Penetration Testing an Embedded System

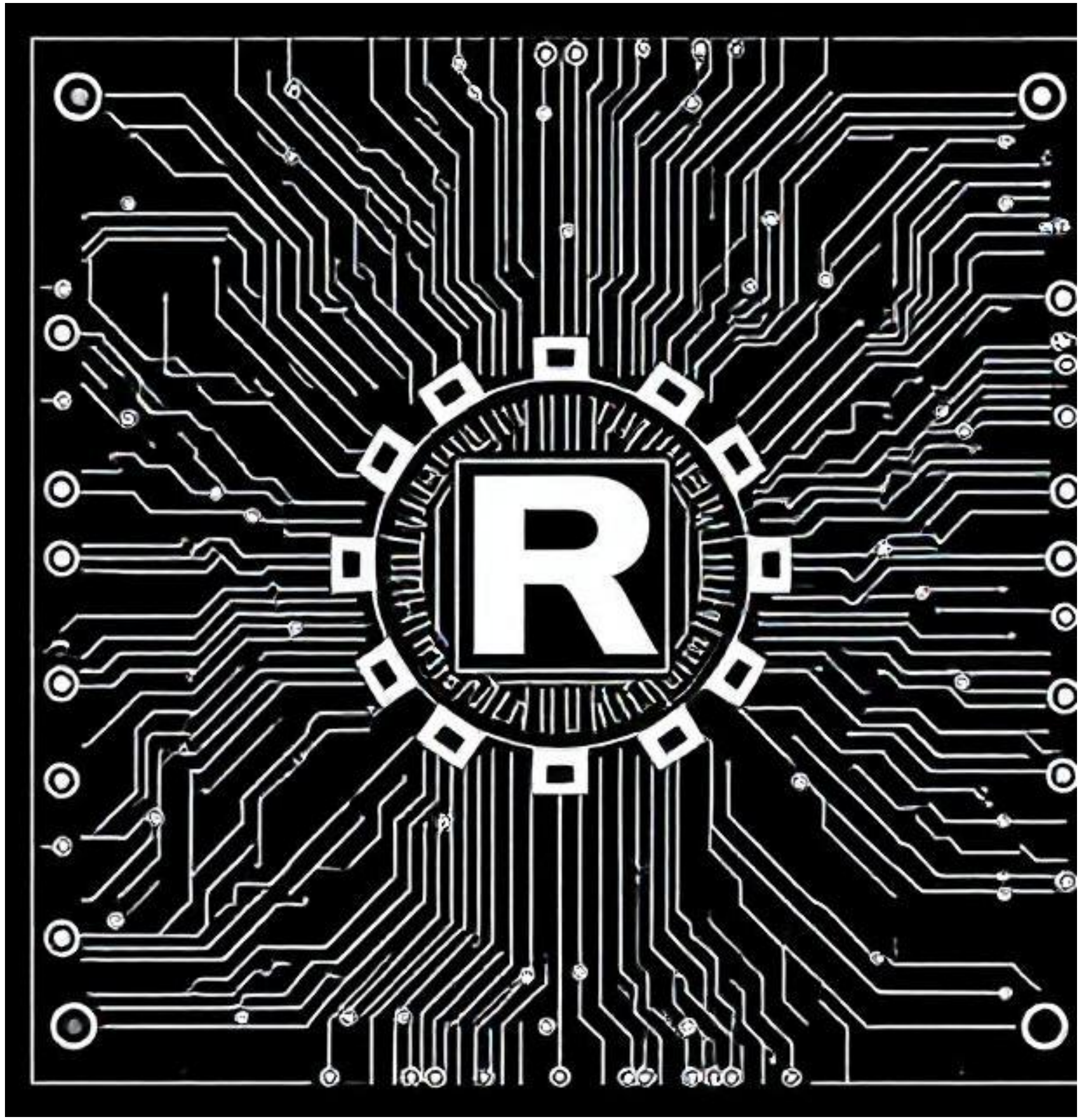
Kenneth Rockson

FOR – Flying Object Radar System

- The **Flying Object Radar (FOR)** is a **handheld, radar-based embedded system** designed to detect, track, and classify airborne objects within a **1-mile radius**. It identifies targets such as **birds, drones, planes**, and other flying objects in real time.
- FOR features an integrated **touchscreen graphical user interface (GUI)** that enables users to:
- **Visualize detected objects** and their movement
- **Interact with each target** to retrieve distance and direction data
- **Manually label and categorize** detections via on-screen input
- The device combines a **compact radar sensor**, embedded processor, and user interface to deliver **portable situational awareness** in both civilian and defense scenarios.

