CyberShield

An Approach to Defeat Malware in Edge Computers using Hardware Diversity

The Second IFIP Workshop on Intelligent Vehicle Dependability and Security (IVDS)

June 23-26, 2022 – Old Town Alexandria, VA, USA

University Team

Dr. Brock J. LaMeres
Professor, ECE

Dr. Clem Izurieta
Professor, CS

Colter Barney
Grad Student, EE

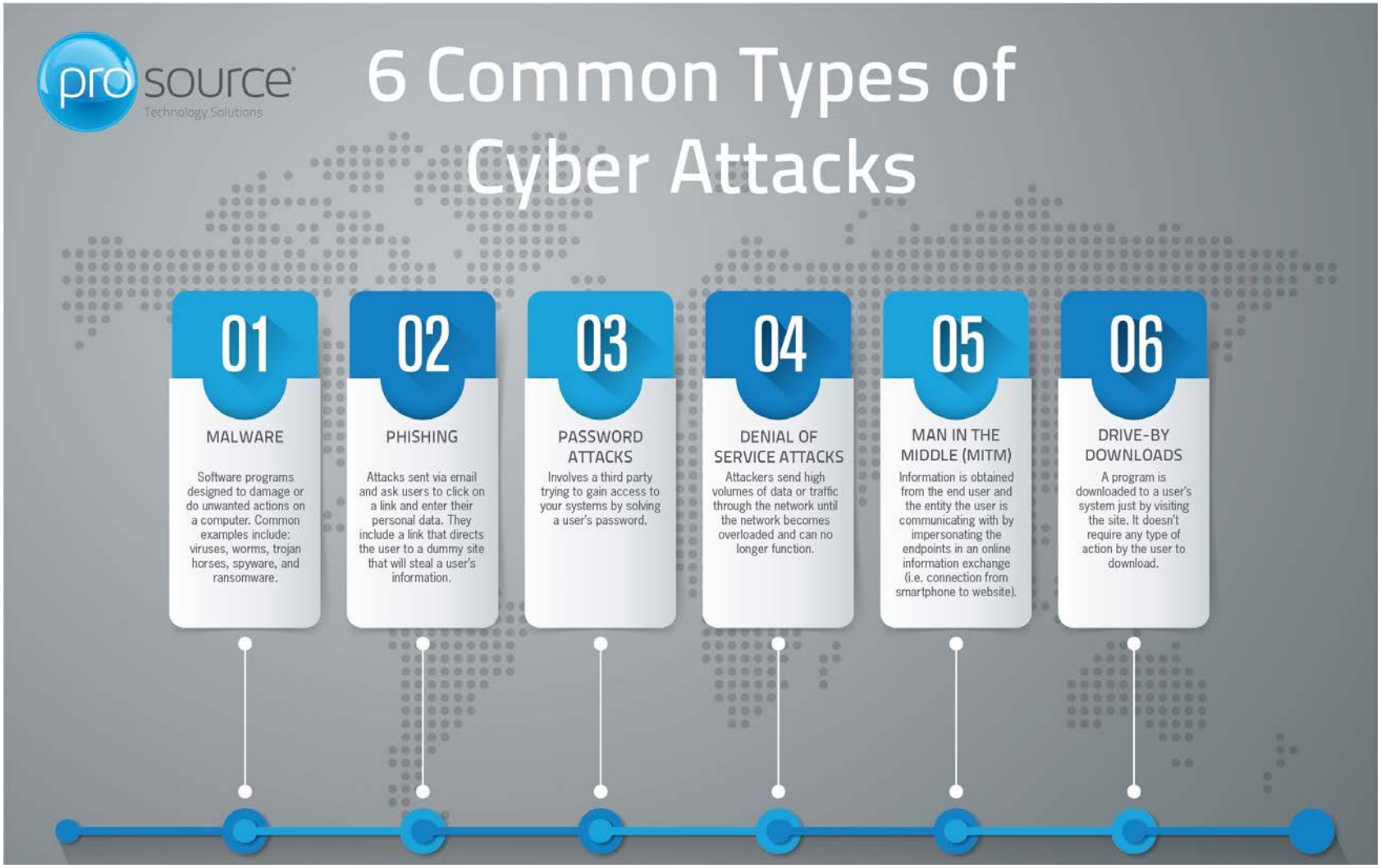
Walker Ward
Undergrad Student, CS

Tristan Running Crane
Grad Student, EE

Industry Mentor

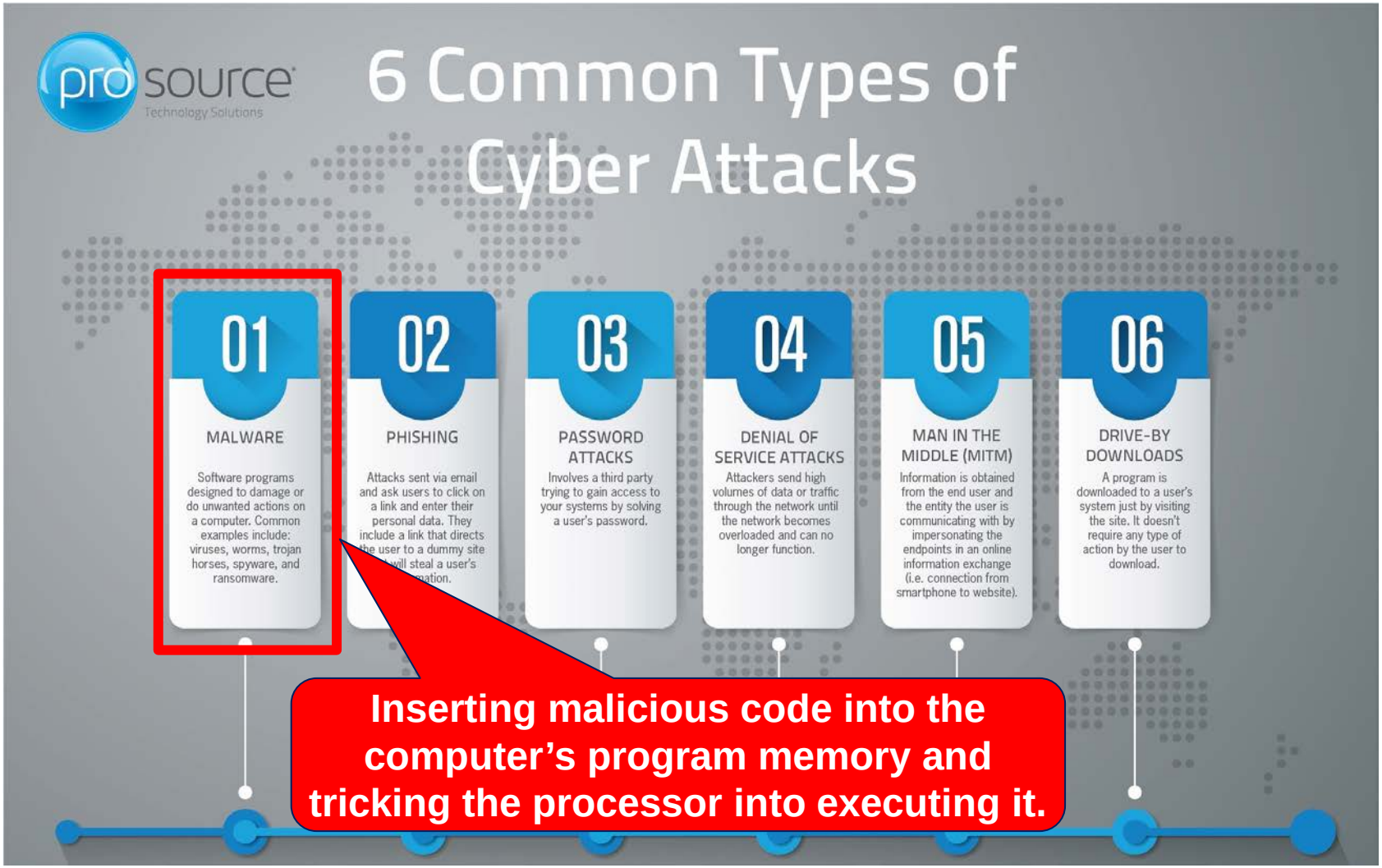
Dr. Jay Lala
Cyber Tech Area Lead
Senior Principal Engineering Fellow

Types of Cybersecurity Attacks



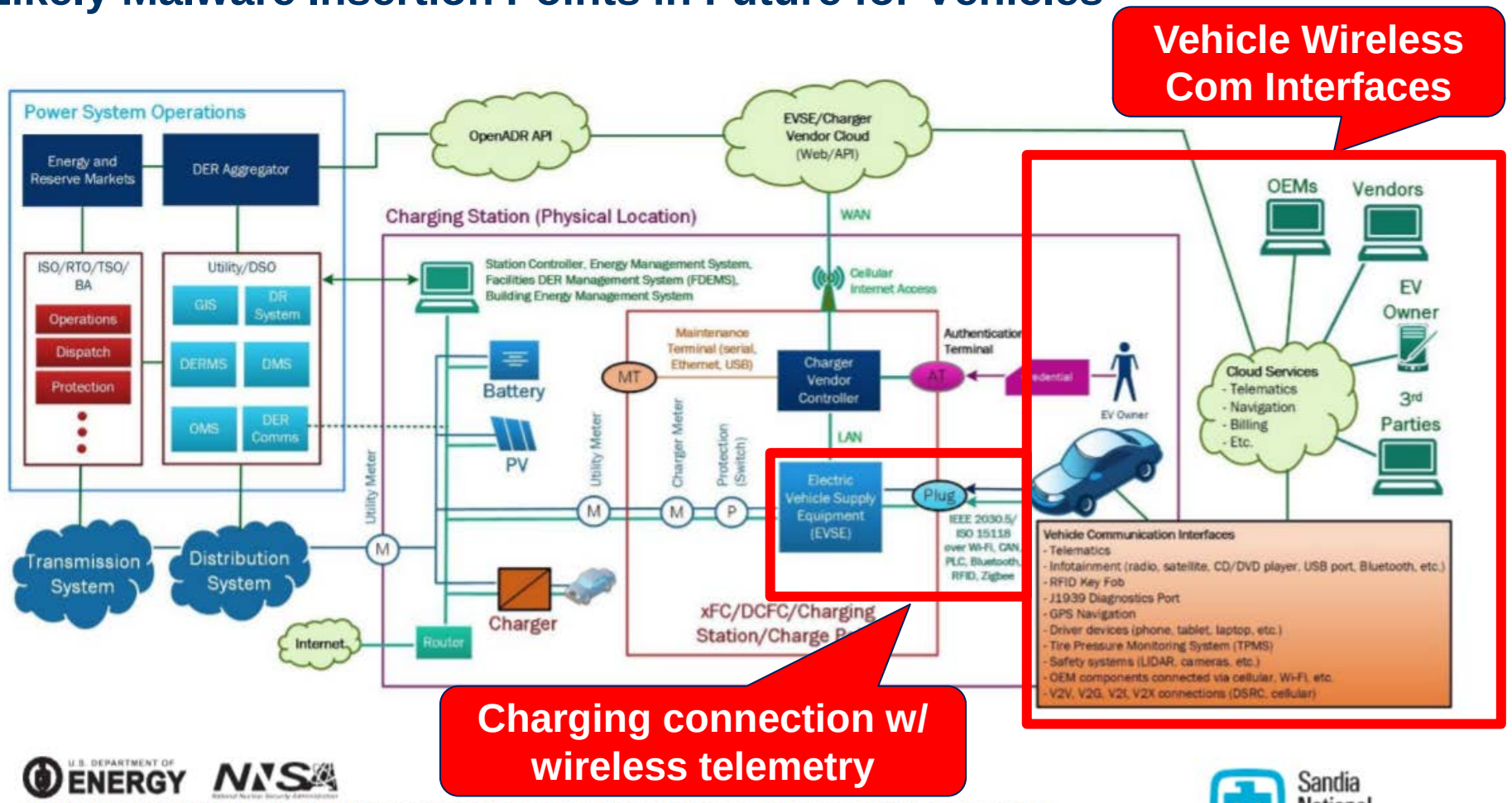
MONTANA STATE UNIVERSITY

Types of Cybersecurity Attacks



MONTANA STATE UNIVERSITY

Likely Malware Insertion Points in Future for Vehicles



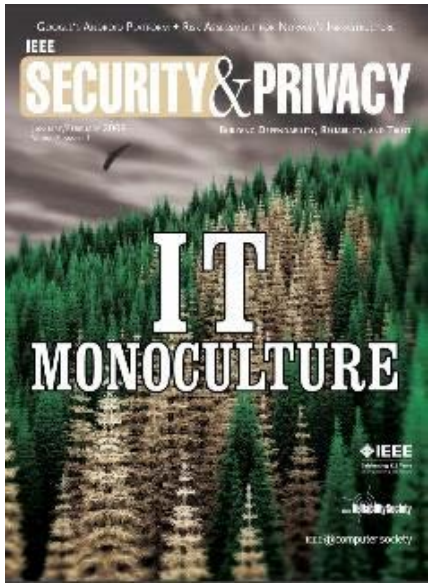
U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2019-6006O

Sandia National Laboratories

MONTANA STATE UNIVERSITY

The Malware Cybersecurity Challenge

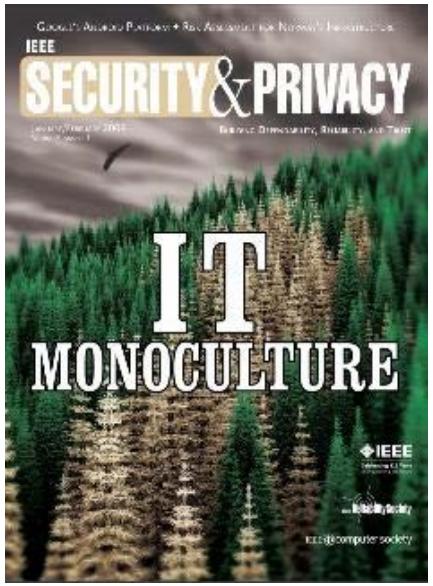


- The nation's cyber infrastructure consists of a massive number of identical computer systems.

- This ***homogeneity*** is advantageous because a single piece of software can be deployed across millions of systems to increase capacity.

MONTANA STATE UNIVERSITY

The Malware Cybersecurity Challenge



- The nation's cyber infrastructure consists
  computer systems.

- This ***homogeneity*** is advantageous beca
  can be deployed across millions of syster

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 Attacker's Advantages Becomes Greater as we Move to Embedded Computing.

Personal Computers

400M sold in 2018.

MONTANA STATE UNIVERSITY

The Attacker's Advantages Becomes Greater as we Move to Embedded Computing.

Personal Computers

400M sold in 2018.

Smart Phones

1.5B sold in 2018.

MONTANA STATE UNIVERSITY

The Attacker's Advantages Becomes Greater as we Move to Embedded Computing.

Personal Computers

400M sold in 2018.

Smart Phones

1.5B sold in 2018.

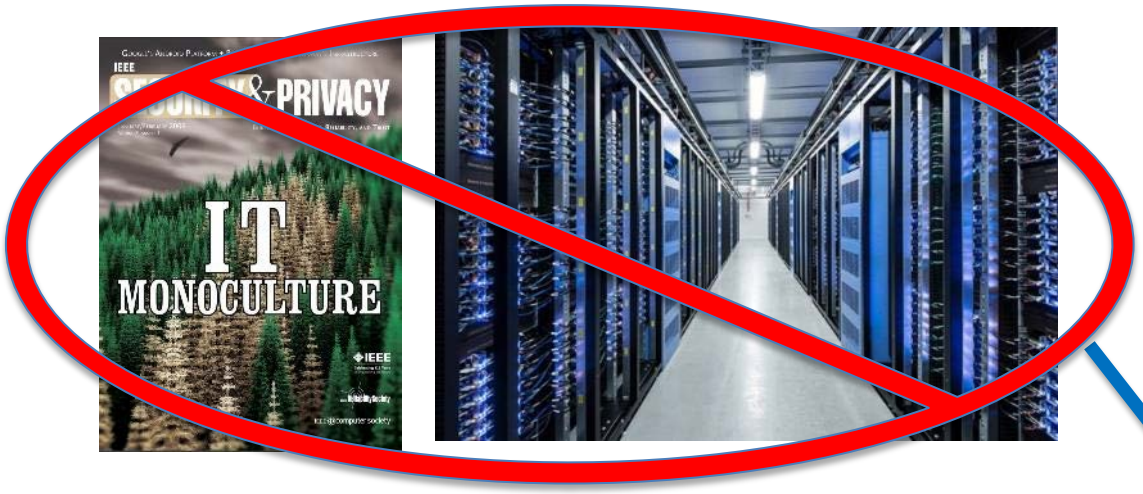
Embedded Computers

25B sold in 2018.

Our Approach

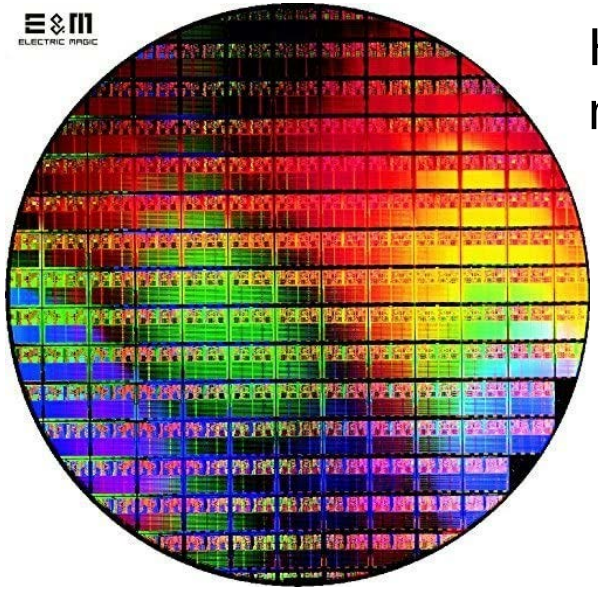
MONTANA STATE UNIVERSITY

If Homogeneity gives the attacker an advantage, let's diversify the network.



Take Away the Attacker's Advantage
by Randomizing the Hardware

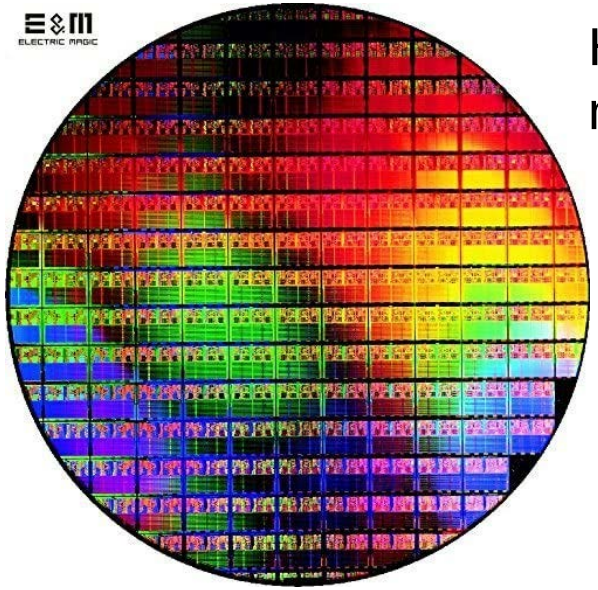
But How Do You Diversify Hardware???



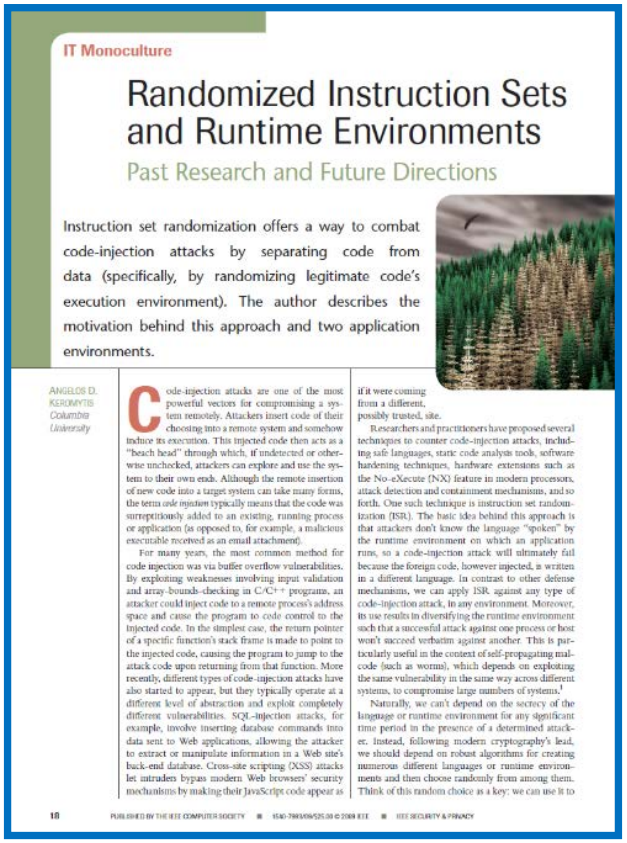
Hardware is fixed and takes months/years to fabricate.

MONTANA STATE UNIVERSITY

But How Do You Diversify Hardware???



Hardware is fixed and takes months/years to fabricate.



IT Monoculture

Randomized Instruction Sets and Runtime Environments

Past Research and Future Directions

Instruction set randomization offers a way to combat code-injection attacks by separating code from data (specifically, by randomizing legitimate code's execution environment). The author describes the motivation behind this approach and two application environments.

ANGELOS D. KEROMYTIS
Columbia University

Code-injection attacks are one of the most powerful vectors for compromising a system remotely. Attackers insert code of their choosing into a remote system and somehow induce its execution. This injected code then acts as a "beach head" through which, if undetected or otherwise unchecked, attackers can explore and use the system to their own ends. Although the remote insertion of new code into a target system can take many forms, the term *code injection* typically means that the code was surreptitiously added to an existing, running process or application (as opposed to, for example, a malicious executable received as an email attachment).

For many years, the most common method for code injection was via buffer overflow vulnerabilities. By exploiting weaknesses involving input validation and array-bounds-checking in C/C++ programs, an attacker could inject code to a remote process's address space and cause the program to cede control to the injected code. In the simplest case, the return pointer of a specific function's stack frame is made to point to the injected code, causing the program to jump to the attack code upon returning from that function. More recently, different types of code-injection attacks have also started to appear, but they typically operate at a different level of abstraction and exploit completely different vulnerabilities. SQL-injection attacks, for example, involve inserting database commands into data sent to Web applications, allowing the attacker to extract or manipulate information in a Web site's back-end database. Cross-site scripting (XSS) attacks let intruders bypass modern Web browsers' security mechanisms by making their JavaScript code appear as if it were coming from a different, possibly trusted, site.

Researchers and practitioners have proposed several techniques to counter code-injection attacks, including safe languages, static code analysis tools, software hardening techniques, hardware extensions such as the No-eXecute (NX) feature in modern processors, attack detection and containment mechanisms, and so forth. One such technique is instruction set randomization (ISR). The basic idea behind this approach is that attackers don't know the language "spoken" by the runtime environment on which an application runs, so a code-injection attack will ultimately fail because the foreign code, however injected, is written in a different language. In contrast to other defense mechanisms, we can apply ISR against any type of code-injection attack, in any environment. Moreover, its use results in diversifying the runtime environment such that a successful attack against one process or host won't succeed verbatim against another. This is particularly useful in the context of self-propagating malcode (such as worms), which depends on exploiting the same vulnerability in the same way across different systems, to compromise large numbers of systems.[1]

Naturally, we can't depend on the secrecy of the language or runtime environment for any significant time period in the presence of a determined attacker. Instead, following modern cryptography's lead, we should depend on robust algorithms for creating numerous different languages or runtime environments and then choose randomly from among them. Think of this random choice as a key: we can use it to

There has been some prior work in the area of randomization of instructions sets in **Virtual Machines**, with promising results.

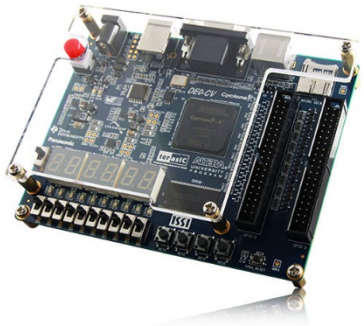
Our Project Focuses on Diversifying Embedded Computers,
not IT Infrastructure *(i.e., Servers)*

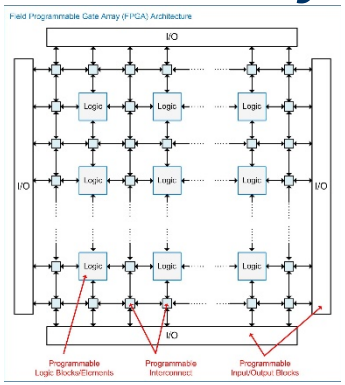
Characteristics of an Embedded Computer

- Smaller (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.

MONTANA STATE UNIVERSITY

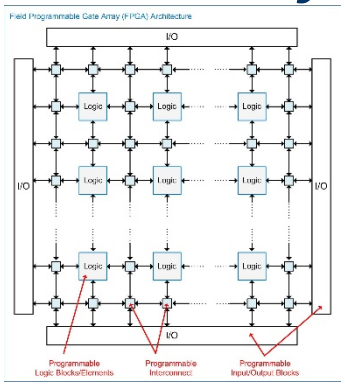
We Exploit the Ability to Implement a Complete Embedded Computer on a *Field Programmable Gate Array* (FPGA)





MONTANA STATE UNIVERSITY

We Exploit the Ability to Implement a Complete Embedded Computer on a *Field Programmable Gate Array* (FPGA)

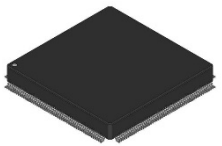


Why is this important?

- *FPGA hardware is designed using a Hardware Description Language (i.e., text).*

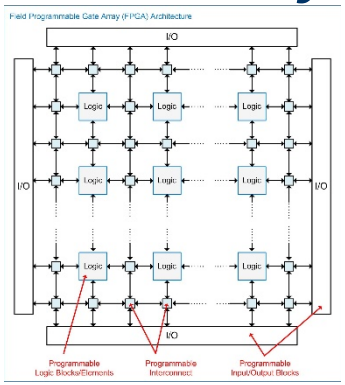
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Synthesis / Implementation



Our Approach

We Exploit the Ability to Implement a Complete Embedded Computer on a *Field Programmable Gate Array* (FPGA)



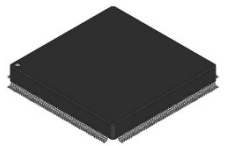
Why is this important?

- *FPGA hardware is designed using a Hardware Description Language (i.e., text).*
- *Once we have a design in an HDL, we can use scripts to create versions of it with alterations.*

HDL Generation Scripts

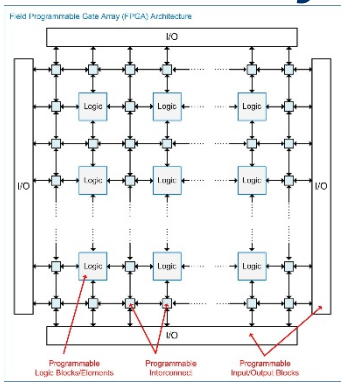
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Synthesis / Implementation



MONTANA STATE UNIVERSITY

We Exploit the Ability to Implement a Complete Embedded Computer on a *Field Programmable Gate Array* (FPGA)



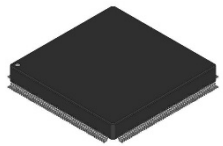
Why is this important?

- *FPGA hardware is designed using a Hardware Description Language (i.e., text).*
- *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.*

HDL Generation Scripts

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

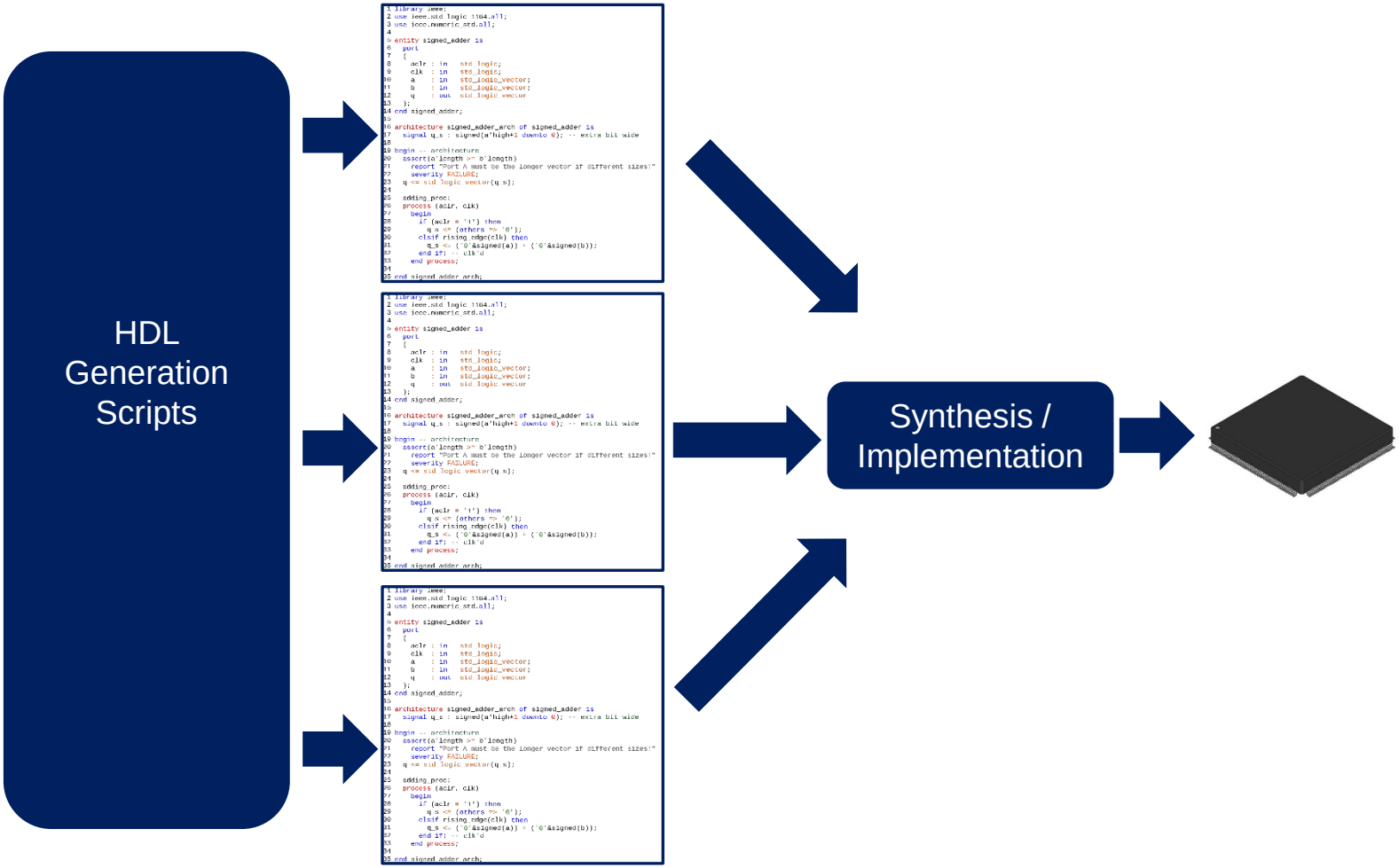
Synthesis / Implementation



Our Approach

MONTANA STATE UNIVERSITY

Once we control the HDL generation, we can make modifications to the design & and even replicate it.



