FirmCred: Detecting Authentication Bypass Vulnerabilities in Firmware from Credential Initialization Perspective

Ziqi Wei
Fudan University
Shanghai, China
zqwei23@m.fudan.edu.cn

Haoyu Xiao
Fudan University
Shanghai, China
hyxiao20@fudan.edu.cn

Yuan Zhang*
Fudan University
Shanghai, China
yuanxzhang@fudan.edu.cn

Chi Chen
Fudan University
Shanghai, China
chichen25@m.fudan.edu.cn

Jiarun Dai
Fudan University
Shanghai, China
jrdai@fudan.edu.cn

Abstract

The widespread deployment of embedded devices has heightened their exposure to remote attacks, often due to the authentication bypass vulnerabilities caused by the use of weak credentials (UoWCs). Existing detection approaches explore three directions: solving concrete credentials as proof, scanning white-listed file contents, or testing with known credentials. However, solving-based approaches are limited by defects of solving techniques; scanning-based approaches overlook unlisted scanning targets; and testing-based approaches fail to discover unknown weak credentials. These limitations cause many UoWC vulnerabilities in firmware to remain undetected.

To address this problem, our insight is that weak firmware credentials fundamentally arise from insecure initialization procedures, and thus can be effectively uncovered by analyzing how credentials are initialized. Leveraging this insight, we present FirmCred, a novel static analysis framework that detects UoWC vulnerabilities in firmware. FirmCred operates in three stages: first, it automatically identifies credential validation sites; second, it identifies validated credentials and constructs the credential initialization expression across firmware components; finally, it infers the deducibility of the initialized credentials. We evaluated FirmCred on 377 firmware images, where it identified 92 previously unknown UoWC vulnerabilities, achieving 63.5% higher recall than state-of-the-art approaches. To date, 25 vulnerability IDs have been assigned to these findings.

CCS Concepts

• **Security and privacy** → *Embedded systems security.*

Keywords

embedded systems security; access control; vulnerability discovery

*Corresponding author.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
CCS '26, The Hague, Netherlands.
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832575

ACM Reference Format:
Ziqi Wei, Haoyu Xiao, Yuan Zhang, Chi Chen, and Jiarun Dai. 2026. FirmCred: Detecting Authentication Bypass Vulnerabilities in Firmware from Credential Initialization Perspective. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3832575

1 Introduction

The rapid increase of Internet of Things (IoT) devices, ranging from portable devices, such as smartwatches, to large machines like transport vehicles, brings more connectivity and convenience to our lives. The market size of IoT is 714.48 billion dollars in 2024 and is expected to reach 4 trillion by 2032 [16]. As these devices are increasingly network-connected, they typically rely on authentication mechanisms to protect systems from being invaded [24]. However, authentication bypass vulnerabilities remain a persistent security threat in IoT ecosystems [68]. Authentication bypass refers to the unintended recognition of an attacker as a valid user, rendering the authentication process ineffective [18, 68]. Among the various causes, the use of weak credentials (UoWC)—such as default or easily deducible passwords [15]—stands out as both widespread and dangerous, enabling effortless privilege escalation [8, 29].

Attackers can exploit a UoWC vulnerability to bypass the firmware authentication and take ownership of victim devices, causing serious cybercrime incidents. One notable example is the Mirai malware [35], which leveraged weak credentials to exploit hundreds of thousands of IoT devices, resulting in millions of dollars in losses and infrastructure destruction [7, 23] or even enabling remote control over vehicles, posing direct threats to personal safety [40]. Given their high prevalence—ranked as the top IoT vulnerability by OWASP since 2018 [28], and severity—our study (in Appendix A) shows 82.7% of them are rated high risk by NVD[26], detecting UoWC in embedded devices is urgent and important.

The detection of UoWC vulnerabilities in embedded devices has been extensively explored [42, 49, 61, 68, 70]. Existing techniques fall into three categories: *testing-based*, *scanning-based*, and *solving-based* approaches. Testing-based approaches [61] attempt to log in using publicly leaked or common default credentials and therefore cannot detect unknown weak credentials. To address this limitation, scanning-based approaches [42, 49] analyze firmware file systems by searching specific files (e.g., /etc/shadow) for default credentials.

However, developers likely store credentials in customized files, binary files, or scripts, rendering this method ineffective. Later work simply scans suspicious static credentials in programs [70], and thus overlooks variable yet guessable credentials that require deep code analysis.

A notable prior work, Firmalice [68], introduces a *solving-based* approach to detect UoWC vulnerabilities. This approach attempts to solve a deterministic input that successfully bypasses the authentication. To achieve this, Firmalice employs symbolic execution[34] to explore the request handling code from user input to post-authentication location. This procedure aims to understand how credentials are validated during authentication, formulating a validation equation that Firmalice then tries to solve. Although theoretically sound, Firmalice faces significant practical challenges that hinder its effectiveness. First, Firmalice requires considerable manual intervention to pinpoint the post-authentication code within the firmware. This lack of automation severely restricts its ability to analyze firmware at scale. Second, by focusing exclusively on network request handling code, Firmalice often overlooks credentials initialized during the firmware's startup process. This prevents it from forming a complete equation necessary for deducing credentials initialized outside its analysis scope. Third, the reliance on symbolic execution[34] makes Firmalice susceptible to well-known issues such as path explosion [38] and the limitations of SMT solver [31]. These factors can lead to exponential computational overhead, rendering false negatives within practical time budgets.

To address the above limitation of prior works, we propose an inferring-based approach. Our key insight is that *a UoWC vulnerability fundamentally results from insecure credential initialization, and can be detected by inferring its deducibility by inspecting its initialization process.* Based on this insight, we propose FirmCred, a static analysis framework for automated UoWC detection in firmware. FirmCred advances the state-of-the-art through three key innovations: First, it **automatically identifies authentication code** based on a dedicated set of semantic features, eliminating manual efforts and enabling large-scale analysis. Second, FirmCred **extends analysis scope** to the entire credential initialization procedure to detect more weak credentials. Finally, FirmCred **replaces heavy symbolic solving with lightweight taint analysis**, which efficiently infers the deducibility of credential initialization.

To demonstrate FirmCred's effectiveness, we conducted a comprehensive evaluation on 377 firmware images sourced from the datasets of FirmAE[60], LFWC[56], and FirmRec[76]. In experiments, FirmCred uncovered 96 UoWC vulnerabilities, including 92 previously unknown vulnerabilities.

Compared with three state-of-the-art (SoTA) methods, FirmCred comprehensively detected all vulnerabilities they discovered, and further identified 87, 61, and 66 more unique vulnerabilities, respectively, highlighting its capability in detecting new UoWC vulnerabilities. Till now, 25 vulnerability IDs have been assigned to our findings.

Contributions. We summarize our contributions below.

- We introduce FirmCred, a novel static analysis framework that detects authentication bypass vulnerabilities caused by UoWC from the perspective of credential initialization.
- We introduce three key techniques to address challenges in this framework: (1) a semantic scoring-based approach to automatically identify credential validation sites in firmware services; (2) a firmware credential access operation (FCAO)-based technique to effectively trace credential initialization procedures across firmware components; and (3) an inferring-based mechanism to identify diverse weak credentials.
- We extensively evaluate FirmCred on a large dataset consisting of 377 real-world firmware images. Our experiments demonstrate its practical effectiveness, leading to the discovery of 92 previously unknown UoWC vulnerabilities, which cover a wide range of weak credential types, including default, hard-coded, and predictable ones.

2 Background

Credential-based Authentication in Firmware. To prevent devices from being controlled by malicious users, firmware commonly authenticates whether a user is allowed to access a specific functionality. This authentication procedure is achieved by validating credentials (e.g., username and password) sent by the user against a device-stored one. Authentication can be implemented in various ways depending on service type and vendor or developer preferences. For example, Telnet and SSH services typically adopt the PAM [17] framework, while web servers can alternatively adopt basic [4], digest[11], or customized form-based authentication [14] mechanisms. However, a flawed authentication implementation may allow an attacker to bypass the enforcement of access control and corrupt the device.

Use of Weak Credential (UoWC) Vulnerability. One of the most severe authentication bypass flaws in firmware arises from the use of weak credentials (UoWC) [15]. A weak credential is one that an attacker can easily deduce without impractical brute-force attempts; a UoWC vulnerability refers to using such a credential for authentication, allowing attackers to bypass authentication. There are three categories of weak credentials: default, hard-coded, and predictable credentials, as defined by the Common Weakness Enumeration (CWE) [36]:

- A *default credential* is a credential that will be employed by different (firmware) installations, and can be changed by administrators. For example, CVE-2024-53937[10] uses a default credential with both user name and password set to `"admin"` in the TELNET service. Although the credential can be changed, users unaware of this weakness may not reset it, allowing easy authentication bypass attacks.
- A *hard-coded credential* refers to an authentication secret embedded directly in firmware, which cannot be modified even by administrators. For example, CVE-2020-10988 [1] is caused because the administrator password is directly encoded as an MD5 hash value in the firmware code, allowing an attacker to infer such a weak credential to bypass the firmware authentication.
- A *predictable credential* is uniquely generated for different devices, but can still be guessed with reasonable efficiency. For example, in CVE-2024-48271[9], the device administrator's password is directly derived from its MAC address. Since the MAC address of a device is public information, an attacker can easily deduce the password and use it to bypass authentication.

In firmware, weak credentials often result from two main causes. First, developers may accidentally leave hard-coded or default credentials in the firmware during testing or development. Second, manufacturers may deliberately use weak or shared credentials to simplify device setup, custom support, or large-scale deployment. Once weak credentials are used in authentication, attackers can easily deduce them by reversing firmware images, which are typically publicly available or can be dumped from devices, causing authentication bypass.

The use of weak credentials (UoWC) remains a widespread and critical issue in embedded firmware. Since 2018, OWASP has consistently ranked it as the top IoT vulnerability [28]. Our analysis of recent UoWC cases in Figure 5 reveals a persistent trend over the past decade, with 82.7% of related vulnerabilities in the NVD [26] rated as high or critical. These findings underscore the urgent need for systematic research into UoWC vulnerabilities.

Problem Scope. This paper focuses on detecting authentication bypass vulnerabilities resulting from UoWC vulnerabilities in firmware. We assume a common threat model that an attacker has access to extractable firmware without source code and aims to deduce weak credentials for authentication bypass. Note that we primarily focus on "initial" weak credentials, which may be mitigated with user configuration (e.g., password reset). However, a recent report [39] shows that only 14% of consumers change default credentials, leaving the remaining devices vulnerable. Therefore, we still consider such weak credentials exploitable.