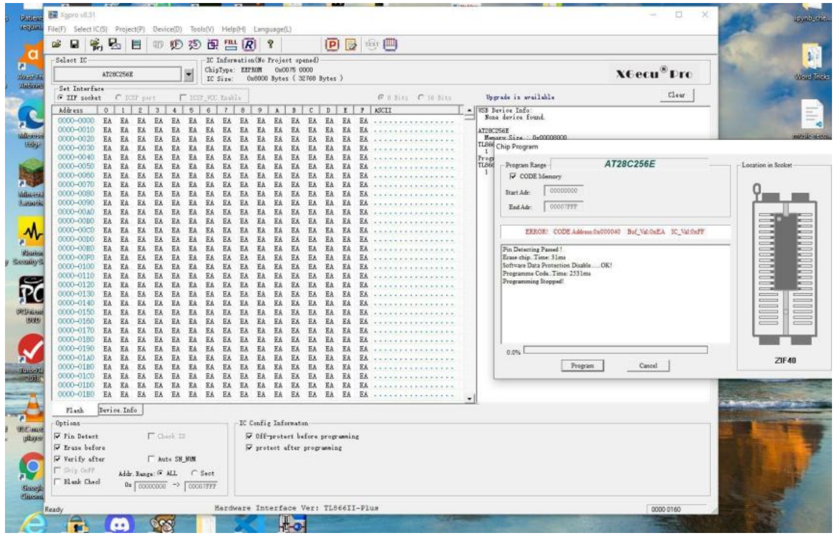
Reverse Engineering FOR: A Multi-Vector Black-Box Challenge

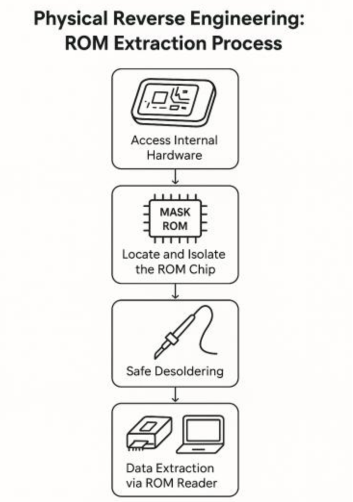
- FOR is a embedded system with no available public documentation, open APIs, or hardware schematics.
- **Objective**: To reverse engineer the device in order to uncover its internal architecture, firmware logic, and network communication protocols.
- **Approach**: The reverse engineering process targets three primary vectors of attack:
    - Hardware: Physical access and firmware extraction
    - Software: GUI exploitation and system behavior analysis
    - Communication: Network protocol interception and decoding



Physical Attack Planning – ROM Extraction

- **Step 1: Access Internal Hardware**
  - Disassemble the FOR device by carefully removing the casing and protective components.
  - Identify key internal elements, focusing on memory storage hardware.
- **Step 2: Locate and Isolate the ROM Chip**
  - Locate the non-volatile
  - ROM is targeted over RAM because it retains data even when powered off.
- **Step 3: Safe Desoldering**
  - Use precision soldering tools and heat control to desolder the ROM chip without damaging adjacent components.
  - Ensure proper anti-static precautions during handling.
- **Step 4: Data Extraction via ROM Reader**
  - Insert the extracted chip into a compatible ROM programmer/reader.
  - Use firmware reading software to extract a binary image of the ROM’s contents.
- **Step 5: Begin Firmware Analysis**
  - Transfer the binary data to a secure analysis environment.
  - Begin static analysis to search for plaintext strings, code branches, function headers, and potential entry points





