NFSU – Journal of Cyber Security and Digital Forensics
Volume – 4, Issue – 1, June 2025
E – ISSN – 2583-7559
International Conference on Role of Blockchain in Digital Forensics and Cyber Security - 2025

Automotive ECUs Cryptojacking: Simulation, Detection and Mitigation in a Virtualized Environment

Satya Sravani Konda[1]

[1],School of Cyber Security and Digital Forensics
National Forensic Sciences University
Gandhinagar, Gujarat, India
satyasravanikonda@gmail.com

Dr. Ravirajsinh Vaghela[2]

[2],School of Cyber Security and Digital Forensics
National Forensic Sciences University
Gandhinagar, Gujarat, India
ravirajsinh.vaghela@nfsu.ac.in

***Abstract*—: Cryptojacking is a covert cyberattack that mines Bitcoin without authorization by using computer resources. It poses a serious risk to Electronic Control Units (ECUs) for automobiles, and they remain underexplored. Due to their susceptibility to specific automotive attacks as firmware tampering, remote code execution (RCE), and others, ECUs may be compromised via network-based exploits or over-the-air (OTA) updates. Attackers can use cryptojacking malware on ECUs once they have gained access to them, which can cause system slowdown, higher power usage, and safety hazards for vehicles.**

The study showcases running cryptojacking malware (XMRig), keeping an eye on system resource usage, implementing a machine learning-based detection system, and utilizing blockchain technology to securely register events. Our solution sends email alerts, after identifying cryptojacking, and uses blockchain to for transparent logging.

The outcomes show anomaly detection, secure logging, and real-time monitoring. Our work gives a scalable method for examining new threats in automotive cybersecurity research.

Keywords: Cryptojacking, Automotive Cybersecurity, Electronic Control Unit, Virtual Machine, Cryptocurrency Mining Malware, Machine Learning, Anomaly Detection, Automated Threat Mitigation, Real-Time Monitoring, Blockchain, Security Alert System, Automotive Embedded Systems, Remote Code Execution.

I. Introduction

Cyberattacks target Electronic Control Units (ECUs) in automobiles. Advanced driver assistance systems (ADAS), infotainment, brakes, engine performance, and other vehicle systems are all managed by ECUs. However, they are vulnerable to cyberattacks due to their vulnerabilities to remote code execution (RCE), etc. [8][11]. Crypto-jacking is one such attack in which the attacker gains control of the ECU's computing capacity to mine cryptocurrencies without the owner's knowledge or approval.

Cryptojacking attacks can severely impact vehicle performance by overloading the ECU, increasing power consumption, overheating critical components, and causing system instability [6]. Unlike traditional malware that directly compromises vehicle safety, cryptojacking is designed to remain hidden while continuously consuming computational resources [2][3]. Performance deterioration, delayed reaction times in automotive operations, and even safety risks might result from this, especially in ECUs with limited resources [1][16]. Moreover, cryptojacking attacks can spread malware throughout car systems by taking advantage of unprotected communication channels and flaws in protocols like the Controller Area Network (CAN) bus [8][11].

Methods like CPU utilization monitoring, network traffic analysis, and machine learning-based anomaly detection work well for spotting cryptojacking activity. [4] [5] [15].

However, researchers are increasingly using virtualized settings to mimic real-world events because testing on physical ECUs is difficult. In order to simulate the behavior of automobile ECUs and evaluate the effects of cryptojacking attacks on these systems, this research suggests using a virtualized environment. Researchers can safely model cryptojacking scenarios, analyze system vulnerabilities, and assess potential mitigation techniques by simulating the setup and capabilities of an ECU [14].

This project's use of blockchain technology for cryptojacking detection improves the security of occurrences that are registered and produces an unchangeable and transparent record of harmful activity [23].
In order to protect automobile ECUs from cryptojacking, this study emphasizes the significance of automated mitigation techniques, secure recording, anomaly detection, and real-time monitoring. It also highlights virtualized environments as a good substitute for traditional cybersecurity research in automotive systems.

Scope of the Paper

The following topics are examined in this paper:

Threats to Automotive ECUs: Discussing Potential Cyber Threats and Their Vulnerabilities.

Cryptojacking in ECUs: Recognizing how ECUs are used for illicit cryptocurrency mining.

Machine Learning-Based Detection: Developing a model to identify cryptojacking activity using system performance metrics.

Automated Response Mechanism: Implementing a system that kills the mining process and alerts the service center.

II. Literature Review

There are now several detection methods and cybersecurity solutions designed specifically for vehicle ECUs because to the growing worry about cryptojacking in automotive systems. The Naseem

MINOS, a deep learning-based lightweight, real-time cryptojacking detection system, proposed at. [1] to identify illicit WebAssembly-based mining activity. Although it takes a novel approach, this solution is mostly concerned with web-based cryptojacking and might not directly address the particular requirements of vehicle ECUs. Likewise, a thorough analysis of cryptojacking malware is presented by Tapiador, Pastrana, and Suarez-Tangil [2], who examine its development, methods, and defenses. Nonetheless, their research provides a wide-ranging perspective without specifically addressing automobile settings.

Using contemporary methods such as intrusion detection systems (IDS) and genetic search algorithms, Chmiel [3] and Bar et al. [9] examine and apply cryptojacking detection strategies. Although intrusion detection systems (IDS) improve network security, evolutionary search algorithms are computationally demanding, which makes it difficult to implement them in ECUs with limited resources. Using CAN IDS, machine learning, and CPU utilization metrics, Caprolu et al. [15] and Gomes et al. [16] concentrate on cryptojacking detection and provide workable techniques that might be modified for real-time detection in automotive systems. The cross-stack strategy for cryptojacking defense put out by Kumar and Kumar [20] integrates many system security tiers and can be expanded to automobile cybersecurity frameworks.

In the context of automotive cybersecurity, Waqar and Anwar [6] explore the financial impact of cryptojacking on electric vehicles (EVs), analyzing how malicious mining affects battery life and charging infrastructure. Although the paper focuses on EVs, its findings could be adapted to traditional automotive systems. InfoBeyond Technology [8] introduces VehChain, a blockchain-inspired cryptographic solution designed to secure CAN bus communications against cyber threats. Kumar et al. [11] propose blockchain integration for CAN bus intrusion detection systems with ISO/SAE 21434 compliance, though cryptojacking-specific scenarios remain unexplored. El Fellah et al. [12] further investigate blockchain-based security solutions for connected vehicles, enhancing cybersecurity in the automotive supply chain. Additionally, Alkhatib et al. [21] explore the potential of BERT language models for CAN bus intrusion detection, showcasing advancements in AI-driven automotive cybersecurity.

For enhanced cryptojacking detection, Ying et al. [14] introduce CJSpector, a hardware trace and deep learning-based detection method that could enhance in-vehicle threat identification. Aponte-Novoa et al. [13] revisit classifier-based cryptojacking detection approaches, which can be adapted to vehicle ECUs for improved accuracy. Bursztein et al. [5] propose CoinPolice, a neural network-based system for detecting cryptojacking attacks. While neural networks are promising, newer detection methods must be considered. Tekiner, Aras, and Uluagac [4] introduce an IoT cryptojacking detection mechanism using network traffic analysis. Although designed for smart home networks, its techniques may inspire adaptive solutions for automotive environments.

