Oops, It Halted Again: Exploiting PLC Memory for Fun and Profit in Industrial Control Systems

Wooyeon Jo, Irfan Ahmed
Department of Computer Science,
Virginia Commonwealth University, USA

Abstract

Programmable Logic Controllers (PLCs) are critical to industrial control systems (ICS), yet their memory remains a prime target for exploitation. While traditional attacks focus on network intrusions, PLC memory manipulation enables sophisticated attacks, such as malicious process control and supply chain backdoors. Existing security measures, including intrusion detection systems (IDS), fail to detect these threats, necessitating a systematic approach to analyzing and exploiting PLC memory. This paper presents a machine learning-driven framework for PLC memory exploitation, identifying critical regions vulnerable to unauthorized access and manipulation. Using extracted features such as entropy-based and structural characteristics, we classify PLC memory into exploitable segments, including metadata and control logic. Our method enables precise targeting of PLC memory for adversarial access, injection, and modification, operating independently of PLC-specific semantics. By training on an M221 PLC, we demonstrate its generalization across architectures, successfully exploiting PLCs with distinct instruction sets. We evaluate our approach on three PLCs from two vendors, actively probing memory to elicit responses such as accept, deny, halt, and compromise. The results expose inconsistencies in memory protections across PLC architectures, reinforcing the need for improved memory integrity in ICS environments. As part of our research, we identified and disclosed a critical PLC memory vulnerability (CVE-2024-11737)

1 Introduction

Industrial control systems (ICS) are the backbone of critical infrastructure, governing processes in sectors such as energy, manufacturing, and chemical industries [1, 2]. At the core of these systems lie programmable logic controllers (PLCs), responsible for executing automated operations. However, PLC security has historically been neglected, making these devices an attractive target for adversarial exploitation. While network-based attacks on ICS have received significant attention, attacks targeting PLC memory remain a largely underestimated yet highly effective vector for system compromise.

Recent high-profile cyberattacks [3–8] demonstrate an evolution towards more sophisticated ICS intrusions, moving beyond traditional network penetration to direct manipulation of embedded controllers with specific timing and intended commands [9–11]. Unlike network attacks, memory exploitation bypasses communication-level defenses, enabling attackers to manipulate PLC execution logic, inject persistent backdoors, and alter system behavior at a fundamental level. Even stealthy supply chain attacks can be carried out by embedding malicious logic into firmware or modifying memory-based control flows.

Despite its criticality, PLC memory remains one of the least protected components in industrial systems [12]. Security mechanisms such as intrusion detection systems (IDS) focus on network-layer anomalies, leaving memory-based attacks largely undetected. Furthermore, memory acquisition and analysis are highly constrained by vendor restrictions, proprietary formats, and device-specific architectures, limiting the ability to systematically analyze PLC memory vulnerabilities [13].

In this research, we propose a machine learning-driven approach to PLC memory exploitation, systematically classifying critical memory regions that can be targeted for adversarial access, injection, and modification. Unlike traditional forensic methods, our approach does not rely on PLC-specific semantics, making it applicable across different architectures. We train our model on an M221 PLC and successfully demonstrate its generalization across multiple PLCs, even those with distinct instruction sets.

We evaluate our method on three PLCs from two vendors (i.e., Schneider Electric's Modicon M221 and M241 and Allen-Bradley's 1756), actively probing their memory to elicit various system responses, including accept, deny, halt, and compromise. Our findings reveal critical inconsistencies in memory protections across PLC architectures, exposing the feasibility of targeted memory exploitation. By demonstrating these vulnerabilities, this work underscores the urgent need

for improved memory integrity protections in ICS environments while also providing a blueprint for future offensive security research in PLC exploitation.

The remainder of this paper is structured as follows. Section 2 provides essential background, outlining the challenges of PLC memory analysis and attack feasibility. Section 3 defines the threat model and attack vectors, detailing adversarial strategies leveraged against PLC memory. Section 5 introduces the selected machine learning features designed for PLC memory classification, followed by Section 7.1, which evaluates their effectiveness through comparative and correlation analysis. Section 6 describes the experimental environment, memory acquisition methods, and attack execution strategies. Section 7 presents the results, analyzing attack outcomes and their implications for PLC security. Section 9 provides an in-depth discussion of our findings, addressing key limitations. Finally, Section 10 summarizes our findings.

2 Background and Challenges

Industrial control systems (ICS) rely on programmable logic controllers (PLCs) to automate physical processes across critical infrastructure sectors. These PLCs operate within industrial networks, utilizing industrial communication protocols over Ethernet to execute control logic and interface with sensors and actuators, as shown in Figure 1. Unlike general-purpose computing systems, PLCs are embedded devices with constrained resources and minimal built-in security, making them susceptible to targeted memory attacks.

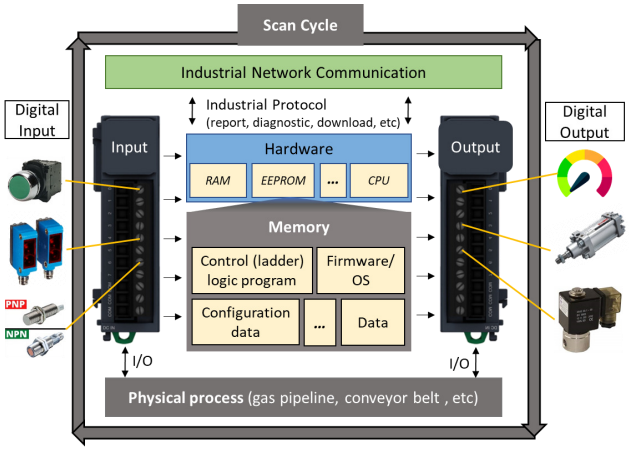

Figure 1: PLC Overview

Control logic is developed externally using engineering software before being deployed to PLCs over a network. This software often integrates Human-Machine Interface (HMI) functionalities, enabling operators to monitor and adjust industrial processes remotely. Given that all control logic transfers occur over industrial communication protocols, any compromise in these interactions can lead to unauthorized access, manipulation, or injection of malicious control logic.

To support execution, PLCs maintain a diverse set of memory structures, including execution logic, configuration data, and system metadata. Among these, some memory regions are of particular interest for adversarial operations. Critical memory regions store execution-relevant data such as compiled control logic and system metadata, which, if manipulated, could alter PLC behavior or cause system-wide failures. Unlike generic data, these critical regions offer high-value targets for memory exploitation, enabling persistent modification, stealthy backdoors, or direct control over physical processes.

Despite the significance of PLC memory security, existing protections are either absent or highly inconsistent across vendors. Unlike network-based attacks, memory-based exploits bypass communication-layer defenses and directly manipulate execution states, making them particularly stealthy and difficult to detect. The lack of standardization in PLC memory structures further complicates security analysis, creating a fragmented security landscape where attackers can exploit vendor-specific inconsistencies.

2.1 Problem Statement

Extracting and analyzing PLC memory presents a critical attack surface, yet traditional forensic techniques demand extensive manual effort and provide limited insight into exploitable regions. This research introduces a machine learning-driven approach to systematically classify critical memory regions for adversarial manipulation. By mapping these regions, we enable precise targeting for access, injection, and modification attacks, exposing fundamental weaknesses in PLC memory protection mechanisms. Our method demonstrates that memory exploitation is feasible across multiple PLC architectures without reliance on PLC-specific semantics, significantly broadening the attack surface for ICS threats.

2.2 Challenges

Executing effective memory exploitation on PLCs presents several significant challenges. Unlike traditional IT systems, PLC memory is highly fragmented, lacks standardization, and often incorporates proprietary structures that complicate both analysis and attack execution. Furthermore, attackers must deal with inconsistent memory layouts, ambiguous metadata structures, and countermeasures that hinder precise targeting. The following challenges outline key obstacles attackers face when attempting to exploit PLC memory.

Challenge 1: Memory Space Mapping in PLCs

PLCs integrate multiple chips, often combining embedded memory with external components such as System-on-Chip (SoC) architectures. These memory segments are mapped into a unified address space, but the mapping strategy varies across

vendors. Without precise knowledge of memory segmentation, attackers cannot reliably locate or manipulate critical memory regions.

Furthermore, PLCs often rely on vendor-specific memory layouts that lack standard documentation. This forces attackers to either reverse-engineer the memory mapping manually or perform blind, iterative testing to infer the correct addresses—both of which introduce delays and uncertainty in exploitation attempts.

Challenge 2: The Absence of Ground Truth

Unlike operating systems such as Linux, which have standardized memory layouts and well-documented file systems, PLC memory lacks a universal structure. Each vendor defines its own proprietary format, meaning there are no predefined ground truth references to rely on when analyzing memory dumps.

For attackers, this uncertainty poses a significant challenge. Memory regions may shift across firmware versions, control logic may be stored in different sections, and even metadata structures can vary. As a result, an exploit developed for one PLC may not be directly transferable to another model, requiring additional reconnaissance and adaptability in attack strategies.

Challenge 3: Every Data Segment Is a Potential Pointer

PLCs frequently use direct memory references to optimize performance, meaning that many seemingly random data values may serve as pointers. Because most PLCs operate within a 4GB memory space, these pointers are commonly 4-byte values within the `0x00000000`–`0xFFFFFFFF` range.