3 Motivation

3.1 Motivation Example

Figure 1 showcases a UoWC vulnerability for D-Link firmware DSL-6740C (CVE-2024-48271[9]), which allows an attacker to log into the router devices as an administrator. This credential is first initialized by a firmware startup script `S30setnode.sh` (lines 10 – 13). The script then stores the initialized credential in a local database `xmldb`, also known as a device-stored credential. When the Web server `httpd` receives a remote login request from a user, it enters a request handling function `alpha_auth_check`. This function compares the user-provided credential with the device-stored one loaded by calling `xget_str`. The login request is permitted only when these credentials are identical (line 5,6).

In this procedure, the device-stored credential is weak because it is initialized using two deducible operand values, a constant string `"740c"` (line 13) and the device's MAC address (line 10). The constant string can be extracted directly from the firmware, and the MAC address is public to the network, so an attacker can easily acquire these values. Eventually, the attacker can follow the same credential initialization procedure to calculate the actual credential and fully bypass the authentication. This vulnerability exposes all affected devices to remote unauthorized administration and has received a CVSS[6] score of 8.8 with high severity.

Existing methods cannot detect such a UoWC vulnerability without a comprehensive analysis of the credential initialization procedure. Testing-based approaches [61] only used known constant credentials and thus did not discover this vulnerability, because the weak credential is related to the MAC address and varies across devices. Scanning-based [42, 49] approaches and Stringer [70] only

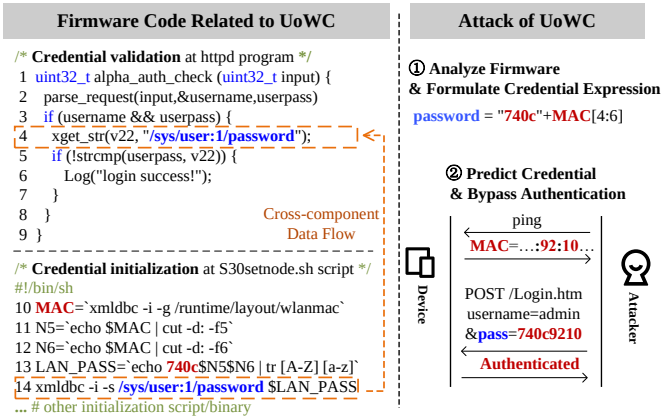


Figure 1: Motivation example.

examined static files or constant string comparisons. Their pattern-based approaches only fit specific default or hard-coded credentials and hardly generalize to UoWC vulnerabilities that involve complex and dynamic credential initialization. The solving-based approach, Firmalice [68], also failed in this case. Specifically, it only explored the request handling code in the `alpha_auth_check` function and overlooked the initialization script `S30setnode.sh`, which contains significant information required for computing a credential solution. In addition, even when Firmalice considered the initialization script, it is still challenging to give a deterministic solution for the credential because the concrete value of the MAC address depends on specific devices. Giving such non-static values is infeasible for static analysis and demands human intervention.

3.2 Insight & Challenges

To address these limitations, our key insight is that weak credentials are determined by its initialization mechanism. Specifically, a credential is vulnerable if its derivation can be reconstructed by an adversary. There are two aspects of this observation: First, when firmware initializes credentials, the underlying logic is often public due to the general availability of firmware images [46]. Second, when initialization relies on low-confidentiality data (e.g., the MAC address in Figure 1, attackers can easily infer the inputs. By combining the exposed logic with predictable data, attackers can derive valid credentials to bypass authentication.

Based on this insight, we propose to detect the UoWC vulnerabilities based on the credential initialization procedure in three steps. (1) *Identify credential validation sites*, such as line 5 in Figure 1; this code location is the starting point for UoWC vulnerability detection. (2) *Track data-flow backward to understand how the credential is initialized*; this step extracts an expression that denotes the credential initialization procedure. (3) *Detect UoWC vulnerabilities by inferring the deducibility of extracted credential expressions*; here, we start by identifying credential validations instead of credential initialization procedures due to the following two considerations. First, according to the definition of the UoWC vulnerability, we must check whether a weak credential is used for vulnerability detection to mitigate false alarms. Second, credential validation is typically associated with rich semantic code features that facilitate

identification, which will be detailed in §4.1. Although this idea seems straightforward, it faces several key challenges when applied to real-world firmware:

C1: Identifying Credential Validation Sites in Binary. In the source code scenario, credential validation is typically identified through names (symbols) of known library functions used by authentication [52, 81]. However, embedded firmware images are generally closed-source and compiled from C/C++ code [68]. As a result, they often lack library documentation, and function names are frequently stripped, rendering function-name-based approaches ineffective in such scenarios. Furthermore, as discussed in §2, manufacturers commonly implement custom authentication procedures with diverse code patterns. This variability hinders identifying validation code using code clone detection approaches[51, 59, 71], often leading to high false negative rates. To overcome these challenges, prior work has relied on manual reverse engineering to identify post-authentication code [68]. While effective, this process demands substantial expert effort and does not scale well.

C2: Handling Cross-component data-flow of Credential Initialization. data-flow analysis in embedded firmware remains an open challenge due to complex binary-to-binary and binary-to-script interactions [67, 73], as well as inherent difficulties such as inter-procedural and pointer analysis [46, 48, 55, 66, 67, 76]. These challenges hinder complete backward tracking of credential initialization. Take Figure 1 as an example, the credential is first initialized within a script and then propagated to an authentication function in a web server binary. This propagation traverses a path that involves irrelevant initialization and request-handling code. Accurately tracking such a noisy path is particularly challenging due to the inherent binary analysis difficulties [80], especially when the credential is propagated indirectly through storage media (e.g., from line 14 to line 4). Without modeling these complex credential data-flow, especially those involving cross-component propagation, existing static data-flow analysis techniques [46, 48, 67, 76] would fail to comprehensively recover credential initialization procedures, consequently leading to false negatives.

C3: Detecting Variable Weak Credentials. To detect a weak credential, the key task is to determine whether an initialized credential is deducible. This task ultimately checks whether the credential is calculated from deducible data sources. For hard-coded or default credentials, the credential data sources are typically constants, which can be easily deduced. However, the deducibility of variable credential data sources, such as MAC addresses in Figure 1, is more difficult to determine but crucial for vulnerability detection. To address this issue, Firmalice [68] treats device output as accessible by attackers and maps them to non-static operands. However, this method does not scale well, as it depends on physical device access and suffers from limited coverage of dynamic testing. In fact, Firmalice analyzes only three embedded devices, highlighting the need for a more scalable and systematic approach.

3.3 Key Ideas

To address the three challenges above, we propose FirmCred, which incorporates three key techniques: semantic scoring-based credential validation site identification (for **C1**), **F**irmware-**C**redential-**A**ccess-**O**peration-based (FCAO-based) credential initialization expression extraction (for **C2**), and inferring-based UoWC vulnerability detection (for **C3**). This section presents the key ideas of each technique, and the technical details will be provided by §4.

T1: Semantic Scoring-based Credential Validation Site Identification. To address Challenge C1, we observe that despite variations across vendors and the lack of documentation or symbols, authentication typically follows a three-phase structure: input parsing, credential validation, and output generation. These phases introduce recurring code patterns that can be matched for identifying credential validation sites. Input parsing extracts user-supplied credentials (e.g., usernames, passwords), often through recognizable patterns [46, 67]. Credential validation involves comparing these inputs with stored values, using byte-level comparisons and cryptographic functions (e.g., SHA-256 from OpenSSL). Output generation produces messages or logs to indicate authentication outcomes. We identify authentication by aggregating the above indicators rather than relying on a single feature, as detailed in §4.1. This multi-feature approach enhances robustness and mitigates false positives and false negatives.

T2: FCAO-based Credential Expression Extraction. To address challenge C2, our idea is to effectively traverse cross-component credential initialization data-flow with minimum consideration of irrelevant firmware code, which increases the success rate of extracting comprehensive firmware credential initialization expressions. For this purpose, the key observation is that credentials are usually transferred between components through a set of firmware credential access operations (FCAOs). FCAOs read or write credentials through specific storage spaces, including global memory region, data files (e.g., configurations), and peripheral devices (e.g., nvram [2] and flash [13]), which is the primary cause of the cross-component data-flow. Therefore, we first infer all FCAOs and then connect them to effectively track how a credential is initialized by one or multiple components. Taking Figure 1 as an example, there are two FCAOs: one reads the password at the credential validation site (line 4) by calling `xget_str`; the other writes the initialized credential at line 14 by executing `xmldbc`. As both FCAOs access the same intermediate storage, which is indexed by */sys/user:1/password*, one can infer line 14 once it has identified line 4, which is achieved by our previous technique.

T3: Inferring-based UoWC Vulnerability Detection. To address Challenge C3, we leverage recent advances in large language models (LLMs) by combining conservative rules with LLM-assisted analysis. For constant expression operands, their deducibility is straightforward by conservative rules. For other operands, we leverage LLM's robust code understanding capabilities. By providing the surrounding code context, the LLM can determine whether the data is easily deducible by an attacker. For example, considering the context around lines 10-14 in `S30setnode.sh`, the LLM can deduce that the credential at */sys/user:1/password* is initialized using a static prefix `"740c"` (line 14) combined with the device's MAC address

(line 10), which is deducible, so the combination of these deducible data yields a predictable credential.

4 Design

Figure 2 shows the overall design of FirmCred. Given a firmware image, FirmCred first unpacks it using Binwalk[5] to extract the firmware file system. Then, FirmCred incorporates a semantic scoring-based approach to identify credential validation sites within the extracted binary programs (§4.1). Starting from these sites, FirmCred backtracks the data-flow to understand the credential initialization procedure, and extracts the corresponding credential initialization expression. This process leverages firmware credential access operations (FCAOs) to efficiently track cross-component data-flow (§4.2). Finally, FirmCred infers whether the extracted expressions are deducible and detects UoWC vulnerabilities based on the inference results (§4.3).

4.1 Credential Validation Site Identification

Authentication procedure fundamentally checks whether the user-provided and device-stored credentials are equal by comparing their bytes, so we define the corresponding byte-comparison function calls (e.g., `memcmp()`) as credential validation sites. To pinpoint credential validation sites, FirmCred primarily focuses on semantic features that characterize the broader authentication procedures, involving three main steps: input parsing, credential validation, and output generation. The identification unfolds in three steps. First, we define a comprehensive set of code features associated with these authentication steps (§4.1.1). Next, FirmCred searches for these features within firmware and then leverages the identified features, in conjunction with data-flow analysis, to locate candidate credential validation sites (§4.1.2). Finally, for each candidate, FirmCred aggregates the evidence from all its corresponding features by calculating a score. This score is then used to filter and confirm the true validation sites (§4.1.3). This scoring-based approach is inherently more reliable than approaches relying on any single feature, as the latter are prone to false positives or negatives arising from feature misidentification, omission, or inaccuracies in data-flow analysis. Note that credential validation sites do not necessarily contain all of these features. These features serve as indicative signals of credential validation, rather than definitive evidence.

