BACnet or "BADnet"? On the (In)Security of Implicitly Reserved Fields in BACnet

Qiguang Zhang [†], Junzhou Luo [†¶], Zhen Ling[†*], Yue Zhang[‡],
Chongqing Lei[†], Christopher Morales[§], Xinwen Fu[§]
[†]Southeast University, Email: {qgzhang, jluo, zhenling, leicq}@seu.edu.cn
[‡]Shandong University, Email: zyueinfosec@gmail.com
[§]University of Massachusetts Lowell, Email: christopher_moralesgonzalez@student.uml.edu, xinwen_fu@uml.edu
[¶]Fuyao University of Science and Technology

Abstract*—Building Automation Systems (BASs) are crucial for managing essential functions like heating, ventilation, air conditioning, and refrigeration (HVAC&R), as well as lighting and security in modern buildings. BACnet, a widely adopted open standard for BASs, enables integration and interoperability among heterogeneous devices. However, traditional BACnet implementations remain vulnerable to various security threats. While existing fuzzers have been applied to BACnet, their efficiency is limited, particularly due to the slow bus-based communication medium with low throughput. To address these challenges, we propose BACsFuzz, a behavior-driven fuzzer aimed at uncovering vulnerabilities in BACnet systems. Unlike traditional fuzzing approaches focused on input diversity and execution path coverage, BACsFuzz introduces the token-seize-assisted fuzzing technique, which leverages the token-passing mechanism of BACnet for improved fuzzing efficiency. The token-seize-assisted fuzzing technique proves highly effective in uncov-ering vulnerabilities caused by the misuse of implicitly reserved fields. We identify this issue as a common vulnerability affecting both BACnet and KNX, another major BAS protocol. Notably, the BACnet Association (ASHRAE) confirmed the presence of a protocol-level token-seize vulnerability, further validating the significance of this finding. We evaluated BACsFuzz on 15 BAC-net and 5 KNX implementations from leading manufacturers, including Siemens, Honeywell, and Johnson Controls. BACs-Fuzz improves fuzzing throughput by 272.49% to 776.01%over state-of-the-art (SOTA) methods. In total, 26 vulnerabilities were uncovered—18 in BACnet and 8 in KNX—each related to implicitly reserved fields. Of these, 24 vulnerabilities were confirmed by manufacturers, with 9 assigned CVEs.

I. Introduction

Building Automation Systems (BASs) are integrated networks of hardware and software designed to monitor, control, and automate various building systems. These systems manage key functions such as HVAC&R, lighting, and security. As an open standard, Building Automation and Control Networks (BACnet) facilitates seamless integration and interoperability among diverse devices—an essential feature in the rapidly growing smart building technology market. According to a ReportLinker forecast [1], the global BAS market is projected to reach $277 billion by 2027, driven by the increasing demand for energy efficiency, safety, and comfort. BACnet International [2] reports that BACnet holds a 77% global market share, with 1,539 official manufacturers worldwide [3], underscoring its critical role in this evolving sector.

While many known threats to BACnet stem from the lack of security measures such as encryption and integrity in the specification, our study has identified a class of implicitly reserved fields that may introduce software and system security issues. Although the protocol carefully defines field names and their corresponding lengths, their practical utilization may not fully leverage the theoretical maximum capacities. For instance, the `Actual Window Size` field (as specified on *Page 847* of the BACnet standard [4]) is designed to occupy 8 bits, yet its values are limited to the range between `0` and `0x7F` (127), implicitly leaving the upper range unused (i.e., up to `0xFF` or 255). If manufacturers fail to implement proper input validation, inputs beyond the expected range could trigger undefined behavior, resulting in data corruption, system crashes, or arbitrary code execution by attackers.

Fuzzing is an intuitive and widely adopted approach for uncovering BAS vulnerabilities arising from poor input validation. However, existing techniques [5]—relying on random mutations and neglecting BAS network transmission overhead—often suffer from inefficiency and limited effectiveness. We observe that BACnet devices commonly use the Master-Slave/Token-Passing (MS/TP) link layer, which suffers from extremely low data throughput. The low throughput exacerbates the inefficiencies of fuzzing BACnet. As a result, even well-crafted mutated inputs experience prolonged transmission times, significantly reducing overall fuzzing efficiency.

To address the inefficiency of fuzzing BACnet, we conducted a protocol-level analysis of MS/TP and identified the token-passing mechanism as the primary bottleneck. At one time, only a single token exists on the network, and all devices must compete for it. Only the device holding the token is allowed to transmit. As a result, a fuzzer must acquire the token prior to transmitting mutated packets to the target, substantially limiting execution speed. Further analysis identified a critical protocol behavior: if the token holder detects unauthorized

*Corresponding author: Prof. Zhen Ling of Southeast University, China.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230794
www.ndss-symposium.org

transmissions, it voluntarily relinquishes the token to preserve network integrity. We utilize this behavior and develop the token-seize-assisted fuzzing technique: A fuzzer injects unauthorized frames, forces legitimate token holders to abandon control, prevents other devices from regenerating the token, and effectively monopolizes network access. This enables continuous and uninterrupted transmission of test cases and boosts the efficiency of fuzzing implicitly reserved fields.

In this paper, we propose BACsFuzz, a token-seize-assisted fuzzer to explore the impact of implicitly reserved fields. By exploiting BACnet's unique network protocol behavior, BACsFuzz generates and injects test cases targeting these fields. Compared to existing SOTA methods, BACsFuzz improves fuzzing throughput by at least 272.49% and up to 776.01%. We also extend BACsFuzz to KNX [6], another major BAS protocol. Since the token-passing mechanism is unique to BACnet, the token-seize-assisted fuzzer cannot be applied to KNX. For KNX, we focus on validating the impact of implicitly reserved fields. We evaluate BACsFuzz on 15 BACnet and 5 KNX implementations from leading manufacturers, including Siemens, Honeywell, and Johnson Controls. We uncovered 26 vulnerabilities: 18 in BACnet and 8 in KNX, all linked to implicitly reserved fields. Manufacturers confirmed 24 vulnerabilities, and 9 were assigned CVEs. This finding highlights that the misuse of implicitly reserved fields is a general issue affecting major BAS protocols.

We further demonstrate the impact of these vulnerabilities through three representative attack case studies. The first case involves the PPM-1U32.BPF I/O module, where mutating the `SLEN` field causes the device to forward attacker-controlled messages to downstream devices, enabling command injection via a trusted intermediary. The second case targets the BASRT-B router, where manipulating the `Service Choice` field results in a persistent Denial-of-Service (DoS) that disrupts communication between BACnet/IP and MS/TP. The third case examines a BACnet Secure Connect (BACnet/SC)-enabled management system. Although BACnet/SC secures BACnet traffic using certificate-based encryption over TLS-encrypted WebSockets, we demonstrate that malformed MS/TP messages can traverse legacy paths and crash the management software—without requiring a valid certificate. These vulnerabilities have serious consequences: attackers could manipulate physical security controls, router failures may delay critical gas or smoke alerts, and DoS attacks on management software could reduce system visibility. These findings emphasize the need to secure both cryptographic channels and legacy, unauthenticated trust boundaries.

Our major contributions can be summarized as follows:

- **Novel Attack Surface with Domain-Specific Insights.** Our work highlights a previously underexplored class of vulnerabilities in the BACnet protocol, stemming from the misuse of implicitly reserved fields. These fields, although defined with specific lengths in the BACnet specification, are often underutilized or improperly handled by manufacturers. The misuse of implicitly reserved fields is a general issue affecting BACnet as well as KNX, two major BAS protocols.
- **Advancing Protocol Fuzzing for BAS.** We propose BACsFuzz [1], the first behavior-driven fuzzer tailored for BACnet security analysis. We introduce three key techniques: an implicitly reserved field-based mutation policy, a token-seize-assisted throughput optimization, and a byte stream format-oriented field consistency verification, enabling efficient and effective security testing of BACnet implementations.
- **Discovery of Real-World Vulnerabilities.** Our evaluation of BACsFuzz on 20 BAS implementations uncovered 26 unknown vulnerabilities, 24 confirmed by manufacturers, including 9 CVEs. The token-seize vulnerability was acknowledged by ASHRAE as a protocol-level flaw, while Honeywell underscored the broader impact: "This vulnerability is part of a standardized protocol used by billions of devices."

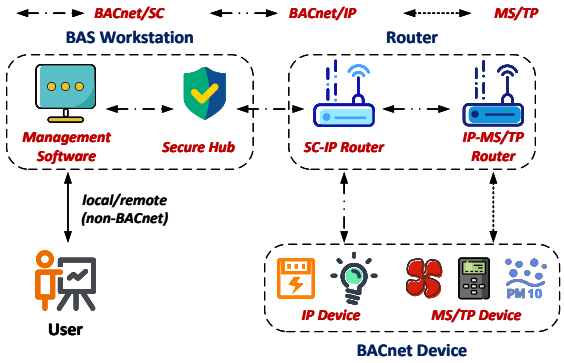

Fig. 1: Typical BACnet Network Topology.

II. Background

A. BACnet Network Topology

BACnet supports a flexible network topology that can integrate diverse devices. Figure 1 illustrates the topology of a typical BACnet network, which consists of a BAS workstation, routers, and various BACnet devices:

- **BAS Workstation:** The workstation runs the central management software responsible for configuring, monitoring, and diagnosing BACnet field devices. It serves as the operator's interface and, in secure deployments, communicates with the network over BACnet/SC through the Secure Hub, which acts as a message broker for encrypted communication.
- **Router:** BACnet routers act as protocol bridges between heterogeneous network segments. Specifically, SC-IP routers translate secure WebSocket-based BACnet/SC traffic into BACnet/IP, while IP-MS/TP routers translate BACnet/IP traffic into MS/TP.
- **BACnet Devices:** These devices include sensors, actuators, and controllers that implement BAS functions over different data link layers, most commonly BACnet/IP and MS/TP. These field devices perform automation tasks in response

[1]https://github.com/isZzzz/BACsFuzz.

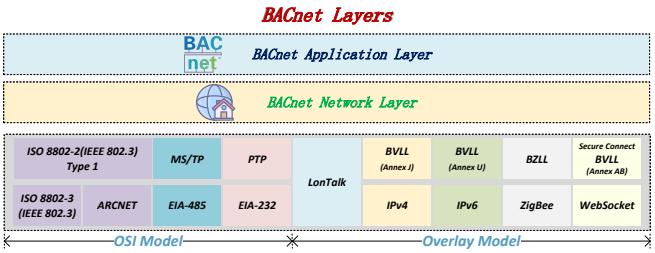

Fig. 2: BACnet Architecture. (Note: The concept of overlay is introduced by the authors and does not exist in the specification.)

to control commands from the management software and transmit operational data through the network.

This represents the most recent secure BACnet network topology enabled by BACnet/SC. However, in real-world deployments, most systems include only the management software, IP-MS/TP routers, and BACnet devices—without implementing the full secure infrastructure.

B. BACnet Protocol Architecture

BACnet adopts a four-layer architecture consisting of the Physical, Data Link, Network, and Application layers, and also can work as an overlay over other protocols as illustrated in Figure 2.. The color shading indicates typical protocol groupings across layers, tailored for different deployment scenarios. To support diverse networking environments, BACnet operates atop existing communication technologies through protocol overlays. For example, BACnet over IPv4 is referred to as BACnet/IP, over IPv6 as BACnet/IPv6, over ZigBee as BACnet/ZigBee, and over secure WebSocket as BACnet/SC.

- **The Application Layer** provides the communication services required by the applications to perform their functions, such as monitoring and control of HVAC&R.
- **The Network Layer** provides the translation of global addresses to local addresses, routes messages through one or more networks, and accommodates differences in network types and maximum permissible message sizes.
- **The Data Link Layer** organizes data into frames and regulates access to the medium. BACnet supports multiple protocols, with MS/TP being one of the most commonly used for inter-device communication [7].
- **The Physical Layer** provides a means of connecting the devices and transmitting the signals that convey data.

