Forensic Science International: Digital Investigation 49 (2024) 301754

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA

MARS: The first line of defense for IoT incident response

Karley M. Waguespack [a,b,1], Kaitlyn J. Smith [c,1], Olame A. Muliri [a,b,1], Ramyapandian Vijayakanthan [d,1], Aisha Ali-Gombe [a,b,*]

[a] Center for Computation and Technology, Louisiana State University, United States
[b] School of Electrical Engineering & Computer Science, Louisiana State University, United States
[c] Baton Rouge Magnet High School, United States
[d] Department of Computer and Information Science, Towson University, United States

A B S T R A C T

The proliferation of Internet of Things (IoT) devices across homes, businesses, and industrial landscapes has significantly increased our capability to gather data and automate tasks. Despite their ubiquity, these devices are notably resource-constrained and frequently lack robust security defenses, presenting a substantial risk of intrusion and cyber threats. To address these concerns, we propose a novel anomaly-based host intrusion detection system specifically designed for IoT devices, titled *MARS* (Memory Anomaly Recognition System). *MARS* is designed to function as a crucial component in the incident response framework, acting as an early detection system for potential security breaches within an organization's network or systems. The fundamental architecture of *MARS* leverages the device's memory as a key indicator for monitoring system-level events. To enhance its security and integrity, *MARS* is embedded within a Trusted Execution Environment—a secure, hardware-isolated region of a microcontroller protected from untrusted software. This design choice not only makes *MARS* tamper-proof but also ensures reliable monitoring of the device's memory. Deviations from established memory baselines, indicative of a security compromise, are detected through an anomaly detection algorithm hosted on a remote server. Our evaluation of the *MARS* prototype on STM32L562QEI6QU showed our proposed architecture can achieve decent scalability while maintaining trust, accuracy, and robustness of memory changes.

1. Introduction

IoT devices are small pieces of hardware commonly embedded within larger systems. These devices consist of input and output devices connected to a small computing device known as a microcontroller (MCU). As the sensors gather information about the surroundings, the MCU instructs the outputs to act on the environment accordingly. Since these devices are extremely small and resource constrained, they commonly communicate with control servers which perform additional processing and issue commands. Although IoT technology is widely recognized for its domestic applications, it also plays a crucial role in Industrial Control Systems (ICS), where it is used to monitor equipment and directly modify its functioning. Like traditional computers, these systems are susceptible to malware infections, which can lead to the destruction of equipment or even physical catastrophes.

ICS malware is often in the form of worms, and it is designed to spread strategically through industrial networks. These threats typically begin by exploiting vulnerable peripheral devices and gradually progress toward more critical components, such as devices that manage Programmable Logic Controllers (PLCs). The integration of Information Technology (IT) and Operational Technology (OT) in ICS introduces vulnerabilities that enable these worms to wreck havoc. A classic example is the Stuxnet worm, an infamous malware that severely impacted Iran's nuclear infrastructure. Intrusion detection systems (IDS) can serve as a first line of defense for such attacks. Despite the prevalence of IDS for conventional computing environments, little work has been done to create solutions for IoT devices. Given that attacks on ICS frequently exploit security flaws on peripheral devices, deploying intrusion detection systems on such devices is critical for establishing a first line of defense.

To advance this area of research, we proposed *MARS*: a host-based IDS that uses device memory to detect anomalous activity. *MARS* is implemented partially through the firmware on the IoT device. The component of the IDS residing on the MCU lives within a Trusted Execution Environment (TEE) to make the system tamper-proof in the event of a malware infection. *MARS* contains a memory acquisition component built into the TEE that routinely captures memory dumps of the Rich Execution Environment (REE). These dumps get sent over the

* Corresponding author. Center for Computation and Technology, Louisiana State University, United States.
E-mail addresses: kwagu14@lsu.edu (K.M. Waguespack), kjsmith415@gmail.com (K.J. Smith), omulir1@lsu.edu (O.A. Muliri), rvijay1@students.towson.edu (R. Vijayakanthan), aaligombe@lsu.edu (A. Ali-Gombe).
[1] Authors contributed equally.

https://doi.org/10.1016/j.fsidi.2024.301754

Available online 5 July 2024

network to a classification server where a machine learning algorithm is used to classify the dump as benign or anomalous. Once the classification is computed, it is sent back to the MCU so that the device can be reset and securely booted if needed. In addition to creating a system design around TEEs, we implemented a prototype on an STM32-based MCU commonly used for prototyping IoT solutions. The design, implementation, and evaluation of this system are discussed in the following sections.

In summary, the contributions of our research are as follows:

- **Trust and Integrity** - The use of a TEE provides hardware isolation which protects the IDS from potential tampering or deactivation by malware, ensuring the system's integrity.
- **Accuracy** - The anomaly-based approach leverages machine learning algorithms to dynamically identify deviations from normal operational patterns, facilitating the accurate and early detection of novel and sophisticated malware variants.
- **Robustness** - By monitoring memory footprint, the IDS is capable of detecting a wide range of anomalous behaviors, from subtle manipulations and changes by malware to overt attempts at disrupting device operations.

The remainder of this paper is organized as follows: Section 2 discusses relevant background information; Section 3 discusses the overall design which can be generalized to other MCUs; Section 4 discusses the implementation of the prototype; Section 5 includes an evaluation of the prototype; Section 6 discusses related work in IoT intrusion detection systems and how they compare to ours; and finally, Section 7 ends with a conclusion.

2. Background

Before discussing the design of *MARS*, we will introduce a background of intrusion detection systems. These include the types of intrusion detection systems and details on how TEEs work. We then briefly introduce our proposed approach and explain how it could swiftly halt attacks such as Stuxnet.

2.1. Network-based IDS (NIDS) vs. host-based IDS (HIDS)

NIDS monitor network activity to detect threats. These systems impose less load on devices since they only monitor traffic. However, their limitations include the inability to monitor encrypted network traffic and access host–level activities. To this end, they can be circumvented by malware that leverages encrypted communication or leaves minimal footprint on the network.

