IoTBec: An Accurate and Efficient Recurring Vulnerability Detection Framework for Black Box IoT devices

Haoran Yang*†#, Jiaming Guo*†#, Shuangning Yang‡, Guoli Zhao*†, Qingqi Liu*†, Chi Zhang*†, Zhenlu Tan*†, Lixiao Shan*†, Qihang Zhou*†, Mengting Zhou*, Jianwei Tai‡, Xiaoqi Jia*†✉
*Institute of Information Engineering, Chinese Academy of Sciences, CHINA
Email: {yanghaoran, guojiaming, zhaoguoli, liuqingqi, zhangchi2024, tanzhenlu, shanlixiao, zhouqihang, zhoumengting, jiaxiaoqi}@iie.ac.cn
†School of Cyber Security, University of Chinese Academy of Sciences, CHINA
‡School of Internet, Anhui University, CHINA
Email: realyangshuangning@gmail.com, 24012@ahu.edu.cn

***Abstract*—The proliferation of IoT devices has driven a rise in vulnerability exploits. Existing vulnerability detection approaches heavily rely on firmware or source code for analysis. This reliance critically compromises their efficiency in real-world black-box scenarios. To address this limitation, we propose IoTBec, a novel firmware and source-code independent framework for recurring vulnerability detection. IoTBec innovatively constructs a Vulnerability Interface Signature (VIS) based on black-box interfaces and known vulnerability information. The signature is designed to match potential recurring vulnerabilities against target devices. The framework then deeply integrates this signature-based detection with Large Language Model (LLM)-driven fuzzing. Upon a match, IoTBec automatically leverages LLMs to generate targeted fuzzing payloads for verification.**

To evaluate IoTBec, we conducted extensive experiments on devices from five major IoT vendors. Results show that IoTBec discovers over 7 times more vulnerabilities than the current state-of-the-art (SOTA) black-box fuzzing methods, with 100% precision and 93.37% recall. Overall, IoTBec detected 183 vulnerabilities, 169 of which were assigned CVE IDs. Among these, 53 were newly discovered and had an average CVSS 3.x score of 8.61, covering buffer overflows, command injection, and CSRF issues. Notably, through LLM-driven fuzzing, IoTBec also discovered 25 previously unknown vulnerabilities. The experimental evidence suggests that IoTBec's unique firmware and source-code independent paradigm enhances detection efficiency and enables the discovery of novel and variant vulnerabilities. We will release the source code for IoTBec and the experiment data at https://github.com/IoTBec.

I. Introduction

In recent years, the number of IoT devices, such as routers, switches, and cameras, has increased rapidly, with widespread applications in both personal and industrial scenarios. It is projected that the global active IoT device count will surge to approximately 40 billion by the end of 2030 [1]. However, the security situation for IoT has deteriorated significantly, with frequent attack incidents [2], [3]. Research indicates that over half of IoT devices contain security vulnerabilities [4]. Therefore, the rapid, efficient, and reliable detection of vulnerabilities in IoT devices has become a pressing issue in cybersecurity.

Current vulnerability detection methods for IoT devices primarily fall into two categories: static analysis [5], [6] and dynamic analysis [7]–[11], collectively referred to as model-based vulnerability detection. Static analysis involves examining IoT device firmware or source code to find potential security vulnerabilities without executing the program, such as taint analysis and symbolic execution. In contrast, dynamic analysis detects program behavior during actual execution, triggering anomalies through input mutation and other techniques to uncover exploitable security flaws. However, these methods are facing fundamental limitations in practical application:

L1: Firmware Acquisition and Decryption. Acquiring firmware for a large number of IoT devices is extremely difficult, as it is frequently not publicly accessible. Furthermore, a considerable number of firmwares are protected by encryption mechanisms, which critically impede subsequent analysis and testing [7].

L2: High-Fidelity Emulation. Even with successful firmware acquisition, achieving high-fidelity emulation of firmware is highly challenging. Especially for enterprise-grade devices, existing firmware emulation tools have limited capabilities for replicating peripheral interactions and system environments, often failing to achieve high fidelity, which leads to relatively low emulation success rates [12], [13].

Due to these limitations, white-box testing becomes infeasible in most practical IoT scenarios [14]–[17], while grey-box testing is constrained by unrealistic assumptions and stringent setup requirements. Consequently, black-box testing has become the most practical approach to IoT security detection. Traditional black-box testing methods, such as equivalence partitioning [18], [19] and boundary value analysis, primarily

#: Equal contribution ✉: Corresponding author

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.2026.240634
www.ndss-symposium.org

target functional defects and performance issues, making them ineffective in discovering security vulnerabilities. Existing static analysis techniques are largely impractical in black-box scenarios due to the lack of firmware and source code. Therefore, current black-box vulnerability detection for IoT devices typically relies on dynamic methods, primarily exemplified by fuzzing [7], [8]. However, in black-box scenarios, the absence of firmware and prior knowledge limits the effectiveness of seed generation and mutation strategies, leading to generally low efficiency in vulnerability discovery through fuzzing. Currently, LABRADOR [9] is the leading black-box IoT fuzzing solution, which introduces response-guided control flow inference to enhance fuzzing efficiency. Nonetheless, it still relies on firmware access for static analysis, making it unsuitable for a truly black-box scenario.

Recurring vulnerability detection, as an essential complementary technique to model-based vulnerability detection, has been proven to enhance vulnerability discovery capabilities effectively by leveraging known vulnerability information to find new vulnerabilities with similar flaws [20]. In IoT firmware, recurring vulnerabilities are also frequently detected. However, existing recurring vulnerability detection methods typically rely on firmware or source code to generate vulnerability signatures, making them infeasible to apply directly in black-box testing scenarios. To achieve effective recurring vulnerability detection where such resources are unavailable, three core challenges need to be addressed:

C1: How to design a black-box scenario applicable vulnerability signature.

C2: How to efficiently generate these vulnerability signatures and build a ground truth database.

C3: How to leverage this constructed signature ground truth to conduct automated, precise recurring vulnerability detection on black-box IoT devices.

To address these challenges, we propose IoTBec, an automated recurring vulnerability detection framework for black-box IoT devices. IoTBec integrates Interface Signatures and Vulnerability Signatures, leveraging the device interface structural features, network data packets, and open-source vulnerability information to automatically discover potential recurring vulnerabilities without relying on source code or firmware.

The following is IoTBec's vulnerability detection procedure. First, we propose a method for constructing the Interface Signature (IS). We use a depth-first traversal algorithm to extract the POST interfaces, network request data packets, and associated UI context information for each interface, thereby constructing a unique IS to achieve a structured representation of device functional interfaces. Subsequently, we leverage a vulnerability analyzer LLM with meticulously designed prompts to assist in automatically generating the Vulnerability Signature (VS). Specifically, IoTBec crawls official CVE [21] descriptions and open-source vulnerability reports. Through the LLM's summarization capabilities, it efficiently extracts key vulnerability characteristics, including CVE ID, vulnerability type, affected POST interface, and critical parameter, to construct VS. Finally, within the same device, IoTBec innovatively proposes the Vulnerability Interface Signature (VIS) by automatically extracting common information through matching key fields, such as the POST interface between IS and VS. This process effectively integrates device interface information with vulnerability characteristics, providing a conceptual approach and a concrete solution for C1 and C2.

To verify IoTBec's effectiveness and efficiency, we designed a complete experimental process. First, we constructed a ground truth database of VIS for recurring vulnerability detection by selecting six products from each of the five mainstream IoT vendors (Tenda, TOTOLINK, D-Link, TP-Link, Linksys). We then selected 21 devices from the same product series as a test set, for which only IS was constructed. For each IS in the test set, we performed modular similarity matching against signatures in the ground truth database. For each successfully matched IS-VIS pair, we used the request data packet of the test device's IS to generate fuzzing seed inputs. Subsequently, combining the matched VIS and corresponding open-source reports, the fuzzing LLM generated targeted fuzzing payloads. All payloads were executed automatically with the monitoring strategies. The entire process—from crawling vulnerability information, LLM-driven signature and payload generation, to testing, monitoring, and alerting—achieved a high degree of automation for solving C3.

Results of these experiments show that compared to existing SOTA black-box IoT fuzzing methods, IoTBec discovered an average of 7 times more vulnerabilities and obtained 53 CVE IDs. IoTBec achieved 100% alert precision and a recall rate of 93.37%. Notably, during fuzzing, relying on prompts and the LLM's mutation capabilities, IoTBec discovered 25 vulnerabilities beyond the ground truth. Among these, 4 vulnerabilities were confirmed not to have been reported in prior CVEs for the corresponding products. IoTBec not only significantly improved the efficiency of black-box IoT device vulnerability discovery methods but also, by combining recurring vulnerability detection with fuzzing, broke through the limitation of traditional recurring vulnerability detection to only "find duplicated vulnerabilities". The main contributions of this paper for black-box IoT testing scenarios are as follows:

1) **First to propose a recurring vulnerability detection framework for black-box IoT devices:** We designed the IoTBec framework based on VIS, combining known vulnerability characteristics with device interfaces to build a high-quality VIS ground truth database. Using the matching algorithm and fuzzing strategy, we achieved accurate and efficient detection of recurring vulnerabilities in black-box scenarios.
2) **Comprehensive Evaluation on real-world devices:** We conducted systematic experiments on 27 real-world devices from five mainstream vendors. Experimental results demonstrate that IoTBec discovered over 7 times vulnerabilities compared to boofuzz and Snipuzz, highlighting its superior effectiveness.
3) **Discovery of a large number of high-severity vulnerabilities:** IoTBec discovered 183 vulnerabilities, 53 of which were newly assigned as CVE (average CVSS

3.x score [22] of 8.61), covering taint-based buffer overflow and command injection vulnerabilities, and web-frontend vulnerabilities represented by CSRF, breaking through the limitations of traditional recurring vulnerability detection methods.

II. Motivation and Threat Model

A. IoTBec's Core Motivation

In practical IoT device testing scenarios, firmware is often not publicly available or cannot be extracted [23], [24]. In black-box testing scenarios, model-based detection methods are often ineffective. Although Snipuzz [7] effectively improved the efficiency of black-box IoT testing by using device responses to mutate payloads, the responses from most IoT devices are relatively simple and lack sufficient information to guide fuzzing mutations effectively. This severe information deficit prompted us to investigate how to leverage high-quality, accessible information within a black-box scenario to assist fuzzing. Through in-depth research and observation, we identified two types of information with significant potential:

1) **High-Quality Open-Source CVE Vulnerability Reports:** The number of CVEs for IoT devices has been immense and has increased rapidly over the past five years. Mainstream CNA organizations, such as MITRE Corporation [25], typically require submitters to provide key information during the CVE review process, including affected product versions, vulnerability types, descriptions, and exploitation details.
2) **IoT Device UI Context and Network Request Packet Characteristic:** IoT devices are commonly accompanied by UI management interfaces. Their UI structure and functional modules are highly readable, and similar functional modules across different devices exhibit strong consistency. Concurrently, capturing network request packets during UI interaction can yield network packets containing POST interface and parameter information for corresponding functional modules.