BACnet utilizes Protocol Data Units (PDUs) for network communication, encapsulating data and control information for transmission across various layers:

- **Application Protocol Data Unit (APDU):** Located at the application layer, the APDU manages application-specific messages, such as requests to read or write device attributes, facilitating communication between devices.
- **Network Protocol Data Unit (NPDU):** Operating at the network layer, the NPDU encapsulates the APDU and incorporates addressing and routing information, ensuring proper delivery across local and wide-area networks.
- **Link Protocol Data Unit (LPDU):** At the data link layer, the LPDU encapsulates the NPDU, managing framing and transmission across various physical media. In BACnet/IP, the BACnet Virtual Link Control (BVLC) handles this role over IP networks, while in MS/TP, the LPDU manages physical connections, framing, and error control.
- **Physical Protocol Data Unit (PPDU):** Situated at the physical layer, the PPDU encapsulates the LPDU and represents the actual transmission signals, whether electrical or wireless, that carry the higher-layer units through the network's physical media to complete data transmission.

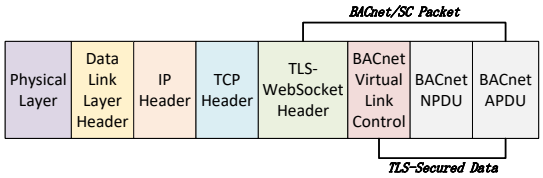

Fig. 3: BACnet/SC Packet Structure.

Layering Ambiguity in the Specification: The classification of lower-layer protocols in the BACnet standard is ambiguous and potentially confusing. For example, IPv4 and WebSocket are designated as physical layer protocols for BACnet/IP and BACnet/SC, respectively. This classification likely reflects the BACnet packet structure. As shown in Figure 3—based on our experiments—BACnet/SC encapsulates a TLS-encrypted BACnet packet within a WebSocket payload. From BACnet's perspective, the WebSocket layer lies beneath the virtual link control layer of BACnet/SC (i.e., its data link layer), and is therefore treated as physical. A similar rationale applies to classifying IPv4 under BACnet/IP. We clarify the architecture in Figure 2 and treat BACnet/IP, BACnet/IPv6, BACnet/ZigBee, BACnet/SC and LonTalk as overlay protocols since these protocols run BACnet over underlay networks such as IP. We have submitted a request to ASHRAE to clarify this layering ambiguity.

III. Motivation and Challenges

In this section, we outline the motivations, threat model, and challenges. We also provide a brief summary of our solutions, with detailed discussions deferred to §IV.

A. Motivation

Firmware analysis can reveal security vulnerabilities, but is often impractical for mainstream BAS devices due to proprietary constraints and hardware protections. Therefore, we adopt black-box network fuzzing, which remains broadly applicable. We observe that existing BACnet fuzzers exhibit significant limitations in efficiency. As described in §II-A, management software typically communicates with field devices through BACnet routers that convert BACnet/IP traffic to MS/TP. For ease of deployment, most fuzzers inject mutated BACnet/IP packets at the management software, relying on

routers to forward them downstream. However, this architecture introduces two major bottlenecks: malformed packets may be dropped by intermediates, and the MS/TP link imposes severe throughput constraints. These issues significantly hinder fuzzing coverage and effectiveness. To date, no solution has effectively addressed these limitations. The most relevant prior effort, BASE [5], functions as a black-box random fuzzer. Although it heuristically identifies sensitive fields, its effectiveness is constrained by the semantic complexity of BACnet. Ill-formed packets are often rejected silently, resulting in low yield and long discovery times. Furthermore, while BASE uncovered vulnerabilities in BAS devices, it lacked analysis of the underlying protocol-level semantics that caused them. In contrast, our work advances BACnet fuzzing by introducing a semantically guided approach tailored to the protocol specification, enabling discovery of deeper and previously overlooked flaws.

B. Threat Model

BACnet-based building automation systems are widely deployed in large-scale environments such as offices, hotels, and campuses. In such settings, hundreds to thousands of field devices—including sensors, actuators, and controllers—are installed throughout buildings and often remain operational for decades without removal or rewiring. These devices are typically located in inaccessible areas (e.g., ceilings, mechanical rooms, rooftops), making it impractical to isolate or test them individually in situ. To this end, our experimental setup involves multiple concurrently connected devices on a shared MS/TP network, mirroring real-world BAS architectures and operating conditions. While we recognize the value of single-device lab setups for component-level testing—and agree they are preferable when feasible—our strategy is necessary for penetration testing and offers greater flexibility.

We assume that the fuzzer has access to the MS/TP network—a realistic condition in adversarial scenarios. While most field devices are physically inaccessible, MS/TP interfaces are often exposed in semi-public locations (e.g., behind wall-mounted panels in hotel rooms or inside breakout boxes in conference facilities). An attacker with brief physical access to such points could connect to the network and inject malformed traffic, potentially affecting devices across zones. Importantly, this setup is also critical for evaluating BACnet/SC deployments. Complete systems feature end-to-end communication between the management software and field devices, traversing components such as Secure Hubs, SC-IP routers, IP-MS/TP routers, and MS/TP endpoints. Single-device lab setups cannot replicate this layered architecture or reveal end-to-end vulnerabilities. Although BACnet/SC encrypts BACnet/IP traffic via WebSocket tunnels, the encrypted messages are ultimately routed to field devices over MS/TP. We leverage this architecture by injecting malformed packets at field devices so that these packets can traverse the router chain in reverse and encrypted channels, and reach BACnet/SC endpoints, bypassing encryption with no need of a valid certificate and exposing vulnerable components within BACnet/SC.

BACnet has a strict requirement for a secure network and does not aim to protect against attackers already present on the same network as the device. However, prior researchh [8], [9], [10] has demonstrated that once an attacker gains access to a BAS network, they can exploit network-level vulnerabilities to control devices using normal commands. The vulnerabilities uncovered through our fuzzing efforts are system- and implementation-level flaws that persist even in security deployments, thereby significantly complementing existing work.

C. Challenges and Solutions

We now outline the challenges (**C**) we encountered and the solutions (**S**) we devised to overcome them.

(C-1) Complexity in BACnet Message Field Mutation. BACnet messages typically consist of over 30 fields. Assuming each field's data type is a byte (with a value range of 0-`0xFF`), the mutation space for an entire message can reach 256^{30}. Traversing all possible mutations across all fields is computationally infeasible, necessitating the identification of key fields for targeted mutation. Existing mutation strategies primarily focus on well-defined fields, such as control commands [11], [12] and syntax-related fields [5], which directly affect protocol functionality. However, these approaches often implicitly assume that field value definitions are clear and complete, overlooking vulnerabilities arising from unclear and incomplete value definitions. Such gaps can lead to implementation errors that existing methods fail to detect.

(S-1) Implicitly Reserved Field-Based Mutation Policy (§IV-A) To address the challenges arising from protocol design and implementation inconsistencies, we introduce a novel perspective on field mutation by focusing on implicitly reserved fields—fields that are not explicitly significant in protocol functionality but exhibit undefined or poorly specified value ranges. Such fields can act as potential attack vectors, as their undefined nature often leads to implementation inconsistencies or errors. Our strategy systematically identifies these fields and develops targeted test cases to explore their impact. To facilitate testing, we leverage specialized APDUs for error handling, enabling controlled violation scenarios to probe the protocol's error-handling mechanisms. By triggering these APDUs through precise mutations, we can observe deviations from expected behavior, thereby revealing weaknesses stemming from improperly handled or incomplete value definitions.

(C-2) Low Data Throughput in Bus Networks. Higher throughput—generating and testing more inputs per unit time—is crucial for timely security assessments, as it increases the likelihood of rapidly discovering vulnerabilities. However, our analysis reveals that despite the fuzzer transmitting a high volume of mutated packets at a rapid pace, only a small fraction successfully reaches the bus network. Traditional bus-based Industrial Control System (ICS) protocols, such as Modbus, can process hundreds of packets per second [13], while CAN can handle thousands [14]. In stark contrast, the MS/TP protocol processes only a few packets per second, making it significantly slower. This throughput limitation becomes increasingly pronounced as the number of devices connected to

the bus network increases, further exacerbating the bottleneck. These challenges underscore the urgent need for a comprehensive analysis of the factors constraining throughput.

(S-2) Token-Seize-Assisted Throughput Optimization(§IV-B) To address low throughput in bus networks, we analyzed the MS/TP protocol and identified the token-passing mechanism as the primary bottleneck. This mechanism enforces a single-token policy, where only one token exists on the network at any given time. The token-holding device is the only one permitted to initiate requests, while all other devices must compete for the token, causing throughput limitations. Our analysis uncovered an exploitable protocol behavior: If the token-holding device detects network activity from another device—such as when a malicious device transmits data without holding the token—it erroneously assumes the presence of multiple tokens and relinquishes its own token to maintain network integrity. By exploiting this behavior, the fuzzer can seize and retain the token, monopolizing network resources for continuous and rapid data transmission.

(C-3) Black-Box Nature in Monitoring Fuzzing Status. Effective fuzzing requires precise monitoring of the target device's execution state for reliable exception detection. However, the diversity and proprietary nature of hardware and firmware introduce variability in responses to erroneous inputs, complicating robust monitoring. Traditional techniques [11], [12] often rely on device liveness checks to monitor DoS. While effective for identifying DoS, these methods fail to detect logical vulnerabilities that violate protocol semantics.

(S-3) Byte Stream Format-Oriented Field Consistency Verification(§IV-C) To address the limitations of traditional monitoring approaches, we analyze the target device's response packets directly, leveraging the unique characteristics of the BACnet message format. BACnet messages are transmitted as a continuous byte stream, where the position and value of fields are critical for correct parsing and interpretation. Our method defines the expected positions and valid value ranges of fields in response packets based on the BACnet specification. Comparing actual field values to their expected ranges reveals flaws, such as violations of protocol semantics.

IV. BACsFuzz Design

As shown in Figure 4, BACsFuzz is designed to accelerate fuzzing and evaluate the impact of implicitly reserved fields on devices. First, BACsFuzz employs an implicitly reserved field-based mutation policy (referred to as the mutation policy) to optimize input generation by identifying such fields and linking them to protocol-defined error-handling mechanisms (§IV-A). Once inputs are generated, BACsFuzz leverages token-seize-assisted throughput optimization (referred to as throughput optimization). This is achieved by bypassing the token-passing mechanism (§IV-B), effectively exploiting key protocol behaviors and significantly improving throughput. This optimization allows BACsFuzz to test various mutation strategies efficiently, revealing that implicitly reserved fields can lead to major security issues in BACnet. Finally, BACsFuzz incorporates byte stream format-oriented field consistency verification (referred to as consistency verification), which defines expected response patterns based on the protocol specification (§IV-C). Deviations from these patterns are flagged as anomalies, enabling vulnerability detection.

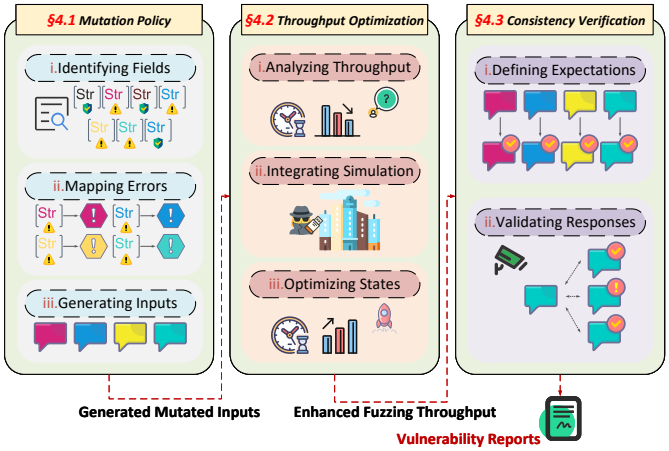

Fig. 4: Architecture of BACsFuzz.

A. Mutation Policy

The mutation policy explores the BACnet field mutation space. By focusing on implicitly reserved fields—those under- or partially defined—it targets implementation ambiguities and evaluates the robustness of error-handling mechanisms. Specifically, these implicitly reserved fields are characterized by incomplete or undefined value ranges in the BACnet specification. Such under-specification can lead to boundary checks being overlooked, potentially exposing security vulnerabilities. To illustrate how these fields are classified, consider a one-byte field that can be divided into three categories:

- **Fully Defined Fields:** These fields utilize the entire value range (0-`0xFF`). For example, as shown in Table I, the `Hop Count` field, specified on *Page 57* of the BACnet specification [4] in Section "*6.2.3 Hop Count*", is designed to occupy one byte (8 bits). The specification restricts its values to the range of 0 to `0xFF`.
- **Explicitly Reserved Fields:** These fields explicitly define reserved values (e.g., `0x80-0xFF`) alongside currently used values (e.g., 0-`0x7F`), thus covering the entire value range. For instance, as specified in Section "*6.2.4 Network Layer Message Type*" on *Page 57* of the BACnet specification [4], the `Message Type` field is one octet and specifies all possible values, including currently used values (0-`0x09`, `0x12`, `0x13`) and reserved ranges (`0x0A-0x11`, `0x14-0xFF`).
- **Implicitly Reserved Fields:** These fields define only a portion of the value range (e.g., 0-`0x7F`), leaving the remainder (e.g., `0x80-0xFF`) undefined. For example, the `SLEN` field in Section "*6.2.2 Network Layer Protocol Control Information*" (*Page 55* [4]) occupies one octet (0-`0xFF`) but only accepts the values `0x01`, `0x02`, `0x03`, and `0x06`. Fields with completely undefined value ranges also fall into this category. For instance, the `DADR` and

TABLE I: Partial Field Analysis Results for NPDU.

