CASE STUDY

Firmware Analysis and Security Assessment of Embedded Smart Home Devices: A Laboratory-Based Pedagogical Framework

Kimbely Thornton[1] (kthornton@se.edu), Rachel Whitfield[1], Samantha Reeves[1]

[1]Department of Chemistry, Computer, Physical Sciences, Southeastern Oklahoma State University, Durant, OK, USA

Abstract

The widespread adoption of Internet of Things devices in residential and commercial environments has increased the risk for cyberattacks, creating an urgent demand for professionals skilled in firmware analysis and embedded systems security. Preparing computer science and cybersecurity students with practical experience in these areas remains a persistent challenge in higher education. This paper presents a structured pedagogical framework comprising five progressive laboratory exercises that guide students through the process of analyzing, testing, and securing a representative smart-home monitoring prototype built on a low-cost microcontroller platform. The exercises employ a combination of static firmware inspection, dynamic behavioral testing, and network traffic analysis to expose vulnerabilities across multiple architectural layers of the web application embedded in the device. By working through these exercises, students gain hands-on competence in firmware extraction, credential exposure analysis, input validation assessment, and protocol-level weakness identification. The framework is designed for integration into undergraduate cybersecurity curricula and uses affordable, readily available hardware to minimize barriers to adoption. A pilot evaluation with upper-division students indicates gains in self-reported competence while highlighting the need for larger studies using objective assessment measures. The laboratory structure, setup requirements, and assessment approach are described to support curricular adoption and replication.

Keywords — Embedded Systems Security; Firmware Analysis; Internet of Things; Cybersecurity Education; Smart Home Devices; Vulnerability Assessment

1 Introduction

The proliferation of network-connected embedded devices has fundamentally transformed the technology environment over the past decade. Early Internet of Things (IoT) research anticipated rapid growth in connected sensing, actuation, and networked embedded systems across consumer, industrial, and civic settings [1,2]. This expansion encompasses a broad spectrum of device categories, from industrial sensors and medical wearables to consumer-oriented smart home products such as network cameras, smart thermostats, voice assistants, and connected appliances. The convenience afforded by these devices has driven rapid consumer adoption, yet it has simultaneously expanded the attack surface available to malicious actors [3].

The security implications of this expansion are substantial. Large-scale attacks leveraging compromised IoT devices, such as the Mirai botnet incident in 2016, demonstrated that vulnerabilities in embedded firmware can be weaponized to launch distributed denial-of-service (DDoS) attacks of unprecedented scale [4,5]. Research has shown that a significant proportion of commercially available embedded devices ship with known vulnerabilities, including hard-coded credentials, unencrypted communication channels, and insufficient input validation mechanisms [6,7]. These weaknesses are often attributable to the constrained development environment of embedded systems, where computational resources, memory capacity, and power budgets impose design trade-offs that may compromise security [8].

Against this backdrop, the cybersecurity workforce faces a well-documented skills gap. Cybersecurity education research has repeatedly emphasized the difficulty of preparing graduates with the practical, systems-oriented skills expected in professional security roles [9]. Traditional computer science and cybersecurity curricula have historically emphasized software development from the ground up, with comparatively less attention devoted to the analysis, testing, and modification of existing systems [10,11]. This pedagogical gap is particularly acute in the domain of embedded systems, where students must contend with hardware–software interactions, cross-compilation toolchains, binary analysis, and constrained operating environments that differ markedly from conventional desktop or server programming contexts.

Reverse engineering, broadly defined as the process of analyzing a subject system to identify its components and their interrelationships [12], offers a powerful methodological framework for addressing this gap. When applied to embedded IoT devices, reverse engineering enables the systematic examination of firmware images, communication protocols, and application logic to uncover security weaknesses that may not be apparent through conventional testing approaches [13]. Furthermore, the discipline of reverse engineering cultivates transferable analytical skills, including pattern recognition, systematic decomposition, and hypothesis-driven investigation. These skills are broadly applicable across cybersecurity practice [14].

Despite the clear educational value of reverse engineering for IoT security, existing pedagogical resources remain limited. Prior work has proposed tools and frameworks for teaching reverse engineering of compiled executables [15], UML-based analysis of web applications [16], and firmware analysis of full-stack Linux-based

Submitted: June 8, 2023 Accepted: June 22, 2023 Published: October 1, 2023
www.pearlacademicpublishing.com

devices [13]. However, few comprehensive laboratory curricula target the specific intersection of embedded microcontroller-based IoT devices, web application security, and firmware analysis in a format suitable for undergraduate instruction.

