EPiC Series in Computing

Volume 111, 2026, Pages 1–16

Proceedings of The 21st Asia Joint Conference on Information Security

LLM-assisted Generation of Pseudo-C2 Servers for IoT Malware Dynamic Analysis

Kouki Hasui[1], Shingo Matsugaya[123], Makoto Shimamura[2], and Masaki Hashimoto[1]

[1] Graduate School of Science for Creative Emergence, Kagawa University
hashimoto.masaki@kagawa-u.ac.jp
[2] Trend Micro, Inc.
[3] Japan Cybercrime Control Center

Abstract

Most IoT malware operates as botnets dependent on Command and Control (C2) servers, but the short-lived nature of attack infrastructure often leaves samples dormant without C2 communication, hindering dynamic analysis. This paper proposes a system that combines Ghidra with a Large Language Model (LLM) to extract communication specifications from a malware binary and automatically generate a pseudo-C2 server. Experiments using Mirai demonstrate that the proposed system semantically interprets binary control structures and extracts all 20 core protocol elements in agreement with the ground truth (100% specification extraction accuracy). The generated pseudo-C2 server fully reproduces seven of ten DDoS attack vectors with attack behavior consistent with the original C2. When applied to a customized variant created by modifying the publicly available Mirai source code, the method succeeds end-to-end—from specification extraction through pseudo-C2 generation to attack reproduction—demonstrating that the LLM infers specifications from binary structures without relying on pre-trained knowledge. This approach extends the applicability of LLMs from analysis assistance to the automated construction of dynamic analysis environments.

Keywords: IoT Malware, Mirai, Pseudo-C2 Server, Automatic Generation, Static Analysis, Large Language Model

1 Introduction

In recent years, malware targeting IoT devices has caused increasing damage. According to the NICTER Observation Report 2024 published by the National Institute of Information and Communications Technology (NICT)[1], the number of cyberattack-related communications observed in 2024 reached an all-time high, with a substantial portion targeting IoT devices. In particular, since the 2016 source code release of "Mirai"[2], countless variants have proliferated, making rapid response to the growing volume of samples an urgent challenge.

The analysis of IoT malware faces two major technical barriers. First, IoT malware targets a variety of CPU architectures such as MIPS and ARM, and many samples are stripped of symbol

information, demanding considerable expertise and cost for static analysis[3]. Second, attackers tend to take down C2 servers within short periods[4]; when the C2 server is no longer operational at the time of analysis, the malware remains dormant and exhibits no attack behavior, making it difficult to identify the threat through dynamic analysis. To address this problem, analysts have traditionally constructed "observation C2 servers" by manually reverse-engineering each binary[5], but this process incurs high analysis costs and cannot keep pace with the rapidly growing volume of variants.

To overcome these challenges, this paper proposes a system that combines the reverse engineering tool Ghidra with a Large Language Model (LLM) to extract communication specifications from IoT malware binaries and automatically generate a pseudo-C2 server. The contributions of this study are threefold:

1. **Logical extraction of communication specifications by an LLM and demonstration of generality**: We show that an LLM, through static analysis, can comprehensively extract connection parameters, packet structures, and attack command systems even from binaries lacking symbol information. In particular, by logically interpreting binary control structures (such as `switch-case` statements), the LLM accurately identifies even "missing entries" in the attack vector ID system without hallucination, demonstrating analytical capabilities that go beyond simple keyword extraction. In the evaluation on Mirai, the LLM extracted all **20 core elements** of the communication protocol in agreement with the ground truth. Furthermore, when applied to a customized variant created by intentionally modifying the publicly available Mirai source code, the proposed method correctly extracted the modified values, confirming that the LLM infers specifications from binary structures without relying on pre-trained knowledge and thereby establishing the generality of the approach.

2. **Reproduction of attack behavior**: By issuing diverse attack commands from the automatically generated pseudo-C2 server, the system fully reproduced 7 of 10 attack vectors with protocol-conformant attack packets matching the behavior under the original C2.

3. **Automated construction of a dynamic analysis platform**: Through these capabilities, the proposed system constructs a dynamic analysis environment that does not depend on external C2 servers, automating the majority of an analysis process that traditionally required advanced reverse engineering skills.

The remainder of this paper is organized as follows. Section 2 reviews related work, and Section 3 describes the proposed method in detail. Section 4 presents the experimental design and results, and Section 5 discusses the findings. Finally, Section 6 concludes the paper.

2 Related Work

This section organizes existing approaches to dynamic analysis of IoT malware along the axis of "C2 server dependence," and clarifies the positioning of the present study.

2.1 Challenges in Dynamic Analysis of IoT Malware

To comprehensively grasp the impact that malware ultimately has on a system, dynamic analysis—in which the sample is executed in an isolated environment and its communications

and system calls are observed—is indispensable. However, much IoT malware is of the botnet type that begins activity only upon receiving commands from a C2 server, and this introduces an inherent challenge: unless communication with the C2 server is established, the malware does not exhibit its intended malicious behavior.

In response to this challenge, research on dynamic analysis platforms specialized for IoT environments has been advanced. "V-Sandbox," proposed by Le et al.[6], incorporates countermeasures against IoT-malware-specific anti-analysis techniques into a QEMU-based emulation environment, enabling the observation of concealed behavior. However, while such sandboxes resolve the execution-environment challenges of running malware on heterogeneous architectures, the reception of attack commands still requires a live, operational C2 server, leaving the fundamental issue of C2 dependence unresolved.

2.2 Approaches to C2-Independent Analysis and Their Limitations

To enable dynamic analysis even for samples whose live C2 servers have been taken down, four broad approaches have been attempted.

