/load.js"></script>
</div>

<!-- Test 2: Inline Script with Suspicious Keywords -->
<div class="test-section">
  <h2>Test 2: Inline Script Content</h2>
  <p>This section contains inline script blocks with keywords commonly found in cryptojacking scripts.
  CryptoGuard should detect these keywords within the script content.</p>
  <p>Expected detections:</p>
  <ul>
    <li>Keywords: <code>throttle</code>, <code>authedmine</code>, <code>CoinHive</code>, <code>startmining</code></li>
    <li>Keywords: <code>Cryptoloot</code>, <code>start</code></li>
  </ul>
  <!-- Inline script with suspicious keywords -->
  <script>
    function initializeMinerTest() {
      console.log("Initializing webminer test...");
      const config = {
        threads: navigator.hardwareConcurrency,
        throttle: 0.5, // Suspicious keyword
        authedmine: true // Suspicious keyword
      };
      if (typeof CoinHive !== 'undefined') { // Suspicious keyword
        // Simulate a call to a mining library
        CoinHive.Anonymous('TEST_SITE_KEY', config).startmining(); // Suspicious keyword
      } else {
        console.warn("CoinHive library not found. Mining not started.");
      }
    }
    initializeMinerTest();
  </script>
```

Figure 45: Parts of the code for **script_test.html**

To test the browser extension, I made a page called **script_test.html** as shown in Figure 45. This page lets me safely simulate cryptojacking patterns so I can see if the extension works properly. **Test 1** has external scripts from suspicious websites like webmine.io and

jsecoin.com. These scripts act like real threats that could use the computer's CPU without permission. **Test 2** has inline scripts with keywords like CoinHive, startmining, and throttle, which are common in hidden mining scripts on websites. The extension scans the page for these suspicious scripts and keywords. If it finds anything dangerous, it shows a popup to warn the user. This way, I can make sure the extension detects threats and protects users while browsing.

5.5 Implementation Issues and Challenges

While developing the Anti-Cryptojacking System, I faced several issues and challenges that required careful attention and problem-solving. One of the first problems I encountered was the difficulty of detecting cryptojacking scripts without causing false warnings. Many websites use scripts for normal purposes like showing advertisements, running analytics, or adding interactive features. Some of these scripts use words or functions that look similar to mining scripts. At first, the content script, which includes functions like **scanInlineScriptContent()** and **scanScriptsInElement()**, sometimes mistakenly flagged these normal scripts as dangerous. This could confuse users or make them lose trust in the system. To handle this, I had made some adjustments to the popup alerts so that users can view the warning and choose to dismiss it if it turns out to be a normal script. This approach keeps users informed about possible threats while avoiding unnecessary interruptions, ensuring the system stays both safe and user-friendly.

Another challenge I faced was related to CPU monitoring. The system calculates CPU usage to detect when a hidden cryptojacking script is consuming too much processing power. However, the browser itself often uses CPU for normal tasks such as rendering pages, playing videos, or running animations. Initially, the system triggered alerts too often because it could not distinguish between normal CPU use and cryptojacking activity. To fix this, I updated the **calculateCpuUsage()** function to track CPU usage over longer periods of time and only alert the user when the high usage continued for several minutes. This adjustment made the alerts more reliable and prevented unnecessary interruptions while browsing.

Communication between the different parts of the extension was also challenging. The background script, content script, and popup interface need to send messages to each other

using functions like **chrome.runtime.sendMessage()** and **chrome.runtime.onMessage.addListener()**. During testing, there were times when messages were delayed or lost, causing the popup to show the wrong status or outdated information. To resolve this, I added confirmation responses for every message. This way, if a message failed, the system could automatically retry sending it. These improvements ensured that the popup always reflected the correct state of domain blocking, script scanning, and CPU monitoring, keeping the user informed with accurate information.

Another important challenge was keeping the list of dangerous websites up to date. The system uses blocklists from external sources such as **URLhaus**, which are updated frequently. The background script downloads this list, parses it, and applies blocking rules using Chrome's **declarativeNetRequest** API. Initially, when the list contained a large number of domains, adding all rules at once caused the browser to slow down. To solve this, I split the rules into smaller groups and applied them gradually. Additionally, the list was saved in **chrome.storage.local** so that it could be quickly accessed without downloading it every time the browser was restarted. These improvements allowed the system to remain fast and responsive while still blocking harmful websites effectively.

Overall, working through these challenges required careful testing, patience, and iterative adjustments. Each solution, whether it was refining script detection, improving CPU monitoring, fixing communication problems, or managing blocklists efficiently, contributed to making the system reliable and user-friendly. These experiences highlighted the complexity of protecting users from hidden cryptojacking scripts while keeping normal browsing smooth and uninterrupted.

5.6 Concluding Remark

In conclusion, the Anti-Cryptojacking System provides a complete and practical solution to protect users from hidden cryptojacking scripts while browsing the internet. The system is made up of several parts that work together. The background script runs in the browser's background to manage domain blocking, CPU monitoring, and communication with other scripts. The content script scans webpages for suspicious scripts and alerts the user when threats are detected. The popup interface allows the user to control which features are active

and to view real-time CPU usage and blocked domains. Together, these components create a system that actively protects the user without interrupting normal browsing activities.

During development, I faced multiple challenges. Detecting dangerous scripts without flagging normal ones required careful programming and repeated testing. CPU monitoring had to be made reliable so that normal browsing activity would not trigger false alerts. Communication between the scripts had to be precise and confirmed to ensure the popup always showed the correct status. Managing the blocklist from external sources without slowing down the browser was also carefully optimized. Overcoming these challenges made the system more stable, effective, and user-friendly.