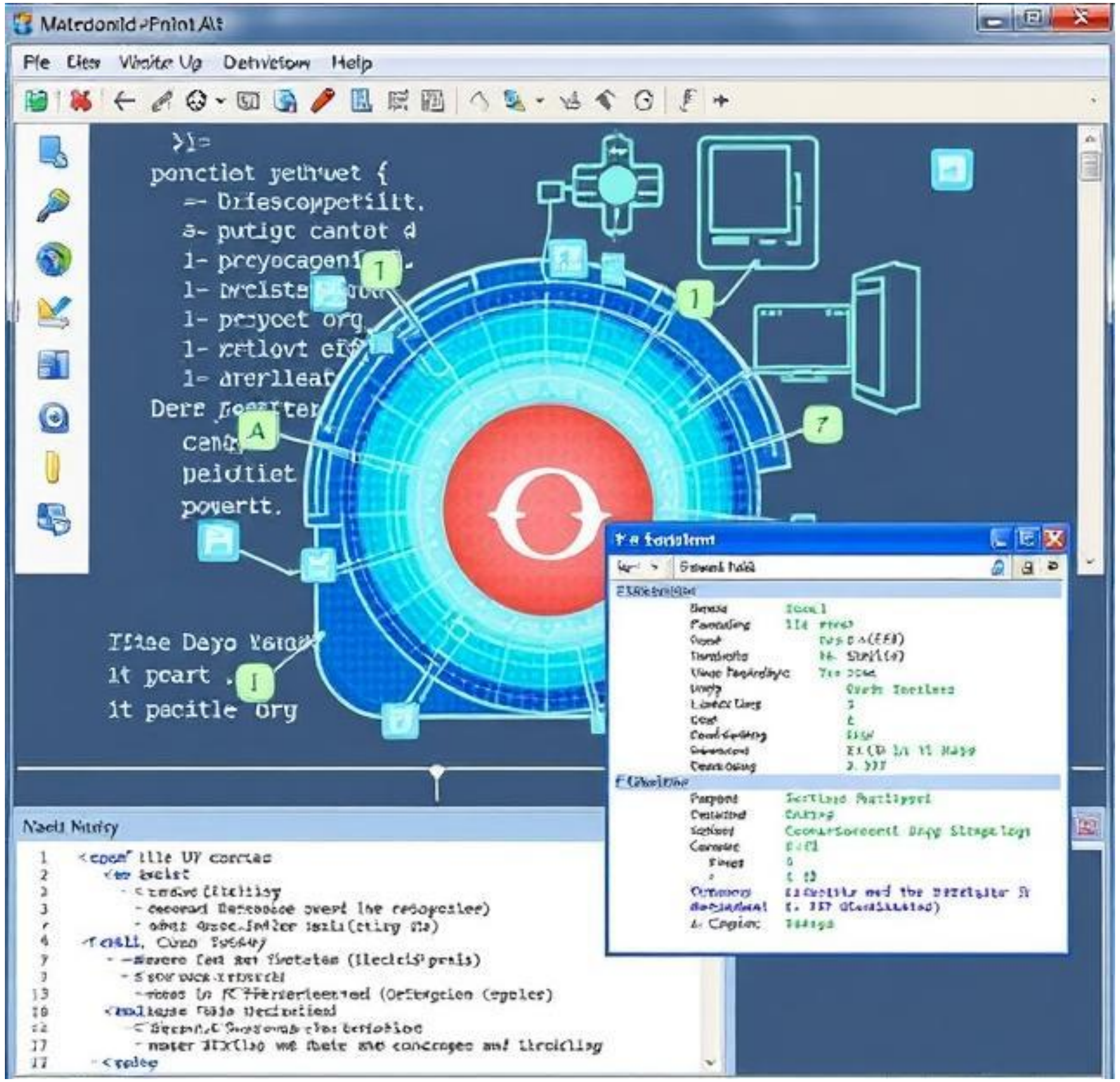
Software Attack Planning – GUI Exploitation

- During my information gathering phase, we identified an input field designed to label detected flying objects. Recognizing this as a potential attack surface, we initiated input validation and injection testing. Our approach involved several stages:

- **Buffer Overflow Testing:** We tested the field's capacity by submitting a long string ("ABCABCABC1234567890ABCABCABC") to assess buffer handling and potential overflow vulnerabilities.

