CyberShield
Intrusion Tolerant Flight Computing Through
Hardware Obfuscation

CySER Summer Workshop
May 26, 2023

Chris Major
Research Assistant, Electrical and Computer Engineering
Types of Cybersecurity Attacks

6 Common Types of Cyber Attacks

01. **Malware**
   - Software programs designed to damage or do unwanted actions on a computer. Common examples include viruses, worms, trojan horses, spyware, and ransomware.

02. **Phishing**
   - Attacks sent via email and ask users to click on a link and enter their personal data. They include a link that directs the user to a dummy site that will steal a user’s information.

03. **Password Attacks**
   - Involves a third party trying to gain access to your systems by isolating a user’s password.

04. **Denial of Service Attacks**
   - Attackers send high volumes of data or traffic through the network until the network becomes overloaded and can no longer function.

05. **Man-in-the-Middle (MITM)**
   - Information is obtained from the end user and the entity the user is communicating with by impersonating the endpoints in an online information exchange (i.e., connection from smartphone to website).

06. **Drive-by Downloads**
   - A program is downloaded to a user’s system just by visiting the site. It doesn’t require any type of action by the user to download.
Types of Cybersecurity Attacks

Inserting malicious code into the computer’s program memory and tricking the processor into executing it.
The Malware Challenge

- The nation’s cyber infrastructure consists of a massive number of identical computer systems.
- This homogeneity is advantageous because a single piece of software can be deployed across millions of systems to increase capacity.
However, this gives an attacker a significant advantage in terms of effort relative to system defenders by re-using their attack across numerous systems.
The Embedded Advantage

Personal Computers

400M sold in 2018.
The Embedded Advantage

Personal Computers
400M sold in 2018.

Smart Phones
1.5B sold in 2018.
The Embedded Advantage

Personal Computers
400M sold in 2018.

Smart Phones
1.5B sold in 2018.

Embedded Computers
Our Approach

● Our focus is on diversifying **embedded computers** (not infrastructure).

● Embedded systems have ...
  - Smaller physical dimensions (sometimes 8-pin packages)
  - Lower Clock Frequencies (1MHz - 16MHz)
  - Smaller memories (256k to 1M)
  - Dedicated software, not general-purpose
  - Often no OS other than real-time scheduler

● Embedded systems are often **homogenous processor families**.
Removing The Advantage

If *homogeneity* gives the attacker an advantage, **diversify the network** and **randomize the hardware**.
Diversifying Hardware

Most computer hardware is **fixed** and takes months/years to fabricate.

There has been some prior work in the area of randomization of instructions sets in **Virtual Machines**, with promising results.
Field Programmable Gate Arrays (FPGAs)

• FPGAs are digital logic devices that can be configured into any computational architecture.
  - Logic Matrix
  - Reconfigurable
• FPGAs can take advantage of parallelism and redundancy for hardware acceleration.
• Commercial availability and extensive development support make FPGAs easy to access and implement.
Field Programmable Gate Arrays (FPGAs)

- FPGA hardware is designed using a **Hardware Description Language (HDL)**.
- Once we have a design in an HDL, we can use scripts to create versions of it with alterations.
- The HDL can be created at **compile-time**.
Field Programmable Gate Arrays

- HDL is generated into **Real-Time Logic (RTL)**
- RTL is generated through the FPGA vendor’s **synthesis and implementation suite**.
- A bitstream is generated that can translate the RTL into logic block placement within the FPGA.
Heterogenous Cores

HDL Generation Scripts

Synthesis / Implementation
Heterogenous Cores

Baseline Computer
- Original Processor
- Open-Source Doc
- Known Opcodes
- Compiler Supported

HDL Generation Scripts

Synthesis / Implementation
Heterogenous Cores

Baseline Computer
- Original Processor
- Open-Source Doc
- Known Opcodes
- Compiler Supported

HDL Generation Scripts

Synthesis / Implementation

We can create copies of the baseline computer with different instruction opcodes before synthesis.
Heterogenous Cores

We can create copies of the baseline computer with different instruction opcodes before synthesis.

This results in “functionally equivalent, heterogeneous cores” on the FPGA that run as a redundant system.
A Malware attack will insert execution binaries into each of the 3x cores’ program memory on the FPGA.
Heterogenous Cores

A Malware attack will insert execution binaries into each of the 3x cores’ program memory on the FPGA.

But since the attacker compiled the malware for the publicly-available Baseline computer’s opcodes, it is the only one that executes the malware.
A Malware attack will insert execution binaries into each of the 3x cores’ program memory on the FPGA.