Our Approach

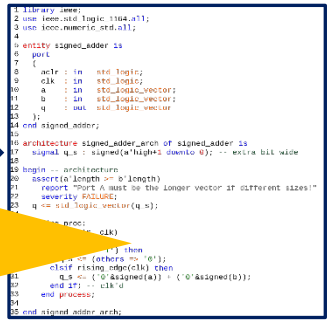
MONTANA STATE UNIVERSITY

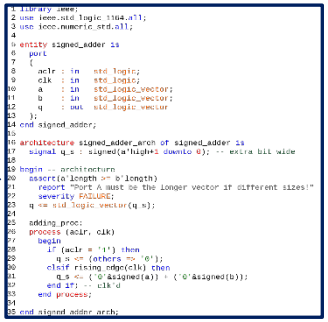
Once we control the HDL generation, we can make modifications to the design & and even replicate it.

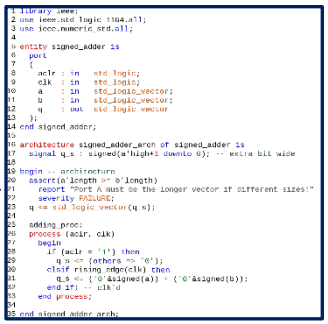
Baseline Computer

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

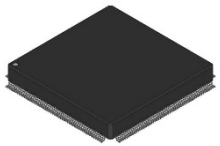
HDL Generation Scripts







Synthesis / Implementation



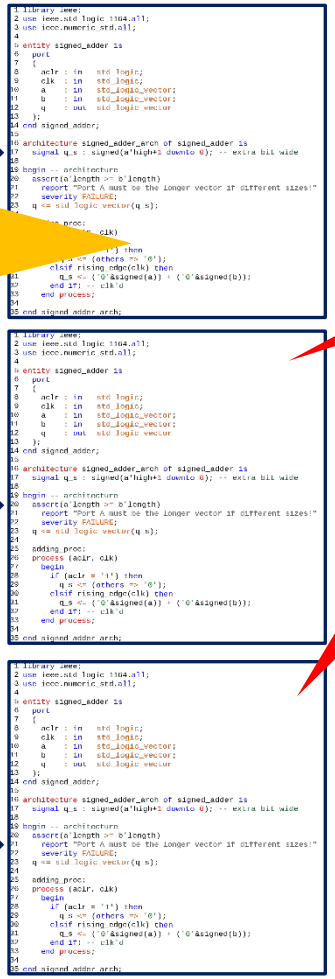
Our Approach

Once we control the HDL generation, we can make modifications to the design & and even replicate it.

Baseline Computer

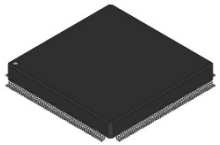
- Original Processor
- Open-Source Doc
- Known Opcodes
- Compiler Supported

HDL Generation Scripts



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

Synthesis / Implementation



Our Approach

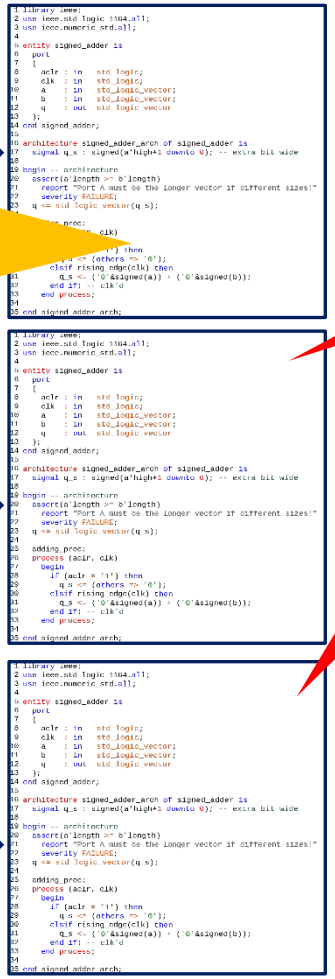
MONTANA STATE UNIVERSITY

Once we control the HDL generation, we can make modifications to the design & and even replicate it.

Baseline Computer

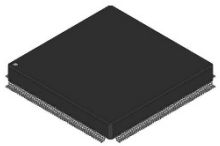
- Original Processor
- Open-Source Doc
- Known Opcodes
- Compiler Supported

HDL Generation Scripts



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

Synthesis / Implementation

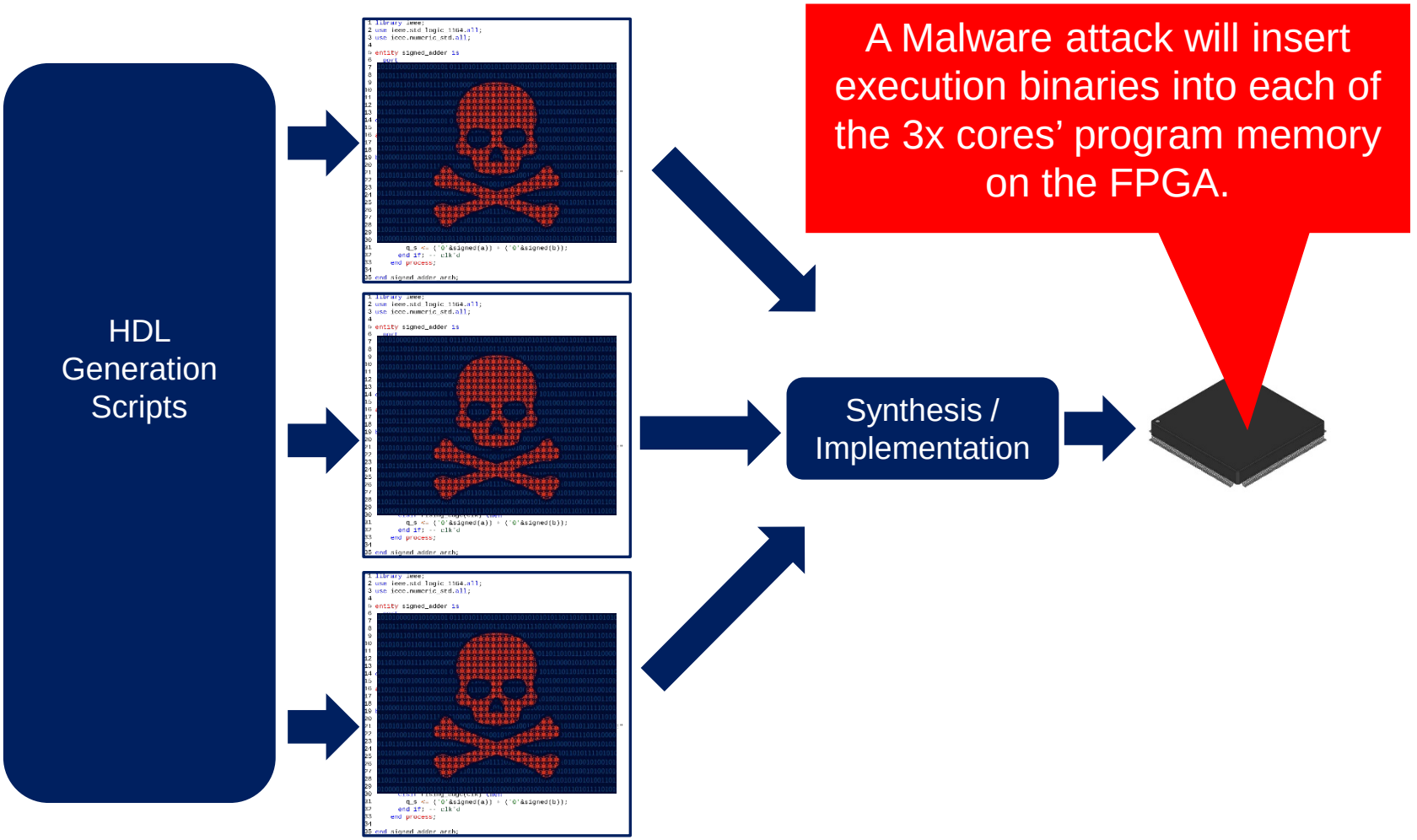


This results in “functionally equivalent, heterogeneous cores” on the FPGA that run as a redundant system.

Our Approach

MONTANA STATE UNIVERSITY

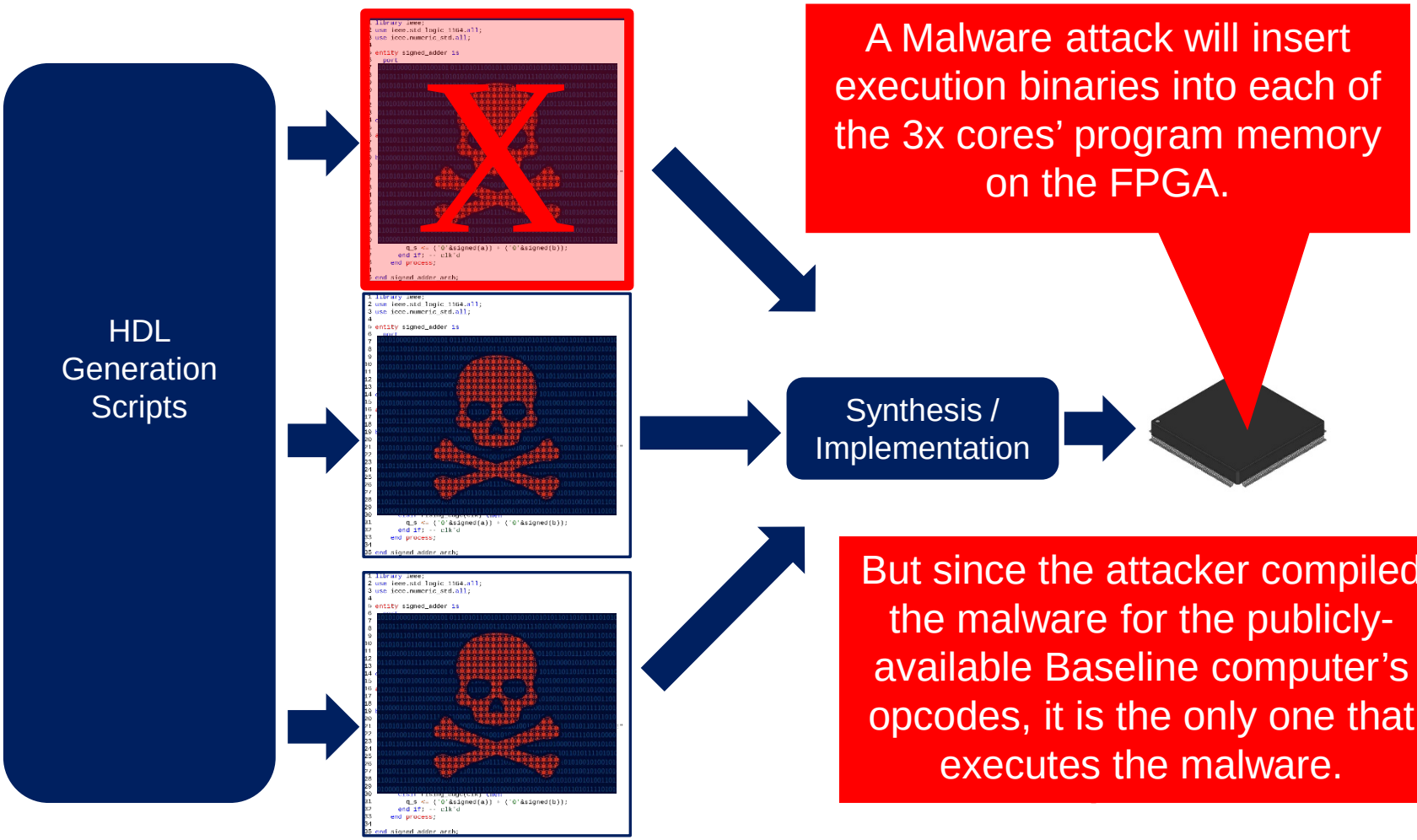
Once we control the HDL generation, we can make modifications to the design & and even replicate it.



Our Approach

MONTANA STATE UNIVERSITY

Once we control the HDL generation, we can make modifications to the design & and even replicate it.



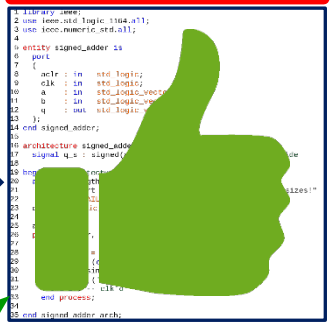
Our Approach

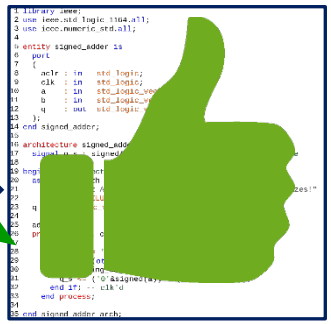
MONTANA STATE UNIVERSITY

Once we control the HDL generation, we can make modifications to the design & and even replicate it.

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

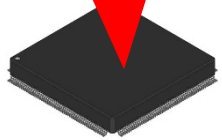
We can either throw an exception or run a pre-defined routine to remove the malware.





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

Synthesis / Implementation

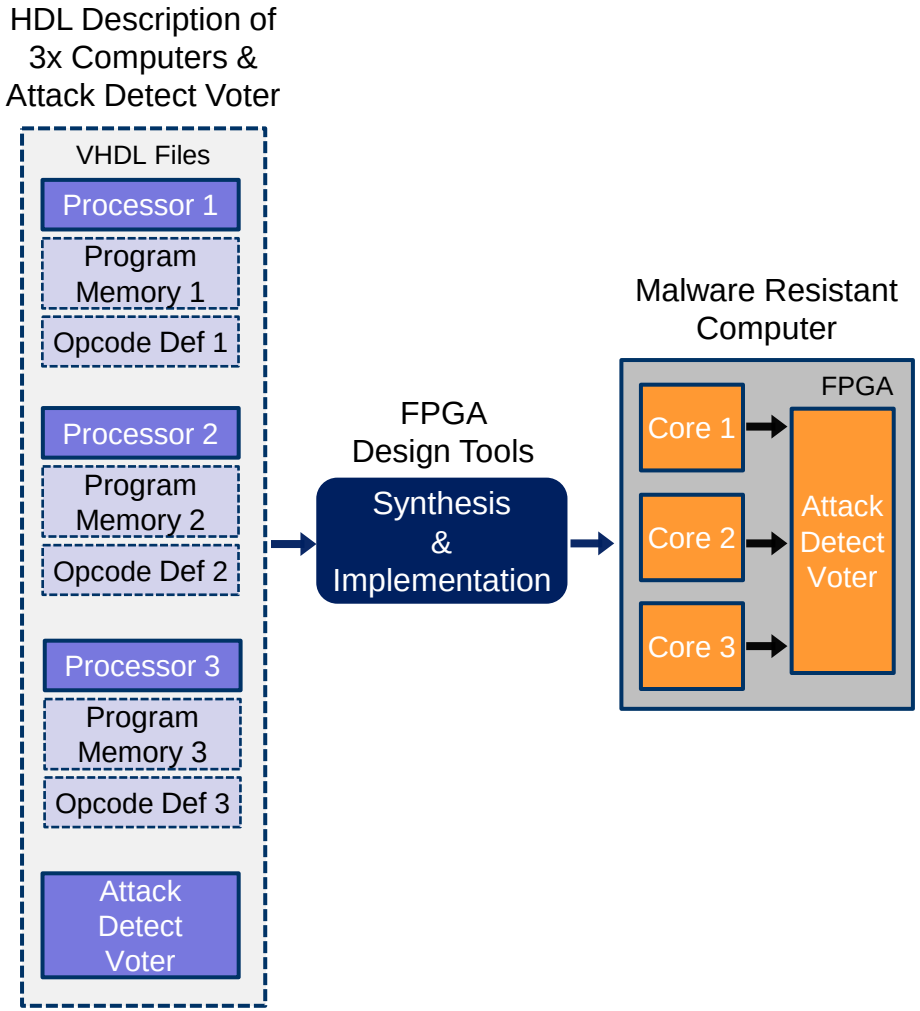


But since the attacker compiled the malware for the publicly-available Baseline computer's opcodes, it is the only one that executes the malware.

But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?

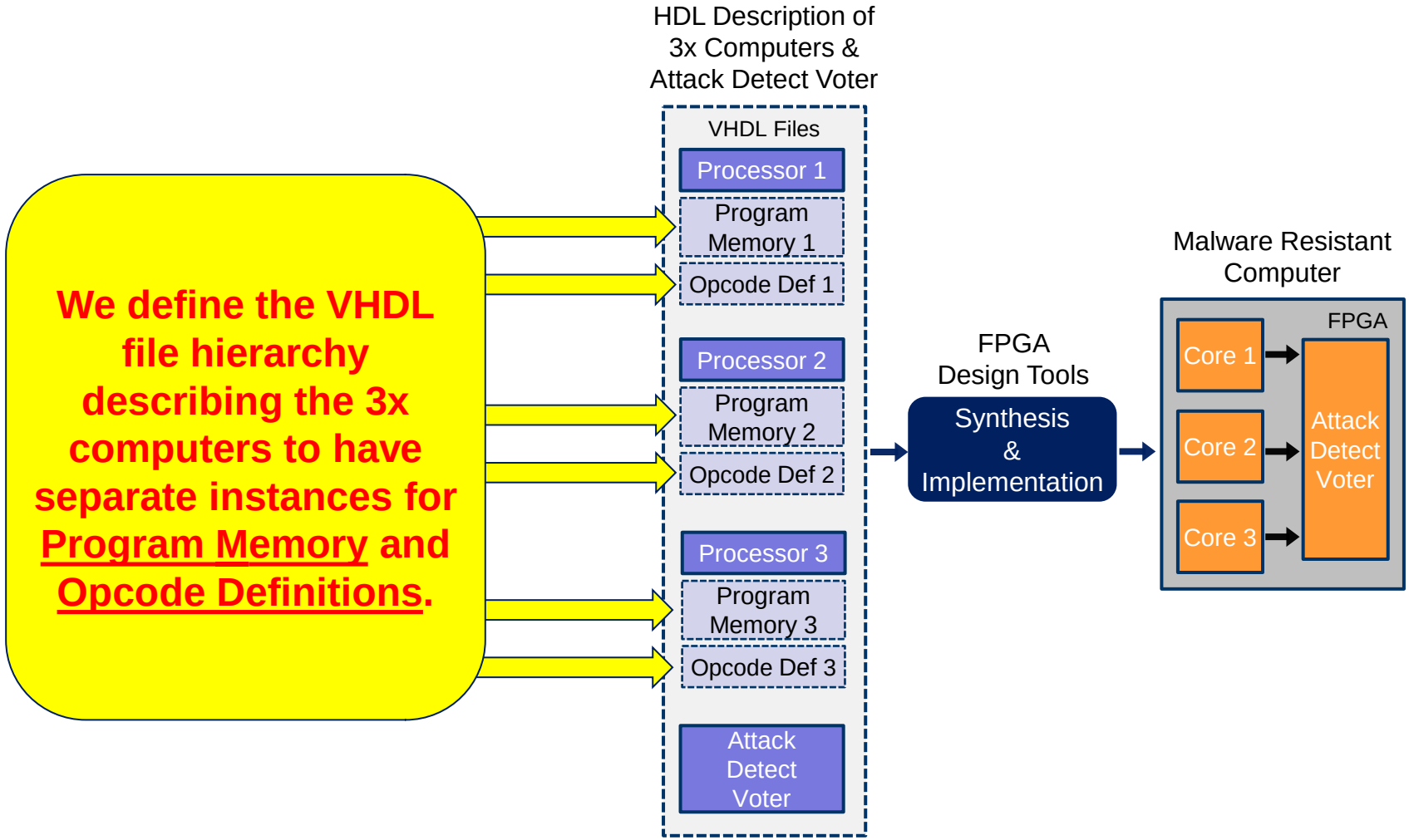
MONTANA STATE UNIVERSITY

But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



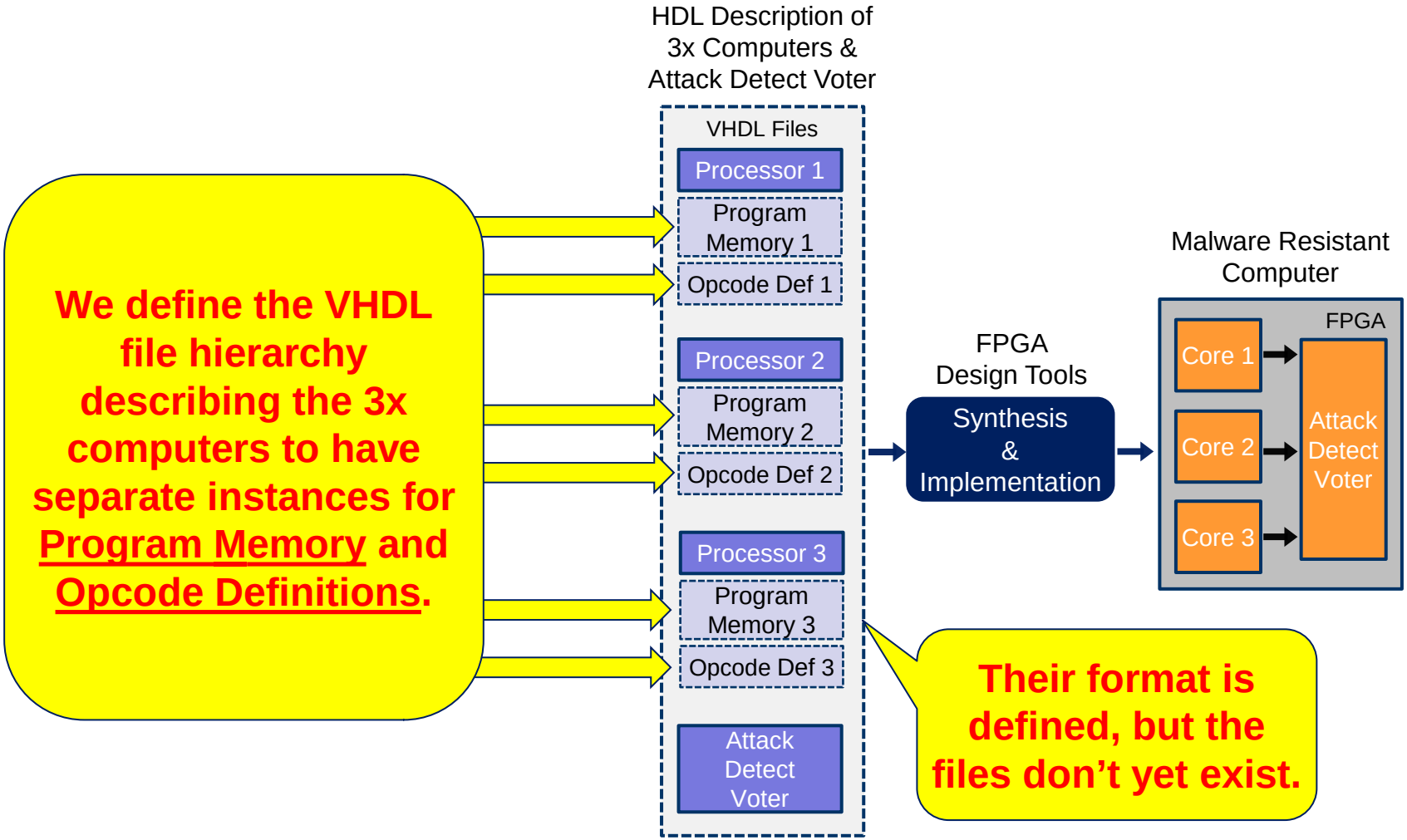
Our Approach

But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



MONTANA STATE UNIVERSITY

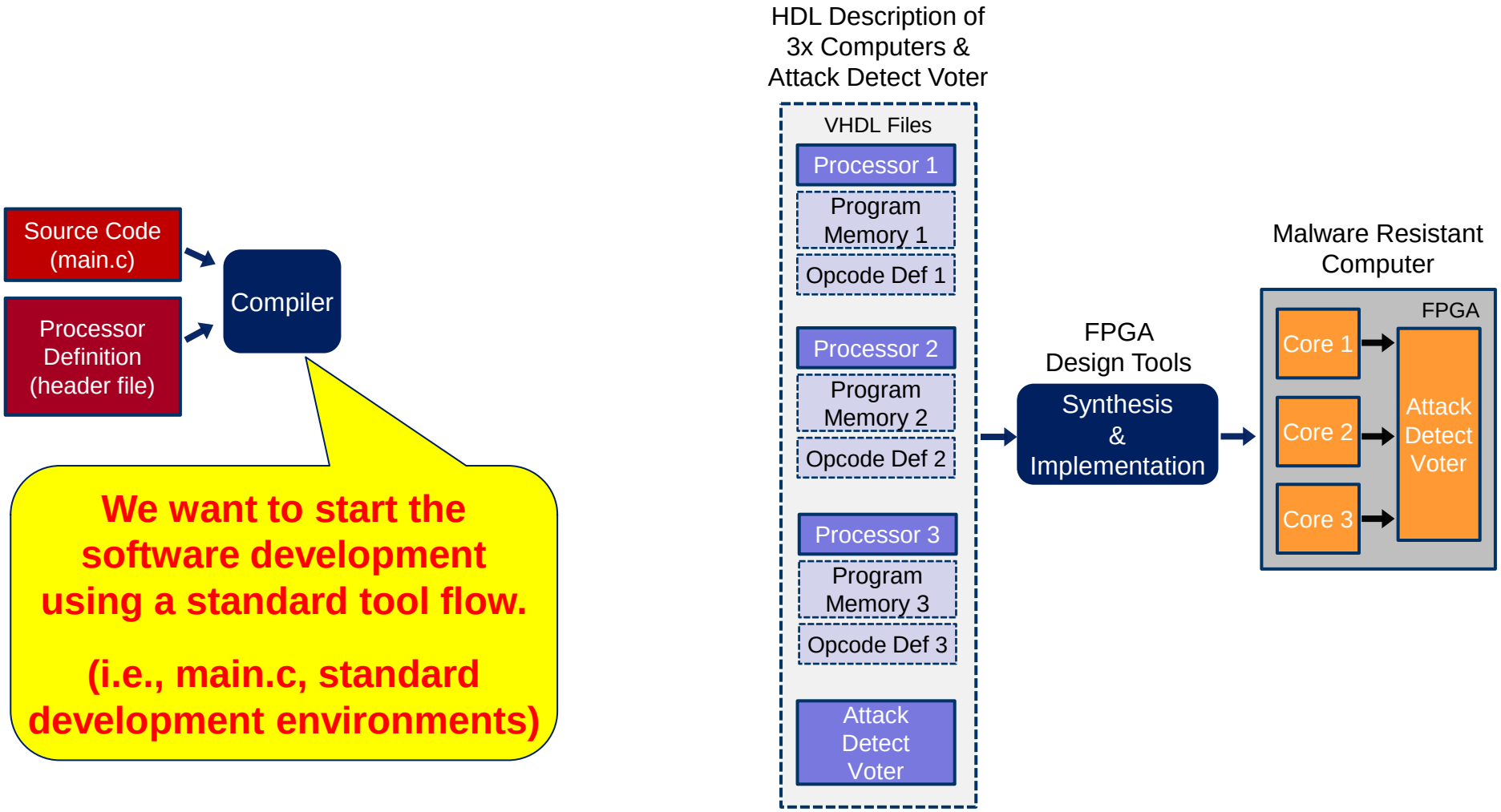
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Our Approach

MONTANA STATE UNIVERSITY

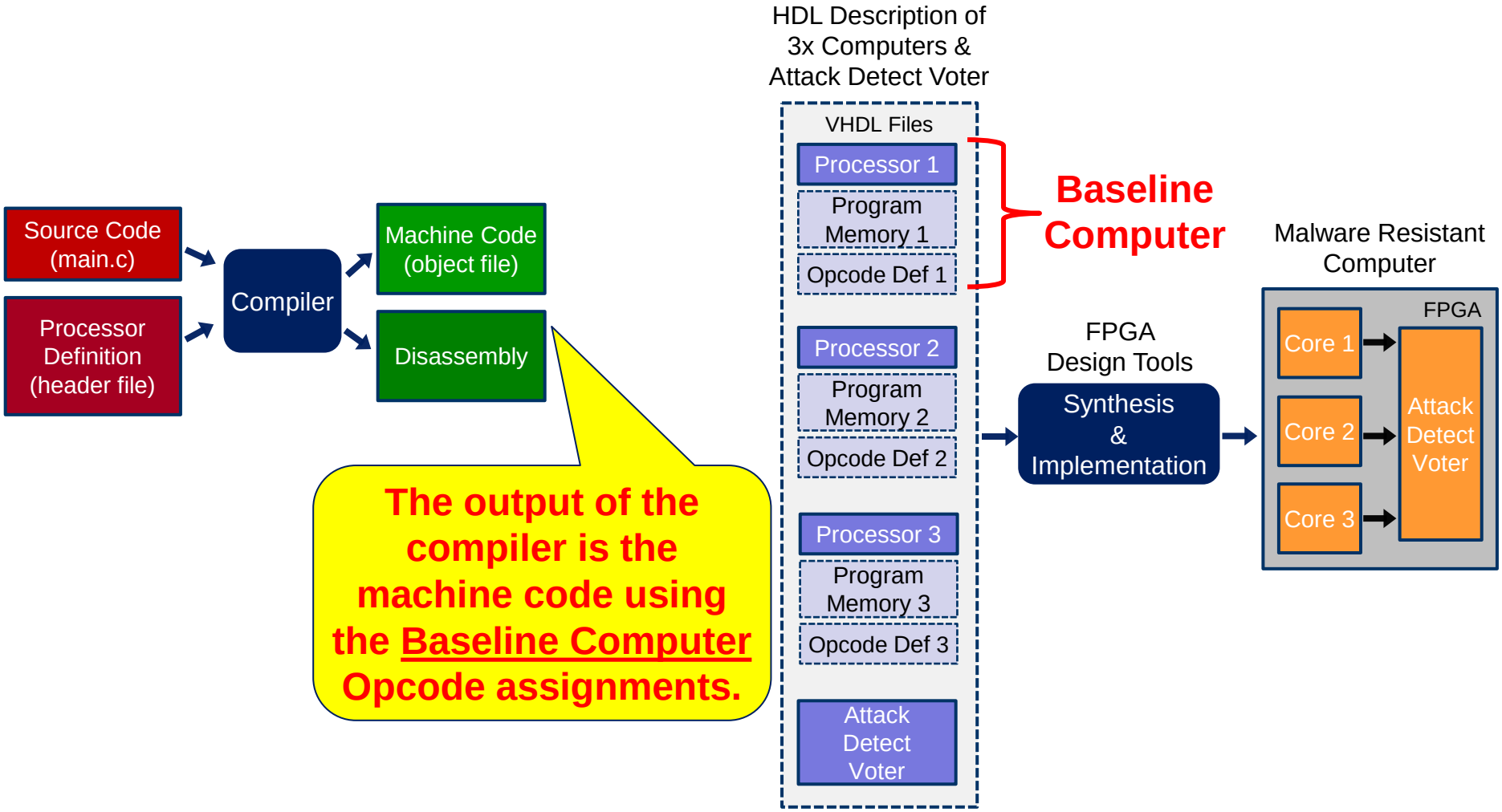
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Our Approach

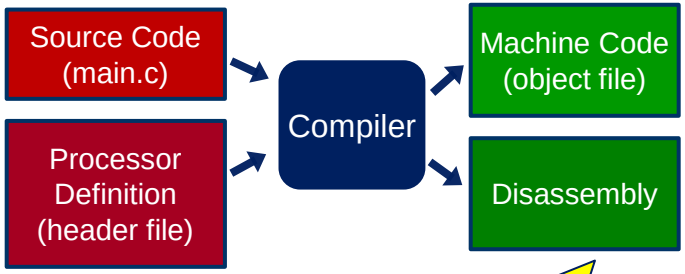
MONTANA STATE UNIVERSITY

But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?