Table 1: Semantic features used for credential validation site identification.

Category by Stage		Semantic Feature
Compilation		Unstripped authentication function names
Authentication	Input parsing	Credential keywords
	Credential validation	Credential comparisons Cryptographic operations
	Output generation	Post-validation messages

4.1.1 Semantic Features of Authentication. We first define a set of semantic features that characterize authentication procedures. As described in Table 1, we classify these features into four categories based on their introduction stages: compilation, input parsing, credential validation, and output generation (as shown in Figure 3). We demonstrate each in the following parts.

① **Compilation Features.** Compilation features refer to authentication function names reserved by compilation, including function names unstripped or reserved for dynamic linking [12]. These names (e.g., alpha_auth_check in Figure 1) directly represent authentication semantics, and thus help identify credential validation sites within the corresponding functions.

② **Input Parsing Features.** Input parsing features refer to keywords that are used to parse user-provided credentials. We use these keywords due to two considerations. First, during authentication, input parsing locates user-provided credentials using specific keywords (i.e., credential keywords) before validating them, which thus marks starting points for identifying credential validation sites. Second, such keyword-based input parsing paradigms are prevalent in embedded firmware images and are also used by existing firmware binary analysis techniques [46, 67]. Therefore, we extract credential keywords as features for credential validation site identification.

③ **Credential Validation Features.** This category refers to code features directly related to the authentication procedure. The authentication procedure checks user-provided credentials against a device-stored one using byte-comparison functions (e.g., `strcmp` in Figure 3), sometimes with cryptographic operations beforehand (e.g., `env_SHA256` in Figure 3). Accordingly, we design two features directly related to credential validation: cryptographic (e.g., encryption and hashing) and byte-comparison operations.

④ **Output Generation Features.** After credential validation, authentication typically generates output messages. Specifically, these post-validation messages can serve two purposes. One serves as a response containing the authentication result (e.g., HTTP 401 status code indicating unauthorized access). The other is a diagnostic message like `"Login success!"` shown in Figure 1. Empirically, reverse engineers can directly search these semantic values to locate the authentication procedure effectively. Therefore, we define post-validation messages as a valuable semantic feature for authentication identification.

4.1.2 Candidate Identification. FirmCred starts by searching for semantic features previously defined and then uses the detected features to identify potential credential validation sites as candidates. For an effective search for semantic features, FirmCred limits the searching scope to binary programs, as firmware images typically validate credentials in back-end binary programs instead of front-end scripts [46]. Furthermore, FirmCred leverages the well-established whitelist-based and parser-based technique [67] to filter network-facing programs (e.g., HTTP daemons) and dependency libraries while removing irrelevant programs (e.g., BusyBox). Due to customization authentication implementations, defining concrete semantic features (e.g., what authentication function names are) is challenging. As a countermeasure, we use the large language model (LLM) [20] to identify candidates more comprehensively. Next, we detail how FirmCred searches for features in each category and uses them to identify candidate credential validation sites.

① **Candidate Identification with Compilation Features.** In closed-sourced scenarios, directly identifying the authentication

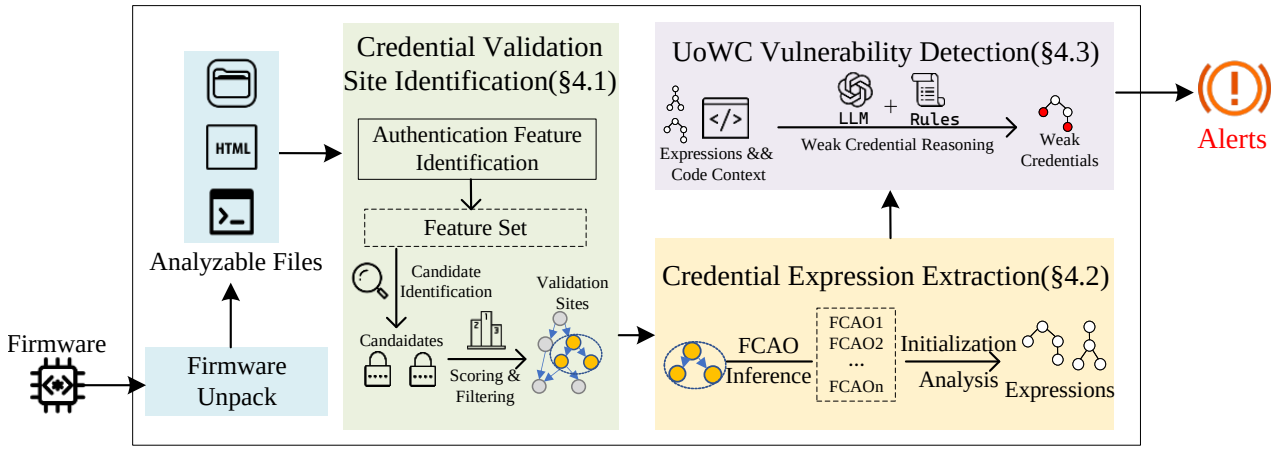


Figure 2: FirmCred overview.

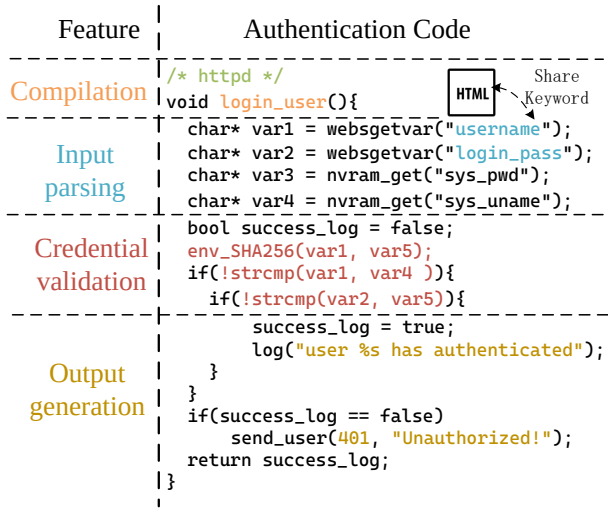


Figure 3: Examples of semantic features used for credential validation site identification.

function based on unstripped function names is still non-trivial due to the lack of documentation. To address this issue, our key observation is that firmware likely reuses code from open-sourced projects [45] (e.g., OpenWrt [27]), and thus derives the authentication function names. Therefore, we manually extract authentication function names from these projects based on their documentation/-code, which is a one-time effort and does not require reverse engineering firmware binaries; then FirmCred matches the extracted names to find authentication APIs. In other situations, FirmCred prompts the LLM to recognize authentication API names, as the LLM is trained on a larger dataset involving more authentication code. For each identified authentication function, FirmCred regards byte comparisons (e.g., `strcmp`) called by this function as candidate credential validation sites.

② **Candidate Identification with Input Parsing Features.** To identify credential parsing logic, FirmCred extracts authentication-related keywords from both standard and customized frameworks. For standard authentication, we manually define common keywords such as `"Basic"`, `"Digest"`, `"username"`, and `"password"`, based on widely adopted protocols [4, 11]. These common keywords will sometimes be used in non-authentication routines, such as using `"username"` for credential configuration instead of validation. To reduce such false positives, we observe that configuration routines typically parse multiple (i.e., more than three) inputs, so we de-prioritize input parsing features in such routines as detailed in Appendix B. For customized authentication [44], FirmCred leverages the large language model (LLM) to analyze unpacked front-end code and extract credential keywords (e.g., username, password) from the request data construction routine by searching predefined authentication keywords (e.g., *"login"*), and then extracts credential keywords. After that, FirmCred applies existing techniques [46] to locate back-end parsing logic. It then performs forward taint analysis from the data received from identified credential keywords to detect tainted byte comparisons (i.e., strcmp[33] and memcmp[22] function families), marking potential credential validation points.

③ **Candidate Identification with Credential Validation Features.** Intuitively, FirmCred can search for byte comparison function calls (e.g., `memcmp`) and treat them as credential validation site candidates. However, this introduces many false positives because not all byte comparisons validate credentials. Actually, FirmCred leverages two features that depict specific authentication code patterns to identify credential validation sites. First, authentication involves both a username and a password and sequentially compares them. This leads to a specific pattern where one byte comparison dominates another in the control-flow graph. Second, a user-provided credential will be transformed (e.g., hashed) using cryptographic operations before being compared. So when FirmCred finds common cryptographic operations before byte comparisons, it identifies the comparison as a candidate. These two features are more discriminative than simply using byte comparisons.

④ **Candidate Identification with Output Generation Features.** FirmCred primarily identifies output generation features, i.e., post-validation messages, by analyzing strings extracted from binary programs. We defer this semantic understanding task to the LLM for judging the relevance of the message to authentication. When

FirmCred finds a relevant message, FirmCred considers the previous candidate site in the control-flow graph more likely to be a real candidate.

4.1.3 Candidate Scoring and Filtering. We adopt a scoring-based approach to identify credential validation sites. Here, two issues may hinder effectiveness. First, individual features are not equally reliable; for example, the LLM sometimes misidentifies output generation features as related to authentication due to hallucination. Second, certain features (e.g., function names) may be absent for some validation sites. To identify credential validation sites more robustly and comprehensively, FirmCred adopts a scoring and filtering approach that aggregates multiple features. As shown by Eq 1, the approach first assigns a distinct weight (w_i) for different feature categories (represented by i); then, it adds the weight to the candidate site score if the corresponding feature exists (i.e., $n_i = 1$); At last, it identifies candidates whose scores exceed an empirically chosen threshold (K) as real credential validation sites.

$$\sum_{i=1}^{T} n_i \cdot w_i > K \tag{1}$$

During the scoring procedure, the feature weights are empirically decided based on how reliable the features are. We consider the compilation and input parsing features more reliable, as they directly represent authentication semantics (e.g., names of authentication functions) and many of them are identified based on standard/well-known implementations. Other features are considered less reliable as they either implicitly represent authentication semantics (e.g., post-validation messages) or easily collide with general features in non-authentication code (e.g., cryptographic operations and nested byte comparisons). Therefore, FirmCred assigns higher weights to compilation and input parsing features, while giving less and equal weights to other features. The specific data of threshold, weight are detailed in Appendix B.