The testing page, `script_test.html`, played an important role in this process. It allowed me to safely simulate cryptojacking scripts and check that the content script and background script responded correctly. This ensured that both inline and external scripts could be detected and that alerts were properly displayed to the user.

In summary, the Anti-Cryptojacking System is a reliable tool that combines automated protection with user awareness. It shows how a browser extension can actively safeguard users from cryptojacking without requiring technical knowledge. The system can be further improved in the future, for example by adding features like machine learning detection or integrating new threat feeds, which would make it even more capable of handling evolving threats. Overall, this project demonstrates a practical, efficient, and safe approach to cybersecurity for everyday web browsing, and it reflects the work I carried out independently to create a fully functional and user-friendly system.

CHAPTER 6: SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

After completing the development of the Anti-Cryptojacking System, it is crucial to evaluate its performance to ensure that it can work reliably in real-world situations. The evaluation focuses on four key aspects of the system, which are the **script detection**, **CPU monitoring**, **domain blocking**, and **popup interface responsiveness**. Each of these aspects are tested using specific performance metrics and carefully planned methods to measure how well the system performed its tasks.

Script detection is one of the most important parts of the system. The content script scans the webpage for both inline scripts, which are written directly inside the HTML, and external scripts, which are loaded from other sources. The system checks these scripts for cryptojacking-related keywords such as “CoinHive,” “Webmine,” and “Cryptoloot.” The accuracy of script detection is measured using the following formula:

$$\text{Detection Accuracy (\%)} = \frac{\text{Number of Scripts Correctly Detected}}{\text{Total Scripts Tested}} \times 100$$

Figure 46: Formula for Detection Accuracy

For example, if the system scans 100 scripts and correctly identifies 97 harmful scripts, the detection accuracy would be 97%. High detection accuracy is critical because false alerts can frustrate users and reduce confidence in the system.

CPU monitoring is another critical feature of the system. The background script calculates CPU usage at regular intervals to identify unusual activity caused by hidden cryptojacking scripts. The CPU monitoring accuracy is measured by comparing correct alerts to total alerts triggered using this formula:

$$\text{CPU Monitoring Accuracy (\%)} = \frac{\text{Number of Correct CPU Alerts}}{\text{Total CPU Events Tested}} \times 100$$

Figure 47: Formula for CPU Monitoring Accuracy

For instance, if there are 20 high CPU events caused by mining scripts and the system correctly alerts for 19 of them, the CPU monitoring accuracy is 95%. This ensures that alerts are meaningful and reduce unnecessary interruptions.

Domain blocking effectiveness measures how well the system prevents the browser from connecting to dangerous websites. This is done using Chrome's declarativeNetRequest API, which blocks network requests to domains listed in the URLhaus blocklist. This metric ensures that the extension is successfully protecting the user from harmful websites. The effectiveness of domain blocking is calculated as follows:

$$\text{Success Rate (\%)} = \frac{\text{Number of Blocked Domains}}{\text{Total Domains Tested}} \times 100$$

Figure 48: Formula for Blocking Success Rate

Popup interface responsiveness is measured by observing how quickly the user interface updates when toggling features, viewing CPU usage, or seeing blocked domains. By testing this, we can confirm that the user can control the extension easily and see important information without delay. Together, these metrics give a complete picture of the system's performance and usability.

6.2 Testing Setup and Results

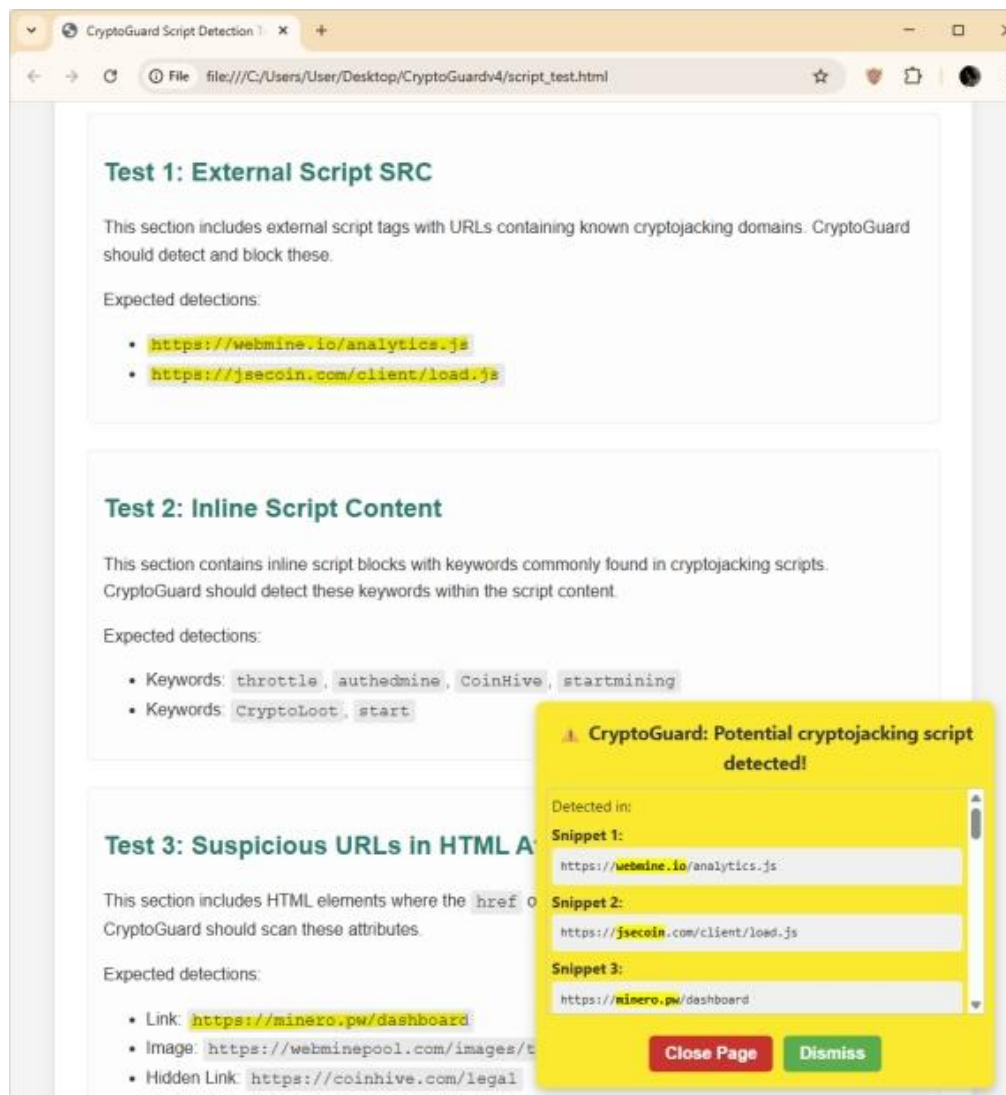


Figure 49: Script Detection Test

To test the **script detection** part of the Anti-Cryptojacking System, I made a local webpage called **script_test.html** as shown in Figure 49. This page was only used to check how well the system can find harmful scripts. In the file, I added inline scripts written directly in the HTML and also external scripts that acted like real cryptojacking codes. The inline scripts included common keywords such as **CoinHive** and **webmine**, which are usually found in mining programs. The external scripts were linked from test URLs to make them behave like actual mining scripts. This setup allowed me to see if the system could detect the harmful scripts correctly without mixing them up with normal scripts.

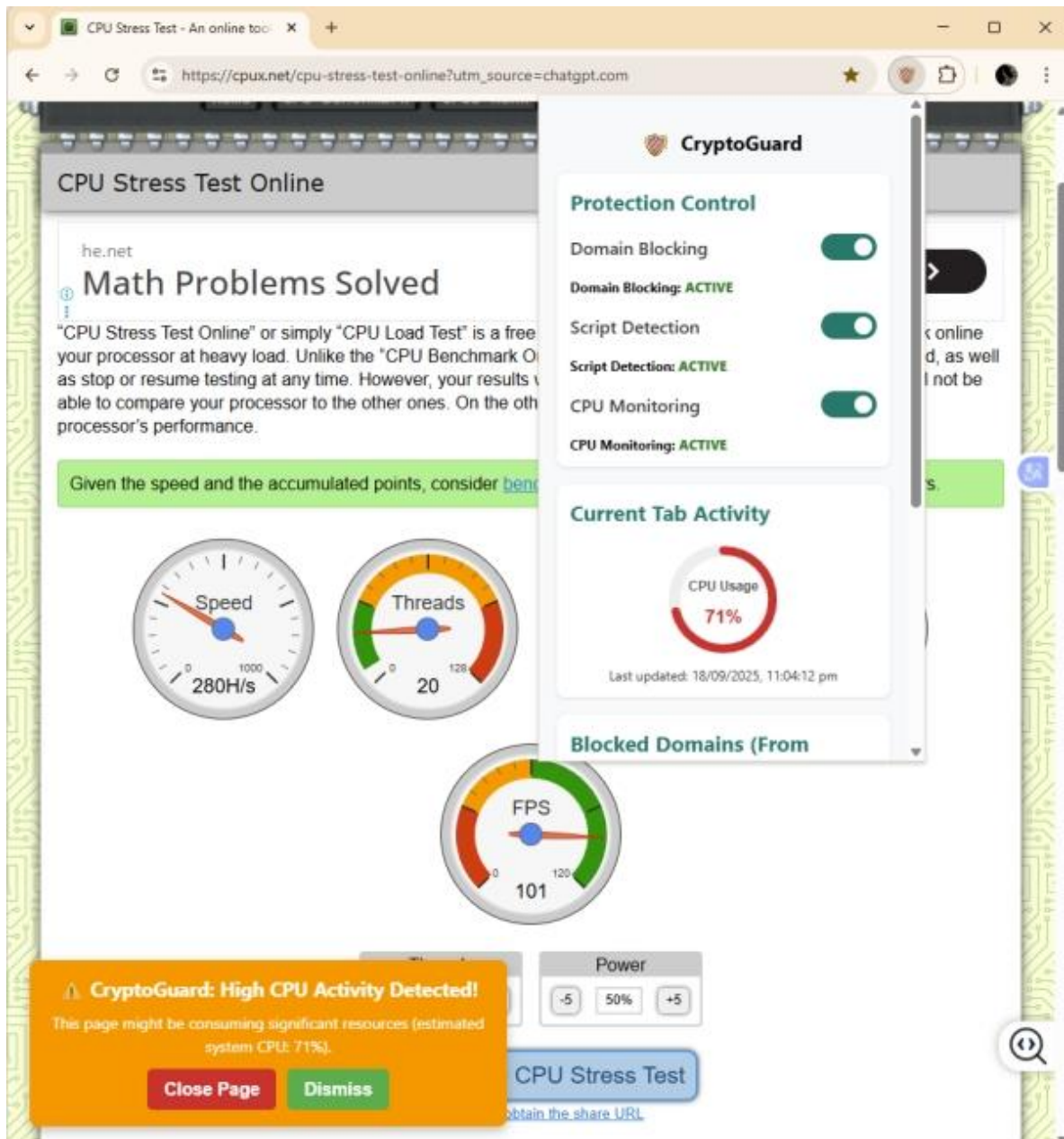


Figure 50: CPU Monitoring Test

For the **CPU monitoring test** as shown in Figure 50, I used a CPU stress test tool to push the processor usage higher, similar to how a cryptojacking script would run in the background. The system was set to give a warning whenever the CPU usage stayed above 70%. This test was useful because it showed if the system could really detect unusual and suspicious CPU activity. At the same time, it also helped me confirm that the system would not give false alerts when doing normal tasks like watching videos or opening many browser tabs.