For attackers, this introduces a major difficulty: any 4-byte sequence could potentially be a pointer. Since pointers do not follow a strict pattern, a naïve overwrite attack risks breaking the execution flow. Attackers must first identify valid pointer structures and understand their reference hierarchy to craft effective exploits.

Challenge 4: Identifying the Root of Execution

All PLC data is structured hierarchically, with metadata guiding execution flow [14]. Upon boot, PLCs load data from predefined regions, processing multiple layers of metadata before executing control logic. However, the exact location of the root metadata varies between devices, and any miscalculation in targeting can result in an ineffective or unintended attack.

Attackers seeking to alter control flow or inject payloads must precisely locate and manipulate these root metadata entries. If an attack modifies secondary metadata instead of the root reference, the exploit may be ignored or overridden by the PLC's internal processes.

Challenge 5: Leveraging RAP for Exploitation

The Redundant Address Pin (RAP) phenomenon [9] presents a unique opportunity for adversaries. RAP creates multiple memory regions that appear to contain the same data, but only one holds the actual data. Since these redundant regions are mapped within the same memory space, attackers can abuse this behavior to redirect writes to controlled areas while affecting execution-critical data.

By targeting a RAP-mapped shadow region instead of modifying the primary control logic directly, an attacker can inject malicious modifications in an indirect and stealthy manner. However, this method is only feasible if RAP regions can be reliably identified and their existence confirmed within the target system.

Challenge 6: Targeting for Precise Exploitation

A major challenge in PLC exploitation is the lack of static memory addresses for critical data. Control logic does not always download in the same location. Each time a new control program is downloaded, the memory layout may shift slightly [15, 16], introducing randomized offsets for critical memory regions.

For an attacker, this presents a major problem: An exploit must precisely target control logic memory—a misaligned payload could result in a failed attack or an unintended denial-of-service (DoS) condition. Even a single-bit misalignment can degrade a sophisticated exploit into a simple DoS attack, severely reducing its impact. To overcome this, attackers must dynamically infer control logic locations using indirect methods, such as memory heuristics, execution tracing, or real-time reconnaissance techniques.

These challenges illustrate the complexity of PLC memory exploitation. Attackers must navigate fragmented memory layouts, infer undocumented structures, and execute payloads under strict real-time constraints. Moreover, exploiting vendor-specific memory quirks, such as RAP behavior, can enhance attack effectiveness by allowing stealthy modifications. Understanding and overcoming these barriers is essential for developing reliable and scalable PLC memory exploitation techniques.

3 Threat Model and Attack Vectors

This section outlines the attack framework by defining the assumed threat model, detailing the attack vectors used to exploit PLC memory, and categorizing system responses under adversarial conditions. First, we establish the attacker's capabilities and environmental constraints in the threat model. Then, we describe the specific attack vectors employed to manipulate memory structures. Finally, we analyze how the PLC reacts to these attacks in the PLC response taxonomy, which classifies observable system behaviors during exploitation.

3.1 Threat Model

In this paper, we assume an attacker capable of accessing PLC memory via an embedded proprietary protocol over Modbus. Direct and indiscriminate memory access attempts are easily blocked by system protections, requiring the attacker to employ a more strategic approach. To maximize the impact of the attack, we first analyze the PLC memory using machine learning techniques to classify different memory regions and identify critical boundaries. This allows for precise targeting of memory areas that can be manipulated to induce a desired system state.

While a comprehensive attack covering the entire memory space is conducted for evaluation purposes, the primary attack strategy focuses on targeted exploitation of memory areas classified through ML analysis.

3.2 Attack Vectors

The proposed attack consists of three distinct methods: access (read), inject (write), and modification (write).

Access: The attacker attempts to read from memory to infer structural properties. This includes accessing regions classified as valid by ML analysis, targeting memory boundaries for potential bypass mechanisms, and attempting to read beyond established boundaries to elicit unintended system behavior.

Inject: The attacker floods the PLC memory with arbitrary writes, aiming to induce misoperation or force the system into a halt state. Unlike ML-based manipulation attacks, injection does not require prior memory analysis and is applied indiscriminately, even to denied memory addresses.

Modification: This assumes a more sophisticated adversary capable of manipulating metadata values to cause critical failures or embed malicious code into executable areas. Unlike inject attacks, modification attacks are selectively executed on memory regions that were previously accepted during access attempts, ensuring precise exploitation.

By analyzing PLC responses to targeted memory exploitation, we establish a structured attack framework that not only identifies exploitable memory regions but also categorizes system behavior under adversarial conditions. This taxonomy of responses enables a refined offensive strategy, bridging the gap between reconnaissance and effective exploitation in ICS.

4 PLC Memory and Targets

As presented in Figure 2, the workstation and PLC interact through engineering software designed to manage various

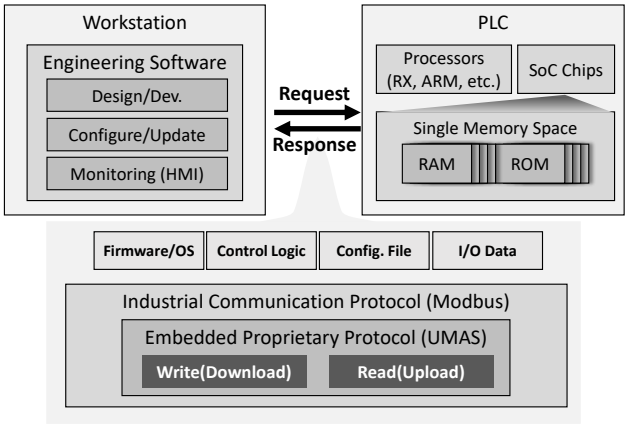

Figure 2: Workstation and PLC Communication

control functions. The engineering software on the workstation acts as a bridge for legitimate control functions. However, its reliance on proprietary industrial protocols introduces security weaknesses, as attackers can repurpose these mechanisms to manipulate PLC memory outside of normal constraints.

In this example, a Schneider Electric M221 PLC is depicted, where the workstation and PLC communicate via industrial communication protocols. These protocols facilitate the transfer of control data, ensuring the correct operation of the PLC in managing critical industrial processes. While Schneider Electric utilizes its proprietary protocol (UMAS) for communication, this is not unique to Schneider [14, 17, 18]. Other manufacturers, such as Allen-Bradley, use proprietary protocols like PCCC [19], which are not publicly disclosed but are designed for operational convenience.

The request/response communication structure shown in the center also applies to the embedded UMAS protocol, which enables the download and upload capabilities of the PLC. Using UMAS Write commands, direct access to the PLC's memory space is granted, enabling the download of control logic to the PLC. Typically, PLCs have two operational modes: Program mode, where the physical process is halted to allow new program downloads, and Run mode, where the process is active and downloads are restricted. While official engineering tools restrict write operations during Run mode, protocol-level write access remains available. This inconsistency provides an attack vector for injecting unauthorized control logic while avoiding standard security controls.

Like modern mobile devices, the PLC integrates a SoC architecture, consolidating various processing components. Unlike traditional computing systems, PLC memory lacks clear separations between execution-critical and temporary data, allowing an attacker to overwrite essential functions without triggering conventional integrity checks.

Our proposed technique aims to distinguish **critical data** from the multitude of PLC data. We categorize all data types present in PLC memory into four classifications: **M**etadata,

Code Data, **P**adding, **E**t cetera. Among these, **critical data** includes the **M** and **C** categories, while **P** and **E** are treated as general data with lower priority.

Metadata (M): Metadata structures define how PLC memory is organized, including file tables, execution references, and pointers. Pointers are a prime attack vector, as they control memory navigation. Manipulating metadata enables execution hijacking, stealthy persistence, or privilege escalation. Since metadata lacks universal standards, attackers must infer structures dynamically, increasing the complexity of precise targeting.

Code Data (C): Code data governs PLC behavior, including compiled control logic, firmware, and raw opcodes. As PLCs lack modern memory protections, modifying or injecting code enables direct manipulation of physical processes. With no runtime integrity checks, these modifications persist until manually overridden, making code memory a high-value attack target.

Padding (P): Padding consists of repetitive filler values (e.g., `0x00`, `0xFF`) used for alignment or reserved space. Attackers can abuse padding to hide payloads, evade forensic detection, or induce unintended memory behaviors. As padding regions may overlap with execution-critical data, distinguishing benign padding from exploitable space is a key challenge.

Et Cetera (E): This category includes miscellaneous runtime data, temporary buffers, and logging artifacts. While not directly critical, these regions may expose execution traces, aiding attack planning. Overlooked runtime variables could also introduce unexpected entry points for indirect exploitation [6, 20].

5 PLC Memory Features

Figure 3 outlines the process from raw memory acquisition to the identification of potential exploitable regions, addressing key challenges (*C1-C5*) through feature extraction and region aggregation.

C1 PLCs integrate multiple chips within a single address space, lacking a standard mapping scheme. This requires dynamic inference to differentiate memory regions.

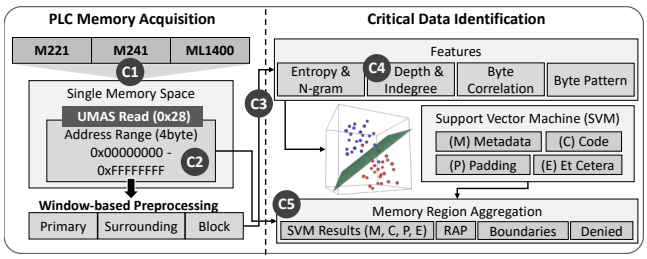

Figure 3: PLC Memory Analysis Pipeline

C2 With no standardized memory layout, we rely on protocol-defined address fields to establish an initial reference for segmentation.