4.2 Credential Expression Extraction

Credential initialization Expression. Credential initialization expression reveals how the credential is initialized from a data-flow perspective, FirmCred leverages credential initialization expression for later credential deducibility inference and UoWC vulnerability detection. As showcased by Figure 4-(b), credential expressions consist of operations and data operands. The operands can be either constants (e.g., *"AM"*) or variables (e.g., *MACAddress*). The operation is represented by an abstract placeholder (denoted by $\odot$) that can refer to any data operation. Since an attacker can always infer the operation by reversing the firmware, the operation doesn't affect whether the expression is deducible.

FirmCred performs backward taint analysis [55, 77] starting from identified credential validation sites to trace how device-stored credentials are initialized, which finally extracts comprehensive credential expressions. The credential expression extraction algorithm is shown in Algorithm 1. As described in §3, the backward data-flow analysis can be interrupted, which is primarily caused by firmware credential access operations (FCAOs) to allow cross-component data propagation. To tackle the problem, we propose a common representation of such FCAOs (§4.2.1), then FirmCred automatically infers such FCAOs from firmware (§4.2.2), and finally leverages them to comprehensively traverse credential initialization procedures and extract the corresponding expressions (§4.2.3).

Algorithm 1 Credential Initialization Expression Extraction

Require: Img (Firmware image), $CVsite$ (Credential Validation Sites)
Ensure: $CIexp$ (Credential Initialization Expression)

```
1:  FCAO_read ← InferReadByTaint(CVsite)
2:  while not all FCAO are processed do
3:      FCAO_write ← InferWrite(FCAO_read, Img)
4:      WriteCon ← InferContent(FCAO_write)
5:      if WriteCon ∉ {Constant} then
6:          FCAO_read' ← InferReadByTaint(FCAO_write)
7:          AddFCAO(FCAO_read')
8:      end if
9:      UpdateExp(CIexp, FCAO_read, FCAO_write)
10: end while
11: return CIexp
```

4.2.1 Firmware Credential Access Operation (FCAO). FCAOs refer to operations that read or write credentials through specific storage spaces. The storage spaces can be global memory regions, data files, or peripheral devices. To access credentials, FCAOs typically require a value that represents the storage index of a credential, such as indexing a credential by its key in nvram [2]—a peripheral that organizes persistent data as a key-value store for efficient access. Note that such storage-index-based FCAOs are prevalent, as shown by our empirical evaluation in §5.4.2.

To depict FCAO-based data-flows, we formalize an FCAO as a three-element tuple (*STORAGE*, *INDEX*, *OPERATION*). *STORAGE* denotes the specific storage space (e.g., nvram); *INDEX* records the access index for a credential; *OPERATION* represents either read or write. Based on this formalization, we can effectively connect cross-component data-flow by identifying both sides (determined by *OPERATION*) that read/write the same credential (identified by the *STORAGE* and *INDEX*). Next, we detail how FirmCred infers FCAOs and uses them for credential initialization analysis.

4.2.2 FCAO inference. FirmCred infers FCAOs with two stages. First, FirmCred identifies FCAOs that read device-stored credentials at the credential validation sites identified in §4.1 through backward taint analysis, as shown in procedure `InferReadByTaint` and line 1 in Algorithm 1. Second, based on known FCAO read operations (i.e., FCAO-reads) FirmCred infers FACOs that write the same credential storage (i.e., FACO-writes) as shown in line 3 in Algorithm 1.

① **FCAO Read Inference.** FirmCred starts by identifying the variable that denotes device-stored credentials at the credential validation sites (line 1 in Algorithm 1). As these sites also compare user-provided credentials, FirmCred should distinguish between both kinds of credentials. To achieve this, FirmCred first recognizes variables that convey user-provided credentials; these credentials are read/parsed from the network inputs, and can be identified with taint analysis starting from these input reading/parsing sites. FirmCred identifies these sites using both input parsing features described in §4.1.2 as well as existing input reading function identification approaches [63] to mitigate false negatives. After identifying

a user-provided credential variable, FirmCred naturally identifies another compared variable as the device-stored credential.

After that, FirmCred determines the *STORAGE* element of FCAO, i.e., where the credential is read, using backward taint analysis. Specifically, FirmCred first marks the device-stored credential variable as tainted and traverses use-def chains to find how the variable is originally defined. This analysis recursively analyzes relevant functions to handle inter-procedural data flows. This backward data-flow analysis stops when an FCAO is met (i.e., sink), which requires an FCAO inference to continue the analysis. Specifically, FirmCred recognizes three types of such FCAO sinks, where each indicates reading a specific FCAO storage space.

- **Reading Data Files.** The first type of sink is standard file reading function calls (e.g., `fopen`, `read`), which indicates that the credential is derived from data files.
- **Reading Global Memory Regions.** The second type of sink is reading a global variable, which indicates that the credential is managed through the surrounding global memory region to reduce I/O operations.
- **Reading Peripheral Devices.** This sink type is represented by device-reading functions such as *nvram_get* and *flash_read*. We focus on high-level wrapper functions rather than low-level operations (e.g., *ioctl()*), as they exhibit greater consistency across firmware. To identify such functions, we observe that device access typically appears as read/write pairs. FirmCred extracts external library functions and detects these pairs through three analyses: (1) constant analysis, which identifies functions invoked with shared constants indicating common device indices; (2) name analysis, which matches functions with similar naming patterns (e.g., *nvram_get*/*nvram_set*); and (3) data flow analysis, which verifies that one function produces data consumed by the other. Specifically, forward taint analysis confirms data-producing (device-reading) functions whose return values propagate to known data consumers (e.g., string or output functions), while backward taint analysis identifies data-consuming (device-writing) functions whose arguments originate from known data sources (e.g., file or device reads).

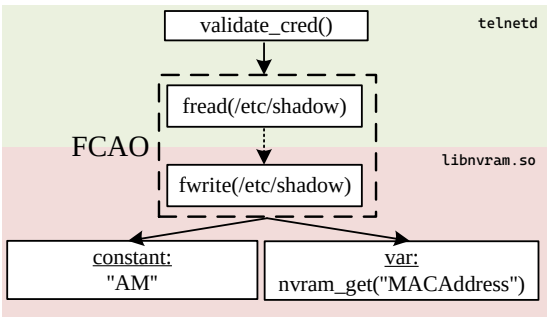
Finally, FirmCred identifies the *INDEX* element of FCAO, which is essential for finding FCAO writes. As different storage spaces demand different access indexes, FirmCred infers their *INDEX* accordingly. For the data file, FirmCred uses the filename extracted from standard file functions (e.g., `fopen`) as the `INDEX`, which is aligned with the previous approach for multi-binary data-flow analysis [67]. For the global memory region, the credential is typically indexed through a structure offset for calculating the credential pointer; therefore, FirmCred uses the offset extracted from data-flow analysis. For the peripheral device, we extract constant function arguments as `INDEX`. This process is straightforward by using high-level APIs. For example, `nvram_get` function calls typically pass a constant `INDEX` as the first argument.

② **FCAO Write Inference.** Given an FCAO read, FirmCred infers corresponding write operations to construct a complete credential data-flow. Since FCAO reads and writes share the same `INDEX` and `STORAGE` with symmetric access patterns, FirmCred first identifies potential FCAOs based on these attributes and then isolates actual write operations. For global-memory FCAOs, FirmCred locates writes to the associated global variables via cross-reference analysis. For file- or device-based FCAOs, where `INDEX` values are typically distinctive constants, FirmCred identifies candidate FCAOs by searching for these constants (e.g., file names or storage indices).

Such an index-based FCAO inference strategy generalizes well, but can be imprecise due to constant collision—the index constant is used for non-writing purposes. To address this issue, we propose two designs that target binary programs and plain text files. For binary programs, FirmCred filters FCAOs on file and device storage that call previously known/identified data-writing functions (e.g., `fwrite`, and `nvram_set`). For plain text files, FirmCred uses the LLM to understand whether the constant actually writes the data or directly holds a data value, which will be described in §4.2.3.

A special case arises when FCAOs write credentials without using `INDEX`. This occurs when the firmware initializes the storage space by sequentially copying data along with its corresponding indexes, without using the index. To capture such FCAO writes, FirmCred identifies initialization procedures based on function name suffixes such as `_init`, `_default`, and `_factory`, and then inspects the copied indexes to detect FCAO writes.

4.2.3 Credential Initialization Analysis and Expression Extraction. FirmCred recursively tracks FCAOs to reconstruct credential initialization procedures, no matter whether it involves cross-component data-flow. For each FCAO read, FirmCred first infers all its corresponding FCAO writes as described before. Then, FirmCred tracks what values are written by these FCAOs using intra-component data-flow analysis (line 6,7 in Algorithm 1). For binary programs, such an analysis uses backward taint analysis from FCAO writes until it reaches constants or FCAO reads. For scripts, FirmCred leverages the LLM to extract the FCAOs exhibit writing semantics and to extract the actual written values. FirmCred repeatedly performs the above steps until no new FCAOs write can be identified. This algorithm reconstructs the data dependency graph of an initialized credential, which corresponds to a credential expression as showcased by Figure 4 (line 9 in Algorithm 1). Based on such expressions, FirmCred further infers weak credentials.



(a) Credential data dependency.

$$cred = \text{“AM”} \odot nvram_get(\text{“MACAddress”})$$

(b) Credential expression.

Figure 4: An example of Credential Initialization Expression

4.3 UoWC Vulnerability Detection

For each extracted credential initialization expression, FirmCred inferring its deducibility by inspecting its data operands. If all data operands are deducible, the credential is deemed weak. FirmCred determines deducibility using both conservative rules and LLM assistance. The conservative rules treat only static data (e.g., strings or configuration files) as deducible, since attackers can deterministically infer them. However, these rules may miss non-static but still predictable data. Such data include device-specific identifiers (e.g., MAC addresses or UUIDs) that, although unique, can be obtained by attackers via alternative methods (e.g., probing for MAC addresses or issuing UPnP M-SEARCH requests to retrieve UUIDs [32]). When credentials are derived from such information, they become predictable and insecure. This reflects a common misconception in embedded device development: that uniqueness equates to secrecy.

To recognize non-static but deducible data operands in credential initialization expressions, FirmCred inspects their surrounding code and data context using the LLM. Specifically, FirmCred selects FCAO reads that do not originate from any writes and extracts their string representation and code context. The string representation directly denotes the meaning of the data, while the code context provides auxiliary information, especially when FCAOs in scripts use unmodeled instructions, showcased by Figure 1.

5 Evaluation

We conducted extensive experiments to evaluate FirmCred's performance in detecting real-world UoWC vulnerabilities (**RQ1**), compare it with SoTA UoWC vulnerability detection approaches (**RQ2**), and measure the effectiveness of its internal designs (**RQ3**).