The first approach actively probes the Internet for live C2 servers. "CnCHunter," by Davanian et al.[7], identifies C2 infrastructure using a man-in-the-middle (MITM) technique, and its successor "C2Miner"[8] uses old malware samples as decoys to discover live C2 servers. However, since their success depends entirely on the existence of a live C2 server, they cannot be applied in the modern threat landscape, where attack infrastructures are short-lived[4], to samples whose C2 has already been taken down.

The second approach involves analysts manually reverse-engineering the binary and constructing a local observation C2 server[5]. While this approach is independent of C2 availability, it requires advanced reverse engineering skills and substantial human cost for each sample, making rapid response to the daily emergence of new variants difficult.

The third approach attempts to automatically generate counterpart systems. Musch et al.[9] proposed "Chameleon," which targets web applications and constructs a low-interaction honeypot by learning response templates from observations of real systems; however, this presupposes a live system as the learning source and cannot be directly applied to samples whose C2 servers have been taken down. Borzacchiello et al.[10] applied symbolic execution to remote access trojan (RAT) binaries to synthesize pseudo-C2 servers, but symbolic execution faces critical weaknesses such as path explosion and the difficulty of constraint solving for cryptographic processing, and its target is limited to RATs, posing scalability limitations for modern IoT malware with custom loop processing and obfuscation.

The fourth approach reactively generates counterpart communication while observing malware responses during dynamic analysis. "RIoTMAN,"[11] proposed by Darki et al., achieves C2-server impersonation through iterative adaptation of the target environment configuration and automated engagement with the malware, demonstrating that approximately 79% of 3,024 IoT malware samples could be induced to transition to DDoS-attack or proliferation phases. While RIoTMAN is the prior study most closely aligned with the present work in problem awareness, its reactive response generation does not entail deep protocol understanding of the binary, leading to inherent limitations in the comprehensiveness of communication specifications and in the qualitative fidelity of attack reproduction.

2.3 Application of LLMs to Malware Analysis

In recent years, research applying the high inferential capabilities of Large Language Models (LLMs) to malware analysis has been accelerating[12]. LLMs can infer the original semantics

Table 1: Comparison of related work with this study

Category	Representative work	LLM	C2 dep.	Limitations / Difference from this work
Dynamic analysis (platform/observation)	Le [6] (V-Sandbox)	×	Required	A live C2 is required to receive attack commands
Dynamic analysis (C2 discovery)	Davanian [7, 8]	×	Mandatory	Analysis becomes infeasible after C2 takedown
Pseudo-C2 construction (static/automatic)	G DATA [5], Musch [9], Borzacchiello [10]	×	Not required	High human cost / requires learning source / path explosion / RAT-only
Pseudo-C2 construction (dynamic/reactive)	Darki [11] (RIoTMAN)	×	Not required	Reactive response generation limits comprehensiveness of specifications
LLM application (analysis support)	Li [13] (IRIS), Fujii [14]	○	Not required	Limited to static analysis support; does not reach attack-infrastructure generation
This work	**Pseudo-C2 automatic generation**	○	**Not required**	**Automatically generates a C2 based on LLM-driven specification extraction; full observation possible even after C2 takedown**

of processing from instruction patterns and context even in assembly code that lacks symbol information, and have the advantage of avoiding path explosion because they do not depend on rigorous mathematical models like symbolic execution.

As representative studies, "IRIS," by Li et al.[13], automatically generates analysis rules (taint specifications) for a static analysis tool (CodeQL) using an LLM, improving vulnerability detection capabilities by approximately twofold. Fujii and Yamagishi[14] input decompiled malware code into an LLM to generate functional descriptions, and demonstrated that they could comprehensively explain malware behavior with up to 90.9% accuracy.

However, these studies position the LLM merely as an "auxiliary tool for static analysis," and there is no example that goes as far as automatically constructing dynamic-analysis infrastructure (such as a pseudo-C2 server) capable of actually interacting with malware based on the extracted specifications.

2.4 Positioning of This Study

A comparison between the related work described above and the present study is organized in Table 1.

The novelty of this study lies in applying the inferential capabilities of LLMs not only to "analysis" but also to the "automatic generation of attack infrastructure (a pseudo-C2 server)." Specifically, based on specifications extracted by static analysis, the LLM autonomously generates implementation code for a C2 server. This makes it possible to forcibly establish communication in an isolated environment and to observe detailed attack behavior even for samples whose live C2 servers have been taken down. In other words, the present method is an approach that simultaneously realizes "automation of static analysis" and "elimination of C2 dependence in dynamic analysis," positioning it as an integrated analysis platform that comprehensively addresses the limitations of existing research.

3 Proposed Method

3.1 System Overview

This study proposes a "pseudo-C2 server automatic generation system." As illustrated in Figure 1, the proposed system consists of the following two steps.