C3 Since PLCs use direct memory references, any 4-byte value may function as a pointer. Our features identify and differentiate valid references.

C4 Execution flow is metadata-driven, but its location varies. Depth and Indegree features help trace execution flow, while others detect binary code regions.

C5 RAP regions share identical content, but only one is referenced. Depth and Indegree distinguish valid references, and Region Aggregation consolidates metadata to reconstruct coherent memory structures.

By addressing these challenges, our method transforms raw memory into structured insights, enabling precise exploitation of PLC memory regions.

We designed our feature selection process to focus on patterns emerging from differences between adjacent values, similar to how CNN kernels detect spatial relationships in images [21]. While instruction sets may differ across PLCs, the structural organization of 4-byte fields, such as registers and commands, remains consistent in terms of positional alignment. By leveraging this property, our features capture underlying relationships in memory, facilitating the identification of critical regions. This approach enables the extraction of metadata and structured data features using various infometrics [18], distinguishing control logic from non-critical data structures. These features play a key role in overcoming the challenges discussed earlier, providing a robust foundation for memory classification and exploitation. A detailed breakdown of each feature is provided in the following sections.

5.1 Windows

Windows are crucial elements in dealing with byte-level binary data. Our proposed method also extracts features and performs classification on a byte-level basis, so we have defined the following three different windows as described in Figure 4:

Primary window is set to 4 bytes and determines the size of the data subject to feature extraction. The reason for setting it to 4 bytes is that, in all known PLC memory forensic studies, without exception, any discovered pointer value was 4 bytes in size [11, 15, 18, 22–24]. This is because even though PLCs manage SoC memory within a single range, the total size does not exceed 4GB. However, extracting features from 4-byte chunks of data to represent a region is highly challenging. Therefore, we have defined a second window.

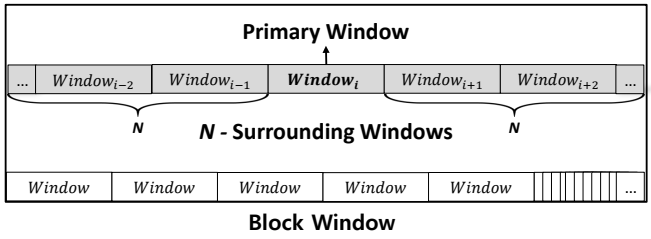

Figure 4: Feature-based Comparative Analysis on Different Architectures and Regions

Surrounding Window is currently set to 4bytes, this window size must be a multiple of the Primary Window size. It is used to compare the data subject to feature extraction with the preceding and following data. For instance, byte correlation involves calculating the features of the current data along with the preceding and following bytes. This significantly impacts the total computation time, and feature extraction can take over 10 hours for 1GB of memory depending on computational power.

Block Window assumes the size of a data block, also known as a data chunk. Data may not start from the first byte, and there might be bitmaps, padding, file types, sizes, magic numbers, etc. before valid values appear. This window size is used to search the subsequent area to verify the validity of data such as pointers. For example, when checking if a pointer is valid, the system not only checks the data value but also scans the following area based on this window size.

5.2 Entropy

Entropy is a measure of randomness or unpredictability in a dataset, and it is widely used in various fields, including digital forensics, to identify regions of interest within binary data. In the context of PLC memory analysis, entropy can help distinguish between structured data, such as control logic or other critical data, and unstructured or less significant data [25, 26].

To calculate this feature, we analyze the frequency distribution of byte values within a block window. First, we count the occurrences of each byte value in the primary window. Then, we calculate the probability of each byte value by dividing its frequency by the total number of bytes in the block. Using these probabilities, we compute the entropy, which quantifies the level of randomness in the data segment as:

$$H(W_i) = -\sum_{x=0}^{255} P(x) log_2 P(x) \quad (1)$$

5.3 N-gram

Applying N-gram to byte data is similar to measuring byte frequency, with the key difference being the measurement of the frequency of specific combinations of bytes of size N to achieve more precise targeting. When certain combinations hold significant meaning for PLCs, this feature can become a sniper rifle. If N-gram is applied based on a database constructed through reverse engineering, it becomes a highly precise feature for identifying code data.

However, we did not use a database obtained from primal research [18]. Instead, we leveraged the distribution obtained from entropy probability calculations. This approach ensures that the feature remains effective for unknown PLCs in the future, as it avoids dependency on specific instruction sets [27, 28]. By using entropy distributions, our N-gram feature can dynamically adapt to various PLC environments, making it a robust tool for forensic analysis without relying on predefined datasets.

5.4 Depth and Indegree

Depth and indegree are features derived from data suspected to be pointers. According to Challenge 3, if the memory space range is 4 bytes, and the memory space size is 4GB, every 4-byte value could potentially be a pointer. During the feature extraction process, if the data is not identified as padding, we first calculate the depth and indegree.

Depth measures the levels a data point references into the memory, essentially how deep the pointer chain goes.

Indegree measures how many times a specific data point, which is the data from the primary window, is referenced by other data points, indicating the level of referred.

For example, consider an offset o with a value v defined as a data point $D_n : o(v)$. Let D_0 be the data point at offset 0x1FED4 containing the value 0x15650 is expressed as 0x1FED4 $D_0 : 0x1FED4(0x15650)$. The depth and indegree tracking would proceed as follows:

$[Depth, Indegree]D_n$: Offset o (Value v)
$[3,0]D_0$: 0x1FED4 (0x15650)
$[2,1 \leq]D_1$: 0x15650 (0x7044F6C)
$[1,1 \leq]D_2$: 0x7044F6C (0xAAAA)
$[0,1 \leq]D_3$: 0xAAAA (0x0000)

In this chain of data points, the depth from D_0 to D_3 starts at three levels deep and decreases to one level. For indegree, the minimum value is represented as $1 \leq$. This is because each identified data point has at least one reference, but as the window explores the memory space, additional references from other metadata can be discovered. By evaluating these features, we can better understand the structure and relevance of different data points within the PLC memory space, helping to identify critical data and the overall RAP structure.

5.5 Byte Correlation

Byte correlation can be categorized into two primary types, measured by window size, which assesses the byte distance

or variation between successive prior and subsequent byte streams. The first feature measures the byte stream distance among data units, while the second feature measures quantifying the distance between extracted info-metrics.

Due to the inherent nature of metadata having a predefined order and specification of information, the values measured through byte correlation tend to be constant. For example, inode, prominent metadata of Ext4, stores four timestamps—access, modification, creation, and deletion—in a continuous sequence for file management [29]. Dewald et al. [30] utilized the fixed relationships between these timestamps, each having distinct meanings and timings, to propose a technique for extracting metadata.

As exemplified earlier, this feature is employed to differentiate metadata within consecutive byte streams. It identifies specific correlations such as sequential, dependent, or proportional relationships between byte streams. These correlations are utilized to distinguish metadata, reflecting their distinct characteristics as:

$$C(W_i) = \sum_{j=1}^{S} \sum_{k=1}^{N} (|b_j^i - b_j^{i-k}| \bullet |H((W_i)) - H(W_{i-k})| + |b_j^i - b_j^{i+k}| \bullet |H((W_i)) - H(W_{i+k})|) \quad (2)$$

Where:

- k denotes the position of the surrounding windows.
- $|b_j^i - b_j^{i-k}|$ captures the difference between corresponding byte positions.
- $|H((W_i)) - H(W_{i-k})|$ represents the difference in entropy values.

5.6 Byte Pattern

This feature compares the bit similarity between the surrounding window and the primary window, reflecting a localized byte frequency. Due to the multiple intermingled chips specified in Challenge 1, it is generally difficult to identify patterns using a byte frequency approach typically applied to single files. However, it is still necessary to detect recurring patterns in environments where specific instruction (function) codes, such as metadata and ICS payloads, are used. Therefore, we compare the bit similarity between windows as a measure of localized byte frequency and record it as a byte pattern feature for the data corresponding to the primary window as:

$$B(W_i) = \sum_{j=1}^{S} \sum_{k=1}^{N} (popcnt(b_j^i \oplus b_j^{i-k}) + popcnt(b_j^i \oplus b_j^{i+k})) \quad (3)$$

Where:

- k denotes the position of the surrounding windows.
- $\oplus$ represents the XOR operation.
- $popcnt(x)$ counts the number of set bits in the binary representation of x.

6 Experimental Setup and Attack Execution

This section outlines the experimental setup, memory acquisition process, machine learning-based classification, and the execution of targeted memory attacks. As summarized in Table 1, our experiments involved three PLCs: M221, M241, and ML1400. These PLCs share both commonalities and key differences. M221 and M241 belong to the same vendor family (Schneider Electric Modicon series), while M241 and AB1756 share the ARM-based architecture. Additionally, each PLC requires a different engineering software suite for control logic deployment, resulting in significant variations in memory layout even between closely related models like M221 and M241.

6.1 Memory Access Profiling and Acquisition

As shown in the *Access* column in Table 1, memory access profiling and acquisition were conducted using different methods depending on the PLC model. The objective was twofold: first, to assess access permissions by probing memory regions using protocol-based read operations; second, to retrieve accessible memory contents for further analysis and attack execution.

