ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.2654

Fortifying IoT Ecosystems: A Hybrid Security Framework with Blockchain & Artificial Intelligence

Junior Gonzalez[1], Winaldo Walker[2], Cedric Green[3]
Faculty Mentor: Aparicio Carranza, PhD[4]
Faculty Co-Mentors: Harrison Carranza, MS[5], Miguel Bustamante, PhD[6]
[1,2,3,4,6]*Vaughn College of Aeronautics and Technology, USA, junior.gonzalez@vaughn.edu, winaldo.walker@vaughn.edu, cedric.green@vaughn.edu, Aparicio.Carranza@vaughn.edu, Miguel.Bustamante@vaughn.edu*
[5]*Bronx Community College of CUNY, USA, Harrison.Carranza@bcc.cuny.edu*

***Abstract*– The rapid expansion of the Internet of Things (IoT) requires security solutions that address vulnerabilities while operating within resource-constrained environments. Our solution introduces a hybrid framework that combines Blockchain for decentralized, tamper-resistant communication with Artificial Intelligence (AI) for real-time anomaly detection. A local IoT testbed has been implemented using an RFID sensor, blockchain nodes, and edge AI modules to continuously monitor system behavior. This system has been evaluated for latency, throughput, power consumption, and detection accuracy. Results demonstrate that combining blockchain's immutability with AI's adaptability enhances IoT security and reliability in practical deployments.**

***Keywords*-- Anomaly Intrusion Detection System, Artificial Intelligence, Blockchain, Edge Detection, Internet of Things.**

I. Introduction

In this paper we introduce a hybrid framework that combines Blockchain for decentralized, tamper-resistant communication with Artificial Intelligence (AI) for real-time anomaly detection. In the following subsections we present background information for our subsequent implementation.

A. Internet of Things

The Internet of Things (IoT) is a rapidly expanding network of interconnected, resource-constrained devices capable of sensing, processing, and exchanging data across distributed infrastructures. It enables automation, real-time monitoring, and intelligent decision-making in domains such as healthcare, transportation, smart homes, and industrial systems [1]. However, this widespread connectivity amplifies critical IoT security concerns as limited device memory, processing power, and energy make traditional defense mechanisms difficult to implement, leaving systems vulnerable to spoofing, eavesdropping, and unauthorized access [2]. To address these challenges, this project implements a secure IoT access-control subsystem using an STM32 microcontroller with an RC522 RFID module, Wi-Fi communication, and blockchain-based logging to ensure data integrity and prevent tampering. Additionally, AI-driven anomaly detection deployed through edge computing enables real-time identification of unusual or malicious device behavior, reducing reliance on cloud resources and strengthening resilience against emerging threats [3]. Through this combination of blockchain, edge intelligence, and lightweight hardware, the system enhances trust, transparency, and overall security across interconnected IoT nodes.

B. AI Hybrid Intrusion Detection

Within the proposed framework, the Hybrid Intrusion Detection System (HIDS) serves as the core intelligent defense component responsible for real-time threat identification and classification. This system integrates Suricata, a signature-based network intrusion detection engine [4], with an AI-driven anomaly detection layer implemented in Python. Snort provides robust packet inspection and rule-based detection for known attack patterns [5], while the AI module leverages machine learning algorithms—such as Logistic Regression or Convolutional Neural Networks (CNNs) - to detect deviations from normal traffic behavior that may indicate emerging or zero-day attacks [6]. The hybrid design capitalizes on the strengths of both approaches: Snort ensures precision in detecting signature-matched threats [4], and the AI layer enhances adaptability and reduces false positives by learning from evolving network behaviors [7]. Deployed on a Raspberry Pi, this configuration demonstrates how lightweight edge devices can perform advanced intrusion detection in IoT environments, bridging efficiency and intelligence. When integrated into the broader blockchain-secured framework, the HIDS not only enhances detection accuracy but also contributes to a distributed trust model, ensuring that validated alerts are immutably recorded and shared across interconnected IoT nodes [6].