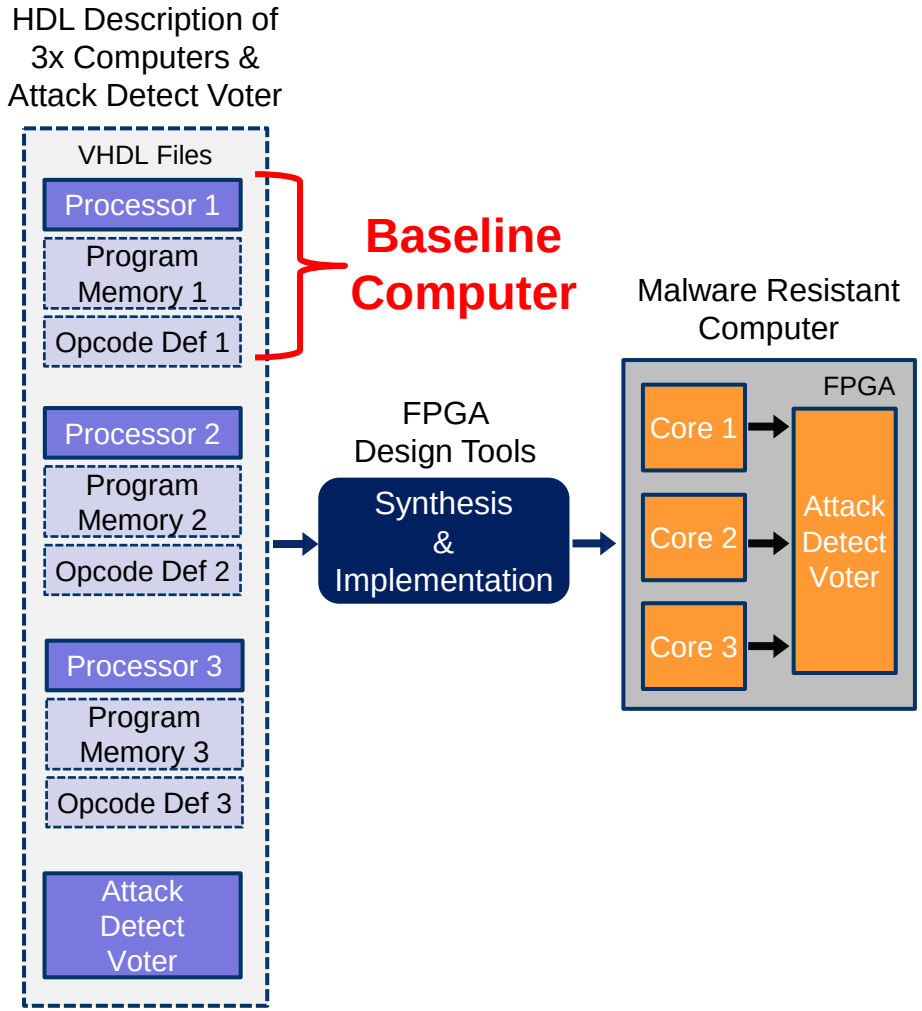


Our Approach

But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



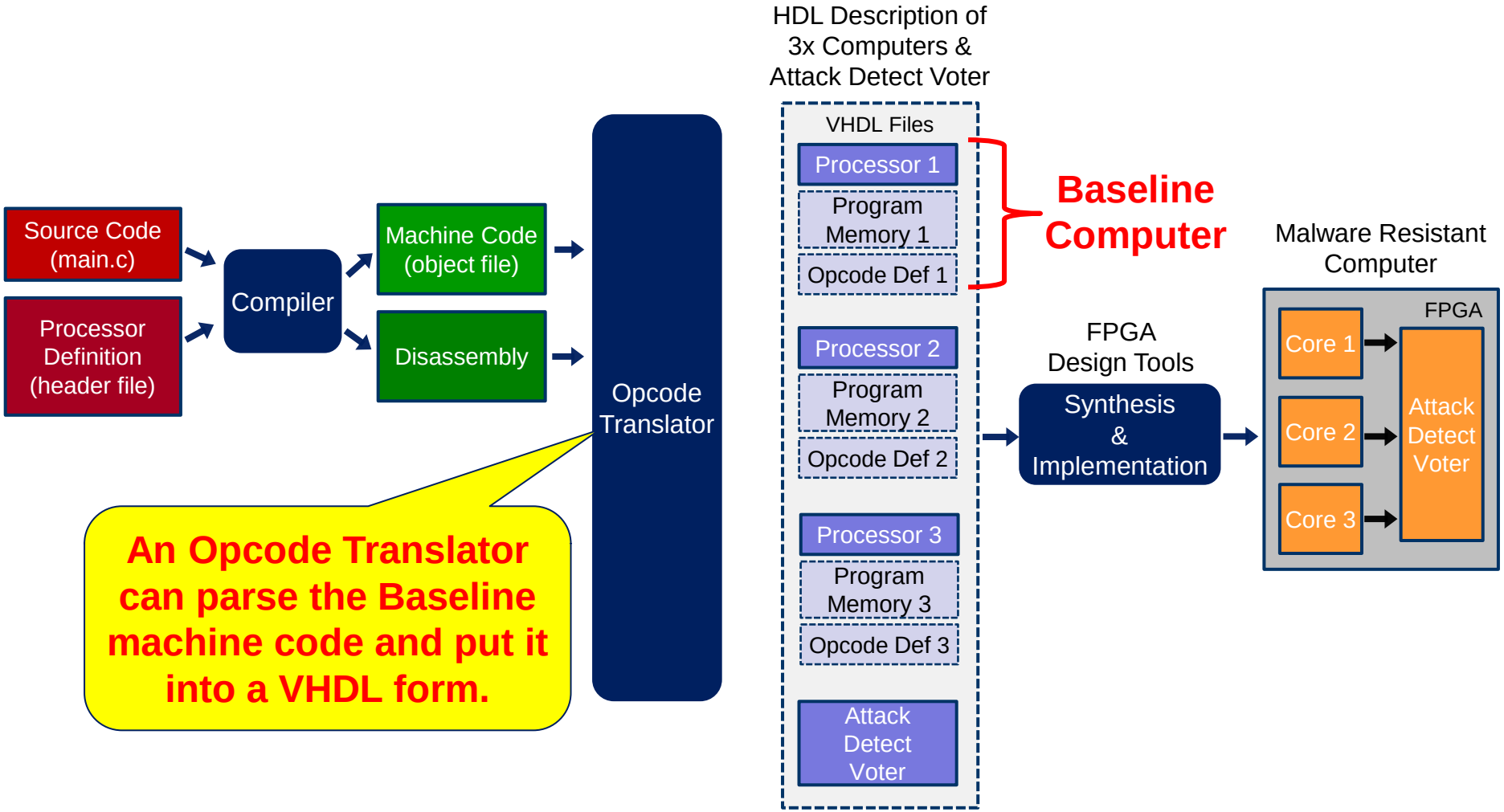
The disassembly gives us details of which fields in the machine code are Opcodes vs. Operands.



Our Approach

MONTANA STATE UNIVERSITY

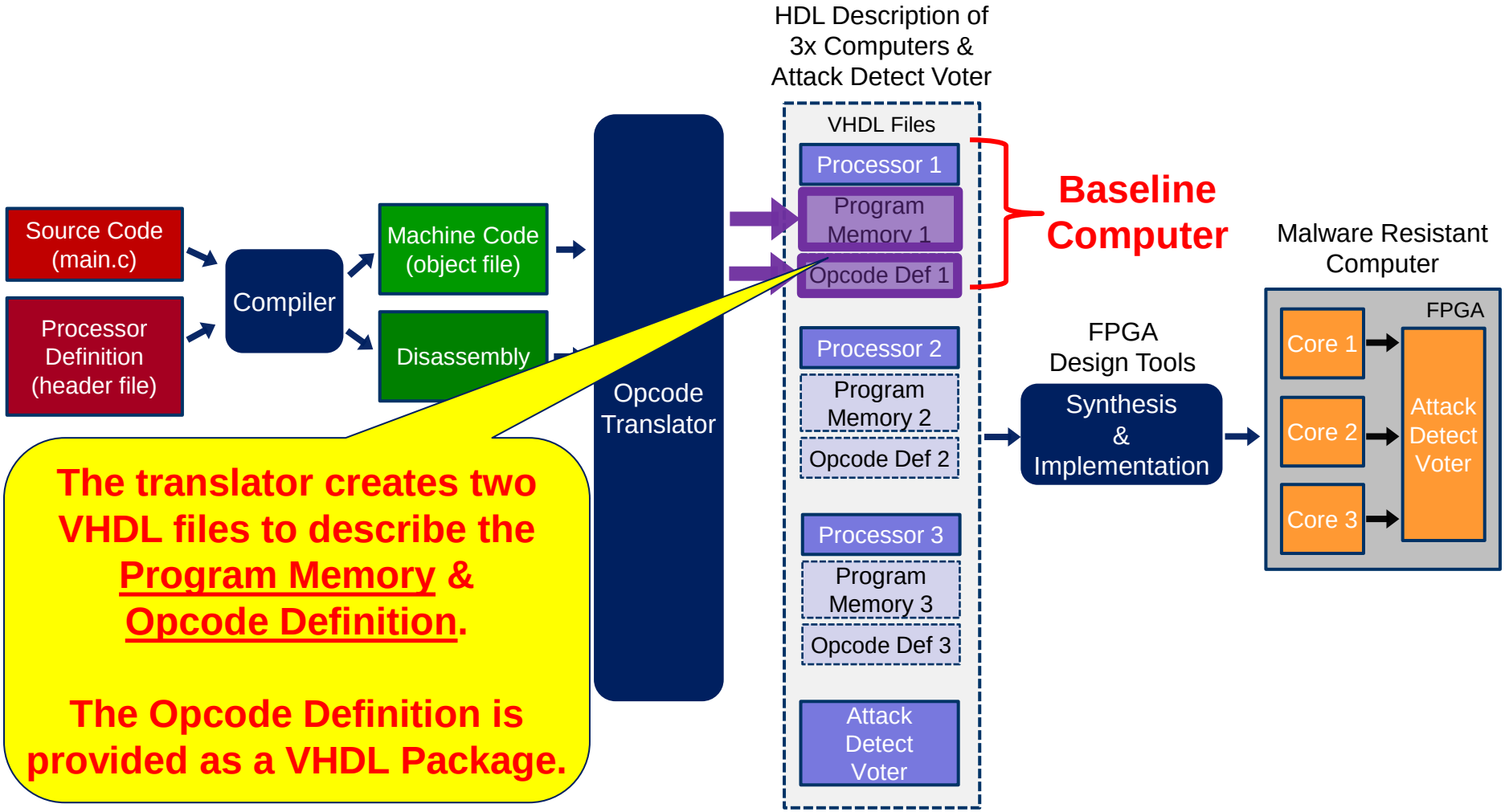
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Our Approach

MONTANA STATE UNIVERSITY

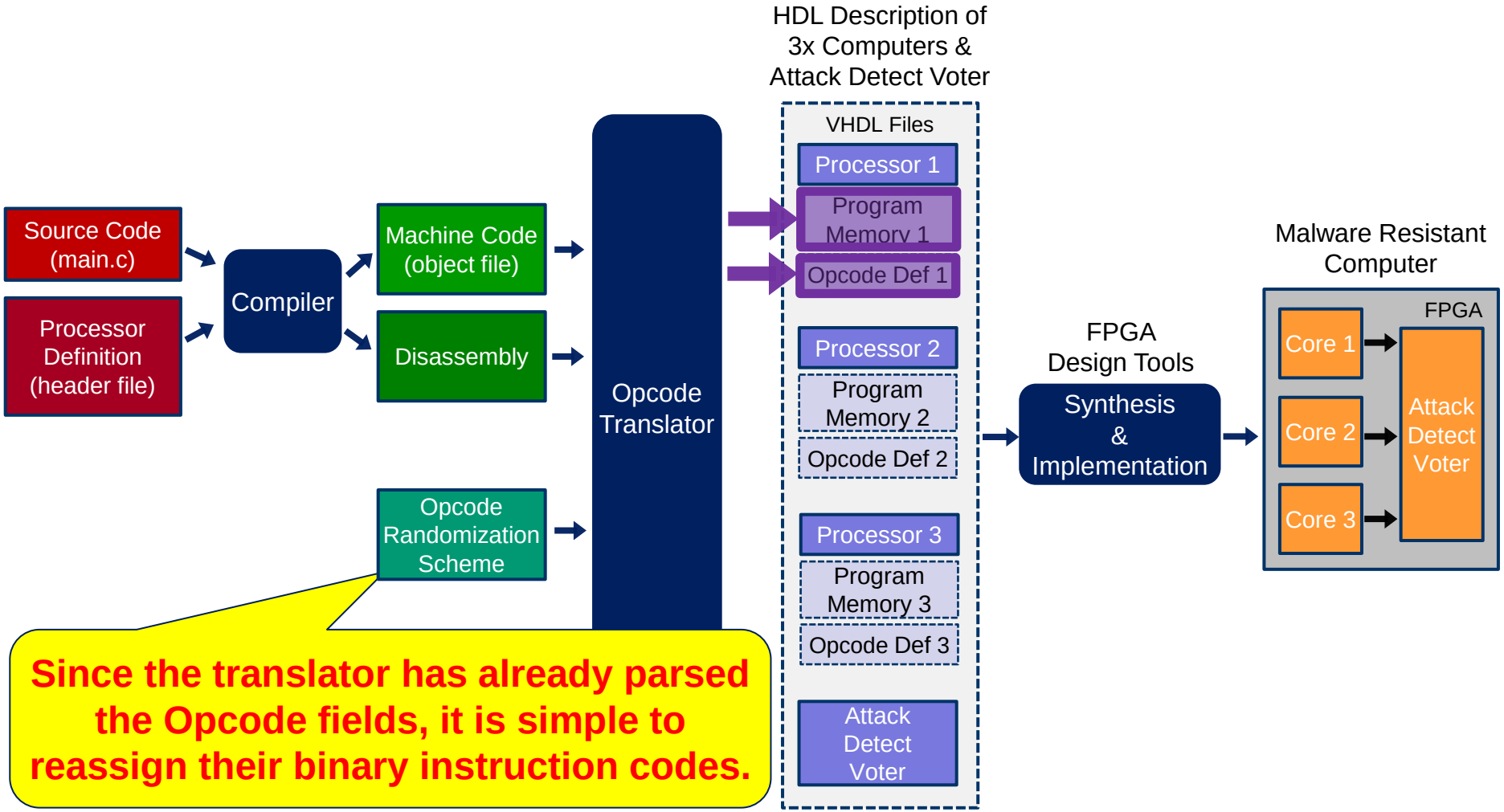
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Our Approach

MONTANA STATE UNIVERSITY

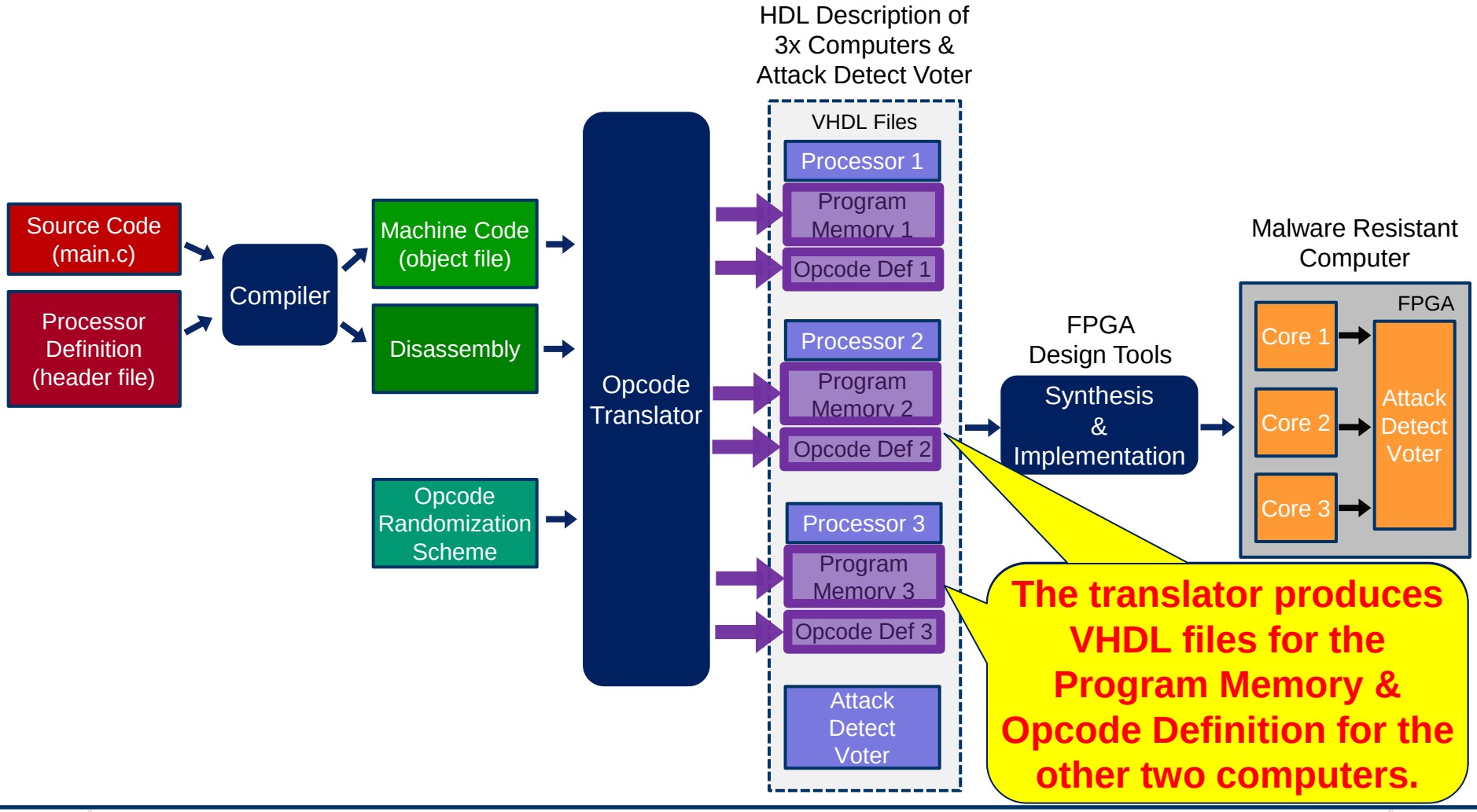
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Our Approach

MONTANA STATE UNIVERSITY

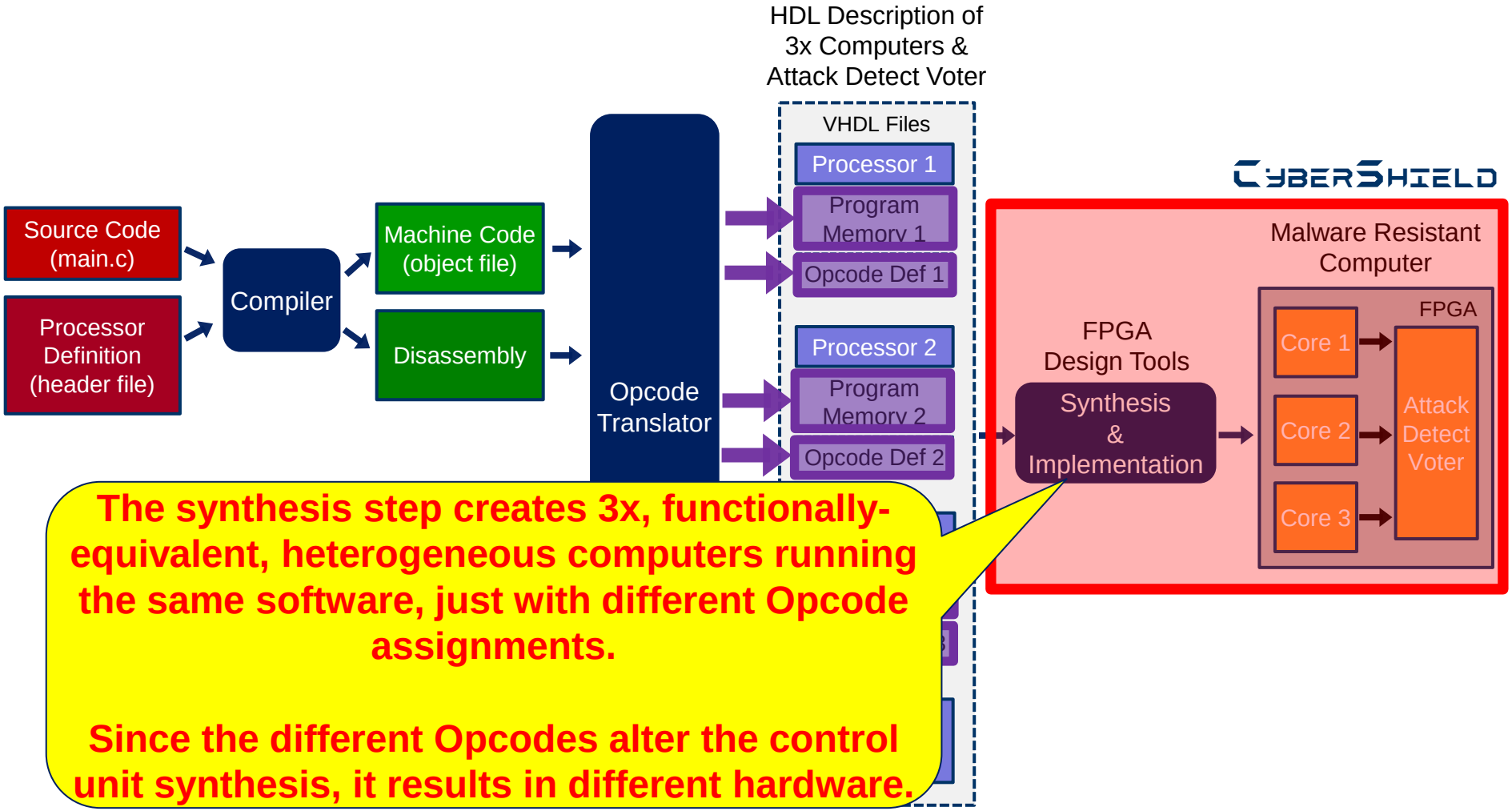
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



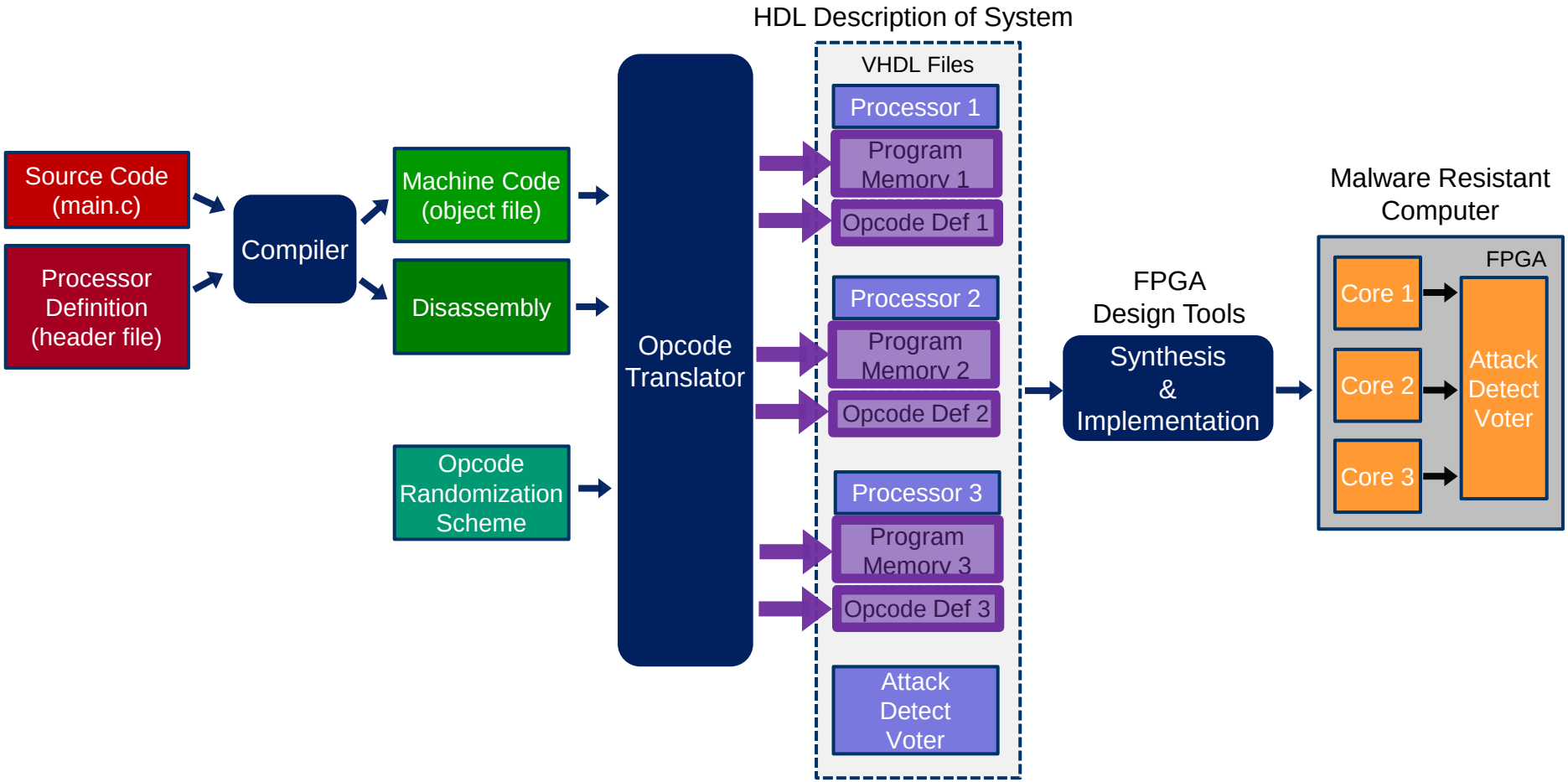
Our Approach

MONTANA STATE UNIVERSITY

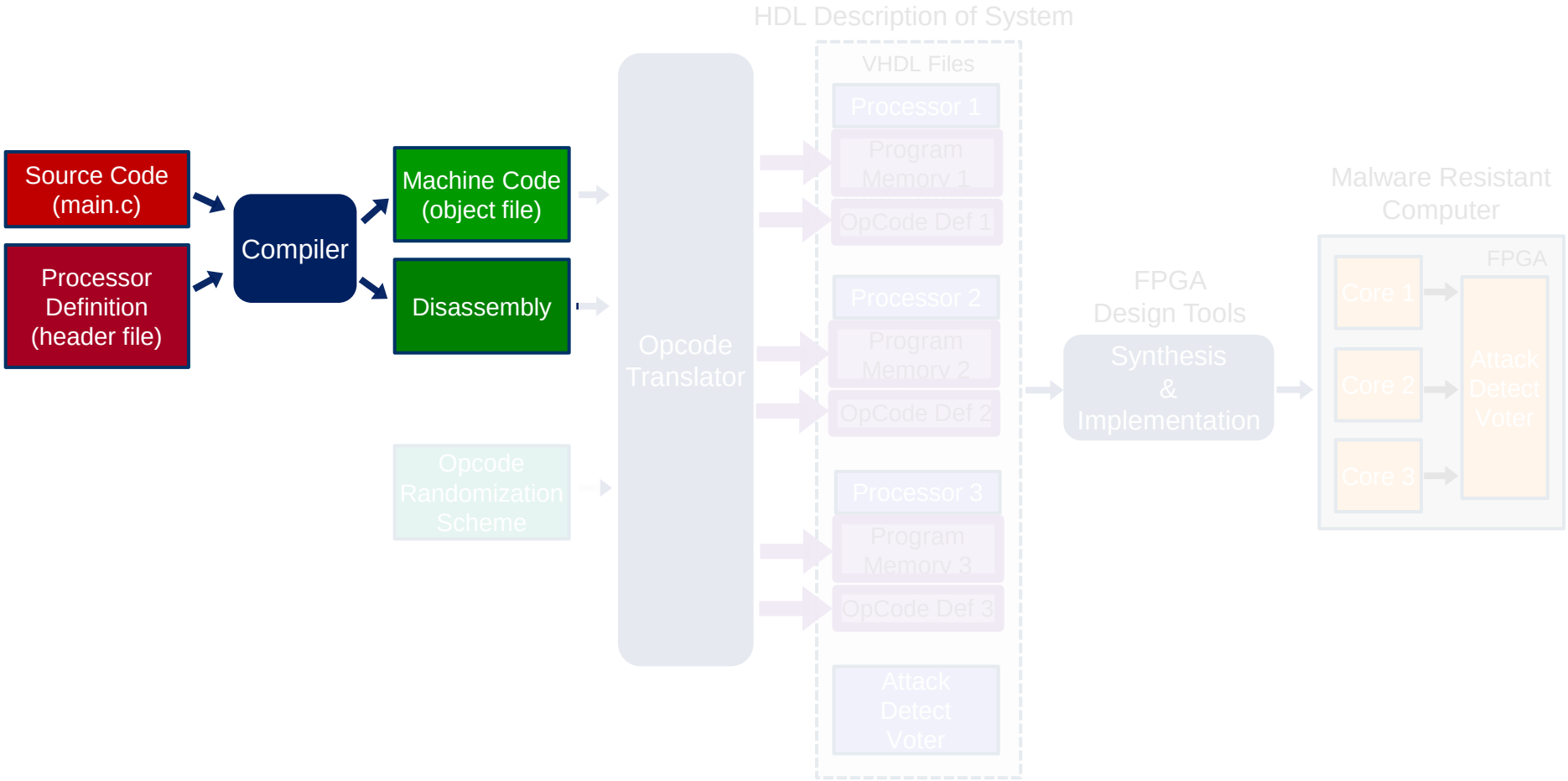
But how do we map the original source code opcode assignments used by the compiler into the two heterogenous cores?



