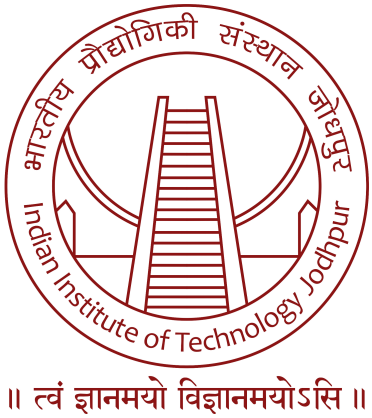


EEL3090: Embedded Systems

Course Project ES5

Exploration of Hardware Security in Embedded Systems

Under the guidance of
Dr. Binod Kumar

By

Shah Priyanshi Udaykant (B21EE062)
Soham Niraj Deshmukh (B21EE067)

Index

Abstract:

Embedded systems are specialized computing systems intended to perform specific tasks in environments that are deemed restricted. They are typically composed of firmware, or software, that governs their operation, and hardware components.

Because memory management controls how data and instructions are stored and accessed during program execution, memory management is a crucial part of embedded systems. Dictionaries, or associative arrays, provide a simple way to arrange and retrieve data that is stored in memory. They are therefore suitable for the data and instruction storage in embedded devices.

This paper's goal is to examine the risks that pointer subterfuge, buffer overflow, code injection, and other techniques bring to embedded systems and to explain the likely consequences of these risks.

Introduction:

Embedded systems are essential constituents of contemporary technology, providing power to devices spanning various sectors. Nevertheless, their dependence on firmware and peripheral interfaces renders them susceptible to code injection assaults thereby potentially jeopardizing the integrity and confidentiality of the system. This report investigates the diverse methodologies employed by malicious actors to implant malevolent code into embedded systems, resulting in unauthorized entry, manipulation of data, or paralysis of the system. By acquiring knowledge regarding the characteristics of these assaults and possible countermeasures, programmers can fortify the security stance of embedded systems and reduce the likelihood of exploitation.

In addition to this, embedded systems are indispensable in an extensive range of technological applications, encompassing industrial control systems and consumer electronics. Nevertheless, the extensive utilization of these devices renders them highly desirable prey for malevolent entities aiming to undermine the integrity and security of systems.

The idea of pointer subterfuge attacks in embedded systems is examined in this report, with a particular emphasis on how attackers change pointers in memory to reroute control flow or obtain unauthorized access. In particular, we study a situation in where data and instructions are stored in memory using dictionaries, which makes it easier to launch a pointer subterfuge attack.

Furthermore, we delve into buffer overflow vulnerabilities, emphasizing the importance of implementing strong security measures in the design of embedded systems. By understanding these threats and potential mitigation strategies, developers can strengthen the security posture of embedded systems and mitigate the risk of exploitation.

Approach

CPU

- We created a CPU capable of performing four instructions: ADD, SUB, MUL, and DIV.
- The CPU is equipped with fully writable memory, with separate memory blocks for instructions and data (operands). The data and instructions are directly accessible via pointers.
- The CPU can connect to and receive input from a variety of peripheral devices, including UART and GPIO, as well as output via UART, GPIO, and a 7 segment decoder.
- We have also included the option of running a specific Assembly instruction directly.
- The CPU's stack pointer, named 'current_state', is also kept updated with the following instructions, as well as JUMPs and BRANCHES.

UART

- The UART is engaged whenever an IRQ (enable interrupt) request is made.
- The UART is responsible for data transmission and receiving and includes a dedicated transmitter and receiver.

Seven Segment Decoder

- The Seven-Segment Display directly displays (prints) data transmitted to it via UART or CPU.
- It can be designed to display more than ten values by modifying the state vector.
- It can also be reset to clear all preset data.

UpCounter

- An UpCounter has also been built to keep track of data characters sent/received via the UART.

Vulnerabilities and Attack Scenarios

Code injection attacks attempt to compromise the integrity of embedded systems by introducing malicious code into memory and altering the program flow to execute it. These attacks can be classified according to the breached security objectives or the method used to conduct the attack.

Common goals of code injection attacks include:

1. Compromising System Integrity: Attackers aim to undermine the integrity of further program code or sensitive data stored within the system.
2. System Paralysis: By injecting malicious code, attackers can disrupt normal system operation, rendering it inoperable or unstable.
3. Unauthorized Access to Sensitive Data: Attackers seek to gain access to sensitive data stored within the system, potentially leading to data breaches or leakage of confidential information.