For Schneider Electric PLCs (M221, M241), the proprietary UMAS protocol was utilized, specifically leveraging the Read function (`0x28`). The retrievable memory range varied across models due to differences in memory segmentation and access enforcement. In contrast, the Allen-Bradley AB1756 does not support direct address-based memory access via the PCCC protocol. Instead, memory was extracted via a JTAG interface, revealing unique characteristics such as Redundant Address Pin (RAP) regions, which impact attack feasibility.

The access constraints and retrieval findings for each PLC are detailed below:

M221 : The protocol permitted memory retrieval up to `0xFFFFFFFF` (4GB), the maximum addressable range.

M241 : While M241's accessible address space extended up to `0x4FFDD000` (1.24GB), the effective retrievable memory was limited to 64MB (`0x4000000`). Two key factors contributed to this conclusion:

Beyond the 64MB region, all accessible addresses returned `0x00` values, leading to their classification as padding. While some addresses beyond this range exhibited successful read access, others were denied, but

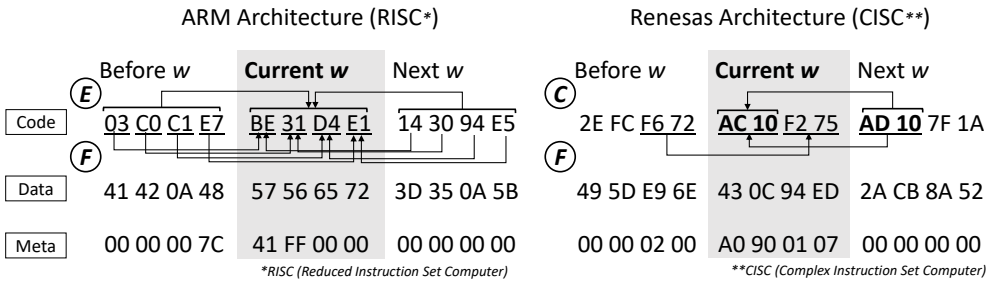

Figure 5: Feature-based Comparative Analysis on Different Architectures and Regions

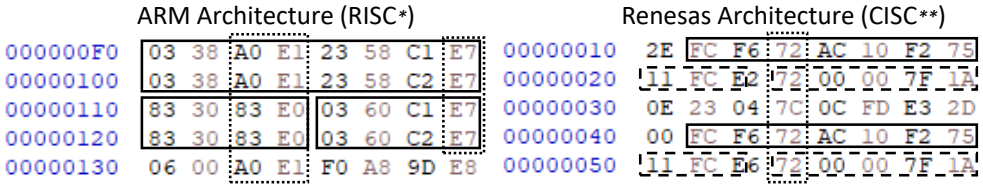

Figure 6: An example of N-gram detected binary

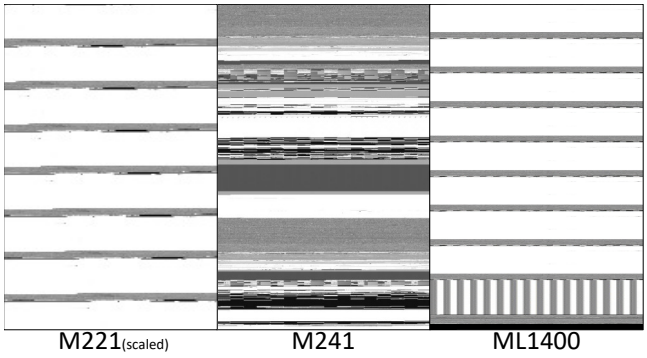

Figure 7: Visualization of Memory Dense of Three PLCs

since these regions did not contain meaningful memory content, they were excluded from our automated analysis and attack scope.

According to Schneider Electric's M241 Logic Controllers Programming Guide, the official memory organization document explicitly states that the system and user memory areas together comprise 64MB of RAM [31].

ML1400 : Unlike M221 and M241, Allen-Bradley AB1756 does not permit conventional memory acquisition via its proprietary PCCC protocol, as it structures memory into predefined "items" rather than allowing direct address-based access. Instead, we acquired ML1400's memory via JTAG debugging.

Figure 7 reveals that AB1756 exhibits large RAP (Redundant Address Pin) regions, each with a size of `0x100000` (16MB). Machine learning analysis determined that only the first RAP region contains real memory, while the remaining regions result from memory mirroring effects due to the RAP phenomenon.

This observation aligns with hardware-level analysis of the AB1756 board's address pins, which identified "Don't Care" bits ($A_{24} - A_{26}$) [15]. These bits indicate that memory addressing beyond a certain range is undefined, reinforcing the conclusion that only 16MB of ML1400's memory is physically valid while the remaining RAP regions (152 MB) are just a replica of address pin redundancy.

The memory acquisition results in Figure 7 reveal RAP structures across M221, M241, and ML1400. In M221, RAP regions are magnified (as indicated in the figure) due to large empty spaces, emphasizing data density. The repeating regions indicate RAP areas, where read operations return identical data, but writes affect all linked regions simultaneously, allowing the RAP structure to be inferred based on modified address ranges and sizes. Localized RAP regions in M221 and M241 suggest redundancy in lower-order address pins, whereas ML1400's widespread RAP effects indicate redundancy in higher-order pins.

6.2 SVM-Based Target Identification

The memory contents of each PLC were analyzed using SVM-based classification, identifying critical memory regions, including metadata (M) and executable code segments (C). As summarized in Table 2 under the *Memory Distribution* row, the classification results revealed a predominant presence of padding regions across all three PLCs. Specifically, regions filled with `0x00`, `0xFF`, `0x55` and `0xEE` constituted the majority of memory contents.

As depicted in Figure 7, M221's excessive padding obscured meaningful data structures, making direct visualiza-

PLC Model	Vendor	Series	CPU Architecture	Acquisition	Attack	Access	Detect
M221	Schneider Electric	Modicon	Renesas	Yes	Yes	UMAS Protocol	Yes
M241	Schneider Electric	Modicon	ARM	Yes	Yes	UMAS Protocol	Yes
AB1756	Allen-Bradely	ControlLogix	ARM	Yes	No	JTAG	Yes

Table 1: PLC Experiment Summary

tion ineffective. Thus, to facilitate meaningful memory density comparison across PLCs, the figure focuses on an expanded view of the regions where executable content is concentrated—specifically, the RAP (Redundant Address Pin) regions.

The training dataset consisted of approximately 60 control logic programs [32, 33] as *Code*. For every control logic sample, three associated *metadata* datasets—Configuration 1, Configuration 2 [11], and undefined metadata(around `0x200-0x500`)—were included, totaling around 180 instances. Additionally, the dataset incorporated over 60 control logic project files compressed in 7z format, classified as *Etc* data, ensuring a diverse representation of structured and unstructured memory content. This comprehensive training set reinforced the classifier's ability to distinguish metadata, code, and auxiliary data within the memory space, improving the robustness of our segmentation and attack planning.

Since SVM classification operates on a per-window basis, post-processing is required to refine the results. Each classified window belongs to a larger memory segment, and the segmentation of these windows into memory chunks is determined based on the continuity of critical data regions (M, C). The proportion of metadata (M) or code (C) within a given chunk directly influences its classification as a distinct memory region.

This segmentation process is integral to attack execution. As shown in Figure 8, we aggregating critical memory region boundaries, we establish attack priorities, guiding subsequent write-based (0x29) memory manipulations. These targeted attacks are informed by SVM-derived classifications, ensuring that critical areas—such as metadata structures and executable control logic—are prioritized for exploitation.

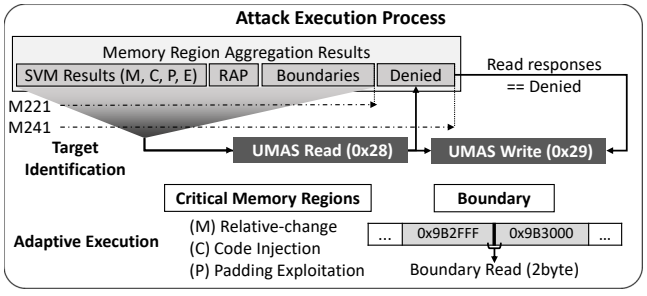

Figure 8: PLC Memory Attack Strategy

6.3 Attack Execution Strategy

As shown in Figure 8, the attack execution targeted PLCs using the UMAS protocol. Consequently, AB1756 was excluded from active attack attempts due to its PCCC-based memory management, which does not permit direct memory reads/writes in the same manner. For M221 and M241, the attack leveraged UMAS Read (`0x28`) and Write (`0x29`) functions. Read was primarily used to evaluate access permissions across memory regions, while Write was utilized for direct memory manipulations.

The attacks were designed based on the ML-driven target identification and memory region aggregation results. The classified memory types—metadata (*M*), code (*C*), and padding (*P*) provided a structured attack surface, allowing for targeted exploitation strategies. Attack vectors were categorized into five key approaches, focusing on memory corruption, boundary analysis, and privilege escalation.

(M) Relative-Change Attack Metadata regions were targeted for modification attack due to their high sensitivity to minor alterations. Since metadata structures often manage execution flow, even a ± 1 modification could result in execution failures. Given that memory management systems typically operate on 4-byte units, shifting the pointer value by a single byte in a metadata structure could cause severe disruptions, leading to immediate PLC halts.