Testbed for Demonstration

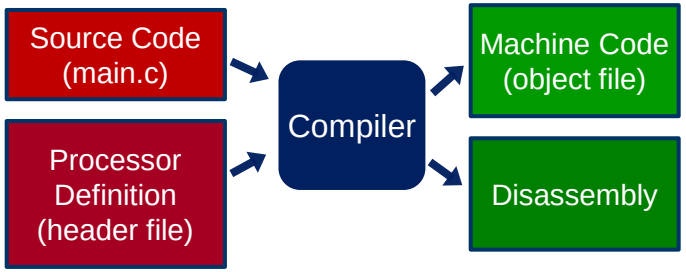


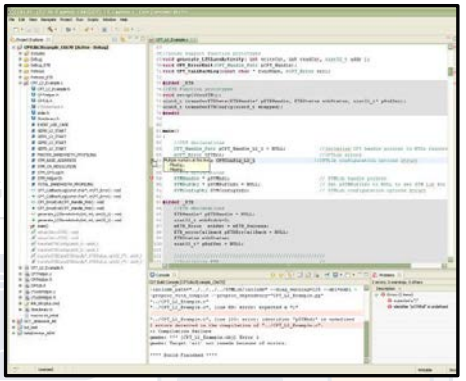
Testbed for Demonstration



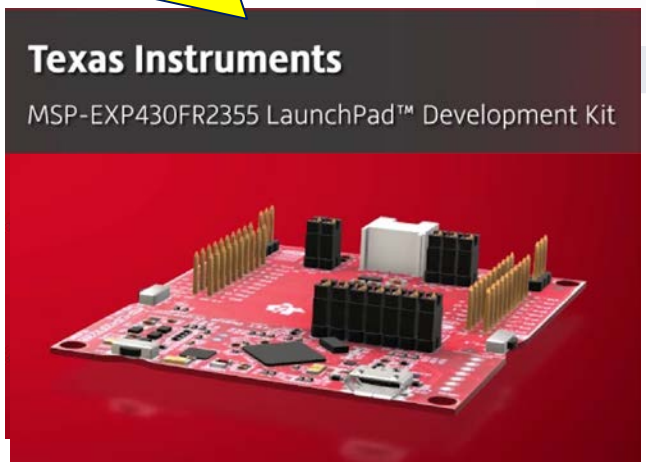
Testbed for Demonstration

Standard Eclipse Programming Environment Supporting C and Assembly.

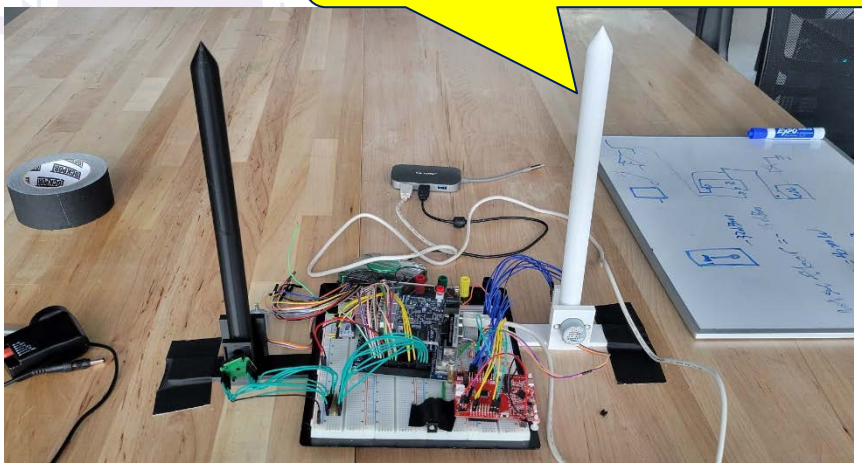




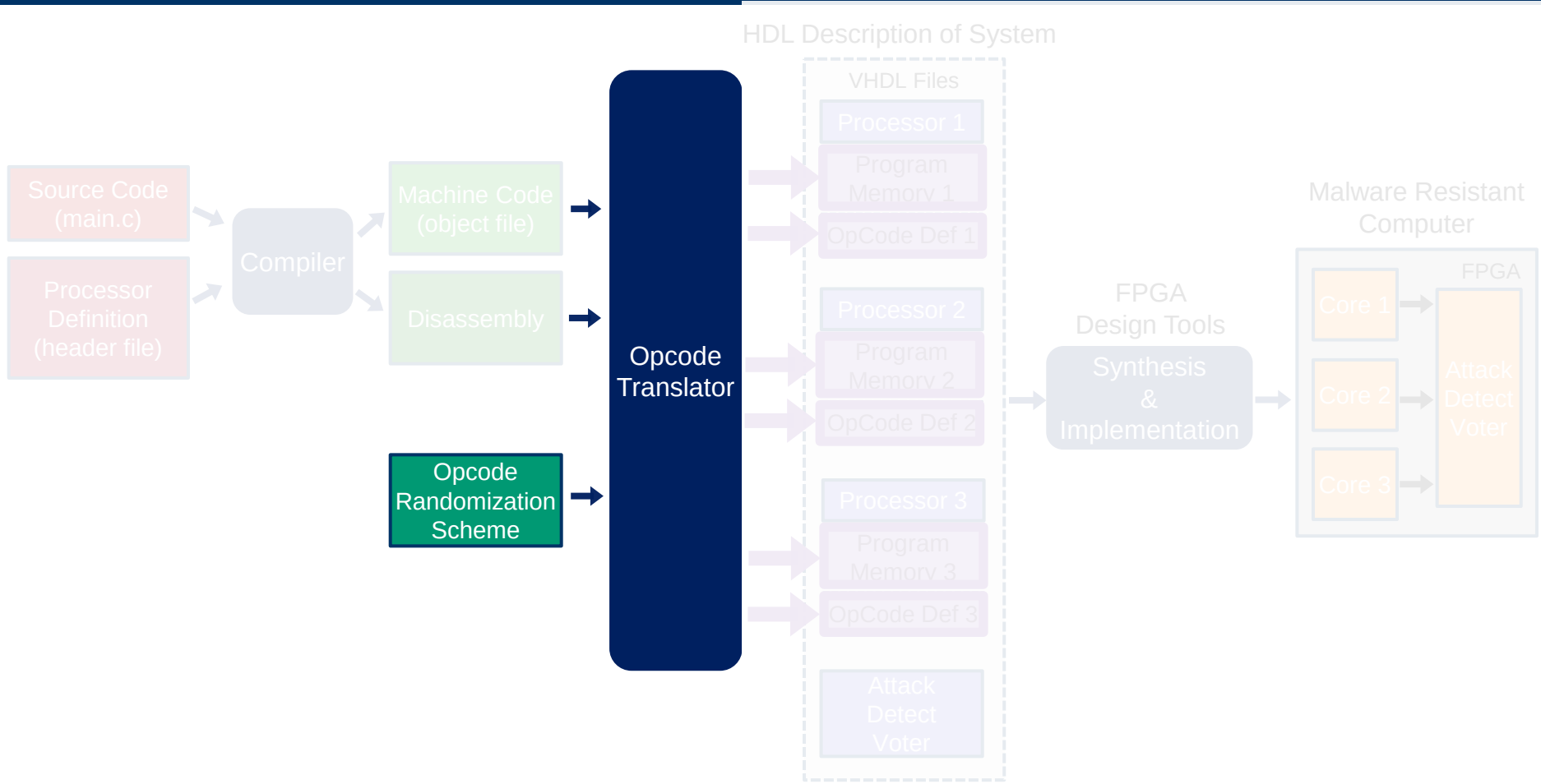
Targeting a widely-used Microcontroller, the MSP430. A 16-bit RISC processor.



Testbed program (main.c) to keep a missile upright

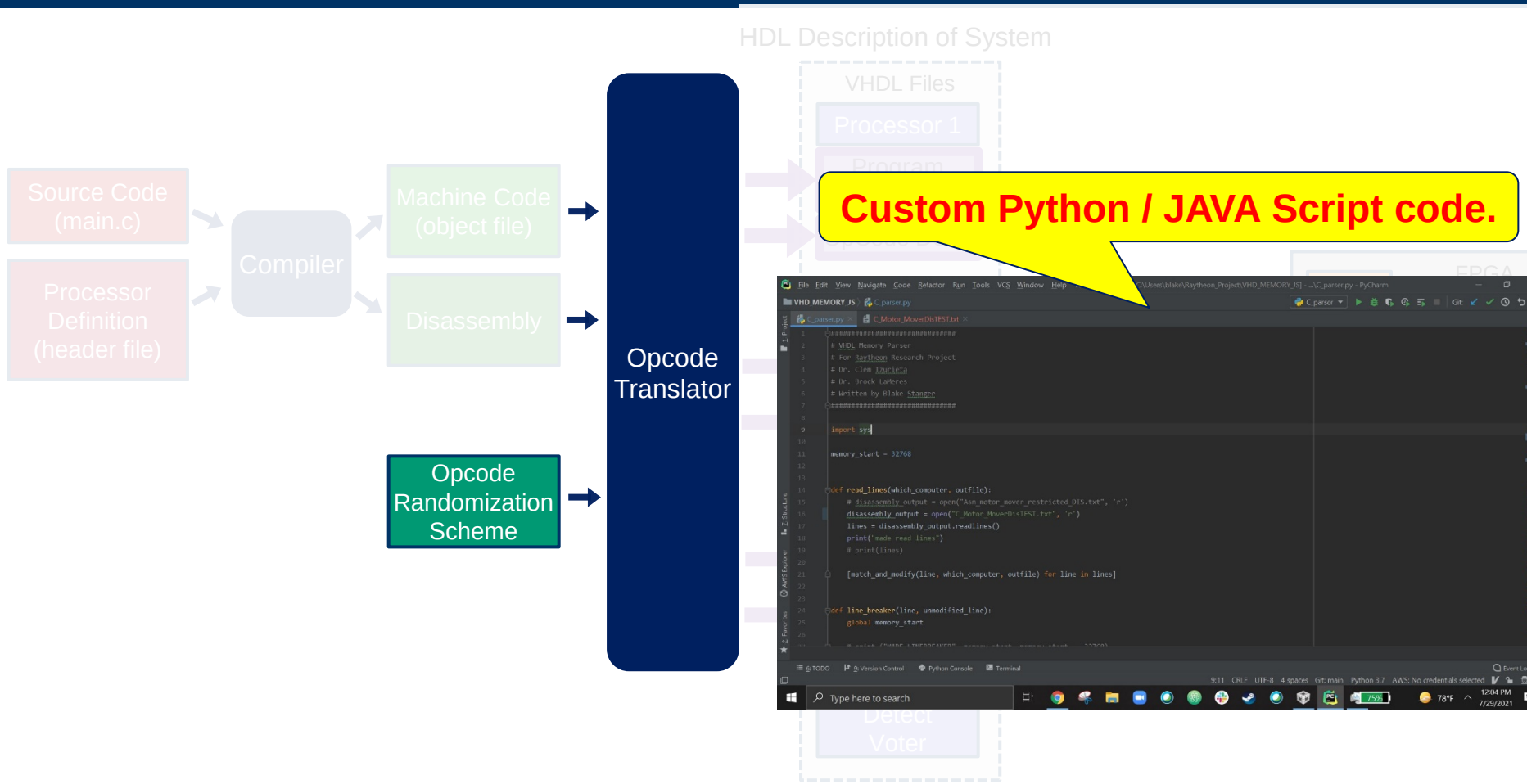


Testbed for Demonstration

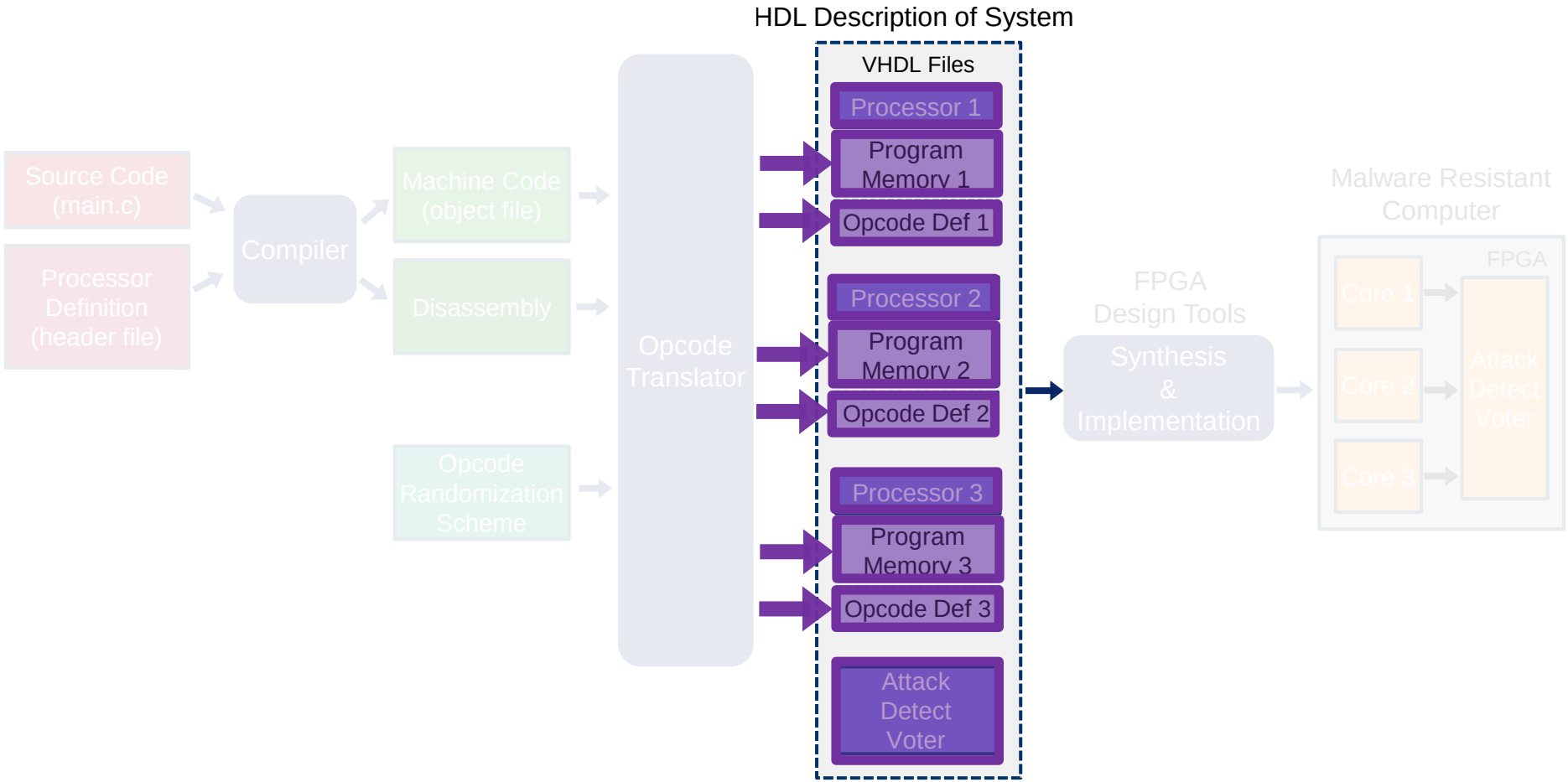


Testbed for Demonstration

MONTANA STATE UNIVERSITY



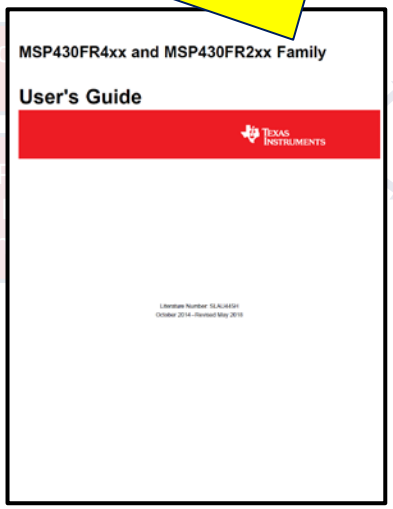
Testbed for Demonstration

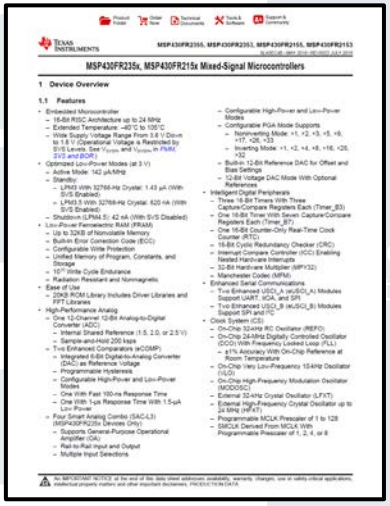


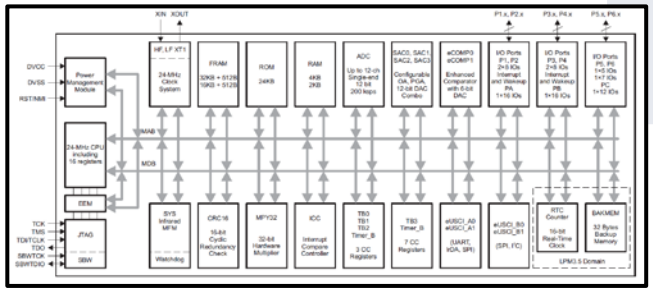
Testbed for Demonstration

MONTANA STATE UNIVERSITY

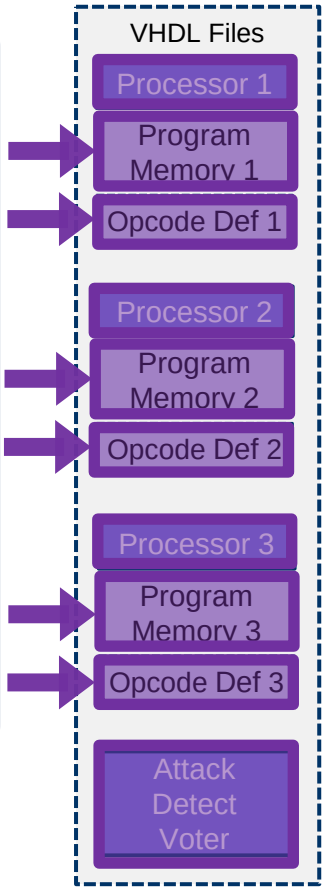
We built a fully functional MSP430 in VHDL from the TI datasheets.



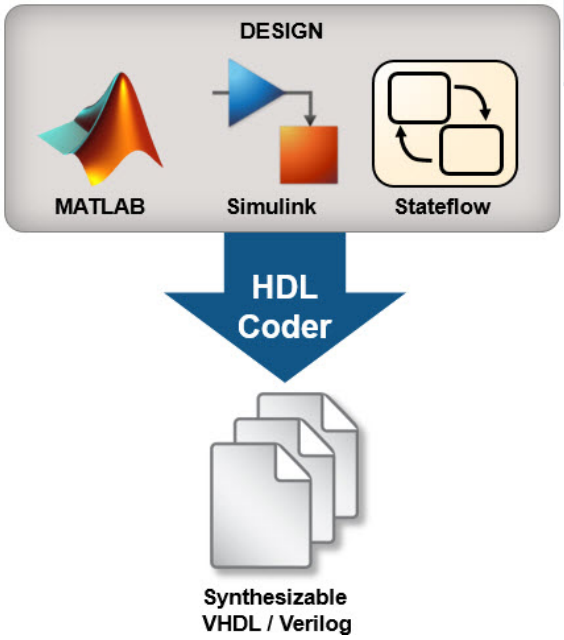




HDL Description of System



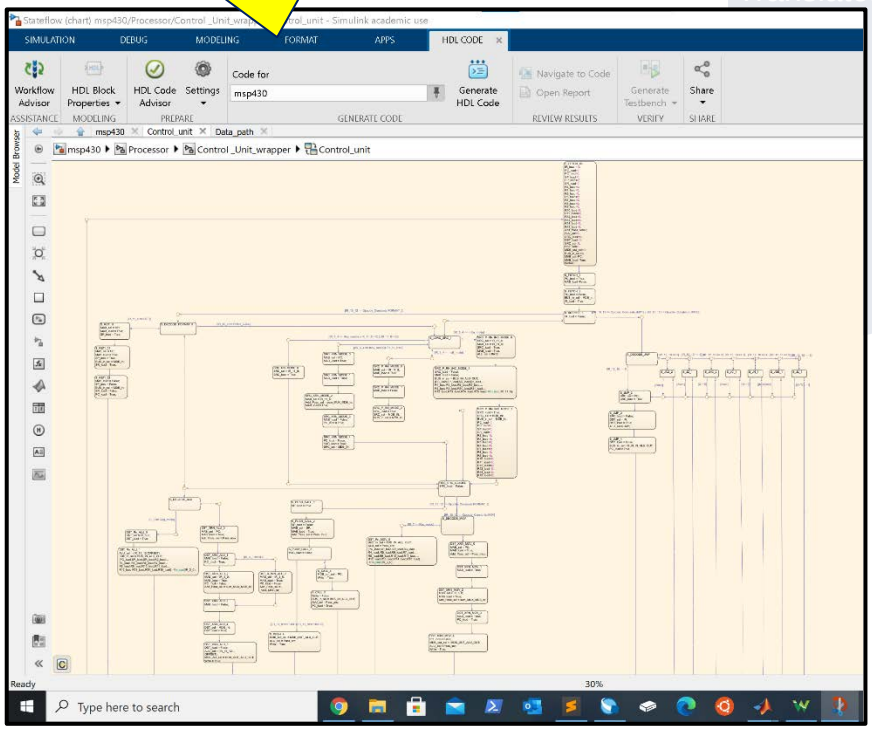
We used Matlab Simulink HDL Coder to generate the VHDL from a graphical/functional description.

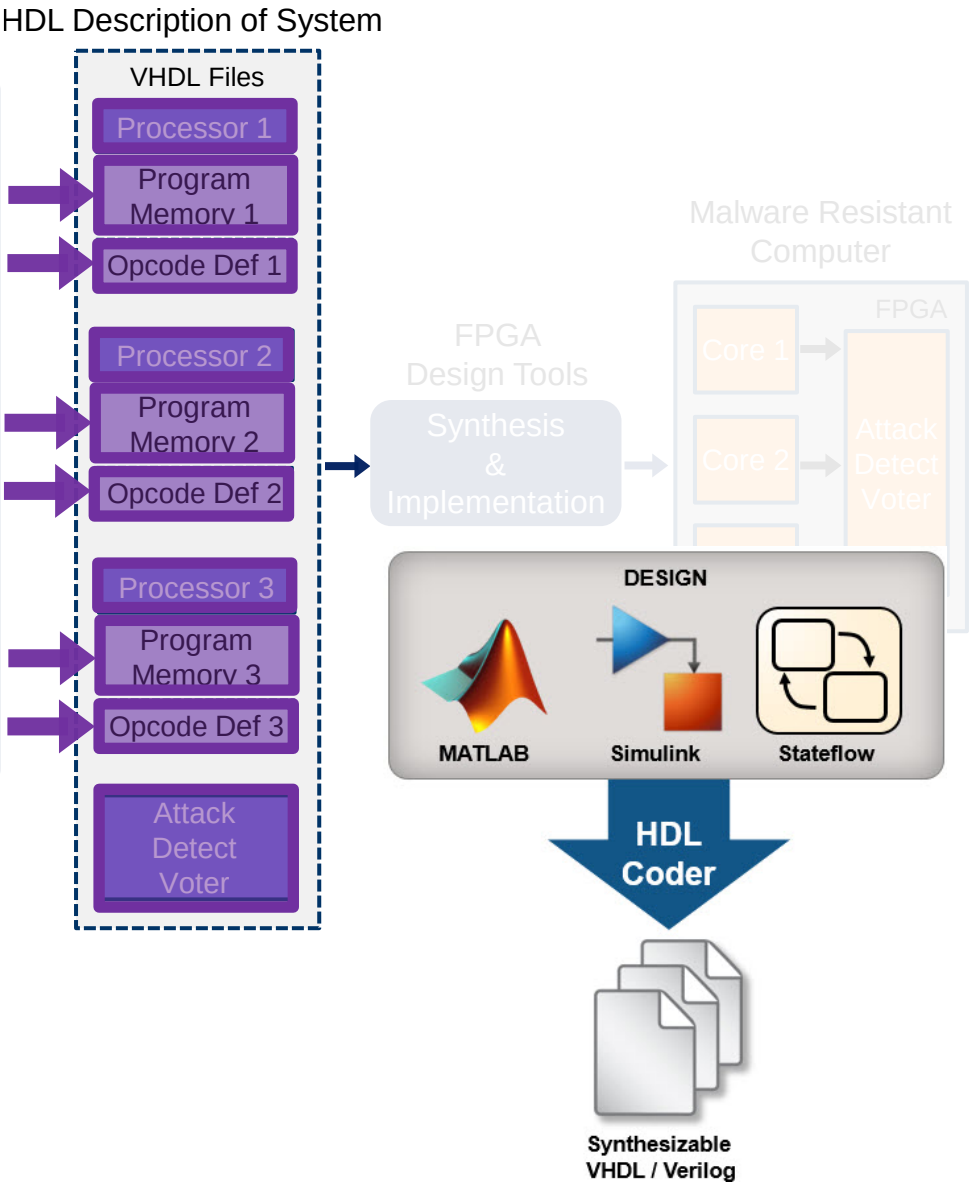


Testbed for Demonstration

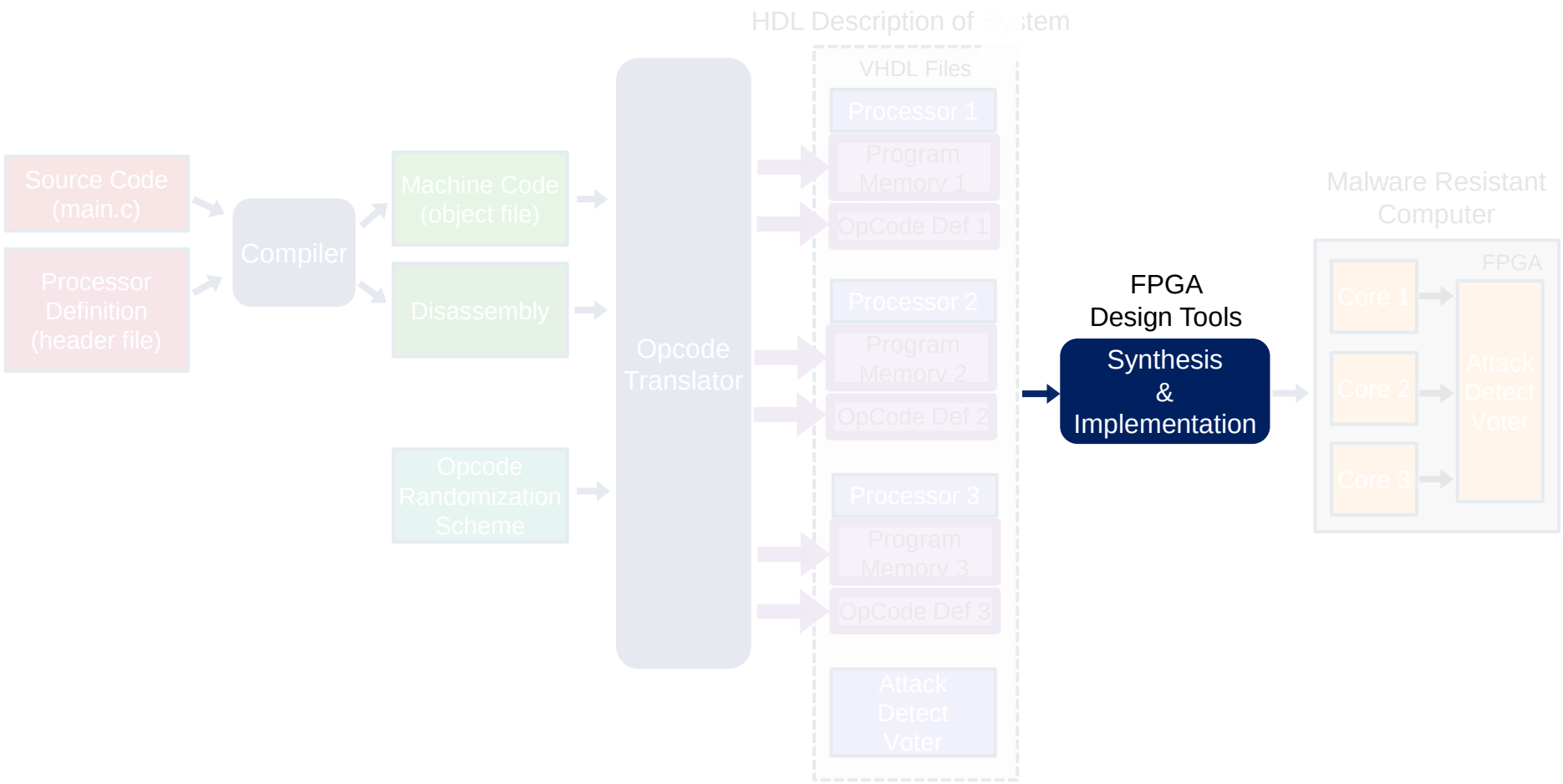
MONTANA STATE UNIVERSITY

The entire control unit is described as a graph and then converted into VHDL by the HDL coder toolbox.