5.1 General Experiment Setup

Prototype. We implemented a prototype of FirmCred with about 17K lines of Java and 4K lines of Python. The credential validation site identification and the credential initialization analysis are implemented with Ghidra[37] FirmCred scales to all architectures and platforms supported by Ghidra. The source code and running environment is at https://zenodo.org/records/21598603.

Dataset. We evaluated our prototype on Linux-based firmware collected from three sources: FirmAE [60], LFWC [56], and FirmRec [76], aiming for high vendor diversity. As FirmAE mainly includes pre-2019 samples, we incorporated LFWC to include newer firmware and randomly selected 10% of post-2019 samples from both datasets. To further expand vendor coverage, we added samples from FirmRec. The final dataset consists of 377 firmware images from 13 vendors. After unpacking, we identified 1,212 network-facing binaries. These binaries serve as entry points for further analysis, from which FirmCred may potentially explore additional programs. A dataset overview is provided in Table 2.

Environment. All experiments were conducted on a machine equipped with a 64-core Intel Xeon 5218 CPU and 246 GB of RAM, running Ubuntu 22.04.

5.2 Vulnerability Detection Experiments (RQ1)

Experiment Overview. This experiment aims to demonstrate the capability of FirmCred in detecting UoWC vulnerabilities in the real-world image dataset. This experiment used the same configuration and dataset described in §5.1.

Table 2: Overall firmware dataset.

Vendor	# Images	# Target Binaries
Asus	68	384
D-Link	30	58
EDIMAX	1	3
EnGenius	5	14
NETGEAR	80	161
TP-Link	49	179
TRENDnet	33	62
Ubiquiti	66	220
Belkin	6	12
Linksys	14	40
Zyxel	5	11
TOTOLINK	9	32
Tenda	11	36
Total	377	1,212

Results on Vulnerability Detection. In all, FirmCred raised 156 alerts. We manually validated the discovered vulnerabilities, either by constructing Proof-of-Concepts (PoCs) or manual verification due to a lack of devices/robust emulation techniques [58, 60]. Our manual verification follows two criteria: (1) the credential is involved in authentication, which is confirmed by inspecting its usage context; and (2) the affected service is reachable by default or with user configurations (i.e., is not dead code), which is verified by analyzing whether these services are referenced in initialization programs (e.g., */sbin/init*, */etc/rcS*). If either of the two conditions is not met, we consider the alert as an FP.

After alert validation, we confirmed 96 real UoWC vulnerabilities. This indicates a precision of 63.5%, which is quite acceptable compared with existing static firmware vulnerability detection techniques [46, 67, 76]. We further classified the detected vulnerabilities according to their weak credential types, as described in §2. We found 85, 9, and 2 vulnerabilities corresponding to default, hard-coded, and predictable credentials. To sum up, we found default credentials were the most prevalent among weak credentials, while hard-coded and predictable credentials still threatened the security of embedded devices and are essential to detect. FirmCred detected all types of these weak credentials, showing its superior value.

Results on Running Performance. We measured the time required by each phase of FirmCred and the total analysis time. Our prototype analyzed 69% of firmware samples within 2 hours. On average, the credential validation site identification cost 0.36 hours and the credential initialization analysis cost 1.47 hours. Therefore, FirmCred required around 1.83 hours to examine each firmware image on average.

FirmCred's main performance overhead stems from data-flow analysis using the Ghidra decompiler, which incurs extra cost from unused artifacts like type inference. Since Ghidra processes one binary at a time, repeated reloads for binaries involved in multiple FCAOs add an estimated 20.2% to the analysis time. These overheads can be further diminished by improving binary re-loading through engineering efforts.

Table 3: Results on our dataset of firmware samples. For each vendor, we reported the number of firmware images and detected UoWC vulnerabilities through FirmCred and three baselines. For each tool, we calculated the precision and vulnerability detection ratio. For FirmCred and Firmalice, we calculated the time.

Vendor	# Img	Precision								Vuln detection ratio (Recall on current dataset)†					Time[hh:mm]		
		Firmcred		Firmalice		Scanning		Testing*									
		# Alarms	Prec	# Alarms	Prec	# Alarms	Prec	# Alarms	Prec	# Ground truth	Firmcred	Firmalice	Scanning	Testing	# CVSite‡	Firmcred	Firmalice
Asus	68	18	22.2%	0	-	4	75.0%	3	100.0%	4	100.0%	0.0%	75.0%	75.0%	817	439:26	220:18
D-Link	30	17	64.7%	0	-	6	100.0%	3	100.0%	11	100.0%	0.0%	54.5%	27.3%	99	137:48	59:32
EDIMAX	1	1	100.0%	0	-	1	100.0%	1	100.0%	1	100.0%	0.0%	100.0%	100.0%	2	02:23	01:51
EnGenius	5	1	100.0%	0	-	1	100.0%	0	-	1	100.0%	0.0%	100.0%	0.0%	3	01:11	00:41
NETGEAR	80	37	48.6%	3	100.0%	13	100.0%	5	100.0%	18	100.0%	16.7%	72.2%	27.8%	880	320:56	495:29
TP-Link	49	20	85.0%	0	-	6	83.3%	0	-	17	100.0%	0.0%	23.5%	0.0%	139	118:50	145:36
TRENDnet	33	12	83.3%	1	100.0%	7	71.4%	0	-	7	100.0%	28.6%	57.1%	0.0%	106	60:54	266:32
Ubiquiti	66	8	100.0%	0	-	0	-	0	-	8	100.0%	0.0%	0.0%	0.0%	74	92:24	10:58
Belkin	6	4	50.0%	2	100.0%	2	0.0%	0	-	2	100.0%	100.0%	0.0%	0.0%	12	08:27	85:36
Linksys	14	10	60.0%	3	100.0%	0	-	5	100.0%	6	100.0%	33.3%	0%	83.3%	136	93:17	150:08
Zyxel	5	4	75.0%	0	-	0	-	3	100.0%	3	100.0%	0.0%	0.0%	100.0%	16	08:00	45:09
TOTOLINK	9	8	100.0%	0	-	1	100.0%	3	100.0%	8	100.0%	12.5%	37.5%	37.5%	10	23:56	02:17
Tenda	11	16	62.5%	0	-	8	25.0%	8	100.0%	10	100.0%	0.0%	20.0%	80.0%	87	54:47	34:57
Summary	377	156	61.5%	9	100.0%	49	71.4%	30	100.0%	96	100.0%	9.4%	36.5%	31.3%	2381	1349:21	1519:04

† Vulnerability detection ratio is the detection recall based on our sampled dataset.
‡ CVSite: Number of Credential Validation Sites identified by FirmCred.
* For testing-based methods, we manually verify whether the UoWC vulnerabilities identified by FirmCred involve common weak credentials. Precision is inherently 100%.

False Negative Evaluation on N-day Dataset. To conduct a false negative evaluation of FirmCred, we conducted an experiment assessing FirmCred's ability to reproduce N-day vulnerabilities. We examined all CVE identifiers issued after 2020 related to UoWC vulnerabilities with clear descriptions and extractable firmware. We have also conducted a thorough search for historical vulnerabilities in the selected 377 firmware images in our dataset. In the end, 18 CVEs provided sufficiently detailed descriptions and extractable firmware for analysis. We evaluated FirmCred, Firmalice [68], Scanning-based method [49], and testing-based method [61] on this dataset. The results show that FirmCred can reproduce 17 out of 18 of these vulnerabilities, achieving a high recall rate of 94.4%. Whereas Firmalice, the scanning-based method, and the testing-based method achieve 11.1%, 50%, and 33.3%, respectively. The missing case is due to the failure of cross-reference analysis on global data, which in turn results from decompilation errors. Detailed statistics and analysis of this experiment can be found in the full version of our paper in [41].

False Positive Analysis. FirmCred reported a total of 60 false positives. After a comprehensive manual examination, we summarized four primary factors. First, we conservatively regarded 23 alerts as corresponding to dead code, as we failed to identify any initialization procedure that would start the corresponding services. Second, 21 false positives are caused by inaccuracies in credential validation site identification. We will explain the detailed reasons in §5.4. Third, 13 false positives result from inaccurate FCAO inference, which extracted wrong credential initialization expressions from infeasible data-flows (e.g., overtaint). Finally, 3 false positives were attributed to LLM hallucinations, i.e., the LLM mistakenly reported non-deducible data operands in credential initialization as deducible. For example, the LLM considered a data operand denoted by *"SYSPS"* to be deducible, whereas it actually refers to the system password and is non-deducible. Despite introducing false positives, LLM successfully identified two real-world predictable credentials, demonstrating that the incorporation of LLMs is effective.

5.3 Comparison Experiments (RQ2)

Experiment Overview. This experiment compares FirmCred with the SoTA approaches in UoWC detection with three metrics, i.e., precision, recall (depicted by vulnerability detection ratio), and running performance. To evaluate recall, we constructed a ground truth by summing up vulnerabilities detected by all tools, which aligns with existing practices [76]. We adopted the same experimental setup as in §5.1 for this experiment. For each alert, we manually validated it following the same criteria in §5.2. Then, we calculated the precision and vulnerability detection rate for each method. For FirmCred and Firmalice, we compared their running time.

Baselines. We selected three baselines representing SoTA approaches: Firmalice [68] representing solving-based methods, [49] representing scanning-based methods, and [61] representing testing-based methods. As the baseline tools were generally unavailable, we carefully re-implemented each based on original descriptions, following best practice with our best efforts. For Firmalice, we re-implemented it based on the authors' suggestions with the advanced symbolic execution engine angr [3]. The details are presented in [41]. The scanning-based method [49] follows a straightforward approach: it extracts files commonly used by PAM authentication (i.e., */etc/shadow* and */etc/passwd*) and identifies default credentials (i.e., hashed passwords) from them. We adopted the same approach in our implementation and treated all discovered hashed passwords as weak credentials, because an attacker only needs to know one password (by buying a device or brute-forcing) to compromise all affected devices. For the testing-based approach, its original paper leveraged only 24 known weak credentials for testing; we further extended this list with John the Ripper's weak password list [19], which involves 3546 common weak passwords, to enhance its capability. Other related studies, such as those on cryptographic API misuse [62, 74] and weak credential detection [64, 72, 81] on other platforms, are excluded from our comparison, as they either fall outside the scope of this research or do not apply to firmware analysis.

Results on Recall (Vulnerability Detection Rate). The overall results of the experiment can be found in Table 3. Among all tools, FirmCred achieved the best vulnerability detection rate of 100%, while Firmalice, scanning-based approach, and testing-based approach were only 9.4%, 36.5%, and 31.3%, respectively. In addition, FirmCred successfully detected all vulnerabilities found by the other baselines, and detected 24 vulnerabilities missed by all of them (two are showcased in the case study in full version of the paper, see [41]). This result demonstrates the superior vulnerability detection ability of FirmCred over existing approaches, which scales to a wider range of UoWC vulnerabilities.