The computers with randomized opcodes don’t recognize the malware.

But since the attacker compiled the malware for the publicly-available Baseline computer’s opcodes, it is the only one that executes the malware.

We can either throw an exception or run a pre-defined routine to remove the malware.
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
- But what if the attacker guesses our instruction codes?
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
- But what if the attacker guesses our instruction codes?
- What if we had **TWO** sets of different instruction code assignments?
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
- But what if the attacker guesses our instruction codes?
- What if we had **TWO** sets of different instruction code assignments?
- Then we could compare opcodes between the computers and if they are ever the SAME, it is malware.
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
- But what if the attacker guesses our instruction codes?
- What if we had **TWO** sets of different instruction code assignments?
- Then we could compare opcodes between the computers and if they are ever the SAME, it is malware.
The CyberShield Concept

- Compile-time creation of obfuscated computing hardware.
- But what if the attacker guesses our instruction codes?
- What if we had **TWO** sets of different instruction code assignments?
- Then we could compare opcodes between the computers and if they are ever the SAME, it is malware.
- We could add **MULTIPLE** sets of instruction codes to **MULTIPLE** computers.
The CyberShield Concept

- Why add more than two computing cores?
- That’s the flight computing component.
The CyberShield Concept

- Why add more than two computing cores?
- That’s the flight computing component.
The CyberShield Concept

- Why add more than two computing cores?
- That’s the flight computing component.
Spaceflight Computing

MSU has a history of building and deploying space computers.
RadPC combines reconfigurable logic and redundancy mechanisms to protect program execution from radiation faults.
RadPC in Space
Timeline of RadPC missions and demonstrations
RadPC – Flight Heritage

- RadPC has extensive mission history and **flight heritage**.
  - 8 high altitude balloons
  - 2 sounding rockets
  - 3 International Space Station missions
  - 2 Cubesat satellites
  - 1 upcoming lunar mission
- Flight heritage determines the aerospace/defense industry’s **confidence in implementing hardware and software**.
What is the connection between space radiation and malware?
Fault Resilience

- What is the connection between space radiation and malware?
  - Unexpected error injection
  - Manipulation of program data
  - Mitigation through redundancy
CyberShield + RadPC

- How do we leverage RadPC’s fault recovery mechanisms?
CyberShield + RadPC

• How do we leverage RadPC’s fault recovery mechanisms?
  - Use the **Quad Modular Redundancy (QMR)** standard from RadPC to create 4 cores.
  - Ensure each core only understands its own opcodes.
  - Add a **voting mechanism** to determine which core has been attacked.
  - Add a **recovery mechanism** to refresh the faulted core in hardware.
Functionally Equivalent Systems
“MSP430 vs. CyberShield”

Both running closed-loop control code to keep missile upright and accepted setpoint angles over UART.

UART for angle setpoint (47° or 61° or 79°).

Stepper motor for control

Angle sensor
Within The Memory Map ...

Program Vulnerabilities
(Classic Buffer Overflow Attack)

Data Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>x2000</td>
<td>decode_array</td>
</tr>
<tr>
<td></td>
<td>frequency</td>
</tr>
<tr>
<td></td>
<td>set_angle</td>
</tr>
<tr>
<td></td>
<td>rx_index</td>
</tr>
<tr>
<td></td>
<td>RXBUF</td>
</tr>
<tr>
<td></td>
<td>index</td>
</tr>
</tbody>
</table>

Global Variables

Stack

ISR Return Addresses

Local Variables

x3000

while(1){
    for(index=x0FFFF;index>0;index--){
        _NOP();
        temp = RXBUF[0];
        if(temp == '1'){
            set_angle = 47;
        }else if (temp == '2'){
            set_angle = 79;
        }else{
            set_angle = 61;
        }
        if(rx_index == 1){
            rx_index=0;
        }
        temp = decode_array[PIIN];
        if(temp<set_angle){
            //enable stepper motor
            P2OUT |= (BIT1); //set direction
        }else if (temp=set_angle){
            P2OUT &=~(BIT2); //enable stepper motor
            P2OUT |=~(BIT5); //set direction
        }else{
            //disable stepper motor
            temp = set_angle;
        }else{
            P2OUT |=BIT2; //disable stepper motor
        }
        frequency = 4000 - 63*(temp);
    }

    //Service UART
    #pragma vector = TIMERO0_VECTOR
    #pragma vector = EUSCI_A1_VECTOR
    _interrupt void Timer_ISR()
    TB0CC0=+frequency;
    P2OUT |=BIT4; //frequency=1;
    TB0CTLO &=~ PCIFG;
    //
}

//Service UART
#pragma vector = EUSCI_A1_VECTOR
_interrupt void ISR_EUSCI_A1( void ) {
    RXBUF[rx_index++] = UC0RXBUF;
    UC1IFG &= ~UC1RXIFG;
}
Within The Memory Map ...