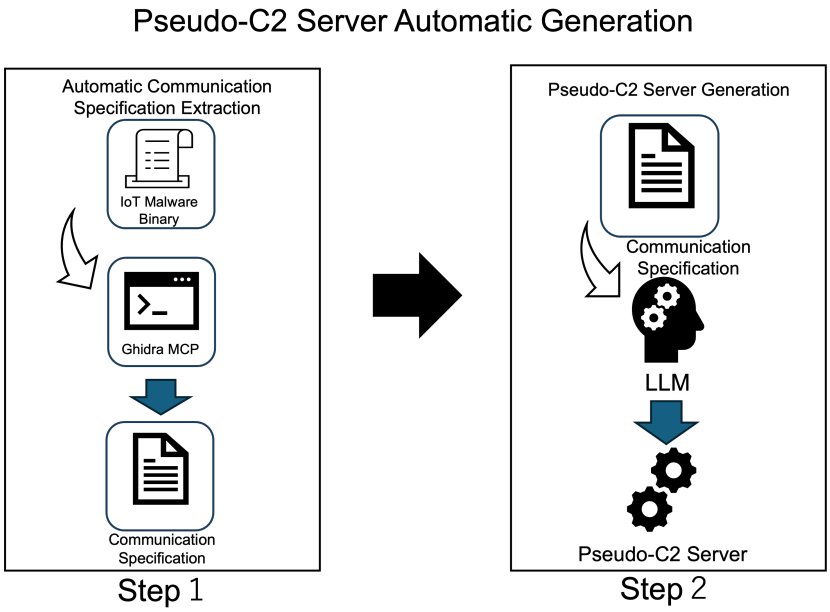

Figure 1: Overall flow of the proposed pseudo-C2 server automatic generation

1. **Communication specification extraction (Step 1)**: Taking an IoT malware binary as input, the system extracts the protocol and command system used to communicate with the C2 server through static analysis using Ghidra and an LLM, and outputs the result as a textual specification document.
2. **Pseudo-C2 server generation (Step 2)**: Using the specification document obtained in Step 1 as input, the system feeds it to the LLM to automatically generate a pseudo-C2 server program that can communicate with the malware.

By providing the specification document obtained in Step 1 as a clear constraint to the LLM in Step 2, this two-step design enables the malware analyst to construct a dynamic analysis environment without depending on advanced reverse engineering skills.

3.2 Communication Specification Extraction (Step 1)

In Step 1, the malware binary is loaded into Ghidra, and the LLM analyzes the decompiled output to extract the communication specifications required for connecting to the C2 server. The information items targeted for extraction are organized into three categories as shown in Table 2: connection establishment, protocol structure, and command system. The LLM infers each of these items from constant definitions, control structures (such as conditional branches and `switch-case` statements), and API calls within the decompiled pseudo-code, and outputs the result as a textual specification document.

In particular, since most IoT malware obfuscates its C2 server location and related strings using custom encryption routines to conceal the C2 address, the LLM analyzes such obfuscation routines (e.g., table-based XOR encryption) to penetrate the obfuscation layer and extract the correct values. This capability enables the proposed method to reach analytical depths

Table 2: Structure of the communication specification document extracted in Step 1

Category	Item	Content
Connection	C2 server address	The destination to which the malware attempts to connect
	Handshake data	Verification byte sequences (e.g., magic numbers) used to authenticate the communication
Protocol structure	Header/payload structure	Header size and the position/size of the payload length field
	Endianness	Byte order of numerical data (big/little)
Command system	Attack vector ID	Mapping between attack command IDs from the C2 and the corresponding attack methods (e.g., SYN flood)
	Argument structure	Serialization format of attack targets, durations, etc.

unattainable by simple static string extraction, distinguishing it from conventional signature-based analysis.

3.3 Tool Support Mechanism Design

The proposed system adopts the publicly available tool "GhidraMCP" [15] as the foundation for providing binary analysis context from Ghidra to the LLM. However, through the evaluation experiments in this study, we found that the existing GhidraMCP feature set occasionally fails to provide the LLM with the information needed for certain analysis targets. Specifically, the existing implementation lacks adequate functionality for retrieving constant values hardcoded at specific binary addresses (such as magic numbers and encryption keys) and short string constants referenced after decryption. While GhidraMCP supports the retrieval of high-level information such as decompilation results and call graphs, it does not provide the low-level functionality of directly accessing raw byte sequences stored at specific memory addresses. As a result, the LLM cannot receive such information as context, leading to extraction failures (details are described in Section 4.3).

To address this limitation, we extended GhidraMCP by implementing three additional MCP tools: (i) `read_bytes`, which reads raw byte sequences from arbitrary memory addresses in hexdump format; (ii) `read_string_at`, which reads NULL-terminated ASCII strings; and (iii) `get_section_info`, which retrieves the list of memory blocks (`.text`, `.rodata`, `.data`, `.bss`, etc.) of an ELF binary. These extensions enable the LLM to access low-level information stored at arbitrary memory addresses within the binary, including raw byte sequences, NULL-terminated strings, and memory block layouts. As demonstrated in Section 4.3, this tool extension plays an essential role in eliciting the inferential capabilities of the LLM and enabling specification extraction based on binary structures rather than pre-trained knowledge. This tool extension is positioned as an essential component of LLM-assisted binary analysis.

3.4 Pseudo-C2 Server Generation (Step 2)

In Step 2, the communication specification document output in Step 1 is fed to the LLM as a prompt to automatically generate a pseudo-C2 server capable of communicating with the malware. Python is specified as the implementation language, given its ease of socket communication control and high readability.

The prompt design for the generation process emphasizes three points. First, the contents of the specification document are explicitly stated as constraints to be strictly followed, ensuring binary compatibility for low-level specifications such as endianness and header length. Second,

detailed log output and error handling are required to facilitate troubleshooting when communication fails. Third, a natural-language interface for issuing attack commands is required to enhance analyst usability.

The pseudo-C2 server generated as a result possesses the following capabilities to support dynamic analysis:

- **Handshake establishment**: Verifies post-connection data (such as magic numbers) and maintains communication only with legitimate bots, thereby excluding unrelated scanning traffic.
- **Attack command generation**: Automatically converts natural-language attack names entered by the analyst (e.g., "SYN flood") into the malware-specific binary values (Vector IDs) based on the ID correspondence table extracted in Step 1, allowing the analyst to invoke arbitrary attack methods through intuitive operations without needing to be aware of the binary values.
- **Debugging features**: Logs all sent and received packets in real time in both hexadecimal (hexdump) and ASCII format, enabling byte-level verification of consistency between the specifications and actual communication packets.