Field	Len.	Range	Spec. Val.	Category
Version	8	[0x0,0xFF]	0x1	Implicitly Reserved
NSDU	1	[0x0,0x1]	[0x0,0x1]	Fully Defined
Reserved1	1	[0x0,0x1]	0x0	Explicitly Reserved
Des. Spec.	1	[0x0,0x1]	[0x0,0x1]	Fully Defined
Reserved2	1	[0x0,0x1]	0x0	Explicitly Reserved
Src. Spec.	1	[0x0,0x1]	[0x0,0x1]	Fully Defined
Exp. Reply	1	[0x0,0x1]	[0x0,0x1]	Fully Defined
Priority	2	[0x0,0x3]	[0x0,0x3]	Fully Defined
DNET	16	[0x0,0xFFFF]	[0x1,0xFFFF]	Implicitly Reserved
DLEN	8	[0x0,0xFF]	[0x0,0x3],0x06,0x07	Implicitly Reserved
DADR	/	/	/	Implicitly Reserved
SNET	16	[0x0,0xFFFF]	[0x1,0xFFFE]	Implicitly Reserved
SLEN	8	[0x0,0xFF]	[0x1,0x3],0x06	Implicitly Reserved
SADR	/	/	/	Implicitly Reserved
Hop Count	8	[0x0,0xFF]	[0x0,0xFF]	Fully Defined
Message Type	8	[0x0,0xFF]	[0x0,0xFF]	Fully Defined
Vendor ID	16	[0x0,0xFFFF]	/	Implicitly Reserved

"/" : not defined in the protocol specification.

TABLE II: Some Error Codes in Protocol Implementation.

Type	Error Field	Error Code	Description
BACnet-Reject-PDU	reject-reason	1	buffer-overflow
		2	inconsistent-parameters
		3	invalid-parameter-data-type
		4	invalid-tag
		5	missing-required-parameter
		6	parameter-out-of-range
		7	too-many-arguments
		8	undefined-enumeration
		9	unrecognized-service
		10	invalid-data-encoding
BACnet-Abort-PDU	abort-reason	4	segmentation-not-supported
		7	window-size-out-of-range
		11	apdu-too-long

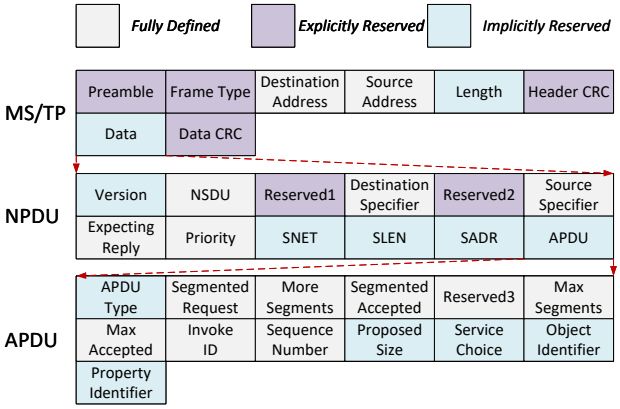

Fig. 5: Field Identification in `ReadProperty` Requests.

`SADR` fields in Section "*6.2.2.2 DADR and SADR Encoding*" (*Page 56* [4]) are not specified with any value information.

To mitigate potential issues, BACnet introduces redundancy mechanisms, including a general error-handling framework to manage invalid field values. However, device manufacturers may not fully or correctly implement these mechanisms due to misinterpretation of the specification or oversight of undefined value ranges, leading to unexpected device behavior or security vulnerabilities. Consequently, assessing the completeness of error-handling implementations across devices is critical. The specification defines several APDUs, including `BACnet-Reject-PDU`, `BACnet-Abort-PDU`, and `BACnet-Error-PDU`, which notify the sender when a request is invalid. These APDUs serve as observable indicators, revealing whether a device has internal mechanisms for managing such errors. Each APDU includes an associated error code (as shown in Table II) that specifies the nature of the error, but not all fields can trigger every type; different fields activate only specific subsets of these errors.

To identify implicitly reserved fields, we employ a Large Language Model (LLM) to extract structural field definitions, including length, value ranges, and reserved indicators. We manually verify the LLM-generated results for both accuracy and completeness, achieving 95.77% correctness (Figure 10), and further analyze misclassified cases. Details of the LLM-based extraction methodology and its evaluation are provided in Appendix A. For each implicitly reserved field, we first identify the specific error types it can trigger based on the specification. If a field is linked to one or more error types, we deliberately mutate its value to provoke these errors and observe how the device responds. For fields with no associated error types, we inject values outside their defined range to assess how the device processes undefined inputs. Since vulnerabilities can emerge from interactions among multiple fields, testing them in isolation may overlook such compound effects. To address this, we iteratively mutate combinations of fields and monitor device behavior under these compound input conditions. Although devices typically return only a single error code—often corresponding to the first fault encountered—this approach still enables the detection of inter-field vulnerabilities that would otherwise remain hidden.

For example, as shown in Figure 5, the `ReadProperty` request must include two parameters: `Object Identifier` and `Property Identifier`. The correct data type for `Object Identifier` is `BACnetObjectIdentifier`. To evaluate the device's error-handling for incorrect data types or missing parameters, we design two cases: First, we construct a `ReadProperty` request where the `Object Identifier` is erroneously set to the `BACnetPropertyIdentifier`. We expect the device to return a `BACnet-Reject-PDU` with `error code = 3` (*invalid-parameter-data-type*). Failure to do so indicates a weakness in handling parameters with incorrect types. Second, we construct a `ReadProperty` request with the `Object Identifier` parameter deliberately omitted. We expect the device to return a `BACnet-Reject-PDU` with `error code = 5` (*missing-required-parameter*). Failure to do so reveals a deficiency in handling missing required parameters.

B. Throughput Optimization

The throughput optimization addresses performance bottlenecks inherent in BACnet's token-passing mechanism. By eliminating inefficiencies like token waiting and idle states, it accelerates data transmission for rapid execution of fuzzing test cases. Specifically, the MS/TP data link layer, based on the EIA-485 physical standard, operates under a

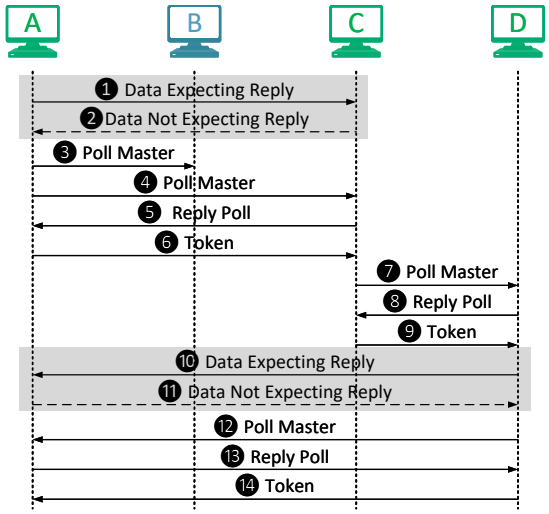

Fig. 6: Token-Passing Process.

Master-Slave architecture with a token-passing mechanism to regulate network access. The MS/TP network supports up to 255 devices, with master nodes [2] assigned addresses in the range 0 to `0x7F`, and slave nodes from `0x80` to `0xFF`. The core feature of this mechanism is the *token*, a unique privilege identifier that ensures fair access by allowing only the token-holding master node to initiate data transmissions. A token-holding master node may transmit up to `0xFF` data frames before passing the token to the next master node. Both master and slave nodes must transmit data frames in response to requests from master nodes. Since slave nodes are not permitted to hold the token, MS/TP allows them to reply immediately upon receiving a request. Likewise, master nodes may respond without needing to reacquire the token.

MS/TP defines five standard frame types. Two frames, `BACnet Data Expecting Reply` and `BACnet Data Not Expecting Reply`, are used for data transmission, while the other frames are essential for network coordination. The token-holding master node determines the next token recipient by transmitting a `Token`. After completing data transmission, the master sends a `Poll For Master` to identify the next eligible master node. Upon receiving a `Reply To Poll For Master` from another master node, the token is transferred to that node. Consider a scenario in an MS/TP network with master nodes `A`, `C`, and `D`, as illustrated in Figure 6. Node `A` sends a request to node `C` while holding the token (Step 1). Node `C`, although not holding the token, can respond to the request (Step 2). After completing its data transmission, node `A` sends a `Poll For Master` to node `B` to determine if it requires the token (Step 3). If node `B` is absent and does not respond, node `A` continues polling subsequent addresses until reaching node `C` (Step 4). Even if node `C` has no data to transmit, it must respond with a `Reply To Poll For Master`, after which node `A` passes the token to it (Steps 5-6). Node `C` then polls the next node `D` (Steps 7-9). After node `D` finishes its data transmission (Steps 10-11), it continues polling until reaching address `0x7F`, then wraps around to start polling from address 0. This cycle repeats until node `A` responds with a `Reply To Poll For Master`, regaining the token (Steps 12-14). While the token-passing mechanism ensures fair access, it introduces throughput limitations. Even in this simplified scenario with only three master nodes, the mechanism introduces significant delays—only 4 out of 14 steps involve actual data transmission (Steps 1, 2, 10, and 11). In real-world networks, where up to 127 master nodes can be present, these delays are exacerbated, leading to severe throughput inefficiencies.

The token-passing mechanism ensures that only one token exists on the bus network at any given time, preventing collisions and ensuring orderly communication. Each node operates under a Master Node Finite State Machine (MNFSM), which governs its behavior and status, as shown in Figure 7. To maintain network stability, the MNFSM includes a *SilenceTimer* to monitor periods of network silence. According to the specification, if a master node without the token detects prolonged silence, it assumes the token is lost and autonomously generates a new one. To prevent multiple tokens from existing simultaneously, the MNFSM ensures that a token-holding master node relinquishes its token if it detects data transfers from other nodes (as data transfer can only occur when the token is held). However, if a (malicious) node continuously sends data, it can force the token-holding node to relinquish its token. As per the specification, nodes without the token assume it is still in use and continue waiting for it to be passed. Due to the (malicious) node's persistent data transmission, the token is never passed to the waiting nodes, leaving them unable to generate a new token or participate in normal communication. Appendix B provides detailed information on the impacted node.

This protocol behavior presents unique opportunities for fuzzing. In this context, the goal of fuzzing is to continuously seize the token and inject malformed inputs. By exploiting this behavior, a fuzzer can bypass the token-waiting process without disrupting responses from target devices, thereby ensuring uninterrupted network activity and maximizing throughput. To leverage this, we developed a simulator that bypasses routers to directly interact with MS/TP devices. The simulator optimizes the MNFSM to force token-holding nodes to relinquish their tokens, preventing other nodes from acquiring or generating a new token. As a result, the remaining nodes are locked in a perpetual waiting state, enabling the fuzzer to maximize throughput and significantly improve efficiency.

Step (I): Simulated Device Integration. We first implement a simulated device that directly connects the fuzzer to BACnet nodes. As discussed in §II-A, traditional BACnet communication typically involves routers that translate BACnet/IP packets into MS/TP packets, introducing unnecessary complexity and overhead during fuzzing. By simulating direct MS/TP connections with our fuzzer, we eliminate router dependencies, simplifying the fuzzing process. Specifically, our setup uses open-source projects such as bacnet-stack [15]

[2]The term "*node*" is used to refer to "*device*" in the specification [4]. Therefore, these terms are used interchangeably.