C. Ethereum Based Blockchain Network with Multiple Nodes

Our focus has been on designing and deploying a private Ethereum blockchain network that securely records and stores sensor activity logs. The network consists of multiple Ethereum nodes distributed across team members' laptops, each operating as a peer within the private chain. Every sensor detection is captured as a transaction and written to the blockchain through a smart contract, ensuring that all nodes maintain a consistent and tamper-proof ledger. By utilizing Ethereum's consensus mechanism and cryptographic validation, the system provides data integrity, transparency, and decentralized control without reliance on a central server. This configuration allows each participant to access, verify,

and audit sensor event data in real time while maintaining a secure and scalable environment for future expansion.

In the previous section and subsections an Introduction was presented, in Section II, Theoretical Framework is discussed in detailed, followed by Section III, the methodology employed is detailed, Section IV, some possible Future Works are listed and finally our Conclusion is stated in Section V.

II. THEORETICAL FRAMEWORK

A. Internet of Things (IoT) Node

The IoT subsystem in this project was built around an STM32 microcontroller, functioning as a secure node for access control, environmental monitoring, and sensor data acquisition. It integrates RFID authentication, IR motion sensing, and temperature monitoring to provide a real-time overview of physical conditions. The node establishes WiFi connectivity to transmit sensor events and authentication logs to a blockchain network, enabling decentralized, tamper-resistant record-keeping. This architecture supports the foundational principles of IoT, including distributed sensing, autonomous control, and intelligent decision-making [1].

The IoT node is capable of:

- **RFID-Based Authentication:** The system reads RFID tags to identify authorized users and triggers access control logic accordingly.
- **Environmental Sensing:** IR motion detection and temperature sensors provide context-aware monitoring for physical spaces.
- **Secure Communication:** Wi-Fi connectivity allows nodes to communicate with the blockchain network while maintaining data integrity and confidentiality.

The IoT subsystem addresses challenges inherent to resource-constrained embedded devices, such as limited memory, processing power, and energy availability [2]. Traditional security mechanisms are often too heavy for microcontrollers like the STM32; therefore, lightweight protocols, efficient sensor polling, and selective data transmission are employed to preserve performance while ensuring security. Additionally, blockchain integration allows secure logging of device events, enabling the system to maintain an immutable record of user authentication and sensor readings.

Finally, combining IoT sensing with edge-level intelligence provides the foundation for a hybrid security model. While the STM32 node handles real-time authentication and environmental monitoring, data forwarded to the blockchain network ensures decentralized trust, and AI-driven anomaly detection can identify unusual behavior or potential threats [3]. This layered approach enhances the overall resilience, reliability, and trustworthiness of the IoT system, ensuring it operates effectively within modern cyber-physical infrastructures.

B. Ethereum Blockchain Ledger for Secure Communication

The blockchain subsystem is based on a private Ethereum network designed to provide decentralized, tamper-resistant storage for IoT sensor events and security logs. Ethereum is a distributed ledger platform that combines cryptographic hashing, peer-to-peer networking, and consensus mechanisms to ensure data integrity and trust without reliance on centralized authority. In this architecture, each participating device operates as a blockchain node, maintaining a synchronized copy of the ledger and collectively validating all recorded transactions.

The private Ethereum network consists of multiple nodes distributed across team members' laptops and a Linux-based IoT device, all connected through a secure VPN mesh network. This configuration enables direct peer-to-peer communication while isolating the blockchain from public networks. Sensor detections generated by the IoT subsystem are encapsulated as transactions and submitted to the blockchain through a smart contract interface. Once confirmed, these transactions are permanently written into blocks and replicated across all nodes, ensuring consistency and immutability of the recorded data. Key theoretical principles underpinning the blockchain subsystem include:

- **Decentralization:** No single node controls the ledger; instead, all nodes participate equally in transaction validation and data replication, eliminating single points of failure.
- I**mmutability:** Cryptographic hashing and block chaining prevent retroactive modification of stored sensor logs, preserving forensic integrity.
- **Consensus:** Ethereum's consensus mechanism ensures that all participating nodes agree on the order and validity of transactions before they are committed to the ledger.
- **Transparency and Auditability:** Every node maintains access to the full transaction history, enabling independent verification and real-time auditing of sensor events.