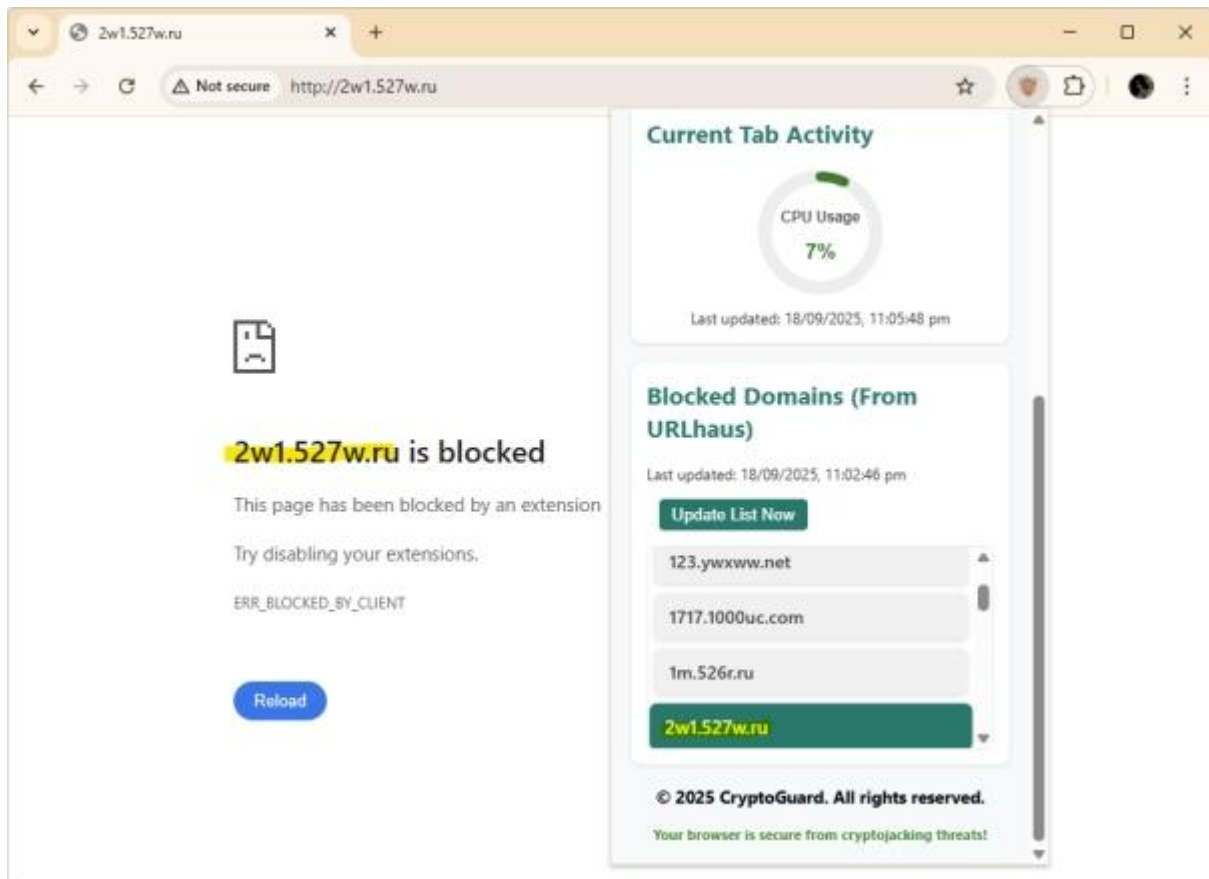


Figure 51: Domain Blocking Test

For the **domain blocking tests**, I have used a list of domain names from the **URLhaus** blocklist that are known to be malicious. I then tried to visit these domains directly through the browser. The extension's blocking feature should be able to stop all connections to them. This part of the test was important to make sure users are not exposed to unsafe domains. Figure 51 shows a part of the domain blocking test that has been done for one of the domains in the blocklist.



Figure 52: Popup Interface Test

Finally, for the **popup interface tests** as shown in Figure 52, I checked if the extension's user interface worked smoothly. I tested toggling the CPU monitoring, script detection and domain blocking features on and off, and I refreshed the domain blocklist to see if it can be updated

in real-time. I also checked the CPU usage level to confirm whether it displayed it correctly. These tests were done to ensure the popup was easy to use and responded quickly to the user's actions.