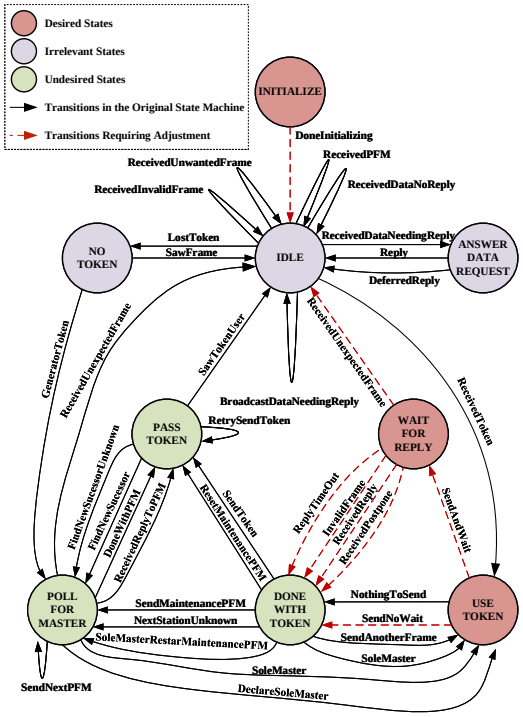

Fig. 7: Master Node Finite State Machine.

and BACpypes [16], installed on a computer to handle BACnet communication across all protocol layers. This allows for comprehensive simulation of BACnet operations. Access to the bus network is established via a USB-to-EIA-485 adapter, which converts USB signals into bus-level signals and allows the fuzzer to function as a legitimate BACnet device.

Step (II): State Machine Optimization. This step enhances the state machine by prioritizing states essential for the fuzzer's operation, thereby improving throughput. The states are organized into three categories:

1) **Desired States.** Desired states are those that the fuzzer must maintain to ensure maximum throughput.
   - *INITIALIZE*: Configures node settings on reset or power-up, establishing parameters for network communication.
   - *USE_TOKEN*: Enables the node to transmit data frames, maintaining token control for uninterrupted transmission.
   - *WAIT_FOR_REPLY*: Pauses operations to receive responses, which are critical for managing expected communication. This state allows the fuzzer to analyze target device feedback, informing subsequent actions.
2) **Irrelevant States.** Irrelevant states do not contribute to the effectiveness of fuzz testing.
   - *IDLE*: Waits for incoming frames when the node lacks the token, determining the next steps based on frame type. Since the fuzzer continuously sends data without needing token possession, this state is redundant.
   - *NO_TOKEN*: Activates when no network activity is detected, potentially generating a new token. As the fuzzer runs continuously without token, this state is unnecessary.
   - *ANSWER_DATA_REQUEST*: Responds to frames that require a reply to maintain network communication. Since the fuzzer initiates requests requiring responses from other devices, this state is redundant.
3) **Undesired States.** Undesired states hinder the fuzzing process and should be avoided to maintain effectiveness.
   - *POLL_FOR_MASTER*: Monitors responses to node polling to identify or verify nodes on the network. Minimizing this state helps prevent network resets or token reassignments, thereby stabilizing token control and improving fuzzing performance.
   - *DONE_WITH_TOKEN*: Decides whether to pass the token or initiate a poll. Eliminating this state aligns with the goal of maintaining continuous token possession, thereby enhancing fuzzer performance.
   - *PASS_TOKEN*: Manages token passing to the next node to ensure equitable network access. Removing this state guarantees uninterrupted token control by the fuzzer.

Adjustments to the state machine are critical for efficient data transfer. Directly transitioning from *INITIALIZE* to *USE_TOKEN* bypasses the initial *IDLE* state, expediting the initiation of fuzzing activities. In the *USE_TOKEN* state, the node either remains or transitions to *WAIT_FOR_REPLY*, depending on whether a reply is required. After receiving a reply, the node immediately returns to *USE_TOKEN*, maintaining seamless and uninterrupted operation.

C. Consistency Verification

The consistency verification method ensures the accuracy of protocol implementations by systematically analyzing device responses. It derives expected field positions and values from the specification, flagging any deviations as potential vulnerabilities. Specifically, the BACnet protocol utilizes a byte stream format for message transmission. Unlike structured formats such as JSON or XML—where fields are explicitly separated and identifiable using tags or magic words—BACnet messages adopt a continuous byte stream format. In this format, the position and value of each field are critical for accurate parsing and interpretation. Modifications, deletions, or insertions of fields can significantly alter the interpretation of subsequent bytes. We observe that errors or discrepancies in field ordering can propagate through the byte stream, disrupting downstream fields, as BACnet messages are parsed sequentially from start to end. For example, an error in the first byte may compromise the interpretation of the entire message, while an error in the penultimate byte can affect the final byte, potentially triggering a cascade of failures. This error propagation highlights the importance of strict adherence to byte positioning and field values. If the protocol implementation is flawed, such modifications can initiate a chain reaction, leading to parsing failures or deviations from expected behavior.

To leverage this feature, we define the expected positions of fields (e.g., identifiers and sequence numbers) in response packets, emphasizing fields that must maintain consistent

values between requests and responses. For each field, a valid value range is strictly defined based on the protocol specification. By comparing the actual values at these positions in the response to the expected ranges, any deviations are flagged as potential implementation errors or non-compliance. If no response is received, it is classified as a potential DoS. For instance, the specification requires the `Version` field to be `0x01`, and the `APDU Type` field to fall between 0 and `0x07`. Discrepancies in these values may indicate vulnerabilities or errors in the protocol's implementation. To validate an exception, we replay the exploit packet multiple times and observe the resulting behavior. If anomalous responses persist across repeated attempts, the exception is confirmed as a potential vulnerability.

V. Evaluation

In this section, we introduce the experimental setup and present the evaluation to address the following five questions:

- ***RQ1:*** Can BACsFuzz effectively uncover vulnerabilities in BAS implementations?
- ***RQ2:*** To what extent does BACsFuzz enhance fuzzing throughput compared to SOTA approaches?
- ***RQ3:*** What is the contribution of each BACsFuzz component to efficient vulnerability discovery?
- ***RQ4:*** How does BACsFuzz perform in terms of detection capability compared to existing tools?
- ***RQ5:*** Are the vulnerabilities caused by implicitly reserved fields generalizable to other BAS protocols?

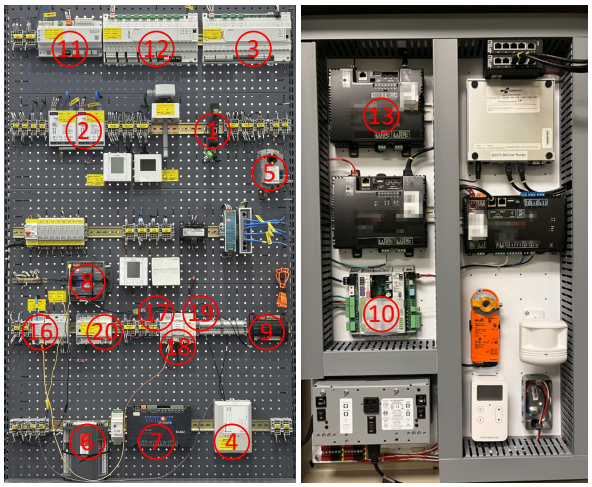

Fig. 8: BACnet Devices Used in Our Experiments.

A. Experiment Setup

We implement a prototype of BACsFuzz using *bacnet-stack* [15], *BACpypes* [16], and *Misty* [17], with the latter serving as a bridge between *bacnet-stack* and *BACpypes* to facilitate seamless integration. Our implementation follows the ISO 16484-5 specification [4] for BACnet. For KNX, we develop a generation-based fuzzer grounded in the KNX Standard v3.0.0 [18]. Field analysis is supported by the ChatGPT Team as the underlying LLM. Our setup includes an Ubuntu workstation with a 2.80 GHz Core i7 CPU, 16 GB of RAM, and a USB-to-EIA-485 adapter to interact with field devices. We evaluate 20 widely deployed BAS devices from 9 leading manufacturers—6 of which rank among the global top 10—providing representative coverage of the mainstream BAS market [19]. Each device was fuzzed for 10 hours while monitoring for potential vulnerabilities (see Figure 8).

B. Vulnerability Detection in Devices (RQ1)

After analyzing and deduplicating the observed anomalies, BACsFuzz successfully uncovered 26 previously unknown vulnerabilities across 20 BAS devices, as summarized in Table III. Among these, 8 vulnerabilities were found in 5 KNX devices (see §V-F), demonstrating the generalizability of our approach to other BAS protocols. For the BACnet evaluation, BACsFuzz identified 18 vulnerabilities across 15 distinct implementations. All findings were responsibly disclosed to the respective manufacturers. Seventeen BACnet vulnerabilities have been confirmed, with nine assigned CVEs: CVE-2024-4292, CVE-2024-4511, CVE-2024-4791, CVE-2024-9787, CVE-2025-0xxx, CVE-2025-0xxx, CVE-2025-24510, CVE-2025-40555, and CVE-2025-40556. The evaluated devices span diverse link layers and device types. To illustrate the security implications of these vulnerabilities, we present three detailed cases (**CS**).

(CS-I) Command Injection Attacks against PPM: PPM-1U32.BPF devices are expansion I/O modules that bridge sensors and actuators in BAS networks. They collect input signals and manage output commands, playing a central role in control logic. Using BACsFuzz, we discover that mutating the `SLEN` field to an out-of-spec value triggers an exception in the PPM. Rather than discarding the malformed message, the PPM constructs and transmits a new packet, using a specific byte in the original APDU as the destination address. Crucially, the forwarded packet is generated by the PPM itself and conforms to BACnet packet formatting. For example, if BACsFuzz (address `0x37`) sends a message to the PPM (`0x01`) with a specific APDU byte set to `0x04`—the address of an alarm actuator—the PPM constructs and forwards a packet containing attacker-controlled data (e.g., `0x567...FF`) to device `0x04`. This enables BACsFuzz to indirectly issue arbitrary commands to downstream devices via a trusted intermediary. Experiments show that the target device accepts the relayed command without validation, as MS/TP lacks peer authentication. This vulnerability could allow an attacker to manipulate the I/O module, potentially disrupting legitimate control flows. For example, a maliciously crafted packet might exploit the PPM's behavior to suppress alarm signals. Such interference with sensor-actuator coordination could result in unauthorized access or delayed critical responses, thereby increasing operational risk.

(CS-II) DoS Attacks against BASRT-B: Following a similar approach, we evaluate the BASRT-B, a router bridging BACnet/IP and MS/TP. When the fuzzer sends a malformed packet,

TABLE III: Vulnerability Detection Result by BACsFuzz.