Tanana [18] focuses on behavioral detection of cryptojacking malware, and Xu at [19] examined covert cryptojacking attacks, highlighting the need for continuous advancements in detection. As cryptojacking techniques evolve, approaches like integrating blockchain, AI, and cybersecurity protocols are essential for securing modern automotive systems against these threats.
Ahmad Ali et al. [23] introduce BCALS, a blockchain-based log management system, ensures tamper-proof cryptojacking incident records. The integration of immutable logging mechanisms can strengthen cybersecurity frameworks against emerging threats.

1. Methodology

To detect cryptojacking within automotive **Electronic Control Units (ECUs)**, we developed a **simulated environment** using a **virtual machine (VM)** to replicate an ECU's computational behavior. Since direct access to **Renesas automotive ECUs** was not available, our VM was configured to mimic the characteristics of **Renesas R-Car and Renesas RH850 microcontrollers**, which are widely used in modern vehicles.

Automotive Use of Renesas ECUs
Renesas ECUs are extensively used in various **automotive applications**, particularly in:

- **Infotainment and Connectivity:** The **Renesas R-Car series** is used in **Tesla, Toyota, Honda, Nissan, Mercedes Benz, Daimler truck and Volkswagen vehicles,** to power advanced **infotainment, digital cockpits, and driver assistance systems**.
- **High-Performance ECUs:** The **Renesas RH850 series** is deployed in **ADAS (Advanced Driver Assistance Systems), powertrain control, and body electronics** in **Mercedes-Benz, Ford, BMW, and Stellantis vehicles** for high-performance automotive computing.

1. Attack Simulation
To analyze cryptojacking activity, we executed **XMRig**, a widely used cryptocurrency mining software, within our **virtual ECU environment**. This simulated an unauthorized cryptojacking attack, mirroring real-world threats where malware infiltrates vehicle ECUs. The attack path was structured as follows:

1. **Initial Infection via Infotainment (Renesas R-Car):** The malware gains access through a compromised infotainment system, exploiting vulnerabilities in **Wi-Fi, Bluetooth, USB, or Over-the-Air (OTA) updates**.
2. **Lateral Movement to High-Performance ECU (Renesas RH850):** The malware spreads to a high-computation ECU, hijacking processing power for unauthorized cryptocurrency mining.
3. **Resource Exploitation:** The cryptojacking malware utilizes excessive CPU cycles, leading to **system slowdowns, overheating, and power drain** in the vehicle.

2. Data Collection
To monitor system behavior, we developed a **Python-based monitoring script (monitor_xmrig.py)**, which continuously logged key system performance metrics such as:

- **CPU usage spikes**
- **Memory consumption trends**
- **Active processes & mining signatures**
- **Power consumption fluctuations**

The collected data was stored in system_monitoring_data.csv for real-time analysis and detection.

3. Detection Mechanism

A **machine learning-based anomaly detection** model was developed to distinguish cryptojacking activity from normal ECU operations. Our approach included:

- **Supervised Learning:** Random Forest classifier training models on normal vs. cryptojacked system behavior.
- **Detection Script:** Implementing threshold-based rules to flag excessive CPU utilization linked to unauthorized mining.

4. Data Transfer & Analysis
To process and refine the collected system logs, we used **Secure Copy Protocol (SCP)** to transfer the data from the VM to a **Windows machine**. Feature extraction techniques were applied to differentiate cryptojacking anomalies from standard ECU operations.

5. Service Center Alert Mechanism
Upon detecting crypto jacking behavior, the system was programmed to:

1. **Terminate the malicious mining process** to prevent resource exploitation.
2. **Send an automated alert to the vehicle service center**, allowing for further analysis of vulnerabilities.

6. Blockchain Integration for Incident Logging
To ensure secure and tamper-proof logging of cryptojacking incidents, we integrated blockchain technology into our detection system. A private blockchain network (e.g., Hyperledger Fabric) was deployed to record detected cryptojacking events in an immutable ledger. This prevents unauthorized modifications and ensures transparency in forensic analysis.

Upon detecting cryptojacking, the system automatically logs key details—such as timestamps, affected ECU, and process information—onto the blockchain via smart contracts. Authorized personnel can retrieve and verify these records, enhancing trust, accountability, and regulatory compliance in automotive cybersecurity.

2. Lab Setup

The model **(Fig 1)** illustrates ,The cryptojacking detection and automotive cybersecurity lab which is designed to simulate attacks, monitor system performance, and detect cryptojacking activity in a controlled environment. The lab setup includes the following components:

1. Hardware Setup:
   - High-Performance Computer/Server: The lab uses a system with at least 4GB of RAM and a multi-core processor to run multiple virtual machines (VMs) and handle heavy processing tasks required for monitoring and detection.
   - Virtualization Platform (VMware/VirtualBox): These tools are used to create isolated test environments, where cryptojacking attacks and system monitoring can be simulated without affecting the host system. Each virtual machine is configured to replicate different attack scenarios and system configurations.
2. Network Setup:
   - The lab includes wireshark to analyse network traffic.
3. Operating Systems:
   - Linux (Kali): Used for the primary testing environment. It runs cryptojacking simulations and detection mechanisms while also monitoring system metrics.
4. Blockchain Lab Setup:

   To ensure **immutable logging** of cryptojacking incidents, we integrated blockchain technology using **Web3.py, Ganache, Infura, and Moralis** for decentralized and transparent record-keeping. The setup includes: Ganache functions as a local Ethereum test blockchain for simulating decentralized cryptojacking logs. The system uses smart contracts that were deployed on Ganache before their deployment to a live network. The detection system uses Infura to access a remote Ethereum node which enables blockchain interaction without requiring a complete node.

   The Python library Web3.py enables Python developers to interact with smart contracts on the Ethereum network. The detection script uses Web3.py to perform secure logging of cryptojacking events. The Moralis API enables users to interact with the blockchain through its simple methods for obtaining transaction logs and contract data and performing forensic analysis of crypto-jacking incidents.

