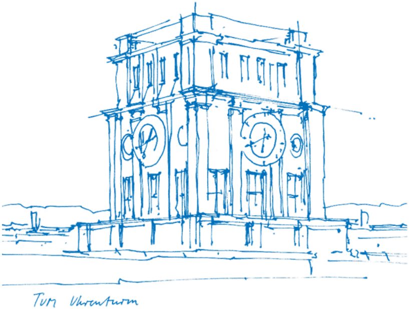
Embedded Systems and Security
Exercise 5

Matthias Probst
Technische Universität München
School of Computation, Information and Technology
Chair of Security in Information Technology
23rd November 2023



Outline

Prologue

Memory Organisation

- Macroscopic Level: Sections on the XMC4500
- Microscopic Level: Stack Frame Organisation

Memory Vulnerabilities: Size Matters

- Information Leakage by Pretending to Be Longer
- Remote Code Execution by Being Longer Than Expected

Epilogue

Outline

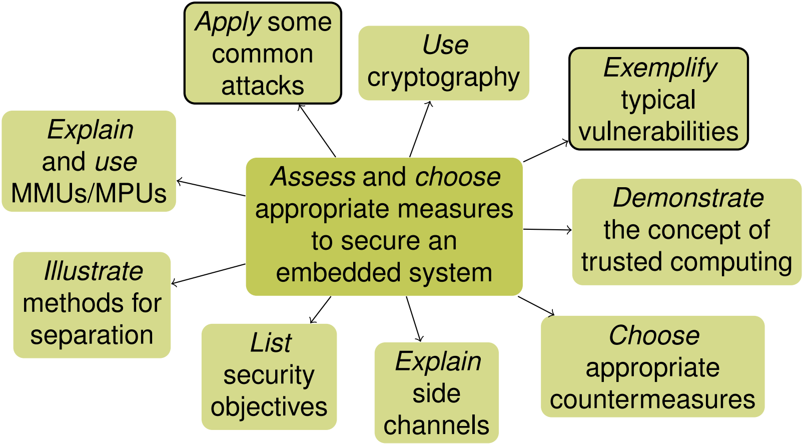
Prologue

Memory Organisation
- Macroscopic Level: Sections on the XMC4500
- Microscopic Level: Stack Frame Organisation

Memory Vulnerabilities: Size Matters
- Information Leakage by Pretending to Be Longer
- Remote Code Execution by Being Longer Than Expected

Epilogue

Today in Intended Learning Outcome 1



Assess and *choose* appropriate measures to secure an embedded system

Apply some common attacks

Use cryptography

Exemplify typical vulnerabilities

Demonstrate the concept of trusted computing

Choose appropriate countermeasures

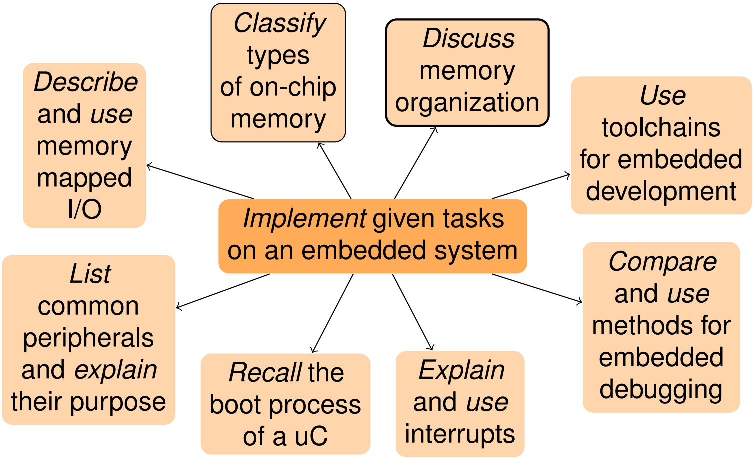
Explain side channels

List security objectives

Illustrate methods for separation

Explain and *use* MMUs/MPUs

Today in Intended Learning Outcome 2



Implement given tasks on an embedded system

Describe and *use* memory mapped I/O

Classify types of on-chip memory

Discuss memory organization

Use toolchains for embedded development

List common peripherals and *explain* their purpose

Recall the boot process of a uC

Explain and *use* interrupts

Compare and *use* methods for embedded debugging

Goals For Today

- Explain how the Heartbleed vulnerability works
- Light up an LED via remote code execution through a stack based buffer overflow
  - Tell where code, static data, heap, stack are located in embedded systems
  - Analyse a stack frame and locate important values
  - Program an exploit for it