Means of Launching Attacks:

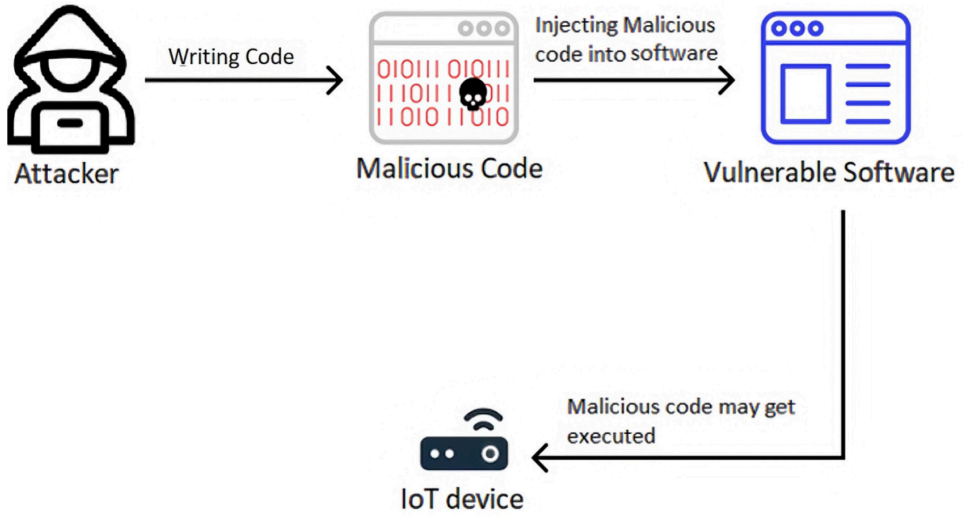
Code injection attacks leverage various techniques to achieve their goals, including:

1. Buffer Overflows: Exploiting vulnerabilities in programs that handle input data, attackers can overflow buffers and overwrite critical memory locations, such as return addresses or function pointers, to hijack the control flow and execute arbitrary code.
2. Heap-Based Attacks: Manipulating dynamically allocated memory blocks, attackers can overwrite heap metadata to redirect control flow and execute malicious code.

3. Return-Oriented Programming (ROP): By chaining together existing code snippets, attackers can construct malicious programs without injecting new code, leveraging the existing instruction sequences in shared libraries.
4. Pointer Subterfuge: Modifying function pointers or data pointers, attackers can redirect the control flow or manipulate memory accesses to execute malicious code or gain unauthorized access.

We have implemented the techniques of *Buffer Overflow Attack* and *Pointer Subterfuge* discussed above to understand their implications firsthand and evaluate their potential impact on embedded systems security.

Through these implementations, we aim to gain practical insights into the vulnerabilities present in embedded systems.



Buffer Overflow Attack

Scenario Overview:

In this, an attacker aims to exploit a buffer overflow vulnerability in a web server application. The web server application processes user input such as HTTP requests but lacks proper input validation and boundary checks, making it susceptible to buffer overflow attacks.

Attack Steps:

1. The attacker crafts a malicious input, such as a long string or specially formatted data, designed to overflow the buffer allocated by the web server application to store user input.
2. The attacker sends the crafted input to the vulnerable web server application, typically through a network connection or by submitting a form on a website.
3. Due to the lack of input validation and boundary checks, the malicious input overflows the buffer allocated by the web server application. The excess data overwrites adjacent memory locations, potentially including critical data such as return addresses or function pointers.
4. By carefully crafting the malicious input, the attacker manipulates the overwritten memory locations to control the program's execution flow. Common targets for control flow hijacking include overwriting the return address of a function or a function pointer stored in memory.
5. With control of the program's execution flow, the attacker can redirect it to execute arbitrary code of their choosing. The attacker typically places shellcode or other malicious payloads in the overwritten memory locations, which are then executed when the program continues execution.

Pointer Subterfuge

Scenario Overview:

Over here we consider an embedded system where dictionaries are used to represent instructions and data in memory. The instruction dictionary stores function names as keys and their corresponding memory addresses or function objects as values, while the data dictionary stores data names and their values. An attacker exploits vulnerabilities in the system to manipulate function pointers stored in the instruction dictionary, redirecting the control flow to execute malicious code.

Attack Steps:

1. The attacker identifies vulnerable function pointers stored in the instruction dictionary.
2. The attacker crafts or identifies malicious code to be executed when the function pointer is manipulated.
3. Through buffer overflow or other memory corruption vulnerabilities, the attacker overwrites a function pointer in the instruction dictionary with the address of the malicious code.
4. The attacker triggers the execution of the manipulated function pointer, causing the system to follow the modified control flow.
5. The malicious code is executed within the context of the compromised application, allowing the attacker to achieve their objectives.