Based on the accessibility of the aforementioned external information, we also observed the phenomenon of recurring vulnerabilities in IoT devices. Due to the highly repetitive functional modules and interaction logic commonly found in IoT firmware, recurring vulnerabilities frequently appear across different physical devices. Taking the buffer overflow vulnerability CVE-2022-38314 [26] in the Tenda AC18 router as an example, this vulnerability occurs at the POST interface */goform/saveParentControlInfo*, which is used for fine-grained internet access management based on specified times, dates, URLs, etc. Through a deep analysis of related CVE reports and actual devices, we discovered that this vulnerability exists in at least 18 Tenda devices with similar functionality. The web navigation-bar UI on these devices often includes the keyword "Parent Control." Furthermore, the data packets for requests to */goform/saveParentControlInfo* mainly include the critical parameters required to trigger the vulnerability: *deviceId*, *urls*, and *time*, as explicitly mentioned in the corresponding CVE reports. The accessibility of these features provides the necessary conditions for precise, multidimensional recurring vulnerability detection in a black-box scenario. Similarly, a buffer overflow vulnerability in *loginauth*, a standard function used to handle login logic in TOTOLINK devices, affects at least 16 products.

Numerous cases illustrate that recurring vulnerability detection offers a unique and efficient avenue for discovery. While the current SOTA method for recurring vulnerability detection in IoT devices, FirmRec [20], is effective, it heavily relies on firmware, which cannot be applied to the black-box scenarios described above. Based on a deep understanding of the prevalence of recurring vulnerabilities in IoT devices and the consistency of their externally visible features, such as UI elements, POST interfaces, and network packet parameters, we proposed the Vulnerability Interface Signature (VIS) method and the IoTBec framework. Our goal is to leverage this accessible external information in a black-box scenario to construct a novel type of vulnerability signature. This signature is then combined with fuzzing under a recurring vulnerability detection framework, thereby enabling efficient vulnerability discovery without relying on firmware or source code.

B. Availability of IoTBec's Core Information Sources

The effectiveness of IoTBec depends on the quality and accessibility of the black-box visible information sources. Accordingly, we conducted a detailed availability analysis of the two core information sources that IoTBec builds upon: IoT device UI context information and open-source CVE vulnerability information, to validate their widespread accessibility in real-world IoT devices.

1) Open-Source CVE Vulnerability Information: In recent years, the number of vulnerabilities in IoT devices has continued to increase. Research indicates that over 50% of devices possess critical security vulnerabilities that can be directly exploited [27], and the number of publicly disclosed vulnerabilities has increased by 136% between 2022 and 2024 [28]. For IoT device vulnerabilities, CVE descriptions and vulnerability reports in references on the CVE official website typically include crucial information, such as affected product versions, vulnerability types, descriptions, and exploitation details. To verify the widespread availability of this critical information for constructing Vulnerability Signatures, we conducted a survey of 500 CVE IDs from four mainstream manufacturers—Tenda, TOTOLINK, D-Link, and TP-Link—between May 26th and 29th, 2025. The detailed survey results are presented in Appendix subsection A, confirming that the vast majority of CVE descriptions and their associated reports contain the necessary details for building effective Vulnerability Signatures. Notably, CVEs issued by CNA VULDB [29] surged between 2024 and 2025, accounting for over 40% of the total vulnerabilities from the four manufacturers mentioned above during that interval. VULDB is a promising CNA organization. Its procedures for vulnerability submission are notably more comprehensive and transparent. Furthermore, VULDB offers faster email responses and consistently

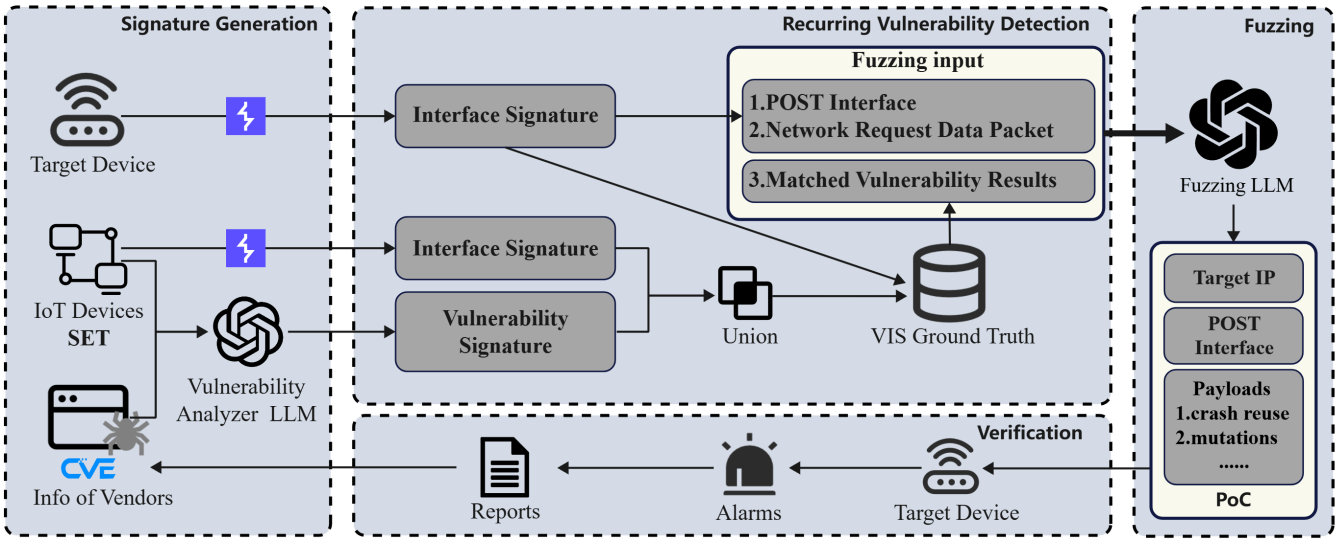

Fig. 1. IoTBec Workflow

provides more standardized vulnerability descriptions while ensuring the ready accessibility of reference links. Our survey revealed that an exceptionally high proportion (approximately 93.2%) of CVEs assigned by VULDB between 2024 and 2025 contained all the necessary information for the VS construction of IoTBec. From the perspective of industry status and vulnerability growth trends, the signature generation mechanism based on CVEs and public vulnerability reports is broadly applicable and long-term effective. It can cover current mainstream devices and is also suitable for the continuous evolution of future IoT vulnerability detection needs. Open-source vulnerability information can be used to generate high-quality Vulnerability Signatures, which provide necessary support for subsequent payload construction during the vulnerability fuzzing stage.

2) IoT Device UI Context Information: Reports indicate that IoT device management is typically carried out through web UI or mobile apps, which are used for registration, control, and configuration [30]. Among currently vulnerable IoT devices, routers account for over 50% [31], and the vast majority of them are equipped with rich UI interfaces. We researched 85 devices (including routers, extenders, network bridges, and cameras) from 8 well-known IoT vendors. We found that the device UI is not only universally present but also plays a central role in device configuration and management. Specifically, all of these IoT devices employ a uniform UI structure consisting of a navigation-bar and hierarchical pages. This highly consistent design demonstrates strong commonality and transferability, even across different manufacturers, device types, and versions. To be precise, the hierarchical path of navigation bar fields can accurately delineate the device's functional hierarchy and serve as a unique identifier for functional interfaces, providing reliable support for Interface Signature matching. Our study indicates that IoTBec's use of the interface structure is not a restrictive prerequisite but rather a natural choice, given the universal presence and high degree of consistency of current IoT device interfaces, which ensures its broad applicability across the vast majority of IoT devices.

C. IoTBec's Threat Model

This section outlines the capabilities and the research scope of the IoTBec framework. Our threat model specifically targets black-box IoT device vulnerability detection scenarios, with the fundamental assumption that the security analyst cannot obtain the target device's firmware or source code, nor can they perform any form of physical access or internal debugging. In this stringent black-box scenario, the analyst primarily relies on network reachability, enabling them to log in and access the target IoT device's web management interfaces and open network services via the internet or a local network. The security analyst is assumed to possess robust external information-gathering capabilities, including accessing public CVE databases and their associated vulnerability reports, capturing web traffic and data packets using proxy tools, and traversing the device's web interface structure. Furthermore, we assume the analyst can leverage the LLMs to process unstructured vulnerability information and infer a large number of potential attack vectors. Under these assumed capabilities, the primary target of the analysis is to discover recurring security vulnerabilities related to publicly disclosed CVEs on new, insufficiently tested IoT devices; to identify novel or variant vulnerabilities of these known issues; and ultimately, to exploit these vulnerabilities to perform denial-of-service or command execution on the devices. IoTBec is specifically designed to assist security analysts in efficiently and automatically identifying remote vulnerabilities triggered via web interfaces, such as buffer overflows and command injections, within this defined threat model.