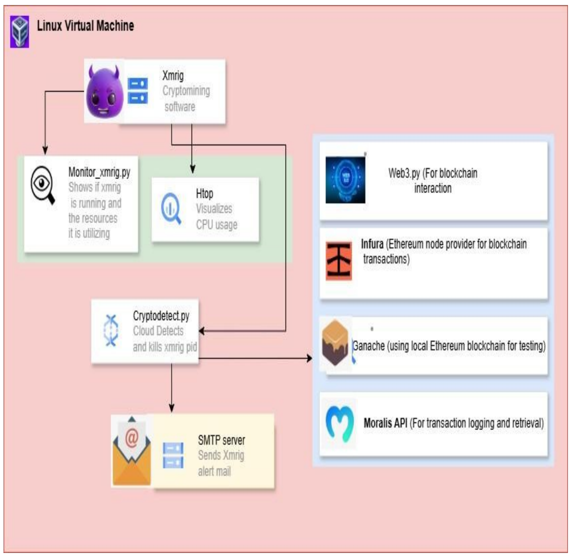

Linux Virtual Machine
Xmrig
Cryptomining software
Monitor_xmrig.py
Shows if xmrig is running and the resources it is utilizing
Htop
Visualizes CPU usage
Cryptodetect.py
Cloud Detects and kills xmrig pid
SMTP server
Sends Xmrig alert mail
Web3.py (For blockchain interaction
Infura (Ethereum node provider for blockchain transactions)
Ganache (using local Ethereum blockchain for testing)
Moralis API (For transaction logging and retrieval)

Fig. 1. The Proposed Cryptojacking detection framework

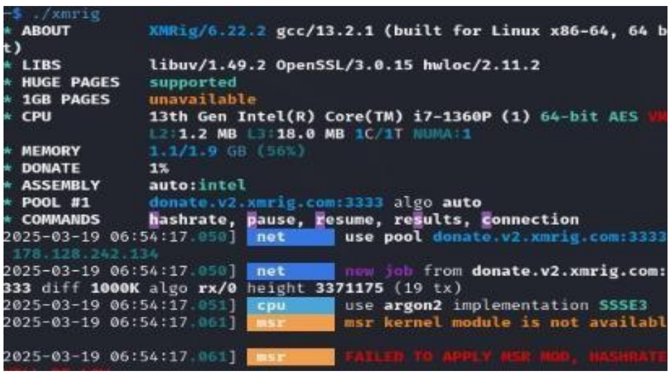
The setup involves various scripts working to monitor, detect, respond and mitigate to cryptojacking.

3. Tools Used and Configurations

This section outlines tools used in virtualized environment setup, including configurations of all the components shown in **(Fig 1)**

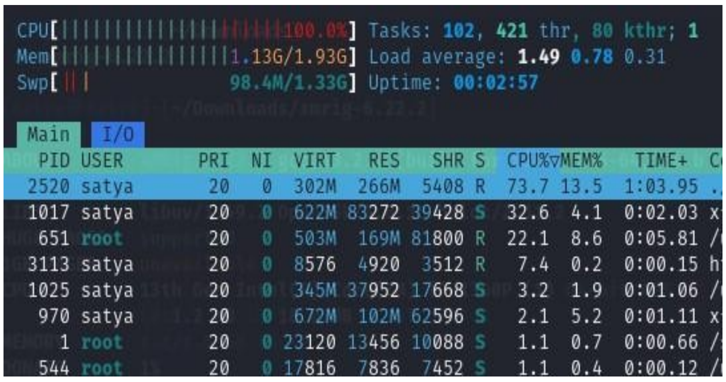
Cryptojacking Simulation:

XMRig: A cryptocurrency mining software as shown if **(Fig.2.)** used to simulate a crypto jacking attack. It is installed on a virtual machine to exploit system resources for cryptocurrency mining, simulating a real-world crypto jacking attack.



```
-$ ./xmrig
 * ABOUT        XMRig/6.22.2 gcc/13.2.1 (built for Linux x86-64, 64 b
t)
 * LIBS         libuv/1.49.2 OpenSSL/3.0.15 hwloc/2.11.2
 * HUGE PAGES   supported
 * 1GB PAGES    unavailable
 * CPU          13th Gen Intel(R) Core(TM) i7-1360P (1) 64-bit AES
                L2:1.2 MB L3:18.0 MB 1C/1T NUMA:1
 * MEMORY       1.1/1.9 GB (56%)
 * DONATE       1%
 * ASSEMBLY     auto:intel
 * POOL #1      donate.v2.xmrig.com:3333 algo auto
 * COMMANDS     hashrate, pause, resume, results, connection
2025-03-19 06:54:17.050]  net       use pool donate.v2.xmrig.com:3333
 178.128.242.134
2025-03-19 06:54:17.050]  net       new job from donate.v2.xmrig.com:
333 diff 1000K algo rx/0 height 3371175 (19 tx)
2025-03-19 06:54:17.051]  cpu       use argon2 implementation SSSE3
2025-03-19 06:54:17.061]  msr       msr kernel module is not availabl

2025-03-19 06:54:17.061]  msr       FAILED TO APPLY MSR MOD, HASHRATE
```

Fig.2 . simulation of xmrig (cryptomining script).



```
CPU[||||||||||||||||||||||100.0%] Tasks: 102, 421 thr, 80 kthr; 1
Mem[||||||||||||||||1.13G/1.93G] Load average: 1.49 0.78 0.31
Swp[|||              98.4M/1.33G] Uptime: 00:02:57

 Main  I/O
  PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  C
 2520 satya       20   0  302M  266M  5408 R  73.7 13.5  1:03.95 .
 1017 satya       20   0  622M 83272 39428 S  32.6  4.1  0:02.03 x
  651 root        20   0  503M  169M 81800 R  22.1  8.6  0:05.81 /
 3113 satya       20   0  8576  4920  3512 R   7.4  0.2  0:00.15 h
 1025 satya       20   0  345M 37952 17668 S   3.2  1.9  0:01.06 /
  970 satya       20   0  672M  102M 62596 S   2.1  5.2  0:01.11 x
    1 root        20   0 23120 13456 10088 S   1.1  0.7  0:00.66 /
  544 root        20   0 17816  7836  7452 S   1.1  0.4  0:00.12 /
```

Fig.3 . system resources usage visualizing tool.

- **htop**: A system monitoring tool used on the test machine to observe CPU, memory, and network usage during the cryptojacking attack. It helps visualize the resource drain caused by the cryptomining software as shown in **(Fig.3).**

Scripting and Automation:

- **Python**: Python scripts are used to monitor system performance during the cryptojacking attack. The script (monitor_xmrig.py) collects system metrics like CPU usage and memory consumption, which are logged into a CSV file.
- **SCP (Secure Copy Protocol)**: Used to securely transfer the collected monitoring data (CSV logs) from the Linux VM to a Windows machine for analysis and script making.

Detection Mechanism:

- **Custom Detection Algorithm**: The detection mechanism is based on analyzing the system metrics collected during the cryptojacking attack. A machine learning algorithm is used to identify abnormal resource consumption patterns indicative of cryptojacking.
- **Alert System**: Python scripts automatically trigger alerts when cryptojacking is detected, sending notifications to a service center for intervention and also terminates the process running

Blockchain Logging:

To **securely log cryptojacking incidents**, a blockchain-based logging mechanism was integrated using **Web3.py, Ganache, Infura, and Moralis**. The configurations include:

- **Ganache:** Used as a **local Ethereum test blockchain** to simulate a decentralized cryptojacking log. Smart contracts were deployed on Ganache before moving to a live network.
- **Infura:** Provides a **remote Ethereum node**, enabling the detection system to interact with the blockchain without the need to run a full node.
- **Web3.py:** A Python library used to communicate with Ethereum-based smart contracts. The detection script interacts with Web3.py to **log cryptojacking events** securely.
- **Moralis API:** Facilitates blockchain interaction by providing easy access to transaction logs, contract data, and forensic analysis of cryptojacking incidents.