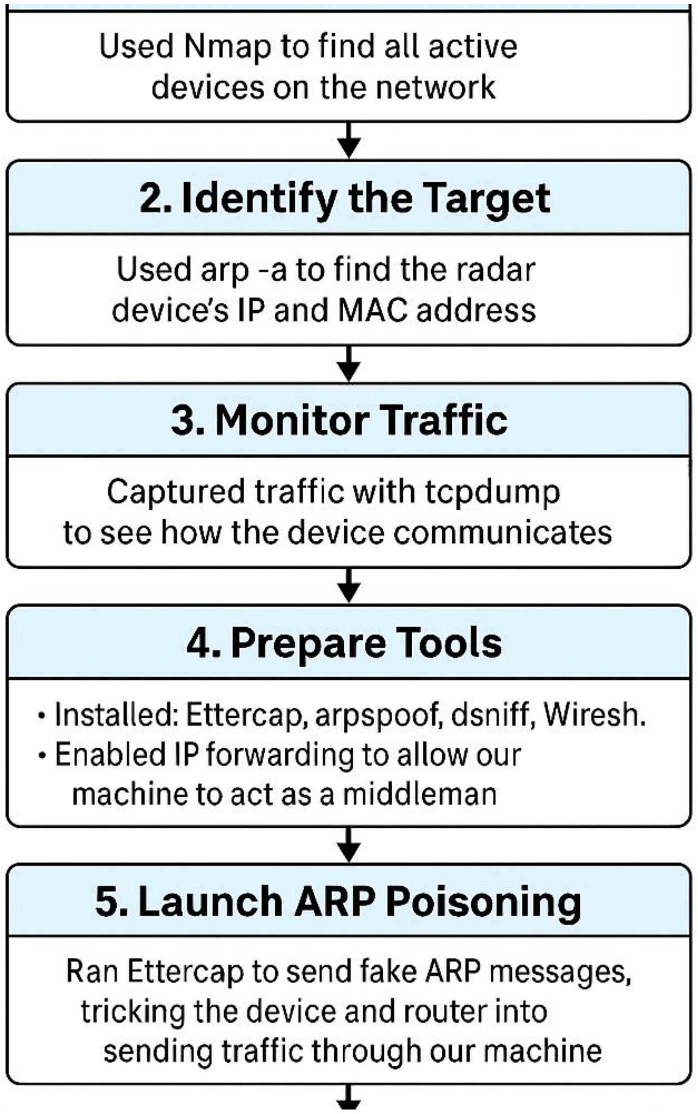
- **SQL Injection Testing:** We attempted to inject SQL syntax:
    - SELECT * FROM system_config WHERE setting LIKE '%root_password%',
  - to probe for backend database vulnerabilities.

- **Special Character and Scripting Testing:** We began with simple special characters like ">" to observe system behavior. We then escalated to attempts like
    - "Bird<script>alert(document.cookie)</script>" to test for Cross-Site Scripting vulnerabilities and analyze error responses for potential information leakage like file paths or function names.



Network Attack Planning – ARP Poisoning

- Scan the Network
  - Used Nmap to find all active devices on the network.
  - Identify the Target
  - Used arp -a to find the radar device's IP and MAC address.
- Monitor Traffic
  - Captured traffic with tcpdump to see how the device communicates.
- Prepare Tools
  - Installed: Ettercap, arpspoof, Wireshark.
  - Enabled IP forwarding to allow our machine to act as a middleman.
- Launch ARP Poisoning
  - Ran Ettercap to send fake ARP messages, tricking the device and router into sending traffic through our machine.
- Confirm Success
  - Checked the ARP table with arp -a and looked at traffic flow to confirm interception.
- Capture & Analyze Data
  - Used Wireshark to inspect traffic in real-time.
  - Focused on HTTP, DNS, and other protocols to understand how the device communicates.