Unlike public Ethereum networks, the use of a private blockchain allows full control over node membership, consensus participation, and network parameters. This approach improves performance, reduces transaction latency, and avoids transaction fees while maintaining the core security properties of blockchain technology. Authorized nodes may also be designated as block producers, ensuring continuous block generation and reliable logging of time-sensitive sensor data.

From a security perspective, the Ethereum blockchain functions as a trusted logging layer within the overall system architecture. Events recorded on the ledger cannot be altered or deleted without detection, making the blockchain well-suited for storing authentication events, environmental sensor readings, and intrusion alerts generated by higher-level security components. This immutable record strengthens accountability and supports post-incident analysis.

Finally, integrating the Ethereum blockchain with IoT and intrusion detection components enables a layered cyber-

physical security model. While IoT nodes handle real-time sensing and access control, and Suricata provides network-level threat detection, the blockchain ensures that all critical events are securely preserved in a decentralized ledger. This combination enhances system resilience, supports distributed trust, and establishes a scalable foundation for future extensions such as smart contract–based access policies or automated incident response mechanisms.

C. *Suricata Intrusion Detection Engine*

Suricata is an open-source, high-performance Intrusion Detection and Prevention System (IDPS) designed for deep packet inspection, traffic classification, and rule-based threat detection. Its multi-threaded architecture enables efficient performance even on resource-constrained platforms such as the Raspberry Pi, making it suitable for IoT security applications. Suricata's signature-based rule sets allow accurate detection of known attack patterns, including ICMP flooding, DNS tunneling, and HTTP-based exploits.

Key capabilities include:

- **Protocol Parsing:** Suricata natively decodes protocols such as DNS, HTTP, TLS, and others, enabling granular inspection and context-aware detection.
- **EVE JSON Logging:** Events are exported in structured JSON format to */var/log/suricata/eve.json*, providing machine-readable logs for downstream analysis and correlation.
- **Rule Extensibility:** Custom rules can be authored to detect IoT-specific attack behaviors or device anomalies not covered by default rule sets.
- **AI Integration:** The structured EVE JSON logs serve as a high-quality data source for pre-processing and training machine learning models, supporting hybrid IDS architectures.

D. *Suricata AI-Based Anomaly Detection*

Signature-based intrusion detection systems (IDS), such as Suricata, excel at identifying known attacks but inherently fail to detect zero-day exploits or novel threat patterns because their detection logic depends on predefined signatures. To overcome this limitation, the system integrates an AI-driven anomaly-detection layer implemented in Python using the *scikit-learn* machine learning library. The model employs **Isolation Forest**, an unsupervised learning algorithm that isolates anomalous samples by recursively partitioning the feature space.

Traffic features extracted from Suricata logs include source IP, destination IP, source port, destination port, and protocol. Categorical variables (IP addresses, protocols) are encoded numerically using *LabelEncoder*, while continuous numerical fields (port values) are retained. The model is periodically updated with new traffic data, enabling it to learn evolving patterns within IoT environments and maintain relevance over time.

This hybrid IDS architecture combines the strengths of both detection paradigms. Signature-based detection provides high precision for known threats, while the AI model contributes adaptability by learning baseline patterns and identifying deviations indicative of emerging or previously unseen attacks. Furthermore, the anomaly-detection layer helps reduce false positives by contextualizing alerts and filtering out benign irregularities that would otherwise be flagged.

III. METHODOLOGY

A. *IoT Node Development*

The methodologies used ensure the IoT can be reproduced and the commands used were working through a Linux OS environment. The required components for creating the IoT nodes were: STM32-B-L475E-IOTA1 Development Board, STM32CubeIDE, Git, Python3, Minicom, RC522 RFID Module & RFID Cards and Jumper Wires.

