Project Title: PLC (Programmable Logic Controllers) Firmware Extraction and Vulnerability Discovery

Short Project Description:

A framework for analyzing PLC firmware to identify vulnerabilities. This framework extracts firmware from PLCs, decompiles and analyzes its components, and generates reports to improve industrial control system (ICS) security.

Component	Role
Firmware Acquisition Module	Extracts firmware from PLCs
Firmware Unpacker	Parses firmware structures and extracts filesystem
Static Vulnerability Analyzer	Scans for insecure configurations, code bugs, etc
Reverse Engineering Tools	Decompiles binary code if necessary
Reporting Engine	Produces firmware vulnerability reports

Component Details:

1. **Firmware Acquisition Module**:
    - Methods:
        - Direct firmware download from vendor sites.
        - Extract via JTAG/UART from physical PLCs.
        - Etc.
2. **Firmware Unpacker**:
    - Decompresses firmware images (gzip, lzma, custom formats).
    - Reconstructs filesystem from firmware blob.
3. **Static Vulnerability Analyzer**:
    - Looks for:
        - Hardcoded passwords
        - Insecure web servers
        - Buffer overflows in binaries
        - Outdated libraries
        - Etc
4. **Reverse Engineering Tools**:
    - Ghidra, Binwalk, Radare2 to dig into compiled binaries.
5. **Reporting Engine**:
    - Lists vulnerabilities found in firmware.

Overall System Flow:

- Input: PLC firmware image
- Output: Firmware vulnerability report
- Focus: **Reverse engineering and static analysis**.

Internal Functioning of Each Module:

1. Firmware Acquisition Module

- **Sources**:
    - Public vendor portals
    - Manual device dumps:
        - JTAG/UART extraction (needs hardware interfaces)
        - SPI Flash chip reading (hardware level)
    - Etc

---

2. Firmware Unpacker

- **Unpacking steps**:
    - Identify compression:
        - Binwalk signature detection.
        - Entropy analysis (high entropy ➔ compressed/encrypted).
    - Unpack known formats:
        - LZMA, gzip, squashfs, cramfs, etc
    - File system recovery:
        - Rebuild extracted Linux file systems if firmware is Linux-based.

---

3. Static Vulnerability Analyzer

- **Scanning**:
    - Identify:
        - Hardcoded credentials in configs
        - Open Telnet servers
        - Outdated software (e.g., BusyBox, OpenSSL, etc)
        - Etc
- **YARA rules**:
    - Detect common malware modifications in PLC firmware (e.g., Mirai-like implants).

---

4. Reverse Engineering Tools

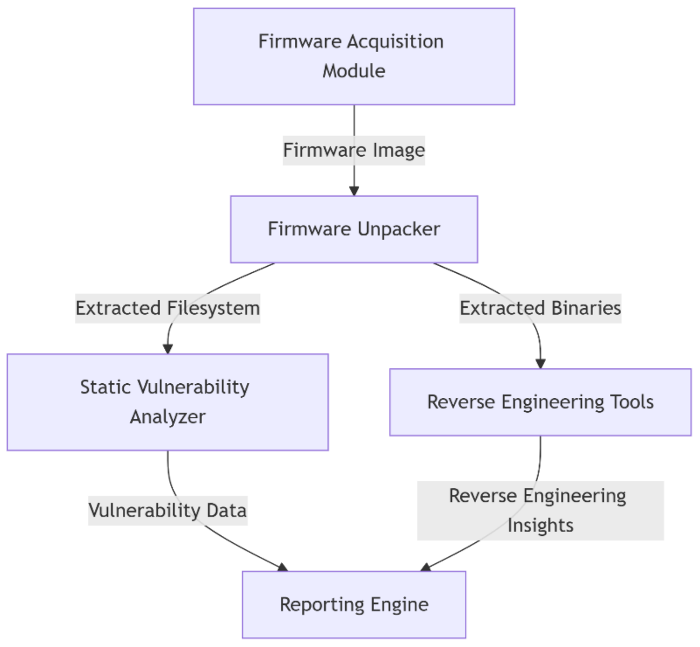
- **When binaries are present**:
  - Ghidra reverse-engineers unknown binaries.
  - Analysis focuses on:
    - Network services
    - Web servers inside PLCs
    - Management interfaces
    - Etc

---

5. Reporting Engine

- **Sections**:
  - Vulnerable services
  - Extracted credentials
  - Suggested firmware hardening steps
  - Etc