This paper addresses that gap by presenting a laboratory framework for teaching firmware analysis and security assessment of embedded smart home devices. The framework is organized around five progressive exercises that guide students in examining a smart home environmental monitoring device built on a low-cost microcontroller platform. The device hosts an embedded web application that provides sensor data visualization and device configuration capabilities, offering a realistic and pedagogically tractable target for security analysis. We present a structured laboratory curriculum that teaches firmware extraction, credential analysis, input validation testing, network protocol inspection, and secure remediation for an embedded IoT web application. The laboratory is designed based on a pedagogical framework that combines static analysis, dynamic testing, and network-level assessment to provide students with an understanding of embedded systems security. Furthermore, we discuss findings from a study that evaluated the effectiveness of the framework with a cohort of upper-division undergraduate students and demonstrated measurable improvements in practical security assessment skills.

2 Related Work

The academic literature on reverse engineering as a discipline traces its modern formalization to the early 1990s, when Chikofsky and Cross [12] introduced a widely adopted taxonomy distinguishing reverse engineering from related activities such as restructuring and re-engineering. Müller et al. [17] subsequently presented a comprehensive roadmap for the field, identifying key research challenges and application domains. These foundational works established the conceptual vocabulary that continues to inform both research and pedagogy in the area.

In the context of IoT device security, Shwartz et al. [13] proposed a detailed methodology for reverse engineering "full-stack OS devices" that run modern operating systems such as Linux. Their approach emphasizes firmware image extraction and analysis as a primary technique for identifying security vulnerabilities. While comprehensive for its target device category, this methodology does not directly address the challenges associated with microcontroller-based devices that run bare-metal firmware or lightweight real-time operating systems. Such devices, which constitute a large fraction of the consumer IoT market, present distinct analysis challenges due to their proprietary toolchains, limited debugging interfaces, and tightly coupled hardware–software architectures.

Costin et al. [6] conducted a large-scale analysis of embedded firmware images collected from device manufacturers, identifying thousands of vulnerabilities including hard-coded credentials, cryptographic weaknesses, and known vulnerable library versions. Their work demonstrated that automated static analysis of firmware images can reveal systemic security deficiencies across the embedded device ecosystem. Chen et al. [18] extended this line of research by developing techniques for dynamic analysis of Linux-based embedded firmware via emulation, and Zaddach et al. [19] proposed the AVATAR framework to bridge hardware-dependent and emulated analysis environments.

From a pedagogical perspective, Ali [10] articulated a compelling case for incorporating reverse engineering into undergraduate software engineering curricula, arguing that the ability to comprehend and modify existing systems is an essential professional competency insufficiently addressed by traditional design-oriented coursework. Taylor and Collberg [15] developed Tigress, an automated code obfuscation tool paired with a web-based learning environment that enables instructors to generate reverse engineering challenges of calibrated difficulty. Student evaluations of this tool indicated that it effectively lowered the barrier to entry for practicing reverse engineering, though its scope was limited to compiled C programs rather than embedded systems or IoT applications.

Bellettini et al. [16] proposed WebUml, a tool for automated extraction of UML models from web applications as an aid to reverse engineering. While useful for understanding the architecture of web-based systems, this approach does not address the hardware and firmware layers central to IoT device security analysis.

The broader cybersecurity education literature has emphasized the importance of experiential, laboratory-based learning for developing practical security skills. Conklin et al. [9] analyzed structural factors influencing cybersecurity education in the United States and emphasized the importance of aligning curricula with practical workforce needs.

Several studies have examined the security of IoT devices from a technical perspective without specifically addressing pedagogical applications. Alaba et al. [20] provided a comprehensive survey of IoT security threats, taxonomizing vulnerabilities across physical, network, software, and encryption layers. Neshenko et al. [21] conducted an exhaustive survey of IoT vulnerabilities and empirically examined Internet-scale exploitation patterns. These surveys provide valuable context for understanding the technical risks that educational programs must prepare students to assess.

The present work builds upon these foundations by providing an integrated laboratory curriculum that combines firmware analysis, web application testing, and network security assessment within a single pedagogical framework targeting microcontroller-based IoT devices. To our knowledge, no prior work has offered a similarly comprehensive sequence of laboratory activitis specifically designed for teaching embedded IoT security assessment at the undergraduate level using affordable, representative low-cost hardware.