4. Experimental Work and Results Analysis

Performance Metrics of Detection Algorithm

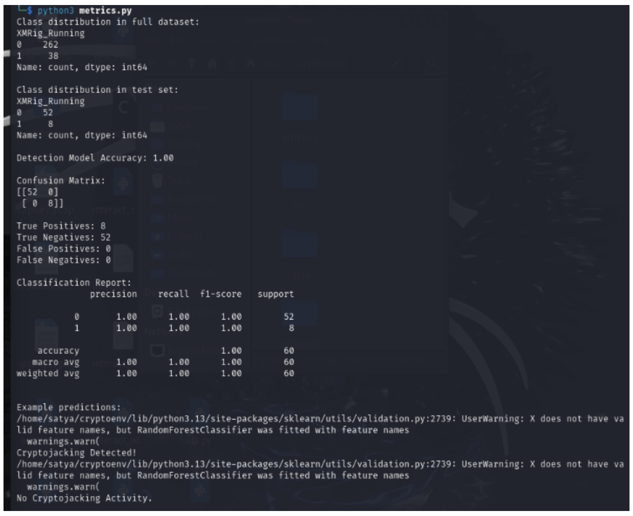
The detection model was evaluated using classification metrics such as accuracy, precision, recall, F1-score, and a confusion matrix **Fig.4.** The performance on test data gave the following results:

Table 1. Predicted and Actual Values

	Predicted Positive	Predicted Negative
Actual Positive	8	0
Actual Negative	0	52

Table 2. Performance Metrics:

Metric	Class 0	Class 1	Accuracy
Precision	**1.00**	**1.00**	**1.00**
Recall	**1.00**	**1.00**	**1.00**
F1- Score	**1.00**	**1.00**	**1.00**



```
└─$ python3 metrics.py
Class distribution in full dataset:
XMRig_Running
0    262
1     38
Name: count, dtype: int64

Class distribution in test set:
XMRig_Running
0    52
1     8
Name: count, dtype: int64

Detection Model Accuracy: 1.00

Confusion Matrix:
[[52  0]
 [ 0  8]]

True Positives: 8
True Negatives: 52
False Positives: 0
False Negatives: 0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        52
           1       1.00      1.00      1.00         8

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60


Example predictions:
/home/satya/cryptoenv/lib/python3.13/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have va
lid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
Cryptojacking Detected!
/home/satya/cryptoenv/lib/python3.13/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have va
lid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
No Cryptojacking Activity.
```

Fig.4.Performance metrics of detection algorithm

The model achieved a perfect detection accuracy of 100%, with no false positives or false negatives recorded in the test set. This confirms the robustness of the Random Forest-based detection mechanism on the given dataset.

Feasibility Analysis

The implementation was tested in a virtualized environment that closely mimics real automotive ECUs, proving the feasibility of using lightweight Python scripts and machine learning models for Anomaly detection without overloading system resources. Blockchain-based logging was successfully deployed using a local Ethereum test network (Ganache) and Web3.py integration.

The system can be realistically implemented in:

- Automotive ECUs with Linux-based OS
- Telematics or infotainment gateways
- Fleet management diagnostic servers

The modular nature of the detection and logging system ensures easy deployment with minimal performance overhead.

Feasibility Evaluation:

Feasible-

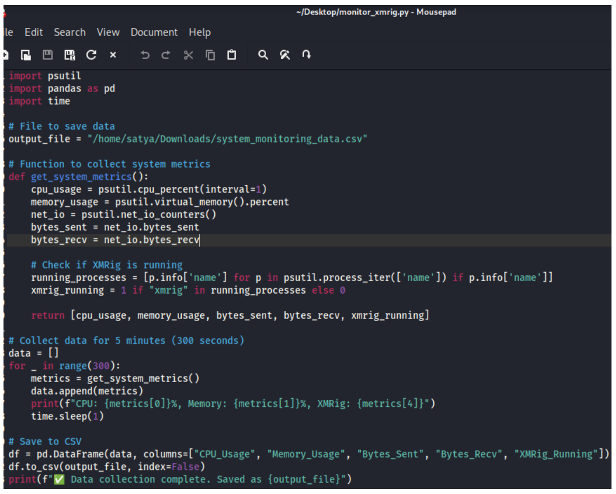
- Detection using System Metrics
- Machine Learning Detection (Random Forest)
- Process Termination
- Email Alert System
- Blockchain Logging
- VM-based Dataset Generation

Pending - Integration into Existing ECU CAN IDS

We simulated crypto mining using XMRig software, which was deployed on the Linux VM to mine cryptocurrency and use system resources. The attack was monitored using htop, which tracked CPU, memory, and network usage. We made Python scripts to log these metrics into CSV files for further analysis.

Data Collection & Monitoring:

During the cryptojacking attack, system metrics such as CPU usage, memory consumption, and network activity were monitored and logged by the code with wrote.Logs were collected over a predefined period, typically 15 minutes to 1 hour. The SCP protocol was used to transfer these logs from the Linux machine to a Windows machine for further processing and analysis. The following **(Fig. 5)** shows the script for monitoring system resources, their usage, and crypto mining activity.

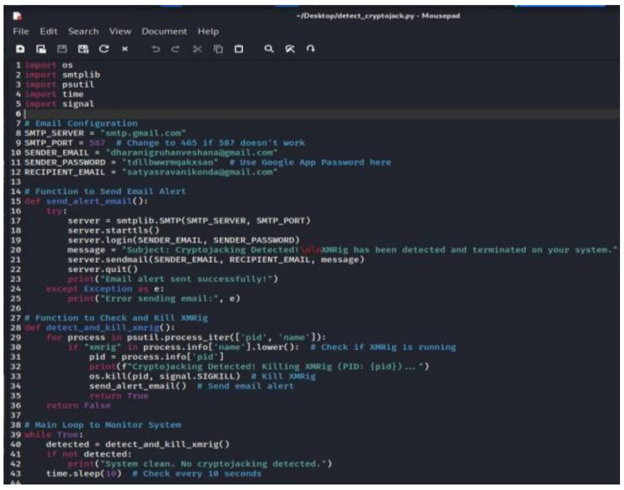

~/Desktop/monitor_xmrig.py - Mousepad
ile Edit Search View Document Help

```python
import psutil
import pandas as pd
import time

# File to save data
output_file = "/home/satya/Downloads/system_monitoring_data.csv"

# Function to collect system metrics
def get_system_metrics():
    cpu_usage = psutil.cpu_percent(interval=1)
    memory_usage = psutil.virtual_memory().percent
    net_io = psutil.net_io_counters()
    bytes_sent = net_io.bytes_sent
    bytes_recv = net_io.bytes_recv

    # Check if XMRig is running
    running_processes = [p.info['name'] for p in psutil.process_iter(['name']) if p.info['name']]
    xmrig_running = 1 if "xmrig" in running_processes else 0

    return [cpu_usage, memory_usage, bytes_sent, bytes_recv, xmrig_running]

# Collect data for 5 minutes (300 seconds)
data = []
for _ in range(300):
    metrics = get_system_metrics()
    data.append(metrics)
    print(f"CPU: {metrics[0]}%, Memory: {metrics[1]}%, XMRig: {metrics[4]}")
    time.sleep(1)

# Save to CSV
df = pd.DataFrame(data, columns=["CPU_Usage", "Memory_Usage", "Bytes_Sent", "Bytes_Recv", "XMRig_Running"])
df.to_csv(output_file, index=False)
print(f"✅ Data collection complete. Saved as {output_file}")
```

Fig.5. Script for monitoring resources.