B. *The Workspace Environment Set Up*

i. Connection Establishment to Raspberry PI: The STM32 board works in hand with the Raspberry Pi as a singular IoT node. Due to resource constraints of the STM32, it solely cannot act as a node in the blockchain ledger. Therefore, it will send the RFID scans to the Raspberry Pi over WiFi and then the ultimately sent over to the blockchain ledger. In Fig. 1 the Python TCP communication script is shown.

```python
import socket, time

HOST = "0.0.0.0"
PORT = 8002

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((HOST, PORT))
server.listen(1)

print(f"Listening on {HOST}:{PORT}...")

conn, addr = server.accept()
print("STM32 connected from:", addr)

while True:
    # Send periodic message
    message = "ping from python"
    conn.sendall(message.encode())
    print("Sent:", message)

    # Receive (non-blocking-ish)
    conn.settimeout(0.1)
    try:
        data = conn.recv(1024)
        if data:
            print("STM32:", data.decode('utf-8', errors='ignore'))
    except:
        pass

    time.sleep(2)

```

Fig. 1 Python TCP Server Script

ii. Setting up STM32 Workspace: On the Linux PC terminal, we create a directory in which all the necessary files

for the STM32 board will be saved. Detailed setup steps are shown below in Fig. 2, Fig. 3, Fig. 4. and Fig. 5.

iii. Open up STM32CubeIDE

iv. Define I/O for RC522 RFID Module

```
#ifndef RFID_IO_H
#define RFID_IO_H

#include "stm32l4xx_hal.h"

#define MFRC522_CS_Port       GPIOB
#define MFRC522_CS_Pin        GPIO_PIN_9

#define MFRC522_RST_Port      GPIOB
#define MFRC522_RST_Pin       GPIO_PIN_2


void RFID_IO_Init(void);
void RFID_SPI_Init(void);

extern SPI_HandleTypeDef hspi1;

#endif

```

Fig. 2 rc522_io.h code

v. Defining includes and Variables in main.c

```
#include "rc522_io.h"
#include "RC522.h"
#include "string.h"
#include "stdint.h"
#include <stdio.h>
```

Fig. 3 Include Files

vi. Initializing Modules

```
/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* Configure the system clock */
SystemClock_Config();
RFID_IO_Init();
RFID_SPI_Init();

MFRC522_Init();
/* Configure LED2 */
BSP_LED_Init(LED2);
```

Fig. 4 Modules Initialization

vii. Create Helper Functions

```
/* Helper function to convert UID to a string */
static void FormatUID(char *buffer, uint8_t *uid, uint8_t uidSize)
{
    int index = 0;
    for (int i = 0; i < uidSize; i++)
    {
        index += sprintf(&buffer[index], "%02X", uid[i]);
        if(i < uidSize - 1)
            buffer[index++] = ':';  // add : between bytes
    }
    buffer[index] = '\0';
}
```

Fig. 5 FormatUID Function

viii. Modify the Sending Data Function

B. *Flashing the Board and Sending Data – The process is shown in Fig. 6*

i. Activate Python Script on Raspberry PI

ii. Open a UART Terminal for Monitoring

iii. Flash the STM32 Board

iv. Record the Outcome

```
cpe326@raspberrypi:~ $ sudo nano tcp_server.pi
cpe326@raspberrypi:~ $ sudo nano tcp_server.py
cpe326@raspberrypi:~ $ python3 tcp_server.py
Listening on 0.0.0.0:8002...
STM32 connected from: ('172.20.10.8', 5469)
STM32: RFID_SCAN:63:90:72:11:90

STM32: RFID_SCAN:73:6E:76:0D:66
```

Fig. 6 Raspberry Pi Receiving Data

C. *Ethereum Blockchain Ledger Development*

The following methodology ensures that the Ethereum-based blockchain ledger can be reproduced across all participating nodes. Any commands explicitly shown were

executed and verified within native Windows, Mac Operating Systems (MacOS) or Linux operating system environments.