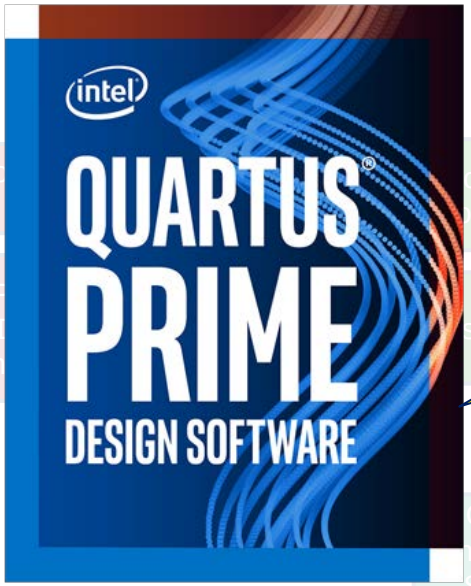




Testbed for Demonstration



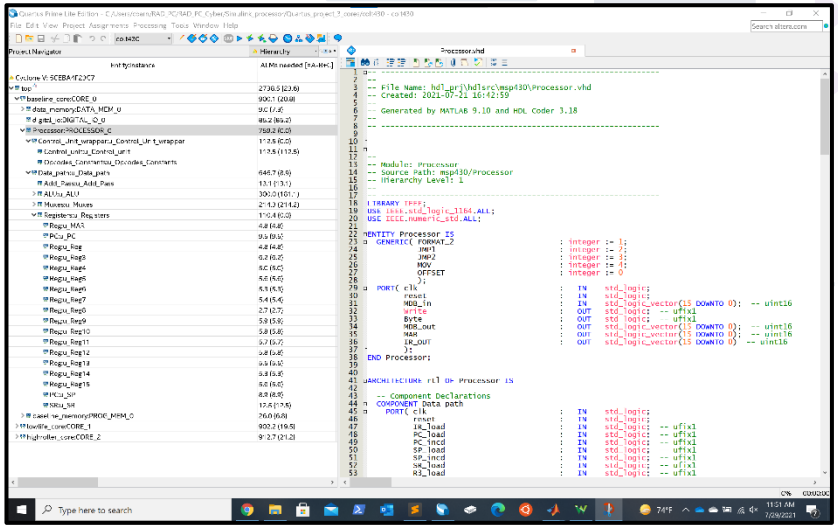
Testbed for Demonstration

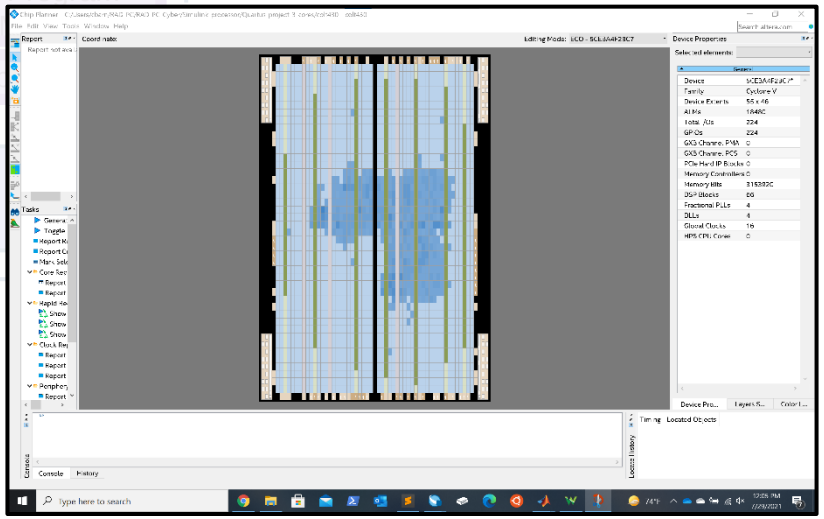


We used the intel Quartus FPGA design tools.

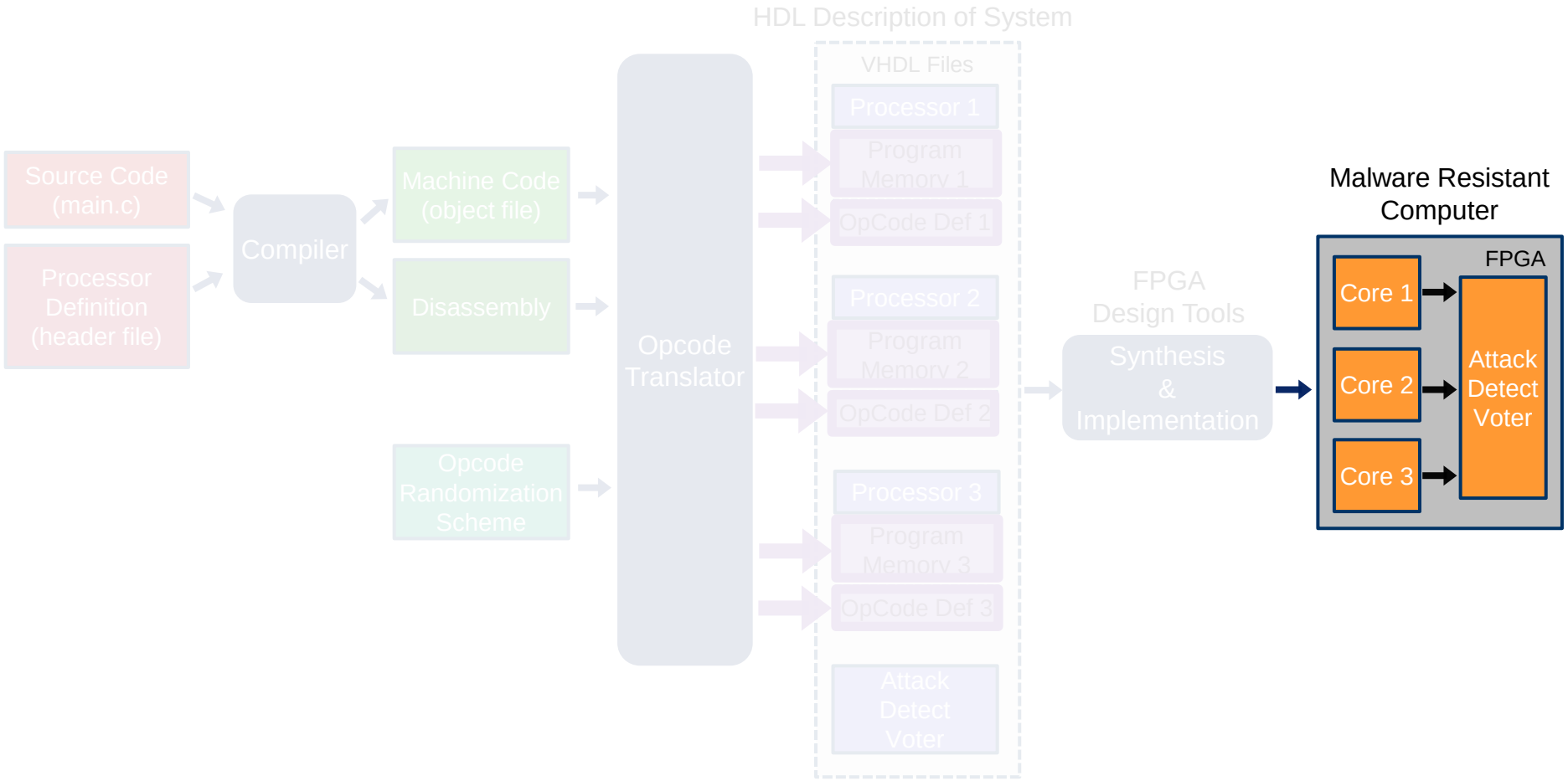
FPGA
Design Tools

Synthesis
&
Implementation





Testbed for Demonstration



Testbed for Demonstration

MONTANA STATE UNIVERSITY

HDL Description of System

VHDL Files

Processor 1

Source Code (main.c)

Processor Definition (header file)

Compiler

Machine (objec

Disassembly

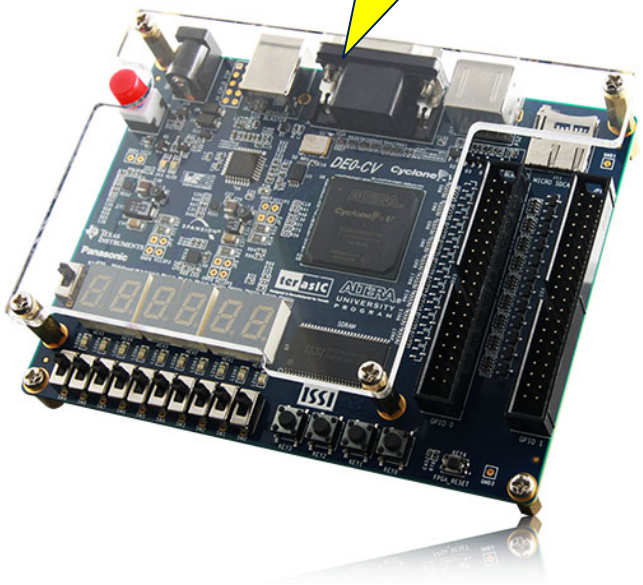
Opcode Translator

Program Memory 2

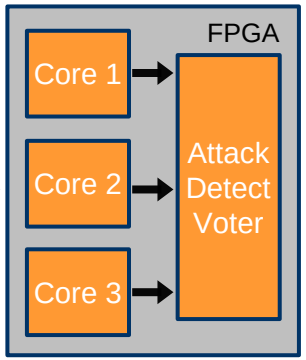
FPGA Design Tools

Synthesis & Implementation

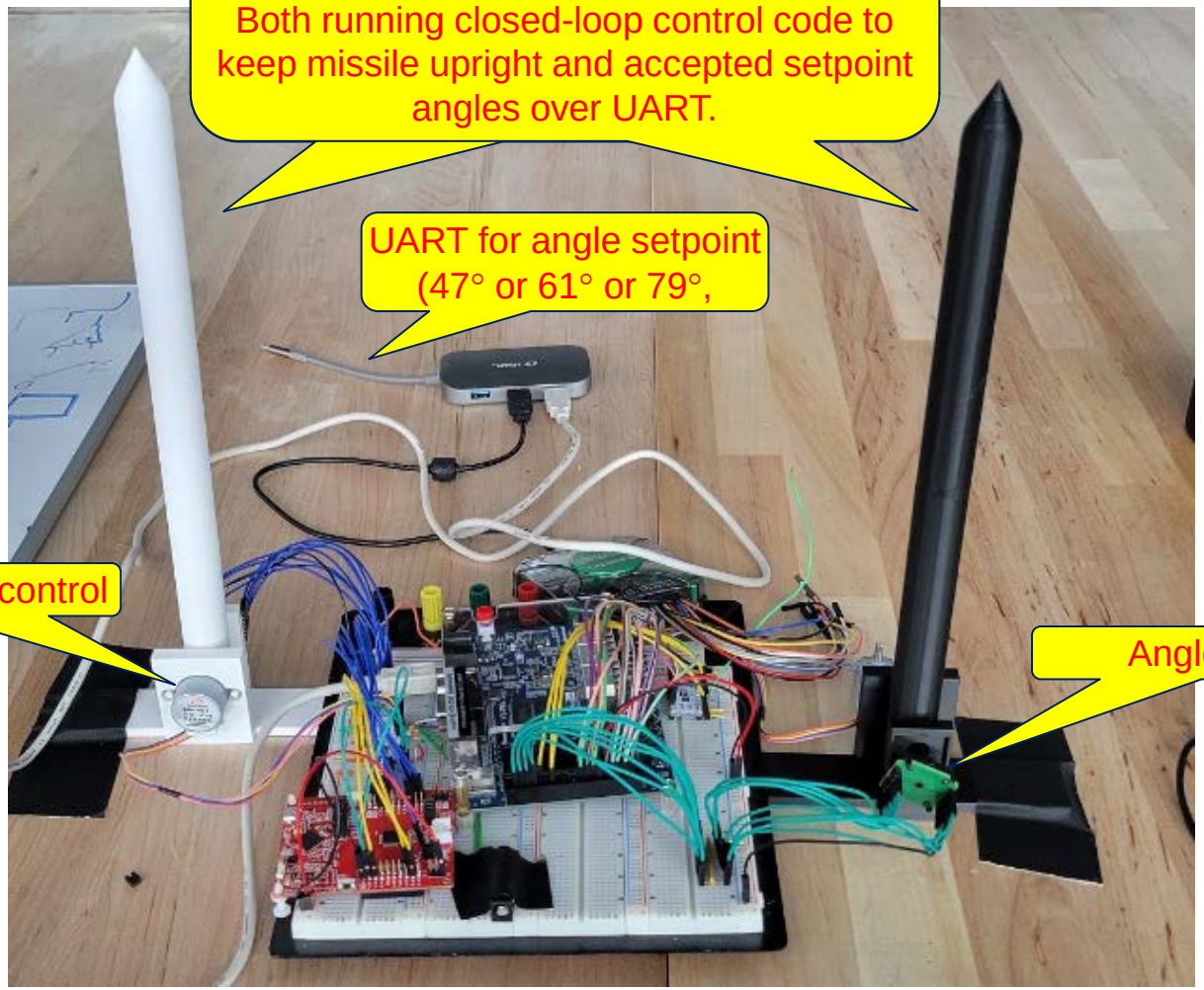
We used the DE0-CV FPGA board with an intel Cyclone V FPGA.



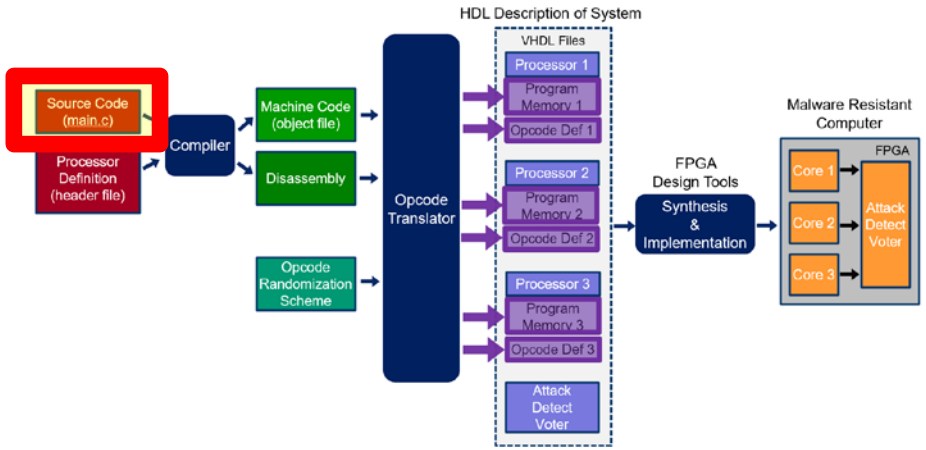
Malware Resistant Computer



Demonstration Under Attack



Demonstration Under Attack



Program Description

The computer periodically sends the stepper motor its setpoint angle. The send frequency is dictated by a timer that triggers and interrupt.

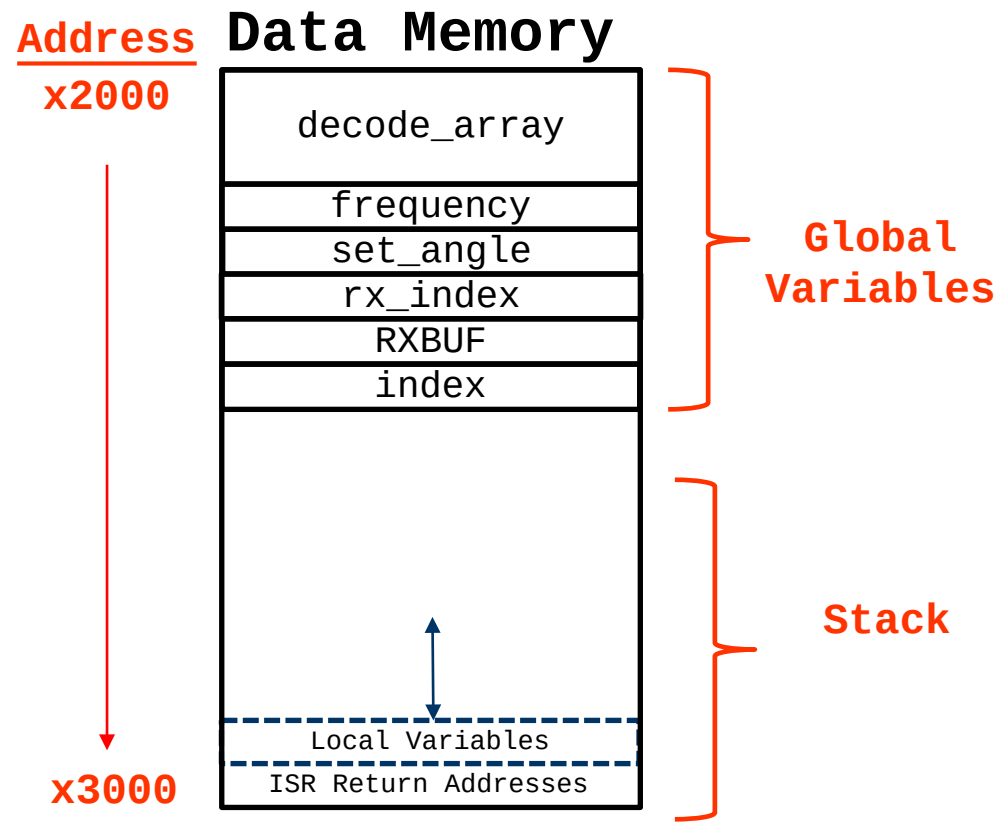
The computer continuously reads the actual angle of the missile from the sensor and compares it to the setpoints. It adjusts motor accordingly.

New setpoints are received asynchronously from a user over UART. A Rx on the UART link triggers an IRQ.

Demonstration Under Attack

```
    while(1){
        for(index=0xFFFF;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[P1IN];

        if(temp<set_angle){
            P2OUT &=~(BIT2); //enable stepper motor
//          P2OUT |=(BIT1); //set direction
            P2OUT &=~(BIT5); //set direction
            temp = set_angle-temp;
        }else if (temp>set_angle){
            P2OUT &=~(BIT2); //enable stepper motor
//            P2OUT &=~(BIT1); //set direction
            P2OUT |=(BIT5); //set direction
            temp = temp-set_angle;
        }else{
            P2OUT |=BIT2; //Disable stepper motor
        }
        frequency = 4000 - 63*(temp);
    }
}

#pragma vector = TIMER0_B0_VECTOR;
interrupt void Timer_ISR(){
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)



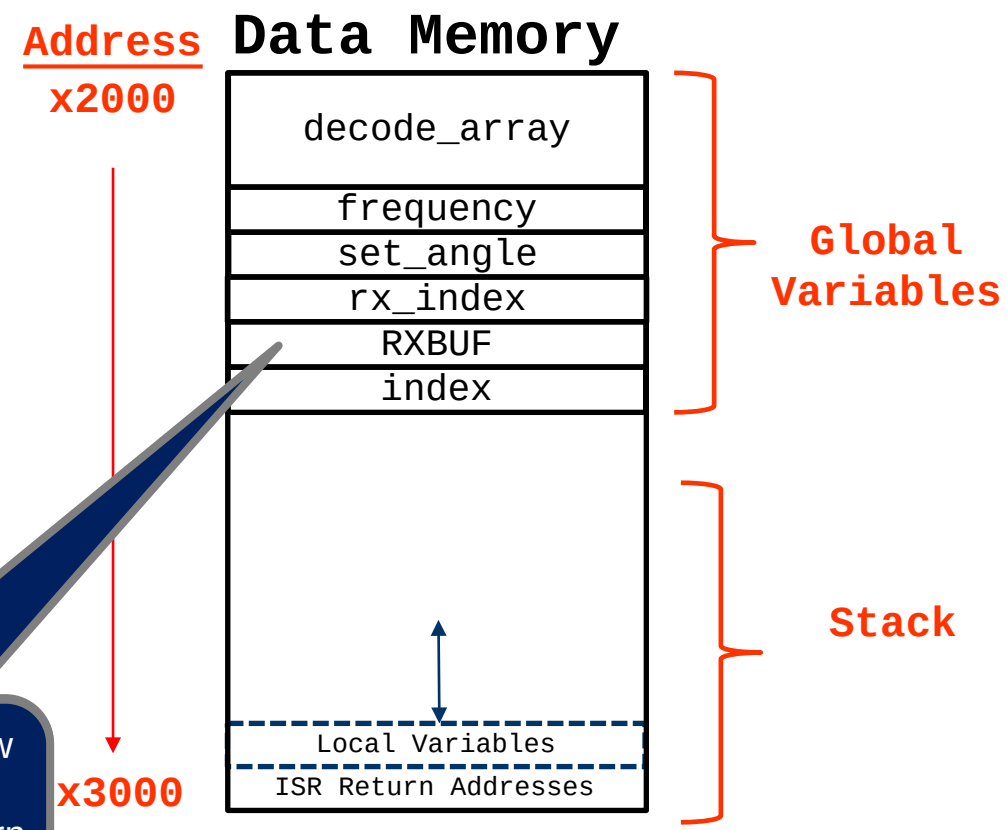
Demonstration Under Attack

```
    while(1){
        for(index=0xFFFF;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[P1IN];

        if(temp<set_angle){
            P2OUT &=~(BIT2); //enable stepper motor
//          P2OUT |=(BIT1); //set direction
            P2OUT &=~(BIT5); //set direction
            temp = set_angle-temp;
        }else if (temp>set_angle){
            P2OUT &=~(BIT2); //enable stepper motor
//            P2OUT &=~(BIT1); //set direction
            P2OUT |=(BIT5); //set direction
            temp = temp-set_angle;
        }else{
            P2OUT |=BIT2; //Disable stepper motor
        }
        frequency = 4000 - 63*(temp);
    }
}

#pragma vector = TIMER0_B0_VECTOR;
interrupt void Timer_ISR(){
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

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

Program Vulnerabilities (Classic Buffer Overflow Attack)



Demonstration Under Attack

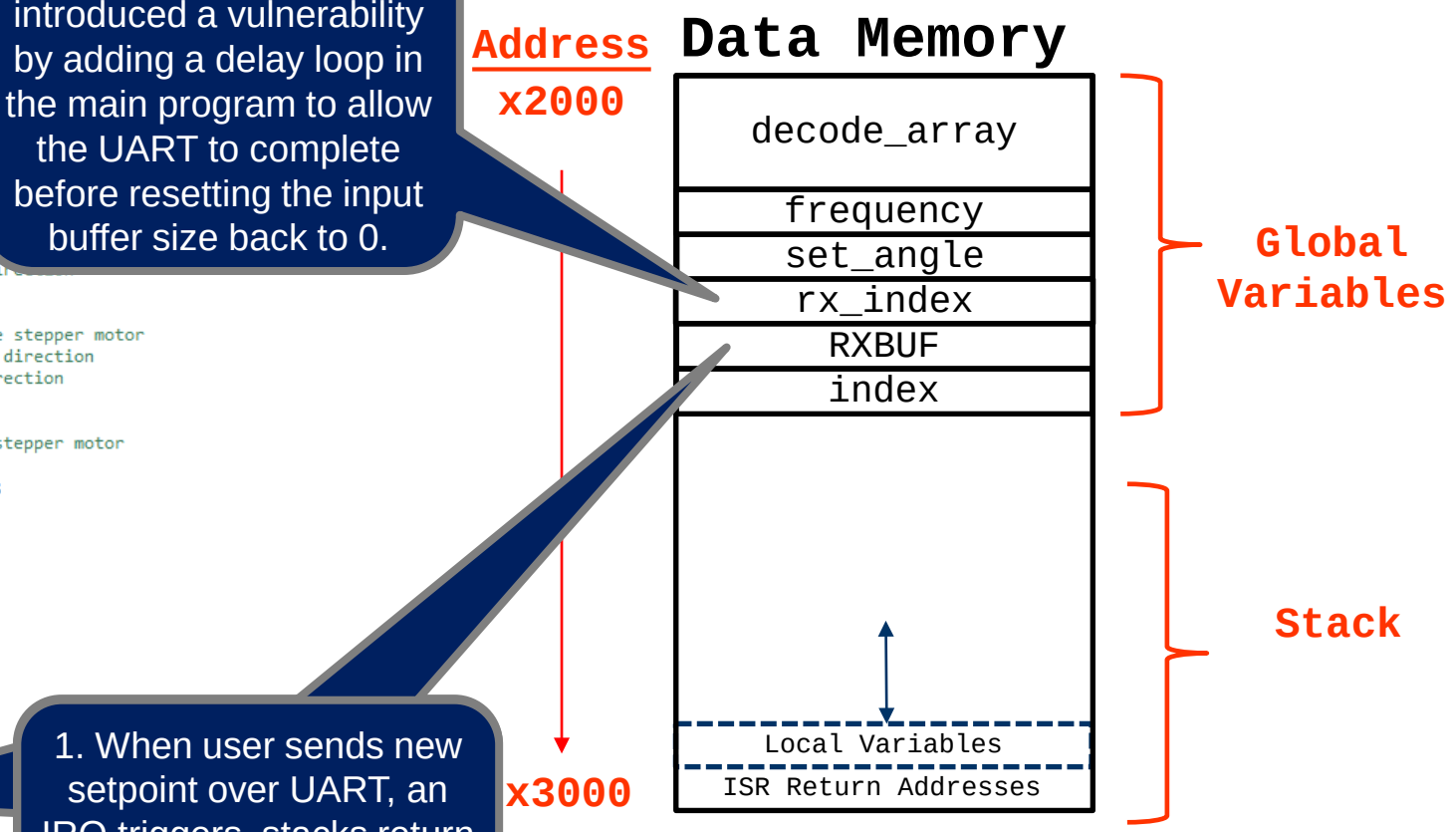
MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>set_angle){
        P2OUT &=~(BIT2); //enable stepper motor
//            P2OUT &=~(BIT1); //set direction
        P2OUT |=(BIT5); //set direction
        temp = temp-set_angle;
    }else{
        P2OUT |=BIT2; //Disable stepper motor
    }
    frequency = 4000 - 63*(temp);
  }
}

#pragma vector = TIMER0_B0_VECTOR;
interrupt void Timer_ISR(){
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

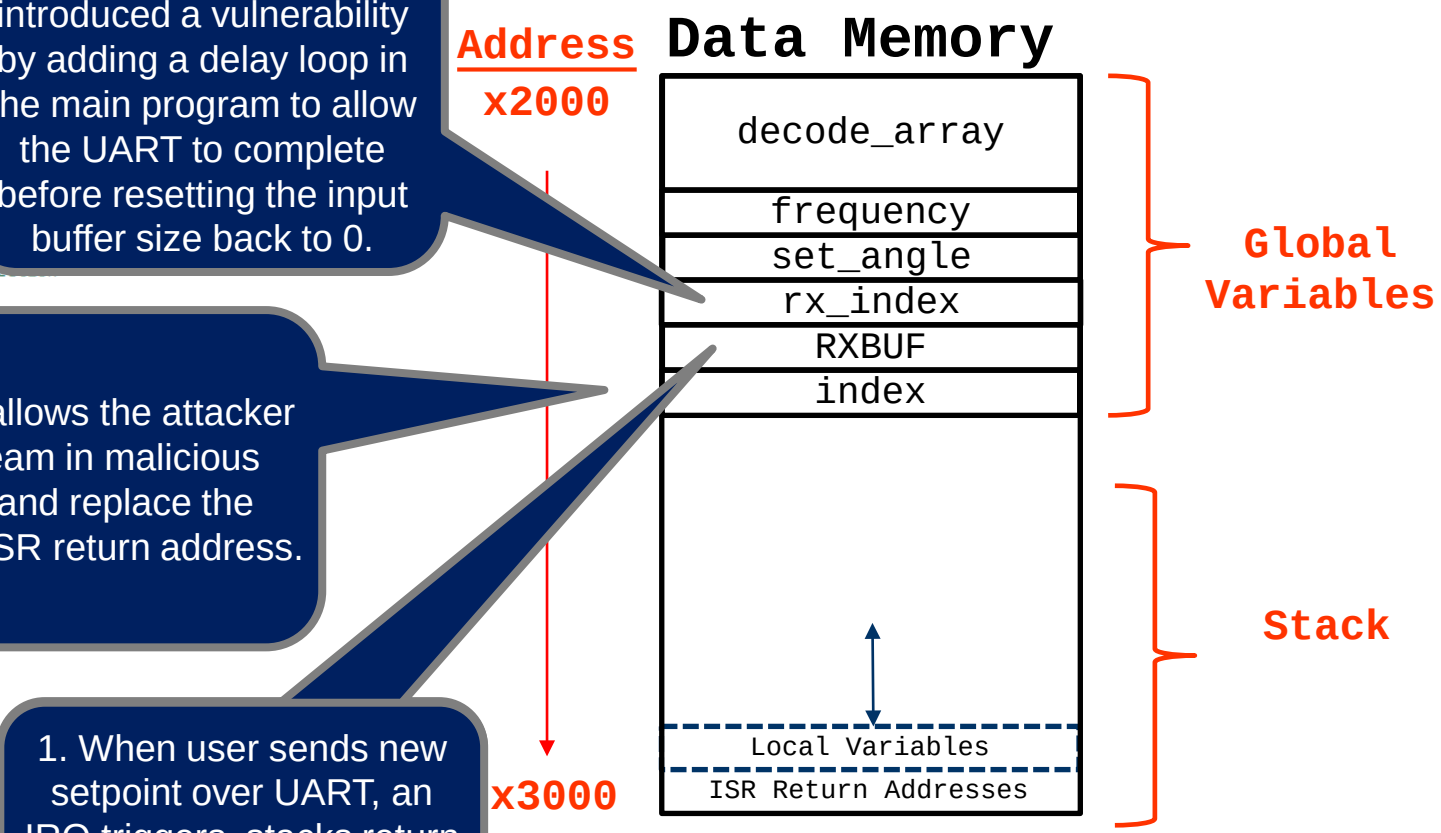
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

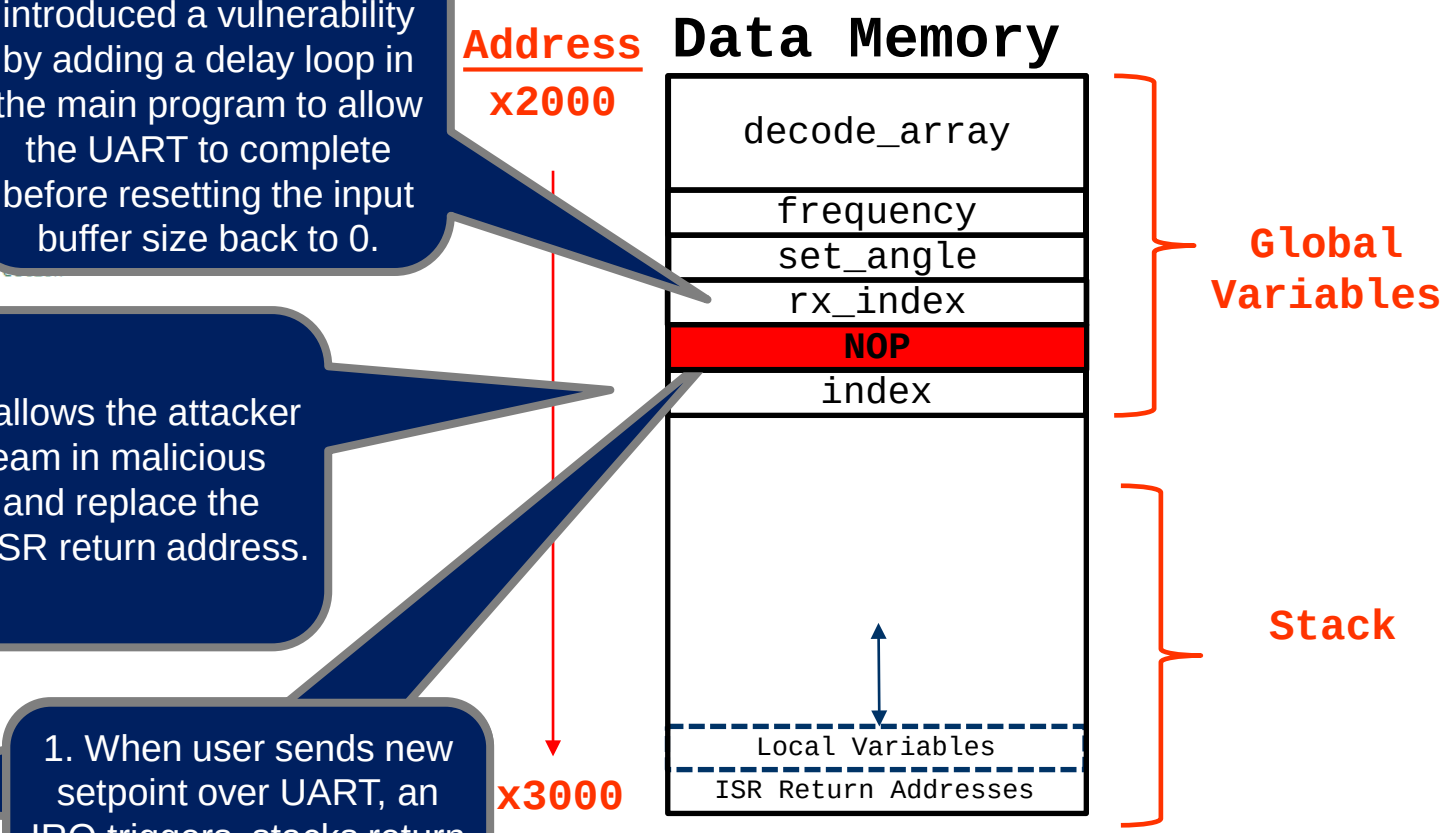
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