By contrast, HIDS monitors system-level activities, such as device logs and system calls. The use of detailed host information enhances detection accuracy which results in a system that cannot be circumvented as easily. However, such detailed monitoring can place significant load on the device. Further, without proper isolation, malware could disable the HIDS.

2.2. Signature-based vs. anomaly-based detection

Signature-based detection uses known patterns to identify malware, offering ease of implementation and low resource consumption. However, its effectiveness is limited to known malware variants, with new variants requiring manual pattern updates. This approach can be circumvented through slight code modifications.

Anomaly-based detection employs machine learning or statistical techniques to dynamically identify anomalous behavior based on baseline behaviors, offering resilience against unknown threats. However, it is complex and resource intensive.

2.3. Trusted Execution Environments

TEEs provide a hardware-isolated region for executing sensitive software. This region is separate from the main operating system, also known as the REE. This separation divides an MCU into a secure world and a non-secure world, allowing for the allocation of hardware resources to either world. TEEs ensure that the secure world remains protected despite compromises in the REE. Devices using this technology are loaded with two firmware binaries–one for the TEE and another for the REE. The TEE has control during device initialization before control is passed to the REE. After this shift in control, the TEE can be re-entered via secure-callable functions. Communication between the secure and non-secure worlds occurs via a secure communication channel, and both the hardware and device drivers enforce proper isolation when this channel is in use. Fig. 1 illustrates the TEE's role in dividing the MCU into secure and non-secure worlds. Such hardware features are provided by chip manufacturers such as ARM, and they are paramount for managing sensitive data and processes.

2.4. Proposed approach

We will now introduce *MARS*, a host- and anomaly-based IDS that utilizes device memory for early intrusion detection. This system serves as the initial defense in an incident response life cycle. Our approach is structured around a client-server model and ensures isolation from potential malware threats. To achieve isolation, the memory acquisition will reside within the TEE on the IoT device. On the other hand, the anomaly detection will reside on an external server.

2.4.1. Threat

Imagine that a Stuxnet-like malware seeks to compromise a nuclear facility's network, aiming to disrupt centrifuges. Malware authors discover that this plant uses special IoT devices connected to PLCs to manage and monitor the centrifuges—a scenario that is becoming increasingly common in ICS. This malware targets the vulnerable IoT device so that it can directly send commands to the connected PLC. Next, its goal is to damage the centrifuges by covertly increasing the rotational speed. Additionally, since the IoT device is responsible for sending readings to the control system for monitoring, the malware spoofs the readings to further camouflage its destructive activities. This stealthy behavior makes it virtually impossible for operators to notice any unusual behavior until it is too late.

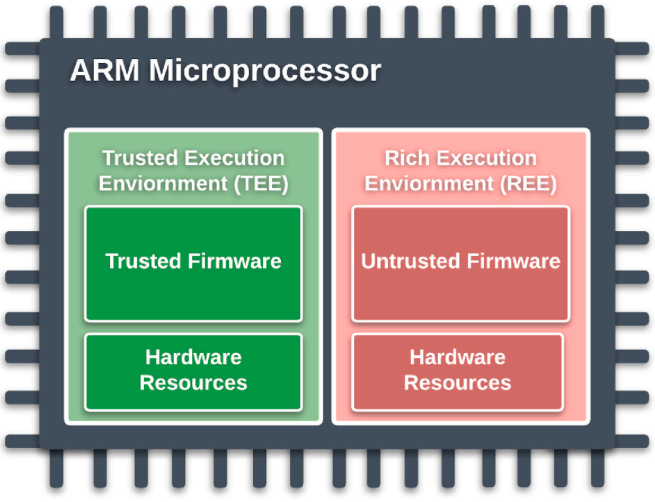

Fig. 1. Hardware isolation achieved through the implementation of a trusted execution environment.

2.4.2. IDS integration

Incorporating *MARS* within the IoT device could significantly alter the outcome of such an attack. If the infected IoT device could detect anomalous behavior, the malware could be stopped in its tracks before causing significant damage.

3. Design

Having established the motivation of *MARS*, we will now discuss its architecture. Our proposed IoT HIDS, depicted in Fig. 2, uses device memory for anomaly detection. The rationale behind using memory is that IoT device activities are routine. Thus, unlike traditional computing systems, IoT memory remains mostly static over time. For instance, a smart thermostat is specifically designed to monitor and regulate temperature. It would be unexpected and anomalous for such a device to engage in an unrelated task, such as recording ambient audio. Such deviations in functionality are indicative of anomalies and can be detected through analysis of runtime activities.

With this understanding, *MARS* is designed to capture the IoT device memory at the client-side. It then transfers the device memory to a classification server that first performs preprocessing and then classification using a Convolutional Neural Network (CNN). The CNN model is trained to detect whether the memory patterns represent normal operation or anomalies.

In designing of the IDS components, we have identified some critical requirements essential for overcoming the challenges associated with developing HIDS:

1. The IDS must be segmented into a **client-server** architecture given the resource limitations.
2. The client portion of the IDS must be completely **isolated** from untrusted software to maintain integrity in the event of a compromise.
3. The anomaly detection algorithm should be placed on a remote server to **optimize** performance.

While the exact implementation for the proposed IDS will differ from chip to chip depending on the features, tools, and libraries offered by the MCU manufacturer, the overall proposed design discussed can be used as a blueprint when implementing the system on a new MCU.

3.1. Client-side components

Since *MARS* uses the device memory for classification, we must have a tool for extracting MCU memory. The memory acquisition component is coded directly into the TEE firmware. To create a TEE, part of the flash on the MCU is allocated to the secure region (corresponding to the TEE) while the remainder is allocated to the non-secure world (corresponding to the REE). The flash bank in the non-secure world houses the main device firmware, IoT process, as well as any other untrusted software that may be on the device. In addition to housing untrusted software, it may have various GPIO pins and serial communication interfaces allocated to it so that the main IoT process can use these during its operation.