4 Experimental Design and Results

4.1 Experimental Design

4.1.1 Target and Evaluation Approach

To verify the effectiveness of the proposed method, we conduct empirical experiments targeting the publicly available binary of "Mirai," a representative IoT malware whose specifications are known. As the reference dataset, we adopt "Mirai-Source-Code,"[2] a repository published on GitHub by J. Gamblin. This repository is curated from the leaked Mirai source code with incomplete files and configurations removed to allow compilation, and is widely referenced as the standard reference implementation of Mirai in the security research field.

The rationale for selecting Mirai as the verification target is threefold. First, since the leak in September 2016, Mirai has continued to spawn numerous variants such as "Satori" and "Okiru,"[16] establishing it as the de facto standard of the IoT threat landscape. Second, because the communication protocol definitions described in the public source code (under `mirai/bot/`) are available, the specifications extracted by the proposed method from binaries can be rigorously compared against the known ground truth. Third, Mirai exhibits a structure in which a wide range of DDoS attack vectors over UDP/TCP/GRE/HTTP/DNS are controlled by a single-byte attack ID (Vector ID), making it well-suited for verifying the comprehensiveness of specification extraction.

4.1.2 Evaluation Metrics

We evaluate the effectiveness of the proposed method using the following two metrics.

(1) Specification Extraction Accuracy: The percentage of items, among the core elements of the Mirai communication protocol (connection parameters and attack vector ID system) in the specification document obtained in Step 1, that match the ground truth derived

from the public source code.

$$\text{Extraction Accuracy (\%)} = \frac{\text{Number of items matching the ground truth}}{\text{Total number of ground-truth items}} \times 100 \quad (1)$$

(2) Attack Reproduction Success Rate: The classification of each attack method into the following three categories based on the result of issuing attack commands from the pseudo-C2 server generated in Step 2.

- **Fully reproduced**: The packet structure (protocol type, flags, header, payload length, etc.) matches that of the original C2, and the attack is carried out to completion.
- **Partially reproduced**: The bot receives the attack command and starts the attack function, but the attack is not completed due to environmental factors.
- **Environmental constraint**: The attack behavior itself cannot be observed due to the constraints of the closed experimental environment.

4.1.3 Network Configuration

The experimental environment was constructed on a closed network built on a virtualization platform, comprising three virtual machines: a Bot machine (`192.168.1.102`) that executes the Mirai samples, a C2 machine (`192.168.1.10`) that runs either the original C2 or the pseudo-C2 server, and a Target machine (`192.168.1.50`) that observes incoming attack packets via `tcpdump`. The network is physically and logically isolated from the external Internet, eliminating the risk of malware leaking outside. Considering the ease of dynamic analysis, we executed Mirai samples on the bot machine that had been cross-compiled to x86 32-bit, taking into account that the majority of target IoT devices use 32-bit processors and following the standard build configuration of the reference Mirai-Source-Code repository. This applies to both Experiment I and Experiment II.

In addition, we individually verified the completion of the measurement interval (10 seconds) for every attack method through packet captures, confirming that no measurement was interrupted or truncated.

4.1.4 Setup of Communication Control

The Mirai sample used in this experiment attempts to connect to a hard-coded destination IP (`65.222.202.53`) and TCP port 80, and additionally judges whether to continue running based on whether Google Public DNS (`8.8.8.8`) responds at startup. To establish communication with the pseudo-C2 in a closed environment without modifying the binary at all, we applied a three-stage communication control to the bot machine at the OS level using `iptables` and routing tables.

First, a static route to the hard-coded destination was added, which prevents immediate packet drops due to "unknown destination" in an environment without a default gateway and allows the sample's send operations to complete normally. Second, traffic to `8.8.8.8` was redirected via DNAT to the local pseudo-C2, returning a pseudo response that satisfies the malware's Internet-connectivity check and thereby allowing the bot's startup sequence to proceed. Third, packets destined for the hard-coded C2 address (`65.222.202.53:80`) were transparently forwarded via DNAT and port translation to the pseudo-C2 server (`192.168.1.10:23`).

These controls realize an environment in which the sample can be operated under the control of the pseudo-C2 server without any modification to the sample, even in the closed environment.

Note that in Experiment II described in Section 4.3, the C2 connection destination on the sample side is intentionally changed, so the third control is replaced with a setting specific to Experiment II (details are given in Section 4.3).

4.1.5 LLM Selection

For the selection of an LLM, we first conducted preliminary verification using locally executable open-source models (Llama 3 70B, DeepSeek-Coder-V2, etc.) from the perspective of confidentiality. However, in the task of comprehensively extracting C2 communication specifications from the fragmentary decompiled code that Ghidra outputs, these models proved insufficient in contextual understanding, with frequent hallucinations occurring in parameter identification. Accordingly, this study adopts Anthropic's commercial models, which possess higher inferential capabilities, via the CLI interface "Claude Code." Specifically, "Claude Opus 4.5" was used in Experiment I (Section 4.2), and "Claude Opus 4.7" was used in Experiment II (Section 4.3) due to a model update during the experimental period; we confirmed that both versions reliably perform specification extraction and pseudo-C2 server generation under the proposed method. The analysis prompt employs neutral wording such as "extract the specifications related to the communication of the binary currently selected," and no interruption of analysis by the LLM's safety mechanisms occurred.