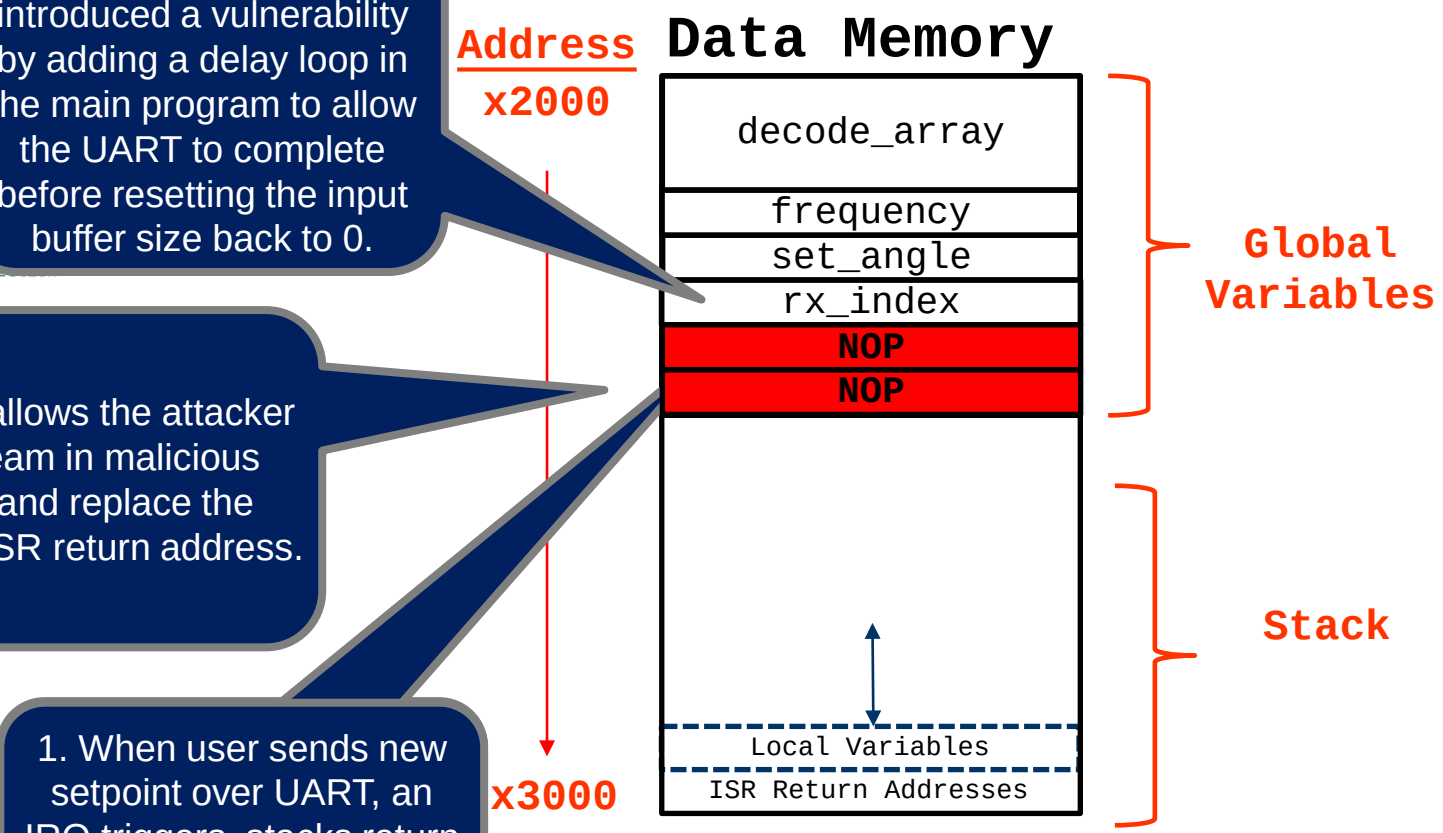
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

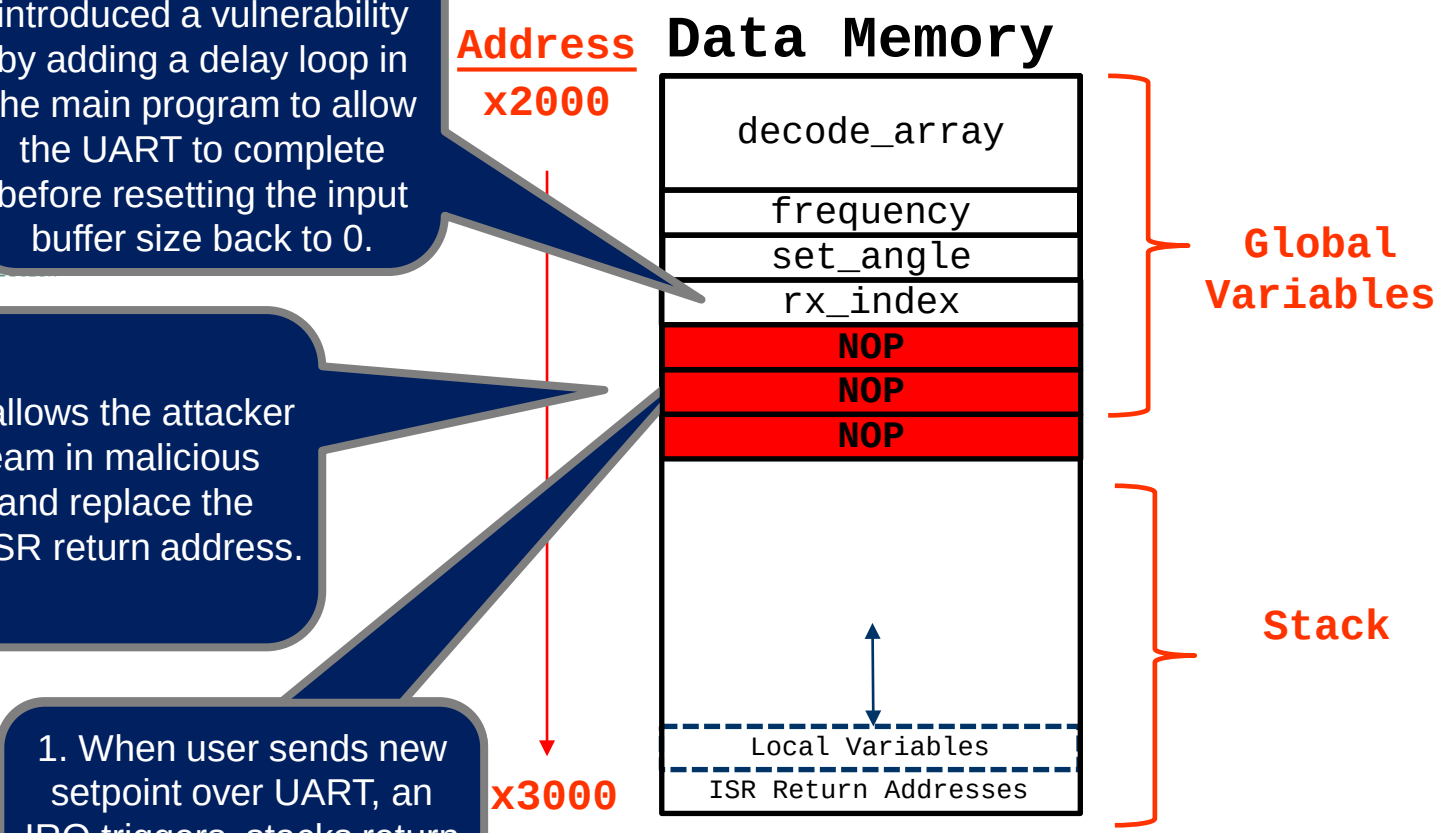
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

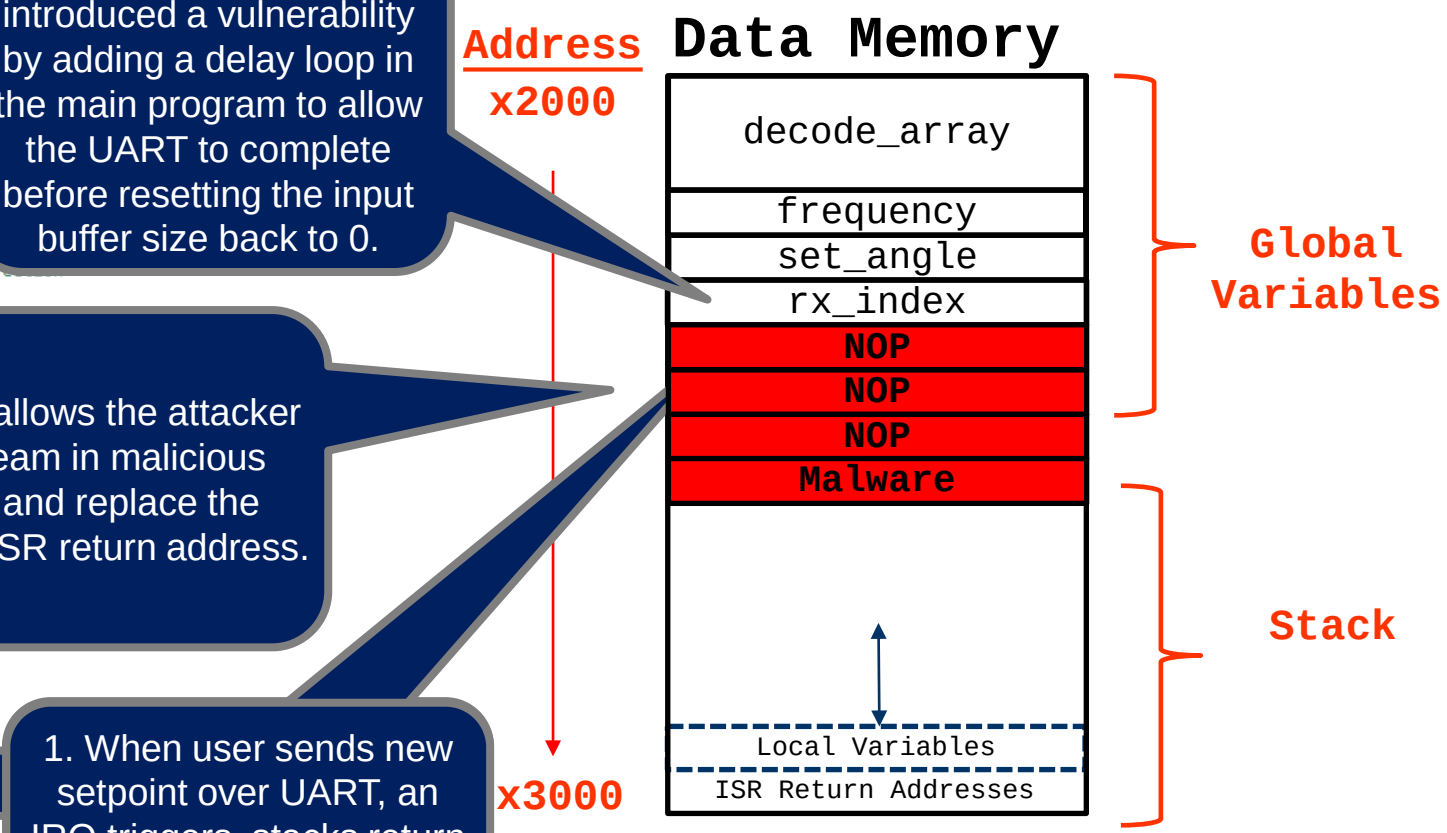
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

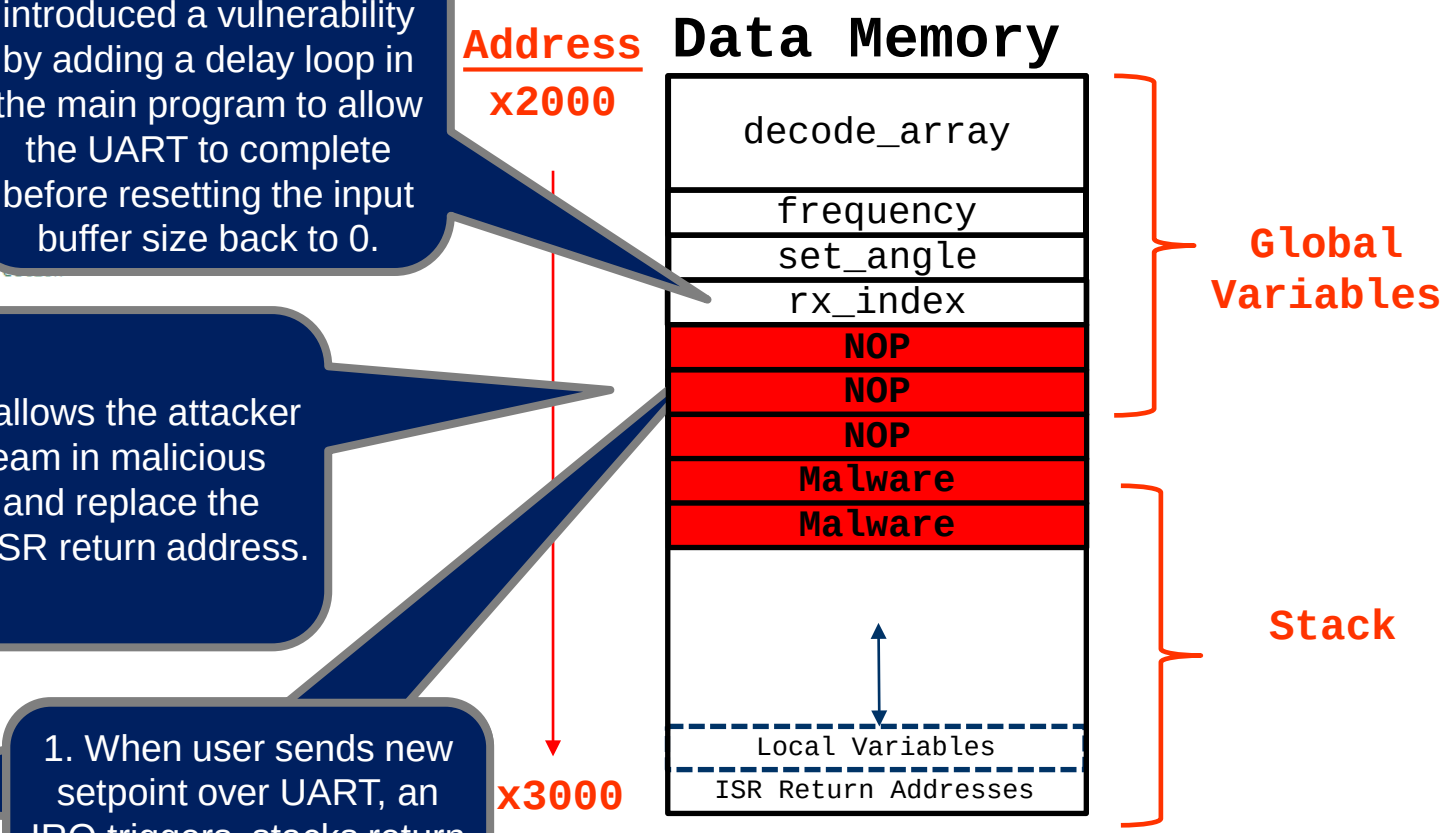
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

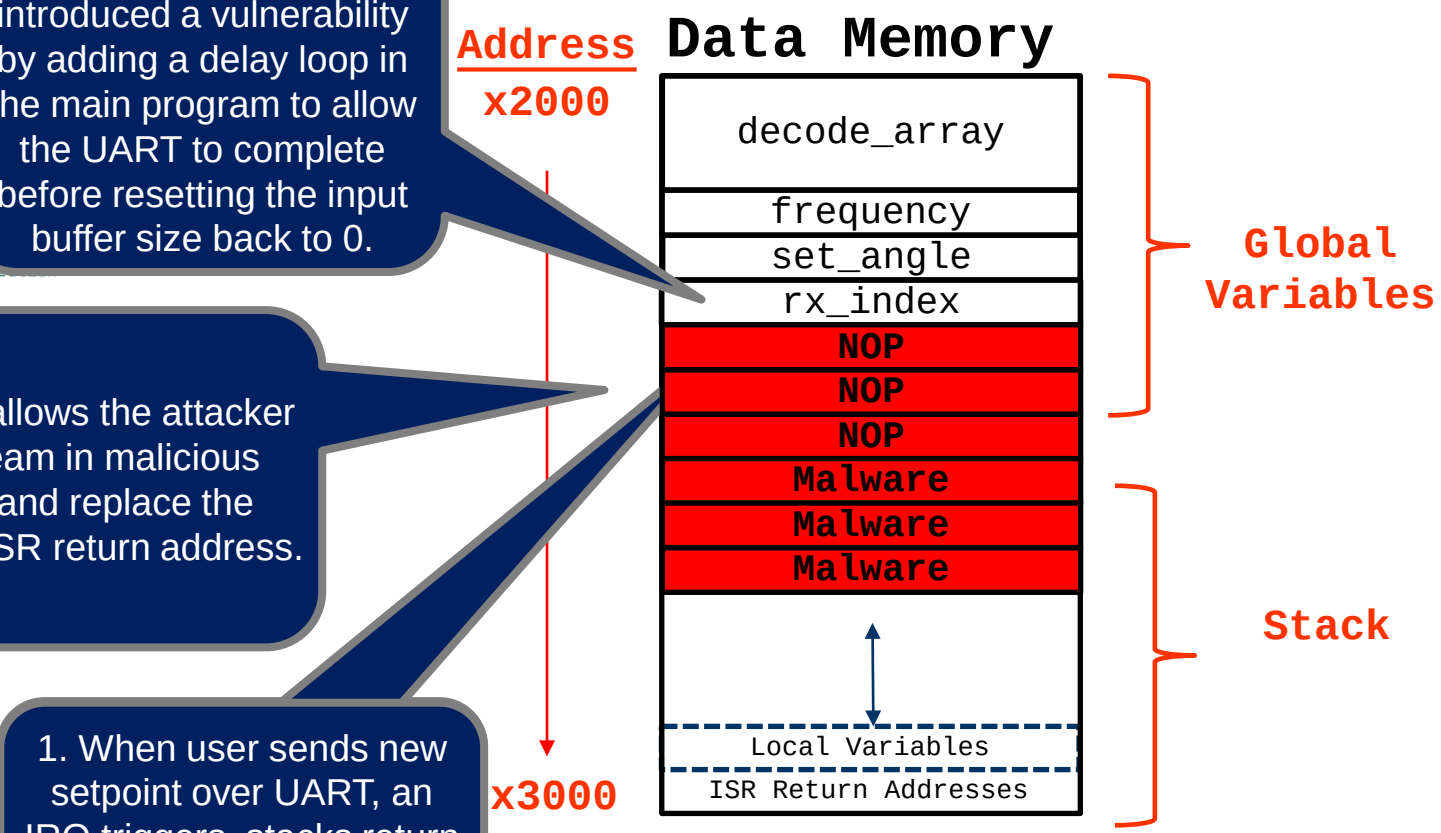
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

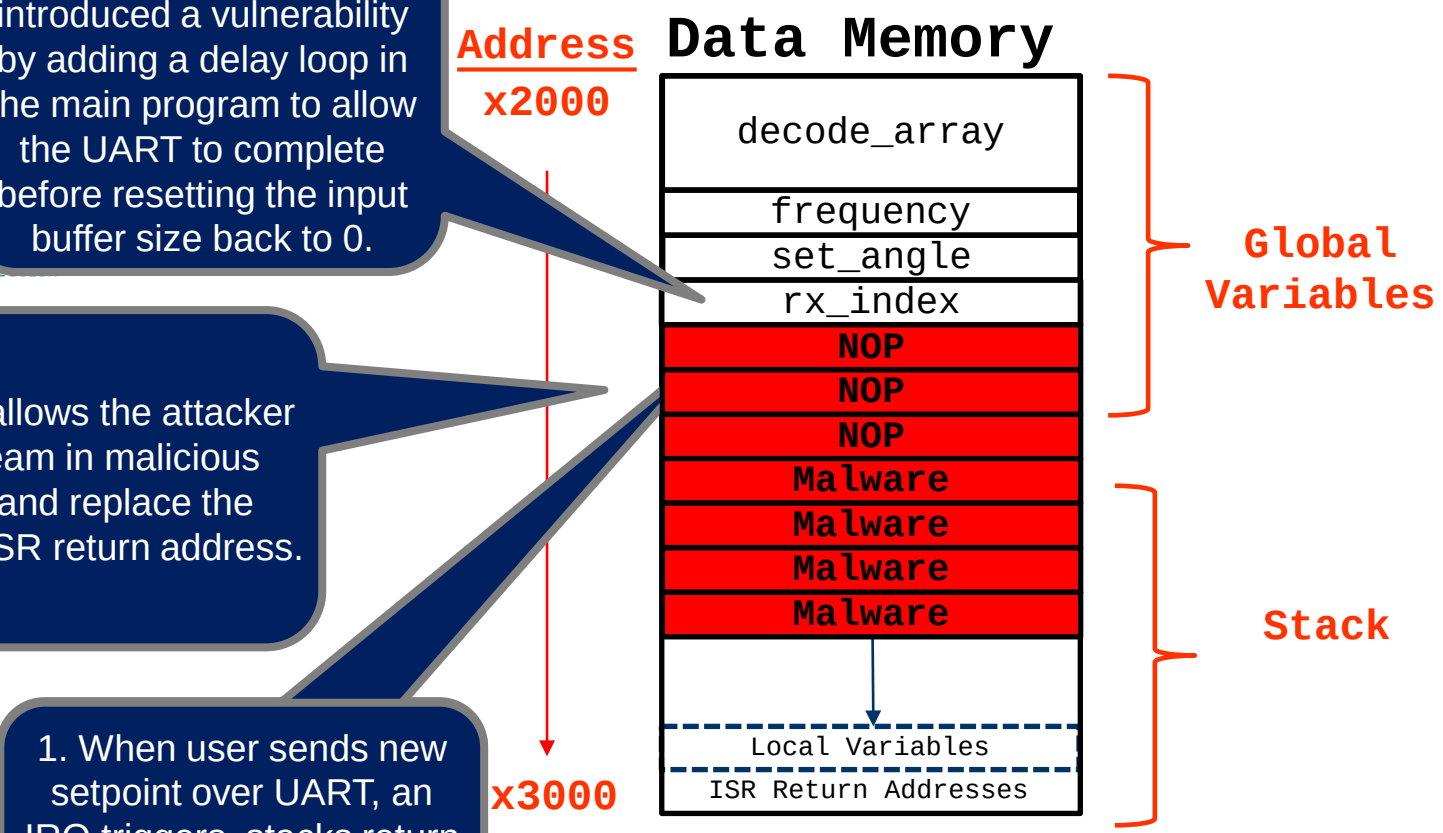
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

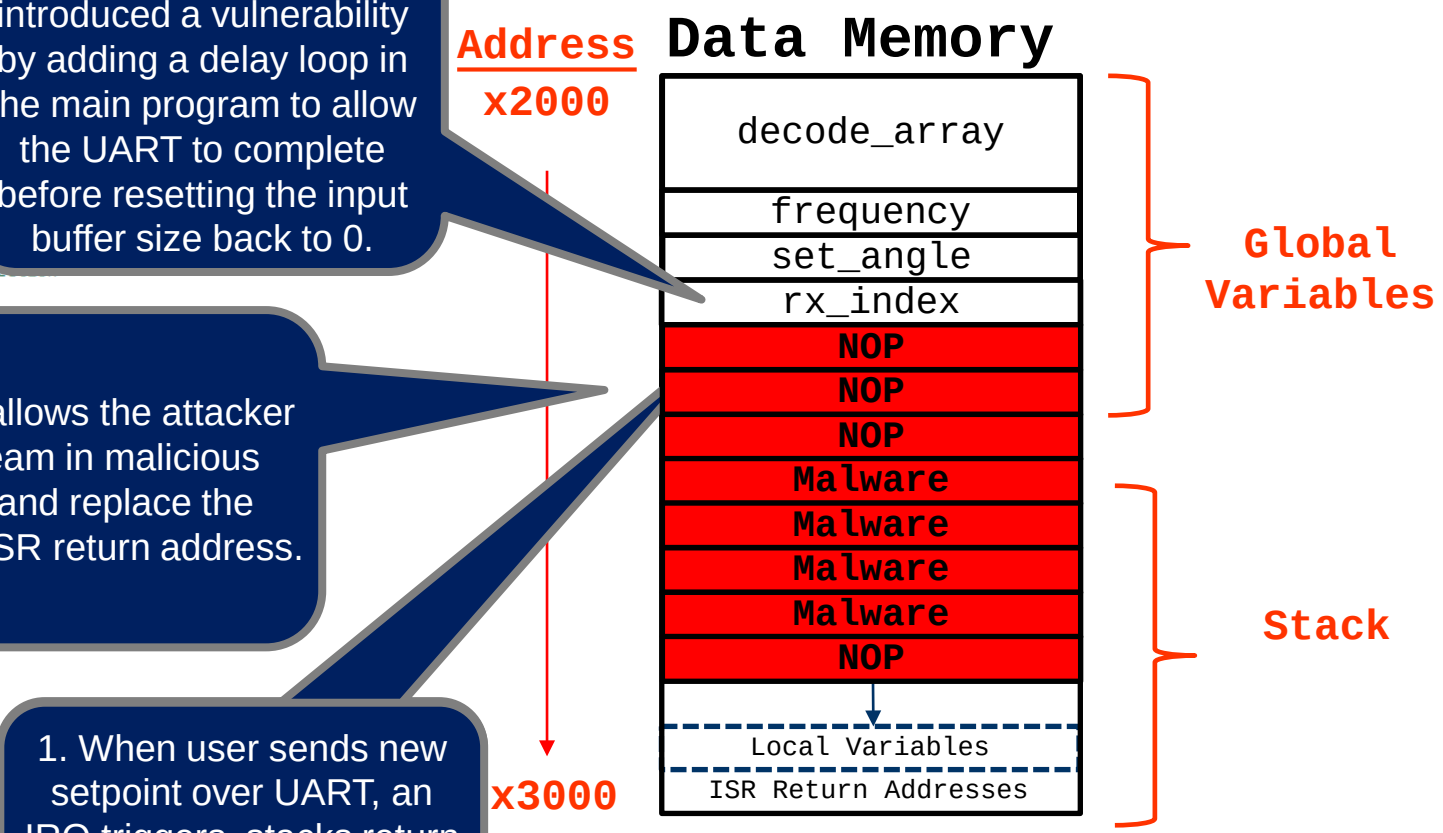
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

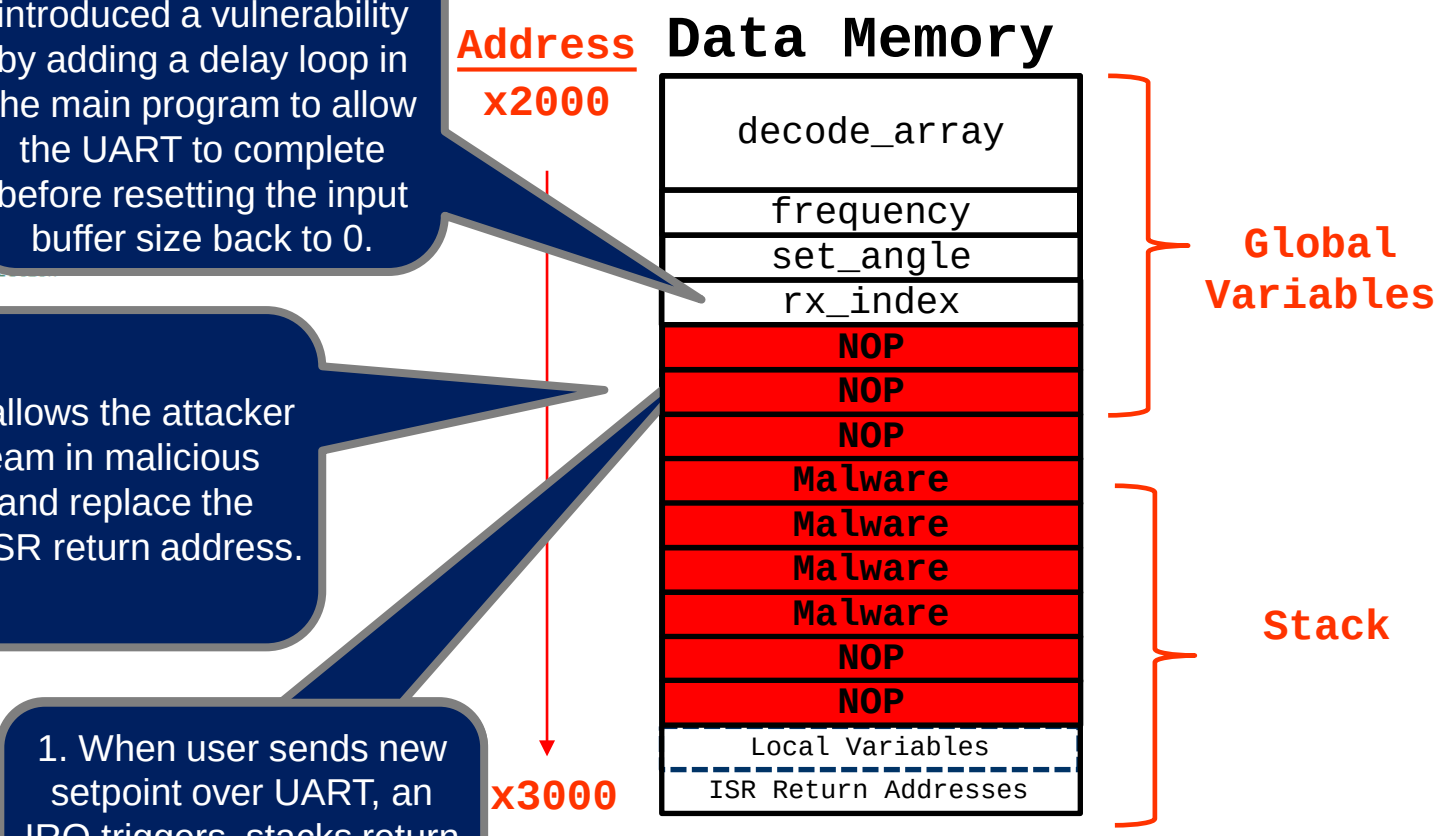
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

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.