As shown in Fig. 2, the secure flash partition houses all client-side functions of the IDS, including the memory acquisition and network code that sends memory blocks to the server. The trusted environment also has a watchdog timer, which forces a device reset once it fully counts down. These timers are commonly used to monitor software execution and detect system faults. Precisely, an MCU can be made to reset if an important software component has failed to execute in time by embedding the code that resets the watchdog timer inside of that software component. We use a watchdog timer to enforce the timely capture of memory samples within the system.

3.1.1. Memory acquisition module

This module is tasked with collecting features for the CNN classification. Since ensuring the integrity of the memory image is vital for the effectiveness of the model, this module operates entirely within the TEE.

The memory acquisition component occurs in two steps. First, it transfers memory blocks from the REE to the TEE. Next, it transfers the memory from the TEE to the classification server. Recall that IoT memory tends to remain mostly static over time. Because of this, we minimize unnecessary memory transfer by analyzing each memory block for alterations before sending memory. This is accomplished by hashing each block and comparing it to its previous hash. These hashes are stored in an array within the TEE, indexed by block number. The TEE first computes and stores these hashes during device initialization. The array is then updated continuously during device operation if any block's hash changes.

For any blocks that have been modified, the module records the

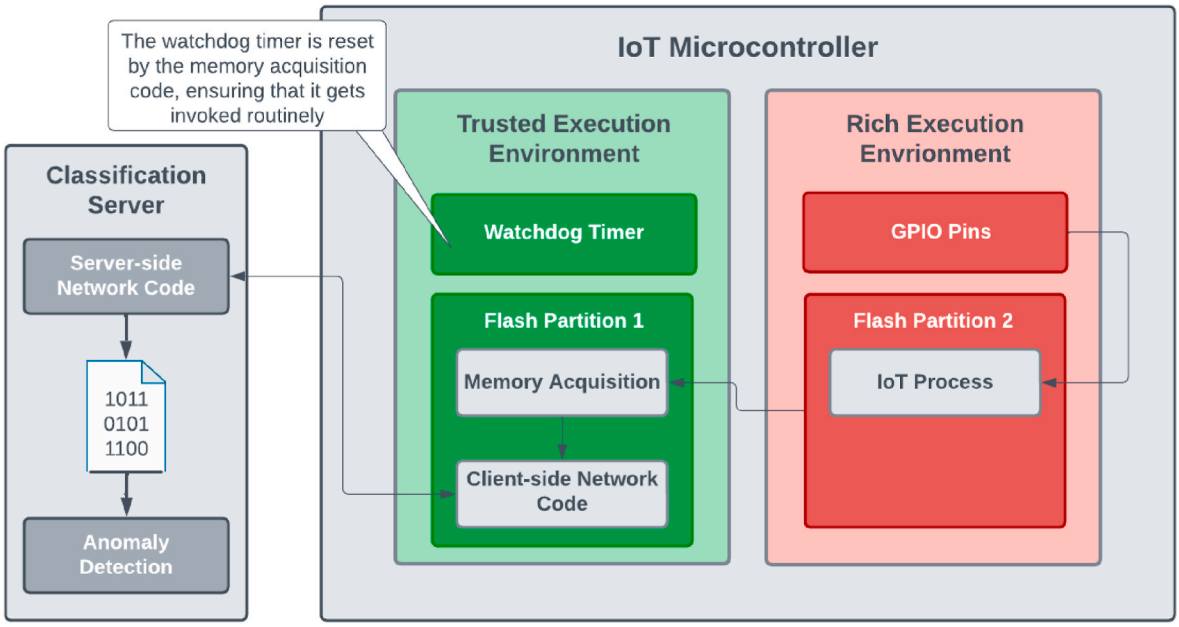

Fig. 2. Generalized system architecture of a host-based intrusion detection system for IoT devices.

block number in a list and keeps a tally of the total number of altered blocks. After all blocks are checked, memory transfer occurs only for blocks that have been modified. The first step involving the transfer of memory from the REE to the TEE can be accomplished via an on-chip Direct Memory Access (DMA) or interrupt-driven I/O. The second step involving sending memory to the server is accomplished by formatting an encrypted TCP stream. This stream includes the count of changed blocks, the list of block numbers, and the actual memory blocks.

3.1.2. Client-side network module

Once the data stream is properly formatted, it is transmitted through a network interface dedicated to the secure world. The transmission of data and the reception of the classification are managed by the client-side network code on the IoT device. The efficiency of this entire process partly depends on the hardware. Many contemporary MCUs are equipped with features like DMA or Serial Peripheral Interface (SPI), which greatly accelerate memory transfer. However, the network hardware's speed could be a significant limiting factor, particularly if it is integrated into the MCU itself.

Sending memory data over the network could disrupt the IoT device's normal operations if not handled properly. We propose that memory acquisition should be done in a blocking manner whenever possible to promote stability and to capture a precise snapshot of the memory at a given time. However, the process of transmitting this data over the network should be non-blocking, using parallel or concurrent programming techniques. This approach requires buffering the changed memory blocks within the secure-world before transmission. To this end, the memory devices must be partitioned to allow the secure-world to store both the trusted firmware and the changed memory from the REE. We believe this is a viable approach, given the typically simple nature of most IoT processes in the non-secure world.

3.1.3. Watchdog timer

The watchdog timer plays a key role in enforcing the routine capturing of memory. Recall that the TEE can only regain control when the IoT process sends a request through a secure interface. To ensure that this occurs, the code to reset the watchdog timer is embedded within the memory acquisition code. If the REE neglects to activate memory acquisition, possibly due to malware infection or system errors, the timer will count down completely, causing the device to reset. Resetting can eliminate some malware, like the Mirai Botnet (Antonakakis et al., 2017). However, more sophisticated malware might persist in non-volatile memory by altering firmware. For these scenarios, a secure boot mechanism that verifies the device firmware is needed. Such a mechanism could reside in the TEE. Upon receiving the classification, if the secure world identifies the latest memory dump as anomalous, the watchdog timer will be allowed to count down fully, triggering a device reset. If no anomaly is detected, the device continues normal operations. This process repeats periodically, governed by an interrupt, and continues during the device's operation.