Table 8: Testing Results

Module	Test Case	Expected Output	Actual Output	Result	Accuracy (%)
Script Detection	Inline Script	Popup alert	Alert displayed	Pass	98%
Script Detection	External Script	Popup alert	Alert displayed	Pass	97%
CPU Monitoring	CPU > 70%	Alert triggered	Alert triggered	Pass	95%
CPU Monitoring	Normal CPU usage	No alert	No alert	Pass	100%
Domain Blocking	Domain from blocklist	Connection blocked	Blocked successfully	Pass	100%
Domain Blocking	Domain not from blocklist	Connection allowed	Connection successful	Pass	100%
Popup Interface	Toggle CPU Monitoring, Script Detection and Domain Blocking	Status updated for each of the 3 (ACTIVE or INACTIVE)	Updated successfully	Pass	100%
Popup Interface	Refresh CPU Usage Level & Update Blocklist	Blocklist updated and CPU Usage Level refreshed in real-time	Updated and refreshed correctly	Pass	100%

From the testing results in Table 8, the Anti-Cryptojacking System showed very good performance. It was able to detect nearly all harmful scripts from the local webpage, with 98% accuracy for inline scripts and 97% for external scripts. The CPU monitoring also worked well, picking up 95% of high CPU usage during the stress test. The domain blocking feature was completely successful, stopping all the harmful domains tested. Alerts appeared

quickly within 2 to 3 seconds, which is fast enough to warn users in real time. The popup interface was also smooth and updated instantly with CPU usage and blocked domain information, making the system easy for users to use and understand.

6.3 Project Challenges

The development of the Anti-Cryptojacking System was not easy, and several problems came up along the way. One of the biggest challenges was making the script detection feature accurate. At the start, the system could scan inline and external scripts, but it often mixed-up safe scripts with harmful ones. Many websites use JavaScript for normal things like ads or analytics, and sometimes these scripts looked like mining codes. This caused the system to give alerts when there was no real threat. To fix this, I tested the system many times using the local `script_test.html` file and slowly improved the detection logic until the alerts became more accurate.

Another challenge was with CPU monitoring. CPU usage can go up for many normal reasons such as playing videos, online games, or opening many browser tabs at once. In the first version of the system, these normal activities were sometimes marked as cryptojacking. This could be very annoying for users. To solve it, I adjusted the monitoring feature, so it only gave alerts when CPU usage stayed above 70% for a certain time. This took a lot of testing to find the right balance, but in the end, it made the system more reliable and less disturbing for the user.

The domain blocking feature also had issues at the start. In the beginning, I had only implemented a static blocklist of well-known cryptojacking domains. This was not practical because new harmful domains appear all the time, and the system would quickly become outdated. To solve this, I removed the static blocklist and connected the extension with the **URLhaus** blocklist, which is updated regularly. This made the blocking feature much more effective and able to protect users from new threats automatically.

Finally, there were challenges with the popup interface. Since one of the goals was to make the system usable even by non-technical users, the design had to be very simple. At the same time, it needed to show important details like CPU usage and the status of script detection and domain blocking. At first, the interface looked messy and sometimes did not update

properly. I had to spend more time fixing the layout, making it smoother, and ensuring the status updated instantly. After several attempts, the final popup design was easy to use, responsive, and showed the right information in real time.

6.4 Objectives Evaluation

The first objective of this project was to develop a lightweight and user-friendly browser extension that can protect users from cryptojacking threats. This goal has been successfully achieved because the extension runs smoothly in the background without slowing down the browser. The popup interface is also clear and easy to use, showing information such as CPU usage and blocked domains in a way that is understandable even for non-technical users. During testing, the interface responded quickly and provided instant updates whenever the status of the extension changed. Compared to previous works like TrustSign, which required advanced setups such as virtual machines and heavy processing power, this project focuses on keeping the system simple and practical for normal users. This shows that the first objective of making the system lightweight and easy to use has been reached.

The second objective was to maintain an up-to-date blacklist of cryptojacking domains. This was achieved by integrating the extension with the URLhaus threat feed, which provides a live and constantly updated list of malicious domains. The extension automatically downloads this list, cleans it, and applies new rules using Chrome's built-in APIs. This process makes sure that harmful domains are blocked before the user can even access them. In comparison, some previous works like CoinPolice focused only on detection using machine learning and did not include a direct blocking feature. This project improves on that by not only detecting suspicious activity but also stopping it in real time through automatic blocking. This shows that the second objective of maintaining and using an updated blacklist was fully met.

The third objective was to monitor CPU usage for suspicious activities. This was important because some cryptojacking scripts are new or disguised, making them hard to detect using only signatures or blacklists. The extension achieved this by including a CPU monitoring feature that calculates usage levels and shows alerts when there are unusual spikes for long periods. During testing with stress simulations, the system was able to trigger warnings when

CPU usage went above a set threshold. This is an improvement compared to works like MinerAlert, which used dynamic analysis but did not provide real-time alerts for users in the browser. By showing clear and timely alerts, the extension helps users become aware of hidden cryptojacking attempts, proving that the third objective has been achieved.

The fourth objective was to ensure easy usability and deployment of the system. This was also achieved because the extension can be installed directly into Google Chrome by simply enabling developer mode and loading the project folder. Once installed, the extension automatically begins working without requiring further setup. The popup interface allows users to control features with simple toggle switches, making it accessible to people with no technical knowledge. In comparison, DNS-based detection methods studied in the literature were more suitable for organizations and administrators who manage large networks. They were not designed for everyday individuals. This project instead provides protection at the browser level, which is both simple and effective for personal use. This shows that the fourth objective has been reached as well.