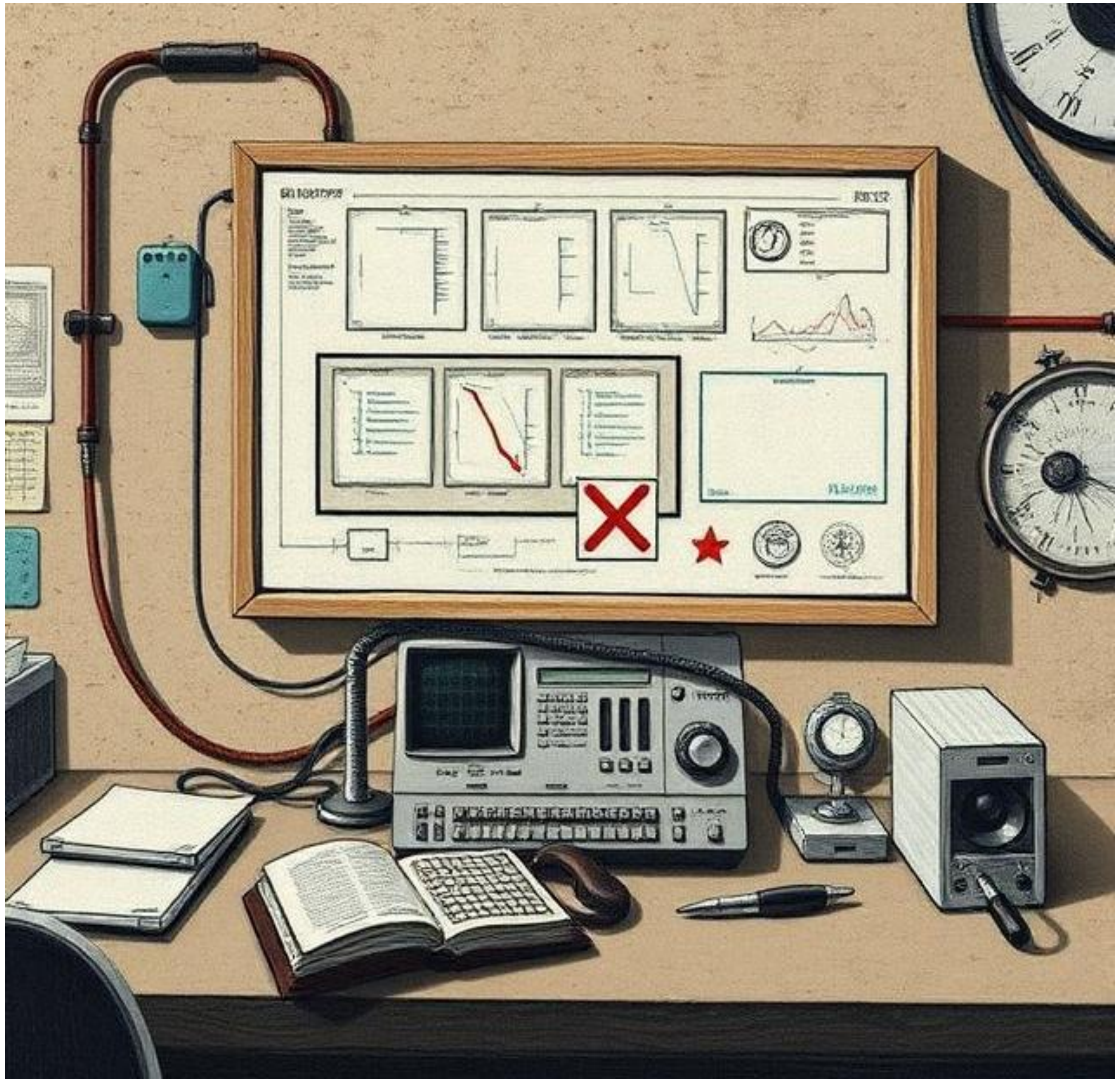


Final Solution – Reverse Engineering the FOR System

- Through this reverse engineering process, I:
  - **Mapped the internal architecture** of the FOR system
  - **Identified key vulnerabilities** across all attack surfaces
  - **Demonstrated how an attacker could extract the ROM, exploit the GUI, or execute ARP Poisoning attack**

- The result: a comprehensive security evaluation, highlighting the need for:
  - Firmware encryption & secure boot
  - Hardened access control on debug interfaces
  - Encrypted and authenticated communication protocols

- This approach can serve as a model for future security assessments of embedded radar-based systems.



Questions & Discussion

All thoughts, feedback, critiques, and suggestions are welcome and appreciated!