i. Required Tools for Creating the Blockchain Ledger: Get (*Go Ethereum Client*), Tailscale VPN Mesh Network, Linux, Windows, or MacOS, Ethereum Genesis Configuration File (*genesis.json*), Terminal or Command-Line Interface (*PowerShell, Bash, or Zsh*), IoT Device with Linux-Based OS (*Raspberry Pi or equivalent*)

ii. Setting Up the Blockchain Network Environment

Establish Secure Peer-to-Peer Networking

Install the Ethereum Client (Geth)

iii. Initializing Ethereum Nodes

Create the Node Data Directory

Create an Ethereum Account

Initialize the Node with the Genesis File

iv. Launch the Blockchain Node

v. IoT Device Blockchain Node Setup

Install Geth on the IoT Device

Join the VPN Network

Initialize and Launch the IoT Node

vi. Verifying Blockchain Operation

vii. Record the Outcomes

D. AI-Based Intrusion Detection System

This section presents the complete methodology used to design, install, configure, and implement the AI-driven Detection System (AI IDS) for the IoT intrusion-detection project. The methodology combines Suricata's signature-based detection with a Python-based anomaly-detection engine developed using *scikit-learn*. The process consists of two major components:

(a) Installation and configuration of Suricata to generate structured network events.

(b) Development and deployment of an AI anomaly-detection model using Isolation Forest.

The following subsections document each step in detail, ensuring the process can be reproduced accurately.

i. Suricata Installation and Configuration - Before installing Suricata, the Raspberry Pi was updated to ensure the latest package versions were available. This prevents dependency conflicts during installation.
Commands executed: ***$ sudo apt update*** *&&* ***sudo apt upgrade -y***

ii. Installing Required Dependencies - Suricata requires several libraries for packet capture, regular expression parsing, cryptography, and rule-processing. All dependencies were installed using:

***$ sudo apt install -y ***
***libpcre2-dev libyaml-dev zlib1g-dev ***
***libcap-ng-dev libmagic-dev libnet1-dev ***
***libpcap-dev libjansson-dev libnss3-dev ***
***libgeoip-dev liblua5.3-dev pkg-config ***
$ build-essential cmake automake autoconf libtool git

These packages enable Suricata to parse protocols, manage rules, handle JSON events, and perform deep packet inspection.

iii. Adding the Suricata PPA - To obtain the latest stable release, the official OISF PPA repository was added.

$ sudo add-apt-repository ppa:oisf/suricata-stable
$ sudo apt update

This ensures the system receives up-to-date rules and security patches.

iv. Suricata Installation - Suricata was installed:

$ sudo apt install suricata -y
$ suricata --version

v. Configuration of EVE.JSON Logging - Suricata logs events in many formats, but the AI IDS requires structured JSON logs. The configuration file was edited as seen in Fig. 7.

$ sudo nano /etc/suricata/suricata.yaml

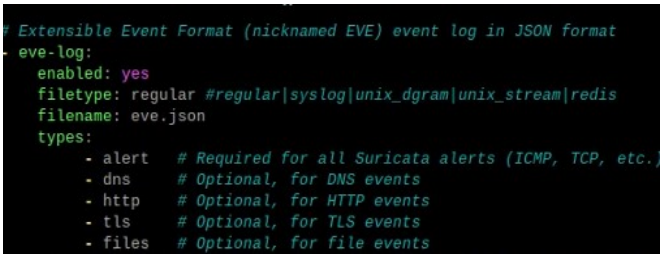

Fig. 7 Suricata Configuration

vi. Starting and Testing Suricata - Suricata was first tested manually on the Raspberry Pi's wireless interface.