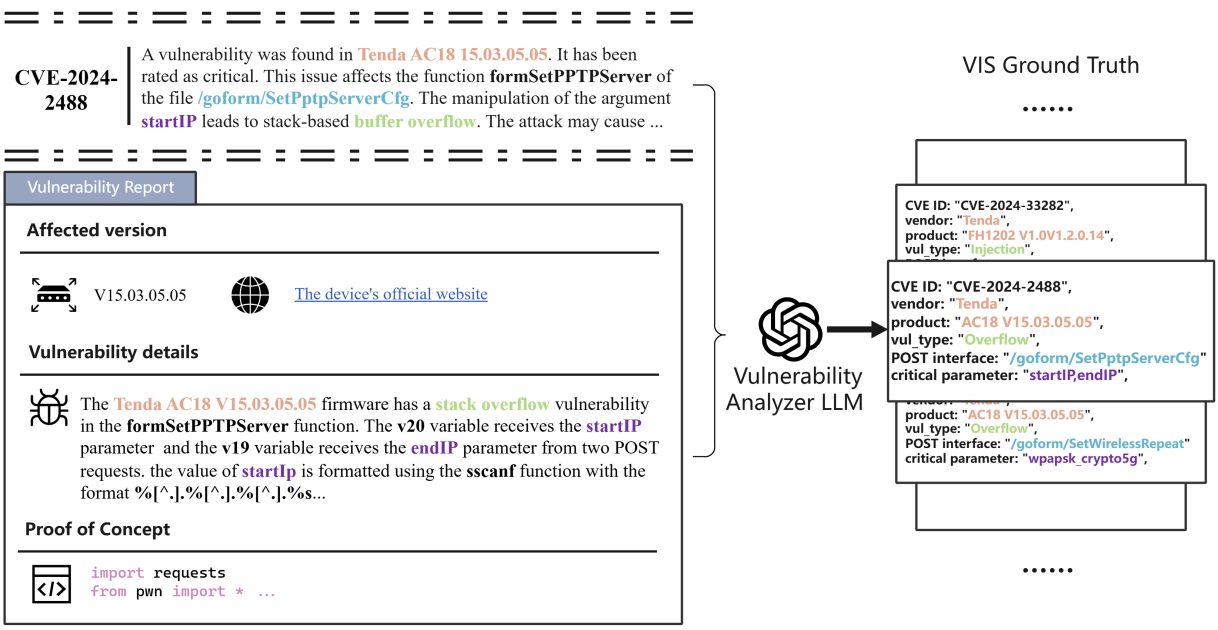

Fig. 2. Illustration of Vulnerability Signature Extraction with the Vulnerability Analyzer LLM

III. Methodology

This section will elaborate on the technical design and algorithmic implementation of the IoTBec framework. Figure 1 illustrates the overall architecture of IoTBec, whose workflow primarily consists of the following three stages:

1) **Information Collection and Signature Generation:** IoTBec acquires and generates Vulnerability Signatures (VS) and Interface Signatures (IS) from the POST interface, network packets, and public CVE reports of a subset of IoT devices.
2) **Vulnerability Interface Signature (VIS) Construction and Recurring Vulnerability Matching:** IoTBec integrates the IS and VS into the VIS and creates a ground truth database. For a device under test, it extracts its IS and matches it against a ground-truth database using an algorithm to identify potential recurring vulnerabilities.
3) **LLM-Driven Fuzzing and Vulnerability Validation:** For each matched recurring vulnerability, IoTBec utilizes the fuzzing LLM to generate and execute fuzzing payloads, and it validates the vulnerability trigger through a monitoring module.

IoTBec achieves end-to-end automation, from information acquisition to vulnerability validation, without any reliance on the target device's firmware or source code. The following subsections detail the signature design, the signature matching algorithm, and the LLM-driven fuzzing strategy.

A. Signature Design

This subsection will detail the structure of the three core signatures within IoTBec.

1) Vulnerability Signature(VS): For both the VS discussed in this section and the IS detailed later, IoTBec strictly adheres to the principle of using only information accessible within the threat model. Given that firmware and source code are inaccessible, leveraging open-source CVE information to create Vulnerability Signatures is the optimal choice. Typically, a Proof of Concept (PoC) for an IoT device vulnerability contains two key components: the POST interface and the payload. Combined with the affected product versions, this set of information can precisely characterize a specific vulnerability. As illustrated in Figure 2, for each open-source CVE, the VS we designed includes the following key fields: CVE ID, vendor, affected product, vulnerability type, POST interface, and critical parameters. IoTBec opts for "critical parameters" instead of complete payload information because many public vulnerability reports do not directly disclose the full exploitation payload. In contrast, critical parameters are necessary conditions for triggering the vulnerability, and their availability is more stable and widespread. These critical parameters, in combination with the network packets from the IS, will jointly guide the fuzzing LLM to construct the final exploitation payload intelligently.

IoTBec automatically generates these JSON-formatted Vulnerability Signatures using a vulnerability analyzer LLM. We first obtain the official CVE description from https://cve.mitre.org and retrieve associated public vulnerability reports (e.g., vendor security advisories or third-party research reports) from its references section. Subsequently, we use a specially designed prompt to guide the LLM in performing information extraction and structuring. In this prompt, the inputs include the CVE description and the full text of the vulnerability report, explicitly instructing the LLM to assume the role of a cybersecurity analyst. Its task is to extract or infer the aforementioned VS fields and to output the result exclusively in a strictly defined JSON object format. The prompt provides

specific instructions, for instance, directing the LLM to find the POST interface within the PoC section of a report and to look for terms like "parameter" or "argument" in the vulnerability explanation. If a field is not explicitly mentioned in the text, the LLM is instructed to populate it with an empty string. This fine-grained guidance ensures precise extraction of key information from descriptions and vulnerability reports. Even when faced with difficulties such as broken links or brief descriptions in older reports, the vulnerability analyzer LLM can still maximize the completeness of the signature based on the available information.

2) Interface Signature(IS): In the black-box vulnerability testing of IoT devices, interfaces often serve as the key entry point for discovering vulnerabilities. Besides the affected vendor and product, IoTBec's IS is primarily composed of three components: the POST interface, the parameters in the packet (specifically, only parameter names), and context information from the device's UI.

The POST interface is the most direct and vital identifier for an interface. Concurrently, we use the structure and content of the data packet as a seed for subsequent fuzzing, and we record its data format in the IS. Notably, for the data packet, we include only the parameter names in the IS, excluding their specific values. This decision is based on two primary considerations:

1) **The dynamic and unstable behaviors of parameter values:** The parameter values for an IoT device's interface often change with each packet capture. They lack sufficient stability and objectivity to serve as a reliable feature of the interface.
2) **Optimizing the generalization capability of the LLM:** A data packet with specific values could, to some extent, constrain the generalization capability of the LLM. The parameter values obtained from packet captures typically lack a reference value for vulnerability discovery. Conversely, by using the critical parameter information provided by the VS and the context from the vulnerability report, the LLM can more effectively mutate high-quality payloads.

In addition to the POST interface and the network data packet, the navigation-bar path and trigger button information from the device's UI are also critically important. Taking the Tenda AC18 router as an example, the two POST interfaces */goform/SetNetControlList* and */goform/setPptpUserList* are formally similar, and their data packet parameters (e.g., both are named *list*) are identical. These two interfaces can easily lead to confusion during the subsequent signature matching algorithm. However, the contextual information from the device's UI corresponding to these two interfaces is significantly different: */goform/SetNetControlList* corresponds to the 'Save' button on the 'Bandwidth Control' sub-page under 'Advanced Settings', while */goform/setPptpUserList* corresponds to the '+New' button on the 'PPTP Server' sub-page under 'VPN.' By characterizing the interface's corresponding navigation-bar path and trigger button field, we can more accurately represent the unique functional context of the interface. This approach enables us to effectively distinguish between situations where the back-end interface and the data packet appear similar but serve different functions. As a result, it significantly boosts the precision of the IS and reliability of the matching process. The construction process for the IS is illustrated in Figure 3.

3) Vulnerability Interface Signature(VIS): The VIS is at the core of the IoTBec framework. It is designed to bridge the gap between abstract vulnerability knowledge (VS) and specific device interface details (IS), thereby enabling precise recurring vulnerability detection in black-box scenarios. For a given device, IoTBec associates the extracted VS with its corresponding IS using the affected vendor, product, and POST interface as key linking fields. We integrate this information through a logical union operation, successfully combining the vulnerability's characteristic features with the interface's specific information. This forms a more comprehensive and indicative signature, providing both the conceptual approach and concrete solution for C1 and C2.

Ultimately, a complete VIS contains the following key fields: vendor, affected product, POST interface, form parameter (parameter names only with parameter format), and the UI navigation-bar path and trigger button field. Through this meticulously designed fusion, the VIS not only inherits the precise description of the vulnerability type and parameters from the VS but also incorporates the unique characterization of the interface's function and UI context from the IS. This ensures that during the subsequent signature matching process, interfaces with the same or similar vulnerabilities can be identified more accurately. It also provides an exact attack vector and guidance on mutations for the fuzzing LLM.

B. Signature Matching Algorithm

This subsection describes the signature matching algorithm used in IoTBec to identify potential recurring vulnerabilities, as shown in Algorithm 1. The algorithm calculates a weighted fuzzy similarity score between the IS of the device under test and the entries in the ground truth database of VIS. A matching decision is then made based on a predefined threshold.

To calculate this score, IoTBec leverages the string fuzzy matching capabilities provided by the rapidfuzz library to independently score each key field in the signatures, which are then aggregated according to predefined weights. Specifically, for any two signature entries being compared (one IS from the test device and one VIS from the ground truth), the algorithm iterates through their common key fields. For each field, it extracts its string representation and calculates a fuzz.ratio similarity score between them (ranging from 0 to 100). This score is then multiplied by the predefined weight corresponding to that field. After all the field scores are weighted and accumulated, the sum is divided by the total weight to yield the final weighted fuzzy similarity score.

Considerations for the Weights: To ensure a comprehensive and precise matching process, we adopted a multidimensional equal-weighting strategy. The three core sets of IS—the POST interface, the form parameters with format, and the UI context information—are assigned equal weights. Each set represents a

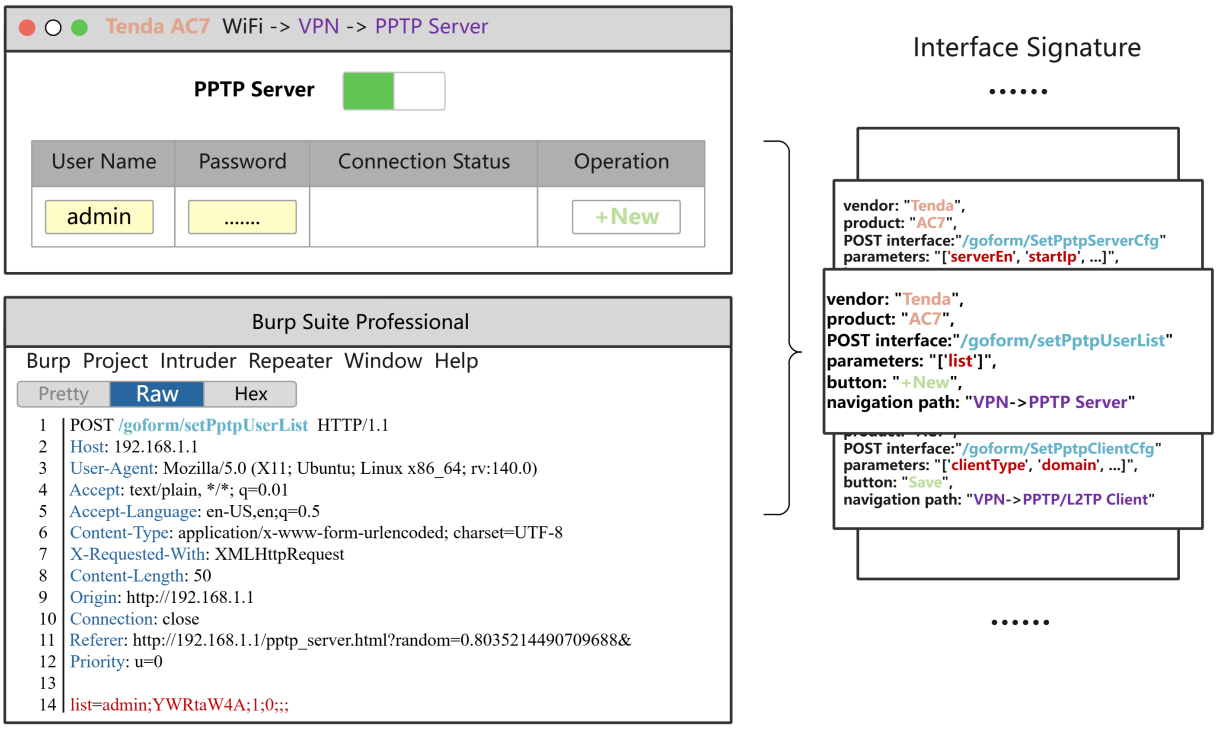

Fig. 3. Illustration of Interface Signature Extraction

distinct and indispensable dimension of an interface's identity. Furthermore, we observed that the complexity of device UI context, POST Interfaces, and the size of data packets can vary significantly across different IoT devices. Therefore, the equal-weighting strategy serves as a robust solution, ensuring that no single feature is underestimated or overrated in the matching process, which is crucial for precisely characterizing and distinguishing interfaces and, consequently, identifying recurring vulnerabilities accurately.