Outline

Prologue

Memory Organisation
- Macroscopic Level: Sections on the XMC4500
- Microscopic Level: Stack Frame Organisation

Memory Vulnerabilities: Size Matters
- Information Leakage by Pretending to Be Longer
- Remote Code Execution by Being Longer Than Expected

Epilogue

Problem 5.1: Sections on the XMC4500

1. Sketch the order of sections in a regular OS based program
   1.1 Indicate in which direction heap and stack grow
   1.2 Give for each section an example what it contains
2. Solve using a `*.lst` file of an XMC4500 program:
   2.1 Determine the location of before mentioned sections on the XMC4500
   2.2 Give the maximum size of the main stack and heap
   2.3 A *stack overflow* happens, when too much data is placed on the stack and it crashes into the heap.
   Can this – or what else – happen on the XMC4500?
3. A computer program may create a file on the hard disk to store user or configuration data permanently. Explain where an embedded system may store such data.

Problem 5.1: Sections on the XMC4500

1. Sketch the order of sections in a regular OS based program
   1.1 Indicate in which direction heap and stack grow
   1.2 Give for each section an example what it contains
2. Solve using a `*.lst` file of an XMC4500 program:
   2.1 Determine the location of before mentioned sections on the XMC4500
   2.2 Give the maximum size of the main stack and heap
   2.3 A *stack overflow* happens, when too much data is placed on the stack and it crashes into the heap.
   Can this – or what else – happen on the XMC4500?
3. A computer program may create a file on the hard disk to store user or configuration data permanently. Explain where an embedded system may store such data.

Problem 5.1: Sections on the XMC4500

1. Sketch the order of sections in a regular OS based program
   1.1 Indicate in which direction heap and stack grow
   1.2 Give for each section an example what it contains
2. Solve using a `*.lst` file of an XMC4500 program:
   2.1 Determine the location of before mentioned sections on the XMC4500
   2.2 Give the maximum size of the main stack and heap
   2.3 A *stack overflow* happens, when too much data is placed on the stack and it crashes into the heap.
   Can this – or what else – happen on the XMC4500?
3. A computer program may create a file on the hard disk to store user or configuration data permanently. Explain where an embedded system may store such data.

Problem 5.2: Stack Frame Organisation

1. Sketch a typical stack frame as created by GCC (=most C compilers)
2. Discuss what happens if, e.g., a too long string
   is placed in one of the local variables

Problem 5.3: Buffer Overflow Attacks

1. The previous attack executes injected code on the stack. Decide if code execution on the stack is possible on the XMC4500. (w/o MPU)
2. Assume the stack is not executable, what else could be the target of a buffer overflow attack?

The Break

This is a three-minute break[1]

[1] According to learning scientists your brain will memorize better if the stream of information is frequently interrupted by breaks. This works the better the smaller the bursts of information are, i.e. three minutes of break every thirty minutes is more efficient than nine minutes of break every ninety minutes. A one to ten minutes scheme would be even better, but we need to trade-off at some point ;)

Outline

Prologue

Memory Organisation

- Macroscopic Level: Sections on the XMC4500
- Microscopic Level: Stack Frame Organisation

Memory Vulnerabilities: Size Matters

- Information Leakage by Pretending to Be Longer
- Remote Code Execution by Being Longer Than Expected

Epilogue

Example: Heartbleed

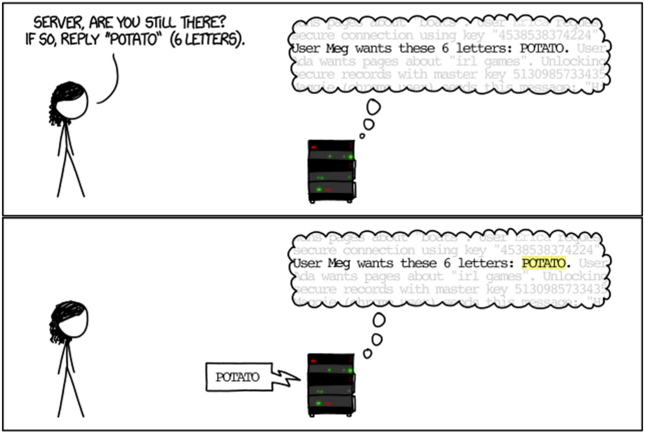

SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
ans pages about "boats". User Erica reques
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H
ans pages about "boats". User Erica reques
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H
POTATO