3.2. Server-side components

The server-side of *MARS* employs an anomaly detection algorithm that analyzes the memory of the non-secure world and categorizes it as either benign or anomalous. Therefore, this component is made up of the client-server-side communication module, the preprocessing module, the training module, and the classification module.

3.2.1. Client-server communication

On the server side, the communication module manages the reception of the memory images from the client. Some of its functions include decrypting the data stream, parsing the received information, extracting the changed memory blocks, and integrating these blocks into the most recent memory dump. After the dump is fully reassembled, the memory is then passed to the preprocessing module.

3.2.2. Feature preprocessing

In any machine learning-based system, effective anomaly classification depends on (1) **meticulous feature selection and extraction**, (2) **robust preprocessing**, and (3) **an accurate training and detection algorithm**.

Preprocessing is a critical step in machine learning, because it prepares and transforms raw data into a suitable format for machine learning models. Our approach uses a binary blob of memory which can be preprocessed in a variety of ways. For unstructured binary data, one can extract features such as byte frequency distribution, n-gram sequence, or string analysis. Alternatively, it can be encoded into an alternative form such as audio or image to provide even more robust features, as was shown in the work of (Vijayakanthan et al., 2023a, 2023b). For our proposed feature preprocessing, we tested the two approaches below:

- **Binary Blob Analysis:** Memory dumps are processed to generate n-gram sequences: contiguous sequences of n items from a given data stream. This method aids in intrusion detection by analyzing the patterns and frequency of these n-grams, providing a set of features that represent specific byte patterns in the memory.
- **Audio Spectrum Transformation:** Memory dumps are converted into audio spectrums. Key features that are extracted include MFCCs, Mel spectrograms, and chroma variants (chroma_stft, chroma_cqt, and chroma_cens).

As we detail in Section 4, we found that audio spectrum transformation provided robust features as well as the best performance.

3.2.3. Training and classification module

This module hosts the training and anomaly detection component of the IDS. It takes the extracted audio spectrum features above and passes them into a CNN model which is trained to differentiate between normal and abnormal (or unknown) memory instances. We chose CNN for its effectiveness in extracting complex patterns in high-dimensional, intricate datasets—an area where other machine-learning approaches fall short. CNN models are particularly effective in identifying local patterns, such as frequency, amplitude, and duration—all of which are crucial for audio classification. The proposed CNN model features a sequential architecture with fully connected layers—including three sets of convolutional layers, each paired with max-pooling layers, followed by a flattening step and then another three sets of dense layers. Learnable filters in the convolutional layers create feature maps highlighting specific patterns, which are then processed through activation functions like ReLU to add non-linearity. Pooling layers help in learning more complex representations of the input data. The model iteratively improves its understanding of the complex data through these layers. Finally, the features are flattened and passed through fully connected layers for the binary classification task. In our model, a class label '1′ denotes normal memory and '0' indicates a potentially abnormal memory image.

Thus, the interaction between the server and client in this proposed HIDS architecture unfolds as follows: The client first acquires and transmits the memory data to the server. Upon receipt, the server preprocesses and extracts features from this memory image. These features are then input into the anomaly detection algorithm for binary classification. After analyzing the data, the algorithm determines whether it is benign or anomalous and sends this classification result back to the MCU via the server-side communication. Subsequently, the MCU is enabled to act accordingly using the watchdog timer.

4. System implementation

This section details the implementation of a proof-of-concept for *MARS* using the STM32L562QEI6QU MCU, henceforth referred to as "STM32". The STM32 is an ARM TrustZone-enabled device, selected for

its relevance in prototyping IoT solutions. To meet the hardware isolation requirements essential for the IDS, we leveraged a TEE provided by ARM TrustZone which partitioned the system into secure and non-secure realms. This setup provides an environment that simulates a real-world IoT device. Although our implementation is specific to this board, the principal design can be adapted across different MCUs.

4.1. Client implementation

The client component of the IDS has three functions: (1) secure acquisition of device memory, (2) transfer of memory to the server, and (3) the handling server responses. Prior to discussing these functions, we will explain how we chose to partition the STM32 device.The STM32 device is equipped with two 256 KB flash banks. In our implementation, the entirety of one flash bank is allocated to the REE, while the second is dedicated to the ARM TrustZone. This allocation strategy simplified the configuration bits that needed to be loaded into the STM32 registers. In addition to having 256 KB of flash from the STM32 allocated to TrustZone, we had all 4 MB of flash from the ESP32 chip, which we leveraged for memory buffering. The STM32's SRAM, which serves as the device's runtime memory, was allocated exclusively to TrustZone, preventing modification by the REE. For data transfer, we allocated several peripherals, including a DMA and SPI interface to TrustZone. The DMA facilitates secure communication and data transfer from the non-secure to the secure environment, while the SPI, connected to an external ESP32 board, enables memory transfer over the network. This setup leverages the ESP32's Wi-Fi capabilities to transmit memory images to the classification server since the STM32 does not have integrated Wi-Fi. It is important to note that since the ESP32 is wired to a secured SPI interface, it is considered as belonging to TrustZone in its entirety, thus is our ability to use it within the IDS design without violating our need for isolation.

4.1.1. Memory acquisition process

The memory acquisition process is shown in Fig. 3. The component of the memory acquisition residing on the STM32 is built directly into the TEE firmware in approximately 3000 lines of C using device drivers. It is initiated by a secure-callable function from the REE, triggering the DMA to snapshot memory in blocks. These blocks are securely transferred 1024 bytes at a time to the secure world before being sent through TCP. At the initial setup of a target device, our design mandates a full memory dump to be executed n number of times, where n is determined by the user. These initial memory images are crucial for server-side model training, tailored specifically to the target device.