$ sudo suricata -c /etc/suricata/suricata.yaml -i wlan0

We verified that rule parsing, packet capture, and logging functioned correctly.

vii. Updating Rule Sets - Community rules were fetched using:

$ sudo suricata-update
$ sudo systemctl restart suricata

viii. Running Suricata as a Service - Suricata was configured to run persistently on boot:

$ sudo systemctl enable suricata
$ sudo systemctl start suricata
$ sudo systemctl status suricata

At this stage, Suricata continuously generates real-time alerts to ***eve.json***, enabling direct AI ingestion

E. AI-Based IDS Development Using Isolation Forest

i. Environment Setup - The AI IDS implementation was developed inside a dedicated Python virtual environment to isolate dependencies and prevent system-wide conflicts

a. Install Python Dependencies

$ sudo apt update
$ sudo apt install python3-venv python3-pip -y

b. Create Project Directory

A dedicated folder was created for the AI IDS:

$ mkdir ~/ai_ids

$ cd ~/ai_ids

c. Install Required Python Libraries

$ python3 -m venv venv

$ source venv/bin/activate

The command prompt switched to indicate the environment was active.

d. Install Required Python Libraries

Inside the virtual environment, machine learning and data-processing libraries were installed:

$ pip install --upgrade pip

$ pip install pandas scikit-learn numpy

These libraries support JSON log parsing, feature engineering, model training, and real-time anomaly detection.

ii. Integrating Suricata with the AI Engine - Configuring EVE.JSON for Machine Learning

To ensure compatibility with the anomaly-detection pipeline, Suricata JSON logs were configured to include:

- src_ip
- dest_ip
- src_port
- dest_port
- proto
- timestamp

iii. Development of the AI Anomaly-Detection Model: A Python script named ai_ids.py was created in the project directory: ***nano ai_ids.py,*** the script implements the full anomaly-detection pipeline.

iv. Detailed Pipeline Architecture

a. Real-Time Log Monitoring

The script continuously monitors Suricata's *eve.json* using:

i. *load_new_lines(file_path, last_pos)*

This retrieves only newly appended JSON objects, enabling low-latency detection.

b. Parsing and Preprocessing Suricata Events

Each log entry is:

1. Loaded from JSON

2. Filtered for events containing "*alert*"

3. Parsed into a Python dictionary with:

- src_ip
- dest_ip
- src_port
- dest_port
- proto
- timestamp

This produces two parallel datasets:

- **df_orig,** human-readable data
- **df_enc,** numerically encoded data for ML

Categorical fields (***src_ip, dest_ip***, ***proto***) are encoded using: ***LabelEncoder()***

Numeric features (ports) are left unchanged.

c. Feature Selection

The model uses:

['src_ip', 'dest_ip', 'src_port', 'dest_port', 'proto']

as the system's primary behavioural indicators.

d. Isolation Forest Initialization

An Isolation Forest model is created with:

***IsolationForest* (*contamination=0.01, n_estimators=100, random_state=42*)**

This unsupervised model learns baseline network behavior and flags outliers as potential attacks.

Before receiving real traffic, the model is seeded with a simple placeholder dataset to avoid empty-fit errors:

dummy = pd.DataFrame({...})

model.fit(dummy)

e. Real-Time Anomaly Detection Loop

The main loop performs the following steps every 2 seconds *(POLL_INTERVAL = 2):*

1. Read new Suricata events

2. Preprocess and encode features

3. Run anomaly detection:

model.predict(df_enc[FEATURE_COLUMNS])

4. Combine predictions with original data

5. Filter anomalies (-1 predictions)

6. Print alerts to console

7. Append suspicious events to:

~/ai_ids/alerts.csv

f. Incremental Learning

After each cycle, the model retrains on new normal data: ***model.fit(df_enc[FEATURE_COLUMNS])***

This allows the IDS to adapt dynamically to evolving IoT traffic patterns.

v. Deployment of the AI IDS as seen in Fig. 8.