#	Device Type	Device Model	Manufacturer	Protocol	Firmware Version	Vulnerability	Type
1	Router	BASRT-B	Contemporary Controls	MS/TP & BACnet/IP	2.7.2	✔(V1) ✔(V2) ✔(V3)	DoS DoS DoS
2	Router	HMI1002-ARM	Sunfull Automation	MS/TP & BACnet/IP	2.0.4	✔(V4)	BOF
3	Controller	PXC16.3-UCM.A	Siemens	MS/TP	PAACV3.3 BACnet4.3g	✔(V5) ✔(V6)	DoS Unknown
4	I/O Module	PPM-1U32.BPF	Siemens	MS/TP	Digital PPM V1.00	✔(V7) ✔(V8) ✔(V9)	DoS Unknown CI
5	Controller	ATEC 550-440	Siemens	MS/TP	BZ39 Rev 2.0	✔(V10) ✔(V11)	DoS DoS
6	Controller	Pub6438s	Honeywell	MS/TP	1.00 (build 9b)	✔(V12)	LE
7	I/O Module	VYKON IO-22U	Honeywell	MS/TP	1.2.00	✗	/
8	Controller	VMA1632	Johnson Controls	MS/TP	6.2.0.1054	✗	/
9	Controller	DFM-B800	Delta	MS/TP	†	✔(V13)	Unknown
10	Controller	Zone Controller	Company X	MS/TP	***	✔(V14)	DoS
11	Controller	PXC4.E16	Siemens	BACnet/IP	02.20.152.15	✗	/
12	Controller	PXC36.E.A	Siemens	BACnet/IP	EX36V3.3 BACnet4.3g	✔(V15)	Unknown
13	Router	BACnet Router	Company X	BACnet/IP & BACnet/SC	***	✔(V16)	DoS
14	Software	BMS	Company X	BACnet/SC	***	✔(V17)	DoS
15	Software	Secure Hub	Company X	BACnet/SC	***	✔(V18)	DoS
16	Router	Secure N 146/03	Siemens	KNX IP & KNX TP	V3 & V4	✔(V19) ✔(V20)	Unknown Unknown
17	Interface	Secure N 148/23	Siemens	KNX IP & KNX TP	V4	✔(V21)	Unknown
18	Interface	IPS/S3.1.1	ABB	KNX IP & KNX TP	01	✔(V22)	Unknown
19	Router	BNIPR-00/00.S	GVS	KNX IP & KNX TP	0.1.19	✔(V23) ✔(V24)	DoS LE
20	Interface	BNIP-00/00.S	GVS	KNX IP & KNX TP	1.7.8	✔(V25) ✔(V26)	DoS LE

Note (i) Sensitive information for entries #10, #13, #14, and #15 has been anonymized at Company X's request, in line with their disclosure preferences.
Note (ii) All vulnerabilities, except for V13 and V22 (which are still under investigation), have been confirmed by the respective manufacturers.
Note (iii) "DoS" denotes Denial-of-Service, "BOF" denotes Buffer Overflow, "CI" denotes Command Injection, and "LE" denotes Logic Error. The "Unknown" category refers to vulnerabilities where abnormal behavior was observed during testing, but the underlying cause (e.g., BOF or LE) could not be determined. This is mainly due to the lack of access to device firmware, which prevents in-depth analysis. Furthermore, as of the submission date, the corresponding manufacturers had not yet provided clarifications regarding the root causes.
† The firmware version field was left blank because the information was unavailable or undisclosed.

the router enters a fault state and becomes unresponsive. The only recovery method is a manual power cycle—physically unplugging and reconnecting the device. Until then, the router stops forwarding all traffic, resulting in a persistent DoS. Upon investigation, we find that this vulnerability stems from the implicitly reserved field `Service Choice`. Each device declares supported services by specifying the valid range of this field. Injecting a specific unsupported value triggers the crash. An attacker could exploit this flaw to paralyze routing functionality, severing communication between IP-based systems and MS/TP field devices. For example, safety-critical sensors—such as gas or smoke detectors—communicate via MS/TP and rely on routers to relay alerts to supervisory systems. A router crash would break this channel, potentially delaying emergency responses and increasing safety risks.

(CS-III) DoS Attacks against BACnet/SC: BACsFuzz demonstrates its ability to affect BACnet/SC deployments without a valid certificate. As shown in Figure 1, the BACnet Management Software (BMS) (#14) connects to a Secure Hub (#15), which routes messages through an SC-IP Router (#13) and an IP-MS/TP Router (#1) to reach the Zone Controller (#10), an MS/TP field controller. In normal operation, messages follow the path: BMS → Secure Hub → SC-IP Router → IP-MS/TP Router → Zone Controller. While BACnet/SC secures BACnet/IP traffic via WebSocket tunnels, this protection ends at the IP-MS/TP Router and does not extend into the MS/TP domain. Exploiting this trust model, BACsFuzz injects malformed MS/TP messages with valid LPDU/NPDU frames that propagate upstream, ultimately reaching the BMS without requiring certificate-based authentication. Testing revealed a DoS vulnerability in the BMS triggered by repeated malformed messages targeting an implicitly reserved field. Specifically, the BMS's `Device Object` process crashed after encountering multiple `ArrayIndexOutOfBoundsExceptions`, each caused by malformed inputs. Internally, once a crash counter exceeded a threshold, the process was terminated, disrupting management functionality. Given that the BMS is typically

responsible for device coordination and system monitoring, such a DoS condition can impair supervisory control and visibility. Notably, this attack does not require a valid certificate and bypasses BACnet/SC's intended trust boundary—highlighting a critical gap between encrypted IP channels and the unprotected MS/TP domain. While the vulnerability does not cause an immediate full-system failure, it reveals how adversaries may exploit architectural boundaries to degrade availability in BACnet/SC deployments.

C. Throughput Improvement (RQ2)

To evaluate the effectiveness of BACsFuzz in improving fuzzing throughput, we conducted controlled experiments with seven out of ten devices that support the MS/TP data link layer. Device #1, functioning as a router to establish the experimental environment, was excluded from the experimental goals. Device #9, an MS/TP slave node that does not participate in token-passing, was also excluded. Additionally, Device #10 was reserved for other experiments and thus excluded from this evaluation. For the control group, we employed a method similar to BASE [5], a SOTA tool for fuzzing BACnet. In this setup, mutated BACnet/IP packets were generated by the management software and transmitted to the router (Device #1), which then forwarded them to the target BACnet devices. In contrast, the experimental group utilized our simulated device with an optimized state machine to send packets directly. In BACnet networks with multiple devices, a significant portion of time is often consumed acquiring the token rather than transmitting packets (see §IV-B). To simulate such challenging scenarios, we incrementally increased the number of devices and measured the number of fuzzing packets successfully processed by Device #3 within a fixed time frame based on the responses received. A `ReadProperty` request is sent from the fuzzer to the target device, with the fuzzer pausing to wait for a response before issuing the next request. This request type is chosen because most mutated packets fail to qualify as valid `BACnet-Confirmed-Request-PDU` packets and therefore do not trigger device responses. As a result, the fuzzer frequently waits for a timeout before proceeding to the next iteration, which can significantly distort throughput measurements. By contrast, using a valid `ReadProperty` request ensures a response from the device, enabling a more accurate assessment of throughput. We conduct experiments over three time spans: 5 minutes, 10 minutes, and 30 minutes, resulting in 42 experiments in total (i.e., $7 \times 2 \times 3$). Based on the experimental data, we first examine the throughput degradation observed in BASE due to token seizing and how BACsFuzz mitigates this issue, followed by a quantitative evaluation of the throughput improvement provided by BACsFuzz.

Throughput Degradation in SOTA Approaches. We calculate throughput changes relative to the single-device setup as the number of devices increases, with the results summarized in Table IV. Even in the single-device setup, BASE requires a router for packet delivery, introducing two nodes—the router and the target—which already trigger token-passing. Similarly, when a fuzzer connects directly to the target, the two still form a minimal BAS network where token-passing remains necessary. Nonetheless, thanks to BACsFuzz's strategy, high throughput is preserved even in such minimal configurations. As shown, BASE suffers significant throughput degradation as the number of devices increases—dropping by 4.11% with two devices and by 57.77% with seven devices over a 30-minute period. In contrast, BACsFuzz maintains stable throughput, fluctuating within a ∼2% range. Furthermore, our strategy is essential for practical penetration testing scenarios and enables flexible deployment across diverse BAS environments.

TABLE IV: BASE Throughput Degradation Analysis.

Min.	Type	2 vs.1	3 vs.1	4 vs.1	5 vs.1	6 vs.1	7 vs.1
5 min	BASE	-4.72%	-12.31%	-20.93%	-28.52%	-44.36%	-57.41%
	BACsFuzz	1.50%	-1.56%	0.23%	-0.35%	0.00%	0.02%
10 min	BASE	-3.97%	-12.66%	-25.35%	-31.21%	-48.37%	-58.83%
	BACsFuzz	0.08%	0.04%	0.06%	0.09%	-0.04%	0.08%
30 min	BASE	-4.11%	-13.59%	-19.48%	-29.83%	-47.45%	-57.77%
	BACsFuzz	-1.30%	0.11%	0.13%	0.05%	-1.06%	-0.68%

TABLE V: BACsFuzz Throughput Improvement Analysis.

Min.	Type	1 dev	2 devs	3 devs	4 devs	5 devs	6 devs	7 devs
5 min	BASE	1,357	1,293	1,190	1,073	970	755	578
	BACsFuzz	5,122	5,199	5,042	5,134	5,104	5,122	5,123
	↑	277.45%	302.09%	323.70%	378.47%	426.19%	578.41%	786.33%
10 min	BASE	2,820	2,708	2,463	2,105	1,940	1,456	1,161
	BACsFuzz	10,239	10,247	10,243	10,245	10,248	10,235	10,247
	↑	263.09%	278.40%	315.87%	386.70%	428.25%	602.95%	782.60%
30 min	BASE	8,240	7,901	7,120	6,635	5,782	4,330	3,480
	BACsFuzz	30,693	30,295	30,728	30,732	30,708	30,367	30,485
	↑	272.49%	283.43%	331.57%	363.18%	431.10%	601.32%	776.01%

n dev(s) : number of devices in the MS/TP network.
↑ : throughput improvement of BACsFuzz, compared to BASE.

Throughput Improvement by BACsFuzz. To evaluate the improvement achieved by BACsFuzz, we report the number of exchanged packets in Table V. The comparison demonstrates BACsFuzz 's superior performance in packet delivery. Over a 30-minute trial, BACsFuzz improves throughput by 272.49% in the single-device setup and by as much as 776.01% when scaled to seven devices.

D. Ablation Study (RQ3)

To evaluate BACsFuzz 's impact on vulnerability discovery efficiency, we assessed several strategies: the *Implicitly Reserved Field-Based Mutation Policy* (S_{im}), the *Token-Seize-Assisted Throughput Optimization* (S_{to}), and the *Byte Stream Format-Oriented Field Consistency Verification* (S_{bv}). For comparison, we also included a random mutation strategy (S_{rm}) and a router-based transmission method (S_{rt}). These strategies were combined into different variants, each evaluated for its effectiveness in discovering vulnerabilities. For instance, $S_{im,to}$ represents a combination of S_{im} and S_{to}. Each fuzzer variant incorporated S_{bv} to detect vulnerabilities. A full list of possible variants is provided in Table VI. Each fuzzer variant was executed for 10 hours on the devices listed in Table III. Devices corresponding to vulnerabilities V15-V18 do not support native MS/TP, but can be reached via intermediate routing (e.g., SC-IP and IP-MS/TP routers). To

TABLE VI: Number of Fuzzing Packets Needed to Expose Vulnerabilities (Lower is Better)

Vuln.	$S_{rm,rt,bv}$	$S_{rm,to,bv}$	$S_{im,rt,bv}$	$S_{im,to,bv}$
V1	✗	✗	✗	1,74
V2	28,705	75,264	5,344	6,668
V3	✗	✗	91	53
V4	✗	✗	5,28	288
V5	✗	✗	113,599	26,524
V6	✗	235,133	✗	528
V7	✗	19,612	✗	70
V8	✗	4,928	✗	190
V9	✗	✗	✗	365
V10	✗	✗	✗	2,301
V11	✗	66,614	✗	309
V12	✗	✗	✗	63
V13	✗	780	✗	146
V14	56,189	✗	✗	4,707

✗: the vulnerability is not found.

TABLE VII: Comparison with BASE, AFLnet, and BooFuzz.

	V1	V2	V3	V5	V7	V10	V11	V14	V16
BACsFuzz	174	6,668	53	26,524	70	2,301	309	4,707	146
BASE	✗	8,521	✗	✗	✗	✗	✗	✗	✗
BASE(S_{to})	✗	✗	✗	✗	80,403	✗	366,791	✗	✗
AFLnet	✗	✗	✗	✗	✗	✗	✗	✗	✗
AFLnet (S_{to})	✗	✗	✗	✗	✗	✗	✗	✗	✗
BooFuzz	✗	✗	✗	✗	✗	✗	✗	✗	✗
BooFuzz (S_{to})	✗	16,067	✗	✗	29,700	✗	✗	✗	✗

ensure a fair comparison, S_{to} was only applied to devices with native MS/TP connectivity. Additionally, experiments were conducted in a single-device configuration to prevent multiple devices from influencing the performance of the router-based transmission method. A 70 ms timeout, based on the average 50 ms response latency, was selected to balance responsiveness and fuzzing throughput. The number of packets required to trigger each vulnerability is summarized in Table VI.

The results highlight the effectiveness of BACsFuzz in enhancing vulnerability discovery. The *Implicitly Reserved Field-Based Mutation Policy* (S_{im}) consistently outperforms baseline strategies, uncovering more vulnerabilities with fewer packets. This supports BACsFuzz's hypothesis that implicitly reserved fields are highly vulnerable. Complementing this, the *Token-Seize-Assisted Throughput Optimization* (S_{to}) significantly improves packet delivery speed and increases the likelihood of triggering vulnerabilities by bypassing token-passing delays and enabling continuous packet transmission. However, S_{im} alone is limited by routing restrictions during BACnet/IP to MS/TP conversion. Routers strictly validate NPDU fields such as `DNET`, `DLEN`, and `DADR` to perform address resolution. Mutated packets with malformed NPDU headers are discarded before reaching the target device, even if their payloads are valid. As a result, many effective test cases are lost in transit. By combining S_{im} with S_{to}, BACsFuzz bypasses the router and delivers packets directly to field devices, eliminating routing constraints and enabling unrestricted mutation across the entire packet. This ensures that test cases reach their destination and can execute fully. The synergy between S_{im} and S_{to} is essential for uncovering deep-seated vulnerabilities in constrained BACnet network environments.