4.2 Experiment I: Evaluation on the Original Mirai

4.2.1 Communication Specification Extraction Results

As a result of LLM-based analysis, the proposed method successfully extracted the parameters required for C2 communication in a comprehensive manner from the obfuscated initialization function (`table_init`) and the attack processing loop (`attack_init`) within the binary. The extraction results are shown in Table 3. The targets of extraction comprise four connection parameters (C2 IP address, port, Magic Number, and Bot ID Length), five command-packet structure items (offset, size, and byte order of each field), and 11 attack vector ID items (`0x00`–`0x0A`, including the missing entry `0x08`), totaling 20 items. In particular, all of the following matched the ground truth: the XOR-encrypted C2 IP address, the packet structure recovered by tracing the control flow of the parsing routine (`attack_parse`) within the binary, and the Vector ID mapping extracted from the registration logic. A particularly noteworthy point is that the LLM correctly recognized the fact that Vector ID `0x08` is, by definition, a missing entry; the significance of this observation is discussed in Section 5.

Based on these results, the specification extraction accuracy in this experiment matched the ground truth on all **20 items**, achieving **20/20 (100%)**. Furthermore, the keep-alive mechanism (a 2-byte `0x0000` sent at 60-second intervals) and other control mechanisms necessary for maintaining a long-lived connection were also extracted, providing the foundational information needed for long-term operation of the pseudo-C2 server.

4.2.2 Attack Reproduction Results

We verified the attack behavior triggered by the proposed pseudo-C2 server in comparison with the original C2 server. After the bot machine started up, the pseudo-C2 server completely reproduced Mirai's specific handshake sequence and was successfully recognized as a legitimate C2: immediately after TCP connection establishment, it correctly received the magic number

Table 3: Communication specification extraction results in Experiment I (all 20 items)

Category	Item	LLM Extraction Result	Ground Truth
Connection parameters	C2 IP Address	65.222.202.53	65.222.202.53
	C2 Port	80/TCP	80/TCP
	Magic Number	0x00000001	0x00000001
	Bot ID Length	1 byte	1 byte
Packet structure	Offset 0x00	Packet Length (2B, BE)	Match
	Offset 0x02	Attack Duration (4B, NBO)	Match
	Offset 0x06	Attack Vector ID (1B)	Match
	Offset 0x07	Number of Targets (1B)	Match
	Offset 0x08–	Target Entries (IP+mask)	Match
Attack Vector ID	0x00	UDP Generic Flood	Match
	0x01	UDP VSE Flood	Match
	0x02	UDP DNS Flood	Match
	0x03	TCP SYN Flood	Match
	0x04	TCP ACK Flood	Match
	0x05	TCP STOMP Flood	Match
	0x06	GRE IP Flood	Match
	0x07	GRE Ethernet Flood	Match
	0x08	**(missing)**	**Recognized as missing**
	0x09	UDP Plain Flood	Match
	0x0A	HTTP Flood	Match

`0x00000001` sent by the bot, completed the bot ID exchange, and then transitioned to the keep-alive maintenance phase, indicating that the LLM accurately understood Mirai's communication initialization sequence.

For TCP-based attacks (SYN flood, ACK flood, STOMP flood), the pseudo-C2 server issued attack commands and the bot generated attack packets accordingly. For SYN/ACK floods, packet structures (sequence numbers, source IPs, destination ports, etc.) consistent with the original C2 environment were observed. For STOMP, although TCP three-way handshake attempts were observed, full attack execution was not completed; this is treated as "partial reproduction" on the bot side. For UDP-based attacks (UDP Generic, VSE, Plain), the pseudo-C2 server faithfully reproduced UDP packet generation; in particular, for UDP VSE flood, the bot generated attack packets containing the Source Engine query payload (`0xFF 0xFF 0xFF 0xFF 0x54 Source Engine Query 0x00`), demonstrating that the LLM accurately interpreted vendor-specific protocol structures from the binary as well. For GRE-based attacks (GRE IP, GRE Ethernet), the pseudo-C2 server triggered the generation of GRE-encapsulated packets; particularly noteworthy is that for GRE Ethernet flood, GRE-encapsulated Ethernet frames were correctly generated, indicating that the LLM was able to handle the unique encapsulation specifications of the GRE protocol.

For DNS Flood and HTTP Flood, no attack packets were observed at the target. Since these attacks presuppose protocol-conformant communication with an external DNS server or web server, the cause is considered to be the absence of the corresponding attack-target services on the closed network. This observation failure is not a limitation of the proposed method but stems from the constraints of the experimental environment, and the issuance of attack commands itself was processed normally on the C2 side. This perspective is discussed in Section 5.

Table 4 summarizes the reproduction result for all 10 attack methods. Among the 10 attack vectors, 7 were fully reproduced with protocol-conformant attack packets matching the behavior under the original C2, 1 (TCP STOMP) was partially reproduced (attack command received and connection attempted, but full execution not completed due to dependence on target-side responses), and 2 (DNS Flood, HTTP Flood) failed to be observed due to environmental

Table 4: Reproduction results for all attack methods

Attack Method	Result
TCP SYN Flood	Fully reproduced
TCP ACK Flood	Fully reproduced
TCP STOMP Flood	Partially reproduced (connection attempts only)
UDP Generic Flood	Fully reproduced
UDP VSE Flood	Fully reproduced
UDP Plain Flood	Fully reproduced
GRE IP Flood	Fully reproduced
GRE Ethernet Flood	Fully reproduced
DNS Flood	Observation failed (no DNS server in the environment)
HTTP Flood	Observation failed (no web server in the environment)

Table 5: Modifications and extraction results for Variant II

Item	Original Value	Variant II Value	LLM Extraction Result
Magic Number	`0x00000001`	`0xCAFEBABE`	`0xCAFEBABE`
Encryption key	(original key)	`0xDEADBEEF`	`0xDEADBEEF`
C2 IP address	`65.222.202.53`	`192.168.99.99`	`192.168.99.99`
C2 Port	80/TCP	80/TCP	80/TCP
Attack Vector IDs (10)	`0x00`–`0x0A`	Randomly reassigned	Exact match with modified values

constraints in the closed network.