a. Make the Script Executable

$ chmod +x ai_ids.py

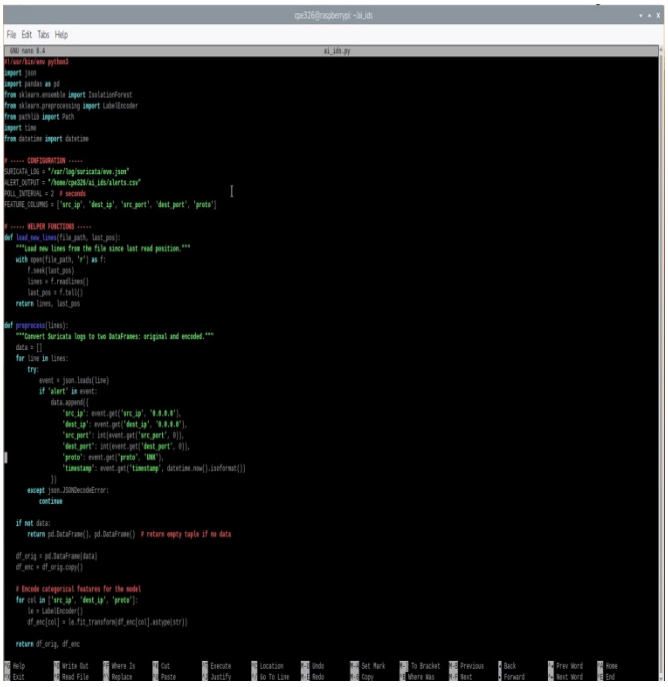

Fig. 8 AI IDS Python Script

b. Run the System

$ cd ~/ai_ids

$ source venv/bin/activate

$ python ai_ids.py

The terminal then displays:

AI IDS running... Monitoring Suricata logs in real-time. This action is shown in Fig. 9.

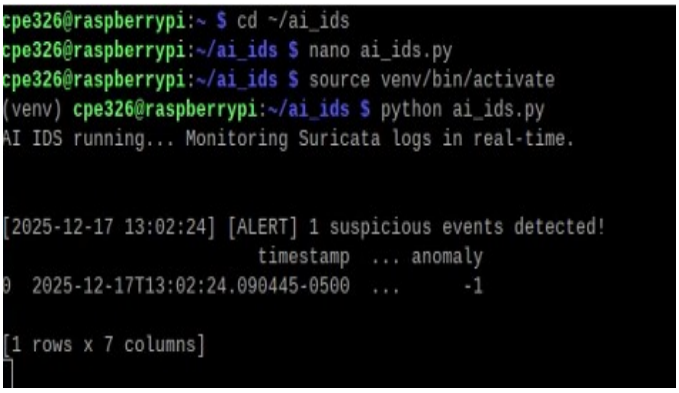

Fig. 9 Python Script Successfully Executed

c. Viewing Alerts

Suspicious events can be monitored via:

$ tail -f ~/ai_ids/alerts.csv

The result is shown in Fig. 10.

This file logs:

- Timestamp
- src_ip
- dest_ip
- src_port
- dest_port
- protocol
- anomaly score

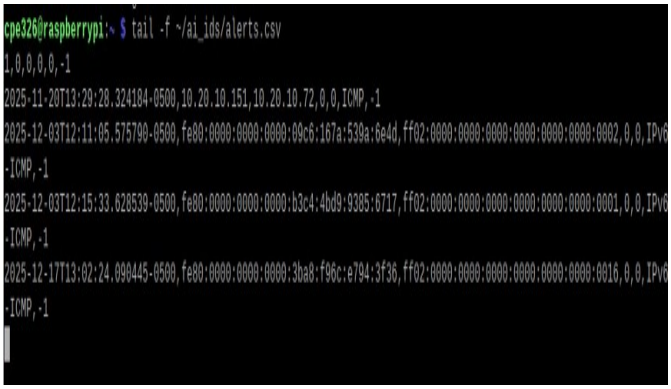

Fig. 10 AI-IDS CSV Logs

IV. FUTURE WORKS

Due to time constraints, course scope limitations, and hardware availability, the system developed in this project represents a foundational implementation rather than a fully expanded deployment. While the core components were successfully designed and integrated, further development and evaluation were beyond the available project timeline. As a result, several areas have been identified for future expansion and improvement:

One key area of future work involves scaling the IoT system to support a larger number of sensor nodes and distributed devices. Expanding the system would enable a more realistic evaluation of network performance, blockchain transaction throughput, and node synchronization under increased load. This would provide greater insight into how the architecture performs in real world IoT environments with multiple concurrent data sources.