E. Comparison with SOTA Fuzzers (RQ4)

We compared BACsFuzz with three most relevant SOTA fuzzers: BASE [5], AFLnet [20], and BooFuzz [21]. BASE is a BAS-specific fuzzer that probes packet structures without prior protocol knowledge. AFLnet is a grey-box fuzzer that extends AFL [22] to support network protocols, relying on source or binary code for coverage guidance. BooFuzz is a black-box fuzzing framework, and Fuzzowski [23] is a BooFuzz-based tool targeting industrial protocols, like BACnet. However, Fuzzowski is still under development.

To ensure a fair comparison, we extended and adapted these tools as follows: (i) Since BASE is not open-source, we re-implemented partial functionality based on its paper, manually specifying packet structures and applying its mutation strategies. (ii) We modified AFLnet to remove its reliance on source code, enabling fuzzing with only response codes. (iii) We followed the Fuzzowski approach to complete BooFuzz. (iv) We enabled BooFuzz's "fullrange" option, which tests all possible field values. (v) Each tool was given tailored inputs: AFLnet used a raw request corpus, and BooFuzz was configured with precise BACnet packet structures. (vi) To overcome BACnet router filtering, we integrated *Token-Seize-Assisted Throughput Optimization* (S_{to}) into all tools, allowing direct interaction with MS/TP devices. While BASE, AFLnet, and BooFuzz are capable of detecting DoS vulnerabilities, they fail to identify non-DoS vulnerabilities. Therefore, our comparison focuses on DoS detection in devices. As shown in Table VII, BASE detected vulnerability V2 with 8,521 packets, while AFLnet and BooFuzz failed to detect vulnerabilities in their original configurations. After incorporating S_{to}, BASE(S_{to}) detected vulnerabilities V7 (80,403) and V11 (366,791), while BooFuzz(S_{to}) identified vulnerabilities V2 (16,067) and V7 (29,700). In comparison, BACsFuzz detected all vulnerabilities, including V2, V7, and V11, with far fewer packets.

A detailed analysis of the inefficiencies in existing fuzzers reveals a common limitation: both AFLnet and BooFuzz face issues with BACnet packets containing magic fields, such as the length field, which indicates the total packet length. Mutation strategies that modify these fields can inadvertently create invalid packets, causing the target device to discard them. Furthermore, each tool has its own set of limitations: BASE identifies some magic fields, but these identifications are often inaccurate, leaving many fields untested. BooFuzz has a "fullrange" option that could theoretically identify all vulnerabilities by exploring all possible scenarios, but the excessive time required makes this approach impractical. With the "fullrange" option disabled by default, BooFuzz focuses primarily on boundary values (e.g., minimum, maximum, null), potentially overlooking vulnerabilities arising from non-boundary values. Additionally, BooFuzz generates redundant test cases, treating combinations such as $A=0,\ B=1,\ C=0$ and $B=1,\ A=0,\ C=0$ as distinct, even though they represent equivalent inputs. On the other hand, AFLnet, which relies solely on response codes, is ineffective for messages that do not generate responses.

F. Impact of Implicitly Reserved Fields in KNX (RQ5)

To demonstrate the generality of our approach, we extend the analysis to KNX. The KNX Association provides its official specifications (V3.0.0), along with the KNX Specifications Navigator—a ChatGPT-based assistant trained on the standard that enables direct querying of field information. We evaluated five KNX-certified router-type devices supporting both KNX IP and TP communication. Similar to BACnet, KNX allows packet injection at the IP layer via the management software. Using our method, we identified eight previously unknown vulnerabilities in these devices, seven of which have been acknowledged by the respective manufacturers.

These findings highlight a critical gap in the KNX certification process. Although KNX devices pass conformance testing using tools like EITT [24], which conduct tests based on clearly defined field semantics, our tool uncovered bugs caused by implicitly reserved fields. Thus, despite passing standard certification tests, devices remain vulnerable to malformed inputs targeting these implicitly reserved fields. For example, our disclosure to Siemens confirmed this gap and acknowledged that certain reserved fields lack semantic definitions beyond fixed binary values (e.g., zero) in the specification. This ambiguity hampers both implementation and testing, reinforcing our conclusion that implicitly reserved fields pose a systemic weakness in protocol design and validation. Our results demonstrate that the issues identified in BACnet also manifest in KNX, and that our methodology generalizes well to other BAS protocols, uncovering similarly latent flaws.

VI. Discussion

BACsFuzz presents certain limitations that also suggest promising directions for future research. First, although we considered firmware analysis, firmware images were unavailable. Major manufacturers like Siemens restrict firmware access to certified engineers, and most devices implement protections such as locked debug interfaces and flash readout prevention, rendering firmware extraction infeasible. Under these constraints, black-box network fuzzing remains the most practical approach. Second, our mutation strategy targets semantic ambiguities—particularly those in implicitly reserved fields, a category largely overlooked by existing fuzzers. In contrast, prior work in domains such as LTE [25], [26] focuses on injecting invalid values into well-defined fields to assess implementation robustness. Our approach goes one step further by evaluating the completeness of protocol design, uncovering vulnerabilities that stem from under-specified or ambiguous fields. While this strategy effectively reveals semantic flaws, it may miss low-level implementation bugs such as memory corruption. Future extensions could incorporate more advanced fuzzing techniques to broaden the scope of detection. We discuss complementary methods and recent advances in §VII.

VII. Related Work

BACnet Security. Studies reveal vulnerabilities in BACnet across multiple layers [27], [28], [29], [30], including eavesdropping, DoS attacks [30], weak authentication [31], [32], [33], and lack of encryption [34]. Intrusion detection is explored in [35], [36], [37], [38], [39], [40], while data manipulation is demonstrated by Peacock et al. [41]. Device discovery risks and network mapping attacks are discussed in [42], [43]. Recent work by Zhang et al. [5] introduced BASE, a protocol-aware fuzzer. Our work complements this direction by targeting two previously overlooked issues: the security implications of implicitly reserved fields and the throughput bottlenecks caused by the token-passing mechanism.

Network Fuzzing. Network protocol fuzzing has been widely studied. Some approaches [44], [45] model protocol states and message sequences to generate semantically valid inputs. Others [11], [12], [46] infer protocol semantics from companion apps or hub-device traffic. Response-guided fuzzers [47], [48] use observable feedback to guide mutation. With the emergence of LLMs, recent work [49], [50] automates protocol grammar extraction from textual specifications. Other studies [51], [52], [53] focus on prioritizing semantically critical message fields to improve precision. Despite this progress, prior efforts largely overlook implicitly reserved fields. We are the first to systematically investigate their impact and demonstrate practical exploits in real-world BAS deployments.

Medium Monopolization. Several works examine medium access monopolization attacks, where adversaries exploit protocol logic or low-level behavior to block legitimate communication. CANflict [54] uses peripheral conflicts to win CAN arbitration and suppress others. CANAttack [55] injects high-priority messages to dominate the bus. BROKENWIRE [56] disrupts EV charging by jamming CCS communications. Miller et al. [57] disable ECUs via CAN error-handling abuse. Although these works use medium monopolization primarily to cause DoS, we repurpose this behavior to optimize throughput. Our design removes token wait cycles and enables uninterrupted injection, significantly boosting fuzzing performance.

VIII. Conclusion

In this paper, we present BACsFuzz, a behavior-driven fuzzing tool designed to uncover vulnerabilities in BACnet, a widely adopted BAS protocol. By leveraging protocol-specific behaviors such as the token-passing mechanism, BACsFuzz overcomes BACnet's low-throughput limitation, significantly enhancing fuzzing efficiency. This approach uncovers vulnerabilities, particularly those related to implicitly reserved fields, which are common in both BACnet and KNX. Our experiments show throughput improvements ranging from 272.49% to 776.01% over SOTA methods. Testing 20 BAS devices, BACsFuzz identifies 26 vulnerabilities—24 confirmed by manufacturers, including nine with CVEs. Additionally, BACsFuzz successfully fuzzes KNX devices, confirming the widespread presence of implicitly reserved field vulnerabilities. This is the first to apply protocol-specific behaviors to address fuzzing limitations in constrained BACnet environments, demonstrating potential to enhance BAS security.

Ethics Considerations

To ensure safety, all experiments were conducted in a controlled lab environment (see Figure 8), not in operational building systems. Following responsible disclosure practices, we reported the token-seize vulnerability to ASHRAE, which confirmed and escalated it to the SSPC 135 Chair for formal discussion during the in-person meeting in Plantation, Florida (April 28-May 2, 2025). We also disclosed the implicitly reserved field issues to ASHRAE and the KNX Association; both are investigating. All vulnerabilities have been reported to the respective manufacturers. To date, 24 have been confirmed, with nine CVEs assigned. Siemens and Honeywell acknowledged our findings and invited us to their Recognition Programs. We strictly follow applicable regulations and OpenAI's enterprise privacy policy [58], which ensures user conversations are not used for training and remain confidential. Our research is academic and non-commercial, compliant with the fair use provisions defined by copyright law in the U.S. [59], the EU [60], and China [61].

Acknowledgments

We thank the anonymous reviewers for their constructive comments and suggestions. This research was supported in part by National Natural Science Foundation of China Grant Nos. 62232004 and 92467205, by US National Science Foundation (NSF) Award 2325451, Jiangsu Provincial Key Laboratory of Network and Information Security Grant No. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China Grant Nos. 93K-9, and Collaborative Innovation Center of Novel Software Technology and Industrialization. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] ReportLinker, "Building automation systems market - growth, trends, covid-19 impact, and forecasts (2022 - 2027)," https://www.reportlinker.com/p06360537/, October 2022.

[2] BACnet International. (2023) Bacnet protocol expands dominant market share in latest market research report. [Online]. Available: https://bacnetinternational.org/news/bacnet-protocol-expands-dominant-market-share-in-latest-market-research-report/

[3] ——. (2025) Assigned vendor ids. [Online]. Available: https://bacnet.org/assigned-vendor-ids/

[4] I. O. for Standardization, *Building automation and control systems (BACS) — Part 5: Data communication protocol*, ISO 16484-5:2022 ed. Vernier, Geneva, Switzerland: International Organization for Standardization, 2022. [Online]. Available: https://www.iso.org/standard/84964.html

[5] Y. Zhang, Z. Ling, M. Cash, Q. Zhang, C. Morales-Gonzalez, Q. Z. Sun, and X. Fu, "Collapse like a house of cards: Hacking building automation system through fuzzing," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, USA, October 14-18, 2024*. ACM, 2024.

[6] I. O. for Standardization, *Information technology — Home Electronic System (HES) architecture*, ISO/IEC 14543-3-10:2020 ed. Vernier, Geneva, Switzerland: International Organization for Standardization, 2021. [Online]. Available: https://www.iso.org/standard/80934.html

[7] A. Inc. (2024) Bacnet ms/tp communication protocol. [Online]. Available: https://www.accuenergy.com/support/reference-directory/bacnet-mstp/

[8] J. Molina, "Learn how to control every room at a luxury hotel remotely: The dangers of insecure home automation deployment," Black Hat USA, 2014.

[9] T. Brandstetter and K. Reisinger, "security in building automation how to create dark buildings with light speed," Black Hat USA, 2017.

[10] C. Vacherot, "Sneak into buildings with knxnet/ip," DEF CON, 2021.

[11] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[12] N. Redini, A. Continella, D. Das, G. D. Pasquale, N. Spahn, A. Machiry, A. Bianchi, C. Kruegel, and G. Vigna, "Diane: Identifying fuzzing triggers in apps to generate under-constrained inputs for iot devices," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021.