4.3 Experiment II: Verification with a Customized Variant

4.3.1 Purpose of Verification and Design of the Customized Variant

The results of Experiment I in Section 4.2 demonstrated that the proposed method extracts Mirai's communication specifications with high accuracy (specification extraction accuracy of 100%). However, since Mirai's source code has been publicly available together with related analysis reports since the leak of September 2016[2], the possibility that the LLM's pre-trained data contains the ground-truth information cannot be ruled out. Accordingly, it is essential to verify whether the results of Experiment I stem from the LLM's pure inferential capabilities based on binary structures or from supplementation by pre-trained knowledge.

To address this verification objective, this section creates a customized variant (hereafter, "Variant II") by intentionally modifying the publicly available Mirai source code, and applies the proposed method. The modifications applied to Variant II, together with the LLM extraction results obtained by the proposed method, are shown in Table 5. None of these modified values appear in the publicly available Mirai source code, so if the LLM correctly extracts them, this can only be attributed to inference based on the binary structure rather than reliance on pre-trained data.

In particular, for the attack Vector IDs, the identifiers for all 10 attack methods were reassigned to random values unrelated to the originals (`0x91`, `0x73`, `0x2d`, `0x19`, `0x5c`, `0x4b`, `0x82`, `0x27`, `0x6a`, `0x3e`). This was designed such that the LLM cannot reconstruct the ID system unless it correctly analyzes the registration logic of the `attack_init` function.

4.3.2 Communication Specification Extraction Results

For Variant II, we launched a new Claude Code session independent of the one used in Experiment I, and applied Step 1 of the proposed method. This was done to eliminate any influence on the analysis results that residual session context from Experiment I might cause.

As a result, as shown in Table 5 (in the previous subsection), the LLM successfully extracted values matching the ground truth for Variant II for all of the modified items.

A particularly notable point is that, despite the fact that the attack Vector ID system in Variant II differs significantly from the original (e.g., the value corresponding to TCP SYN Flood, which is `0x03` in the original, is reassigned to `0x19` in Variant II), the LLM completely and correctly extracted the new ID system by analyzing the registration logic of the `attack_init` function within the binary. This is strong evidence that the LLM is reconstructing specifications from the actual binary structure, without being misled by the legacy system (`0x00`–`0x0A`) contained in its pre-trained data.

However, two issues in the experimental design were observed during this verification process. First, regarding the extraction of the C2 IP address, in the initial implementation the old IP (`65.222.202.53`) was left within the binary in encrypted form, and the LLM accurately decrypted it and consequently extracted the old IP. This was not a problem on the LLM side, but rather originated from incomplete modification of one of the references in the source code (`mirai/bot/includes.h`); it was resolved by completing the modification. Second, regarding the extraction of the Magic Number, the GhidraMCP we used did not implement the function for retrieving raw byte sequences at specific addresses, so contextual information itself was not provided to the LLM, and the extraction did not succeed. This was addressed through the GhidraMCP extension described in Section 3.3 (Proposed Method). Through subsequent reanalysis after addressing these issues, accurate extraction was achieved for all items. These observations suggest that, in order to elicit the LLM's inferential capabilities, the information-providing capability of the tooling side is decisively important, and we discuss this further in Section 5.3.

4.3.3 Pseudo-C2 Server Generation and Attack Verification

Using the communication specification document extracted from Variant II as input, we automatically generated a pseudo-C2 server using Step 2 of the proposed method. The generated code consists of six files (`main.py`, `server.py`, `protocol.py`, `attacks.py`, `repl.py`, `test_smoke.py`) and accompanying documentation (`README.md`), and includes builder functions corresponding to all 10 attack vectors as well as 19 test cases, demonstrating a high-quality implementation. The generated code also includes defensive implementations such as isolated-environment checks, KEEPALIVE handling, and payload-length validation, demonstrating that the LLM autonomously implements comprehensive functionality based on the specification document.

The generated pseudo-C2 server was operated against Variant II, and 10 types of attacks were triggered. The reproduction results are shown in Table 6, in which 7 of 10 attacks were fully reproduced, with the same three attacks (DNS/HTTP/STOMP) failing to be observed for the same reasons as in Experiment I (DNS/HTTP due to environmental constraints, TCP STOMP due to dependence on connection responses). This indicates that the behavior of the proposed method functions consistently regardless of sample-specific specification changes.

4.3.4 Comparison with Experiment I

We compare the main results of Experiment I (the original Mirai) and Experiment II (Variant II). Regarding specification extraction accuracy, Experiment I achieved a 100% match with the ground truth across all 20 items, and Experiment II likewise achieved a complete match across all four primary items. Regarding attack reproduction success rate, both experiments resulted in equivalent outcomes of 7 out of 10 fully reproduced, and the three attacks that failed to be observed (DNS/HTTP/STOMP) were exactly identical in both experiments.