(C) Code Injection Identified code regions were subjected to data modification by surrounding code data. To avoid immediate detection, the attack leveraged local data duplication—copying adjacent instructions or known executable structures and inserting them into the code region. This was designed to induce either execution failures (halt) or subtle operational anomalies, classified as compromise (*Cmpr*), where the PLC continues to operate but exhibits unexpected behavior.

(P) Padding Exploitation Padding regions were analyzed for their potential use in stealthy attacks. Unlike metadata and code, padding spans vast memory areas with low entropy, making it an ideal space for covert payload storage. The attack involved modifying dispersed sections of the padding region in a block-window approach and subsequently monitoring if the data was altered by the PLC within a short timeframe. This test determined whether padding regions were actively overwritten, allowing an attacker to assess their viability for storing malicious payloads persistently.

Write-Based Access Violation For cases where read (`0x28`) access attempts were denied but the PLC remained operational, write (`0x29`) functions were used to probe further. If a memory region rejected read attempts but still accepted write operations, it indicated a misconfigured access control mechanism. This approach aimed to escalate memory access privileges by leveraging potential discrepancies between read and write permissions.

Boundary Read Probing This attack focused on accessing the precise boundaries between classified memory regions. Memory segmentation inconsistencies could allow boundary-crossing requests to trigger unintended behavior. Using 2-byte precision, access attempts were made at segment edges to detect potential privilege escalations or operational disruptions caused by ambiguous memory segmentation policies.

6.4 PLC Response and Impact

The expected PLC's responses to memory attacks are categorized into four distinct outcomes:

Accept : The request is successfully processed, and the PLC returns a valid response.

Deny : The request is rejected, and the PLC responds with an error code via the protocol.

Halt : The PLC ceases operation and fails to respond, indicating a system crash or critical failure.

Compromise : The PLC enters an unstable state, potentially exhibiting unintended behavior between an accept and halt response. Unlike *halt*, compromise does not necessarily result in an immediate system stop, and unlike *accept*, it may cause unintended state changes. This reaction is only classified as compromise if the attack targets memory regions that are actively used by the PLC, such as overwritten metadata or control logic execution. If the affected memory is not proven to be actively used, the response is categorized as either *accept* or *halt* based on the observed system behavior.

By systematically executing and analyzing these attacks, we identified architectural weaknesses across different PLCs, highlighting inconsistencies in their memory protection mechanisms.

7 Experimental Results and Evaluation

This section evaluates various aspects of the proposed methods. First, we assess the feature selection process to ensure the extracted features accurately distinguish critical memory regions while maintaining independence for machine learning applications. Next, we verify the SVM classification results, demonstrating that the model generalizes across different PLC architectures despite being trained on a single device. Finally, we analyze memory attack outcomes, highlighting the PLCs' responses to targeted read and write operations, revealing vulnerabilities in access control mechanisms.

7.1 Feature Evaluation

To evaluate the effectiveness of the selected features, this section conducted a comparative analysis across different PLC architectures and a statistical evaluation of feature independence using Pearson correlation.

Feature-Based Comparative Analysis

Figure 5 illustrates a comparative analysis of memory content between two PLCs that employ entirely different processors, examined from a feature perspective. This comparison demonstrates how features operate in practical memory examples and evaluates their effectiveness in capturing meaningful patterns and relationships.

As introduced in Section 5, the features are represented in the figure by circled letters, with *C. N-gram*, *E. Byte Correlation*, and *F. Byte Pattern* having been defined. However, *B. Entropy*, which intuitively reflects complexity, and *D. Depth and Indegree*, which are not extracted from relationships between windows, are not addressed in this section.

In an ARM(Advanced RISC Machine) architecture, shown on the left side of Figure 5, little-endian formatting results in the most significant byte (MSB) of a 4-byte sequence corresponding to the instruction. This leads to a distinct clustering of values in the binary code region, a phenomenon independent of processor type. Similarly, Renesas architecture demonstrates analogous clustering. While the exact values may vary depending on the instruction set, the assignment of unique identifiers, such as register numbers, within PLCs perpetuates value repetition. Even when transitions to new windows with higher MSB values occur, the remaining three bytes often exhibit minimal variation from preceding windows. The main reason for this assumption is that a 4-byte window aligns with the instruction sets of all PLC processor architectures and lies in the fact that most modern processors are based on 16-bit (2-byte) or 32-bit (4-byte) systems. Since these architectures operate on multiples of these sizes, a window size that is a multiple of these base units inherently ensures the utility of this feature.

The *C. N-gram* feature, while often yielding zero values more frequently than other features, delivers a powerful effect. In binaries where data is arranged according to specific logic, such as code regions, the repeated use of the same variable is quite common. For instance, assigning a value to a specific register and performing operations often results in placing the same value at the same location. Figure 5 represents this phenomenon as bold within the CISC architecture.

Figure 6 provides a more detailed view of cases where the *C.*

	M221				M241				*ML1400			
	Meta	Code	Padding	ETC	Meta	Code	Padding	ETC	Meta	Code	Padding	ETC
Memory Distribution (†W)	4697	5559	(99.88%)	(0.12%)	7522	28770	(77.31%)	(21.83%)	669	2852	(84.49%)	(15.45%)

*this region is estimated based on the first detected RAP region, which has the highest concentration according to SVM results.
†W: Count by number of windows.

Table 2: Summary of Memory Classifications Across PLCs

N-gram feature is frequently observed. Programs based on the ARM architecture, as shown on the left, tend to exhibit highly repetitive structures, which is why N-grams are commonly used in binary detection [9, 18, 34]. On the right, the Renesas architecture, based on CISC, presents a more complex binary structure, but it is not so different that N-grams fail to be detected altogether.

The *E. Byte Correlation* feature, computed across 4-byte windows (w), effectively identifies sustained ranges of values within the RISC-based ARM architecture. Despite differences in absolute values between ARM and Renesas binaries, the feature highlights a shared tendency of localized range clustering, enabling comparative analysis. Conversely, in the upper-right region of the CISC-based Renesas architecture, the three w do not exhibit similar rightmost byte values. While the before w and current w share similar values, the next w deviates significantly, making it challenging to extract distinguishing patterns for code detection using Byte Correlation alone. However, this limitation is mitigated by complementary features.

The *F. Byte Pattern* feature, in contrast, provides finer granularity by identifying byte-level patterns within each window. This facilitates pinpointing cases where instruction sets and associated parameters occupy the same positions across binaries. Such a granular analysis is essential for identifying recurring instruction patterns and associated metadata within the PLC memory.

Integration with Byte Correlation and Byte Pattern Features The features defined earlier—Byte Correlation and Byte Pattern—play pivotal roles in this evaluation. Byte Correlation reveals proportional or sequential relationships in binary streams, while Byte Pattern captures localized byte frequency dynamics. Together, these features facilitate comprehensive comparisons across processors, transcending architectural variations to highlight fundamental patterns in PLC memory.

Pearson Correlation Analysis of Features

The Pearson correlation analysis shown in Fig. 9 highlights important insights into the relationships among the features.

The Pearson correlation coefficient r_{xy}, calculated by the formula below, measures the linear relationship between two variables x and x, where x_i and y_i represent the individual values of two different features, and $\bar{x}$ and $\bar{y}$ are their respective means. This formula helps determine the degree to which the features vary together.

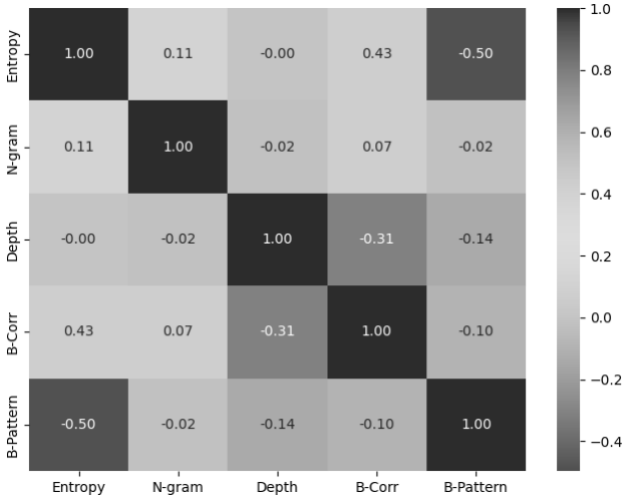

Figure 9: Pearson Correlation Between Features

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4}$$

Infometrics, which combines Entropy and N-gram, shows a positive correlation with both, though the correlation was relatively weak (0.11 with N-gram and 0.43 with Byte-Correlation). Despite the modest correlation, both Entropy and N-gram were deemed effective features to include in the analysis. The evaluation using Pearson correlation further reveals that N-gram is a highly independent feature, as indicated by its low correlations with the other features. This suggests that in PLC memory regions, where repetitive byte values are prevalent—such as padding or non-critical areas—N-gram remains unaffected by other features, reinforcing its independence.

Since Byte-Pattern measures the repetition of bit similarity across the data, it is closely related to the byte values themselves. As a result, it shows a negative correlation with Infometrics (-0.5), yet still maintains sufficient independence to be valuable in the analysis. Its correlation with Entropy indicates that regions with higher value (as measured by Entropy) tend to show more repetition in byte patterns, aligning with the expected relationship between randomness and pattern distribution. However, the relatively low correlation values suggest that Byte-Pattern and the other features provide complementary information rather than redundant insights. As a result, evaluating the degree of correlation among the features using Pearson correlation, we can conclude that all features were qualified to be used in the machine learning model.