Then we examined why the three baselines fell short. We found Firmalice was limited by its vulnerability detection scope and the inherent limitation of symbolic solving. Specifically, Firmalice overlooked 49 vulnerabilities that initialized weak credentials outside request handling code. Besides, its symbolic execution solvers failed to solve credentials based on their hash results, leading to 37 false negatives. The scanning-based method only detected default credentials in narrowed scanning targets, resulting in a low vulnerability detection rate even for default credentials. Specifically, it overlooked 50 default credentials stored in dynamically initialized or unknown storage media. The testing-based method detected 30 (31.3%) vulnerabilities using a well-curated weak credential list, but naturally overlooked all unlisted weak credentials.

Results on Precision. The precision of the testing-based method is inherently 100% with a well-curated list of known weak credentials. Firmalice also achieves 100% precision by providing concrete solutions. The scanning-based method yielded a precision of 71.4%. The false positives were caused by dead code, which is also the primary factor that caused false positives in FirmCred. FirmCred's precision is 61.5%, its false positives were analyzed in §5.2.

Results on Running Performance. We compared the time overhead of FirmCred with Firmalice, as scanning-based and testing-based approaches involve little analysis overhead. As shown in Table 3, FirmCred is on average 11.2% faster than Firmalice, mainly for two reasons. First, FirmCred adopts lightweight taint analysis, whereas Firmalice relies on heavyweight symbolic execution, incurring significant overhead from state forking and constraint solving. Second, Firmalice attempts to recover concrete weak credentials, which is ineffective for non-plaintext representations such as hashes. Notably, this speedup underestimates FirmCred 's analysis efficiency, since FirmCred additionally analyzes credential initialization code ignored by Firmalice, which is essential for 52.1% UoWC vulnerabilities (see §5.4.2). On average, FirmCred analyzes 4.77 binaries, while Firmalice analyzes only the target binary.

5.4 Internal Design Experiments (RQ3)

This section conducts three experiments to demonstrate the effectiveness of FirmCred's internal designs. It answers three questions: (1) how effective is the semantic scoring-based credential validation site identification (§5.4.1); (2) how effective is the FCAO-based credential initialization expression extraction (§5.4.2); (3) how does the use of LLM affect overall effectiveness (§5.4.3). To reduce manual effort due to the labor-intensive nature of result analysis while preserving representativeness, we randomly sampled 10% of the dataset from the previous experiment.

5.4.1 Effectiveness of Semantic Scoring-based Credential Validation Site Identification. This section demonstrates the accuracy of the credential validation site identification. For accuracy measurement, we constructed a ground-truth dataset by reverse-engineering binaries in the sampled dataset. Note that the firmware samples used here are distinct from the threshold dataset (in Appendix B), ensuring a fair evaluation. For this purpose, we invited two security researchers with 3 and 5 years of experience, and asked them to manually identify credential validation sites following 2 criteria: (1) the credential validation sites read user-provided credentials, and (2) the sites validate user-provided credentials with device-stored ones. Finally, the two experts spent around 40 hours to identify 240 credential validation sites as a ground truth. The high labor hours also highlight the need for an automatic credential validation site identification approach.

We have also conducted an experiment to measure the contributions of individual semantic features in authentication identification. The result can be found in the full version of our paper [41].

Overall Effectiveness. We checked credential validation sites identified by FirmCred against the ground truth. The result shows that FirmCred identified 263 credential validation sites among 92 binaries. We found 226 of them to be true positives, representing a precision of 85.9%. FirmCred missed 14 credential validation sites in the ground truth, resulting in a recall of 94.2%.

Upon further investigation of these false positives, we found three primary causes. First, non-credential validation sites may resemble actual authentication logic, such as input parsing involving credential-related keywords outside of authentication, leading to false positives. Second, FirmCred may hallucinate by misattributing features like MAC address or HTTP referrer checks as authentication routines, as these operations share semantic similarities with validation but do not verify credentials. Additionally, over-tainting may introduce false positives when variables unrelated to user credentials are incorrectly marked as tainted, often due to imprecise variable alias analysis or misidentification of stack variable boundaries during decompilation.

False negatives arise from two primary sources. First, they result from inherent limitations of binary static analysis, such as under-tainting due to inaccurate function argument identification, imprecise binary lifting by the decompiler, and limited support for indirect calls. These factors can hinder complete taint propagation, leading to missed authentication-related features. Second, the scoring mechanism relies on empirically chosen weights and thresholds, which may occasionally underestimate critical features.

5.4.2 Effectiveness of FCAO-based Credential Initialization Expression Extraction. This section evaluates the effectiveness of FCAO inference by analyzing its precision and understanding its contribution to UoWC vulnerability detection. First, to evaluate FirmCred's precision in FCAO inference, we manually verified the correctness of all inferred FCAO instances. Among 242 instances, we confirmed 207 real ones, yielding a precision of 85.5%. We identified three primary reasons that caused the false positives regarding inaccurate backward taint analysis. First, FirmCred misidentified functions that read/write non-credential data as credential access operations due to misidentification of device access functions. Second, some

inaccuracy in alias analysis leads to over-tainting; third, the LLM reported inaccurate script analysis results due to hallucination.

Then, we demonstrate how FCAOs contribute to UoWC vulnerability detection by checking the presence of FCAOs in all 96 UoWC vulnerabilities detected by FirmCred. We found that all vulnerabilities used at least one FCAO, and thus, FirmCred would likely fail to extract comprehensive expressions for weak credentials without FCAO-based data-flow analysis. Besides, 23.9% vulnerabilities used more than one FCAO, so they require the recursive FCAO-based tracking method described in §4.2.3. Furthermore, we found 52.1% of UoWC vulnerabilities initialize weak credentials outside of authentication code defined by Firmalice [68]. Therefore, existing approaches that only consider authentication code (e.g., Firmalice) would fail to detect these vulnerabilities.

5.4.3 Effectiveness of Using LLM. This section demonstrates the effectiveness of using LLM in FirmCred.

Experiment Setup. To evaluate the impact of LLM integration, we implement a variant of FirmCred, named FirmCred-nollm, with LLM functionality disabled. Specifically, during credential validation site identification, FirmCred-nollm identifies compilation features only using authentication function names from open-source repositories, identifies input parsing features only using keywords collected from standard protocols and well-known credential keywords (e.g., *"password"*), and does not use output generation features for identification; during credential expression extraction, FirmCred-nollm disables LLM-based script analysis for FCAO inference; during UoWC vulnerability detection, FirmCred-nollm does not use LLM to infer the deducibility of variables, and only uses conservative rules. Then, we compare FirmCred-nollm with FirmCred in three areas: (1) credential validation site identification, (2) FCAO inference, and (3) overall vulnerability detection ability. To avoid excessive manual efforts, (1) and (2) were evaluated on 38 randomly selected firmware images to control manual effort; (3) was evaluated on the dataset described in §5.1.

Experiment Results. Table 4 presents the experiment results. In credential validation site identification, FirmCred-nollm missed 23 (10.7%) sites compared to FirmCred. FirmCred-nollm missed 9 (4.4%) FCAO entries in FCAO inference. In UoWC vulnerability detection, FirmCred-nollm identified 8 (12.3%) fewer vulnerabilities than FirmCred. Five cases result from missing FCAOs in scripts, two predictable credentials are excluded by the conservative UoWC rules, and one is missed due to the absence of credential validation sites. Finally, the overall LLM tokens consumed to process the whole dataset in §5.1 is about 11.5M. Three stages consume approximately 53.1%, 29.5%, and 17.4% of the token, respectively. These results demonstrate that the use of LLM is practical and will improve the overall effectiveness of FirmCred in detecting UoWC vulnerabilities, but it is optional.

Table 4: Comparison result between FirmCred and FirmCred-nollm.

Tool	Cred Validation		FCAO		UoWC	
	Report	TP	Report	TP	Report	TP
FirmCred	263	226	242	207	156	96
FirmCred-nollm	230	203	224	198	142	88

6 Discussion

Use of LLM. The use of LLM is optional in FirmCred, as evaluated in §5.4.3. There are also many alternatives to LLM in FirmCred. For example, credential keywords can be extracted through front-end file analysis[46] with detailed rules, and natural language semantic classification can be achieved through a training-based method[21], because machine learning can classify between multiple targets[25]. Different LLMs may also have different capabilities [30]. Advanced LLM prompting techniques, such as few-shot learning, were not evaluated in this study. We leave them for future work.

Limitations of Static Analysis. FirmCred leverages static analysis to detect UoWC vulnerabilities. However, static analysis is inherently limited and may introduce both false positives (e.g., over-tainting) and false negatives (e.g., under-tainting, missing dynamic information, and incomplete feature coverage) [48, 66]. False positives can be mitigated through dynamic validation techniques, such as verifying service reachability via firmware initialization emulation [60, 75], or by adopting more precise taint analysis approaches [48, 53, 78]. False negatives can be alleviated by incorporating dynamic analysis or designing more comprehensive and sound feature sets for static analysis. Although our method achieved a 100% recall rate in the evaluation, this does not guarantee complete detection of all UoWC vulnerabilities.

Firmware Encryption. FirmCred is also unable to identify weak credentials in encrypted firmware or in firmware files that are decrypted only during the booting process. As prior works on firmware static analysis[46, 55, 67] typically assume non-encrypted firmware as a prerequisite, we consider this limitation acceptable. We leave addressing these challenges to future work.

Mitigations for UoWC vulnerabilities. To mitigate Use of Weak Credential (UoWC) vulnerabilities, IoT developers can adopt three general strategies. First, from the credential perspective, developers should audit their code to eliminate hard-coded or default credentials in production environments. Second, from the service perspective, unnecessary services should be disabled, and devices should enforce secure credential updates during factory reset or initial setup. Finally, for IoT consumers, promptly changing default credentials remains the most effective defense against UoWC attacks. All the vulnerabilities we discovered can be diminished if the above three strategies are applied.

Future Work. FirmCred's methodology is potentially extensible to the detection of other forms of weakly initialized identities, such as weak cryptographic keys. Addressing these flaws may require novel designs for candidate localization (e.g., new feature representations) as well as new identification criteria (e.g., complexity). These challenges represent important yet still largely unresolved research problems in the firmware scope.

7 Related work

Weak Credential Detection in Firmware. Weak credential detection has been extensively studied across multiple domains, including embedded firmware, mobile platforms, and web applications. In the firmware domain, Firmalice [68] pioneered automated detection of authentication bypass vulnerabilities in binary firmware through

symbolic execution. Costin et al. [49] demonstrated widespread deployment of hard-coded credentials and shared cryptographic artifacts in consumer firmware, highlighting systemic weaknesses in embedded authentication design. [61] used weak credential dictionaries to dynamically detect weak credentials in firmware. However, the above methods can detect only a limited subset of UoWC vulnerabilities because they have ignored the process of credential validation and initialization.