Table 6: Attack reproduction results for Variant II

Attack ID	Attack Method	Result
0x91	GRE Ethernet Flood	Fully reproduced
0x73	DNS Flood	Observation failed (no DNS server)
0x2d	HTTP Flood	Observation failed (no web server)
0x19	TCP SYN Flood	Fully reproduced
0x5c	GRE IP Flood	Fully reproduced
0x4b	UDP Generic Flood	Fully reproduced
0x82	TCP ACK Flood	Fully reproduced
0x27	UDP VSE Flood	Fully reproduced
0x6a	UDP Plain Flood	Fully reproduced
0x3e	TCP STOMP Flood	Partially reproduced (connection attempts only)

The fact that the number of successful attack reproductions and the failure pattern coincide across both experiments demonstrates that the proposed method functions stably without depending on individual sample specifications or environmental differences. In particular, Variant II showed that the LLM correctly extracted communication specifications that do not exist in the publicly available Mirai source code (Magic Number `0xCAFEBABE`, encryption key `0xDEADBEEF`, C2 IP `192.168.99.99`, and the reassigned attack Vector ID system), demonstrating that the LLM of the proposed method can infer specifications from binary structures without relying on pre-trained data.

5 Discussion

5.1 Logical Interpretation Capability in Communication Specification Extraction

The specification extraction accuracy in Experiment I reached **20/20 items (100%)**. A particularly noteworthy point is that the LLM correctly recognized the fact that Vector ID `0x08` is, by definition, a missing entry (Table 3). Because LLMs generate responses based on the probabilistic prediction of tokens, they are prone to hallucinations such as "filling in the gap with `0x08` after `0x07`" when generating a sequential list. Nevertheless, in the proposed method, the LLM logically interpreted the structure of the `switch-case` statement implementing attack vector selection during its analysis of the decompiled output, and identified the fact that "no branch label corresponding to `0x08` exists" without hallucination.

This observation suggests that the proposed method is not merely performing pattern matching or keyword extraction, but is rather **extracting specifications based on a semantic interpretation of the program's control structures**. The fundamental reason why the correspondences between attack IDs (10 types within `0x00`–`0x0A` excluding `0x08`) and each attack function (`attack_udp_generic`, etc.) could be comprehensively and accurately extracted is considered to lie in this logical interpretation capability. The fact that the pseudo-C2 server automatically generated in Step 2 was able to reproduce diverse attacks rests on this precise specification extraction in Step 1.

5.2 Inferential Capability of the LLM on Binary Structures

The results of Experiment II shown in Section 4.3 demonstrated that the LLM of the proposed method can infer specifications from binary structures without relying on pre-trained data. In Variant II, the LLM extracted, in agreement with the ground truth, all of the following items that do not appear in the publicly available Mirai source code: Magic Number `0xCAFEBABE`,

encryption key `0xDEADBEEF`, C2 IP address `192.168.99.99`, and the completely reassigned attack Vector ID system. These values are unique to the variant derived from the original Mirai of Experiment I, and are, in principle, unobtainable from public information sources.

In particular, the fact that the LLM was able to accurately follow the reassignment of attack Vector IDs is significant. There was a real possibility that the LLM, influenced by the legacy system (`0x00`–`0x0A`) contained in its pre-trained data, would output incorrect correspondences; in fact, however, it completely reconstructed the new system by analyzing the registration logic of the `attack_init` function. Combined with the logical interpretation capability discussed in Section 5.1, this strongly demonstrates that the LLM is performing inference grounded in the actual state of the binary.

This result empirically rules out the fundamental concern that "the LLM may merely be guessing the correct answers based on prior knowledge." Therefore, the proposed method is suggested to be applicable, in principle, even to novel IoT malware samples with previously unseen communication specifications.

5.3 Importance of the Tool Support Mechanism

The initial failure in Magic Number extraction observed in Section 4.3.2 provides an important insight for this study. That is, no matter how sophisticated the inferential capabilities of an LLM are, inference cannot succeed unless information about the analysis target binary is provided to it as context. Because the existing GhidraMCP did not have a function for extracting constant values at specific addresses, the LLM could not access the raw byte sequences in the memory region corresponding to the Magic Number, and the specification extraction did not succeed.

Through the GhidraMCP extension implemented in this study (Section 3.3), the LLM became able to access the constant values at any address in the binary, and as a result, accurate extraction was achieved for all items. This insight suggests that, in future research on LLM-assisted binary analysis, the essential point lies in optimizing not the "capability of the LLM alone" but the "**combination of the LLM and information extraction tools**." In particular, the question of what kinds of binary information should be provided to the LLM in reverse engineering tasks holds value as an independent research challenge for the future.

5.4 Limitations of This Study

This study currently has the following limitations, which we explicitly state as challenges for future verification.

First, there are **constraints on the attack observation environment**. For DNS Flood and HTTP Flood, no attack packets were observed; this is not a control failure of the proposed method but stems from constraints of the experimental environment. These attack methods presuppose, prior to issuing attack packets, the resolution of an external domain name or the completion of a TCP three-way handshake with a web server. Since the closed network of this experiment did not contain a responsive DNS server or web server, it is considered that the bot timed out at the connection-attempt stage before the attack began. The fact that the same three observation-failure attacks (DNS/HTTP/STOMP) coincide exactly between Experiments I and II supports the interpretation that the observation failure stems from the environment side rather than from the method side. Going forward, we expect that integrating a virtual honeypot (a responsive DNS/web server) into the experimental environment will make these attack methods observable as well.

Second, there is the **limited scope of applicability**. While we demonstrated that the proposed method is effective for analyzing Mirai and its derivative variants, the applicability

to other IoT malware families (Gafgyt/BASHLITE, etc.) and to samples with more sophisticated encrypted communication or custom protocols remains unverified. Although Experiment II demonstrated that the LLM of the proposed method can infer specifications from binary structures, whether comparable accuracy is obtained for malware families with different design philosophies requires separate verification.