7.2 SVM Results Verification

The memory contents of each PLC were analyzed using SVM-based classification, identifying critical memory regions, including metadata and executable code segments. The *Detect* column in Table 1 indicates whether these classified regions were validated through external tools such as binwalk [35]. Successful detection confirms that our classification aligns with actual binary structures, ensuring that code regions identified via machine learning genuinely contain firmware, compiled logic, or executable instructions rather than random or misclassified data. This classification and verification process is crucial for attack execution. By accurately mapping executable memory regions, targeted attacks can be designed to manipulate control logic, induce malfunctions, or trigger unintended PLC states.

Furthermore, Table 3 includes a *Ground Truth* column, which represents verified data obtained through manual forensic analysis or officially documented sources, such as vendor memory layouts. These ground truth values serve as a baseline for evaluating the correctness of our SVM classifications and attack outcomes, ensuring that identified memory regions align with the actual structure and behavior of each PLC.

Notably, padding occupied a substantial portion of the total memory across all PLCs, particularly in the M221, where 99.8% of the acquired memory space was classified as padding. For this reason, Figure 7 highlights the memory-dense RAP region, allowing a comparative analysis of structural similarities across different PLCs.

7.3 Memory Attack Outcomes

Table 3 details the PLC responses to read (`0x28`) and write (`0x29`) operations, highlighting vulnerabilities in different memory classifications.

The effectiveness of the proposed attack methodology was evaluated by executing read (`0x28`) and write (`0x29`) operations against classified memory regions. Table 3 details the observed responses, illustrating how different PLC architectures handle unauthorized memory manipulations.

For the M221, memory writes targeting metadata regions such as *JumpTable* and *Configuration (Conf)* consistently resulted in PLC halts, confirming that even minor perturbations in metadata structures induce execution failures. Similarly, control logic modifications led to compromised states, where the PLC continued operating but exhibited unintended behavior. RAP regions were successfully manipulated, demonstrating that redundant address mappings enable indirect yet persistent alterations to memory.

In contrast, M241 exhibited more restrictive access controls, with large portions of memory returning deny or halt responses. Notably, a segment spanning from `0xC94000` to `0x4000000` contained 284 distinct `0x1000`-sized blocks that, upon write attempts, triggered immediate PLC halts. Despite these restrictions, a 2.9MB region at `0x9B6000` permitted modifications that induced compromise states, indicating partial execution control. These areas and PLC patterns, which are also listed in Appendix A, are in the process of submitting a CVE vulnerability report.

The ML1400, due to its proprietary PCCC-based memory structure, did not permit protocol-based read or write operations. However, JTAG memory acquisition confirmed the presence of RAP effects, with only the first 16MB of mirrored memory containing unique data. This structural redundancy aligns with previous hardware-level analyses of address pin behaviors, further supporting the feasibility of leveraging RAP regions for stealthy attacks.

Ground Truth	Range (hex)	Size	%	Read	Write	Predicted
		M221				
JumpTable	8000 - C90B			Success	Halt	22.0% (M)
	F9B0-FEC3			Success	Halt	15.9% (M)
Conf1	1FF3A	1FFFF		Success	Halt	14.3% (M)
Conf2	7044F6C - 70567FF			Success	Halt	9.8% (M)
RAP	70C4F6C - 70D67FF			Success	Halt	9.8% (M)
RAP	7144F6C - 71567FF			Success	Halt	9.8% (M)
RAP	71CCF6C - 71D67FF			Success	Halt	9.8% (M)
Control Logic	701E0A4-701E337			Success	Compromise	86.7% (C)
		M241				
	0- 660000	6.4 MB	10%	Success	Deny	0.68% (C)
ARM binary	660000- 9B3000	3.3 MB	5.1%	Success	Success	0.85% (C)
	9B3000- 9B4000	4 KB	-	Deny	Halt	-
	9B4000- 9B5000	4 KB	-	Success	Success	
	9B5000- 9B6000	4 KB	-	Deny	Halt	-
	9B6000- C94000	2.9 MB	4.5%	Success	Compromise	P
	*C94000- 4000000	51.4 MB	80%	284 Deny & Halt		
	4000000- D760000					-
	D760000-4FFDD000			Success		-
	4FFDD000-FFFFFFFF			Halt		

*This region is omitted for brevity, more details in appendix A.

Table 3: Observed PLC Responses to Memory Attacks

8 Related Works

PLCs have received growing attention from the security community due to their non-standardized architecture and memory layouts. Prior works have explored PLC vulnerabilities by reverse engineering control logic [9], manipulating runtime state [10], and analyzing authentication protocols [14]. However, few studies have investigated PLC memory at a structural level using machine learning.

Memory acquisition techniques for PLCs have traditionally relied on JTAG access [15], firmware extraction [23], or protocol-based memory access [11,22]. While these methods have enabled static analysis, they do not offer automated classification or fine-grained profiling of memory semantics. Recent work such as [22] introduced a generic memory forensics framework for PLCs, but lacked the use of learned patterns or behavior-based evaluation.

Our work draws on and extends these efforts by introducing a machine-learning-driven approach to classify memory regions across heterogeneous PLC platforms. During this process, we discovered that writes to certain addresses also affected disjoint memory regions—an instance of potentially exploitable memory aliasing [36], later formalized as Redundant Address Pin (RAP). While RAP was first named and exploited in our prior work [9], its initial discovery originated

from the experiments presented here.

Beyond PLC-specific, our classification framework shares conceptual similarities with prior research in binary similarity and architectural inference. Techniques such as semantics-aware function recognition [37], binary code similarity learning [38], and *discovRE* [39] share conceptual ground for memory classification, which abstracts low-level binary differences across architecture to detect functional similarity. While [40] provided an early attempt to categorize and implement memory attacks on PLCs, highlighting foundational risks in memory management. *cpu_rec.py* [41] leverages n-gram features to infer CPU architectures, similar in spirit to our use of statistical patterns for memory domain inference.

To our knowledge, this is the first work to apply machine learning for fine-grained memory classification across PLCs in a security context, while also validating its results through empirical side effects such as PLC halts.

9 Discussion

Our approach adopts a **4-byte window** as the default size for analyzing memory structures, primarily because pointer values are conventionally stored in 4-byte segments. This decision is not solely based on our manual forensic analysis of the M221 PLC but is a well-established convention in memory management systems with constrained storage capacities, such as legacy filesystems. Moreover, the UMAS protocol's address field is structured as 4 bytes, reinforcing the natural alignment of this choice.

However, if the 4-byte window presents a limitation in specific attack scenarios—such as when targeting a PLC with a proprietary protocol using different field sizes—our method allows for dynamic adjustments. The window size can be freely modified by the proprietary protocol specifications, making it a matter of time rather than a fundamental constraint.

Although our features have been extensively evaluated through multiple methodologies, including comparative analysis and ML-driven validation, they may still be perceived as constrained in scalability. A comparable historical reference is the evolution of filesystems—early versions like Ext2 had metadata structures similar to modern PLC memory, relying on direct pointers [42]. However, with the introduction of Ext3, indirect pointers enabled more flexible memory management, and Ext4 adopted extents for handling large-scale files [29].

Despite these advancements in general-purpose storage, PLC memory remains fundamentally different. PLCs are designed as hard real-time systems, where every aspect—from RTOS optimizations to execution timing—is tuned for industrial process stability. Unlike general computing environments that necessitate scalable storage, PLCs have no requirement (or feasibility) for large-scale memory expansion due to strict real-time constraints. Consequently, our method remains highly effective for current and future PLC architectures, as their fundamental memory limitations will persist, ensuring long-term applicability of our exploitation framework.

All Halt and Compromise (*Cmpr*) occurrences recorded in Table 2 and Table 3 represent the minimum observed instances. These values are based on actual experimentation with PLC memory responses, but conducting exhaustive attack trials across all possible cases was infeasible due to the time-consuming nature of rebooting and reconfiguring the PLC after each halt. As a proof of concept, the M221 and M241 PLCs exhibited over 500 unique instances of halts and compromises, demonstrating that our attack method is both practical and highly threatening. Notably, the machine learning-based memory targeting approach proved effective despite the absence of prior manual analysis—particularly for the M241, where even the CPU architecture was not explicitly known. This approach has already been validated through disclosed and ongoing CVEs, with recommendations announced in March 2025 and reported CVSS scores exceeding 9 [43]. Moreover, our results confirm that even a basic Read function is sufficient to induce PLC halts, underscoring the critical security risks posed by proposed methods.

10 Conclusion

This paper demonstrated a machine learning-driven PLC memory exploitation framework, leveraging proprietary protocol-based access to systematically analyze and target critical memory regions. Our approach successfully classified metadata, code, padding, and other data regions using an SVM model trained on a single PLC, yet generalized effectively across different architectures and vendors.

Through attack execution via the UMAS protocol, we systematically probed and manipulated PLC memory, resulting in deny, halts and compromises, across multiple PLCs. Notably, the Read function alone was sufficient to induce PLC failures, exposing fundamental weaknesses in memory access control. Our experiments further revealed the Redundant Address Pin (RAP) phenomenon exists among different vendor PLCs, which adversaries could exploit to perform stealthy and sophisticated attacks.

By evaluating PLC responses across different attack surfaces, we uncovered significant inconsistencies in memory protection mechanisms, highlighting the urgent need for enhanced security measures in industrial control systems. These findings have resulted in both disclosed and ongoing CVE reports, reinforcing the necessity for stricter access enforcement and runtime integrity verification [43].