[13] M. Organization. (2006) Modbus over serial line specification and implementation guide v1.02. [Online]. Available: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[14] I. O. for Standardization, *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical coding sublayer*, ISO 11898-1:2024 ed. Vernier, Geneva, Switzerland: International Organization for Standardization, 2024. [Online]. Available: https://www.iso.org/standard/86384.html

[15] S. Karg. (2019) Bacnet stack an open source bacnet protocol stack for embedded systems. [Online]. Available: https://bacnet.sourceforge.net/

[16] J. Bender. (2015) Bacpypes. [Online]. Available: https://bacpypes.readthedocs.io/en/latest/

[17] RiptideIO. (2020) Misty: BACnet MS/TP Support for BACpypes. [Online]. Available: https://github.com/riptideio/misty

[18] KNX Association, "KNX Support and Developer Resources," https://support.knx.org/, 2025, accessed: 2025-04-19.

[19] Emergen Research. (2023) Top 10 leading companies in the building automation system market in 2023. [Online]. Available: https://www.emergenresearch.com/blog/top-10-leading-companies-in-the-building-automation-system-market-in-2023

[20] V. Pham, M. Böhme, and A. Roychoudhury, "AFLNET: A greybox fuzzer for network protocols," in *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*. IEEE, 2020.

[21] J. Pereyda, "Boofuzz: Network protocol fuzzing for humans," https://github.com/jtpereyda/boofuzz, 2016, accessed: 2025-04-23.

[22] Google, "Afl - american fuzzy lop," https://github.com/google/AFL, 2021, accessed: 2025-04-23.

[23] N. Group, "Fuzzowski: A protocol fuzzing framework for industrial control systems," https://github.com/nccgroup/fuzzowski, 2020, accessed: 2025-04-23.

[24] KNX Association. (2025) Eitt 4.4. [Online]. Available: https://support.knx.org/hc/en-us/articles/10732232002194-EITT-4-4

[25] Y. Chen, Y. Yao, X. Wang, D. Xu, C. Yue, X. Liu, K. Chen, H. Tang, and B. Liu, "Bookworm game: Automatic discovery of LTE vulnerabilities through documentation analysis," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021.

[26] C. Park, S. Bae, B. Oh, J. Lee, E. Lee, I. Yun, and Y. Kim, "Doltest: In-depth downlink negative testing framework for LTE devices," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 2022.

[27] W. Granzer and W. Kastner, "Security analysis of open building automation systems," in *Computer Safety, Reliability, and Security, 29th International Conference, SAFECOMP 2010, Vienna, Austria, September 14-17, 2010. Proceedings*. Springer, 2010.

[28] A. Ozadowicz, "Generic iot for smart buildings and field-level automation - challenges, threats, approaches, and solutions," *Comput.*, vol. 13, no. 2, p. 45, 2024.

[29] P. Ciholas, A. Lennie, P. Sadigova, and J. M. Such, "The security of smart buildings: a systematic literature review," *CoRR*, vol. abs/1901.05837, 2019.

[30] D. G. Holmberg and D. Evans, *BACnet wide area network security threat assessment*. US Department of Commerce, National Institute of Standards and Technology, 2003.

[31] V. Graveto, T. Cruz, and P. Simões, "Security of building automation and control systems: Survey and future research directions," *Comput. Secur.*, vol. 112, p. 102527, 2022.

[32] T. Sasaki, A. Fujita, C. H. Gañán, M. van Eeten, K. Yoshioka, and T. Matsumoto, "Exposed infrastructures: Discovery, attacks and remediation of insecure ICS remote management devices," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022.

[33] O. Gasser, Q. Scheitle, C. Denis, N. Schricker, and G. Carle, "Security implications of publicly reachable building automation systems," in *2017 IEEE Security and Privacy Workshops, SP Workshops 2017, San Jose, CA, USA, May 25, 2017*. IEEE, 2017.

[34] G. Stamatescu, I. Stamatescu, N. Arghira, and I. Fagarasan, "Cybersecurity perspectives for smart building automation systems," in *12th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2020, Bucharest, Romania, June 25-27, 2020*. IEEE, 2020.

[35] W.-H. Choi and J.-H. Lewe, "Advancing fault detection in building automation systems through deep learning," *Buildings*, vol. 14, no. 1, p. 271, 2024.

[36] M. Caselli, E. Zambon, J. Amann, R. Sommer, and F. Kargl, "Specification mining for intrusion detection in networked control systems," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. USENIX Association, 2016.

[37] D. Fauri, M. Kapsalakis, D. R. dos Santos, E. Costante, J. den Hartog, and S. Etalle, "Leveraging semantics for actionable intrusion detection in building automation systems," in *Critical Information Infrastructures Security - 13th International Conference, CRITIS 2018, Kaunas, Lithuania, September 24-26, 2018, Revised Selected Papers*. Springer, 2018.

[38] Z. Zheng and A. L. N. Reddy, "Safeguarding building automation networks: The-driven anomaly detector based on traffic analysis," in *26th International Conference on Computer Communication and Networks, ICCCN 2017, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*. IEEE, 2017.

[39] C. Valli, M. N. Johnstone, M. Peacock, and A. Jones, "Bacnet-bridging the cyber physical divide one hvac at a time," in *2017 9th IEEE-GCC Conference and Exhibition (GCCCE)*. IEEE, 2017.

[40] H. Esquivel-Vargas, M. Caselli, and A. Peter, "Automatic deployment of specification-based intrusion detection in the bacnet protocol," in *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy, Dallas, TX, USA, November 3, 2017*. ACM, 2017.

[41] M. Peacock, M. N. Johnstone, and C. Valli, "Security issues with bacnet value handling," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017*. SciTePress, 2017.

[42] M. Cash, S. Wang, B. Pearson, Q. Zhou, and X. Fu, "On automating bacnet device discovery and property identification," in *ICC 2021 - IEEE International Conference on Communications, Montreal, QC, Canada, June 14-23, 2021*. IEEE, 2021.

[43] H. Esquivel-Vargas, M. Caselli, and A. Peter, "Bacgraph: Automatic extraction of object relationships in the bacnet protocol," in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021, Taipei, Taiwan, June 21-24, 2021 - Supplemental Volume*. IEEE, 2021.

[44] M. E. Garbelini, V. Bedi, S. Chattopadhyay, S. Sun, and E. Kurniawan, "Braktooth: Causing havoc on bluetooth link manager via directed fuzzing," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 2022.

[45] X. Ma, Q. Zeng, H. Chi, and L. Luo, "No more companion apps hacking but one dongle: Hub-based blackbox fuzzing of iot firmware," in *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services, MobiSys 2023, Helsinki, Finland, June 18-22, 2023*. ACM, 2023.

[46] K. Liu, M. Yang, Z. Ling, Y. Zhang, C. Lei, J. Luo, and X. Fu, "Riotfuzzer: Companion app assisted remote fuzzing for detecting vulnerabilities in iot devices," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*. ACM, 2024.

[47] H. Liu, S. Gan, C. Zhang, Z. Gao, H. Zhang, X. Wang, and G. Gao, "Labrador: Response guided directed fuzzing for black-box iot devices," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

[48] X. Feng, R. Sun, X. Zhu, M. Xue, S. Wen, D. Liu, S. Nepal, and Y. Xiang, "Snipuzz: Black-box fuzzing of iot firmware via message snippet inference," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. ACM, 2021.

[49] J. Wang, L. Yu, and X. Luo, "LLMIF: augmented large language model for fuzzing iot devices," in *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024.

[50] X. Ma, L. Luo, and Q. Zeng, "From one thousand pages of specification to unveiling hidden bugs: Large language model assisted fuzzing of matter iot devices," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[51] M. Ren, H. Zhang, X. Ren, J. Ming, and Y. Lei, "Intelligent zigbee protocol fuzzing via constraint-field dependency inference," in *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part II*. Springer, 2023.

[52] H. Wanyan, Y. Lai, J. Liu, and H. Chen, "Ncmfuzzer: Using non-critical field mutation and test case combination to improve the efficiency of ICS protocol fuzzing," *Comput. Secur.*, vol. 141, p. 103811, 2024.

[53] S. Kim and T. Shon, "Field classification-based novel fuzzing case generation for ICS protocols," *J. Supercomput.*, vol. 74, no. 9, pp. 4434–4450, 2018.

[54] A. de Faveri Tron, S. Longari, M. Carminati, M. Polino, and S. Zanero, "Canflict: Exploiting peripheral conflicts for data-link layer attacks on automotive networks," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. ACM, 2022.

[55] D. Oladimeji, A. Rasheed, C. Varol, M. Baza, H. Alshahrani, and A. Baz, "Canattack: Assessing vulnerabilities within controller area network," *Sensors*, vol. 23, no. 19, p. 8223, 2023.

[56] S. Köhler, R. Baker, M. Strohmeier, and I. Martinovic, "Brokenwire : Wireless disruption of CCS electric vehicle charging," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.

[57] M. Rogers and K. Rasmussen, "Silently disabling ecus and enabling blind attacks on the CAN bus," *CoRR*, vol. abs/2201.06362, 2022.

[58] OpenAI, "Enterprise privacy at openai," 2024, accessed: 2024-10-22. [Online]. Available: https://openai.com/enterprise-privacy/

[59] U.S. Copyright Office, "Limitations on exclusive rights: Fair use (section 107)," 2024, accessed: 2024-10-22. [Online]. Available: https://www.copyright.gov/title17/92chap1.html#107

[60] European Parliament and Council, "Directive 2001/29/ec of the european parliament and of the council of 22 may 2001 on the harmonisation of certain aspects of copyright and related rights in the information society," 2001, accessed: 2024-10-22. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32001L0029

[61] The State Council of the People's Republic of China, "Copyright law of the people's republic of china (2010 amendment)," 2010, accessed: 2024-10-22. [Online]. Available: https://english.www.gov.cn/archive/laws_regulations/2014/08/23/content_281474982987430.htm

[62] S. Sun, Y. Liu, D. Iter, C. Zhu, and M. Iyyer, "How does in-context learning help prompt tuning?" in *Findings of the Association for Computational Linguistics: EACL 2024, St. Julian's, Malta, March 17-22, 2024*. Association for Computational Linguistics, 2024.

[63] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[64] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[65] P. Sancheti, K. Karlapalem, and K. Vemuri, "LLM driven web profile extraction for identical names," in *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, Singapore, May 13-17, 2024*. ACM, 2024.

[66] S. Wang, X. Sun, X. Li, R. Ouyang, F. Wu, T. Zhang, J. Li, and G. Wang, "GPT-NER: named entity recognition via large language models," *CoRR*, 2023.

[67] Python Software Foundation. (2024) Pypdf2. [Online]. Available: https://pypi.org/project/PyPDF2/
[68] J. Singer-Vine, "pdfplumber," https://github.com/jsvine/pdfplumber, 2020, accessed: 2025-04-23.

APPENDIX

APPENDIX A
IDENTIFY IMPLICITLY RESERVED FIELDS USING LLM

A. Methodology

Identifying fields requires first understanding and recognizing the structure they belong to. The definition, value, and position of fields are inherently tied to their respective structures. BACnet message structures are highly complex, complicating the task of identifying implicitly reserved fields.

To address this, we leverage LLMs for their advanced natural language processing capabilities. The process begins by pre-processing the specification to extract relevant information for LLM analysis (**Step I**). To improve performance, we apply in-context few-shot learning [62], [63], eliminating the need for fine-tuning or additional training. This involves embedding detailed task instructions and representative examples into each prompt, allowing the model to learn from context. After Step I, prompt engineering (as shown in Figure 9) guides the LLM's reasoning process using Chain-of-Thought (CoT) [64] prompting for multi-step analysis. In (**Step II**), we match field names to the specification and extract relevant descriptions. CoT divides the task into two parallel reasoning chains: (**Step III**) identifies message structures and dependencies within BACnet fields, while (**Step IV**) determines field lengths and value ranges to identify implicitly reserved fields.

Each prompt acts as a Named Entity Recognition (NER) [65], [66] system tailored to the BACnet domain, with the output format defined as a list of dictionaries containing *'T' (type of entity)* and *'E' (entity)*. For example, in Step II, *'T'* may be `FIELD_NAME` (indicating we are identifying field names in specification), and *'E'* identified from the specification is `DNET`, a specific field name. Steps III and IV are executed in parallel, interconnected through a shared *'T'*, i.e., `FIELD_NAME`, allowing BACSFUZZ to effectively parse message structures and identify implicitly reserved fields.