Table 1: Architectural layers of the embedded IoT device and associated security considerations.

Layer	*Description*	*Security Considerations*
Physical	Hardware interfaces (USB/UART, GPIO, JTAG)	Physical extraction of firmware; unauthorized hardware access
Data Link	Wi-Fi (802.11 b/g/n), Bluetooth	Eavesdropping; rogue access point attacks; deauthentication
Network	IP addressing, DNS, DHCP	ARP spoofing; DNS hijacking; network scanning
Transport	TCP connections, HTTP protocol	Man-in-the-middle; session hijacking; unencrypted transport
Application	Embedded web server, sensor logic, configuration handlers	Input validation failures; injection; insecure credential storage

3 Laboratory design

The proposed laboratory focuses on the smart home segment, which represents a prominent category within the broader IoT ecosystem [1]. Devices such as network-connected cameras, smart plugs, environmental sensors, and home automation controllers have become commonplace in residential settings. These devices typically combine a microcontroller or system-on-chip (SoC) with wireless communication capabilities (Wi-Fi, Bluetooth, or Zigbee), one or more sensors or actuators, and an embedded web server that provides a browser-based configuration and monitoring interface [22]. From a security perspective, smart home devices present several characteristic vulnerabilities [20, 21]. Manufacturers frequently prioritize ease of deployment and low unit cost over security hardening, resulting in devices that ship with default or hard-coded credentials, unencrypted communication channels, and firmware that includes known vulnerable library versions. The constrained computational resources of many smart home devices limit the applicability of conventional security mechanisms such as TLS with strong cipher suites, intrusion detection systems, and automatic update mechanisms [8]. Furthermore, consumer users typically lack the technical expertise to audit or harden the security posture of their devices, making them attractive targets for large-scale automated exploitation [4].

3.1 Platform

The laboratory exercises described in this paper center on a compact environmental monitoring device built around a dual-core 32-bit microcontroller with integrated Wi-Fi and Bluetooth capabilities. The specific platform used in our implementation is a widely available development board based on the Espressif SoC family, selected for its low cost (approximately $5–$12 per unit), extensive documentation, and active developer community. The board integrates 520 KB of SRAM, 4 MB of flash memory, and support for external sensors, including temperature, humidity, and motion detectors.

The device runs a lightweight embedded web server application that provides the following capabilities through a browser-based interface:

- Real-time display of environmental sensor readings (temperature, humidity, ambient light level).
- Configurable alert thresholds for each monitored parameter.
- Device configuration options including Wi-Fi network credentials, sensor polling intervals, and display preferences.
- A simple logging facility that records sensor events and configuration changes.

The application is developed using the Arduino framework and compiled using the associated cross-compilation toolchain. The compiled firmware image is uploaded to the device via a USB-to-UART bridge interface. This architecture reflects common design patterns used in many low-cost connected prototypes and provides a realistic yet pedagogically manageable target for security analysis.

For security analysis, the architecture of the device can be decomposed into five conceptual layers, each with distinct attack surfaces and requiring different analytical techniques. Table 1 summarizes these layers and their associated security considerations.

The laboratory encourages students to complete a comprehensive security assessment that addresses vulnerabilities at each layer, as weaknesses at one level can enable or amplify attacks at other levels. The laboratory exercises presented in this paper are structured to span these layers, providing students with experience in cross-layer vulnerability analysis.

The laboratory environment requires the following hardware and software components, summarized in Table 2.

The target hardware can be assembled from low-cost commodity components; in the authors' pilot deployment, the per-station hardware cost, excluding the host computer, was approximately $15–$20.

Table 2: Hardware and software requirements for the laboratory exercises.

Component	*Category*	*Purpose*
Microcontroller board	Hardware	Target IoT device hosting the web application
USB data cable	Hardware	Firmware upload/download and serial communication
Arduino IDE (v1.8.x)	Software	Firmware compilation, upload, and serial monitoring
Firmware extraction tool	Software	Reading binary firmware images from device flash
Linux analysis VM	Software	Binary analysis, string extraction, network analysis
Wireshark	Software	Network traffic capture and protocol analysis
Code navigation tool (cscope)	Software	Cross-referencing and navigating source code databases
Web vulnerability scanner	Software	Automated detection of common web vulnerabilities

Table 3: Summary of vulnerability categories addressed in the laboratory exercises.