A match is identified when the weighted fuzzy similarity score exceeds the threshold, indicating that the IS under test may contain a recurring vulnerability similar to a known entry in the ground truth database.

Selection of the Similarity Matching Threshold: Due to the significant variations of the three core sets of IS, it is pretty challenging to determine a single, "perfect" threshold that is universally applicable to all situations. Through extensive real-world testing and iterative validation, we found that a range between 70 and 80 is most suitable. Ultimately, we set the matching threshold to 75. This choice is intended to strike the following balance:

1) **Minimizing Irrelevant Signature Matches:** A threshold of 75 is sufficiently high to filter out irrelevant signature matches that lack effective guidance for crafting mutations. By setting a relatively high threshold, IoTBec minimizes subsequent fuzzing attempts and improves overall testing efficiency.
2) **Enabling the Discovery of Novel Vulnerabilities:** Importantly, a threshold of 75 also permits the matching of vulnerabilities that have a marginally weaker correlation but still hold potential value. While these matches may not represent recurring vulnerabilities, the similarity in their interface characteristics and parameters is sufficient to provide effective mutation strategies for the subsequent LLM-driven fuzzing. This greatly increases the likelihood that IoTBec will discover novel and variant vulnerabilities, surpassing the limitation of traditional recurring vulnerability detection, which is confined to "finding only duplicated" vulnerabilities.

Ultimately, the choice of a 75 threshold represents a strategic compromise that balances the two competing objectives of precision and novelty discovery.

C. LLM-Driven Fuzzing Strategy

After the signature matching algorithm identifies a potential recurring vulnerability in the device under test, IoTBec proceeds to the vulnerability validation stage. The core of this stage is LLM-driven fuzzing, which aims to intelligently generate and execute highly targeted fuzzing payloads based on the matched vulnerability information. The system iterates through the IS of the device under test. If a particular IS successfully matches a recurring vulnerability (represented by VIS) in the ground truth database, IoTBec activates the LLM-driven fuzzing module. At this point, the fuzzing LLM assumes the role of a professional security fuzzing expert, and its input includes the following key information:

1) **Structured information of the matched recurring vulnerabilities:** This includes the detailed fields from

the VIS of the matched vulnerability, such as the POST interface, critical parameters, and network packets.

2) **The associated original CVE report text:** The complete, original CVE report provides the LLM with rich context and potential exploitation details, supplementing the semantic nuances and attack methods that the VIS may not fully encompass.

These inputs collectively form the knowledge base for the fuzzing LLM's intelligent decision-making and payload construction. The fuzzing LLM follows a meticulously designed prompt to construct the fuzzing payloads in a series of steps:

Seeds Template Preparation: First, the fuzzing LLM analyzes the data packet provided in the IS. This packet's parameters provide the LLM with a basic structural seeds template for generating fuzzing payloads. This ensures that the generated payloads maintain consistency with the target interface's expected input format, thereby improving their parsability.

Intelligent Fuzzing Target Identification: The fuzzing LLM intelligently identifies and prioritizes the targets for fuzzing. It first prioritizes parameter mutation by targeting the key parameters explicitly specified in the matched VIS. Notably, suppose a target parameter that is expressly mentioned in the VIS or vulnerability report is missing from the IS data packets. In that case, the fuzzing LLM will proactively add this parameter to the final payload structure for mutation. Subsequently, leveraging its semantic understanding, the fuzzing LLM further mutates other potentially vulnerable parameters closely related to those explicitly indicated in the matched VIS. Semantically related parameters are more likely to be processed with similar code logic, thereby increasing the likelihood of similar vulnerabilities, for instance, parameters *startIp* and *endIp*, or *security* and *security_5g*. Furthermore, drawing on our experience in vulnerability discovery, we also provide the LLM with several parameter names commonly associated with vulnerabilities (parameters containing keywords such as *name*, *urls*, *time*, *ip*, and *ssid*) to enhance the effectiveness of mutations. This intelligent identification mechanism significantly broadens the scope of the fuzzing, enabling it to discover potentially vulnerable parameters that are not explicitly indicated in known vulnerability reports.

Targeted Payload Generation: The fuzzing LLM's final task is to construct a fixed number of distinct fuzzing payloads. These payloads are highly targeted and diverse, with each one typically focusing on mutating different parameters. Taking overflow vulnerabilities as an instance, the fuzzing LLM uses a uniform placeholder {payload} to avoid directly outputting the actual overflow string (e.g., *"A"*1000*). This placeholder precisely indicates the injection point for the actual fuzzing string, offering significant flexibility by allowing a subsequent fuzzing engine to insert various types of test data. To efficiently reproduce known vulnerabilities, particularly for standard crash reuse scenarios, the fuzzing LLM prioritizes ensuring that its first generated payload is a replica of the parameter names and values from the matched VIS. This approach guarantees high-precision replay for known vulnerabilities and provides a reliable baseline for subsequent mutated payloads. These generated payloads are then automatically sent to the target IoT device. IoTBec's fine-grained monitoring module (detailed in Appendix subsection B) monitors the device's responses and behavioral anomalies in real-time to determine if a vulnerability has been successfully triggered.

The overall process of IoTBec provides a solution for C3. Its efficiency and robustness in black-box IoT device vulnerability detection will be comprehensively demonstrated and validated in the next section—Evaluation.

Algorithm 1 Recurring Vulnerability Matching Algorithm

Require:
$\mathcal{I} = \{IS_i\}$: target IS;
$\mathcal{V} = \{VIS_j\}$: VIS Ground Truth
W: field weights;
τ: similarity threshold
Ensure: Matches : $IS_i \mapsto \{(VIS_j, sim_{ij}) \mid sim_{ij} \geq \tau\}$
function WEIGHTEDSIM(IS, VIS, W)
  $score \leftarrow 0$, $weight \leftarrow 0$
  $\mathcal{F} \leftarrow$ **fields**$(IS) \cap$ **fields**(VIS)
  for each $f \in \mathcal{F}$ **with** $w = W[f]$ **do**
    $s \leftarrow$ `fuzzy_ratio`$(IS[f], VIS[f])$
    $score \leftarrow score + s \times w$
    $weight \leftarrow weight + w$
  end for
  return $score/weight$ **if** $weight > 0$ **else** 0
end function

function RECURRINGMATCH($\mathcal{I}, \mathcal{V}, W, \tau$)
  Initialize empty map Matches
  for IS **in** $\mathcal{I}$ **do**
    for VIS **in** $\mathcal{V}$ **do**
      $sim \leftarrow$WEIGHTEDSIM(IS, VIS, W)
      if $sim \geq \tau$ **then**
        append (VIS, sim) to Matches$[IS]$
      end if
    end for
  end for
  return Matches
end function

IV. EVALUATION

A. Experiment Setup

We implemented IoTBec in approximately 4,500 lines of Python code. Its core modules include: an information collector based on Selenium 4.31.0 and Burp Suite Professional v2023.4.5; a Vulnerability Signature generator, a signature similarity matching engine, and a fuzzing payload generator; and a back-end that integrates interaction with a simulated environment and vulnerability alert monitor. For all LLM operations within IoTBec, we used the general-purpose Large Language Model, OpenAI gpt-4 [32], with the temperature parameter set to 0.1 to ensure highly consistent output.

1) Vulnerability Signature Construction: Our VS construction process is as follows: For each CVE ID, we first obtain its official description from cve.mitre.org and extract detailed information from the public vulnerability reports linked in its References field (e.g., vendor security advisories, third-party security research reports, Exploit-DB entries, etc.). Subsequently, in conjunction with structured prompts and the vulnerability analyzer LLM, we automatically parse this unstructured text. This process is designed to extract key vulnerability features precisely—including the vulnerability type, affected versions, the vulnerability's POST interface, and critical parameters—ultimately generating a JSON-structured Vulnerability Signature.

2) Interface Signature Acquisition: For the construction of an IS for a device under test, the process begins with two manual preliminary steps: an analyst first obtains login authentication, and the device's UI language is set or translated into English. Following these steps, IoTBec then uses a depth-first traversal to navigate all accessible interfaces, extracting their contextual information of the hierarchical navigation path and the key buttons. Subsequently, by configuring the browser's traffic to be routed through Burp Suite as an HTTP/HTTPS proxy, we capture the raw network request triggered by each functional interaction (such as a button trigger or form submission). From this request, we extract the POST interfaces and their associated parameter information. These network data packets are then associated with the corresponding UI contextual features to form a unique IS. This process ensures a comprehensive and structured characterization of the device's interactive interfaces in fully black-box scenarios.

3) Experiment Environment and Device Selection: All experiments were conducted on a Windows 11 host machine equipped with an Intel Core i7-1260P CPU (12 cores, 16 threads) and 48 GB of RAM. The core logic of IoTBec, including the acquisition of CVE and interface information, LLM interaction, signature matching, and payload generation, was executed directly on the host machine. Vulnerability verification and monitoring were performed within an Ubuntu 22.04 LTS virtual machine running on VMware Workstation Pro 17. This VM was allocated 4 CPU cores and 16 GB of RAM. All tools used in the comparative experiments were also run in this same Ubuntu VM environment. The maximum execution time for all fuzzing tools was set to 6 hours.