Step (I): Document Pre-processing. Directly inputting the entire *ISO 16484-5* PDF (23.9 MB) into an LLM is inefficient due to input size limits and irrelevant content. We use *PyPDF2* [67] to extract first-level headings and filter chapters based on keywords like "Application", "Network", "MS/TP" and "BACnet/IP". These chapters are merged into four thematic PDFs. Given that LLMs are not well-suited for reliably recognizing PDF content, we convert the PDFs into four text files using *pdfplumber* [68], reducing the total size to 605 KB with a compression rate of 2.47%. After verification, all token counts fall within the LLM's input limit.

Step (II): Field Matching. Directly extracting field names from the specification using an LLM can result in false positives or false negatives. To mitigate this, we leverage field names from open-source projects like BACpypes [16], which adhere to the BACnet specification. These projects annotate field names to indicate their inclusion in message structures. Since direct matching between project field names and the specification is inefficient due to naming differences, we use the LLM to establish semantic correlations, as shown in *Prompt 1* in Figure 9. For instance, `npduDNET` in BACpypes corresponds to `DNET` in the specification. We compile four field lists for each text file.

Step (III): Structure Resolution. BACnet defines a layered message architecture §II-B, and we analyze each layer individually before integrating the results. For layers with fewer fields, the specification provides explicit message structures, allowing the LLM to identify them accurately (see *Prompt 2* in Figure 9). The APDU, MS/TP, and BACnet/IP structures follow this pattern. However, for layers with more fields, the method's efficiency decreases. BACnet fields can be classified as mandatory or optional. Mandatory fields are essential for protocol execution, while optional fields are included under specific conditions. For example, the `Destination Specifier` in NPDU is mandatory for basic communication, whereas fields like `DNET`, `DLEN`, and `Hop Count` are included only when `Destination Specifier` is set to `0x01`. By matching `FIELD_NAME` and `FIELD_Description` from *Prompt 1*, the LLM can accurately identify both fields and their dependencies, as shown in *Prompt 3* in Figure 9. Once fields and their dependencies are identified, BACSFUZZ uses this structured data to automate the recognition of message structures and resolve dependencies between fields. For each mandatory field, $(Mandatory_{x_1}, \ldots, Mandatory_{x_n})$, LLM-inferred dependencies help identify the optional fields, $(Optional_{y_1}, \ldots, Optional_{y_m})$, required for a valid message structure. For instance, in NPDU, mandatory fields include `Version`, `NSDU`, `Reserved1`, `Destination Specifier`, `Reserved2`, `Source Specifier`, `Expecting Reply`, and `Priority`. We first evaluate the `Version` field and its impact on optional fields, followed by sequential analysis of other mandatory fields. In the first iteration, we identify optional fields influenced by each mandatory field. For example, if `NSDU = 0x01`, the `Message Type` field must be included. Similarly, if `Destination Specifier = 0x01`, fields like `DNET`, `DLEN`, and `Hop Count` must be included. In the second iteration, we examine pairs of mandatory fields and their combined influence on optional fields, continuing this process until all combinations are analyzed.

Step (IV): Implicitly Reserved Fields Identification. The LLM analyzes `FIELD_NAME` and `FIELD_Description` from *Prompt 1* to identify implicitly reserved fields. As outlined in *Prompt 4* and *Prompt 5* (Figure 9), the analysis focuses on field length and protocol-defined value ranges. Based on these, fields are classified as fully defined, explicitly reserved, or implicitly reserved. *Prompt 4* provides the field length and value ranges, while *Prompt 5* checks if the specified ranges match the field length. If they match exactly, the field is classified as fully defined. If there is a mismatch, *Prompt 5* checks for explicitly reserved values. If no such values are found, the field is classified as implicitly reserved.

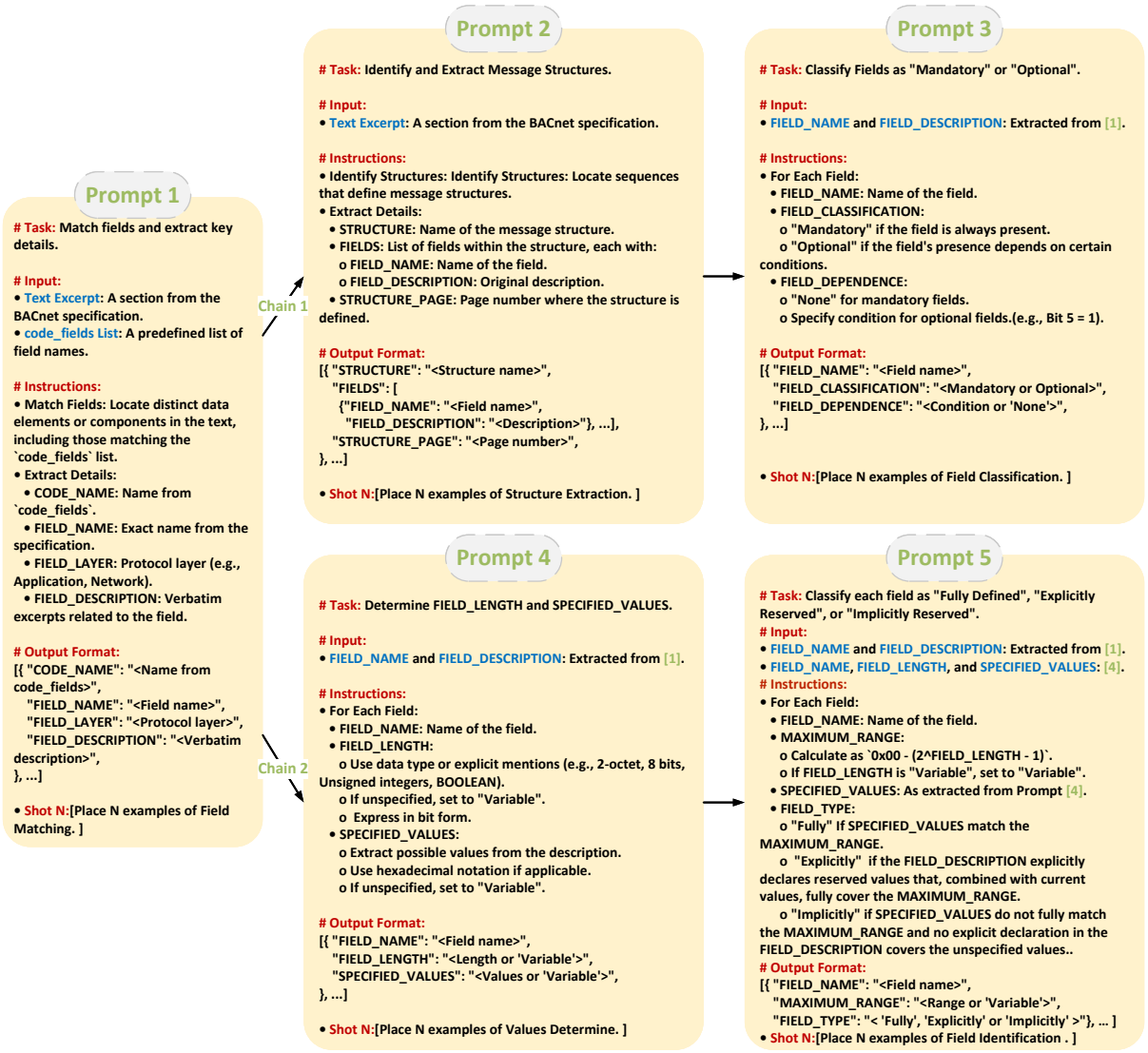

Fig. 9: Prompt Templates.

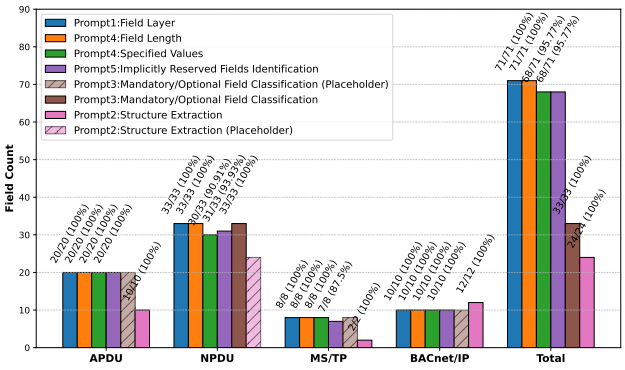

Fig. 10: LLM Analysis Results.

B. Evaluation

To assess the accuracy of the LLM, we manually validate all results for correctness and completeness, as summarized in Figure 10. *Prompt 1* enables the LLM to correctly match 71 fields extracted from open-source projects to their corresponding specification entries and identify the relevant communication layers. *Prompt 2* achieves 100% accuracy (24/24) in identifying APDU, MS/TP, and BACnet/IP message structures, all of which exhibit relatively simple formats. Despite the increased complexity of NPDU structures, *Prompt 3* successfully extracts both mandatory and optional fields, yielding 81 distinct NPDU types. In total, the analysis yields 550 message structures, comprising 136 MS/TP messages and 414 BACnet/IP messages. All 136 MS/TP messages are manually verified as correct, demonstrating the LLM's effectiveness in identifying BACnet message structures. *Prompt 4* and *Prompt 5* extract field lengths with 100% accuracy, while identifying field values and implicitly reserved fields with a verified accuracy of 95.77%. Since this classification is based on a manually validated and complete field set, it also substantiates

the overall completeness of our field identification process.

We manually verify misidentified instances. For example, the DLEN and SLEN fields of the NPDU are mistakenly classified with a value range of 0-0xFF. In reality, the DLEN values are 0x01, 0x02, 0x03, 0x06, 0x07, and the SLEN values are 0x01, 0x02, 0x03, 0x06, which are listed only in *Table 6-2* [4] and not in the specification text. *pdfplumber* does not retain table information when converting a PDF file to text, causing the table to be omitted. The LLM infers a value range of 0-0xFF due to the 1-octet length of these fields, leading to the incorrect conclusion that they are not implicitly reserved. This error is caused by the missing table layout during conversion, not by the LLM's inference. A multimodal model could address this by retrieving table structures, avoiding such errors. In another case, the length of the List of DNETs field in the NPDU is correctly identified as "variable", but its value range is mistakenly marked as 0x0001–0xFFFF. Although a single DNET is 2 bytes with a range of 0x0001–0xFFFF, the LLM incorrectly applied this range to the entire List of DNETs due to the unknown number of DNETs. However, since the field is labeled as "variable", the LLM correctly categorizes it as implicitly reserved, so the hallucination does not affect the final result. In the MS/TP layer, the Header CRC field is incorrectly classified as implicitly reserved. Although 8-bit, it serves as a header verification mechanism, as its name suggests. This highlights the need to analyze field names semantically, since function matters more than size, and it should not be classified as implicitly reserved.

APPENDIX B
TARGET DEVICE STATE TRANSITION DURING FUZZING PROCESS

As shown in Figure 7, when a device holding a token is targeted by BACSFUZZ, it may be in one of five states:

1) ***USE_TOKEN***: the device determines how best to utilize the token. It transitions to DONE_WITH_TOKEN if no further data needs to be sent, or if the sent data frame does not require a reply. Conversely, if a reply is necessary, the device moves to WAIT_FOR_REPLY to await a response.
2) ***WAIT_FOR_REPLY***: the device awaits responses to previously issued requests. Detection of abnormal network activity, such as an attack by BACSFUZZ, leads the device to infer multiple tokens circulating in the network, prompting it to drop its token and transition to IDLE.
3) ***DONE_WITH_TOKEN***: after completing data transmission, the device's next action depends on the current network condition. It either performs POLL_FOR_MASTER polling if the next token recipient is unknown or passes the token via PASS_TOKEN if the recipient is known.
4) ***PASS_TOKEN***: the device attempts to pass the token to the next node. However, due to BACSFUZZ continuously resetting the device's *SilenceTimer* to 0, preventing it from exceeding the threshold, the device transitions to the IDLE state, effectively dropping the token.
5) ***POLL_FOR_MASTER***: the device seeks to determine or confirm the network's master node through polling. If abnormal token activity is detected during a BACSFUZZ attack, the device drops its token and transitions to IDLE.

For devices without a token, potential states include IDLE and ANSWER_DATA_REQUEST. In the IDLE state, the device remains inactive unless a data frame is received that requires a reply. If this occurs, the device transitions to ANSWER_DATA_REQUEST, where it quickly responds to BACSFUZZ's requests before returning to IDLE state.