Lab	*Vulnerability*	*Layer*	*Analysis Method*
1	Hard-coded credentials in firmware	Physical / Application	Static firmware extraction and string analysis
2	Input validation failure	Application	Black-box dynamic testing and source code review
3	Unencrypted HTTP communication	Transport	Network traffic capture and protocol analysis
4	Missing HTTP security headers	Application	Automated vulnerability scanning
5	Insecure firmware update mechanism	Physical / Application	Firmware modification and reflashing

3.2 Vulnerability Analysis Tasks

Through systematic firmware analysis and security testing of the laboratory prototype, five categories of vulnerabilities were identified in the target web application. Each vulnerability category forms the basis of one laboratory exercise. The exercises are sequenced to build progressively from physical-layer analysis to application-layer assessment. Table 3 provides an overview of the five vulnerability categories.

3.2.1 Lab 1: Hard-Coded Credential Extraction from Firmware

The first laboratory exercise addresses the vulnerability of hard-coded credentials stored in the firmware image of the device. This is one of the most prevalent security weaknesses in IoT devices, consistently appearing in vulnerability surveys and security assessments [6, 21]. The exercise demonstrates how physical access to the USB/UART interface of the device enables an attacker to extract the firmware image and recover sensitive credentials embedded in the binary.

In the intentionally vulnerable laboratory firmware, the Wi-Fi network SSID and password used by the device are specified as string literals in the application source code and compiled directly into the firmware binary. When the firmware is uploaded to the device, these credentials are stored in the flash memory without encryption or obfuscation. An attacker with physical access to the device can reverse the firmware upload process to extract the binary image and then search it for readable strings.

The exercise proceeds through the following steps:

1. *Identify the upload toolchain.* Students configure the Arduino IDE to produce verbose output during compilation and firmware upload. By examining the verbose log, they identify the specific binary utility (the firmware flashing tool) used to write the compiled firmware to the flash memory of the device , as well as the command-line parameters used during the upload process.

2. *Determine the read-back parameters.* Students examine the documentation and command-line help output of the identified firmware tool to discover that it supports a flash read operation that reverses the upload process. The key parameters include the communication baud rate, the starting address of the flash region, the size of the flash memory (4 MB in this case), and the output filename for the extracted image.

3. *Extract the firmware image.* Using the identified tool with appropriate parameters, students read the complete contents of the flash memory into a binary file on the host computer. The extraction command

Table 4: Categories of sensitive information recovered from firmware extraction in Lab 1.

Information Type	*Risk Level*	*Implication*
Wi-Fi SSID	Medium	Identifies the target network
Wi-Fi password	Critical	Grants network access to unauthorized parties
API endpoint paths	Medium	Reveals application structure for further attacks
Debug/log messages	Low	Exposes internal logic and error handling paths
Library version strings	Medium	Enables targeted exploitation of known CVEs

specifies a baud rate of 921600 for maximum transfer speed, a starting address of 0x00000000, and a read length of 0x400000 (4 MB).

4. *Analyze the extracted binary.* Students transfer the extracted firmware image to a Linux analysis environment and use the `strings` utility to extract all human-readable character sequences from the binary file. The output contains library identifiers, debug messages, and the hard-coded Wi-Fi credentials.

5. *Locate specific credentials.* Students use the `grep` command with context display options (e.g., `grep -B2 -A2`) to search for the known SSID string within the extracted strings. The Wi-Fi password is typically located near the SSID string in the binary image, confirming the vulnerability.

Table 4 summarizes the types of sensitive information recoverable from the intentionally vulnerable firmware image during this exercise.

Upon completing the extraction and analysis, students are guided through a discussion of mitigation strategies, including provisioning-time credential injection, encrypted credential storage in dedicated flash partitions, and eliminating hard-coded secrets from source code. Students then implement a basic mitigation by modifying the firmware source to read Wi-Fi credentials from a separate, non-volatile storage partition rather than from compiled string constants.

3.2.2 Lab 2: Input Validation Failure and Out-of-Range State Handling

The second laboratory exercise addresses input validation vulnerabilities in the web application embedded in the device. Through a combination of black-box dynamic testing (manipulating application inputs via the web interface) and white-box static analysis (examining the application source code), students discover that certain configuration parameters lack adequate bounds checking, leading to out-of-range states that corrupt device operation or cause system crashes.