Our experiments revealed that executing high-risk vulnerability tests on physical IoT devices may cause irreversible damage. For instance, we discovered that the buffer overflow vulnerability CVE-2024-42545 [33] caused a TOTOLINK A3700R device to crash, rendering it unrecoverable even via the physical reset button. As this vulnerability has been fixed in the latest firmware version, we disclose this information in our paper. To ensure the controllability and reproducibility of our research, all vulnerability testing and validation were conducted within a QEMU [34] system emulation environment. We used binwalk [35] to extract the target IoT device's firmware and obtain its filesystem, which we then used to perform high-fidelity, full-system emulation within a QEMU virtual machine. The QEMU was configured for each experiment, including accurately matching the target device's CPU architecture (e.g., MIPS, ARM) and emulating the necessary network cards and bridges. Although we used firmware to set up the emulated environment, it must be emphasized that IoTBec operates as a black-box tool throughout the entire detection process. It only relies on accessing the IoT device's web interface, publicly available CVE descriptions, and vulnerability reports for its input, which aligns with our threat model.

We selected a total of 27 devices from five mainstream [36]–[39] IoT vendors (Tenda, TOTOLINK, D-Link, TP-Link, and Linksys) for our tests. Among these, 6 devices were used to construct the ground truth database of Vulnerability Interface Signatures, which serves as the baseline for recurring vulnerability detection. The remaining 21 devices were used as the test set to evaluate IoTBec's vulnerability discovery capabilities. We consulted vendor vulnerability information from VULDB to select products with a high number of reported vulnerabilities for constructing the ground truth set (e.g., Tenda AC18). The devices for the test set were chosen from the same product series as those in the ground truth set (e.g., Tenda AC5–AC10, AC15).

B. Vulnerability Detection Experiment

1) VIS Ground Truth Construction: To construct a ground-truth baseline for recurring vulnerability detection, we selected five IoT vendors that are representative across both market share and the number of reported vulnerabilities: Tenda, TOTOLINK, D-Link, TP-LINK, and Linksys. From 6 products across these vendors, we collected 261 CVE IDs over the last 5 years. We used this data to construct a ground truth database of VIS.

During the VS construction process, we utilized the vulnerability analyzer LLM to parse the descriptions and public vulnerability reports associated with these CVE IDs. The experimental results indicate that the vulnerability analyzer LLM effectively extracted structured Vulnerability Signatures from 90.8% of CVE descriptions and vulnerability reports. The few instances of extraction failure were primarily due to overly brief CVE descriptions, missing associated vulnerability reports, or, in a tiny number of cases, erroneous information in the CVE description or report. Concurrently, we successfully extracted IS from all accessible interfaces of the 6 products to construct the ground truth. This success ensures the feasibility of IoTBec's method for generating valid VIS by integrating the VS and IS.

In terms of efficiency, the vulnerability analyzer LLM required only about 5 seconds on average to extract a VS for a single CVE ID, with the prompt and output token counts typically not exceeding 1,200 and 200, respectively. In contrast, extracting all IS for a single device took approximately 1,200 seconds on average, a process that does not require the LLM and is handled entirely by the information collector. Table I details key data for a subset of vendors and products during the ground truth construction process, where *Time* represents the

total duration for extracting all VIS for a single device. This further validates the effectiveness and efficiency of IoTBec in information extraction.

2) Vulnerability Detection Based on Ground Truth: For the test set, we selected IoT devices from the same product series as those in the ground truth for vulnerability testing. For each device in the test set, we extracted its IS using the same method during the ground truth construction phase. The success rate for IS extraction was 86.42%. Notably, for 2 of the devices, we were unable to extract complete IS because their data packets were AES-encrypted. These 2 devices will be tested and analyzed separately in our robust experiment.

We then performed algorithmic matching between the IS of the devices under test and the VIS in the ground truth database. The vulnerability detection results for these test devices are summarized in Table II, where *IS* represents the number of IS successfully extracted from the device's interfaces. *Match* represents the number of known CVEs in the ground truth database that the product's IS matched. This directly reflects IoTBec's ability to identify known recurring vulnerabilities. Upon a match, IoTBec invokes the fuzzing LLM. The LLM uses the network data packet corresponding to the IS as a seed for fuzzing, mutating it in conjunction with the matched recurring vulnerability information VIS and its generalization capabilities. For each Match, the LLM generates 7 distinct test payloads. The *LLM Payloads* shows the total number of payloads generated by the fuzzing LLM during device testing. Typically, for payloads generated from a single match, the LLM's input prompt tokens do not exceed 2,500, and the output tokens do not exceed 250. *Alarm* records the total number of alerts issued by IoTBec's monitoring module, while *RC* represents the number of root-cause vulnerabilities found among those alerts. We determine a root-cause vulnerability by a unique tuple (POST interface, parameter) associated with each alert. Each identified RC is then manually verified to confirm its validity and distinctness. *TP* represents the true positive RC vulnerabilities discovered by IoTBec. In this experiment, IoTBec's alert precision reached 100%, a result attributable to its robust monitoring rules. *Assigned as CVE* denotes the number of these vulnerabilities that have been assigned a CVE number.

The execution time, denoted as *Time*, includes the IS matching time, the LLM payloads generation time, and the test payload transmission time. Since the response time for the same payload can vary across IoT devices due to differences in architecture and functionality, we adopted a uniform, reasonable time measurement method. We counted the total number of requests sent to the target device and factored this into the total duration at a calculated rate of one response per second. To ensure thorough payload verification, IoTBec uses Python's requests library to repeatedly send each payload generated by the LLM to the test IoT device 7 times. If IoTBec's monitoring module issues an alert during this process, the testing for that specific payload is immediately terminated; otherwise, all 7 transmissions are completed. Appendix subsection C presents the detailed results of the vulnerabilities discovered by IoTBec. In terms of efficiency, IoTBec required an average of 101 seconds and 11 test payloads to discover each root-cause vulnerability throughout the experiments. This demonstrates the high detection efficiency of our black-box recurring vulnerability detection framework.

C. Comparison Experiment

To comprehensively evaluate IoTBec's vulnerability discovery capabilities in black-box scenarios, we conducted a comparative experiment against two representative, open-source black-box fuzzing tools: boofuzz and Snipuzz. It is important to note that this comparative experiment did not include grey-box or other black-box testing tools that require firmware acquisition (such as Firm-AFL and Labrador), as their testing prerequisites differ from IoTBec's real-world black-box, firmware and source-code independent paradigm. This comparative experiment primarily focuses on two metrics: the number of vulnerabilities discovered (*Vulnerability*) and the recall rate (*Recall*). For the evaluation of the recall rate, since it is impossible to pre-determine the total number of vulnerabilities on a device to serve as an absolute ground truth, we defined the ground truth for this experiment as the total set of root-cause vulnerabilities discovered by all participating tools combined (IoTBec, boofuzz, and Snipuzz). Based on this collective ground truth, we calculated the proportion of vulnerabilities found by each tool and used it as the recall metric. Table III presents a detailed comparison of IoTBec against the existing black-box fuzzing tools in terms of vulnerability discovery capability and recall rate.

Experimental results show that IoTBec[1] (using gpt-4 as the fuzzing LLM) discovers over 7 times more vulnerabilities than boofuzz and Snipuzz, with a 93.37% recall rate. While IoTBec[2] (using gpt-3.5-turbo) also achieves a 74.49% recall rate, IoTBec[3]'s recall (using chatgpt-4o-latest) is almost identical to that of IoTBec[1]. Based on these results, IoTBec's high efficiency compared to existing tools stems from two main factors. First, in terms of detection scope, boofuzz and Snipuzz must perform exhaustive testing on every interface, resulting in extensive detection with low efficiency. In contrast, IoTBec utilizes its unique VIS mechanism to perform focused testing only on interfaces that match a known recurring vulnerability. This approach significantly reduces the detection scope, enabling deeper exploration and higher discovery rates. For example, on the Tenda FH1201, boofuzz only found the overflow vulnerability in the *page* parameter of the */goform/RouteStatic* interface after exhaustive fuzzing. In contrast, IoTBec matched the IS of */goform/RouteStatic* with the VIS of vulnerability CVE-2023-37714 [40] in Tenda FH1202 and discovered the recurring overflow vulnerability in the *page* parameter with the first payload. By focusing on the */goform/RouteStatic* interface, IoTBec successfully triggered one additional overflow vulnerability in the *mitInterface* parameter with the fifth payload generated by the fuzzing LLM. This vulnerability has been assigned as CVE-2024-41464 [41], which is rated as CRITICAL with a CVSS 3.1 score of 9.8. Second, IoTBec employs a more intelligent payload generation

TABLE I
OVERVIEW OF VIS GROUND TRUTH CONSTRUCTION

ID	Vendor	Product	CVE	VS	Interface	IS	Time(s)	VIS
1	Tenda	AC18 V15.03.05.05	87	75	42	42	1702	31
2	Tenda	FH1202 V1.2.0.14_V1.2.0.14	40	39	43	43	1490	8
3	TOTOLINK	A3700R V9.1.2u.5822_B20200513	30	27	30	30	1187	17
4	D-Link	DIR-816A2_FWv1.10CNB05	43	35	22	22	893	5
5	TP-Link	TL-WR841ND_V11_150616	8	8	36	36	1345	2
6	Linksys	RE7000 1.1.05.003	53	53	8	8	847	6

TABLE II
IOTBEC VULNERABILITY DETECTION RESULTS

ID	Vendor	Interface	IS	Match	Alarm	RC	TP	Assigned As CVE	LLM Payloads	Time(s)
1	Tenda	336	262	148	452	136	136	122	1701	9056
2	TOTOLINK	61	61	7	16	4	4	4	84	480
3	D-Link	36	36	6	17	5	5	5	56	290
4	TP-Link	80	80	5	4	2	2	2	35	227
5	Linksys	32	32	12	42	36	36	36	168	924

strategy. It not only utilizes matched vulnerability information for efficient payload reuse but also leverages the fuzzing LLM to intelligently infer potential vulnerabilities in associated parameters. For instance, in our experiment, the IS of Tenda AC7 */goform/WifiExtraSet* interface matched the Tenda AC18 VIS with the critical parameter of *wpapsk_crypto_5g*. Based on its semantic understanding capabilities, the fuzzing LLM in IoTBec tested the associated *wpapsk_crypto* parameter and newly discovered a vulnerability assigned as CVE-2024-2899 [42], which is rated as HIGH with a CVSS 3.1 score of 8.8. In contrast, the payload mutation strategies of boofuzz and Snipuzz lack this kind of intelligent guidance, making them unable to discover such vulnerabilities within a limited timeframe.