Our work underscores the practicality and severity of memory-based PLC attacks, demonstrating that even basic read and write operations can subvert device integrity. Future research should explore adaptive memory defenses that dynamically mitigate such threats, ensuring the resilience of critical infrastructure against emerging adversarial techniques.

Acknowledgments

This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 17STCIN00001-05-00.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

References

[1] "Critical Infrastructure Sectors," https://www.cisa.gov/topics/critical-infrastructure-security-and-resilience/critical-infrastructure-sectors, 2024, [Online; accessed 29-June-2022].

[2] "Presidential Policy Directive – Critical Infrastructure Security and Resilience," https://obamawhitehouse.archives.gov/the-press-office/2013/02/12/presidential-\policy-directive-critical-infrastructure-security-and-resil, 2013, [Online; accessed 29-June-2022].

[3] "Cyber-Attack Against Ukrainian Critical Infrastructure," https://www.cisa.gov/news-events/ics-alerts/ir-alert-h-16-056-01, 2021, [Online; accessed 29-June-2022].

[4] "Havex Malware," https://www.cisa.gov/uscert/ics/advisories/ICSA-14-178-01, 2022, [Online; accessed 29-June-2022].

[5] R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, May 2011.

[6] "CrashOverride Malware," https://www.cisa.gov/uscert/ncas/alerts/TA17-163A, 2022, [Online; accessed 29-June-2022].

[7] R. Dudley and D. Golden, "The colonial pipeline ransomware hackers had a secret weapon: self-promoting cybersecurity firms," 2021.

[8] B. Johnson, D. Caban, M. Krotofil, D. Scali, N. Brubaker, and C. Glyer, "Attackers deploy new ics attack framework "triton" and cause operational disruption to critical infrastructure," https://www.mandiant.com/resources/blog/attackers-deploy-new-ics-attack-framework-triton, 2021, [Online; accessed 26-Dec-2022].

[9] A. Ayub, W. Jo, and I. Ahmed, "Charlie, charlie, charlie on industrial control systems: Plc control logic attacks by design, not by chance," 2024.

[10] A. Ayub, N. Zubair, H. Yoo, W. Jo, and I. Ahmed, "Gadgets of gadgets in industrial control systems: Return oriented programming attacks on plcs," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2023.

[11] N. Zubair, A. Ayub, H. Yoo, and I. Ahmed, "Pem: Remote forensic acquisition of plc memory in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 40, p. 301336, 2022.

[12] E. López-Morales, U. Planta, C. Rubio-Medrano, A. Abbasi, and A. A. Cardenas, "Sok: Security of programmable logic controllers," *arXiv preprint arXiv:2403.00280*, 2024.

[13] A. Ayub, W. Jo, S. A. Qasim, and I. Ahmed, "How are industrial control systems insecure by design? a deeper insight into real-world programmable logic controllers," *IEEE Security & Privacy*, vol. 21, no. 4, pp. 10–19, 2023.

[14] A. Ayub, H. Yoo, and I. Ahmed, "Empirical study of plc authentication protocols in industrial control systems," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 383–397.

[15] M. H. Rais, R. A. Awad, J. Lopez Jr, and I. Ahmed, "Jtag-based plc memory acquisition framework for industrial control systems," *Forensic Science International: Digital Investigation*, vol. 37, p. 301196, 2021.

[16] I. Ahmed, S. Obermeier, S. Sudhakaran, and V. Roussev, "Programmable logic controller forensics," *IEEE Security Privacy*, vol. 15, no. 6, pp. 18–24, November 2017.

[17] S. A. Qasim, W. Jo, and I. Ahmed, "Pree: Heuristic builder for reverse engineering of network protocols in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 45, p. 301565, 2023.

[18] H. Yoo, S. Kalle, J. Smith, and I. Ahmed, "Overshadow Plc to Detect Remote Control-Logic Injection Attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2019, pp. 109–132.

[19] S. Senthivel, I. Ahmed, and V. Roussev, "Scada network forensics of the pccc protocol," *Digital Investigation*, vol. 22, pp. S57–S65, 2017.

[20] Cybersecurity and I. S. A. (CISA), "Understanding and mitigating russian state-sponsored cyber threats to u.s. critical infrastructure," https://www.cisa.gov/uscert/ncas/alerts/aa22-011a, 2022, [Online; accessed 29-June-2022].

[21] X. Ding, X. Zhang, J. Han, and G. Ding, "Scaling up your kernels to 31x31: Revisiting large kernel design in cnns," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 963–11 975.

[22] R. A. Awad, M. H. Rais, M. Rogers, I. Ahmed, and V. Paquit, "Towards generic memory forensic framework for programmable logic controllers," *Forensic Science International: Digital Investigation*, vol. 44, p. 301513, 2023, selected papers of the Tenth Annual DFRWS EU Conference. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281723000148

[23] M. H. Rais, R. A. Awad, J. Lopez Jr, and I. Ahmed, "Memory forensic analysis of a programmable logic controller in industrial control systems," *Forensic Science International: Digital Investigation*, vol. 40, p. 301339, 2022.

[24] L. Zhang and X. Zhang, "Plcoroner: A novel jtag-based plc memory forensics method," in *2023 3rd International Conference on Electronic Information Engineering and Computer (EIECT)*. IEEE, 2023, pp. 338–342.

[25] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of iot malware and detection methods based on static features," *ICT express*, vol. 6, no. 4, pp. 280–286, 2020.

[26] H. Lee, S. Kim, D. Baek, D. Kim, and D. Hwang, "Robust iot malware detection and classification using opcode category features on machine learning," *IEEE Access*, vol. 11, pp. 18 855–18 867, 2023.

[27] S. Moon, Y. Kim, H. Lee, D. Kim, and D. Hwang, "Evolved iot malware detection using opcode category sequence through machine learning," in *2022 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2022, pp. 1–7.

[28] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun, and K.-K. R. Choo, "An opcode-based technique for polymorphic internet of things malware detection," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 6, p. e5173, 2020.

[29] "Ext4 timestamps," https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Extent_Tree, 2024, [Online; accessed 04-March-2025].

[30] A. Dewald and S. Seufert, "Afeic: Advanced forensic ext4 inode carving," *Digital investigation*, vol. 20, pp. S83–S91, 2017.

[31] https://product-help.schneider-electric.com/Machine%20Expert/V1.1/en/m241prg/m241prg/M2xx_-_Memory_Mapping/M2xx_-_Memory_Mapping-3.htm#XREF_D_SE_0002858_1, 2019, [Online; accessed 12-March-2025].

[32] H. Yoo and I. Ahmed, "Control Logic Injection Attacks on Industrial Control Systems," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2019, pp. 33–48.

[33] "Dfrws 2023 challenge - the troubled elevator," https://github.com/dfrws/dfrws2023-challenge, 2024, [Online; accessed 11-March-2025].

[34] D. Stabili, L. Ferretti, M. Andreolini, and M. Marchetti, "Daga: Detecting attacks to in-vehicle networks via n-gram analysis," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 11, pp. 11 540–11 554, 2022.

[35] ReFirmLabs, "Binwalk: Firmware analysis tool," https://github.com/ReFirmLabs/binwalk, 2024, [Online; accessed 10-March-2025].

[36] J. De Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede, and J. Van Bulck, "Badram: Practical memory aliasing attacks on trusted execution environments," in *46th IEEE Symposium on Security and Privacy*. IEEE, 2024.

[37] S. Wang, P. Wang, and D. Wu, "Semantics-aware machine learning for function recognition in binary code," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 388–398.

[38] D. Tian, X. Jia, R. Ma, S. Liu, W. Liu, and C. Hu, "Bindeep: A deep learning approach to binary code similarity detection," *Expert Systems with Applications*, vol. 168, p. 114348, 2021.

[39] S. Eschweiler, K. Yakdan, E. Gerhards-Padilla *et al.*, "Discovre: Efficient cross-architecture identification of bugs in binary code." in *Ndss*, vol. 52, 2016, pp. 58–79.

[40] Y. Geng, R. Ma, Q. Wei, and W. Wang, "Programmable logic controller memory management vulnerability analysis," in *Journal of Physics: Conference Series*, vol. 2414, no. 1. IOP Publishing, 2022, p. 012015.

[41] L. Granboulan, "cpu_rec.py, un outil statistique pour la reconnaissance d'architectures binaires exotiques," in *Symposium sur la Securité des Technologies de l'Information et des Communications (SSTIC)*, 2017. [Online]. Available: https://www.sstic.org/2017/presentation/cpu_rec/

[42] "Ext2-inode," https://wiki.osdev.org/Ext2#Inode_Data_Structure, 2025, [Online; Accessed 05-March-2025].

