Audit de Securitate - Embedded Systems

May 26, 2025

Obiectiv

Realizarea unui audit de securitate pentru codul sursă dintr-un proiect embedded ESP32, identificând și remediind cele mai relevante vulnerabilități de securitate conform cerințelor laboratorului 10.

Tehnologii utilizate: audit manual (OWASP Top 10, CWE), analiză statică (Semgrep), scanare automată (Trivy), remedieri conform best practices.

Top 3 Vulnerabilități Identificate

1. Folosirea funcției `delay()` în căi critice

Impact: Blocarea execuției timp de mai multe secunde creează o fereastră în care atacatorul poate efectua acțiuni nesupravegheate. Mai mult, lipsa unui watchdog sau sistem de preempțiune degradează fiabilitatea dispozitivului în scenarii reale.

Categorie: CWE-662: Improper Synchronization

Legătură cu OWASP: OWASP IoT Top 10 – **Weak, Guessable, or Hardcoded Passwords (IoT-01)** implică și lipsa unor mecanisme robuste de failover.

```
void loop() {
  client.loop();
  delay(5000); // vulnerabil - blocheaz  procesarea
}
```

Listing 1: Folosirea periculoasă a delay()

Soluție: Utilizarea metodei `millis()` pentru temporizare ne-blocantă:

```
unsigned long previousMillis = 0;
const long interval = 5000;

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    // ac iune periodic
  }
}
```

2. Credențiale WiFi codate în sursă

Impact: Includerea parolelor în codul sursă poate duce la scurgeri de date sensibile dacă repo-ul este compromis sau codul este partajat.

Categorie: CWE-798: Use of Hard-coded Credentials

Legătură cu OWASP: A3: Sensitive Data Exposure

```
const char* ssid = "MyWiFi";
const char* password = "supersecret";
```

Listing 2: Credențiale codate în clar

Soluție: Mutarea credentialelor într-un fișier criptat sau în EEPROM, sau implementarea unui sistem de provisioning:

- stocare criptată + ESP SecureStorage
- integrare cu WiFiManager pentru configurare la prima pornire

3. Lipsa validării inputului

Impact: Lipsa validării inputului permite injecții, exploatarea memoriei și comportamente neprevăzute. Orice input din rețea sau UART trebuie considerat neîncredere.

Categorie: CWE-20: Improper Input Validation

Legătură cu OWASP: A1: Injection, **A5: Security Misconfiguration**

```
int value = Serial.read(); // f  r   validare a valorii citite
```

Listing 3: Citire fără validare

Soluție: Validare strictă:

```
int value = Serial.read();
if (value >= 0 && value <= 127) {
  process(value);
}
```

Integrarea Auditului în Fluxul de Dezvoltare

Auditul de securitate trebuie să fie parte din fiecare commit sau pull request. Mai jos, un exemplu de configurare CI/CD folosind GitHub Actions:

.github/workflows/security.yml

```
name: Security Audit

on: [push, pull_request]

jobs:
  semgrep:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Run Semgrep
      uses: returntocorp/semgrep-action@v1
      with:
        config: auto

  trivy:
    runs-on: ubuntu-latest
    steps:
    - name: Install Trivy
      run: |
        sudo apt-get install -y wget
        wget https://github.com/aquasecurity/trivy/releases/latest/download/trivy_0.44.0
            _Linux-64bit.deb
        sudo dpkg -i trivy_0.44.0_Linux-64bit.deb
    - name: Scan repository
      run: trivy fs --exit-code 1 .

  dependabot:
    runs-on: ubuntu-latest
    steps:
    - name: Enable Dependabot alerts
      run: echo "Managed by GitHub UI"
```

Listing 4: Exemplu pipeline CI pentru audit de securitate

Beneficii:

- Prevenirea integrării de cod vulnerabil în repo.
- Detectarea timpurie a dependențelor periculoase.
- Automatizare completă pentru echipe dev + sec.

Concluzie

Auditul a evidențiat vulnerabilități frecvente în aplicații embedded (IoT), precum lipsa validării inputului, blocaje în execuție și management slab al credentialelor. Integrând procese automate și bune practici în dezvoltare, riscurile pot fi reduse semnificativ, îmbunătățind securitatea generală a produsului.

Integrare cu SonarQube

Descriere

SonarQube este o platformă de analiză statică a codului care detectează bug-uri, code smells și vulnerabilități. Spre deosebire de Semgrep, care folosește reguli predefinite sau personalizate bazate pe sintaxă și semnătură, SonarQube efectuează o analiză mai profundă asupra calității codului și a complexității acestuia.

Avantaje SonarQube

- Identifică erori logice și vulnerabilități în cod.
- Suportă o gamă largă de limbaje (C/C++, Python, Java, etc.).
- Interfață web intuitivă pentru vizualizarea rapoartelor.
- Ușor de integrat în CI/CD.

Integrare SonarQube în GitHub Actions

Mai jos este prezentat un exemplu de integrare a SonarQube în pipeline-ul CI folosind GitHub Actions.
Precondiții:

- Un server SonarQube disponibil (self-hosted sau în cloud).
- Un token de autentificare generat în interfața SonarQube.

```
name: SonarQube Analysis

on:
  push:
    branches:
      - main

jobs:
  sonarcloud:
    name: SonarQube Scan
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Java
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'

      - name: Cache SonarQube packages
        uses: actions/cache@v3
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar

      - name: Run SonarQube Scanner
        env:
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: |
          wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-
              scanner-cli-5.0.1.3006-linux.zip
          unzip sonar-scanner-cli-5.0.1.3006-linux.zip
```

```
          ./sonar-scanner-5.0.1.3006-linux/bin/sonar-scanner \
            -Dsonar.projectKey=project-id \
            -Dsonar.sources=. \
            -Dsonar.host.url=https://sonar.myserver.local \
            -Dsonar.login=${SONAR_TOKEN}
```

Listing 5: Exemplu GitHub Actions - analiza SonarQube

Notă: Pentru proiectele C/C++ embedded, e recomandat ca build-ul să includă și generarea de fișiere `compile_commands.json` pentru analiză mai precisă (cu `bear` sau `clang`).

Comparatie Tools

Tool	Tip	Focus principal	Proiecte Embedded
Semgrep	Analiză statică	Pattern-uri, reguli OWASP	(ușor de configurat)
SonarQube	Analiză statică	Calitate cod, code smells, buguri	(necesită build configurat)
Trivy	Scanare vulnerabilități	Dependențe, imagini Docker	(pentru containere sau deps)

Recomandare: Folosirea complementară a uneltelor – Semgrep pentru reguli rapide OWASP, SonarQube pentru audit profund și Trivy pentru infrastructura asociată (Docker).

Concluzie Finală

În urma auditului de securitate efectuat asupra unui proiect embedded, s-au identificat vulnerabilități comune, dar critice, precum blocarea execuției prin `delay()`, folosirea credentialelor în clar și lipsa validării inputului. Am arătat cum aceste probleme pot fi remediate eficient, conform bunelor practici și cu sprijinul uneltelor automate.

Integrarea unor unelte precum Semgrep, SonarQube și Trivy în pipeline-ul CI/CD oferă un cadru automatizat de detectare timpurie a riscurilor, creând un ecosistem de dezvoltare sigur, scalabil și profesional, chiar și în proiecte embedded.

Sugestii viitoare: adăugarea de fuzzing automatizat, testare pe dispozitive fizice simulate, și audit periodic de dependențe.