After establishing a baseline, the module will then periodically analyze the non-secure flash memory for any modifications, rather than acquiring a full dump. Each memory block is subjected to MD5 hashing to identify any changes. Blocks with hashes that differ from their previously stored values are earmarked for transfer. This selective transfer process efficiently minimizes the data sent to the ESP32 for network transmission, thereby reducing unnecessary memory transfer and enhancing the system's overall efficiency.

4.1.2. Memory transfer

The entire memory or the identified blocks are then transferred to the SPI interface, connected to the ESP32, for network transmission. This stage involves re-computing the memory address of each modified block for direct transfer to the SPI, optimizing the process by bypassing additional buffering in the secure world. The ESP32's DMA facilitates the buffering of incoming data, ensuring efficient network transmission through TCP to the classification server.

4.1.3. Watchdog timer

The system employs an independent watchdog timer allocated to the TrustZone to ensure timely memory acquisition. If the non-secure world fails to trigger the memory acquisition process within a predetermined interval (possibly due to intrusion), the watchdog timer resets the device. This mechanism enforces regular memory checks.

4.1.4. Handling server response

The server, upon receiving and analyzing memory dumps or modified blocks from the IoT device with the developed model, will generate a classification indicating the security status of the device. Namely, '1′ denotes that the device is operating within expected parameters and a '0' indicates the detection of potential anomalous activity that could suggest a security threat or intrusion.

Once the client receives the server's response, it will decide how to respond. If a '0′ signal is received, the IDS will respond by not resetting the watchdog timer. This deliberate inaction would lead to a system reset, part of the predefined security protocols mentioned above.

4.2. Server-side implementation

The server-side architecture of our IDS plays a pivotal role in analyzing memory dumps transmitted from the IoT device to detect anomalies. Developed in about 200 lines of Python, this component leverages the extensive library support of the language to facilitate network communication and integrate with machine learning models. The server–side processes include memory image preprocessing, feature extraction, CNN model development, and client communication module–each critical for the nuanced analysis required to classify memory as benign or potentially malicious.

4.2.1. Memory image preprocessing

Preprocessing of memory images is the initial step in the server's analysis, transforming raw memory dumps into a more analyzable format. Developed in about 130 lines of Python, this transformation is crucial to detecting underlying patterns or anomalies in the runtime memory which may not be discernible in its original binary form. It is these features that get passed to the model for classification. As stated in Section 3, we considered two transformative approaches: memory to binary using n-gram and memory to audio transformation. We will explain these two approaches in detail and illustrate why the memory to audio transformation is the best.

Memory to Binary Using N-Gram - In this transformation, each

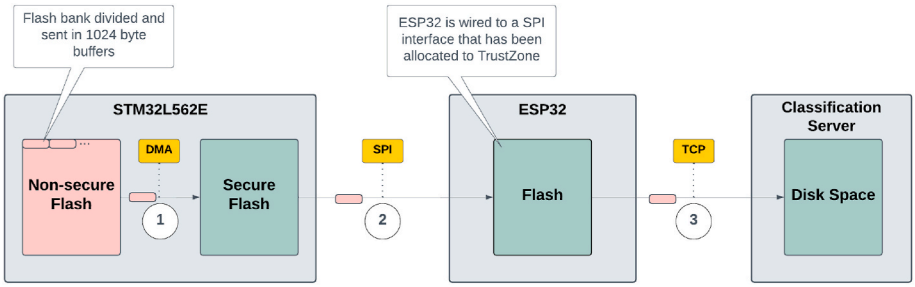

Fig. 3. Memory acquisition process from MCU to classification server.

blob of memory is transformed into a set of n-gram sequences. To analyze a memory blob of 256 KB using n-gram analysis for various sizes of n-gram (1-byte, 2-byte, 3-byte, 4-byte, and 5-byte grams), we first compute the number of n-grams and then multiply the number of n-grams by the size of the n-gram to determine the total bytes represented by each set of n-grams. This computation is represted in equation 1, where size_data represents the total bytes, n represents the number of n-grams, and size_n_gram represents the size of the n-gram.

$$\text{size}_{data} = \text{n} \times \text{size}_{n\_gram} \quad \text{Equation 1}$$

The computation in Table 1 shows that as the size of n-grams increases, the total bytes represented by each set of n-grams exhibits an approximate fold increase directly proportional to the n-gram size. A lower n means less number of bytes to process, and hence, reduction in the computational complexity. However, a lower value of n is often less informative in understanding context. Particularly, in binary analysis, the chosen n should be large enough to capture relevant patterns or anomalies in the data without diluting the analysis with excessive noise. Therefore, given that efficiency is one of the overarching objective of this IDS, this transformation method is not ideal.

Memory to Audio Transformation - This conversion employs audio spectrum transformation techniques, facilitating a richer analysis by highlighting unique characteristics inherent in the memory data. Our transformation goal here had two fundamental requirements:

1. **The converted audio spectrum cannot be significantly larger than the original memory file size.** If the converted audio spectrum is too large, it will be too computationally expensive and time-intensive for the server to process, thus hindering the system's efficiency.
2. **The transformation must be lossless.** The audio spectrum needs to be a complete and accurate representation of the memory image so that no information is lost or compromised and analysis is accurate and undiminished.

To achieve these requirements, we wrote scripts using four libraries for audio spectrum processing: Librosa (librosa/librosa: 0, 2023), Wave (Python Software Foundation), SoX (Chris Bagwell, 2015), and FFmpeg (Developers, 2023). These conversion methods are tested on a 1 KB memory image.

The results as shown in Table 2 indicate that while the output from the Librosa conversion doubles the size of the original memory, the outputs from the three alternative approaches Wave, SoX and FFmeg have almost the exact size of the original memory, with few bytes overhead for the header. Even at higher sampling rates—which allow for greater precision in extracting key features from memory—the file size stays the same for the Wave and SoX library as it changes sampling rates. The FFmpeg library only increases audio spectrum file size by 24 bytes at sampling rates beyond 48 kHz. Applying this result to our 256 KB memory, the audio transformation using Wave and SoX will indeed not result in a size much larger than the original memory, thus satisfying the first requirement.