D. Robust Experiment

We also performed a robustness test on IoTBec. In previous tests, we identified two devices whose captured data packets were AES-encrypted, preventing us from obtaining complete cleartext data. In this scenario, other black-box testing tools were rendered almost entirely ineffective. However, IoTBec was still able to utilize the incomplete IS for algorithmic matching and successfully discovered 16 RC vulnerabilities. The experimental results are presented in Table IV. This result demonstrates IoTBec's significant resilience when faced with the challenging scenario of encrypted data streams.

Additionally, we evaluated the robustness of IoTBec when faced with missing or erroneous CVE reports. As described in the VIS Ground Truth Construction section, our dataset already included cases where CVE descriptions were overly brief, public vulnerability reports were missing, or the CVE details themselves contained inaccuracies. Despite these difficulties, IoTBec still demonstrated strong robustness when extracting VS from this imperfect information.

E. Ablation Experiment

To better understand the impact of each key field within the IS on IoTBec's performance, we conducted an ablation experiment on several representative devices from each vendor. We designed four experimental groups, each compared against a baseline: the baseline group used the complete IS for matching. The other three groups each involved removing one key component from the IS: the POST interface, the data packet (form-data), or the corresponding UI context information (navigation-button). Table V presents the ablation experiment results of Tenda AC7.

Results illustrate that excluding essential fields from the IS leads to a pronounced increase in the total number of matched CVEs in the ablation groups, as reflected by elevated *Match* counts. This indicates that the removal of these fields leads to over-generalization in the matching process, resulting in a higher number of meaningless matches with irrelevant vulnerabilities. Consequently, IoTBec must expend more resources for the fuzzing LLM to generate a larger number of fuzzing payloads. However, despite this increased investment of computational resources, the number of actual *RC* vulnerabilities discovered does not increase. This directly impacts the overall efficiency of the tool, increasing the *LLM Payloads* and the *Time* for vulnerability detection accordingly.

As stated in the section III regarding the rationale for selecting these fields, the results of this ablation study provide strong evidence that the POST interface, the data packet (form-data), and the UI context information (navigation-button) all play a crucial role in the comparison of IS. Collectively, they ensure the specificity of the signatures and the precision of the matching process, which effectively guides subsequent LLM-driven fuzzing to achieve efficient vulnerability discovery.

V. DISCUSSION AND LIMITATION

The successful implementation of IoTBec validates the feasibility and high efficiency of recurring vulnerability detection on black-box IoT devices. Our experiments confirm that this paradigm significantly enhances the efficiency of black-box testing. Compared to SOTA tools boofuzz and Snipuzz, IoTBec discovered over 7 times more vulnerabilities. This

TABLE III
COMPARISON EXPERIMENT WITH SNIPUZZ AND BOOFUZZ

Vendor	Vulnerability			Recall (%)				
	Snipuzz	boofuzz	IoTBec[1]	Snipuzz	boofuzz	IoTBec[1]	IoTBec[2]	IoTBec[3]
Tenda	6	17	136	4.08	11.56	92.52	73.46	93.88
TOTOLINK	1	2	4	20.00	40.00	80.00	60.00	80.00
D-Link	1	3	5	16.67	50.00	83.33	83.33	83.33
TP-Link	0	0	2	0.00	0.00	100.00	100.00	100.00
Linksys	5	4	36	13.89	11.11	100.00	77.78	100.00
SUM	13	26	183	6.63	13.27	93.37	74.49	93.88

TABLE IV
EXPERIMENTAL RESULTS OF ROBUSTNESS FOR ENCRYPTED DATA PACKETS

ID	Product	Interface	Match	Alarm	RC	TP	LLM Payloads	Time(s)
1	Tenda AC8 V4.0 V15.03.05.05	37	20	47	14	14	203	1508
2	Tenda AC10 V4.0 V15.03.05.05	37	20	10	2	2	203	1691

TABLE V
EXPERIMENTAL RESULTS OF ABLATION FOR THE KEY FIELDS SET OF IS

ID	Product	Ablation	Match	Alarm	RC	LLM Payloads	Time(s)
1	AC7	Baseline	31	76	16	217	1413
1	AC7	no POST interface	34	85	16	238	1552
1	AC7	no form-data	32	68	16	224	1528
1	AC7	no navigation-button	37	88	16	252	1649

work also charts a clear path for broader research in the security field. For instance, the external VIS perspective could be applied to enhance white and grey-box testing, and the core concept could be extended to other black-box scenarios such as Industrial Control Systems (ICS).

Despite its effectiveness, IoTBec has several limitations due to the stringent black-box scenarios. First, the framework's applicability depends on the availability of two core information sources: open-source CVE reports and device UI. Although our survey (detailed in subsection II-B) confirms the widespread availability of information, IoTBec's utility would be significantly constrained if a target device's vendor lacks a sufficient history of published CVEs or the device UI. Additionally, the lack of firmware and source code limits the scope of detection, preventing IoTBec from identifying hidden interfaces or debug paths unlinked within the frontend UI. Second, due to variations in UI structure and POST interfaces across different vendors, IoTBec's effectiveness is currently limited to intra-vendor recurring vulnerability detection. Cross-vendor detection is generally unreliable with the current design. We will address this limitation by optimizing our signature design and matching algorithms to improve generalization and enable reliable cross-vendor detection in future work. Finally, IoTBec still requires manual intervention for key processes, such as the preliminary device setup in subsubsection IV-A2 and the initial selection of devices for the ground truth database. IoTBec's performance also depends on the capabilities of the specific LLM. The framework relies on the LLM for vulnerability signature extraction and targeted payload generation. Therefore, selecting a sufficiently powerful model, such as OpenAI gpt-4, is essential for achieving optimal detection performance.

VI. RELATED WORK

A. IoT Firmware Vulnerability Detection

In recent years, as IoT devices have become widely used across various domains, their security threats have grown increasingly prominent. Currently, vulnerability discovery methods for IoT devices primarily fall into two categories: static analysis and dynamic analysis, collectively known as model-based vulnerability detection. Static analysis involves examining IoT device firmware or source code to find potential security vulnerabilities without executing the program. Typical static analysis methods include symbolic execution and taint analysis, primarily used for discovering taint-based vulnerabilities, such as KARONTE [5] and SaTC [6]. However, these methods often suffer from path explosion, leading to low analysis efficiency [43]. In contrast, dynamic analysis detects program behavior during actual execution, triggering anomalies through input mutation and other techniques to uncover exploitable security flaws. These methods typically require execution within simulated or rehosted firmware environments [10], [44]. Yet, due to the complexity of emulating complete systems and external interactions, their operation is complex, and applicability is limited. Furthermore, fuzzing, a standard dynamic analysis method, heavily relies on high-quality initial seeds and effective mutation strategies. In black-box IoT firmware scenarios, the lack of firmware information

makes it extremely challenging to acquire broad-coverage seeds. Additionally, fuzzing often depends on Sanitizer [45] tools (e.g., AddressSanitizer [46], UndefinedBehaviorSanitizer [47]) for runtime anomaly monitoring. However, deploying and running these tools on resource-constrained and highly enclosed black-box IoT devices is very difficult, which further limits the effectiveness of fuzzing.

In the domain of grey-box and black-box IoT device vulnerability discovery, dynamic fuzzing is currently widely adopted. For instance, FirmAFL targets embedded firmware and can achieve efficient vulnerability discovery when combined with device emulation. Boofuzz [11] provides an automated network protocol fuzzing framework; Snipuzz optimizes mutation strategies through message fragment feedback, enabling testing and zero-day vulnerability discovery without firmware. However, in black-box testing with limited prior information, the efficiency and vulnerability coverage of these tools are generally limited. LABRADOR's core idea is to leverage string information from device responses to infer internal firmware execution paths, thereby assisting vulnerability discovery. Nevertheless, there is still a gap between this method and black-box testing scenarios, for LABRADOR requires the static analysis of the firmware, including string extraction and the construction of simplified control flow graphs.

In summary, while existing black-box IoT vulnerability detection tools have improved the ability to discover new vulnerabilities, they generally suffer from persistent issues such as reliance on firmware (even partial reliance) and limited capabilities for retesting known vulnerabilities. This makes them insufficient to meet the demands for efficient, automated, and recurring detection in completely black-box, firmware and source-code independent scenarios.

B. Code Clone and Recurring Vulnerability Detection

Code Clone Detection, a classic software engineering technique, has long been used to identify redundant and repetitive logic in source code. Mainstream methods include string-based matching [48], Abstract Syntax Tree (AST) structure comparison [49], and clone detection based on Program Dependency Graphs (PDGs) [50]. These methods are widely applied in scenarios such as code refactoring, vulnerability propagation analysis, and software maintenance. However, virtually all of them rely on firmware or source code, making them challenging to migrate to black-box IoT device scenarios where source code is unavailable.

For IoT devices, recurring vulnerability detection, as an essential supplement to model-based vulnerability detection, has been proven to enhance vulnerability discovery capabilities effectively. This technique leverages information from known vulnerabilities to discover new ones with similar flaws, thereby significantly boosting the effectiveness of existing detection tools. For example, Tracer [51] increased the vulnerability detection rate of CodeQL by 55.8%, while MVP [52] detected an additional 97 vulnerabilities beyond those detected by existing tools. VulMatch [53] focuses on binary-level Vulnerability Signature extraction, enabling it to "precisely locate vulnerability-related binary instructions to generate a vulnerability signature" and achieve vulnerability discovery in commercial-grade firmware. Recurring vulnerabilities also appear frequently in IoT firmware. One of the main reasons for this phenomenon is the widespread adoption of third-party components and open-source libraries, which are repeatedly integrated across multiple vendors and device models. For example, an analysis of over 6,000 firmware images by AutoFirm [54] found that "67.3% of firmwares still use outdated third-party libraries, with an average patch delay of up to 1.34 years." FirmSec [55] further demonstrated that in a dataset of over 34,000 firmware images, component analysis revealed 128,757 vulnerabilities originating from third-party components. This clearly illustrates the reality of code reuse, which repeatedly leads to vulnerabilities appearing across different products and vendors. Furthermore, the IoT firmware supply chain has long suffered from a component lag issue. A report from Palo Alto Networks [56] points out that "many IoT firmwares still use outdated or deprecated libraries/components that are not updated in a timely manner; once a vulnerability is discovered in such a component, it affects numerous devices." Microsoft has further noted that "the continued use of the deprecated Boa web server SDK is widespread across multiple IoT products, constituting a long-term risk" [57].