Snapshots of code:

```
def buffer_overflow(cpu):
    last_address = [i for i in cpu.data_memory][-1]
    new_address = hex(int(last_address, 16) + 4)
    cpu.current_address = new_address
```

```
def pointer_subterfuge(cpu):

    all_instructions = cpu.instruction_memory
    all_addresses = [i for i in cpu.data_memory]

    function1_address,function2_address = random.randint(0,len(all_instructions)-1),0
    while(function1_address == function2_address):
        function2_address = random.randint(0, len(all_instructions) - 1)

    temp = all_instructions[function1_address]
    cpu.instruction_memory[function1_address],
cpu.instruction_memory[function2_address] =
cpu.instruction_memory[function2_address], temp

    func1 = all_instructions[function1_address]
    func2 = all_instructions[function2_address]

    print("Function 1: " + func1)
    print("Function 2: " + func2)

    temp = getattr(cpu, func1)
    setattr(cpu, func1, getattr(cpu, func2))
    setattr(cpu, func2, temp)
```

Output

Unattacked Transmission through UART

```
Peripheral <circuits.peripherals.GPIO object at 0x0000017FFBFC47F0> added successfully!


UART Transmission
UART1: UART handling interrupt...
Enter the address of data to be transmitted: 0x08

Waiting for transmission...0
...

Waiting for transmission...1
UART1: Transmitting data...
Transmission complete!
Transmitted data =  5

Initiating 7-segment display...
Added segment 0 to state
Added segment 1 to state
Added segment 2 to state
Added segment 3 to state
Added segment 4 to state
Added segment 5 to state
Added segment 6 to state
Added segment 7 to state
Added segment 8 to state
Added segment 9 to state

%%%%%
%
%%%%%
    %
%%%%%

UART Reception
UART2: Receiving data...
Recieved data =  0x08

Initiating 7-segment display...
Added segment 0 to state
Added segment 1 to state
Added segment 2 to state
Added segment 3 to state
Added segment 4 to state
Added segment 5 to state
Added segment 6 to state
Added segment 7 to state
Added segment 8 to state
Added segment 9 to state

%%%%%
%
%%%%%
    %
%%%%%
PS E:\IIT Jodhpur\Embedded Systems\Course Project>
```

Buffer Overflow Attack

```
Adding peripheral - <circuits.peripherals.UART object at 0x000001896421D9F0>...
Peripheral <circuits.peripherals.UART object at 0x000001896421D9F0> added successfully!

Adding peripheral - <circuits.peripherals.UART object at 0x0000018968424730>...
Peripheral <circuits.peripherals.UART object at 0x0000018968424730> added successfully!

Adding peripheral - <circuits.peripherals.GPIO object at 0x0000018968424850>...
Peripheral <circuits.peripherals.GPIO object at 0x0000018968424850> added successfully!

UART Transmission
UART1: UART handling interrupt...
Enter the address of data to be transmitted: 0x08

Waiting for transmission...0
...

Waiting for transmission...1
UART1: Transmitting data...
Transmission complete!
Transmitted data =  5

Initiating 7-segment display...
Added segment 0 to state
Added segment 1 to state
Added segment 2 to state
Added segment 3 to state
Added segment 4 to state
Added segment 5 to state
Added segment 6 to state
Added segment 7 to state
Added segment 8 to state
Added segment 9 to state

%%%%%
%
%%%%%
    %
%%%%%

Transmitted data address = 0x08
Recieved data address = 0x1f

UART Reception
Traceback (most recent call last):
  File "e:\IIT Jodhpur\Embedded Systems\Course Project\Embedded-Hardware-Security\attack_si
mulation.py", line 51, in <module>
    uart2.receiver(cpu.data_memory[cpu.current_address])
KeyError: '0x1f'
```

Pointer Subterfuge Attack

```
PS E:\IIT Jodhpur\Embedded Systems\Course Project> & C:/Users/91738/anaconda3/envs/dlenv/python.exe "e:/IIT Jodhpur/Embedded Systems/Course Project/Embedded-Hardware-Security/pointer_subterfuge_attack.py"
Adding peripheral - <circuits.peripherals.UART object at 0x000001D09F5AD9F0>...
Peripheral <circuits.peripherals.UART object at 0x000001D09F5AD9F0> added successfully!

Adding peripheral - <circuits.peripherals.UART object at 0x000001D0A3758310>...
Peripheral <circuits.peripherals.UART object at 0x000001D0A3758310> added successfully!

Adding peripheral - <circuits.peripherals.GPIO object at 0x000001D0A3758430>...
Peripheral <circuits.peripherals.GPIO object at 0x000001D0A3758430> added successfully!

Before Attack...

Executing instructions...
add operation completed.
Result: 3 stored in 0x06

Execution completed!

Before attack: ['add', 'sub', 'mul', 'div']
After attack: ['div', 'sub', 'mul', 'add']

After attack

Executing instructions...
add operation completed.
Result: 0.5 stored in 0x06

Execution completed!

PS E:\IIT Jodhpur\Embedded Systems\Course Project>
```