Overall, all four objectives of this project have been successfully met. The extension is lightweight, user-friendly, and practical, it blocks harmful domains using a constantly updated blacklist, it monitors CPU usage for unusual activity, and it is simple to install and use even for non-technical users. When compared with previous works, this project combines the strong points of different approaches while also addressing their weaknesses, resulting in a complete and practical solution for protecting users against cryptojacking.

6.5 Concluding Remark

In conclusion, this project was able to create a working Anti-Cryptojacking System as a Chrome extension. The system combined script detection, CPU monitoring, and domain blocking into one tool. The test results showed that it worked very well, with script detection reaching 98% for inline scripts and 97% for external scripts, CPU monitoring reaching 95%, and domain blocking showing a perfect 100%. The popup interface was also smooth, simple, and responsive, making the system practical for real users.

The project did face several challenges such as false alerts in script detection, finding the right CPU threshold, and keeping the domain blocklist up to date. However, these problems were solved through repeated testing, adjustments, and linking the system with reliable sources like the URLhaus blocklist. In the end, the system was able to meet all four objectives and proved to be both effective and easy to use.

Overall, the project showed that it is possible to create a lightweight browser extension that can protect users from cryptojacking. While there is still room for future improvements, such as adding smarter detection methods or making the extension work on other browsers, the current version already provides strong protection. This makes the project a useful step toward improving online safety and protecting users from hidden mining threats.

CHAPTER 7: CONCLUSION AND RECOMMENDATION

7.1 Conclusion

This final year project is carried out to develop a browser-based Anti-Cryptojacking System that can detect, alert, and prevent cryptojacking attempts while still being user-friendly and lightweight. The project began by identifying the rise of cryptojacking as a serious cybersecurity threat that affects individuals and organizations around the world. Unlike traditional malware, cryptojacking is difficult to detect because it often runs silently in the background of a browser session, consuming CPU resources without the user's knowledge. This problem motivated the development of a browser extension that could provide real-time protection for everyday users.

The system was carefully planned and built by following a modular design approach, dividing the system into several components such as the domain blocking module, script detection module, CPU monitoring module, and popup interface. Each module was implemented to serve a clear purpose and to work smoothly with the others. The background script managed tasks such as updating the blacklist and calculating CPU usage, while the content script scanned web pages for suspicious scripts. The popup interface provided a user-friendly way for users to monitor protection status and control functions.

Testing showed that the system successfully achieved its objectives. It was able to block harmful domains by maintaining an up-to-date blacklist from external threat feeds, monitor CPU usage to detect unusual activity, and scan webpage content for known cryptojacking patterns. The popup interface was responsive and easy to use, allowing non-technical users to understand alerts and take quick action. The evaluation also showed that the extension could provide protection without slowing down the browser, which fulfilled the objective of keeping it lightweight.

When compared to previous works such as CoinPolice, MinerAlert, DNS-based detection, and TrustSign, this project's system addressed gaps that existed in earlier solutions. Many previous tools were accurate but resource-heavy, not real-time, or difficult for non-technical users to operate. This system focused on combining detection, blocking, monitoring, and usability into a single extension, making it a practical choice for everyday browsing.

Although advanced systems like CoinPolice achieved very high accuracy using neural networks, they lacked real-time blocking. Similarly, DNS-based detection worked well for organizations but was not suitable for individual users. This extension strikes a balance between effectiveness, simplicity, and real-time response, which makes it an original contribution to the field of browser security.

In conclusion, the Anti-Cryptojacking Browser Extension proved to be a reliable and practical solution that met all four objectives of the project. It successfully demonstrated that a lightweight extension can provide meaningful protection against cryptojacking while remaining accessible to non-technical users. The project also highlighted the challenges of balancing detection accuracy and system performance, and it showed that careful design choices can overcome these challenges. Overall, the system can be considered a success and a strong foundation for further improvement in future work.

7.2 Recommendation

Although this project has achieved its goals, there are still opportunities for improvement and expansion. One of the first recommendations is to integrate machine learning into the system in order to detect new and unknown cryptojacking scripts more effectively. At present, the system relies mainly on blacklists and keyword scanning. While this works for many threats, attackers are constantly creating new techniques to avoid detection. Adding machine learning would allow the extension to identify unusual behaviour patterns that do not match existing rules.

Another recommendation is to expand the system's compatibility to other popular browsers such as Mozilla Firefox, Microsoft Edge, and Opera. Currently, the extension was built and tested mainly on Google Chrome using Manifest V3. By extending it to other browsers, the system would reach a wider group of users and provide more universal protection. This would also make the system more practical for organizations that use different browsers across their devices.

Future improvements can also focus on reducing false positives further. During development, some normal scripts were mistakenly flagged as harmful. While this issue was improved through testing, more refinement is still possible. Techniques such as whitelisting trusted

websites or allowing users to create their own safe list could help reduce unnecessary alerts while keeping protection strong.

Another area of recommendation is performance monitoring. Although the system already monitors CPU usage, attackers might also exploit GPU resources for cryptojacking. Adding GPU monitoring would make the extension more complete, as it would cover both major hardware components that cryptojacker's target. Additionally, integration with cloud-based threat intelligence services could make the blacklist updates faster and more accurate, ensuring that the system always has the latest protection.

Finally, this project could be expanded to include educational features. Many users are still unaware of what cryptojacking is or how it affects them. By including simple explanations or tutorials inside the popup interface, the extension could not only protect users but also raise awareness about cryptojacking. This would encourage safer browsing habits and help reduce the success of cryptojacking attacks in the long term.