Weak Credential Detection in Other Platforms. Several studies have highlighted credential-related security concerns. Rahman et al. [64] and Zhou et al. [81] examined credential leakage and safety problems in Android applications. In the iOS ecosystem, Zhang et al. [72] conducted an empirical study on SDK credential misuse, uncovering that many iOS applications may expose sensitive credentials due to improper SDK integrations. [43, 47, 50, 57, 65, 69] are dedicated to protecting credentials from being leaked, highlighting the importance of credentials on many platforms.

Firmware Static Analysis. Firmware static analysis has been widely employed in vulnerability detection. Approaches like SaTC[46], FITS[63] aim at shortening the data-flow distance of taint analysis to enhance efficiency and improve accuracy. Subsequent approaches, like OctopusTaint[66], EmTaint[48], HermeScan[54], and Lara [79] propose several techniques to improve static analysis accuracy and enhance efficiency. Other works like Karonte[67], MangoDFA[55], and NVWA[80] detect vulnerabilities through larger firmware-centric perspectives.

8 Conclusion

This paper introduced FirmCred, a static authentication bypass vulnerability detection framework that detects the use of weak credentials from an initialization perspective. FirmCred employs a three-step methodology: it first automatically identifies credential validation sites through semantic scoring, and then traces credential initialization procedures across firmware components using a novel firmware credential access operation (FCAO)-based approach, and finally infers deducibility of the initialized credential to detect weak credentials. Evaluated on 377 real-world firmware images, FirmCred demonstrated notable efficacy and scalability, successfully uncovering 92 previously unknown UoWC vulnerabilities and improving the recall rate over state-of-the-art approaches by 63.5%, thus contributing significantly to the security of embedded devices.

9 Ethical Considerations

This work studies the Use of Weak Credential (UoWC) vulnerabilities in real-world IoT devices. Given the potential security impact of vulnerability research, we conducted a stakeholder-based ethics analysis.

Stakeholders. The primary stakeholders include device vendors, end users of affected systems, the broader Internet ecosystem, and the security research community.

Potential Harms. The detected vulnerabilities may be exploited by attackers if they were not carefully disclosed. However, we have minimized such risks through mitigation below. Improper disclosure of vulnerabilities may enable exploitation, resulting in financial loss, service disruption, or reputational harm to vendors. Our study does not involve human subjects, personal data, or interaction with live systems, and therefore does not raise concerns related to privacy or user data misuse.

Mitigation. We have responsibly disclosed all confirmed zero-day vulnerabilities (see the full version of the paper [41]). We hope these disclosures will raise awareness of the potential risks and encourage vendors to release security patches or advisories to protect affected users.

10 Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. We also extend our gratitude to Fudan University for providing the research facilities and platform. We have used generative AI only in minor editorial improvements of the paper. The paper was edited for grammar and spelling using ChatGPT and Grammarly.

References

[1] 2020. CVE-2020-10988. https://nvd.nist.gov/vuln/detail/CVE-2020-10988.
[2] 2020. Non-volatile Random-access Memory. https://en.wikipedia.org/wiki/Non-volatile_random-access_memory.
[3] 2024. angr: A powerful and user-friendly binary analysis platform. https://github.com/angr/angr.
[4] 2024. Basic Authentication. https://en.wikipedia.org/wiki/Basic_access_authentication.
[5] 2024. Binwalk: Firmware Analysis Tool. https://github.com/ReFirmLabs/binwalk.
[6] 2024. Common Vulnerability Scoring System. https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System.
[7] 2024. CVE-2023-27512, SolarView hard-coded credential vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2023-27512.
[8] 2024. CVE-2024-3272 D-Link NAS hard-coded credential vulnerability. https://nvd.nist.gov/vuln/detail/cve-2024-3272.
[9] 2024. CVE-2024-48271. https://nvd.nist.gov/vuln/detail/CVE-2024-48271.
[10] 2024. CVE-2024-53937. https://nvd.nist.gov/vuln/detail/CVE-2024-53937.
[11] 2024. Digest Access Authentication. https://en.wikipedia.org/wiki/Digest_access_authentication.
[12] 2024. Dynamic Linker. https://en.wikipedia.org/wiki/Dynamic_linker.
[13] 2024. Flash Memory. https://en.wikipedia.org/wiki/Flash_memory.
[14] 2024. Form-Based Authentication. https://docs.oracle.com/cd/E15217_01/doc.1014/e12488/v2form.htm.
[15] 2024. Insecure Authentication Methods and Default Credentials. https://owasp.org/www-project-top-10-infrastructure-security-risks/docs/2024/ISR07_2024-Insecure_Authentication_Methods_and_Default_Credentials.
[16] 2024. Internet of Things Market Size. https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307.
[17] 2024. Introduction to the PAM Framework. https://docs.oracle.com/cd/E26502_01/html/E29016/pam-01.html.
[18] 2024. Iot Device Top Vulnerabilities. https://www.fortinet.com/resources/cyberglossary/iot-device-vulnerabilities.
[19] 2024. John the Ripper Password Cracker. https://www.openwall.com/john/.
[20] 2024. Large Language Model. https://en.wikipedia.org/wiki/Large_language_model.
[21] 2024. Machine Learning. https://en.wikipedia.org/wiki/Machine_learning.
[22] 2024. memcmp function description. https://cplusplus.com/reference/cstring/memcmp/.
[23] 2024. MIRAI DDoS Attacks On Dyn. https://en.wikipedia.org/wiki/DDoS_attacks_on_Dyn.
[24] 2024. Misconfigured IOT Devices. https://cybelangel.com/misconfigured-iot/.
[25] 2024. Multiclass Classification. https://en.wikipedia.org/wiki/Multiclass_classification.
[26] 2024. National Vulnerability Database. https://nvd.nist.gov/.
[27] 2024. Openwrt: Linux operating system targeting embedded devices. https://openwrt.org/.
[28] 2024. OWASP Internet of Things Project. https://www.wattlecorp.com/owasp-iot-top-10/.
[29] 2024. Protecting SmartPLC Devices from Critical Hardcoded Credential Vulnerability CVE-2024-28747. https://nvd.nist.gov/vuln/detail/CVE-2024-28747.
[30] 2024. QianWen LLM. https://www.alibabacloud.com/help/en/model-studio/what-is-qwen-llm.
[31] 2024. Satisfiability Modulo Theories. https://en.wikipedia.org/wiki/Satisfiability_modulo_theories.

[32] 2024. Simple Service Discovery Protocol. https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol.

[33] 2024. strcmp function description. https://cplusplus.com/reference/cstring/strcmp/.

[34] 2024. Symbolic Execution Wikipedia. https://en.wikipedia.org/wiki/Symbolic_execution.

[35] 2024. The True Cost of a DDoS Attack. https://www.corero.com/true-cost-of-a-ddos-attack.

[36] 2024. Use of Weak Credentials Vulnerabilities. https://cwe.mitre.org/data/definitions/1391.html.

[37] 2025. Ghidra Software Reverse Engineering Framework. https://github.com/NationalSecurityAgency/ghidra.

[38] 2025. Path Explosion Wikipedia. https://en.wikipedia.org/wiki/Path_explosion.

[39] 2025. Router reality check: 86% of default passwords have never been changed. https://www.ibm.com/think/insights/router-reality-check-86-percent-default-passwords-have-never-been-changed.

[40] 2025. SinoTrack GPS device flaws allow remote vehicle control and location tracking. https://thehackernews.com/2025/06/sinotrack-gps-devices-vulnerable-to.html.

[41] 2026. FirmCred-full-version. https://yuanxzhang.github.io/paper/firmcred-ccs26-full.pdf.

[42] Zafeer Ahmed, Ibrahim Nadir, Haroon Mahmood, Ali Hammad Akbar, and Ghalib Asadullah Shah. 2020. Identifying Mirai-Exploitable Vulnerabilities in IoT Firmware Through Static Analysis. In *2020 International Conference on Cyber Warfare and Security (ICCWS)*. 1–5. doi:10.1109/ICCWS48432.2020.9292382

[43] Sébastien Canard and Roch Lescuyer. 2013. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 381–392.

[44] Javier Carrillo-Mondéjar, Hannu Turtiainen, Andrei Costin, José Luis Martínez, and Guillermo Suarez-Tangil. 2023. HALE-IoT: Hardening Legacy Internet of Things Devices by Retrofitting Defensive Firmware Modifications and Implants. *IEEE Internet of Things Journal* 10 (2023), 8371–8394. https://api.semanticscholar.org/CorpusID:253915589

[45] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards automated dynamic analysis for linux-based embedded firmware.. In *NDSS*, Vol. 1. 1–1.

[46] Libo Chen, Yanhao Wang, Jiaqi Linghu, Qinsheng Hou, Quanpu Cai, Shanqing Guo, and Zhi Xue. 2024. SaTC: Shared-Keyword Aware Taint Checking for Detecting Bugs in Embedded Systems. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 2421–2433. doi:10.1109/TDSC.2023.3307430

[47] Quan Chen, Ahmed M Azab, Guruprasad Ganesh, and Peng Ning. 2017. Privwatcher: Non-bypassable monitoring and protection of process credentials from memory corruption attacks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 167–178.

[48] Kai Cheng, Yaowen Zheng, Tao Liu, Le Guan, Peng Liu, Hong Li, Hongsong Zhu, Kejiang Ye, and Limin Sun. 2023. Detecting vulnerabilities in Linux-based embedded firmware with SSE-based on-demand alias analysis. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 360–372.

[49] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A large-scale analysis of the security of embedded firmwares *(SEC'14)*. USENIX Association, USA, 95–110.

[50] David Del Vecchio, Marty Humphrey, Jim Basney, and Nataraj Nagaratnam. 2005. Credex: User-centric credential management for grid and web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 149–156.

[51] Steven HH Ding, Benjamin CM Fung, and Philippe Charland. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 472–489.

[52] Runhan Feng, Ziyang Yan, Shiyan Peng, and Yuanyuan Zhang. 2022. Automated detection of password leakage from public GitHub repositories. In *Proceedings of the 44th International Conference on Software Engineering*. 175–186.

[53] LIAN GAO and HENG YIN. 2025. BinDSA: Efficient, Precise Binary-Level Pointer Analysis with Context-Sensitive Heap Reconstruction. (2025).