To test the second requirement that these transformations are indeed lossless, we re-transform the output audio in Table 2 above back into binary. The hashes of the original input were then compared to the hashes of this output and the result shows they are identical, indicative that the transformation is indeed lossless.

Therefore, given these test results, our feature preprocessing adopted the memory to audio transformation technique.

Table 1
Output of memory to binary conversion using N-gram.

n-gram size	n-gram	bytes
1-byte gram	262,144	262,144
2-byte gram	262,143	524,286
3-byte gram	262,142	786,426
4-byte gram	262,141	1,048,564
5-byte gram	262,140	1,310,700

Table 2
Output of memory to audio signal transformation.

Sampling Rate (Hz)	Librosa (bytes)	Wave (bytes)	SoX (bytes)	FFmpeg (bytes)
42,000	2044	1044	1044	1078
44,000	2044	1044	1044	1078
48,000	2044	1044	1044	1078
50,000	2044	1044	1044	1102
88,200	2044	1044	1044	1102
96,000	2044	1044	1044	1102
176,400	2044	1044	1044	1102
192,000	2044	1044	1044	1102

4.2.2. *Feature extraction*

Following preprocessing, the server extracts critical features from the transformed memory images to facilitate effective anomaly detection. This step is crucial for reducing the dimensionality of the data while retaining important information that the CNN model can analyze. Key features extracted from the audio spectrum representation of memory data include Mel-frequency cepstral coefficients (MFCCs), Mel spectrograms, and chroma features such as chroma_stft, chroma_cqt, and chroma_cens. These features provide a comprehensive profile of the memory, capturing its unique characteristics and patterns. The extraction process employs various digital signal processing techniques to isolate and quantify these features, preparing them for input into the CNN model. This meticulous approach ensures that the most informative aspects of the memory-encoded audio spectrum are highlighted for analysis.

4.2.3. *CNN model development*

Model Architecture: Our CNN model, implemented using TensorFlow's Keras API, adopts a sequential architecture comprising three convolutional layers, followed by max pooling and dropout layers, a flattening layer, and three dense layers. Developed in about 150 lines of Python, the model's architecture is strategically designed to incrementally capture and analyze the complex patterns present in the input features.

Convolutional Layers: These layers utilize 32, 64, and 128 filters, respectively, to detect various patterns within the input features. Max pooling layers follow each convolutional layer to reduce the dimensionality of the data while retaining essential information. Dropout layers are incorporated to prevent overfitting by randomly omitting a subset of neurons during training.

Dense Layers and Output: The network is built with three dense layers, with the final layer consisting of two neurons and a sigmoid activation function, classifying the input as benign or anomalous. This configuration allows the model to learn and identify complex feature patterns indicative of potential security threats.

Training and Optimization: The model is compiled with the Adam optimizer and binary cross-entropy loss function, chosen for their efficiency and effectiveness in binary classification tasks. The Adam optimizer is particularly suited for this application due to its adaptive estimation capabilities and minimal memory requirements, while binary cross-entropy accurately evaluates the divergence between predicted and actual binary labels.

Fig. 4 summarizes the overall classification process including preprocessing. By the time preprocessing is complete, the raw memory dump is converted to an audio spectrum on which feature extraction occurs. The features are passed to the trained CNN model, and based on these, the memory dump is classified as benign or anomalous.

Fig. 4. Overview of memory-based anomaly detection algorithm.

4.2.4. Network communication

The server communication module has two main functions: (1) it receives the memory data from the client and (2) sends the classification back to the client after the anomaly detection classifies the memory. In the first function, when the server receives the initial setup n-memory dumps, it is designed to forward those images to the memory to audio transformation component. On the other hand, when the server receives modified blocks, it integrates the changed blocks into the previous memory before passing it to the transformation function.

For its second function, the server's response is encapsulated in a TCP packet so that the device can respond appropriately.

5. Testing and evaluation

The primary goal of our approach is to use device memory to distinguish between normal and anomalous activities. As outlined in Section 1, we focus on achieving three key objectives. This section aims to evaluate *MARS* 's proof of concept against these objectives. To achieve this, we developed test beds:

1. Infrared Motion-Sensing Application: This application detects motion and transmits signals to a server. It serves as a hypothetical example for monitoring purposes, such as detecting unauthorized entry into restricted areas of a centrifuge facility.
2. Blinking LED Application: This application is designed to alter its LED color in response to changes in system status, providing a visual indicator of different operational states.

Both applications are configured to relay their findings to a server via Netcat, listening on specific ports dedicated to each application. This setup allows us to evaluate the practicality and effectiveness of our approach in real-world scenarios, measuring its performance across the aforementioned objectives.

5.1. Testing for Trust and Integrity

The goal of this testing is to ensure that memory images are complete and valid during acquisition and after transfer. The apps' non-secure memory transferred to TrustZone was authenticated by routing it to a UART port connected to the embedded programmer. Next, selected memory blocks (including the first and last) were scrutinized against the original content in the non-secure flash bank using a hardware-based memory inspection feature available in STM32CubeProgrammer. It is important to note that this inspection capability, while valuable for validation, is not feasible for inclusion in the IDS for a production IoT device due to its reliance on an embedded debugger that would be absent in the final product. Additionally, the overall size of the memory dumps was confirmed to be precisely 256 KB. Finally, to validate the integrity of the data received server-side, the memory dumps transmitted to the server were cross-referenced with the comprehensive dumps extracted via UART, employing a diff program to ensure a complete match.

5.2. Testing for accuracy

To evaluate the server-side classification engine's accuracy, we compiled a dataset comprising 20 memory images from the Infrared app–representing normal operational data–and 20 memory images from the Blinking LED app–categorized as anomalous data. This dataset range is balanced and does not create inefficiency that would occur if one had to wait on, for example, 100 memory samples to train upon server initialization. This collection of 40 memory images was allocated for training and testing purposes, adhering to a 70:30 split, respectively. Our choice of using a CNN model focuses on training the model to recognize patterns of known images, going beyond surface-level differences to understand each segment's underlying structure and context.