Detection Mechanism:

(Fig. 6) shows the script for detecting cryptojacking activity, terminating the XMRig software and sending an alert email to notify service centers.


~/Desktop/detect_cryptojack.py - Mousepad
File Edit Search View Document Help

```python
import os
import smtplib
import psutil
import time
import signal

# Email Configuration
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587  # Change to 465 if 587 doesn't work
SENDER_EMAIL = "dharanigruhanveshana@gmail.com"
SENDER_PASSWORD = "tdllbwwrmqakxsao"  # Use Google App Password here
RECIPIENT_EMAIL = "satyasravanikonda@gmail.com"

# Function to Send Email Alert
def send_alert_email():
    try:
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()
        server.login(SENDER_EMAIL, SENDER_PASSWORD)
        message = "Subject: Cryptojacking Detected!\n\nXMRig has been detected and terminated on your system."
        server.sendmail(SENDER_EMAIL, RECIPIENT_EMAIL, message)
        server.quit()
        print("Email alert sent successfully!")
    except Exception as e:
        print("Error sending email:", e)

# Function to Check and Kill XMRig
def detect_and_kill_xmrig():
    for process in psutil.process_iter(['pid', 'name']):
        if "xmrig" in process.info['name'].lower():  # Check if XMRig is running
            pid = process.info['pid']
            print(f"Cryptojacking Detected! Killing XMRig (PID: {pid})...")
            os.kill(pid, signal.SIGKILL)  # Kill XMRig
            send_alert_email()  # Send email alert
            return True
    return False

# Main Loop to Monitor System
while True:
    detected = detect_and_kill_xmrig()
    if not detected:
        print("System clean. No cryptojacking detected.")
    time.sleep(10)  # Check every 10 seconds
```

Fig.6. Script for detecting cryptojacking.

Once the data was transferred to the Windows system, it was analyzed to detect cryptojacking characters, such as high CPU utilization and memory consumption. To find these patterns, a machine learning-based bespoke detection method was created. After analyzing the data, our detection system identified abnormal resource usage as cryptojacking activity. The service center was informed of the attack via the notifications that the system issued.

Blockchain-Based Incident Logging

To record the attacks, a blockchain-based recording system was put into the detection framework. This prevents tampering and unwanted changes, ensuring that incidents are safely documented on an immutable ledger. Instead of running a whole node, the system uses Web3.py to communicate with the Ethereum blockchain through Infura. In order to specify how cryptojacking instances are recorded with timestamps, system IDs, and anomalous data, a smart contract was implemented on Ganache. Transparency and traceability are ensured when the technology immediately records an assault on the blockchain.

Blockchain-stored data is retrieved via the Moralis API, allowing authorized service centers to validate cryptojacking instances. By keeping a decentralized, impenetrable record of cryptojacking detections, this integration enhances forensic analysis of attacks that occured.

Result Analysis

- **CPU consumption:** As the XMRig program took advantage of the resources to mine cryptocurrency, CPU consumption increased to 100% during the attack.

- **Memory Consumption**: Memory usage also showed a noticeable increase, which was another indicator of the cryptojacking attack.
- **Network Activity**: Anomalies in network traffic were also observed, as cryptojacking software communicates with remote servers for mining data. **Blockchain Logging Verification:** Every detected cryptojacking log was recorded on the blockchain, ensuring secure traceability..

The detection system successfully identified these resource usage anomalies in real time and generated alerts. The alerts were then forwarded to the service center for remediation. The response time of the system was also evaluated to ensure that alerts were triggered within a reasonable timeframe.

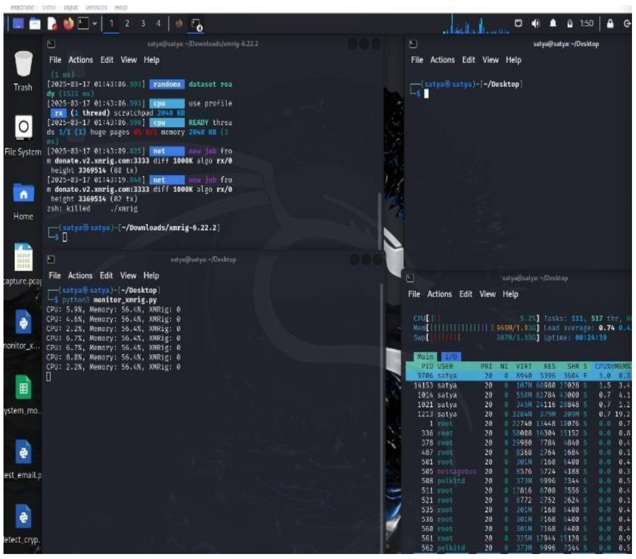
4.1. Scenarios

Scenario 1: Initial System Monitoring (Before Cryptojacking Attack)

In the first scenario, system resources were monitored without any attack. The system was running normally, and the **htop** tool was used to check CPU and memory utilization. The **Python monitoring script** was running, and the system metrics (CPU, memory, and resource usage) were logged into a **CSV file**. In this case, the logs showed normal resource usage with no significant spikes. The script gave the result :