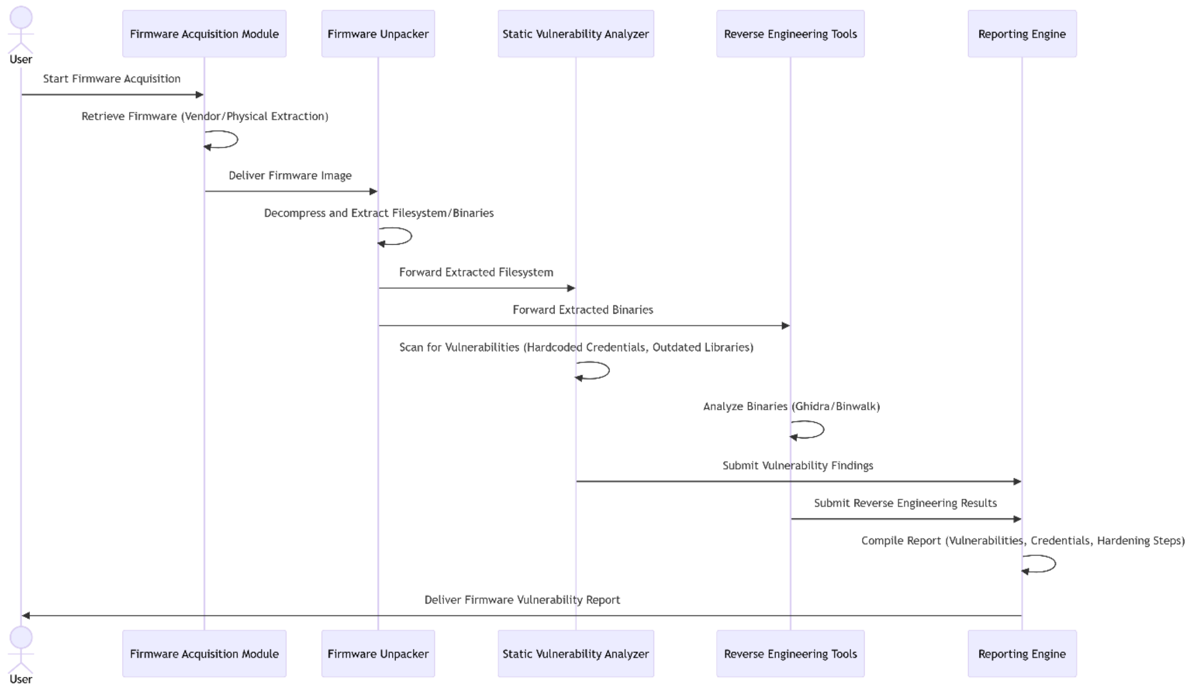
Component Diagram


Firmware Acquisition Module
Firmware Image
Firmware Unpacker
Extracted Filesystem
Extracted Binaries
Static Vulnerability Analyzer
Reverse Engineering Tools
Vulnerability Data
Reverse Engineering Insights
Reporting Engine

+

- **Firmware Acquisition Module** provides raw firmware to the **Unpacker**.
- **Unpacker** splits outputs into two streams:
  - Filesystem data → **Static Analyzer** (for configs, outdated libraries).
  - Binaries → **Reverse Engineering Tools** (for decompilation).
- Both analysis components feed results into the **Reporting Engine** for consolidation.

Sequence Diagram


User
Firmware Acquisition Module
Firmware Unpacker
Static Vulnerability Analyzer
Reverse Engineering Tools
Reporting Engine
Start Firmware Acquisition
Retrieve Firmware (Vendor/Physical Extraction)
Deliver Firmware Image
Decompress and Extract Filesystem/Binaries
Forward Extracted Filesystem
Forward Extracted Binaries
Scan for Vulnerabilities (Hardcoded Credentials, Outdated Libraries)
Analyze Binaries (Ghidra/Binwalk)
Submit Vulnerability Findings
Submit Reverse Engineering Results
Compile Report (Vulnerabilities, Credentials, Hardening Steps)
Deliver Firmware Vulnerability Report

- **User** initiates the process.
- **Firmware Acquisition** retrieves firmware via vendor portals or hardware extraction.
- **Unpacker** decompresses the firmware and separates filesystem/binaries.
- **Parallel Analysis**:
    - Static analysis (credentials, outdated software).
    - Reverse engineering (binary decompilation, network service analysis).
- **Reporting Engine** combines results into a structured report for the user.

Detailed Project Description: PLC Firmware Extraction and Vulnerability Discovery

A system for analyzing PLC firmware to identify vulnerabilities. This system extracts firmware from PLCs, decompiles and analyzes its components, and generates reports to improve industrial control system (ICS) security.

---

1. System Overview

The system extracts PLC firmware via vendor portals or hardware interfaces, unpacks its structure, and performs static analysis to identify vulnerabilities such as hardcoded credentials, outdated libraries, and insecure configurations. It combines reverse engineering and automated scanning to support firmware hardening.

---

2. Component Design & Implementation

2.1 Firmware Acquisition Module

Functionality:

- Retrieves firmware from vendor sources or physical PLCs.

Implementation Steps (e.g.):

1. **Vendor Downloads**:
    - Use vendor APIs or portals (e.g., Siemens Support) to fetch firmware updates.
    - Example: Download `.upd` or `.bin` files for Siemens S7-1200 PLCs.
2. **Hardware Extraction**:
    - **JTAG/UART**: Use tools like **JTAGulator** or **OpenOCD** to interface with PLC debug ports.
    - **SPI Flash Dumping**: Extract firmware directly from memory chips with **Flashrom** or **Bus Pirate**.
3. **Ethical Safeguards**:

    - Ensure proper authorization for hardware extraction.
    - Store firmware securely (encrypted storage).