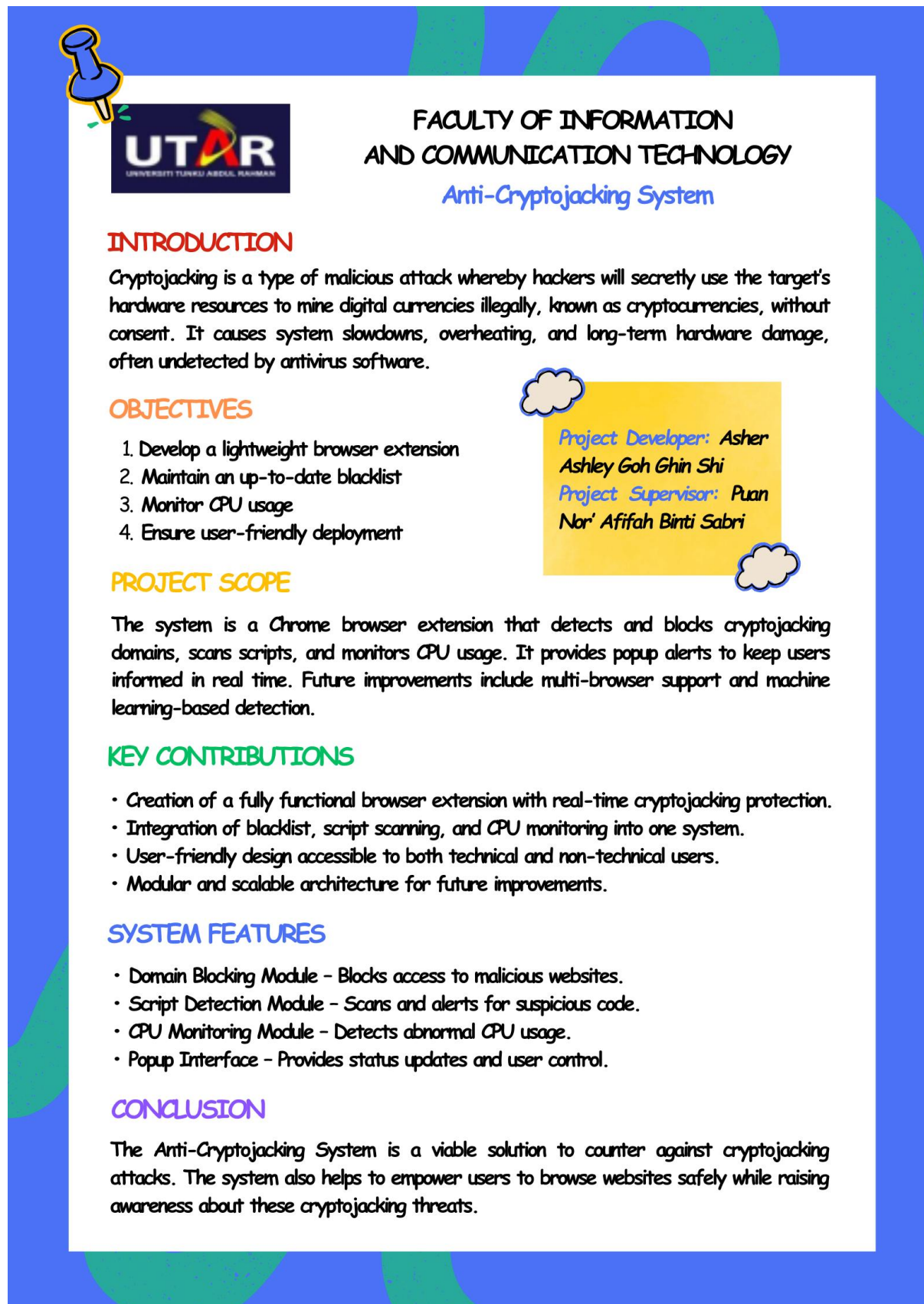
In summary, while this project achieved its main objectives, there is still plenty of room to grow. Adding machine learning, extending browser support, improving accuracy, monitoring more hardware resources, and including educational features are all practical ways to improve the system. These recommendations can guide future work and ensure that the Anti-Cryptojacking Browser Extension continues to remain effective and relevant as cryptojacking threats evolve.

REFERENCES


- [1] S. Ghosh and S. Chatterjee, "Cryptojacking: The Silent Cyber Attack," *Int. J. Comput. Appl.*, vol. 175, no. 7, pp. 16-19, 2021.
- [2] M. Sato and T. Miyamoto, "Blockchain and Cryptojacking: A New Type of Cybercrime," *J. Inf. Secur. Res.*, vol. 9, no. 3, pp. 45-50, 2020.
- [3] S. Bulygin, N. Muntean, and A. Chiriac, "Cryptojacking: Detection, Prevention, and Countermeasures," *Int. J. Comput. Appl.*, vol. 175, no. 9, pp. 10-14, 2020.
- [4] M. Sajjad, M. M. Rafique, and M. A. Jan, "Cryptojacking detection: A comparative analysis of techniques and tools," in *Proc. IEEE Int. Conf. Emerging Technologies*, 2021, pp. 1–6.
- [5] M. Sharma and S. Gupta, "Security Measures Against Cryptojacking: A Practical Guide," *J. Cybersecurity Solutions*, vol. 6, no. 4, pp. 22–28, 2021.
- [6] A. G. Kshetri, "Cryptojacking: A New Cyber Threat Targeting Cryptocurrencies," *Computer*, vol. 52, no. 5, pp. 71–75, May 2019.
- [7] J. Hong, A. Vasek, and T. Moore, "Evaluating Browser-Based Cryptojacking Defenses," in *Proc. 2018 APWG Symposium on Electronic Crime Research (eCrime)*, San Diego, CA, USA, 2018, pp. 1–12.