The outcomes of our training, as shown in Table 3, indicate the model's performance metrics, including a 100 % test accuracy rate, along with Loss, F1-Score, Recall, and Precision values. Notably, the entire process—from memory-to-audio preprocessing through to the execution of training and testing phases—was completed in approximately 43 s, underscoring the system's efficiency.

It is crucial to highlight that *MARS* 's design is inherently flexible, accommodating the integration of various types of classification models as per user or organizational requirements. Implementing a new engine merely involves replacing the existing classification component with the desired alternative, illustrating the system's plug-and-play capability.

5.3. Testing for robustness

To assess the accuracy of our classification system in identifying memory alterations, we implemented a stress testing methodology. This method involves deliberately inducing random bitflips (Antoine Grondin, 2020) within the memory image to simulate byte changes at varying levels: 0.1 %, 0.2 %, 0.5 %, and 1.0 %. The goal of this approach is to determine the extent of changes required for the classification engine to accurately identify the memory as anomalous. Our evaluation results as shown in Table 4 show that our model is able to correctly classify, with 100 % accuracy, changes in memory starting at 1.0 % bytes changed. The discrepancies associated with percentages lower than 1.0 % likely have to do with the regions of memory that were randomly chosen to be bitflipped, as some memory regions are less susceptible to detection due to their lower importance in overall system behavior. These results are very promising considering that the average IoT malware, like Mirai (size = 68 KB), will occupy ~27 % of a 256 KB memory. Our method's ability to detect as little as a 1 % change is significant and suggests it could effectively detect threats with even smaller footprints.

5.4. Testing for performance overhead

Although, performance was not a key objective of this prototype, nonetheless, evaluating overhead is always crucial in system design. In this test, we evaluated *MARS* 's performance overhead during execution. First, we computed the average time it takes to capture the initial n-memory images at the system setup. It is important to note that the

Table 3
Performance metrics of server-side classification engine.

Accuracy	1.0
Loss	$7.892257417552173 \times 10^{-5}$
F1_Score	1.0
Recall	1.0
Precision	1.0

Table 4
Stress testing of System's classification at different percent byte changes of memory.

Bytes Changed	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
0.0 %	Benign	Benign	Benign	Benign	Benign
0.1 %	Benign	Benign	Benign	Benign	Anomalous
0.2 %	Benign	Benign	Anomalous	Benign	Benign
0.5 %	Anomalous	Benign	Benign	Anomalous	Benign
1.0 %	Anomalous	Anomalous	Anomalous	Anomalous	Anomalous

overhead for gathering n-memory dumps is non-disruptive and will only be completed once. Our evaluation on the Infrared app shows the acquisition process for one full dump (including sending over TCP) takes about 5 min to complete. By contrast, the amount of time it takes to acquire and buffer one full dump using the on-chip DMA (without sending over the network) takes 500 ms. While acquiring memory via the on-chip DMA is fast, we found that it takes much longer to transmit this data over the network. Notably, the transfer rate for sending data to the server using the integrated Wi-Fi within the ESP32 was about 1 kB/s, making it an obvious bottleneck in this process. We chose to use integrated Wi-Fi within the ESP32 to workaround the lack of a network interface on the STM32 board. However, faster networking hardware would help minimize or perhaps even eliminate this bottleneck.

Next we computed the average time it takes to identify modified memory blocks, acquire them and then transfer them. Our test indicate it takes 8 s for all the three steps to complete for changes in 1 block. Although acquiring and buffering one block takes 2 ms and sending over the network takes 1 s, there is some overhead involved in synchronizing communication between the STM32 and ESP32 development boards. We also found that we had to implement a message protocol on top of SPI to ensure reliable transfer of data between the two boards. The higher-than-expected transfer time is attributed to the work that must be done to synchronize the two chips and transfer and parse communication signals between them. Thus, the higher overhead is once again caused by the STM32 development board's lack of an on-board network interface. This overhead could be eliminated on a board with integrated networking.

5.5. Limitation and future work

In this paper, we introduce and demonstrate a proof-of-concept for *MARS* designed as a client-server architecture. Although the initial prototype demonstrates promising accuracy, robustness, and integrity, there are limitations in its design and evaluation that require future attention: **(1) Real Malware Attack Evaluation**: Future work should include testing the IDS against actual malware attacks to validate its effectiveness in real-world scenarios. **(2) Complex IoT Application Testing**: Both the client and server components of the system need to undergo testing with more complex IoT applications to ensure their compatibility and performance across diverse environments. **(3) Secure-Boot**: Sophisticated malware may have a way of surviving reboots. Thus future implementations should be extended with a secure-boot mechanism capable of re-flashing the device with clean firmware **(4) SRAM collection**: The current memory acquisition collects flash memory; however, future implementations should be extended to capture SRAM as well to capture variable data. **(5) Client Component Debugging**: Some client components, notably the watchdog timer, have exhibited issues that necessitate further debugging to enhance system reliability and functionality.

