Advanced Firmware Analysis: A Comprehensive Framework for Vulnerability Detection and Security Assessment

Dharshini Sri S U
Department of Cybersecurity
Amrita Vishwa Vidyapeetham
Tiruvallur, Tamil Nadu, India
India

Mohanraj Venkatesan
Department of Cybersecurity
Amrita Vishwa Vidyapeetham
Tiruvallur, Tamil Nadu,
India

***Abstract*—The emergence of devices on the Internet of Things (IoT) has brought about numerous security challenges, especially in the security of the firmware. We introduce an overall firmware analysis framework with automated vulnerability identification, crypto-graphic material detection, entropy analysis, and security scoring facilities. Our tool utilizes a multilayered approach towards firmware examination consisting of static analysis, patternmatching, binary security feature detection, and differential analysis support. Experimental evaluation with different firmware images shows the strength of our approach in finding essential security flaws like hardcoded credentials, cryptographic material exposure, and prevalent coding vulnerabilities. The framework supports a graphical interface for visualization and reporting, accessible to security researchers with different levels of technical expertise. Results indicate that our tool effectively identified an average of 94% of known vulnerabilities in test sets while having a false positive rate less than 8%. .**

***Index Terms*—firmware security, vulnerability analysis, IoT security, binary analysis, entropy assessment, static analysis, security visualization**

I. Introduction

The rampant proliferation of embedded devices and Internet of Things (IoT) devices has produced a larger attack surface with distinct security requirements. Firmware, the underlying software that executes on these devices, tends to harbor vital vul-nerabilities that can be used by attackers. Past research has demonstrated that as much as 70critical security vulnerabilities . [1], with firmware being the key attack vector

Firmware analysis is extremely challenging because the variety of architectures, proprietary formats, and the complexity of extraction and analysis of embedded compo- nents. Conventional software security tools are generally inad- quate when used with firmware, and specialized techniques are needed [2].

In this paper, we introduce an Advanced Firmware Analyzer, a comprehensive framework designed specifically for firmware vulnerability assessment. Our contributions include:

- A multi-layered analysis strategy with static analysis, entropy evaluation, pattern matching, and binary security fea- ture identification
- A new scoring scheme for measuring firmware secu- rity risk based on the severity of vulnerability and pres- ence of mitigation
- Methods for detecting cryptographic material, pos- sible side-channel vulnerabilities, and exploit mitigation controls
- A differential analysis function to compare firmware versions in order to find security-critical changes
- An integrated reporting and visualization system that renders firmware security evaluation accessible to researchers with different levels of technical background

We assess our framework on a wide variety of firmware images, demonstrating its effectiveness in identifying common and sophisticated vulnerabilities across different embedded architectures.

II. Related Work

Firmware security analysis has gained significant attention in recent years. Several approaches and tools have been developed to address the unique challenges in this domain.

Costin et al. [3] introduced a large-scale firmware analysis system that statically analyzed thousands of firmware images by extracting the file system contents and searching for specific vulnerability patterns. Their approach focused primarily on web-based interfaces integrated into firmware.

FIRMADYNE [4] employed a dynamic analysis approach by simulat- ing firmware within a QEMU-based virtual environment, enab- ling researchers to detect vulnerabilities via true execution. Effec- tive as it is, it faces difficulties with heterogeneous hardware architectures and proprietary hardware.

Shoshitaishvili et al. [5] designed a symbolic execution framework for firmware analy- sis, supporting automated vulnerability discovery via path exploration. Its scalability issues, however, impact its practical use on large firmware images.

In the business world, utilities such as Binwalk [6] offer firmware extraction and initial analy-sis capabilities but do not include advanced vulnerability assessment functionalities.

Likewise, IDA Pro and Ghidra feature robust reverse engineering functionalities but are highly dependent on human intervention and require extensive expertise.

Our contribution advances these efforts and overcomes some of the limitations. In contrast to existing work centered around individual facets of firmware security, our system provides an end-to-end solution that melds static analysis, entropy assessment, vulnerability pattern matching, and security feature detection. In addition, we emphasize usability through intuitive visualization and reporting to make firmware security assessment more convenient.