The embedded web application provides a browser-based control panel that allows users to adjust operational parameters, including sensor polling frequency, display refresh rate, and alert thresholds. These parameters are transmitted from the browser to the device via HTTP GET requests containing parameter name-value pairs.

The exercise proceeds as follows:

1. *Explore the web interface.* Students interact with the web application through a browser, noting the available configuration controls and their apparent value ranges. They observe that dropdown menus and slider controls constrain the values that can be submitted through the standard interface.

2. *Identify the request structure.* By examining the developer tools within the browser or the page source, students identify the URL pattern used for configuration commands. The request format follows the structure: `http://{IP}/config?param={name}&val={value}`, where `param` identifies the configuration parameter and `val` specifies the desired value.

3. *Bypass client-side controls.* Students craft HTTP requests directly in the browser address bar, bypassing the client-side input constraints imposed by the dropdown menus. This demonstrates that client-side validation alone is insufficient for security, as the underlying HTTP interface accepts arbitrary parameter values.

4. *Test boundary conditions.* Students systematically submit values outside the expected ranges for the sensor polling interval parameter:

   - A value of 0 causes the device to enter a tight polling loop, consuming all available CPU cycles and rendering the web interface unresponsive.
   - A negative value causes undefined behavior in the timer configuration, resulting in erratic sensor readings.

Table 5: Input validation test cases and observed outcomes in Lab 2.

Input Value	*Severity*	*Observed Behavior*
Valid range (1–10)	None	Device operates normally with specified polling interval
Zero (0)	High	Tight polling loop; web interface becomes unresponsive
Negative (-1)	Medium	Erratic sensor readings; timer misconfiguration
Moderately excessive (15)	Medium	Corrupted sensor data displayed; partial malfunction
Very large (65536+)	Critical	Configuration-state corruption; continuous crash-reboot cycle
Decimal (3.7)	Low	Truncated to integer; no crash but no validation feedback

- An excessively large value (exceeding the 16-bit integer range of the parameter variable) overflows the expected integer range and corrupts dependent configuration state, causing the device to enter a crash-reboot cycle.
- A decimal value is truncated to an integer by the `atoi()` function, which does not itself cause a failure but demonstrates the absence of type checking.

5. *Analyze the source code.* Students examine the relevant handler function in the application source code, confirming that the parameter value is parsed using `atoi()` without any range validation before being assigned to the configuration variable. The absence of bounds checking is identified as the root cause of the observed vulnerabilities.

6. *Use code navigation tools.* Students use `cscope` to trace the parameter variable through the codebase, identifying all locations where it is read, written, or used in computations. This cross-reference analysis reveals the full scope of the impact of the unchecked input, including its effects on timer configuration and memory allocation routines.

Table 5 summarizes the test cases and their observed outcomes.

The exercise concludes with students implementing input validation in the handler function, adding bounds checking to ensure that only values within the valid range (1–10) are accepted, and returning an appropriate HTTP error response for out-of-range inputs. Students verify their fix by repeating the boundary condition tests.

3.2.3 Lab 3: Unencrypted HTTP Communication

The third laboratory exercise addresses the absence of transport-layer encryption in the web application. Because the embedded web server uses plain HTTP rather than HTTPS, all communication between the browser and the device, including configuration commands, sensor data, and any credentials transmitted during setup, is sent in cleartext and is vulnerable to interception by any entity with access to the network.

This exercise introduces students to network traffic analysis using a packet capture tool (Wireshark) and demonstrates the practical consequences of unencrypted IoT device communication.

The exercise proceeds through the following steps:

1. *Configure network monitoring.* Students configure Wireshark on the analysis workstation to capture traffic on the wireless network interface associated with the device network. Appropriate capture filters are configured to isolate traffic to and from the IP address of the device.

2. *Generate application traffic.* While the packet capture is running, students interact with the web application, performing actions that include viewing sensor data, modifying configuration parameters, and submitting Wi-Fi credentials through the device setup page.

3. *Analyze captured packets.* Students apply display filters in Wireshark to isolate HTTP traffic and examine the captured packets. They observe that all data, including configuration parameters and credential values, is transmitted as plaintext within HTTP GET and POST request bodies.

4. *Reconstruct sensitive data.* Using Wireshark's "Follow TCP Stream" feature, students reconstruct complete HTTP request–response exchanges, demonstrating that an eavesdropper can recover the full content of every interaction with the device.