- **CPU**: 5%, **Memory**: 45%, **XMRig**: 0

As shown in **(Fig. 7),** the system remained in a stable state with no cryptojacking activity detected, as expected.



```
(1 ms)
[2025-03-17 01:43:86.593] randomx dataset rea
dy (1523 ms)
[2025-03-17 01:43:86.593] cpu      use profile
 rx (1 thread) scratchpad 2048 KB
[2025-03-17 01:43:86.596] cpu      READY threa
ds 1/1 (1) huge pages 0% 0/1 memory 2048 KB (3
ms)
[2025-03-17 01:43:89.825] net      new job fro
m donate.v2.xmrig.com:3333 diff 1000K algo rx/0
 height 3369514 (88 tx)
[2025-03-17 01:43:19.848] net      new job fro
m donate.v2.xmrig.com:3333 diff 1000K algo rx/0
 height 3369514 (82 tx)
zsh: killed     ./xmrig

┌──(satya㉿satya)-[~/Downloads/xmrig-6.22.2]
└─$ 
```

```
┌──(satya㉿satya)-[~/Desktop]
└─$ python3 monitor_xmrig.py
CPU: 5.9%, Memory: 56.4%, XMRig: 0
CPU: 4.6%, Memory: 56.4%, XMRig: 0
CPU: 2.2%, Memory: 56.4%, XMRig: 0
CPU: 6.7%, Memory: 56.4%, XMRig: 0
CPU: 6.7%, Memory: 56.4%, XMRig: 0
CPU: 8.8%, Memory: 56.4%, XMRig: 0
CPU: 2.2%, Memory: 56.4%, XMRig: 0
```

```
CPU[||                5.2%] Tasks: 111, 517 thr,
Mem[||||||||||||||969M/1.93G] Load average: 0.74 0.4
Swp[||||||       307M/1.33G] Uptime: 00:24:19

Main  I/O
  PID USER       PRI  NI  VIRT   RES   SHR S  CPU% MEM%
 9706 satya       20   0  8940  5396  3604 R   3.0  0.3
14153 satya       20   0  107M 68980 27028 S   1.5  3.4
 1014 satya       20   0  558M 82784 43000 S   0.7  4.1
 1021 satya       20   0  345M 24116 28848 S   0.7  1.2
 1213 satya       20   0 3284M  379M  209M S   0.7 19.2
    1 root        20   0 22740 13448 10076 S   0.0  0.7
  336 root        20   0 58088 16304 15152 S   0.0  0.8
  378 root        20   0 29980  7784  4840 S   0.0  0.4
  487 root        20   0  8368  2764  1604 S   0.0  0.1
  501 root        20   0  301M  7168  6400 S   0.0  0.4
  505 messagebus  20   0  8576  5724  4188 S   0.0  0.3
  508 polkitd     20   0  373M  9996  7344 S   0.0  0.5
  511 root        20   0 17816  8708  7556 S   0.0  0.4
  521 root        20   0  6772  2752  2624 S   0.0  0.1
  535 root        20   0  301M  7168  6400 S   0.0  0.4
  536 root        20   0  301M  7168  6400 S   0.0  0.4
  560 root        20   0  301M  7168  6400 S   0.0  0.4
  561 root        20   0  325M 17944 15128 S   0.0  0.9
  562 polkitd     20   0  373M  9996  7344 S   0.0  0.5
```

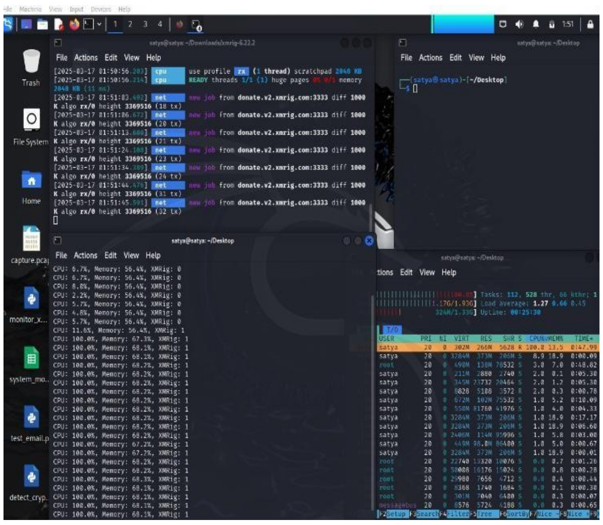
Fig.7. Scenario before running cryptojacking tool.

Scenario 2: Running XMRig (Cryptojacking Attack Simulation)

In the second scenario, we executed **XMRig** on the VM to simulate the attack. The moment **XMRig** was started, it began to consume CPU and memory resources for mining. This increase in resource consumption was immediately visible in the system's **htop** tool, where CPU and memory usage began to spike significantly. The monitoring script logged the following data:

- **CPU**: 100%, **Memory**: 68%, **XMRig**: 1

•



```
[2025-03-17 01:50:56.203]  cpu   use profile  rx  (1 thread) scratchpad 2048 KB
[2025-03-17 01:50:56.214]  cpu   READY threads 1/1 (1) huge pages 0% 0/1 memory
2048 KB (11 ms)
[2025-03-17 01:51:03.492]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (18 tx)
[2025-03-17 01:51:06.672]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (20 tx)
[2025-03-17 01:51:13.680]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (21 tx)
[2025-03-17 01:51:24.108]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (23 tx)
[2025-03-17 01:51:34.389]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (24 tx)
[2025-03-17 01:51:44.470]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (31 tx)
[2025-03-17 01:51:45.591]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (32 tx)
```

```
CPU: 6.7%, Memory: 56.4%, XMRig: 0
CPU: 6.7%, Memory: 56.4%, XMRig: 0
CPU: 8.8%, Memory: 56.4%, XMRig: 0
CPU: 2.2%, Memory: 56.4%, XMRig: 0
CPU: 5.7%, Memory: 56.4%, XMRig: 0
CPU: 4.8%, Memory: 56.4%, XMRig: 0
CPU: 5.7%, Memory: 56.4%, XMRig: 0
CPU: 11.6%, Memory: 56.4%, XMRig: 1
CPU: 100.0%, Memory: 67.3%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 67.2%, XMRig: 1
CPU: 100.0%, Memory: 67.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.3%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
```

Fig.8. After running XMRig , attack was detected.

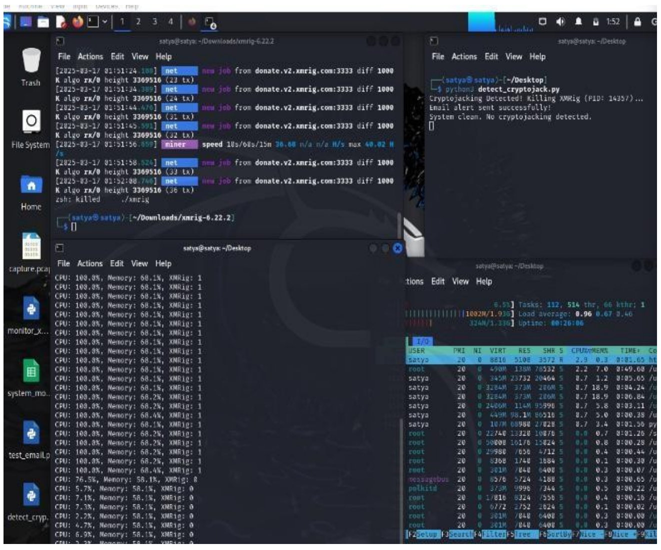
As shown in **(Fig .8)** a clear anomaly, indicating that the cryptojacking software was exploiting the system resources. The **Python script** continuously updated the logs to capture these changes, which were then used for detection.

Scenario 3: Detection Mechanism and Response

In this scenario, the **detection mechanism** was triggered. The detection algorithm, running in the background, analyzed the data from the monitoring script and identified the increase in CPU and memory usage. It recognized cryptojacking activity caused by **XMRig** and automatically executed the following actions:

Terminate XMRig Process: The detection mechanism automatically terminated the XMRig process, stopping the mining activity.