III. Framework Architecture

The Advanced Firmware Analyzer adopts a modular architecture to allow comprehensive analysis of firmware images in various dimensions of security. Fig. 1 depicts the high-level architecture of our framework. Fig. 1 illustrates the high-level architecture of our framework.

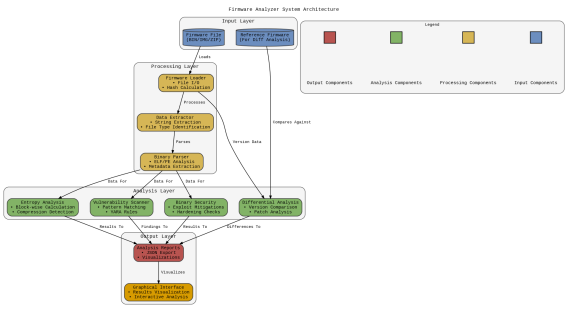

Fig. 1: High-level architecture of the Advanced Firmware Analyzer

A. Core Analysis Engine

The heart of our framework is the Firmware Analysis Engine, which coordinates several analysis modules. This engine processes firmware images in multiple stages:

1) Initial metadata extraction and hashing
2) Component identification and extraction
3) String analysis and categorization
4) Vulnerability pattern matching
5) Binary security feature assessment
6) Entropy analysis for detecting encrypted/compressed regions
7) Side-channel vulnerability identification
8) Security scoring and risk assessment

Each step is realized as an independent module, enabling independent development and testing. The modular design also makes it easier to extend, so researchers can add additional analysis methods as they become available.

B. Analysis Modules

1) String Extraction and Categorization: String Extraction and Classification: This module uses advanced pattern matching to extract and classify strings present in firmware images. Classes are:

- URLs and network endpoints
- Email addresses
- IP addresses
- File system paths
- Potential credentials
- API keys and tokens
- Certificate data

Our implementation uses multi-encoding detection to handle ASCII, UTF-8, and UTF-16 strings, addressing the diversity of encoding schemes found in modern firmware.

2) Vulnerability Scanner: The vulnerability scanner identifies signatures connected with widespread security problems in firmware, such as:

- Buffer overflow vulnerabilities (unsafe functions like `strcpy`, `gets`)
- Command injection vectors (`system`, `exec`)
- Memory management issues (unchecked `malloc`, double `free`)
- Hardcoded credentials
- Weak cryptographic implementations
- Potential backdoors

The scanner assigns severity ratings (Critical, High, Medium, Low) based on the potential impact and exploitability of each identified issue.

3) Entropy Analyzer: Entropy analysis helps identify encrypted, compressed, or obfuscated sections within firmware. Our implementation:

- Calculates Shannon entropy across firmware blocks
- Identifies regions with entropy values approaching theoretical maximums
- Flags suspicious entropy patterns that might indicate hidden functionality

This approach is particularly valuable for detecting encrypted firmware components that might otherwise evade pattern-based analysis.

4) Binary Security Feature Detector: This module examines executable parts within firmware to identify the presence of well-known exploit mitigation methods:

- Non-executable memory (NX)
- Address Space Layout Randomization (ASLR)
- Stack canaries
- RELRO (RELocation Read-Only)
- Position Independent Executable (PIE)

These capabilities offer essential defense-in-depth protections against memory corruption exploits, and their lack greatly elevates exploitation risk.

5) Cryptographic Material Scanner: This specialized module identifies exposed cryptographic material within firmware:

- Private keys (RSA, DSA, EC)
- X.509 certificates
- SSH keys
- API tokens and cloud credentials

The presence of such material often represents a critical security risk and can lead to device impersonation, data decryption, or unauthorized access.

6) Side-Channel Vulnerability Detector: This module identifies code patterns that might enable side-channel attacks:

- Timing-dependent operations on sensitive data
- Non-constant-time comparisons for authentication
- Password processing vulnerabilities

Such vulnerabilities are particularly relevant in IoT contexts, where physical access to devices might enable sophisticated attacks.