However, existing recurring vulnerability detection methods all rely on firmware binaries or source code. They are primarily based on static feature matching, making them difficult to apply in a completely black-box IoT device scenario. In real-world testing scenarios involving physical devices where source code, binaries, or even the entire firmware are unavailable, achieving efficient and automated recurring vulnerability detection remains a critical and urgent technical challenge for the field.

VII. Conclusion

This paper proposes IoTBec, an innovative, firmware and source-code independent framework for recurring vulnerability detection in black-box IoT devices. Its core lies in the unique concept of the Vulnerability Interface Signature (VIS), which is constructed entirely from black-box observable information: device POST interface, network data packets, UI context information, and open-source vulnerability reports. By deeply integrating the VIS with LLM-driven fuzzing, IoTBec achieves an end-to-end automated vulnerability detection process, eliminating any reliance on firmware or source code.

The experimental results confirm the superior performance of IoTBec. Specifically, it significantly surpasses existing black-box SOTA methods in the number of discovered vulnerabilities while maintaining high alert precision. This effectiveness is underscored by the identification of high-risk vulnerabilities, leading to the successful assignment of 53 new CVE IDs. Furthermore, IoTBec demonstrates robustness when processing both encrypted data packets and non-ideal, real-world CVE reports. Finally, our ablation studies validate the essential contribution of the key fields within the Vulnerability Information Structure (VIS).

Through its unique firmware and source-code independent paradigm, IoTBec opens up a new direction in the field of IoT security and has significant, far-reaching practical implications for enhancing device security.

ACKNOWLEDGMENT

This work was partially supported by the Program of Key Laboratory of Network Assessment Technology, the Chinese Academy of Sciences, Program of Beijing Key Laboratory of Network Security and Protection Technology, National Key Research and Development Program of China (No.2021YFB2910109) and National Natural Science Foundation of China (No.62202465). We are grateful to our many research partners, including my girlfriend, for their collaboration. We also thank the anonymous reviewers for their comprehensive feedback. Additionally, inspired by my mother, IoTBec emits an audio signal when an alert is triggered, which is implemented by the *gugu()* function in the code repository.

REFERENCES

[1] S. Sinha, "State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally," IoT Analytics, Sep 2024, [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/.

[2] Z. Comeau, "Securing the Smart Home Network: The Risks of the IoT," CEPRO, Jun 2023, [Online]. Available: https://www.cepro.com/news/securing-the-smart-home-network-the-cybersecurity-risks-of-iot/121547/.

[3] S. Lee, "IoT Vulnerabilities Exposed," Number Analytics, Jun 2025, [Online]. Available: https://www.numberanalytics.com/blog/ultimate-guide-iot-vulnerabilities-network-security.

[4] L. O'Donnell, "More Than Half of IoT Devices Vulnerable to Severe Attacks," Threatpost, Mar 2020, [Online]. Available: https://threatpost.com/half-iot-devices-vulnerable-severe-attacks/153609/.

[5] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Karonte: Detecting insecure multi-binary interactions in embedded firmware," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1544–1561.

[6] L. Chen, Y. Wang, Q. Cai, Y. Zhan, H. Hu, J. Linghu, Q. Hou, C. Zhang, H. Duan, and Z. Xue, "Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 303–319.

[7] 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 *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 337–350.

[8] G. Gao, S. Gan, X. Wang, and S. Zhu, "LLMUZZ: LLM-based seed optimization for black-box device fuzzing," in *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2024, pp. 1209–1216.

[9] 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)*, 2024, pp. 1920–1938.

[10] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, "FIRM-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1099–1114.

[11] J. Pereyda, "boofuzz," GitHub repository, Nov 2022, [Software]. Version 0.4.1, Available: https://github.com/jtpereyda/boofuzz.

[12] H. J. Tay, K. Zeng, J. M. Vadayath, A. S. Raj, A. Dutcher, T. Reddy, W. Gibbs, Z. L. Basque, F. Dong, Z. Smith *et al.*, "Greenhouse: Single-Service rehosting of Linux-Based firmware binaries in User-Space emulation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5791–5808.

[13] C. Wright, W. A. Moeglein, S. Bagchi, M. Kulkarni, and A. A. Clements, "Challenges in firmware re-hosting, emulation, and analysis," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–36, 2021.

[14] M. Pease, "Whether you build them or buy them, iot device security concerns," Website, 2023, [Online]. Available: https://www.nist.gov/blogs/manufacturing-innovation-blog/whether-you-build-them-or-buy-them-iot-device-security-concerns.

[15] NCC Group, "Enterprise connected device vulnerability assessment," Report, 2024, [Online]. Available: https://assets.publishing.service.gov.uk/media/681de9fdd9c9bb76078f7e82/Enterprise_Connected_Device_Vulnerability_Assessment_-_NCC_Group.pdf.

[16] K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. Megas, E. Nadeau, and B. Piccarreta, "Considerations for managing internet of things (iot) cybersecurity and privacy risks," NIST IR 8228, 2019, [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2019/nist.ir.8228.pdf.

[17] National Institute of Standards and Technology (NIST), "(draft) establishing confidence in iot device security: How do we get there?" NIST Cybersecurity White Paper (Draft), 2021, [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.05142021-draft.pdf.

[18] S. Nidhra and J. Dondeti, "Black box and white box testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[19] A. Verma, A. Khatana, and S. Chaudhary, "A comparative study of black box testing and white box testing," *International Journal of Computer Sciences and Engineering*, vol. 5, no. 12, pp. 301–304, 2017.

[20] H. Xiao, Y. Zhang, M. Shen, C. Lin, C. Zhang, S. Liu, and M. Yang, "Accurate and efficient recurring vulnerability detection for IoT firmware," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3317–3331.

[21] The MITRE Corporation, "CVE - Common Vulnerabilities and Exposures," Website, 2025, [Online]. Available: https://cve.mitre.org/.

[22] National Institute of Standards and Technology, "CVSS v3.0 Calculator," National Vulnerability Database, May 2024, [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[23] S. Ul Haq, Y. Singh, A. Sharma, R. Gupta, and D. Gupta, "A survey on IoT and embedded device firmware security: Architecture, extraction techniques, and vulnerability analysis frameworks," *Discover Internet of Things*, vol. 3, no. 1, 2023.

[24] Help Net Security, "Helping researchers with IoT firmware vulnerability discovery," Help Net Security, Nov 2018, [Online]. Available: https://www.helpnetsecurity.com/2018/11/19/iot-firmware-vulnerability-discovery/.

[25] The MITRE Corporation, "Request a CVE ID via the CVE Program," CVE® Program, 2025, [Online]. Available: https://cveform.mitre.org/.

[26] ——, "CVE-2022-38314: Tenda AC18 router stack overflow," National Vulnerability Database, 2022, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-38314.

[27] S. Blanton, "IoT Security Risks: Stats and Trends to Know in 2025," JumpCloud, Jan 2025, [Online]. Available: https://jumpcloud.com/blog/iot-security-risks-stats-and-trends-to-know-in-2025.

[28] S. Rogerson, "Number of vulnerable IoT devices increases 136%," IoT M2M Council, Jun 2024, [Online]. Available: https://www.iotm2mcouncil.org/iot-library/news/iot-newsdesk/number-of-vulnerable-iot-devices-increases-136.

[29] VulDB, "The Vulnerability Database," Website, 2025, [Online]. Available: https://vuldb.com/.

[30] A. S. Gillis and K. Yasar, "What is IoT (internet of things)?" Tech Accelerator, Jul 2025, [Online]. Available: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT.

[31] A. Ribeiro, "Forescout's 2025 report reveals surge in device vulnerabilities across IT, IoT, OT, and IoMT," Industrial Cyber, Apr 2025, [Online]. Available:https://industrialcyber.co/reports/forescouts-2025-report-reveals-surge-in-device-vulnerabilities-across-it-iot-ot-and-iomt/.

[32] OpenAI, "GPT-4," Website, Mar 2023, [Online]. Available: https://openai.com/zh-Hans-CN/index/gpt-4/.

[33] The MITRE Corporation, "CVE-2024-42545: TOTOLINK A3700R buffer overflow in setWizardCfg(ssid)," National Vulnerability Database, Aug 2024, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-42545.

[34] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the 2005 USENIX Annual Technical Conference, FREENIX Track*. USENIX Association, 2005, [Online]. Available: http://www.usenix.org/legacy/event/usenix05/tech/freenix/full_papers/bellard/bellard_html/.

[35] ReFirmLabs, "binwalk: Firmware Analysis Tool," GitHub repository, 2024, [Software]. Available: https://github.com/ReFirmLabs/binwalk.

[36] Reuters, "TP-Link faces U.S. criminal antitrust investigation, Bloomberg News reports," Website, April 2025, [Online]. Available: https://www.reuters.com/technology/tp-link-faces-us-criminal-antitrust-investigation-bloomberg-news-reports-2025-04-25/.

[37] Fortune Business Insights, "Wireless Router Market Size, Share, Industry Growth, Report," Market Report, October 2024, [Online]. Available: https://www.fortunebusinessinsights.com/wireless-router-market-107203.

[38] Verified Market Research, "Wireless Router Market Size, Scope, Forecast," Market Report, October 2024, [Online]. Available: https://www.verifiedmarketresearch.com/product/wireless-router-market/.

[39] Market Research Store, "Global Wireless Routers Market Size, Share, Growth, Trends, Report and Forecast 2024-2032," Market Report, January 2024, [Online]. Available: https://www.marketresearchstore.com/market-insights/wireless-routers-market-784987.

[40] The MITRE Corporation, "CVE-2023-37714: Tenda FH1202 router stack overflow," National Vulnerability Database, 2023, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-37714.

[41] ——, "CVE-2024-41464: Tenda FH1201 router stack overflow," National Vulnerability Database, 2024, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-41464.

[42] ——, "CVE-2024-2899: Tenda AC7 router stack overflow," National Vulnerability Database, 2024, [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-2899.

[43] J. Vadayath, M. Eckert, K. Zeng, N. Weideman, G. P. Menon, Y. Fratantonio, D. Balzarotti, A. Doupé, T. Bao, R. Wang *et al.*, "Arbiter: Bridging the static and dynamic divide in vulnerability discovery on binary programs," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 413–430.

[44] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for Linux-based embedded firmware," in *Proceedings of the 2016 Network and Distributed System Security Symposium (NDSS)*, 2016.

[45] D. Song, J. Lettner, P. Rajasekaran, Y. Na, S. Volckaert, P. Larsen, and M. Franz, "SoK: Sanitizing for security," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1275–1295.

[46] Microsoft, "AddressSanitizer," Microsoft Learn, Sep 2024, [Online]. Available: https://learn.microsoft.com/en-us/cpp/sanitizers/asan.