Image Source: `http://xkcd.com/1354` / CC BY-NC 2.5

Example: Heartbleed

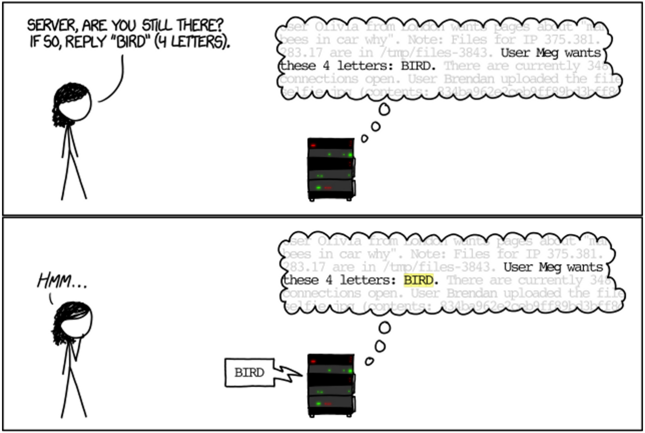

SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
ser Olivia from London wants pages about "ma bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 348 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
HMM...
ser Olivia from London wants pages about "ma bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 348 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
BIRD

Image Source: `http://xkcd.com/1354` / CC BY-NC 2.5

Example: Heartbleed

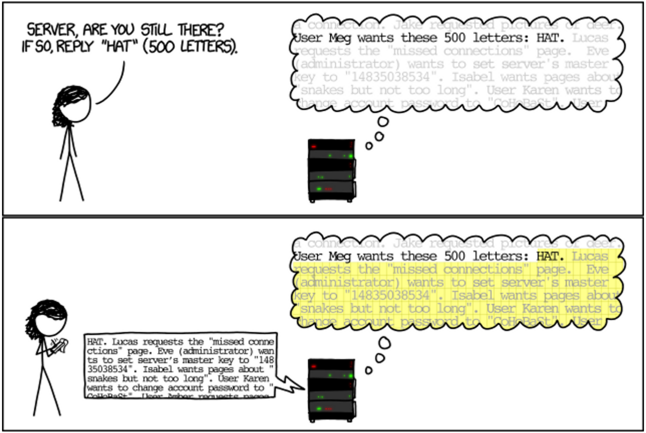

SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

Image Source: `http://xkcd.com/1354` / CC BY-NC 2.5

Recap: Endianness

- Describes how datatypes larger than one byte are ordered in memory

Little Endian Least significant byte in lowest address (=base address)

Big Endian Most significant byte in lowest address (=base address)

Example

```
uint32_t a=0x56789ABC; uint16_t b=0x1234; uint8_t c=0xDE;
```

Big Endian:

```
0x10000000: 56 78 9A BC
0x10000004: 12 34 DE xx
```

How to get MSByte of a?

Little Endian:

```
0x10000000: BC 9A 78 56
0x10000004: 34 12 DE xx
```

How to get MSByte of a?

- Cortex-M4 can be implemented either way
- XMC4500 is little endian – as most embedded processors

Recap: Endianness

- Describes how datatypes larger than one byte are ordered in memory

Little Endian Least significant byte in lowest address (=base address)

Big Endian Most significant byte in lowest address (=base address)

Example

```
uint32_t a=0x56789ABC; uint16_t b=0x1234; uint8_t c=0xDE;
```

Big Endian:

```
0x10000000: 56 78 9A BC
0x10000004: 12 34 DE xx
```

How to get MSByte of a?

```
*((uint8_t *)&a)
```

Little Endian:

```
0x10000000: BC 9A 78 56
0x10000004: 34 12 DE xx
```

How to get MSByte of a?

```
*((uint8_t *)&a+3)
```

- Cortex-M4 can be implemented either way
- XMC4500 is little endian – as most embedded processors

Problem 5.4: Buffer Overflow

The screenshot on the next slide shows GDB while a function vulnerable to a buffer overflow is debugged. It currently handles the string `Hello World!`