Program Vulnerabilities
(Classic Buffer Overflow Attack)

1. When user sends new setpoint over UART, an IRQ triggers, stacks return address, and retrieves new value for RXBUF.
Within The Memory Map ...

Program Vulnerabilities (Classic Buffer Overflow Attack)

1. When user sends new setpoint over UART, an IRQ triggers, stacks return address, and retrieves new value for RXBUF.

2. But the developer introduced a vulnerability by adding a delay loop in the main program to allow the UART to complete before resetting the input buffer size back to 0.
Within The Memory Map ...

Program Vulnerabilities (Classic Buffer Overflow Attack)

1. When user sends new setpoint over UART, an IRQ triggers, stacks return address, and retrieves new value for RXBUF.

2. But the developer introduced a vulnerability by adding a delay loop in the main program to allow the UART to complete before resetting the input buffer size back to 0.

3. This allows the attacker to stream in malicious code and replace the correct ISR return address.

Data Memory

Address

x2000

x3000

Global Variables

Stack

New ISR Return Addr

ISR Return Addresses

Bad Set Angle

set_angle

NOP

NOP

NOP

Malware

Malware

Malware

Malware

NOP

NOP

NOP

New ISR Return Addr

ISR Return Addresses
1. When user sends new setpoint over UART, an IRQ triggers, stacks return address, and retrieves new value for RXBUF.

2. But the developer introduced a vulnerability by adding a delay loop in the main program to allow the UART to complete before resetting the input buffer size back to 0.

3. This allows the attacker to stream in malicious code and replace the correct ISR return address.

Program Vulnerabilities
(Classic Buffer Overflow Attack)
Redundancy Advantage

- Each CPU's memory has the **exact same malware opcodes** ...

<table>
<thead>
<tr>
<th>CPU 00</th>
<th>CPU 01</th>
<th>CPU 02</th>
<th>CPU 03</th>
</tr>
</thead>
<tbody>
<tr>
<td>decode_array</td>
<td>decode_array</td>
<td>decode_array</td>
<td>decode_array</td>
</tr>
<tr>
<td>frequency</td>
<td>frequency</td>
<td>frequency</td>
<td>frequency</td>
</tr>
<tr>
<td>Bad Set Angle</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>rx_index</td>
<td>rx_index</td>
<td>rx_index</td>
<td>rx_index</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Malware</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Malware</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Malware</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Malware</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>NOP</td>
<td>???</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>New ISR Return Addr</td>
<td>New ISR Return Addr</td>
<td>New ISR Return Addr</td>
<td>New ISR Return Addr</td>
</tr>
</tbody>
</table>
Redundancy Advantage

* ... but only **one core** can actually run the malware.*
Redundancy Advantage

- So what kind of processor core should be used for this strategy?
The MSP430 is a 16-bit microcontroller provided by Texas Instruments.

The opcodes are widely available due to public documentation.

The processor’s functions are proprietary and difficult to replicate without insider access.

No open-source softcore versions for FPGAs exist.
The RISC-V Processor

- RISC-V is an open-source Industry Standard Architecture (ISA).
- Base form is a 32-bit integer-based core (RV32I).
- RISC-V cores can be implemented onto FPGAs with easy access to signals.
- A design must conform to the RISC-V ISA to be considered functional.
Our RISC-V HDL Core

- **RISC-V RV32I ISA**
  - 32-bit registers
  - Integer Operations

- **VHDL Components**

- **4k IMEM, 4k DMEM**

- **2 GPIO, 2 UART, 2 SPI**
Software Development

- The **RV32I compiler** is provided by the RISC-V Collaboration.
- Our custom **Makefile** compiles C into binaries, then exports binaries to VHDL.
- Minimal standard libraries are currently supported (stdint.h) in the CyberShield system.
Image Processing Demonstration
Image Processing Demonstration

• We use the **RISC-V** architecture to process an image from a camera.
• A **UART command** specifies if the screen should show the image or the edge detected version.
• The UART is our **attack vector** for our buffer overflow attack.
• Each processor core has **different opcodes** but the same attack vector.
Demonstration Video
Questions?