[54] Zicong Gao, Chao Zhang, Hangtian Liu, Wenhou Sun, Zhizhuo Tang, Liehui Jiang, Jianjun Chen, and Yong Xie. 2024. Faster and Better: Detecting Vulnerabilities in Linux-Based IoT Firmware with Optimized Reaching Definition Analysis. In *Proceedings of the 2024 Network and Distributed System Security Symposium, San Diego, CA, USA*, Vol. 26.

[55] Wil Gibbs, Arvind S Raj, Jayakrishna Menon Vadayath, Hui Jun Tay, Justin Miller, Akshay Ajayan, Zion Leonahenahe Basque, Audrey Dutcher, Fangzhou Dong, Xavier Maso, et al. 2024. Operation Mango: Scalable Discovery of {Taint-Style} Vulnerabilities in Binary Firmware Services. In *33rd USENIX Security Symposium (USENIX Security 24)*. 7123–7139.

[56] René Helmke, Elmar Padilla, and Nils Aschenbruck. 2025. Mens Sana In Corpore Sano: Sound Firmware Corpora for Vulnerability Research. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'25)*. The Internet Society, San Diego, California, USA. https://www.ndss-symposium.org/wp-content/uploads/2025-669-paper-1.pdf

[57] Man Hong Hue, Joyanta Debnath, Kin Man Leung, Li Li, Mohsen Minaei, M Hammad Mazhar, Kailiang Xian, Endadul Hoque, Omar Chowdhury, and Sze Yiu Chau. 2021. All your credentials are belong to us: On insecure wpa2-enterprise configurations. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1100–1117.

[58] Gianluca Stringhini Ioannis Angelakopoulos and Manuel Egele. 2024. Pandawan: Quantifying Progress in Linux-based Firmware Rehosting. In *USENIX Security Symposium*. USENIX Association.

[59] Ling Jiang, Junwen An, Huihui Huang, Qiyi Tang, Sen Nie, Shi Wu, and Yuqun Zhang. 2024. Binaryai: binary software composition analysis via intelligent binary source code matching. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[60] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. 2020. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In *Proceedings of the 36th Annual Computer Security Applications Conference* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 733–745. doi:10.1145/3427228.3427294

[61] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All Things Considered: An Analysis of IoT Devices on Home Networks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1169–1185. https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak

[62] Kaizheng Liu, Ming Yang, Zhen Ling, Yuan Zhang, Chongqing Lei, Lan Luo, and Xinwen Fu. 2024. SAMBA: Detecting SSL/TLS API misuses in IoT binary applications. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2029–2038.

[63] Puzhuo Liu, Yaowen Zheng, Chengnian Sun, Chuan Qin, Dongliang Fang, Mingdong Liu, and Limin Sun. 2023. Fits: Inferring intermediate taint sources for effective vulnerability analysis of iot device firmware. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 138–152.

[64] Olha Mykhaylova, Taras Fedynyshyn, and Artem Platonenko. 2024. Hardcoded credentials in Android apps: Service exposure and category-based vulnerability analysis. *Cybersecurity Providing in Information and Telecommunication Systems II 2024* 3826 (2024), 206–211.

[65] Bijeeta Pal, Mazharul Islam, Marina Sanusi Bohuk, Nick Sullivan, Luke Valenta, Tara Whalen, Christopher Wood, Thomas Ristenpart, and Rahul Chatterjee. 2022. Might I get pwned: A second generation compromised credential checking service. In *31st USENIX Security Symposium (USENIX Security 22)*. 1831–1848.

[66] Abdullah Qasem, Mourad Debbabi, and Andrei Soeanu. 2024. OctopusTaint: Advanced Data Flow Analysis for Detecting Taint-Based Vulnerabilities in IoT/IIoT Firmware. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2355–2369.

[67] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2020. Karonte: Detecting insecure multi-binary interactions in embedded firmware. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1544–1561.

[68] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware.. In *NDSS*, Vol. 1. 1–1.

[69] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. 2019. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*. 1556–1571.

[70] Sam L Thomas, Tom Chothia, and Flavio D Garcia. 2017. Stringer: Measuring the importance of static data comparisons to detect backdoors and undocumented functionality. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 513–531.

[71] Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang. 2022. Jtrans: Jump-aware transformer for binary code similarity detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1–13.

[72] Haohuang Wen, Juanru Li, Yuanyuan Zhang, and Dawu Gu. 2018. An Empirical Study of SDK Credential Misuse in iOS Apps. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 258–267. doi:10.1109/APSEC.2018.00040

[73] Yuhao Wu, Jinwen Wang, Yujie Wang, Shixuan Zhai, Zihan Li, Yi He, Kun Sun, Qi Li, and Ning Zhang. 2024. Your firmware has arrived: A study of firmware update vulnerabilities. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5627–5644.

[74] Yifan Xia, Zichen Xie, Peiyu Liu, Kangjie Lu, Yan Liu, Wenhai Wang, and Shouling Ji. 2025. Beyond Static Pattern Matching? Rethinking Automatic Cryptographic API Misuse Detection in the Era of LLMs. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA006 (June 2025), 24 pages. doi:10.1145/3728875

[75] Haoyu Xiao, Ziqi Wei, Jiarun Dai, Bowen Li, Yuan Zhang, and Min Yang. 2025. HouseFuzz: Service-Aware Grey-Box Fuzzing for Vulnerability Detection in

Linux-Based Firmware . In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 3801–3819. doi:10.1109/SP61157.2025.00213

[76] Haoyu Xiao, Yuan Zhang, Minghang Shen, Chaoyang Lin, Can Zhang, Shengli Liu, and Min Yang. 2024. Accurate and Efficient Recurring Vulnerability Detection for IoT Firmware. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 3317–3331. doi:10.1145/3658644.3670275

[77] Xuexiong Yan, Hengtai Ma, and Qingxian Wang. 2017. A static backward taint data analysis method for detecting web application vulnerabilities. In *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 1138–1141.

[78] Zhuo Zhang, Wei You, Guanhong Tao, Guannan Wei, Yonghwi Kwon, and Xiangyu Zhang. 2019. BDA: practical dependence analysis for binary executables by unbiased whole-program path sampling and per-path abstract interpretation. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–31.

[79] Jiaxu Zhao, Yuekang Li, Yanyan Zou, Zhaohui Liang, Yang Xiao, Yeting Li, Bingwei Peng, Nanyu Zhong, Xinyi Wang, Wei Wang, et al. 2024. Leveraging semantic relations in code and data to enhance taint analysis of embedded systems. In *33rd USENIX Security Symposium (USENIX Security 24)*. 7067–7084.

[80] Jiaxu Zhao, Yuekang Li, Yanyan Zou, Yang Xiao, Naijia Jiang, Yeting Li, Nanyu Zhong, Bingwei Peng, Kunpeng Jian, and Wei Huo. 2025. From Constraints to Cracks: Constraint Semantic Inconsistencies as Vulnerability Beacons for Embedded Systems. In *34th USENIX Security Symposium (USENIX Security 25)*. 685–704.

[81] Yajin Zhou, Lei Wu, Zhi Wang, and Xuxian Jiang. 2015. Harvesting developer credentials in android apps. In *Proceedings of the 8th ACM conference on security & privacy in wireless and mobile networks*. 1–12.

A Study on UoWC Vulnerability

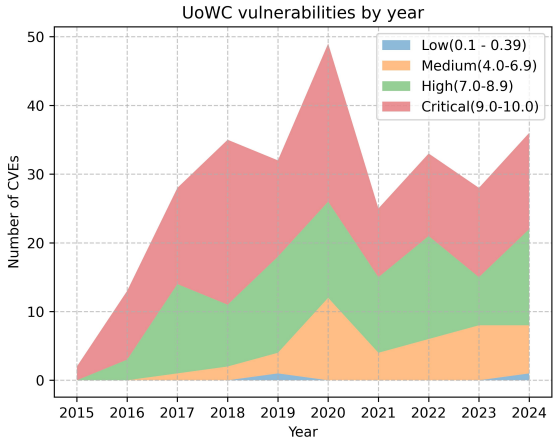


Figure 5: CVSSv3 scores of CVEs caused by weak credentials.

To systematically assess the prevalence and severity of Use of Weak Credential (UoWC) vulnerabilities, we collected a dataset of 290 CVEs reported over the past 10 years. Our dataset encompasses vulnerabilities arising from the use of default, hard-coded, and predictable credentials. We collected the vulnerabilities through querying UoWC-associated CWEs (including CWE-798, CWE-1390, CWE-1391, CWE-1392) in the National Vulnerability Database (NVD) and filtering for entries relevant to embedded firmware. As shown in Figure 5, UoWC vulnerabilities have remained consistently prevalent across the surveyed period, and a significant proportion (82.7%) of these vulnerabilities are classified as high or critical, highlighting the harmfulness of UoWC vulnerabilities.

B Choice of Threshold and Weight

During credential validation site identification, we assign weights to each feature based on the insights detailed in §4.1. Specifically, we assign 10 to the compilation feature, 15 to the input parsing feature, and 5 to the credential validation and output generation feature. To reduce false positives (i.e., non-authentication routines) as described in §4.1.2, we deprioritize input parsing features by multiplying their weight with a decay factor: $w_i' = w_i * e^{3-n}$ when $n \geq 3$, where n is the occurrence count of input parsing features. These weights are used to calculate scores, which will be compared with an empirical threshold to determine credential validation sites.

We conducted an experiment to find an appropriate threshold. Specifically, we selected 20 firmware samples from 13 vendors to form a threshold dataset. We ensured that each vendor has at least one firmware selected. Then, we manually labeled credential validation sites through reverse engineering. After that, we ran FirmCred on this dataset to identify and score candidates of credential validation sites. Finally, we varied the threshold from 4.0 to 11.0 in increments of 0.1, measuring recall, precision, and F1 score of credential validation site identification at each step.

As shown in Figure 6, a threshold of 6.2 to 8.8 yields the best trade-off between recall and precision, achieving an F1 score of 0.874. Therefore, we selected 7.5 as the threshold score for credential validation site identification.

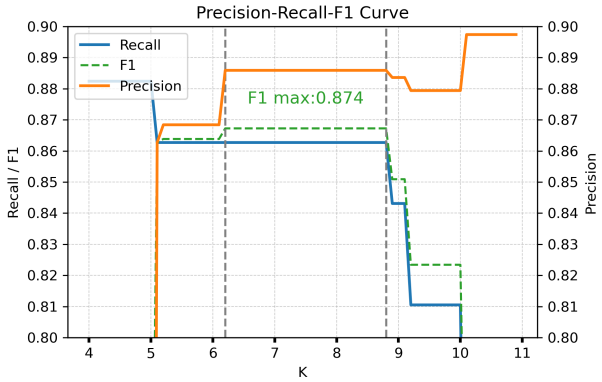


Figure 6: Credential validation site identification results under different score thresholds.