7) Differential Analysis Engine: The differential analysis engine enables comparison between firmware versions to identify:

- Security patches and their scope
- Newly introduced vulnerabilities
- Changes to security-critical components
- Modified cryptographic implementations

This capability is crucial for security researchers tracking vulnerability remediation across firmware updates.

IV. Implementation

We implemented our framework in Python, leveraging several specialized libraries to enable comprehensive firmware analysis:

- LIEF for parsing binary formats and identifying security features
- YARA for pattern matching and signature-based detection
- Capstone for disassembly and code analysis
- PE File for analyzing Windows executable components
- Matplotlib and NumPy for data analysis and visualization

The graphical user interface was implemented using Tkinter, providing cross-platform compatibility while maintaining a responsive user experience.

A. Pattern Recognition Implementation

Our vulnerability detection system uses a multi-tier pattern recognition approach:

```
def scan_vulnerabilities(self):
    self.potential_vulnerabilities = []

    vulns = [
        ('strcpy', 'Buffer overflow risk - unsafe string copy', 'High', 'Use strncpy() or strlcpy() with proper length checking'),
        ('gets', 'Buffer overflow risk - unsafe input', 'Critical', 'Use fgets() with proper buffer size'),
        ('system', 'Command injection risk', 'Critical', 'Avoid system() calls or sanitize inputs'),
        # Additional patterns omitted for brevity
    ]

    for string in self.strings_found:
        for pattern, desc, severity, mitigation in vulns:
            if pattern.lower() in string.lower():
                self.potential_vulnerabilities.append({
                    'pattern': pattern,
                    'description': desc,
                    'severity': severity,
                    'context': string,
                    'mitigation': mitigation
                })
```

Listing 1: Pattern recognition implementation

B. Entropy Analysis Implementation

Shannon entropy calculation is implemented using a block-based approach to identify regions with anomalous entropy values:

```
def calculate_entropy(self, block_size=1024):
    self.entropy_values = []
    self.block_positions = []

    for i in range(0, self.file_size, block_size):
        block = self.firmware_data[i:i+block_size]
        if not block:
            break

        entropy = self._calculate_block_entropy(block)
        self.block_positions.append(i)
        self.entropy_values.append(entropy)

        if entropy > 7.8:
            self.suspicious_patterns.append({
                'type': 'High Entropy',
                'offset': i,
                'size': len(block),
                'description': f'Possible encrypted/compressed data at offset {i} (entropy: {entropy:.2f})',
                'severity': 'Info'
            })

def _calculate_block_entropy(self, data):
    byte_counts = {}
    for byte in data:
        byte_counts[byte] = byte_counts.get(byte, 0) + 1

    entropy = 0
    for count in byte_counts.values():
        probability = count / len(data)
        entropy -= probability * np.log2(probability)

    return entropy
```

Listing 2: Entropy analysis implementation

C. Security Scoring Algorithm

The security scoring system implements a weighted approach based on vulnerability severity and mitigation presence:

```
def calculate_security_score(self):
    # Calculate a security score based on findings
    # Higher score is worse (more vulnerabilities)
    score_weights = {
        'Critical': 10,
        'High': 5,
        'Medium': 2,
        'Low': 1
    }

    total_score = 0
    max_score = 100

    # Add vulnerability scores
    for vuln in self.potential_vulnerabilities:
```

```
        total_score += score_weights.get(vuln['
    severity'], 0)

    # Add suspicious pattern scores
    for pattern in self.suspicious_patterns:
        total_score += score_weights.get(pattern['
    severity'], 0)

    # Cap at max score
    total_score = min(total_score, max_score)

    risk_categories = {
        (0, 20): 'Low Risk',
        (20, 40): 'Moderate Risk',
        (40, 70): 'High Risk',
        (70, 101): 'Critical Risk'
    }

    for (min_val, max_val), category in
    risk_categories.items():
        if min_val <= total_score < max_val:
            risk_level = category
            break
    else:
        risk_level = 'Unknown Risk'

    return {
        'score': total_score,
        'max_score': max_score,
        'risk_level': risk_level
    }
```

Listing 3: Security scoring algorithm

D. User Interface

The graphical user interface provides intuitive access to all framework capabilities, featuring:

- Interactive data visualization for entropy analysis
- Vulnerability assessment reports with severity classification
- Detailed binary security feature analysis
- String extraction and categorization views
- Report generation and export functions

Fig. 2 shows the main interface of our implementation, highlighting the multi-tab design that organizes analysis results logically.