5. *Discuss mitigation challenges.* Students explore the challenges of implementing TLS on resource-constrained microcontrollers, including the computational overhead of cryptographic operations, the memory footprint of TLS libraries, and the complexity of certificate management for devices without a real-time clock or persistent network connectivity.

Table 6 presents the types of information exposed in cleartext during the network traffic analysis.

Table 6: Sensitive information exposed in cleartext HTTP traffic in Lab 3.

Data Category	*Exposure Risk*	*Observable Content*
Sensor readings	Low	Temperature, humidity, light level values
Configuration parameters	Medium	Polling intervals, threshold values, display settings
Wi-Fi credentials	Critical	SSID and password submitted during device setup
HTTP request paths	Medium	Full URL structure revealing API endpoints
Device identification	Medium	MAC address, firmware version, device hostname

Table 7: Missing HTTP security headers identified by automated scanning in Lab 4.

Missing Header	*Risk Level*	*Security Implication*
X-Frame-Options	Medium	Enables clickjacking attacks via iframe embedding
X-Content-Type-Options	Low	Permits MIME-type sniffing by browsers
Content-Security-Policy	Medium	Allows execution of injected scripts (XSS facilitation)
Strict-Transport-Security	Low	Not applicable until HTTPS is enabled; scanner reports it as absent
X-XSS-Protection	Low	Browser-level XSS filtering not activated

3.2.4 Lab 4: Missing HTTP Security Headers

The fourth laboratory exercise uses an automated web vulnerability scanner to identify the absence of standard HTTP security headers in the web application responses. While the previous exercises relied primarily on manual analysis, this exercise demonstrates the role of automated tools in systematic vulnerability assessment and helps students understand both the capabilities and limitations of tool-based approaches.

Students configure an automated web vulnerability scanning tool to assess the web application. The scanner identifies several missing security headers that, while not directly exploitable in isolation, weaken the posture of the application in terms of defense against attacks.

Table 7 summarizes the missing headers identified by the automated scan and their security implications.

The exercise concludes with students modifying the response handler of the embedded web server to include appropriate security headers, verifying the changes by re-running the automated scan, and discussing the relative effectiveness of each header in the context of a resource-constrained embedded device.

This exercise also provides an opportunity for students to critically evaluate the output of automated scanning tools. Students compare the findings of the scanner with the vulnerabilities previously identified through manual analysis, observing that certain critical vulnerabilities (such as the hard-coded credentials and input validation failures) were not detected by the automated tool. This comparison reinforces the importance of combining automated and manual approaches in comprehensive security assessment [23].

3.3 Lab 5: Insecure Firmware Update Mechanism

The fifth and final laboratory exercise addresses vulnerabilities in the update process of the device firmware. The exercise demonstrates that the same USB/UART interface used to upload legitimate firmware can be exploited to install modified or malicious firmware if the device lacks mechanisms for firmware authentication and integrity verification.

The exercise proceeds as follows:

1. *Understand the update process.* Students review the firmware compilation and upload workflow, noting that the flashing tool writes the firmware image directly to the flash memory of the device without verifying the authenticity or integrity of the image.

2. *Modify the firmware.* Students make controlled modifications to the application source code, such as adding a benign marker endpoint that demonstrates unauthorized code execution without collecting or transmitting sensitive data. They compile the modified firmware and upload it to the device.

3. *Verify the attack.* Students confirm that the modified firmware executes without any warning or error indication on the device, demonstrating the absence of application-level firmware integrity checking in the laboratory configuration [24].

Table 8: Firmware update security assessment results in Lab 5.

Security Mechanism	*Present?*	*Consequence of Absence*
Firmware signature verification	No	Arbitrary firmware can be installed
Secure boot chain	No	No assurance of firmware integrity at boot
Flash write protection	No	Firmware can be overwritten without authorization
Firmware version enforcement	No	Downgrade attacks are possible
Physical tamper detection	No	Unauthorized physical access is undetectable

4. *Assess detection difficulty.* Students examine the observable behavior (LED indicators, serial output, web interface) of the device to determine whether the firmware modification produces any detectable artifacts. In most cases, the modification is indistinguishable from the original firmware without performing a new extraction and analysis of the firmware.

5. *Discuss countermeasures.* Students explore mitigation approaches, including secure boot, firmware signing with asymmetric cryptography, flash write-protection mechanisms, and physical tamper-evident enclosures.

Table 8 summarizes the firmware integrity assessment results.