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

Program Vulnerabilities (Classic Buffer Overflow Attack)



Demonstration Under Attack

MONTANA STATE UNIVERSITY

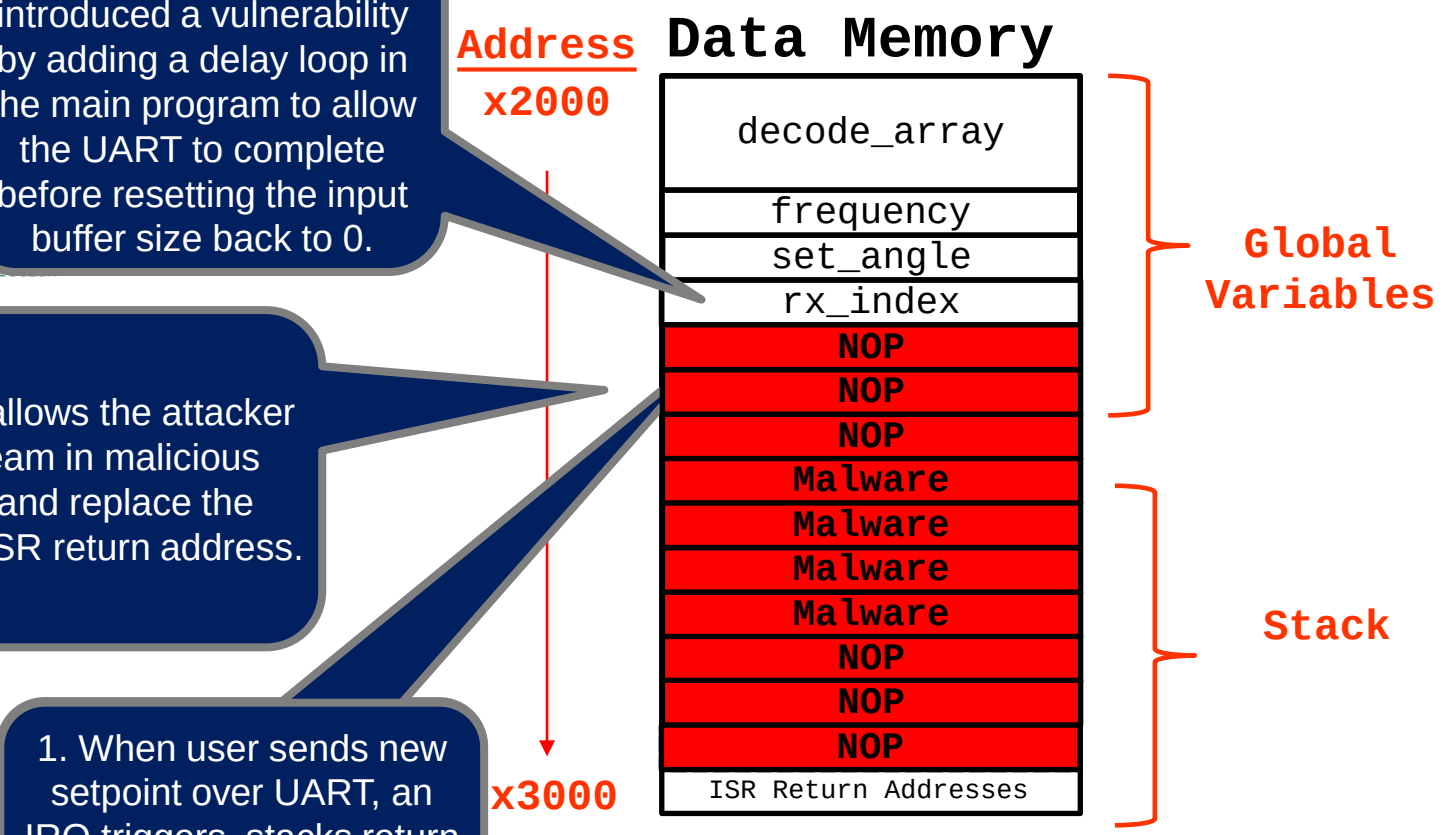
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

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.

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

Program Vulnerabilities (Classic Buffer Overflow Attack)



Demonstration Under Attack

MONTANA STATE UNIVERSITY

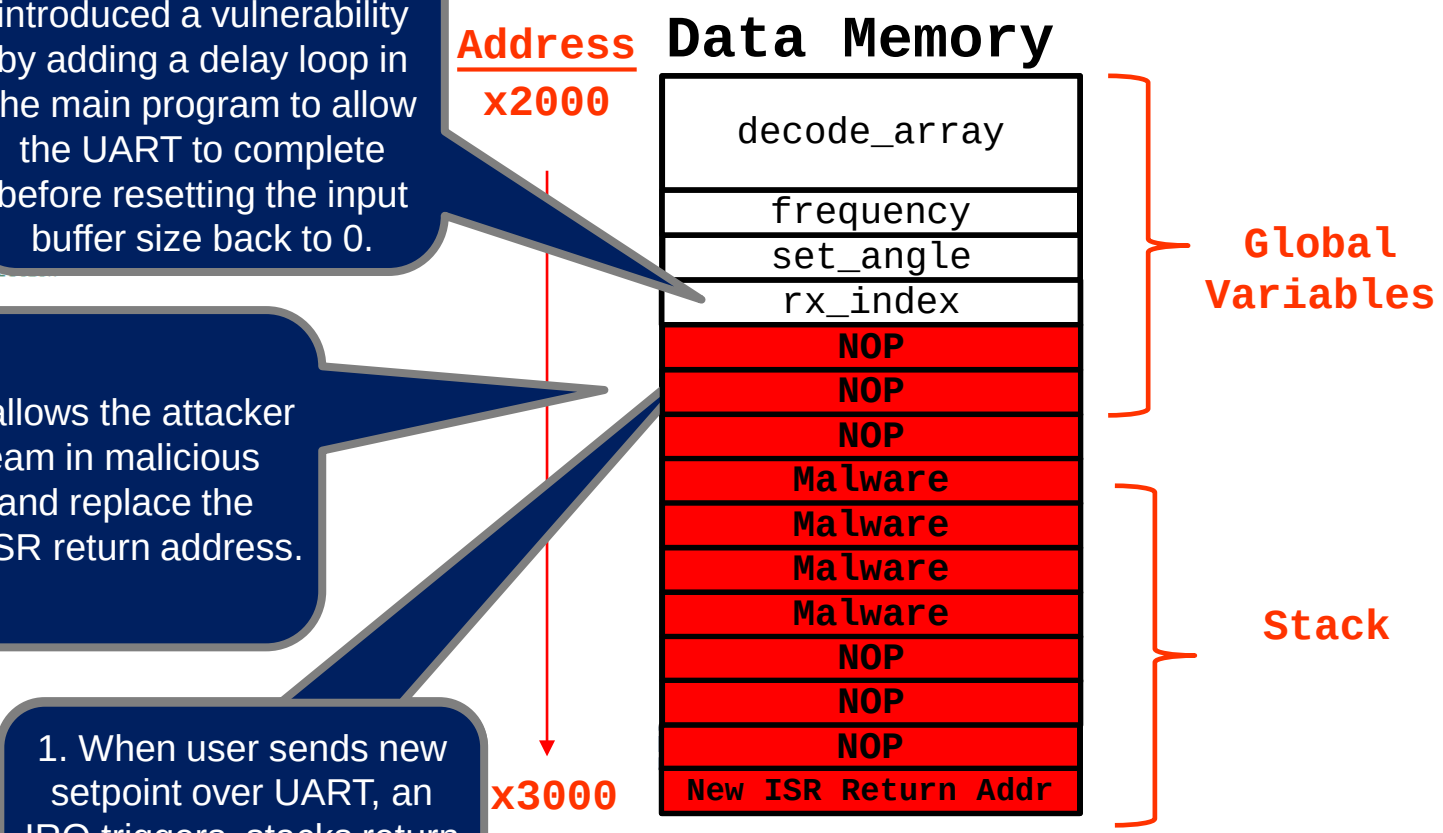
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

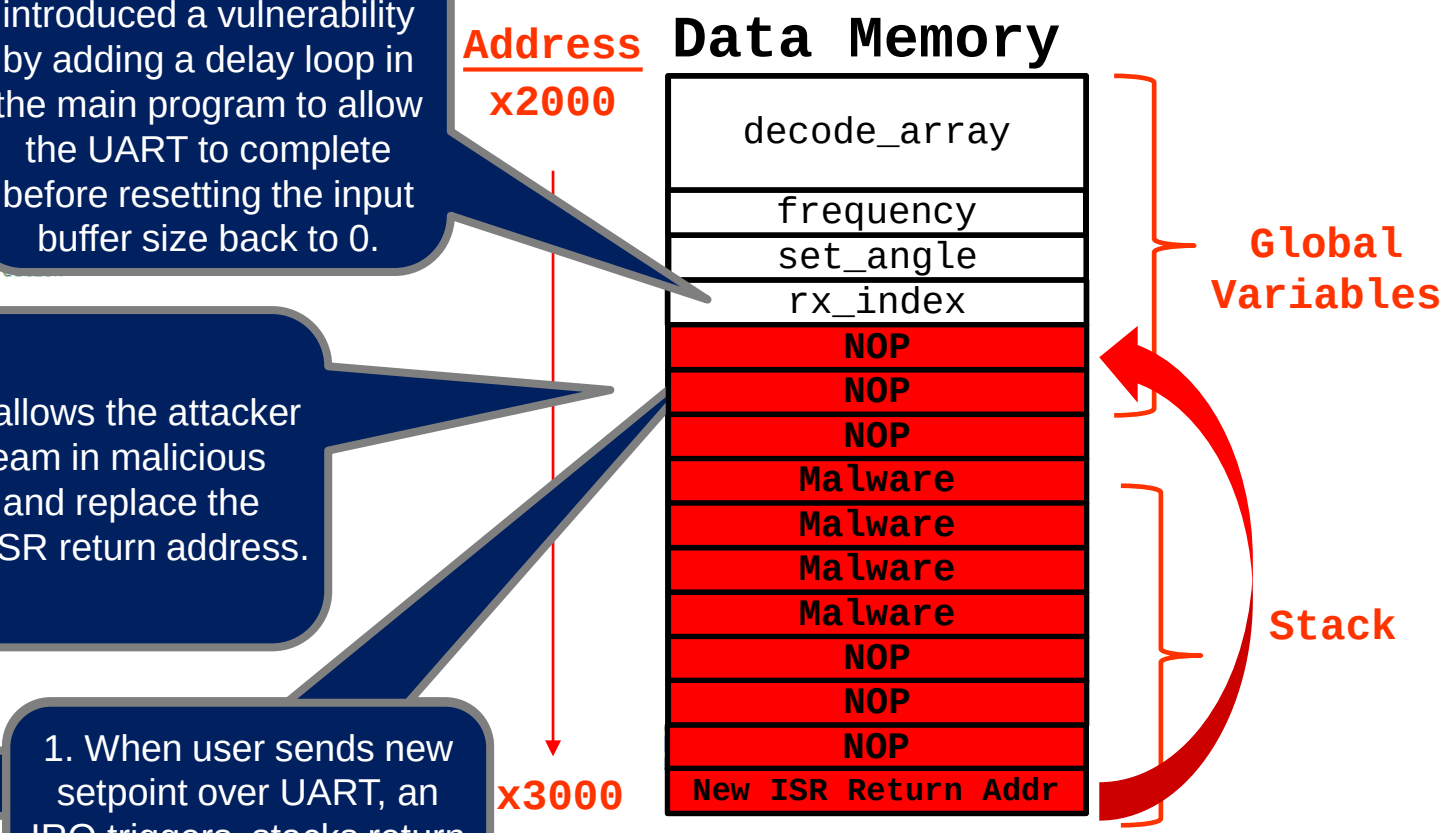
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

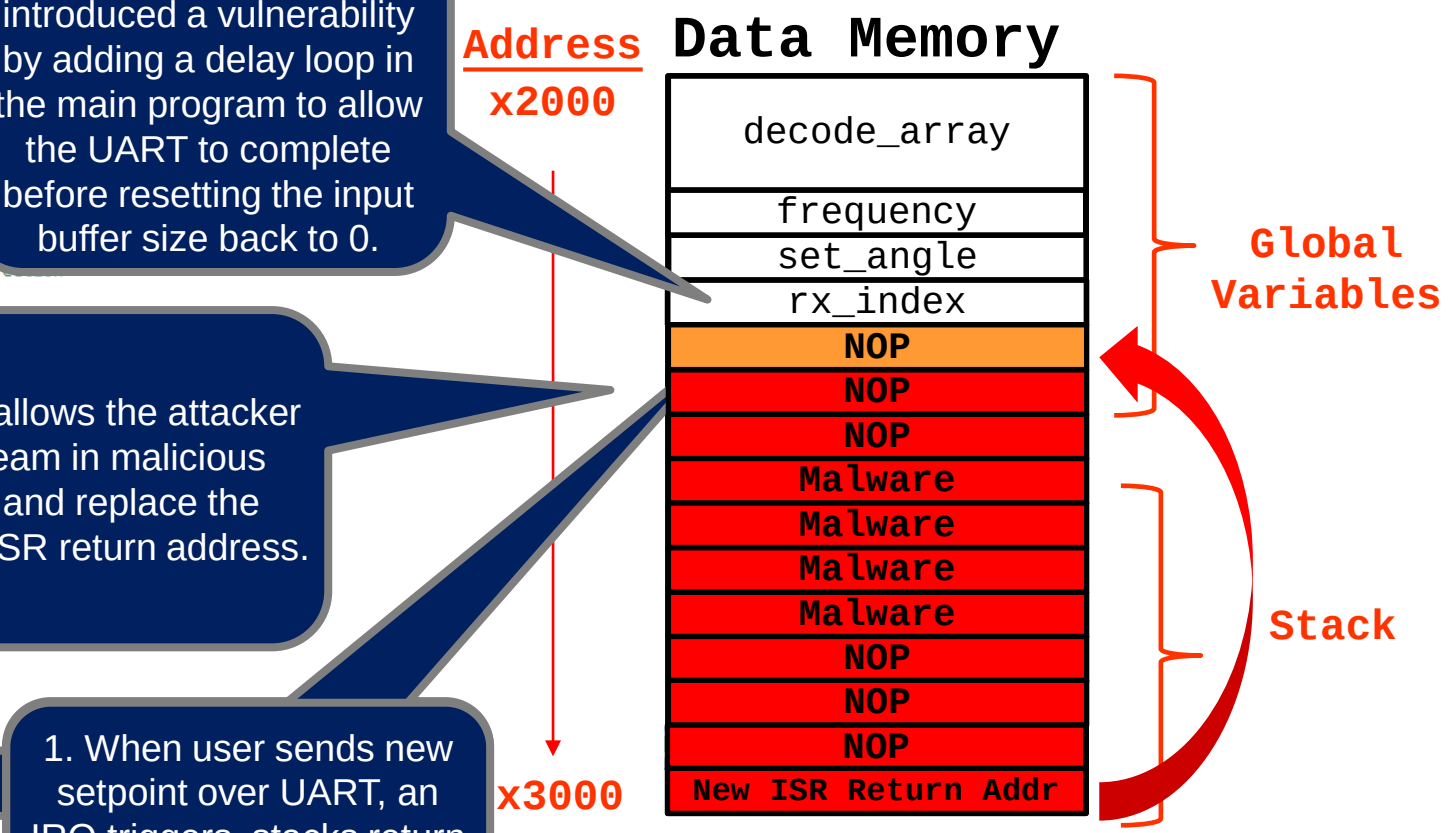
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//      P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//        P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

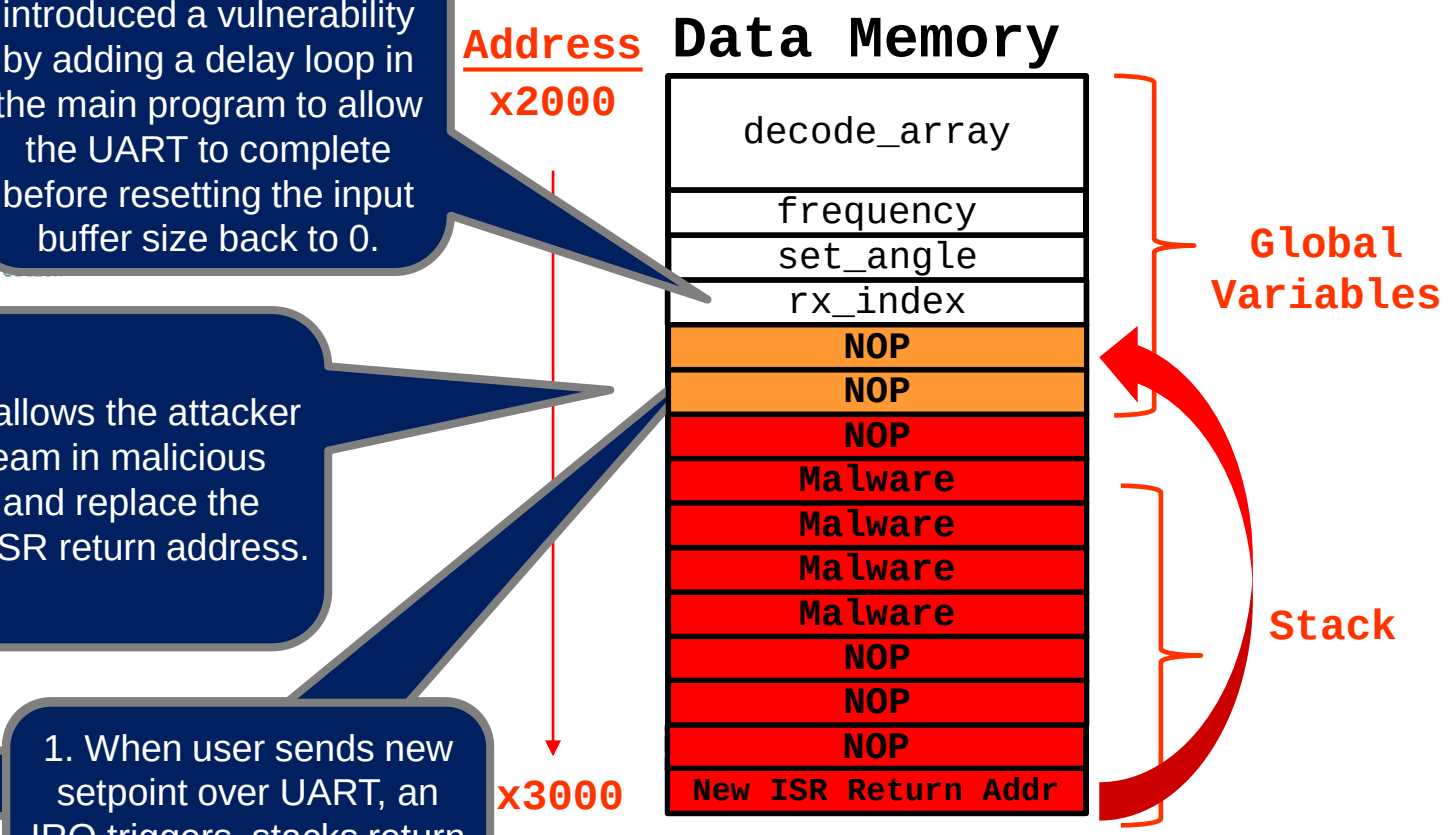
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

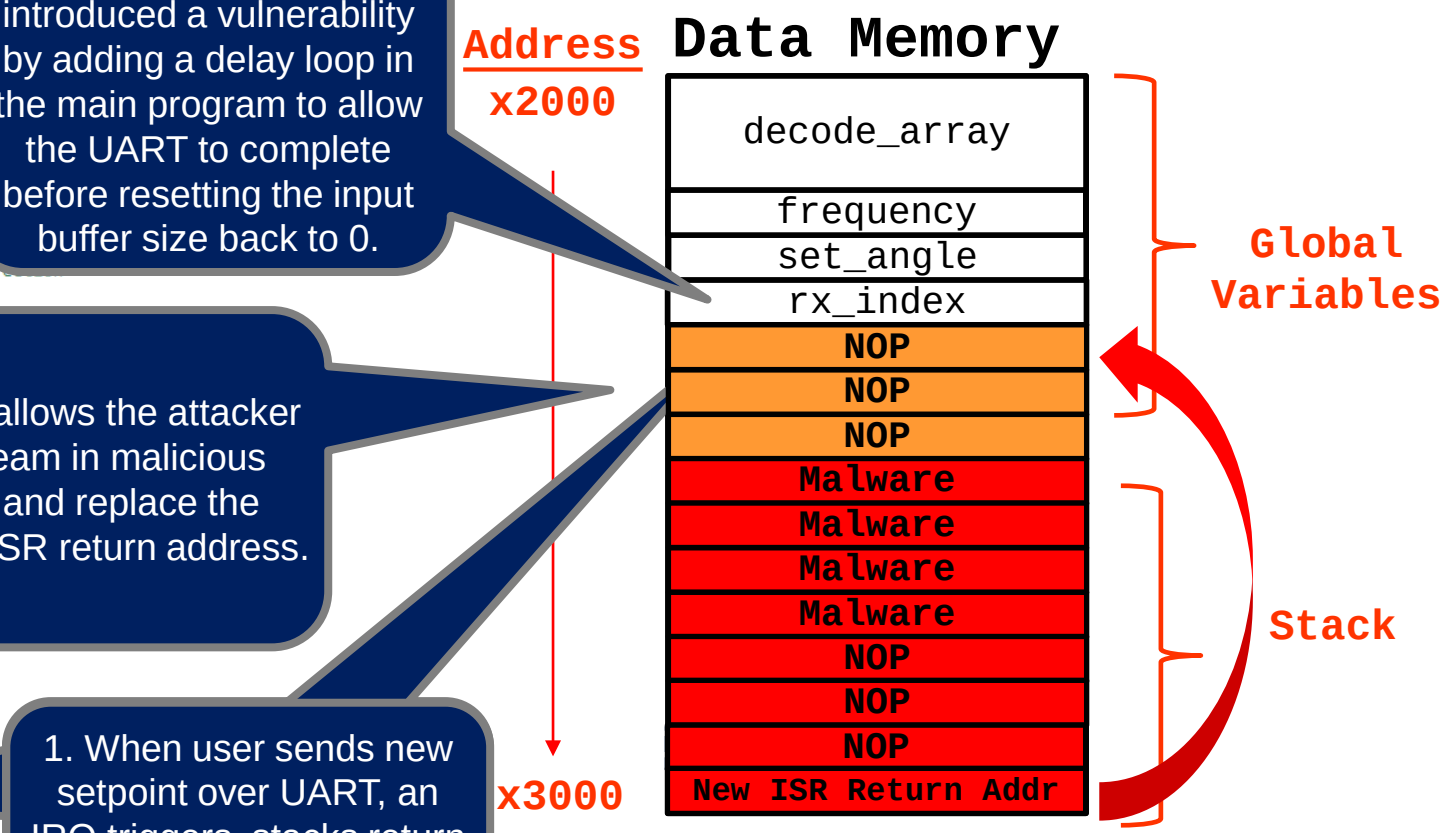
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

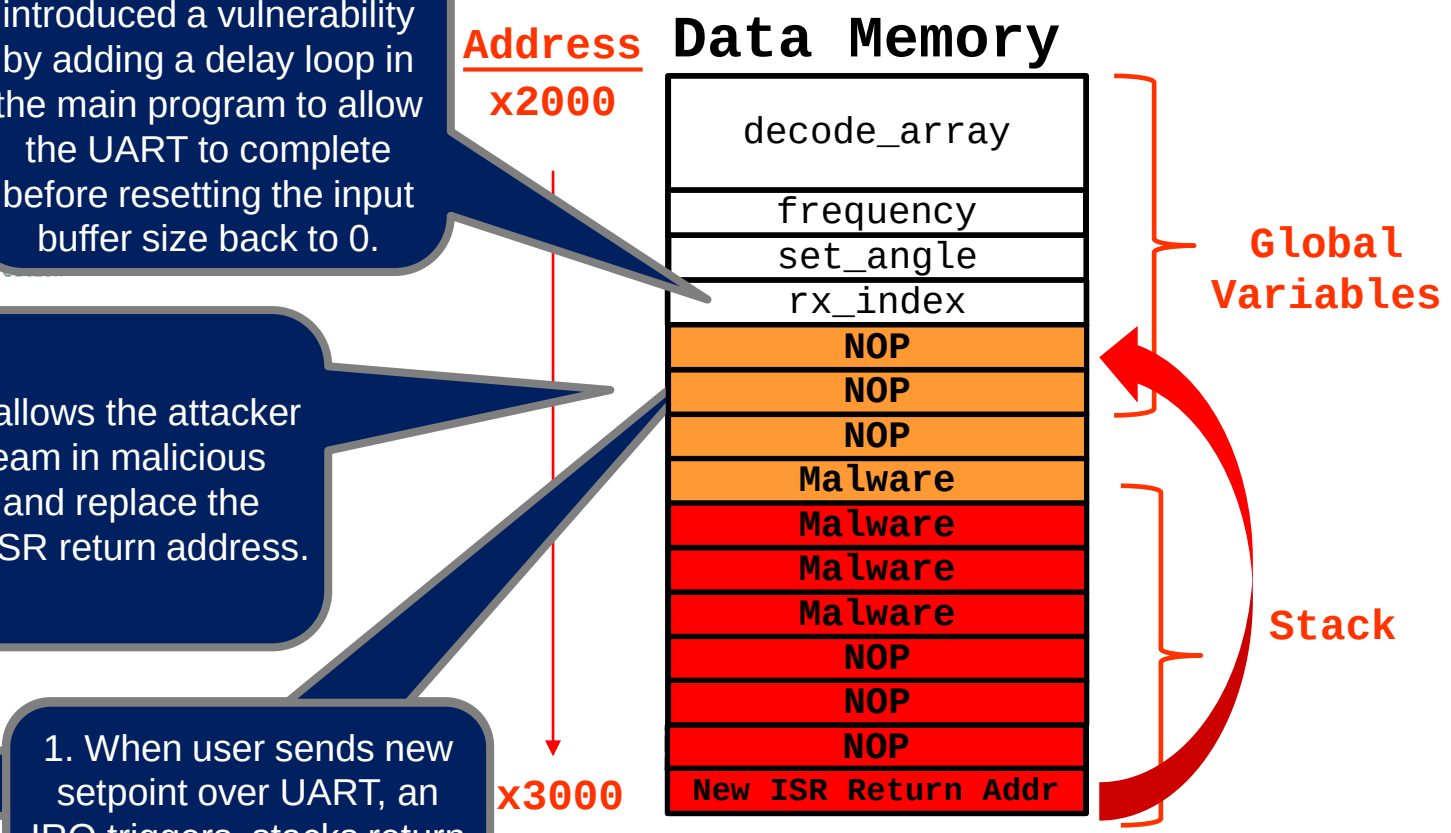
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

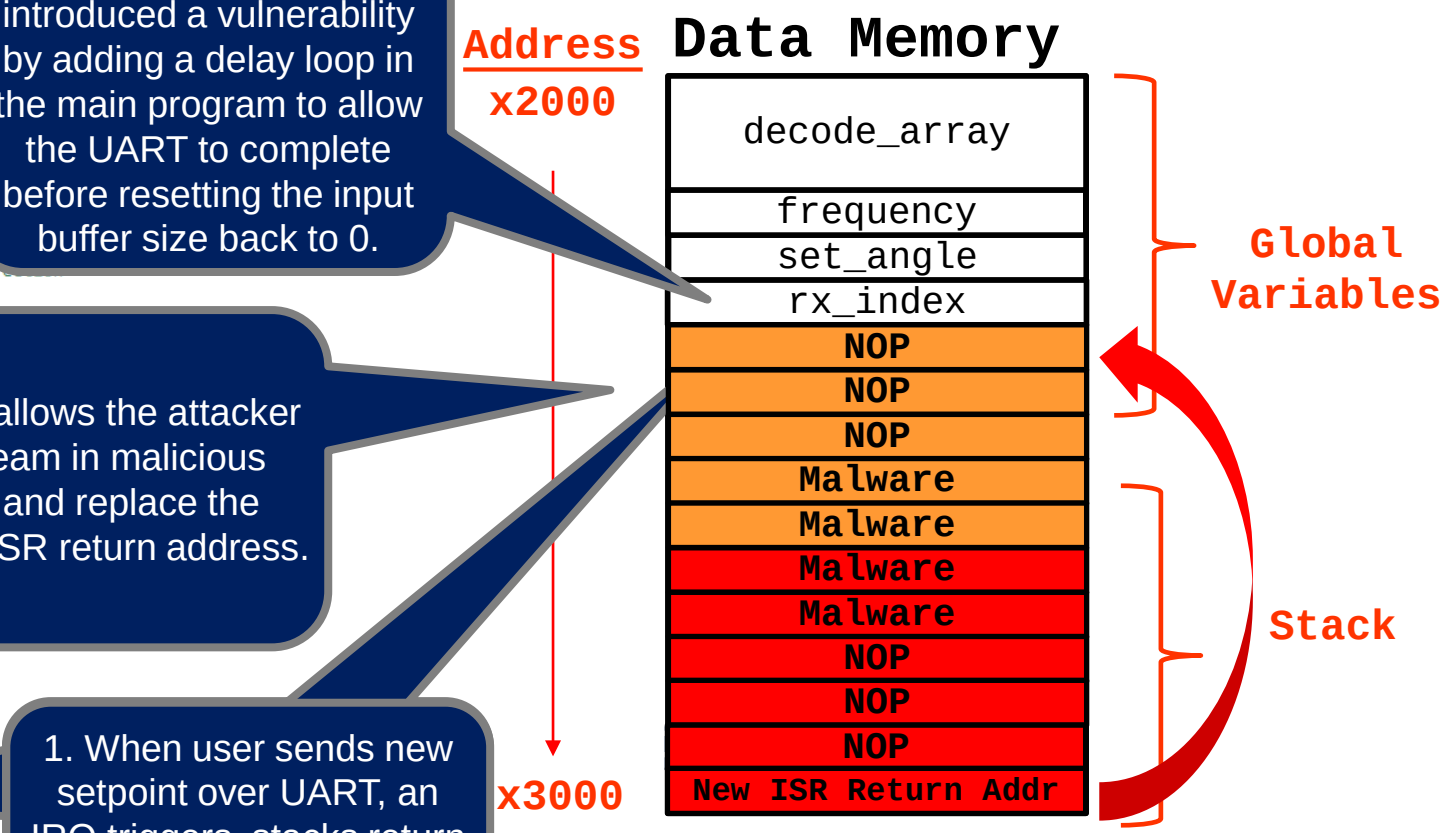
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