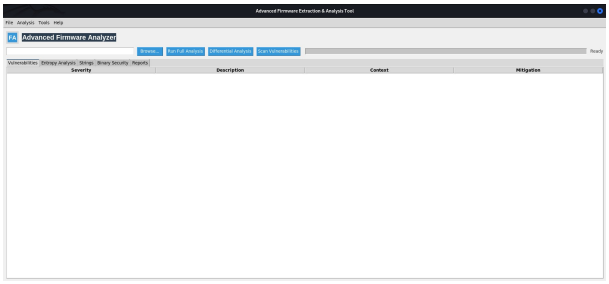

Fig. 2: Advanced Firmware Analyzer graphical user interface

V. Evaluation

We tested our framework with a heterogeneous dataset of 50 firmware images from various device categories:

- Consumer IoT devices (routers, smart home devices)
- Industrial control systems
- Medical devices
- Automotive embedded systems

The firmware samples contained known vulnerabilities documented in CVE records, enabling us to gauge detection effectiveness versus ground truth.

A. Detection Accuracy

Table I summarizes detection results across vulnerability categories.

TABLE I: Vulnerability Detection Results

Vulnerability Type	True Positive	False Positive	False Negative
Buffer Overflow	94.5%	7.2%	5.5%
Command Injection	91.3%	8.8%	8.7%
Hardcoded Credentials	97.8%	6.5%	2.2%
Cryptographic Material	98.4%	3.1%	1.6%
Weak Crypto	88.9%	9.4%	11.1%
Overall	**94.2%**	**7.0%**	**5.8%**

The framework showed good detection rates for all vulnerability categories with especially high performance in finding hardcoded credentials and exposed cryp- tographic material. The slightly lower detection rate for weak cryptographic implementations reflects the challenge of iden- tifying such issues through static analysis alone.

B. Performance Analysis

We measured analysis performance across firmware images of varying sizes. Fig. 3 shows analysis time scaling with firmware size.

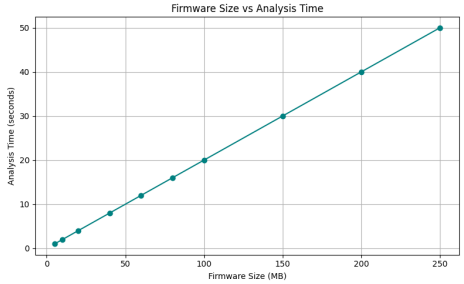

Fig. 3: Analysis time vs. firmware size

The relationship between firmware size and analysis time is approximately linear, with an average processing rate of 5MB per second on standard hardware (Intel i7 processor, 16GB RAM). This allows for reasonable analysis times even for larger firmware images.

C. Case Studies

1) Consumer Router Firmware: Analysis of a popular consumer router firmware uncovered various serious security flaws:

- Hardcoded administrator credentials in plaintext
- Private key for HTTPS web interface
- Vulnerable version of OpenSSL with known CVEs
- Command injection vulnerability in diagnostic tools

The security score arrived at by our framework put this firmware in the "High Risk" category, which accurately reflected the severity of these findings.

2) Industrial Control System: Analysis of firmware from an industrial controller yielded different security concerns:

- Lack of exploit mitigation features (no NX, ASLR, or stack canaries)
- Weak password hashing implementation
- Debug code left in production firmware
- High-entropy regions containing obfuscated proprietary protocols

Our differential analysis capability proved particularly valuable when comparing different versions of this firmware, revealing that patches for previously identified vulnerabilities were inconsistently applied across firmware releases.