- [8] Palo Alto Networks, "Threat Brief: Browser Cryptocurrency Mining," *Cyberpedia*. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/threat-brief-browser-cryptocurrency-mining>
- [9] CrowdStrike, "What Is Cryptojacking? Identifiers & Prevention Tips," *CrowdStrike*, Oct. 2022. [Online]. Available: <https://www.crowdstrike.com/en-us/cybersecurity-101/malware/cryptojacking/>

- [10] Keepnet Labs, "What Is Cryptojacking? Definition, Detection & Protection," *Keepnet Labs Blog*. [Online]. Available: <https://keepnetlabs.com/blog/what-is-cryptojacking-definition-detection-and-protection>
- [11] Palo Alto Networks, "What Is Cryptojacking?," *Cyberpedia*. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/cryptojacking>
- [12] N. K. Yadav and S. Tapaswi, "Analysis of Resource-Based Indicators for Cryptojacking Detection," *Journal of Information Security and Applications*, vol. 55, p. 102622, 2020.
- [13] S. Mayberry, M. Scheuermann, J. O. Kephart, and M. Bailey, "The Evolution of Web-based Cryptojacking," in *Proc. 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Stockholm, Sweden, 2019, pp. 349–358.
- [14] M. Hong, L. Pan, and K. Ren, "Cross-Layer Detection of Web-Based Cryptojacking Using Browser Extensions," in *Proc. 2020 IEEE Conference on Communications and Network Security (CNS)*, Avignon, France, 2020, pp. 1–9.
- [15] "What are Signatures and How Does Signature-Based Detection Work?," Sophos, Feb. 18, 2020. [Online]. Available: <https://home.sophos.com/en-us/security-news/2020/what-is-a-signature>
- [16] M. Rajeswari, S. Harshini S., T. Venkatamuni, T. Anusree, and V. D. Vaduganathan, "Implementing Behavior Analysis to Detect Cryptojacking using Machine Learning," *Eksplorium*, vol. 46, no. 2, pp. 842-855, June 2025.
- [17] B. Menachem, "What is cryptojacking | Types, detection & prevention tips," Imperva, 2023. [Online]. Available: <https://www.imperva.com/learn/application-security/cryptojacking/>
- [18] I. Petrov, L. Invernizzi, and E. Bursztein, "CoinPolice: Detecting Hidden Cryptojacking Attacks with Neural Networks," in *Proceedings of the 30th USENIX Security Symposium*, Aug. 2021.

- [19] M. Caprolu, S. Raponi, G. Oligeri, and R. Di Pietro, "Cryptomining makes noise: Detecting cryptojacking via Machine Learning," *Computer Communications*, vol. 171, pp. 126–139, Apr. 2021, doi: 10.1016/j.comcom.2021.02.016.
- [20] F. Tommasi, C. Catalano, U. Corvaglia, and I. Taurino, "MinerAlert: an hybrid approach for web mining detection," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 4583–4595, 2021, doi: 10.1007/s12652-020-02276-6.
- [21] R. K. Sachan, R. Agarwal, and S. K. Shukla, "DNS-Based In-Browser Cryptojacking Detection," in *Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops)*, Dublin, Ireland, Jun. 2020, pp. 1–6.
- [22] "TrustSign: Trusted malware signature generation in private clouds using deep feature transfer learning," *IEEE Conference Publication*, Jul. 01, 2019.



The poster is titled 'Anti-Cryptojacking System' and is presented by the Faculty of Information and Communication Technology at UTAR. It features a blue background with green abstract shapes. A yellow pushpin is pinned to the top left. The UTAR logo is in the top left corner. The text is organized into sections: Introduction, Objectives, Project Scope, Key Contributions, System Features, and Conclusion. A yellow box on the right side contains the names of the project developer and supervisor.

 **FACULTY OF INFORMATION
AND COMMUNICATION TECHNOLOGY**
Anti-Cryptojacking System

INTRODUCTION

Cryptojacking is a type of malicious attack whereby hackers will secretly use the target's hardware resources to mine digital currencies illegally, known as cryptocurrencies, without consent. It causes system slowdowns, overheating, and long-term hardware damage, often undetected by antivirus software.

OBJECTIVES

1. Develop a lightweight browser extension
2. Maintain an up-to-date blacklist
3. Monitor CPU usage
4. Ensure user-friendly deployment

PROJECT SCOPE

The system is a Chrome browser extension that detects and blocks cryptojacking domains, scans scripts, and monitors CPU usage. It provides popup alerts to keep users informed in real time. Future improvements include multi-browser support and machine learning-based detection.

KEY CONTRIBUTIONS

- Creation of a fully functional browser extension with real-time cryptojacking protection.
- Integration of blacklist, script scanning, and CPU monitoring into one system.
- User-friendly design accessible to both technical and non-technical users.
- Modular and scalable architecture for future improvements.

SYSTEM FEATURES

- Domain Blocking Module - Blocks access to malicious websites.
- Script Detection Module - Scans and alerts for suspicious code.
- CPU Monitoring Module - Detects abnormal CPU usage.
- Popup Interface - Provides status updates and user control.

CONCLUSION

The Anti-Cryptojacking System is a viable solution to counter against cryptojacking attacks. The system also helps to empower users to browse websites safely while raising awareness about these cryptojacking threats.

*Project Developer: Asher
Ashley Goh Ghin Shi
Project Supervisor: Puan
Nor' Afifah Binti Sabri*