1. **System Recovery**: The CPU and memory usage returned to normal levels As shown in **(Fig .9).** .



```
[2025-03-17 01:51:24.108]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (23 tx)
[2025-03-17 01:51:34.389]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (24 tx)
[2025-03-17 01:51:44.470]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (31 tx)
[2025-03-17 01:51:45.591]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (32 tx)
[2025-03-17 01:51:56.859]  miner  speed 10s/60s/15m 36.68 n/a n/a H/s max 40.02 H
/s
[2025-03-17 01:51:58.524]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (33 tx)
[2025-03-17 01:52:08.746]  net   new job from donate.v2.xmrig.com:3333 diff 1000
K algo rx/0 height 3369516 (36 tx)
zsh: killed     ./xmrig

┌──(satya㉿satya)-[~/Downloads/xmrig-6.22.2]
└─$ 
```

```
┌──(satya㉿satya)-[~/Desktop]
└─$ python3 detect_cryptojack.py
Cryptojacking Detected! Killing XMRig (PID: 14357)...
Email alert sent successfully!
System clean. No cryptojacking detected.
```

```
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.4%, XMRig: 1
CPU: 100.0%, Memory: 68.1%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.2%, XMRig: 1
CPU: 100.0%, Memory: 68.4%, XMRig: 1
CPU: 76.5%, Memory: 58.1%, XMRig: 0
CPU: 5.7%, Memory: 58.1%, XMRig: 0
CPU: 7.1%, Memory: 58.1%, XMRig: 0
CPU: 7.3%, Memory: 58.1%, XMRig: 0
CPU: 2.2%, Memory: 58.1%, XMRig: 0
CPU: 4.7%, Memory: 58.1%, XMRig: 0
CPU: 6.9%, Memory: 58.1%, XMRig: 0
```

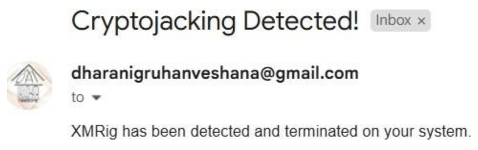
Fig.9. Detected Xmrig was killed and system is clean**.**

2. **Alert Generation**: The detection system generated an alert and sent an email notification to the service center, detailing the cryptojacking event. The email included information about the incident and the action taken (i.e., termination of **XMRig**).

After the detection mechanism executed, the logs showed:

- **CPU**: 7% (Returned to normal), **Memory**: 35% (Returned to normal), **XMRig**: 0 (Process terminated)

As shown in **(Fig .10)** ,The service center received the email alert, confirming the incident was detected and mitigated.



Cryptojacking Detected! Inbox ×

dharanigruhanveshana@gmail.com
to

XMRig has been detected and terminated on your system.

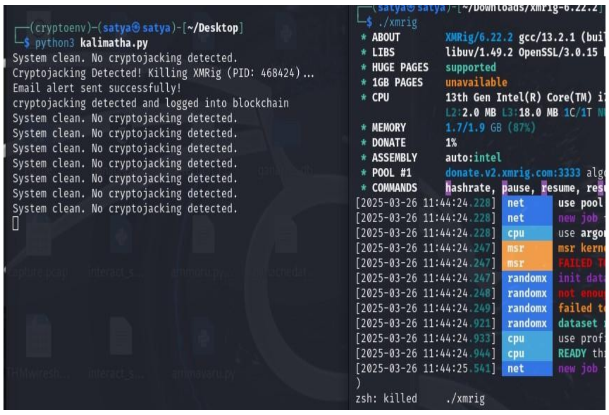
Fig.10. Alert mail sent after attack detection

NFSU – Journal of Cyber Security and Digital Forensics
Volume – 4, Issue – 1, June 2025
E – ISSN – 2583-7559
International Conference on Role of Blockchain in Digital Forensics and Cyber Security - 2025

Scenario 4: Blockchain-Based Incident Logging

In this scenario, once cryptojacking activity was detected, the system **automatically recorded the incident on the blockchain** to ensure **data integrity and transparency**. Using **Web3.py and Infura**, the detection script triggered a smart contract deployed on **Ganache**, logging details **of transaction**. This information was permanently stored on the blockchain, preventing unauthorized modifications or deletions.

As shown in **(Fig. 11)**, the blockchain ledger **successfully recorded** the cryptojacking detection event, ensuring that the incident report remains **tamper-proof** and accessible for future forensic analysis. This blockchain-based approach enhances the security, reliability, and transparency of cryptojacking incident reporting in automotive cybersecurity.



```
┌──(cryptoenv)-(satya㉿satya)-[~/Desktop]
└─$ python3 kalimatha.py
System clean. No cryptojacking detected.
Cryptojacking Detected! Killing XMRig (PID: 468424)...
Email alert sent successfully!
cryptojacking detected and logged into blockchain
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
System clean. No cryptojacking detected.
▯
```

```
┌──(satya㉿satya)-[~/Downloads/xmrig-6.22.2]
└─$ ./xmrig
 * ABOUT        XMRig/6.22.2 gcc/13.2.1 (bui
 * LIBS         libuv/1.49.2 OpenSSL/3.0.15
 * HUGE PAGES   supported
 * 1GB PAGES    unavailable
 * CPU          13th Gen Intel(R) Core(TM) i
                L2:2.0 MB L3:18.0 MB 1C/1T N
 * MEMORY       1.7/1.9 GB (87%)
 * DONATE       1%
 * ASSEMBLY     auto:intel
 * POOL #1      donate.v2.xmrig.com:3333 alg
 * COMMANDS     hashrate, pause, resume, res
[2025-03-26 11:44:24.228]  net      use pool
[2025-03-26 11:44:24.228]  net      new job
[2025-03-26 11:44:24.228]  cpu      use argo
[2025-03-26 11:44:24.247]  msr      msr kern
[2025-03-26 11:44:24.247]  msr      FAILED T
[2025-03-26 11:44:24.247]  randomx  init dat
[2025-03-26 11:44:24.248]  randomx  not enou
[2025-03-26 11:44:24.249]  randomx  failed t
[2025-03-26 11:44:24.921]  randomx  dataset
[2025-03-26 11:44:24.933]  cpu      use prof
[2025-03-26 11:44:24.944]  cpu      READY th
[2025-03-26 11:44:25.541]  net      new job
)
zsh: killed     ./xmrig
```

Fig.11. Blockchain logging the identified cryptojacking

5. Conclusion

Our crypto-jacking detection system identified and mitigated cryptojacking attacks in a controlled environment. By simulating a cryptojacking attack using XMRig, we observed spikes in CPU and memory utilization. The system's monitoring script, coupled with the detection mechanism, effectively tracked these anomalies in real-time and also killed the process associated with Xmrig.

The four implemented scenarios clearly demonstrated the system's capability to:

1. **Monitor normal system activity** without any cryptojacking interference.
2. **Detect significant resource consumption** during a cryptojacking attack.
3. **Automatically respond** by terminating the malicious process, restoring system performance, and sending alerts to the service center.
4. **Log cryptojacking incidents onto a blockchain** for secure and tamper-proof record-keeping.

Our experiment gives the importance of system resource monitoring for the detection and mitigation of cryptojacking activity, ensuring minimum disruption to performance. The integration of detection algorithms and automated responses enhanced the security, making them resilient to crypto-jacking.