Third, the **quantitative evaluation of long-term stability and cost-effectiveness** is insufficient. The experiment in this study was confined to short-term verification centered on 10-second attack observations, and verification of long-term connection maintenance (such as the continuity of keep-alive processing) over hours or days has not been conducted. In addition, regarding the analysis-time reduction effect of the proposed method, a quantitative comparison with conventional manual analysis remains as a challenge for future work. Similarly, the operational cost of the LLM—in terms of API calls, token consumption, and runtime—was not systematically measured in this study; a quantitative cost analysis is essential for assessing the scalability of the proposed method to large-scale analysis settings, and remains an important direction for future work.

6 Conclusion

This paper proposed a pseudo-C2 server automatic generation system that integrates Ghidra with a Large Language Model (LLM), addressing the challenges of short-lived C2 servers and increasing analysis cost in IoT malware dynamic analysis. Experiments on Mirai demonstrated the three contributions of Section 1: the LLM extracted all **20 items** of the core protocol elements with 100% accuracy by semantically interpreting control structures such as `switch-case` statements; the generated pseudo-C2 server fully reproduced 7 of 10 DDoS attacks (TCP/UDP/GRE) with attack behavior consistent with the original C2; and the resulting analysis platform automates a process that traditionally required advanced reverse engineering skills. Furthermore, applying the method to a customized variant of the publicly available Mirai source code (Experiment II) succeeded end-to-end, demonstrating that the LLM infers specifications from binary structures without relying on pre-trained data.

The significance of this study lies in extending LLM applicability from static analysis assistance to automatic generation of attack infrastructure for dynamic analysis, suggesting applicability to novel IoT malware with previously unseen specifications.

Future work includes: (i) verification on other malware families such as Gafgyt (BASHLITE) to enhance generality; (ii) integrating a virtual honeypot (DNS/web servers) to observe all attack vectors including DNS/HTTP Flood; (iii) verifying long-term stability and quantitatively evaluating analysis-time reduction; and (iv) fine-tuning open-source LLMs (such as Llama) for malware analysis to ensure confidentiality and reduce dependence on commercial LLMs.

Acknowledgments

We acknowledge the use of Claude AI assistant (Anthropic) for the English translation of this paper. This work was supported in part by the JSPS/MEXT KAKENHI under Grant 24K14956.

References

[1] National Institute of Information and Communications Technology (NICT). NICTER Observation Report 2024. https://www.nicter.jp/report, 2025. (in Japanese).

[2] J. Gamblin. Mirai-Source-Code. https://github.com/jgamblin/Mirai-Source-Code. Accessed: Feb., 2026.

[3] D. Gomes, E. Felix, F. Aires, and M. Vieira. Static code analysis for iot security: A systematic literature review. *ACM Computing Surveys*, 58(3):1–47, 2025.

[4] Omar Alrawi, Chaz Lever, Kevin Valakuzhy, Ryan Court, Kevin Snow, Fabian Monrose, and Manos Antonakakis. The circle of life: A large-scale study of the iot malware lifecycle. In *Proceedings of the 30th USENIX Security Symposium*, pages 3505–3522, 2021.

[5] G DATA CyberDefense. Reverse engineering and observing an iot botnet, Aug. 2020. Accessed: Feb., 2026.

[6] Hai-Viet Le and Quoc-Dung Ngo. V-sandbox for dynamic analysis iot botnet. *IEEE Access*, 8:145768–145786, 2020.

[7] Ali Davanian, Ahmad Darki, and Michalis Faloutsos. CnCHunter: An MITM-approach to Identify Live CnC Servers. Black Hat USA 2021 (Whitepaper), 2021.

[8] Ali Davanian, Michalis Faloutsos, and Martina Lindorfer. C2Miner: Tricking IoT Malware into Revealing Live Command & Control Servers. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security (ASIA CCS '24)*, pages 112–127, 2024.

[9] Marius Musch, Martin Härterich, and Martin Johns. Towards an Automatic Generation of Low-Interaction Web Application Honeypots. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES '18)*, pages 1–6, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Luca Borzacchiello, Emilio Coppa, Daniele Cono D'Elia, and Camil Demetrescu. Reconstructing c2 servers for remote access trojans with symbolic execution. In Shlomi Dolev, Danny Hendler, Sachin Lodha, and Moti Yung, editors, *Cyber Security Cryptography and Machine Learning*, pages 121–140, Cham, 2019. Springer International Publishing.

[11] Ahmad Darki and Michalis Faloutsos. RIoTMAN: A Systematic Analysis of IoT Malware Behavior. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '20)*, pages 169–182, 2020.

[12] H. Jelodar, S. Bai, P. Hamedi, H. Mohammadian, R. Razavi-Far, and A. Ghorbani. Large language model (llm) for software security: Code analysis, malware analysis, reverse engineering. arXiv preprint arXiv: 2504.07137, 2025.

[13] Z. Li, S. Dutta, and M. Naik. Iris: Llm-assisted static analysis for detecting security vulnerabilities. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[14] S. Fujii and R. Yamagishi. Feasibility study for supporting static malware analysis using llm. arXiv preprint arXiv:2411.14905, 2024.

[15] Laurie Wired. Ghidramcp. https://github.com/LaurieWired/GhidraMCP. Accessed: Feb., 2026.

[16] Antonio Affinito, Savio Zinno, Gennaro Stanco, Alessio Botta, and Giorgio Ventre. The evolution of mirai botnet scans over a six-year period. *Journal of Information Security and Applications*, 79:103629, 2023.