Another important direction is performing deeper system evaluations. Future testing could include detailed latency measurements for blockchain transactions, power consumption analysis on embedded devices, and long term stability testing of the intrusion detection pipeline. Additional experimentation under simulated attack conditions would also help assess detection accuracy, false positive rates, and system responsiveness to both known and unknown threats.

Finally, optimizing edge deployment remains a valuable area for further development. Improvements could focus on reducing computational overhead on intermediary devices, optimizing data pre-processing, and exploring lightweight cryptographic operations suitable for resource constrained hardware. Enhancing edge level efficiency would allow more intelligent processing to occur closer to the IoT devices while preserving security, reliability, and scalability.

Addressing these future enhancements would strengthen the overall system and move it closer to a practical and deployable security framework for modern Internet of Things (IoT) infrastructures.

V. CONCLUSION

This project has demonstrated the successful design and implementation of a secure cyber physical system that integrates IoT sensing, blockchain based data integrity, and network intrusion detection. By combining an STM32 based IoT node, a private Ethereum blockchain ledger, and the Suricata intrusion detection engine with AI based anomaly detection model, the system achieves decentralized trust, secure event logging, and intelligent threat monitoring. Each subsystem was developed with practical constraints in mind, including limited embedded resources, network security requirements, and system scalability. The IoT node effectively performs RFID based authentication and environmental sensing while maintaining reliable communication with the blockchain network through an intermediary edge device. Sensor events are securely transmitted and recorded on the private Ethereum ledger, ensuring immutability, transparency, and verifiability of access and monitoring data. The use of a private blockchain allows the system to retain the core security benefits of distributed ledgers without the overhead and cost associated with public networks. In parallel, the integration of Suricata provides real time network traffic inspection and signature-based intrusion detection. The addition of an AI driven anomaly detection layer enhances the system's ability to identify previously unseen or evolving threats, addressing limitations inherent to traditional rule-based approaches.

Together, these components form a layered security architecture that improves system resilience and supports continuous monitoring. Overall, we highlighted the feasibility of combining IoT, blockchain, and intrusion detection technologies into a unified security framework. The resulting system provides a strong foundation for secure access control, trusted data storage, and intelligent threat detection within modern IoT environments.

REFERENCES

[1] K. Ashton, "That 'Internet of Things' Thing," RFID Journal, vol. 22, no. 7, pp. 97–114, 2009.

[2] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," Computer Networks, vol. 76, pp. 146–164, 2015.

[3] H. Hindy, E. Bayne, R. Atkinson, C. Tachtatzis, and X. Bellekens, "A taxonomy and survey of intrusion detection and prevention systems for the Internet of Things," IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2522–2557, 2020.

[4] Oisf, "OISF/Suricata: Suricata is a network intrusion detection system, intrusion prevention system and network security monitoring engine developed by the OISF and the SURICATA community.," GitHub, https://github.com/OISF/suricata?tab=readme-ov-file.

[5] Gomez, J., & Santin, M. (2009). *Extending Snort with an Anomaly Preprocessor*. Communications in Computer and Information Science, 48, 662–671. https://doi.org/10.1007/978-3-642-02481-8_75.

[6] Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). *A deep learning approach to network intrusion detection*. IEEE Transactions on Emerging Topics in Computational Intelligence, 2(1), 41–50. https://doi.org/10.1109/TETCI.2017.2772792.

[7] Dhanabal, L., & Shantharajah, S. P. (2015). *A study on NSL-KDD dataset for intrusion detection system based on classification algorithms*. International Journal of Advanced Research in Computer and Communication Engineering, 4(6), 446–452.

[8] STMicroelectronics,"STM32CUBEL4/projects/B-L475E-IOT01A https://github.com/STMicroelectronics/STM32CubeL4/tree/master/Projects.

[9] CircuitGatorHQ, "CircuitGatorHQ/RFID-RC522-with-STM32: How to use RC522 RFID with STM32," https://github.com/CircuitGatorHQ/RFID-RC522-with-STM32/tree/main.