4 Evaluation

A preliminary evaluation of the laboratory framework was conducted with a cohort of 24 upper-division undergraduate students enrolled in an elective cybersecurity course. The students had completed prerequisite courses in computer networking, operating systems, and introductory cybersecurity, but students reported no prior coursework specifically focused on embedded systems security analysis or firmware reverse engineering.

4.1 Study Design

Students completed the five laboratory exercises over a six-week period, with approximately 2–3 hours of guided laboratory time allocated per exercise. Each exercise was accompanied by a structured worksheet requiring students to document their analysis process, findings, and proposed mitigations. Students also completed a pre-assessment survey and a post-assessment survey designed to measure changes in self-reported confidence and knowledge across the core competency areas addressed by the exercises.

4.2 Results

Figure 1 presents the mean pre- and post-assessment scores (on a 5-point Likert scale) for the six competency areas assessed.

All six competency areas showed improvements from pre-assessment to post-assessment, with the largest gains observed in firmware extraction and analysis (+2.34) and security remediation strategies (+1.88). These descriptive results suggest that the laboratory sequence was associated with gains in self-reported competence in areas where students initially reported the least confidence.

Student comments collected at the end of the pilot were generally positive. Students reported that using a physical IoT device, rather than a simulated environment, enhanced their engagement and provided a more realistic learning experience. Several students noted that the combination of manual and automated analysis techniques gave them a more comprehensive understanding of the security assessment process than either approach alone would have provided.

5 Conclusion

The security of embedded IoT devices represents a critical and growing concern for both the technology industry and the broader society. Preparing the next generation of cybersecurity professionals to identify, analyze, and mitigate vulnerabilities in these devices requires educational approaches that combine theoretical foundations with substantive hands-on experience. This paper has presented a structured laboratory framework that addresses this need through five progressive exercises targeting firmware extraction, input validation assessment, network traffic analysis, automated vulnerability scanning, and firmware update security.

The framework is built around an affordable microcontroller-based smart home monitoring prototype, making it accessible for deployment in undergraduate cybersecurity courses with limited budgets. The approach utilized in our work addresses multiple layers spanning physical, transport, and application security concerns. By doing

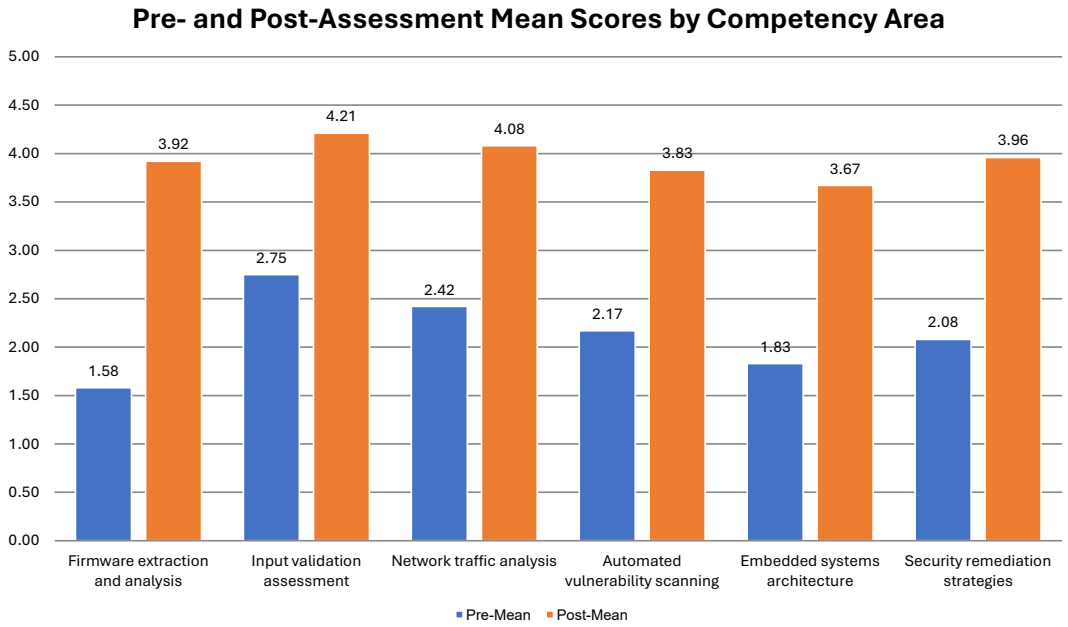

Figure 1: Pre- and post-assessment mean scores (5-point Likert scale) for student self-reported competence ($n = 24$).

this, we aim to help students connect device-level implementation choices with the broader classes of IoT security risks identified in prior work [20].