In conclusion, our crypto-jacking detection mechanism is effective in real-time detection and mitigation, offering a robust solution to protect systems from increasing threats.

Future research directions could include enhancing the detection algorithm, in an actual automotive environment, the monitoring and detection agents can be embedded within the ECU's security modules or connected to telematics systems. Data can be securely transmitted via CAN gateways to diagnostic servers for detection. Upon detection, real-time alerts and mitigation actions (like ECU reboot or disconnection from the network) can be triggered, while blockchain logging ensures forensic traceability. This model aligns with ISO/SAE 21434 cybersecurity standards for automotive systems.

References

[1] Faraz Naseem, Ahmet Aris, Leonardo Babun, Ege Tekiner, A. Selcuk Uluagac, "MINOS: A Lightweight Real-Time Cryptojacking Detection System," 2024. Available: https://www.ndss-symposium.org/ndss-paper/minos-a-lightweight-real-time-cryptojacking-detection-system/

[2] Juan Tapiador, Sergio Pastrana, Guillermo Suarez-Tangil, "SoK: Cryptojacking Malware," 2021. Available: https://ieeexplore.ieee.org/document/9581251

[3] Krzysztof Chmiel, "Review of Selected Modern Cryptojacking Detection Techniques," 2024. Available: https://ii.uni.wroc.pl/media/uploads/2024/12/01/uwr-28-inz-69457-222085.pdf

[4] Ege Tekiner, Abars Aras, A. Selcuk Uluagac, "A Lightweight IoT Cryptojacking Detection Mechanism in Heterogeneous Smart Home Networks," 2023. Available: https://csl.fiu.edu/wp-content/uploads/2023/05/IoT_cryptojacking.pdf

[5] Elie Bursztein, Iban Petro, Luca Invernizzi, "CoinPolice: Detecting Cryptojacking Attacks with Neural Networks," 2020. Available: https://arxiv.org/abs/2006.10861

[6] Azad Waqar, Anwar, "Do Charging Stations Benefit from Cryptojacking? A Novel Framework for Its Financial Impact Analysis on Electric Vehicles," 2023. Available: https://ideas.repec.org/a/gam/jeners/v15y2022i16p5773-d883743.html

[7] Asma Alfradrus, Danda B. Rawat, "Evaluation of CAN Bus Security Vulnerabilities and Potential Solutions," 2023. Available: https://ieeexplore.ieee.org/document/10145267

[8] InfoBeyond Technology, "VehChain: A Blockchain-Reminiscent Cryptographic Solution for CAN Bus Security," 2023. Available: https://infobeyondtech.com/vehchain

[9] Ayush Kumar Bar, Alankshya Rout, Ankush Kumar Bar, "Cryptojacking Detection Using Genetic Search Algorithm," 2023. Available: https://www.researchgate.net/publication/370881984_Cryptojacking_Detection_Using_Genetic_Search_Algorithm

[10] Mansi Girdhar, "Machine Learning-Based ECU Detection for Automotive Security," 2022. Available: https://www.researchgate.net/publication/362851096_Machine_Learning_based_ECU_Detection_for_Automotive_Security

[11] A. Tudor, Camil Jichici, Adrian M, Alfred A, "Blockchain Integration for In-Vehicle CAN Bus Intrusion Detection Systems with ISO/SAE 21434 Compliant Reporting," 2023. Available: https://www.researchgate.net/publication/379667449_Blockchain_integration_for_in-vehicle_CAN_bus_intrusion_detection_systems_with_ISOSAE_21434_compliant_reporting

[12] G. Thirunavukkarasu, "From Car-Jacking to Crypto-Jacking: Is Your Autonomous Vehicle a Target?" LinkedIn, 2023. Available: https://www.linkedin.com/pulse/from-car-jacking-crypto-jacking-your-autonomous-god-thirunavukkarasu

[13] K. El Fellah, I. El Azami, A. El Makrani, H. Bouijij, O. El Azzouzy, "Revolutionizing Automotive Security: Connected Vehicle Security Blockchain Solutions for Enhancing Physical Flow in the Automotive Supply Chain," 2024. Available: https://www.researchgate.net/publication/386878115

[14] R. Kumar and A. Kumar, "A Cross-Stack Approach Towards Defending Against Cryptojacking," 2025. Available: https://www.researchgate.net/publication/343731164

[15] N. Alkhatib, M. Mushtaq, H. Ghauch, J.-L. Danger, "CAN-BERT Do It? Controller Area Network Intrusion Detection System Based on BERT Language Model," 2022. Available: https://ieeexplore.ieee.org/document/10069190/

[16] Fábio Gomes, Miguel Correia, "Cryptojacking Detection with CPU Usage Metrics," 2020. Available: https://ieeexplore.ieee.org/document/9306696

[17] Maurantonio Caprolu, Simone Raponi, Gabriele Oligeri, Roberto Di Pietro, "Cryptomining Makes Noise: Detecting Cryptojacking via Machine Learning," 2021. Available: https://www.researchgate.net/publication/349653111_Cryptomining_makes_noise_Detecting_cryptojacking_via_Machine_Learning

[18] Alan Romano, Yunhui Zheng, Weihang Wang, "MinerRay: Semantics-Aware Analysis for Ever-Evolving Cryptojacking Detection," 2021. Available: https://ieeexplore.ieee.org/document/9286112

[19] Dmitry Tanana, "Behavior-Based Detection of Cryptojacking Malware," 2023. Available: https://ieeexplore.ieee.org/document/9117732

[20] Guangquan Xu, Wenyu Dong, Jun Xing, Wenqing Lei, Jian Liu, Lixiao Gong, Meiqi Feng, Xi Zheng, Shaoying Liu, "Delay-CJ: A Novel Cryptojacking Covert Attack Method Based on Delayed Strategy and Its Detection," 2023. Available: https://www.researchgate.net/publication/360593101_Delay-CJ_A_novel_cryptojacking_covert_attack_method_based_on_delayed_strategy_and_its_detection

[21] Fredy Andrés Aponte-Novoa, Daniel Povedano Álvarez, Ricardo Villanueva-Polanco, Ana Lucila Sandoval Orozco, Luis Javier García Villalba, "On Detecting Cryptojacking on Websites: Revisiting the Use of Classifiers," 2022. Available: https://www.researchgate.net/publication/365834330_On_Detecting_Cryptojacking_on_Websites_Revisiting_the_Use_of_Classifiers

[22] Qianjin Ying, Yulei Yu, Donghai Tian, Xiaoqi Jia, Rui Ma, Changzhen Hu, "CJSpector: A Novel Cryptojacking Detection Method Using Hardware Trace and Deep Learning," 2022. Available: https://www.researchgate.net/publication/363495665_CJSpector_A_Novel_Cryptojacking_Detection_Method_Using_Hardware_Trace_and_Deep_Learning

[23] Ahmad Ali, Abid Khan, Mansoor Ahmed, Gwanggil Jeon, "BCALS: Blockchain-based secure log management system for cloud computing," 2021.Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4272