6. Related work

There have been numerous design algorithms proposed in the literature for both NIDS and HIDS. Anomaly-based NIDS, such as (Ullah Jan et al., 2019; Eskandari et al., 2020), show high accuracy in detecting network anomalies. However, they struggle with attacks that do not alter packet transmission rates and demonstrate inconsistent performance across different intrusion types. Similarly, Fatani et al.'s approach (Fatani et al., 2021) uses a CNN model for feature extraction and displays inconsistent results due to IoT network diversity. In contrast, signature-based NIDS, proposed by Lo et al. (Weng Lo et al., 2022) and Altunay et al. (Altunay and Albayrak, 2023), demonstrate high accuracy using advanced models like GNNs, CNN, and LSTM but face challenges with new or evolving attack patterns. On the other hand, HIDS-based models like those proposed by Gassais et al. (2020) and Shobana et al. (Shobana and Poonkuzhali, 2020), which rely on system call traces, offer high accuracy but encounter limitations due to the heterogeneity of traces or system calls, affecting algorithm performance. The implementation of SEHIDS (Baz, 2022), a Self Evolving Host-based Intrusion Detection System for IoT networks, involved ARM Development Studio and lightweight ANN to adapt dynamically to threats and showed high accuracy with minimal resource consumption. However, the work focuses on specific datasets and a particular IoT device configuration (OpenMote-B) leaving practical deployment challenges and scalability in real-world IoT environments unexplored. Similarly, Mendonca, Robson V. et al. (Mendonca et al., 2022) deployed a Sparse Evolutionary Training (SET) model on Raspberry Pi for cybersecurity in Industry 4.0's IIoT, achieving 99 % accuracy and improving attack detection by 6.25 %. However, the study's focus on specific datasets and MCUs may limit generalizability and potential challenges in scalability and diverse IIoT environments are not extensively discussed. In contrast to existing literature, our proposed approach is not just an algorithm but an overall system design and implementation for IoT device host-based intrusion detection that leverages an effective CNN model for classification to detect changes in device memory.

7. Conclusion

In this paper, we proposed *MARS*: a client-server HIDS architecture that leverages device memory to detect anomalies in IoT operational functionalities. The memory samples which are collected through on-device module that resides in the MCU's TEE are transformed to audio spectrum, from which distinguishing MFCC features are extracted and passed through a trained CNN model for anomaly detection. Our evaluation shows *MARS* meets its goals of integrity, accuracy, and robustness, proving its viability. During tests, *MARS* had fairly minimal performance overhead–indicating its suitability for operational environments without hindering system performance. *MARS* 's secure handling of memory images also ensures its reliability. The robust classification engine is exhibited by solid performance metrics (100 % test accuracy rate, F1-Score, Recall, and Precision) and demonstrates its ability to distinguish between normal and anomalous behaviors. Stress testing, which involved simulating memory alterations through random bit flips, provided further evidence of *MARS* 's ability to detect anomalies under varied conditions. Despite these positive outcomes, the *MARS* prototype has limitations that future work will address: including testing against real malware attacks, compatibility with complex IoT applications, and improving client-component reliability.

Acknowledgements

This work is supported by NSA Grant No. H98230-21-1-0166.

References

Altunay, Hakan Can, Albayrak, Zafer, 2023. A hybrid cnn+lstm-based intrusion detection system for industrial iot networks. Engineering Science and Technology, an International Journal 38, 101322. https://doi.org/10.1016/j.jestch.2022.101322.

Antoine Grondin. Bitflip, 2020.

Antonakakis, Manos, April, Tim, Bailey, Michael, Bernhard, Matt, Bursztein, Elie, Cochran, Jaime, Durumeric, Zakir, Halderman, J Alex, Invernizzi, Luca, Kallitsis, Michalis, et al., 2017. Understanding the mirai botnet. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 1093–1110.

Baz, Mohammed, 2022. Sehids: Self evolving host-based intrusion detection system for iot networks. Sensors 22 (17), 6505.

Chris Bagwell. Sox, 2015.

Developers, FFmpeg, 2023. Ffmpeg.

Eskandari, Mojtaba, Janjua, Zaffar Haider, Vecchio, Massimo, Antonelli, Fabio, 2020. Passban ids: an intelligent anomaly-based intrusion detection system for iot edge devices. IEEE Internet Things J. 7 (8), 6882–6897. https://doi.org/10.1109/JIOT.2020.2970501.

Fatani, Abdulaziz, Elaziz, Mohamed Abd, Dahou, Abdelghani, Al-Qaness, Mohammed A. A., Lu, Songfeng, 2021. Iot intrusion detection system using deep learning and enhanced transient search optimization. IEEE Access 9, 123448–123464. https://doi.org/10.1109/ACCESS.2021.3109081.

Gassais, Robin, Ezzati-Jivan, Naser, Fernandez, Jose, Aloise, Daniel, Dagenais, Michel, 2020. Multi-level host-based intrusion detection system for internet of things. J. Cloud Comput. 9 (62).

Librosa/Librosa: 0.10.1, 2023.

Mendonca, Robson V., Silva, Juan C., Rosa, Renata L., Saadi, Muhammad, Demostenes, Z Rodriguez, Ahmed, Farouk, 2022. A lightweight intelligent intrusion detection system for industrial internet of things using deep learning algorithms. Expet Syst. 39, e12917.

Python Software Foundation. Wave: Python Standard Library. .

Shobana, M., Poonkuzhali, S., 2020. A novel approach to detect iot malware by system calls using deep learning techniques. In: 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), pp. 1–5. https://doi.org/10.1109/ICITIIT49094.2020.9071531.

Ullah Jan, Sana, Ahmed, Saeed, Shakhov, Vladimir, Koo, Insoo, 2019. Toward a lightweight intrusion detection system for the internet of things. IEEE Access 7, 42450–42471. https://doi.org/10.1109/ACCESS.2019.2907965.

Vijayakanthan, Ramyapandian, Ahmed, Irfan, Ali-Gombe, Aisha, 2023a. Swmat: Mel-frequency cepstral coefficients-based memory fingerprinting for iot devices. Comput. Secur. 132, 103298.

Vijayakanthan, Ramyapandian, Waguespack, Karley M., Ahmed, Irfan, Ali-Gombe, Aisha, 2023b. Fortifying iot devices: Ai-driven intrusion detection via memory-encoded audio signals. In: 2023 IEEE Secure Development Conference (SecDev). IEEE, pp. 106–117.

Weng Lo, Wai, Layeghy, Siamak, Sarhan, Mohanad, Gallagher, Marcus, Portmann, Marius, 2022. E-graphsage: a graph neural network based intrusion detection system for iot. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, pp. 1–9. https://doi.org/10.1109/NOMS54207.2022.9789878.