[47] The LLVM Project, "UndefinedBehaviorSanitizer," Clang Compiler User's Manual, 2024, [Online]. Available: https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html.

[48] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[49] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 96–105.

[50] C. K. Roy and J. R. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *2008 16th IEEE International Conference on Program Comprehension*, 2008, pp. 172–181.

[51] W. Kang, B. Son, and K. Heo, "Tracer: Signature-based static analysis for detecting recurring vulnerabilities," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1695–1708.

[52] Y. Xiao, B. Chen, C. Yu, Z. Xu, Z. Yuan, F. Li, B. Liu, Y. Liu, W. Huo, W. Zou *et al.*, "MVP: Detecting vulnerabilities using Patch-Enhanced vulnerability signatures," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1165–1182.

[53] Z. Liu, S. Liu, L. Pan, C. Chen, E. Ahmed, J. Zhang, and D. Liu, "Vulmatch: Binary-level vulnerability detection through signature," in *Network and System Security - 18th International Conference, NSS 2024*, ser. Lecture Notes in Computer Science, vol. 14872. Springer Nature Switzerland, 2024, pp. 409–427.

[54] Y. Chen, F. Ma, Y. Zhang, Y. He, H. Wang, and Q. Li, "AutoFirm: Automatically identifying reused libraries inside IoT firmware at large-scale," 2024. [Online]. Available: https://arxiv.org/abs/2406.12947

[55] B. Zhao, S. Ji, J. Xu, Y. Tian, Q. Wei, Q. Wang, C. Lyu, X. Zhang, C. Lin, J. Wu *et al.*, "A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 442–454.

[56] N. Abu-Ghazaleh and A. Stavrou, "Firmware Forensic Analysis," *TSU Faculty Research Journal*, vol. 9, p. 3, 2017, [Online]. Available: https://digitalscholarship.tsu.edu/cgi/viewcontent.cgi?article=1021&context=frj.

[57] Microsoft Threat Intelligence, "Vulnerable SDK components lead to supply chain risks in IoT and OT environments," Microsoft Security Blog, Nov 2022, [Online]. Available: https://www.microsoft.com/en-us/security/blog/2022/11/22/vulnerable-sdk-components-lead-to-supply-chain-risks-in-iot-and-ot-environments/.

APPENDIX

A. Survey on the Availability of IoT CVE Information for Vulnerability Signature Construction

To validate the feasibility of the Large Language Model (LLM)-driven automated vulnerability signature construction proposed in this paper, we conducted a systematic survey of the availability of CVE information for Internet of Things (IoT) devices. This investigation selected four mainstream manufacturers—Tenda, TOTOLINK, D-Link, and TP-Link—each representative in terms of market share and the number of known vulnerabilities.

We surveyed between May 23 and May 26, 2025, Beijing time, analyzing a total of 500 CVE IDs. For each manufacturer, we randomly sampled 125 CVEs, comprising 50 CVE IDs published between 2024 and 2025, and 75 CVE IDs published in 2023 or earlier. This sampling ratio was chosen based on observed trends: as of July 29, 2025, Beijing time, out of 4239 CVEs from these four manufacturers, 1544 (approximately 36%) were discovered within the last eighteen months. This indicates a significant recent increase in IoT-related vulnerabilities, making it essential to increase the proportion of newer CVEs in our survey.

We used an automated crawler to retrieve the official Description for each CVE ID from cve.mitre.org. We parsed its References field to obtain potential vendor security advisories, third-party reports, security blogs, and other detailed information. Subsequently, we manually assessed whether this information included the critical fields required for vulnerability signature construction: the affected POST endpoint and key parameter information needed by mutation. If the Description or associated report explicitly mentioned parameters crucial for vulnerability triggering, it was marked as "containing key information." Notably, many third-party reports even provided complete Proof-of-Concepts (PoCs), which significantly support the automated extraction of vulnerability signatures.

The survey results, detailed in Table VI, show the success rate of obtaining key information from References, Description, and a combination of both. Our survey reveals a significant improvement in the quality of IoT vulnerability reports. Although early CVE reports (2023 and earlier) presented challenges with information completeness and link accessibility, 63% of these CVE IDs still contained all key information required for vulnerability signatures. More importantly, the IoT domain has a large number of recurring vulnerabilities, many of which have been rediscovered as variants in other devices. With the continuous development of the cybersecurity community and the increasing maturity of vulnerability disclosure practices, the quality of reports from the past two years (2024-2025) has seen a remarkable leap. For CVEs published between 2024 and 2025, we found that up to 84% of cases could obtain all necessary key information for vulnerability signatures from their Description and References. This substantial improvement highlights the industry's progress in standardizing and making vulnerability information more accessible.

Particularly noteworthy is the explosive growth in CVEs issued by CNA VULDB in the IoT domain during 2024-2025, accounting for over 40% of the total vulnerabilities from the four manufacturers above during that period. VULDB is a highly promising CNA organization; compared to MITRE Corporation, it offers advantages such as API-supported submission, faster response times, and transparent processes. VULDB actively assists vulnerability submitters in uploading more comprehensive and standardized CVE Descriptions and ensures the accessibility of reference links. In our survey of 200 CVEs from 2024-2025, 88 were issued by VULDB, and among these, 82 (approximately 93.2%) contained all the necessary information for signature construction.

These survey results powerfully demonstrate that the availability of high-quality open-source vulnerability information is continuously improving. With the rise of emerging CNA organizations like VULDB and the increasing industry emphasis on vulnerability disclosure standardization, we have strong reasons to believe that future CVE reports will be more comprehensive and standardized, and phenomena such as CVE misreporting and incomplete vulnerability information will be significantly reduced. This provides a solid data foundation and the potential for sustainable development for IoTBec's LLM-driven automated vulnerability signature construction.

B. IoTBec's Monitoring Rules

This appendix details the rigorous monitoring strategies IoTBec employs to accurately detect vulnerabilities in the QEMU full-system simulation environment. The design of these strategies stems directly from extensive experience in reproducing vulnerabilities and from deep integration with our simulation setup, ensuring the precision and reliability of our research findings.

1) Buffer Overflow Monitoring Rules: For detecting buffer overflow vulnerabilities in Tenda, TP-Link, and D-Link devices, our monitoring mechanism specifically targets service crashes. In a QEMU full-system simulation, a successful buffer overflow attack typically causes critical services (e.g., the httpd daemon) to crash. IoTBec constructs and sends request payloads via Python's requests module. Upon successful exploitation and service crash, the affected device's IP address becomes inaccessible, causing the Python requests module to capture connection exceptions. The monitor precisely captures these connection exceptions, interpreting them as indicators of a buffer overflow vulnerability hit.

For buffer overflow vulnerabilities in TOTOLINK devices, experiments showed that successful exploitation doesn't always result in a complete system crash. Instead, the vulnerable service returns an HTTP 500 Internal Server Error response code upon successful exploitation. IoTBec's monitoring module also supports using this specific response code as an indicator for buffer overflow vulnerability hits.

2) Command Injection Monitoring Rules: To detect command injection vulnerabilities across Tenda, TOTOLINK, TP-Link, and D-Link devices, IoTBec implements a rigorous out-of-band detection mechanism. After generating a command

TABLE VI
SURVEYS ON OPEN SOURCE CVE VULNERABILITY

Vendor	Year	Interface&parameters in Reference	Interface&parameters in Description	Interface&parameters Available
Tenda	2023 or earlier	39/75	20/75	47/75 = 63%
	2024–2025	40/50	29/50	46/50 = 92%
TOTOLINK	2023 or earlier	45/75	21/75	51/75 = 68%
	2024–2025	34/50	29/50	45/50 = 90%
D-Link	2023 or earlier	19/75	32/75	38/75 = 51%
	2024–2025	37/50	29/50	39/50 = 78%
TP-Link	2023 or earlier	34/75	25/75	49/75 = 65%
	2024–2025	25/50	19/50	37/50 = 74%

injection payload, IoTBec first attempts to write a unique marker file (e.g., 123.txt) to the IoT device's web root directory. Subsequently, IoTBec immediately sends an HTTP GET request to http://{device_IP}/123.txt.

1) If the initial request to http://{device_IP}/123.txt returns a 200 OK response code, it confirms successful arbitrary file creation, indicating a potential command injection vulnerability.
2) Upon this confirmation, IoTBec immediately sends a follow-up command via a specially crafted payload to delete the 123.txt file from the web root.
3) Subsequently, another request is sent to http://{device_IP}/123.txt. If this request returns a 404 Not Found response code, it validates both the command execution capability and the successful deletion of the file, marking it as a confirmed command injection vulnerability.

3) Cross-Site Request Forgery Monitoring Rules: For Cross-Site Request Forgery (CSRF) vulnerabilities involving operations like device reboot or reset, IoTBec's monitoring strategy also includes detecting device crashes. In the QEMU emulation environment, we've observed that specific forged requests, when successfully invoked by the corresponding service, can cause the target system to crash. The monitor identifies such CSRF vulnerability hits by capturing these exception alerts. Notably, this set of experiments was conducted under QEMU user-mode emulation due to specific issues with the simulation environment, enabling precise monitoring of the httpd service behavior and effective identification of vulnerability triggers.

The aforementioned monitoring strategies are meticulously designed and have been refined through extensive reproduction of numerous real-world vulnerabilities within our controlled QEMU simulation environment. As a testament to the effectiveness and reliability of these strategies, the CVEs submitted by IoTBec based on these monitoring principles have received official recognition from MITRE and VULDB, two leading CVE Numbering Authority (CNA) organizations. Looking ahead, we plan to further expand IoTBec's monitoring capabilities by developing mechanisms to detect additional vulnerability types, such as unauthorized access and information disclosure vulnerabilities, to broaden its detection scope.

C. Vulnerabilities Discovered by IoTBec

Our repository presents all vulnerabilities newly discovered by IoTBec in the test set devices, including 47 buffer overflow vulnerabilities, 2 command injection vulnerabilities, and 4 CSRF vulnerabilities.

Ethical Considerations. Our evaluation was executed within a controlled, isolated laboratory environment to guarantee no impact on public networks or external users. All IoT devices utilized in our experiments were legally procured and are the property of the research team. This research strictly adheres to established security ethics guidelines. In accordance with Coordinated Vulnerability Disclosure (CVD) practices, we have responsibly reported all identified vulnerabilities to the relevant vendors. Furthermore, our findings have been submitted to the Common Vulnerabilities and Exposures (CVE) program. We are actively engaged with the vendors to support the remediation of all reported issues. We will track the disclosure status and vendor correspondence for the 53 newly discovered vulnerabilities at https://www.github.com/IoTBec/Reports, updating this repository as details become publicly available.