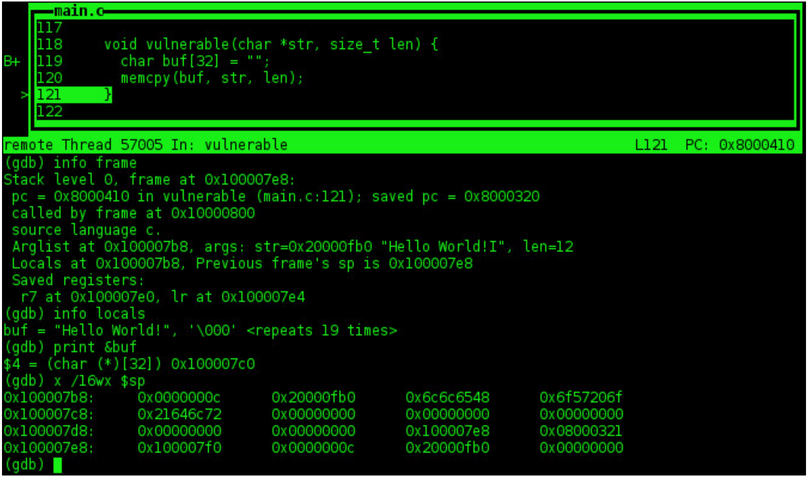
Gather in groups of 3 and answer:

1. Determine where the buffer and the return address are located and the how long your exploit has to be to overwrite the return address
2. Craft an exploit to be sent to the function to trigger a remote code execution. The code to be executed is:

   ```
   fd 46 48 f2 01 12 c4 f6 02 02 80 21 d1 73 c9 09 d1 70 fe e7
   ```

3. The function uses `memcpy` to fill its `buf`. Explain why or why not `strcpy()` would work equally.

Buffer Overflow Stack Excerpt



```
    main.c
   117
   118      void vulnerable(char *str, size_t len) {
B+ 119        char buf[32] = "";
   120        memcpy(buf, str, len);
  >121      }
   122

remote Thread 57005 In: vulnerable                                    L121   PC: 0x8000410
(gdb) info frame
Stack level 0, frame at 0x100007e8:
 pc = 0x8000410 in vulnerable (main.c:121); saved pc = 0x8000320
 called by frame at 0x10000800
 source language c.
 Arglist at 0x100007b8, args: str=0x20000fb0 "Hello World!I", len=12
 Locals at 0x100007b8, Previous frame's sp is 0x100007e8
 Saved registers:
  r7 at 0x100007e0, lr at 0x100007e4
(gdb) info locals
buf = "Hello World!", '\000' <repeats 19 times>
(gdb) print &buf
$4 = (char (*)[32]) 0x100007c0
(gdb) x /16wx $sp
0x100007b8:     0x0000000c      0x20000fb0      0x6c6c6548      0x6f57206f
0x100007c8:     0x21646c72      0x00000000      0x00000000      0x00000000
0x100007d8:     0x00000000      0x00000000      0x100007e8      0x08000321
0x100007e8:     0x100007f0      0x0000000c      0x20000fb0      0x00000000
(gdb)
```

Outline

Prologue

Memory Organisation
- Macroscopic Level: Sections on the XMC4500
- Microscopic Level: Stack Frame Organisation

Memory Vulnerabilities: Size Matters
- Information Leakage by Pretending to Be Longer
- Remote Code Execution by Being Longer Than Expected

Epilogue

Repetition

In which section and in which SRAM
is an uninitialized global variable placed on the XMC4500?

What did the programmer responsible for Heartbleed forgot?

What layout decision for C stack frames
makes buffer overflows so effective?

Repetition

In which section and in which SRAM
is an uninitialized global variable placed on the XMC4500?

Section is device independent the BSS. BSS is located in DSRAM1 on XMC4500.

What did the programmer responsible for Heartbleed forgot?

What layout decision for C stack frames
makes buffer overflows so effective?

Repetition

In which section and in which SRAM
is an uninitialized global variable placed on the XMC4500?

Section is device independent the BSS. BSS is located in DSRAM1 on XMC4500.

What did the programmer responsible for Heartbleed forgot?

To check whether the size reported by the user matches the size of the data sent by the user.

What layout decision for C stack frames
makes buffer overflows so effective?

Repetition

In which section and in which SRAM is an uninitialized global variable placed on the XMC4500?

Section is device independent the BSS. BSS is located in DSRAM1 on XMC4500.

What did the programmer responsible for Heartbleed forgot?

To check whether the size reported by the user matches the size of the data sent by the user.

What layout decision for C stack frames makes buffer overflows so effective?

That the return address lies above local variables. This makes it easily reachable for overwriting via buffer overflows.