[43] S. Electric, "Cve-2024-11737: Schneider electric security notification," 2024, accessed: 2024-12-18. [Online]. Available: https://download.schneider-electric.com/files?p_Doc_Ref=SEVD-2024-345-03&p_enDocType=Security+and+Safety+Notice&p_File_Name=SEVD-2024-345-03.pdf

A Appendix

No.	Ranges			
001-004	9B3000-9B4000	9B5000-9B6000	C94000-C95000	C97000-C98000
005-008	C9A000-C9B000	C9C000-C9D000	CB5000-CB6000	CB8000-CB9000
009-012	CBB000-CBC000	CBE000-CBF000	CC1000-CC2000	CC4000-CC5000
013-016	CDD000-CDE000	CE0000-CE1000	CE3000-CE4000	CE6000-CE7000
017-020	CE9000-CEA000	CEC000-CED000	DF4000-DF5000	DF8000-DF9000
021-024	EFE000-EFF000	F00000-F01000	F14000-F15000	F17000-F18000
025-028	F4E000-F4F000	F51000-F52000	F66000-F67000	F69000-F6A000
029-032	F72000-F73000	F76000-F77000	F7A000-F7B000	F82000-F83000
033-036	FF6000-FF7000	FF9000-FFA000	1004000-1005000	1007000-1008000
037-040	1009000-100A000	101A000-101B000	101C000-101D000	102D000-102E000
041-044	102F000-1030000	1040000-1041000	1042000-1043000	1053000-1054000
045-048	1056000-1057000	105B000-105C000	105E000-105F000	1063000-1064000
049-052	1066000-1067000	1069000-106A000	106C000-106D000	106F000-1070000
053-056	1072000-1073000	1083000-1084000	13F1000-13F2000	13F7000-13F8000
057-060	13FA000-13FB000	1400000-1401000	1403000-1404000	1414000-1415000
061-064	1419000-141A000	141C000-141D000	141F000-1420000	1425000-1426000
065-068	1428000-1429000	142E000-142F000	1431000-1432000	1437000-1438000
069-072	143A000-143B000	1440000-1441000	1443000-1444000	1449000-144A000
073-076	144C000-144D000	1452000-1453000	1455000-1456000	145B000-145C000
077-080	145E000-145F000	1464000-1465000	1467000-1468000	146D000-146E000
081-084	1470000-1471000	1476000-1477000	1479000-147A000	147F000-1480000
085-088	1482000-1483000	1488000-1489000	148B000-148C000	1491000-1492000
089-092	14A3000-14A4000	14B4000-14B5000	14B6000-14B7000	14B9000-14BA000
093-096	14BB000-14BC000	14CC000-14CD000	14CE000-14CF000	14DF000-14E0000
097-100	14E1000-14E2000	14F2000-14F3000	14F5000-14F6000	1506000-1507000
101-104	1509000-150A000	150C000-150D000	156C000-156D000	156F000-1570000
105-108	1571000-1572000	1574000-1575000	1576000-1577000	1579000-157A000
109-112	157B000-157C000	157E000-157F000	1580000-1581000	1583000-1584000
113-116	1586000-1587000	1589000-158A000	158C000-158D000	158F000-1590000
117-120	1592000-1593000	1595000-1596000	1598000-1599000	159B000-159C000
121-124	159E000-159F000	15A1000-15A2000	15A4000-15A5000	15B5000-15B6000
125-128	15B8000-15B9000	15C9000-15CA000	15CB000-15CC000	15DC000-15DD000
129-132	15DE000-15DF000	15EF000-15F0000	15F1000-15F2000	1602000-1603000
133-136	1604000-1605000	1615000-1616000	16B9000-16BA000	16CA000-16CB000
137-140	16CC000-16CD000	16CF000-16D0000	16D2000-16D3000	16D7000-16D8000
141-144	16DA000-16DB000	16DD000-16DE000	1703000-1704000	1706000-1707000
145-148	1709000-170A000	170C000-170D000	170F000-1710000	1712000-1713000
149-152	1715000-1716000	1718000-1719000	171B000-171C000	171E000-171F000
153-156	1721000-1722000	1724000-1725000	1727000-1728000	172A000-172B000
157-160	172D000-172E000	1730000-1731000	1733000-1734000	1736000-1737000
161-164	1739000-173A000	173C000-173D000	173F000-1740000	1742000-1743000
165-168	1745000-1746000	1749000-174A000	2941000-2942000	2943000-2944000
169-172	2A15000-2A16000	2A18000-2A19000	2A1D000-2A1E000	2A20000-2A21000
173-176	2A23000-2A24000	2A25000-2A26000	2A28000-2A29000	2A2B000-2A2C000
177-180	2A2F000-2A30000	2A33000-2A34000	2ABF000-2AC0000	2AC2000-2AC3000
181-184	2ACE000-2ACF000	2AD1000-2AD2000	2AFC000-2AFD000	2AFF000-2B00000
185-188	2B11000-2B12000	2B14000-2B15000	2B16000-2B17000	2B19000-2B1A000
189-192	2B4A000-2B4B000	2B50000-2B51000	2B53000-2B54000	2B59000-2B5A000
193-196	2B5C000-2B5D000	2B62000-2B63000	2B65000-2B66000	2B6B000-2B6C000
197-200	2B6E000-2B6F000	2B74000-2B75000	2B77000-2B78000	2B7D000-2B7E000
201-204	2B80000-2B81000	2B86000-2B87000	2B89000-2B8A000	2B8F000-2B90000
205-208	2B92000-2B93000	2B98000-2B99000	2B9B000-2B9C000	2BA1000-2BA2000
209-212	2BA5000-2BA6000	2BA8000-2BA9000	2BAB000-2BAC000	2BBC000-2BBD000
213-216	2BBF000-2BC0000	2BD0000-2BD1000	2BD3000-2BD4000	2BF4000-2BF5000
217-220	2BF7000-2BF8000	2BFA000-2BFB000	2BFD000-2BFE000	2C0E000-2C0F000
221-224	2C11000-2C12000	2C14000-2C15000	2C17000-2C18000	2C28000-2C29000
225-228	2C30000-2C31000	2C41000-2C42000	2C44000-2C45000	2C47000-2C48000
229-232	2C4A000-2C4B000	2C4F000-2C50000	2D34000-2D35000	2D37000-2D38000
233-236	2D3A000-2D3B000	2D3D000-2D3E000	2D40000-2D41000	2D43000-2D44000
237-240	2D46000-2D47000	2D49000-2D4A000	2D4C000-2D4D000	2D50000-2D51000
241-244	2D54000-2D55000	2D65000-2D66000	2D67000-2D68000	2D78000-2D79000
245-248	2D7A000-2D7B000	2D8B000-2D8C000	2D8D000-2D8E000	2D9E000-2D9F000
249-252	2DA0000-2DA1000	2DB1000-2DB2000	2F6D000-2F6E000	2F7E000-2F7F000
253-256	2F80000-2F81000	2F91000-2F92000	2F93000-2F94000	2FA4000-2FA5000
257-260	2FA7000-2FA8000	2FB8000-2FB9000	2FBB000-2FBC000	2FCC000-2FCD000
261-264	2FD0000-2FD1000	2FE1000-2FE2000	39DC000-39DD000	39FD000-39FE000
265-268	39FF000-3A00000	3A10000-3A11000	3A13000-3A14000	3A24000-3A25000
269-272	3A26000-3A27000	3A29000-3A2A000	3A2B000-3A2C000	3A30000-3A31000
273-276	3A32000-3A33000	3A37000-3A38000	3AB9000-3ABA000	3ABE000-3ABF000
277-280	3CC5000-3CC6000	3CD6000-3CD7000	3CD9000-3CDA000	3CEA000-3CEB000
281-284	3CED000-3CEE000	3CFE000-3CFF000	3D02000-3D03000	3D13000-3D14000
285-287	3D15000-3D16000	3D26000-3D27000		

Table A1: Addresses of the halted 286 regions in M241

Table A1 presents detailed memory address information for 286 regions in the M241 PLC that either deny access (preventing any data read) or cause the PLC to halt as Figure A1. All memory addresses are organized in 4KB (0x1000-byte) increments, suggesting that the fundamental memory management block size in the M241 is 4KB. As shown in Table 3 of Section 7, only two such occurrences were observed be-

Figure A1: PLC State Changes

low 0xC94000, while an additional 284 occurrences were identified beyond 0xC94000.

Figure A2: M241 Board and Chipset with ARM Architecture

Figure A2 shows the M241 PLC board after we disassembled the device for closer inspection. Once the heatsink was removed, we could confirm that, unlike the M221 PLC which employs a Renesas chipset, the M241 uses an ARM-based chipset. Given that the M221, the most famous PLC of the Modicon series, relies on Renesas, finding ARM-structured data during the M241's memory acquisition and classification was unexpected and needed to be verified.

```
00002360  1E FF 2F E1 08 10 66 00 0C 10 66 00 0D C0 A0 E1
00002370  F0 DF 2D E9 04 B0 4C E2 18 D0 4D E2 60 C1 9F E5
00002380  00 40 9C E5 00 08 A0 E1 01 18 A0 E1 3C 30 0B E5
00002390  00 30 E0 E3 38 30 0B E5 01 40 74 E2 00 40 A0 33
```

```
Prologue1 = "0D C0 A0 E1"  # MOV R12, SP
Prologue2 = "2D E9"        # STMFD SP!
```

Figure A3: M241 ARM Binary Example

Table 3 highlights memory regions in the M241 PLC, specifically 0x000000–0x660000 and 0x660000–0x9B3000, which were predicted to contain executable code. To further validate these predictions, we performed a binwalk as mentioned in Section 7 analysis on these regions, revealing recognizable ARM instruction sequences as Figure A3. While not all data within these regions was classified as executable code, the frequency of detected instruction patterns was sufficient to delineate code segments and estimate their boundaries. This analysis confirms that our machine learning-based classification effectively distinguishes executable content from non-executable data within PLC memory, providing a foundation for targeted exploitation and further security analysis.