Output:

- Raw firmware image (e.g., `plc_firmware.bin`).

Tools (e.g.):

- JTAGulator, OpenOCD, Flashrom, Bus Pirate, etc.

2.2 Firmware Unpacker

Functionality:

- Decompresses firmware and extracts filesystems/binaries.

Implementation Steps (e.g.):

1. **Signature Detection**:
    - Use **Binwalk** to identify embedded files and compression formats:
      ```
      binwalk -Me plc_firmware.bin
      ```
2. **Entropy Analysis**:
    - Detect encrypted/compressed regions using `binwalk -E`.
3. **Unpacking**:
    - Extract common formats (e.g., squashfs, cramfs, etc) with `unsquashfs`.
    - Handle custom formats using **firmware-mod-kit**.

Output:

- Extracted filesystem (e.g., `/bin`, `/etc`) and binaries.

Tools (e.g.):

- Binwalk, firmware-mod-kit, dd, unsquashfs, etc.

2.3 Static Vulnerability Analyzer

Functionality:

- Scans firmware for insecure configurations and vulnerabilities.

Implementation Steps (e.g.):

1. **Credential Hunting**:
   - Search for hardcoded passwords in config files:
     ```
     grep -r "password" extracted_fs/etc
     ```
2. **Outdated Software Detection**:
   - Check versions of libraries (e.g., OpenSSL, BusyBox, etc) against CVE databases.
3. **YARA Rules**:
   - Detect malware patterns or insecure code
   - Example:
     ```
     rule Hardcoded_SSH_Key {
       strings: $private_key = "-----BEGIN RSA PRIVATE KEY-----"
       condition: $private_key
     }
     ```

Output:

- List of vulnerabilities (e.g., "Hardcoded SSH key in `/etc/shadow`").

Tools:

- YARA, grep, CVE databases (NVD), etc.

2.4 Reverse Engineering Tools

Functionality:

- Decompiles binaries to uncover hidden vulnerabilities.

Implementation Steps (e.g.):

1. **Decompilation**:
   - Use **Ghidra** to analyze PLC binaries (e.g., `httpd` for web interfaces):
     ```
     ghidraRun  # Launch Ghidra and import binary
     ```
2. **Focus Areas**:
   - Network services (e.g., authentication logic in `libauth.so`).
   - Command injection vulnerabilities in CGI binaries.
3. **Dynamic Analysis (Optional)**:
   - Emulate binaries with **QEMU** to trace execution paths.

Output:

- Decompiled code snippets and vulnerability hypotheses.

Tools (e.g.):

- Ghidra, Radare2, QEMU, etc.

2.5 Reporting Engine

Functionality:

- Generates actionable reports with remediation steps.

Implementation Steps (e.g.):

1. **Data Aggregation**:
   - Merge findings from static analysis and reverse engineering.
2. **Template Design**:
   - **HTML**: Use Jinja2 for interactive tables and search.
   - **PDF**: Convert HTML with **WeasyPrint**.
   - **Etc**.
3. **Remediation Guidance**:
   - Suggest patching outdated libraries, removing hardcoded credentials, and disabling insecure services.

Output:

- Report with:
    - Vulnerability details (CVE IDs, file locations).
    - Risk ratings (Critical/High/Medium).
    - Mitigation steps (e.g., "Upgrade OpenSSL to v3.0.7").
    - Etc.

Tools (e.g.):

- Jinja2, WeasyPrint, Pandas, etc.

---

3. Technology Stack (e.g.)

- **Firmware Extraction**: Binwalk, Flashrom, OpenOCD, etc.
- **Analysis**: YARA, Ghidra, Radare2, etc.
- **Reporting**: Jinja2, WeasyPrint, etc.

---

4. Evaluation & Validation

1. **Accuracy Testing**:
    - Test on firmware with known vulnerabilities (e.g., Siemens SIMATIC S7-1500 CVE-2022-31855).
    - Metrics: True positive rate, false positive rate, etc.
2. **Performance**:
    - Measure extraction time (seconds per MB) and decompilation speed.
3. **Real-World Validation**:
    - Partner with ICS vendors to analyze firmware in controlled environments.

---

5. Development Roadmap

1. **Phase 1**: Build Firmware Acquisition and Unpacker modules.
2. **Phase 2**: Implement Static Analyzer and integrate YARA rules.
3. **Phase 3**: Develop Reverse Engineering workflows and reporting.
4. **Phase 4 (optional)**: Validate on PLCs (e.g., Allen-Bradley, Siemens).

---

6. Challenges & Mitigations (optional)

- **Encrypted Firmware**: Use entropy analysis and collaborate with vendors for decryption keys.
- **Proprietary Formats**: Reverse engineer custom compression algorithms with **Hex Workshop**.
- **False Positives**: Validate findings manually and refine YARA rules.

---

7. Glossary

- **JTAG**: Joint Test Action Group (hardware debugging interface)
- **UART**: Universal Asynchronous Receiver-Transmitter (serial communication)
- **YARA**: Pattern-matching tool for malware research
- **CVE**: Common Vulnerabilities and Exposures