In pointer subterfuge we have implemented a program which randomly changes the pointers of one function call/ data, with that of another function/ data present in the memory. In short, it alters the flow of instructions in an undesirable way and executes irrelevant/malicious instructions.

Discussion:

Embedded hardware systems are essential in diverse industries, including automotive, aerospace, medical devices, and IoT gadgets. However, their unique constraints, including limited resources, real-time requirements, and often mission-critical functionalities, present significant challenges for ensuring security against various vulnerabilities.

The UART (Universal Asynchronous Receiver-Transmitter) peripherals are utilized for data transmission and reception, attackers may exploit vulnerabilities by intercepting or manipulating data mid-communication to execute attacks. This underscores the necessity of considering peripheral vulnerabilities in the overall security strategy.

The discussion of vulnerabilities such as buffer overflow and pointer subterfuge highlights the wide array of risks that embedded hardware systems encounter. An adversary may exploit these weaknesses to compromise the integrity of the system, pilfer sensitive information, or disrupt operations; thus, proactive security measures are crucial. A buffer overflow vulnerability in an automotive control system, for instance, could result in vehicle malfunction and endanger the safety of the driver and passengers.

Through these attacks, we understood that mitigating vulnerabilities in embedded hardware systems requires a multi-faceted approach that encompasses both software and hardware-level defenses. Implementing secure coding standards, such as thorough input validation and meticulous boundary checks, is crucial in order to effectively mitigate the risk of buffer overflow attacks. In addition, the inclusion of hardware security measures such as memory protection units (MPUs) and secure boot protocols can strengthen the system's resistance to exploitation.

Balancing security requirements with the constraints of embedded systems, such as limited processing power, memory, and energy consumption, presents challenges for developers. Security measures must be meticulously customized to address the unique requirements of each embedded program while maintaining optimal performance and functionality.

Integration of the secure development lifecycle is imperative for embedded hardware systems commencing with the design phase and continuing through deployment and maintenance. Threat modeling, risk assessment, secure coding practices, rigorous testing, and ongoing security updates are all components of this strategy in order to address newly discovered vulnerabilities and threats.

Conclusion:

Code injection attacks pose substantial risks to the security of embedded systems as they undermine the availability, confidentiality, and integrity of the system. Through gaining insight into the intricacies of these assaults and implementing suitable countermeasures, developers have the ability to fortify the security stance of embedded systems and reduce the likelihood of malicious code exploiting them.

In addition, pointer subterfuge attacks pose significant risks to the security and integrity of embedded systems. By understanding the attack vectors and employing appropriate mitigation strategies, developers and system administrators can enhance the security posture of embedded systems and mitigate the risk of exploitation by malicious actors.

Furthermore, buffer overflow techniques present formidable challenges to the security of embedded systems. Exploiting these vulnerabilities can lead to unauthorized access, data manipulation, or system paralysis, posing significant risks to the overall integrity and functionality of embedded devices.

In conclusion, securing embedded hardware systems requires a holistic approach that addresses the unique challenges and constraints of the embedded environment. By implementing robust security measures, adhering to best practices, and staying vigilant against emerging threats, developers can enhance the resilience and reliability of embedded hardware systems in the face of evolving cybersecurity risks.

References:

1. https://www.crysys.hu/publications/files/setit/cpaper_bme_PappMB15pst.pdf
2. https://www.researchgate.net/publication/308831768_Embedded_Systems_Security
3. https://www.researchgate.net/publication/224673204_Buffer-Overflow_Protection_The_Theory
4. https://www.cs.ucdavis.edu/~su/publications/pldi07.pdf
5. https://www.cs.purdue.edu/homes/xyzhang/fall07/Papers/delta-debugging.pdf
6. https://www.researchgate.net/publication/2320309_A_First_Step_Towards_Automated_Detection_of_Buffer_Overrun_Vulnerabilities
7. https://www.mdpi.com/1424-8220/23/13/6067