Preliminary evaluation results indicate that students who complete the laboratory sequence demonstrate improvements in self-reported competence across all targeted skill areas, with the strongest gains in areas where prior experience was lowest. The combination of manual exploration and automated, tool-based analysis is intended to reinforce critical thinking skills and help students develop a nuanced understanding of the complementary roles of different assessment methodologies in comprehensive security evaluation [14]. Indeed, the sample size ($n = 24$) is small, limiting the statistical power and generalizability of the findings. Furthermore, self-reported confidence measures may not perfectly correlate with actual competence. Future evaluations should employ larger sample sizes, control groups, and objective skill assessments to provide more rigorous evidence of the effectiveness of the framework.

The laboratory materials, including setup documentation, exercise instructions, and assessment rubrics, are designed for direct adoption by faculty seeking to incorporate embedded IoT security content into their curricula. By providing a structured pedagogical resource, this work aims to contribute to the broader effort to close the cybersecurity skills gap and prepare students for the challenges of securing an increasingly connected world.

While the current framework addresses five categories of vulnerabilities across multiple architectural layers, several avenues for expansion remain. First, the laboratory sequence could be extended to include exercises on advanced topics such as side-channel analysis, hardware fault injection, and radio-frequency protocol vulnerabilities (e.g., Bluetooth Low Energy packet sniffing and replay attacks). Second, integrating machine learning-based sensor data processing into the device firmware would create opportunities for exercises focused on adversarial machine learning and model-manipulation attacks.

Third, the development of automated grading and assessment tools could reduce the instructor workload associated with evaluating student laboratory submissions, facilitating adoption in larger course sections. Fourth, a multi-device laboratory scenario in which students must analyze the interactions and trust relationships among multiple IoT devices within a simulated smart home network would provide a more realistic and challenging learning experience [25].

Finally, longitudinal studies that track the career trajectories of the students and the application of professional skills after completing the laboratory sequence would provide valuable evidence regarding the long-term educational impact of the proposed framework.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[4] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai botnet," in *Proceedings of the 26th USENIX Security Symposium*, pp. 1093–1110, USENIX Association, 2017.

[6] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proceedings of the 23rd USENIX Security Symposium*, pp. 95–110, USENIX Association, 2014.

[7] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan," in *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, pp. 97–106, ACM, 2010.

[8] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed Internet of Things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[9] W. A. Conklin, R. E. Cline, and T. Roosa, "Re-engineering cybersecurity education in the US: An analysis of the critical factors," in *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS)*, pp. 2006–2014, IEEE, 2014.

[10] M. R. Ali, "Why teach reverse engineering?," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–4, 2005.

[11] G. Canfora and M. D. Penta, "New frontiers of reverse engineering," in *Future of Software Engineering (FOSE '07)*, pp. 326–341, IEEE, 2007.

[12] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990.

[13] O. Shwartz, Y. Mathov, M. Bohadana, Y. Elovici, and Y. Oren, "Reverse engineering IoT devices: Effective techniques and methods," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4965–4976, 2018.

[14] G. Canfora, M. D. Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *Communications of the ACM*, vol. 54, no. 4, pp. 142–151, 2011.

[15] C. Taylor and C. Collberg, "A tool for teaching reverse engineering," in *2016 USENIX Workshop on Advances in Security Education (ASE '16)*, USENIX Association, 2016.

[16] C. Bellettini, A. Marchetto, and A. Trentini, "WebUML: Reverse engineering of web applications," in *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*, pp. 1662–1669, ACM, 2004.

[17] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*, pp. 47–60, ACM, 2000.

[18] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for Linux-based embedded firmware," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*, Internet Society, 2016.

[19] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, Internet Society, 2014.

[20] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of Things security: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, 2017.

[21] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[22] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of Things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 10–16, 2017.

[23] D. Votipka, R. Stevens, E. M. Redmiles, J. Hu, and M. L. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*, pp. 374–391, IEEE, 2018.

[24] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, Internet Society, 2013.

[25] J. R. C. Nurse, S. Creese, and D. D. Roure, "Security risk assessment in Internet of Things systems," *IT Professional*, vol. 19, no. 5, pp. 20–26, 2017.