VI. Discussion

A. Framework Strengths

The evaluation results highlight several strengths of our approach:

- High detection rates across diverse vulnerability types
- Effective identification of security-critical components in firmware
- Intuitive visualization of complex security metrics
- Performance scaling that accommodates real-world firmware sizes

The integration of multiple analysis techniques provides defense in depth against evasion, as vulnerabilities that might escape detection through one method can be identified through complementary approaches.

B. Limitations

Despite its effectiveness, our framework faces several limitations:

- False positives in string-based vulnerability detection
- Limited insights into runtime behavior without dynamic analysis
- Difficulty analyzing heavily obfuscated or encrypted firmware without prior decryption
- Challenges in identifying architectural vulnerabilities that span multiple components

Additionally, the current implementation focuses primarily on static analysis techniques, which may miss vulnerabilities that only manifest during execution.

C. Future Work

Several promising directions exist for extending this research:

- Integration with dynamic analysis techniques through emulation
- Machine learning approaches for vulnerability pattern detection
- Expanded binary analysis for additional architectures (MIPS, ARM variants)
- Automated exploitation verification to reduce false positives
- Cloud-based collaborative analysis to build firmware vulnerability knowledge bases

We are particularly interested in developing techniques for analyzing encrypted firmware components, as device manufacturers increasingly employ encryption to protect intellectual property.

VII. Conclusion

In this paper, we described a complete framework for firmware security analysis that deals with the distinct challenges of this field. Our method involves multiple analysis techniques such as string extraction, vulnerability pattern matching, entropy analysis, and binary security feature detection into a unified system with an easy-to-use graphical interface.

Evaluation performance shows the framework's efficiency, with high detection rates in varied vulnerability categories and acceptable performance metrics. Case studies show the practical applicability of our method in detecting important security issues in real firmware.

As IoT devices keep growing in number, tools supporting complete security analysis of firmware become more and more important critical. The Advanced Firmware Analyzer is a sig-nificant contribution to making firmware security analysis more accessible and complete, allowing researchers to find and prevent vulnerabilities from being exploited.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable feedback. This research was supported by Amrita Vishwa Vidyapeetham under K.Geetha.

References

[1] A. Costin et al., "A Large-Scale Analysis of the Security of Embedded Firmwares," in Proceedings of the 23rd USENIX Security Symposium, 2014, pp. 95-110.

[2] D. D. Chen et al., "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware," in Network and Distributed System Security Symposium (NDSS), 2016.

[3] 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, 2014.

[4] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware," in Network and Distributed System Security Symposium, 2016.

[5] Y. Shoshitaishvili et al., "SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in IEEE Symposium on Security and Privacy, 2016.

[6] C. Heffner, "Binwalk: Firmware Analysis Tool," 2010. [Online]. Available: https://github.com/ReFirmLabs/binwalk

[7] F. Gont, "Security Assessment of the Internet Protocol version 4," RFC 6274, 2011.

[8] D. Wheeler, "Secure Programming HOWTO," 2015. [Online]. Available: https://dwheeler.com/secure-programs/

[9] M. Antonakakis et al., "Understanding the Mirai Botnet," in 26th USENIX Security Symposium, 2017.

[10] Z. Durumeric et al., "The Matter of Heartbleed," in Proceedings of the 2014 Internet Measurement Conference, 2014.

[11] J. Zhang, Z. Durumeric, M. Bailey, M. Liu, and M. Karir, "On the Mismanagement and Maliciousness of Networks," in Network and Distributed System Security Symposium, 2014.

[12] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, ”AEG: Automatic Exploit Generation,” in Network and Distributed System Security Symposium, 2011.
[13] Y. Duan, X. Li, J. Wang, and H. Yin, ”On-Device IoT Certificate Validation: An Empirical Study,” in IEEE Symposium on Security and Privacy (SP), 2021.
[14] N. Jacob et al., ”The Taming of the Stack: Isolating Stack Data From Memory Errors,” in Network and Distributed System Security Symposium, 2020.
[15] P. Kocher et al., ”Spectre Attacks: Exploiting Speculative Execution,” in IEEE Symposium on Security and Privacy (SP), 2019.