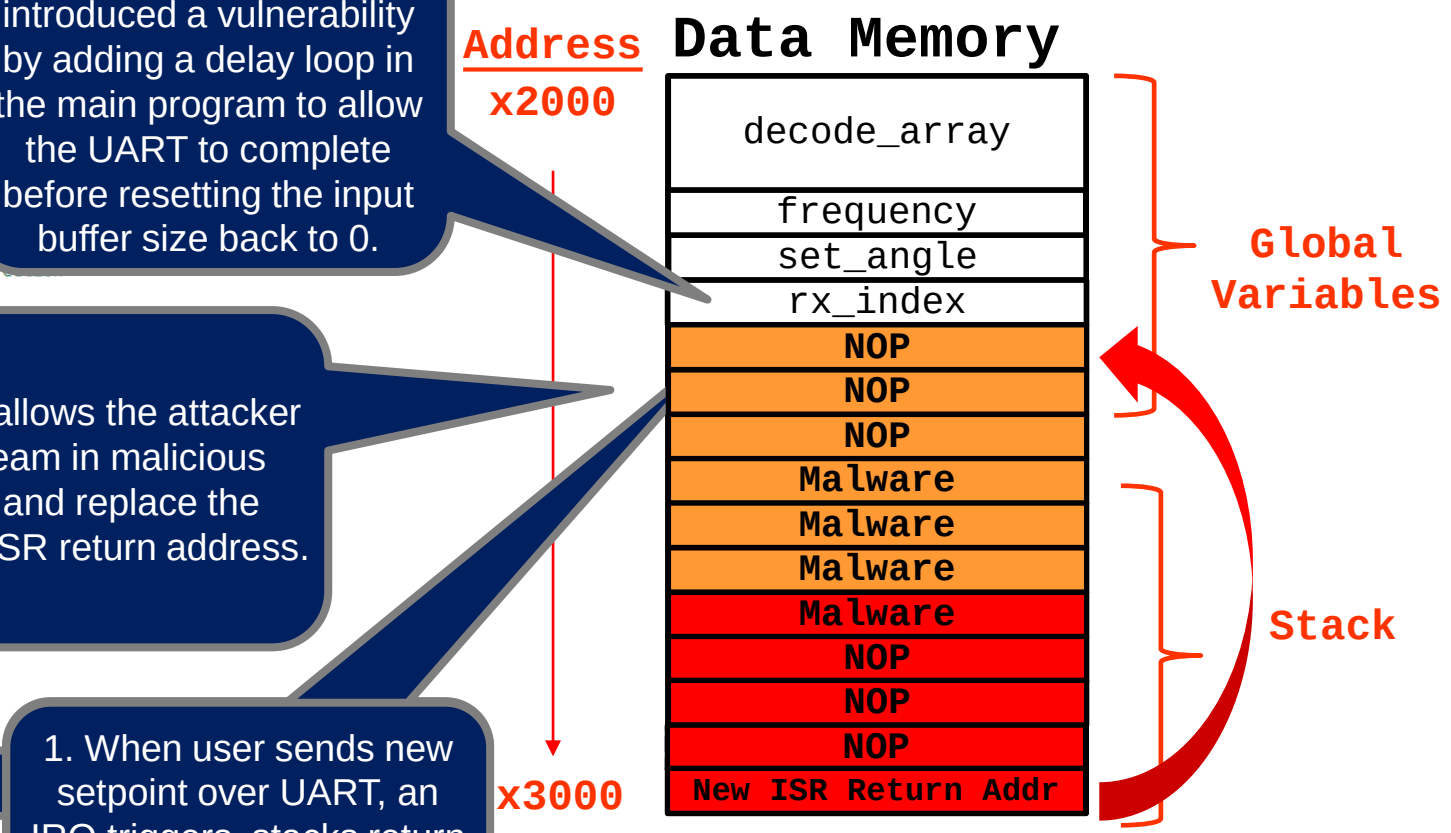
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//          P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//          P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

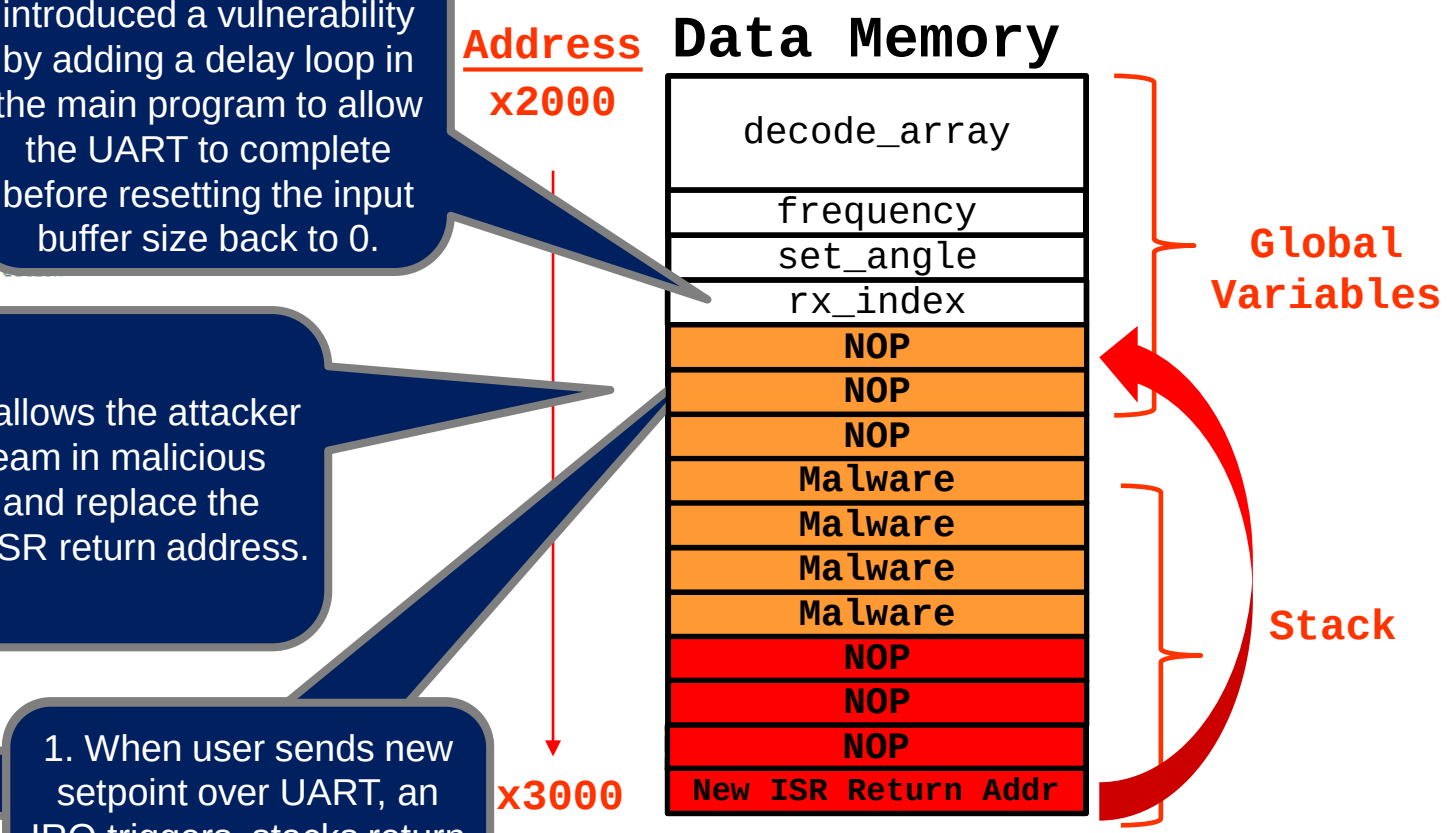
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

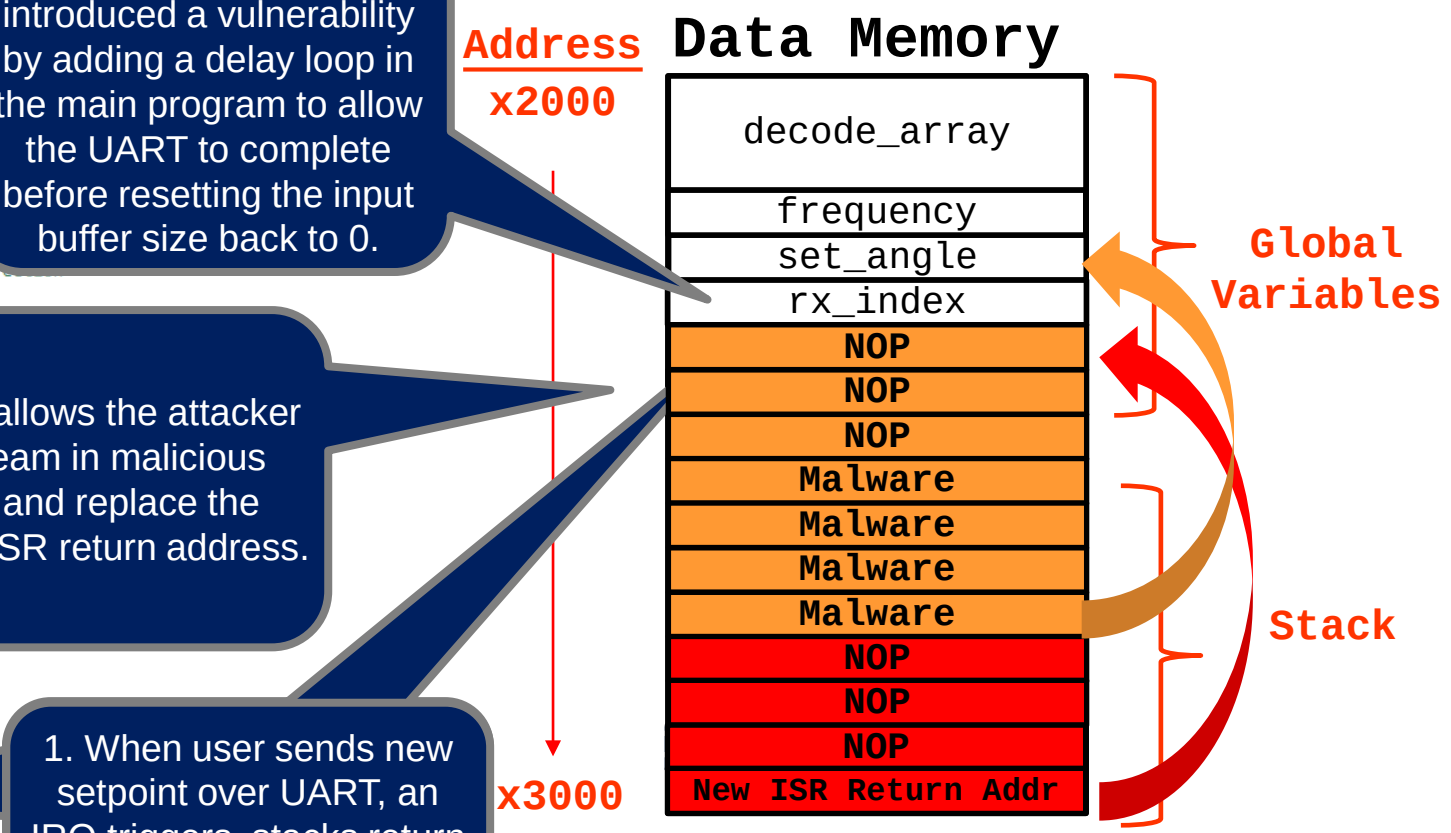
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//      P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//        P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

```
while(1){
    for(index=0xFFFF;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[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//      P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//        P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}
```

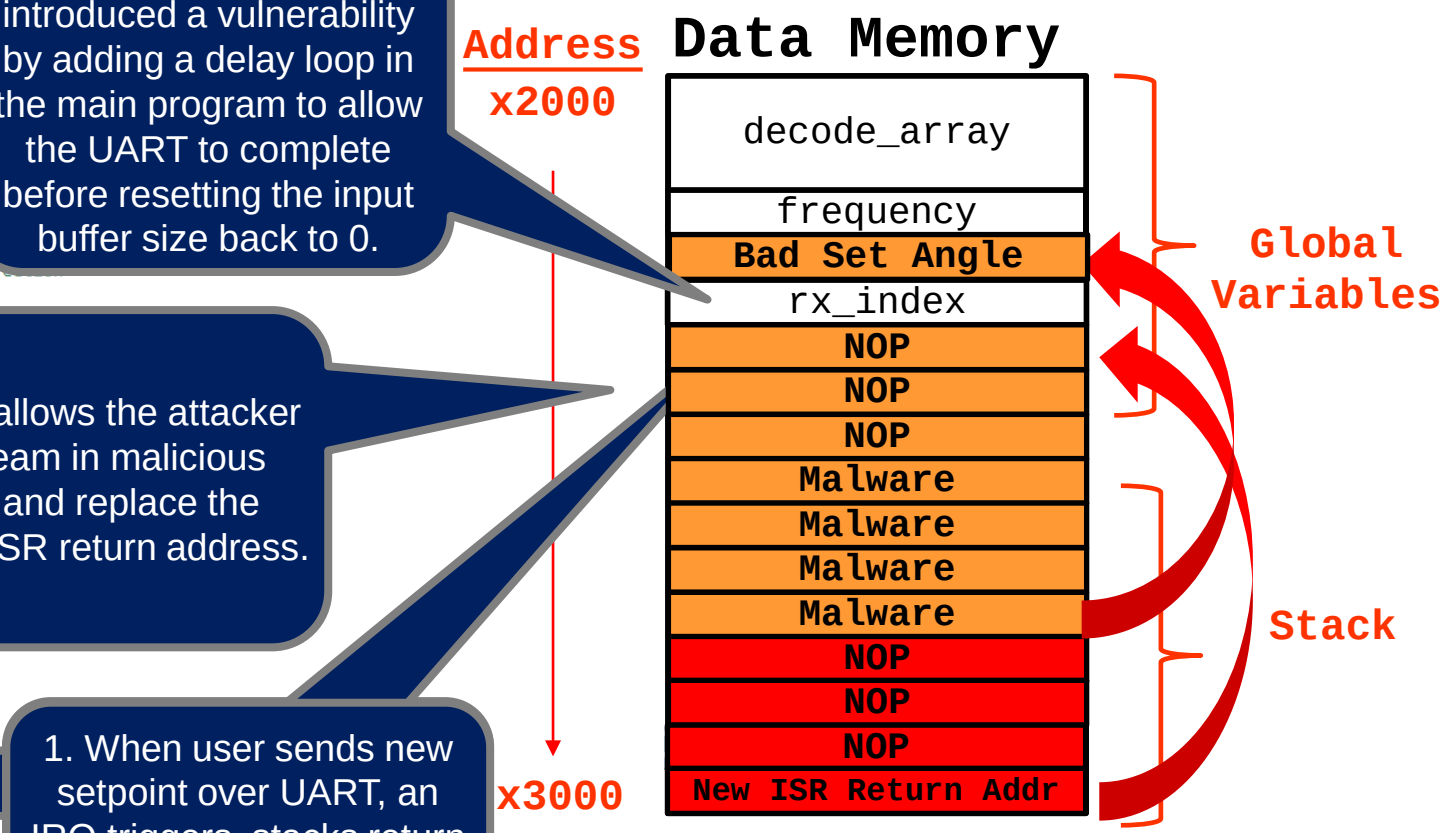
```
// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



Demonstration Under Attack

MONTANA STATE UNIVERSITY

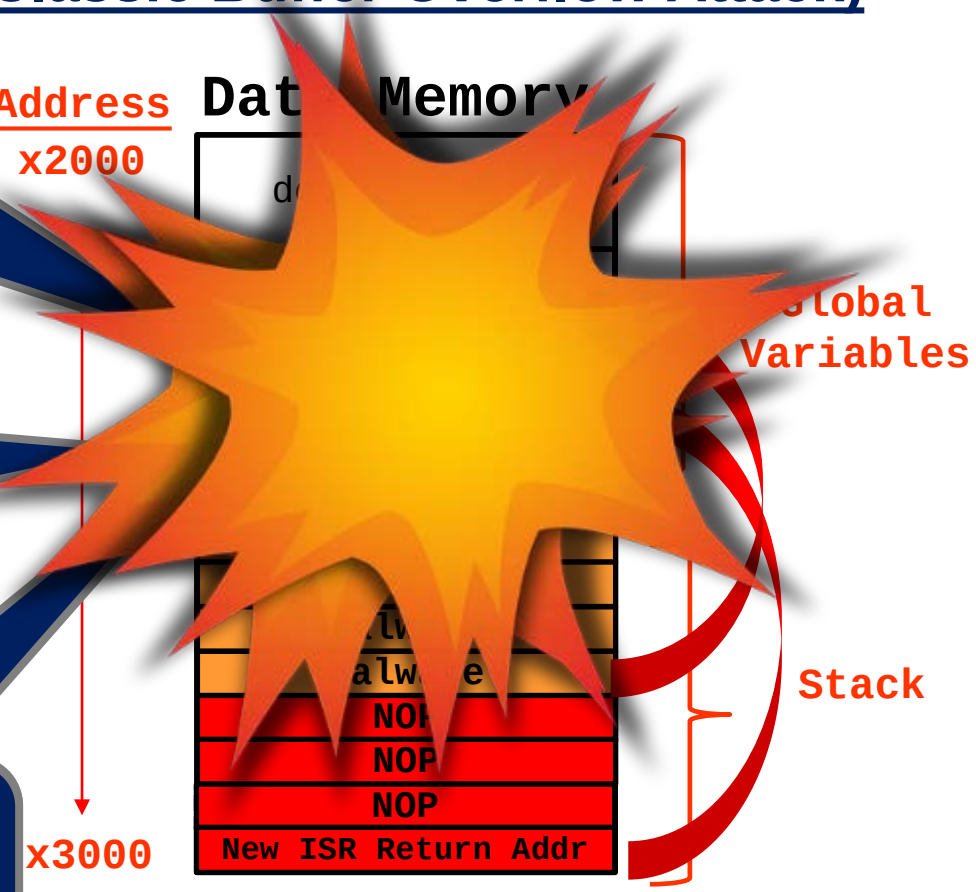
```
while(1){
    for(index=0xFFFF;index!=0;index--){
        _NOP();
    }
    temp = RXBUF[0];
    if(temp == '1'){
        set_angle = 47;
    }else if (temp=='2'){
        set_angle = 79;
    }else{
        set_angle = 61;
    }
    if(rx_index == 1){
        rx_index=0;
    }
    temp = decode_array[P1IN];

    if(temp<set_angle){
        P2OUT &=~(BIT2); //ena
//      P2OUT |=(BIT1); //set d
        P2OUT &=~(BIT5); //set dir
        temp = set_angle-temp;
    }else if (temp>se
        P2OUT &=~(B
//        P2OUT &=
        P2OUT |=(B
        temp = tem
    }else{
        P2OUT |=BI
    }
    frequency = 40
  }
}

#pragma vector = TIMER
interrupt void Timer_I
    TB0CCR0+=frequency;
    P2OUT ^=BIT4;
    //frequency+=1;
    TB0CCTL0 &=~ CCIFG;
//    TB0CCTL0

}

// Service UART
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    RXBUF[rx_index++] = UCA1RXBUF;
    UCA1IFG &= ~UCRXIFG;
}
```

Program Vulnerabilities (Classic Buffer Overflow Attack)

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.

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



MONTANA STATE UNIVERSITY

MSP430 Attack – How it looks in data memory…

```
0x002000  decode_array
0x002000  3800 3728 0018 3427 3908 0000 0017 0D24 0078 3629 0000 3500 0007 0000 1314 127D 6968 0000 6A19 0026 3A00 0000 0000 0E25 7677 0000 6B00 0000 0004 0003 6C6D 0102
0x002070  7574 0000 0073 0000 5E5D 005C 5F72 0071 4748 4400 0049 1D00 4600 4500 0000 2223 0079 007A 004A 1E00 0006 007B 0000 117C 0000 4300 001A 1C1B 3B00 0000 0000 0F00
0x0020E0  3D64 4265 3E00 3100 3C63 [illegible]
0x002100  frequency
0x002100  009D
0x002102  set_angle
0x002102  003D
0x002104  rx_index
0x002104  0001
0x002106  RXBUF
0x002106  8C31
0x002108  index
0x002108  7F97 F680 EDAA E19A 4C9A A300 8D1A 908A 5419 9718 1B89 95E2 C430 D690 6D81 84E0 E714 C184 5E99 B1D3 CEB9 A1DA 7A8D B0B0 4B08 90E1 C3CC A150 680A 8BC0 16BF E452
0x002178  F6B8 8712 10AF C3E8 7ECC 4FD1 56AA 06FF EE0F 8490 577B 92F9 D69F 24C0 EEFB 3670 E25C 8599 5CDA A106 C6A2 2D48 D2DF 2162 C58A 24C4 D59B DB6B F434 FF0E 5FB8 888E
0x0021E8  C69F B6C0 FC9C 0105 2054 2ED0 BC9C B1EE EECA 9211 D7F5 F501 44C9 D483 1D8B A06B E6C8 178A FD8F F460 CE91 4680 5B9C F550 434F F7E0 41D5 224F CA5C E8E4 D0D9 ECE6
0x002258  D517 1529 3A9B C8C0 CF60 6FE1 DC9D C841 CCDD A2A9 18D9 A903 5C1F C7A9 97D3 B041 AE9C A7C9 CCDD A842 C608 C682 C29B DCC0 64E4 1148 EB9D 2CD2 9922 8598 DE97 8A42
0x0022C8  C408 C51C DF99 2481 C269 04A8 89DD 1D40 822E 43AA C1DD 8143 90EF A52A C5BA 0767 0CCB A269 97CB 44CE E54E 0469 12D3 F44B E45A 2180 12DF D04A 8798 E58C 507F D21B
0x002338  F89C 8071 71DB C141 CCD2 24CC DBD9 435B F53F 9B89 D6D1 3244 86E7 8189 A491 BC26 CEDD F1C1 72EF 957B D6B9 0561 618F 9ECA 8698 93C1 499D AAA2 E5F8 C540 5A99 00E3
0x0023A8  C63D 8BB8 329B D455 6627 11C1 5A81 8152 4799 11C1 2AF1 2378 E410 E5C7 D0D5 9D70 CA76 6101 FB8E C4DA AE8B 95F0 4398 8C43 E6AF 9730 5A9B 8D53 6016 D128 C18C B872
0x002418  4B12 C38C 74DF B3D0 7609 D7CC 784E DB40 C569 D488 D1DD CC42 C2FD BE89 549D C942 C017 CD93 5474 C162 E66F C4C9 45DC 9362 C7B8 1563 425F F641 4EF9 428A 71AB BC43
0x002488  85C7 D483 610A 8F6A E04E 0219 D109 C70B E55A 8730 429B 8401 CCD9 81E1 A58B 044A A073 D9C3 9D89 C269 DC2B 9421 37DD 20DA C5FA E2D1 79F9 8719 604A C599 43AE EC0A
0x0024F8  F606 A818 121D C848 C442 A750 5B3A CC48 F4D7 6541 3699 2402 E754 1500 519D 704C 935C A251 668F C04B F4D4 70A1 A4FE A842 EE18 26D9 14BA B140 E07F 260B 5919 C44A
0x002568  8193 4109 5D1F D1C8 47D3 A158 7A9B D64A C49C A54A F20B D568 4E08 8582 3183 AE41 4E50 CCA8 79DB C850 D4A9 F1A4 72CF D251 4480 B688 1BEA 4982 D44E 64C4 3BDB 8188
0x0025D8  C6B3 E698 7B09 4582 F678 07BC 57AC 2BE0 6FD4 A680 31DC 4BB3 C4D9 A581 9452 2377 E031 5005 7399 4144 E293 09C0 850D 22C0 EA93 0329 7EDD 44EE A416 B0A4 7B89 565A
0x002648  688E E219 76DF 7C59 62DB C08D 36A1 ACCB 62B9 B145 2AB9 2800 B411 1588 2899 7021 C45F 8381 F289 E0B1 A39F D363 3EAF DCF1 A792 9F42 689D 66C9 C692 E62C C0DD EE72
0x0026B8  4C09 B644 84CB A76C 440D F7E8 DF59 E540 6E18 3CF8 9BFC 8493 6807 35D1 D2B8 F1C5 28F5 1D9B 5CF3 2D79 E4DB 8F99 2FAB A561 C2F9 80A9 4FAB 81EC A6DA 85C1 52DF 8453
0x002728  57AD FD90 55C9 6D82 561D 8181 B389 A202 C4CB 6587 75D9 81C0 E40E E58F 2ACD 92C1 C19A C9DA 7ADB A4F2 C4A6 958A CA89 F440 470A A582 129C C6D1 96DF A08D 17B9 9CC4
0x002798  2B0E C580 0289 8C52 C618 85A0 58AB A0C3 C8BE 02F0 6E59 9183 F4EF C4D8 FE99 D8C2 ED08 E488 1883 8742 EC48 9581 7097 8454 EE3B 9580 5AD3 8717 A61B 95A0 1A9B E449
0x002808  53AB 9E3A CC56 A284 8361 D6F9 AEF2 A025 017B D679 5999 9116 0677 DD2B DD2E EF6C AB07 21B0 B462 A848 4FF3 B98C 5AAB DC0A 8427 AE45 F444 DD26 331D F09D 1E68 E922
0x002878  E93D 80CA 58E8 1944 18AF A323 74C6 F7E4 3D1D 290F 2F04 E0E4 06FF A319 A1E5 3FE4 5E57 EA2E E069 36A6 0963 D872 BBCF FD62 9727 1722 2C48 2DC6 49AF CD2D 3C7D 3607
0x0028E8  38AB B87E 41C6 B881 292A FB28 668E 2804 C991 8B0D 354A 3CC0 4E0A 92EF B4EA 7C27 9B4F 9967 BC66 5822 7DCF DB61 E4E8 9500 E7E4 1B25 A1CD 8202 B877 FF6C B5CC 9438
0x002958  5257 BBAE 73F4 A010 E0A7 3D3E 498A 1822 E16F 373F 6DAA B439 933A 4D4F 78AC 6543 FDBF 5BA6 ED8D A421 79E7 BCB8 7E8F BDEB 2A0F 2F76 1086 A56C 803F 7337 80A2 852D
0x0029C8  523B F0A3 B39B 25EE 2B0F D951 4860 A806 1D35 CA08 D0CB 14B1 0095 4A9A 1EF8 F429 2867 930B 68E8 A010 2526 ACE3 6EF9 3C2A 2A17 B234 2499 8558 29B6 5AAA 20F5 EC08
0x002A38  375F BB23 A032 210C 91BE 566E FEC5 9C80 BC65 D336 2B0B A1EB B225 D2B0 B2B5 CE15 211F E652 D678 F71C 316B B930 1D3B BEC4 3EB7 7938 04EA 39E7 E237 9879 9729 BC76
0x002AA8  21A5 C579 ECE3 8C02 59E7 D319 08B7 7456 0505 D73F D8BE 900C 063F 9626 048E 9848 8ACD 1F21 729F F54C B4BB 5F38 DCF3 7B32 25AD 397E 14E3 34D0 8B1D BA2E 01F9 AC26
0x002B18  13F7 A10B AEC6 E47A 2662 CA03 4EB1 B400 042D 5639 5CBE AC0E 22B5 B92F 0FFE 5A4C 6AAD B935 51CE 2F6E 21A9 CBA7 6DC4 9E47 202E 4F39 6ED4 EB45 B92C 776D FEDB 7B54
0x002B88  3A9D BC2E 30E4 FAED 403A D063 8870 1206 2128 712B CB6B FB61 0407 937B 6549 A737 D037 7A04 C7B1 CE0C B723 3F22 2AF2 8C04 4EB7 6200 2FC8 9E06 8727 F25C 6FEA DA3C
0x002BF8  08A2 1B28 05E5 4A21 D7F0 FF1C EDA5 80E9 D819 D807 6C9A 8A23 09EF 235F 7FDD 4E27 9B99 B20B DCD0 4F08 DFB5 AB46 16B2 E328 143F 59A3 1E22 BC1C 43A6 9BE9 B9B3 5D60
0x002C68  02F9 F010 C44C A504 CF0F B1C9 E413 E006 A4B5 F429 ED56 6B26 A7BC B827 FC4C EE20 7EE4 DB6B E8BC 2E23 5D77 CF56 D428 3887 86E3 D376 E06C BDA3 0307 A42A B6EC B823
0x002CD8  2E61 FB12 660C 2204 06A5 143A B230 268C 53C5 7725 A5B5 2606 94B7 B229 0CED A3AD 5052 F32A 65C5 9240 BE1D 9178 05E9 B6A8 8CEF EB30 0DE5 3684 EDD7 E3B1 A4E9 B40C
0x002D48  4B28 BE33 0F09 161F C939 9EA1 1F46 9C6C 162C 0205 8CB4 E428 90F9 9AB2 F68C 0564 04B3 332A 7CEE 204C 0F6F BCE0 1C9A C4C0 B181 FC37 EC86 BF50 32C3 991D 3BDA BD72
0x002DB8  62E4 B279 FDED F872 BE70 FB4D 7D48 E888 6679 39EA 1B44 60A0 8F7B 2158 74B4 6E05 94F0 AB38 05AA 4080 05BB 93FA 50DD CA04 1FF1 3324 C54D EB44 BE7B B09C DD87 CE05
0x002E28  A66F E78A 3F39 F01D A527 893A 3BAA E836 FDA7 5BBE [illegible] 0494 6504 8CE5 93DA AC94 8225 851D F5A9 B056 9A32
0x002E98  96FD FE20 7099 ECD8 E503 CB21 7488 F848 1337 4970 [illegible] B4B7 9E80 6FB9 686C 75BC DE84 376D 4065 3871 6448
0x002F08  4CA2 C102 88D7 F2E9 85BC 4BA0 FDC0 3826 32B7 9F33 [illegible] B2E0 BD73 468F 63A7 24FA 3887 E4F7 4622 CD6C 3B08
0x002F78  4C06 586D B91B BC04
0x002F80  _stack
0x002F80  FDA3 BA67 FADE CC97 196E 606F 6FFF 1F96 [illegible] 3567 46F7 3C70 3C37 DA0A 3C60 9866 E6B6 7154 7C04 1C96 C6F6 9C6D 9DA6 BC2A DDF6 5B6B 1CCA 3C86 C179 F565 59CC 9C06
0x002FF0  824A 0000 810A 003D 009D 0009 8176 [illegible] 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
0x003060  3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
```

The vulnerability

What the attacker is after

UART ISR Return Address

MONTANA STATE UNIVERSITY

MSP430 Attack – How it looks in data memory…

```
0x002000  decode_array
0x002000  3800 3728 0018 3427 3908 0000 0017 0D24 0078 3629 0000 3500 0007 0000 1314 127D 6968 0000 6A19 0026 3A00 0000 0000 0E25 7677 0000 6B00 0000 0004 0003 6C6D 0102
0x002070  7574 0000 0073 0000 5E5D 005C 5F72 0071 4748 4400 0049 1D00 4600 4500 0000 2223 0079 007A 004A 1E00 0006 007B 0000 117C 0000 4300 001A 1C1B 3B00 0000 0000 0F00
0x0020E0  3D64 4265 3E00 3100 3C63 2500 0000 3000 524D 4145 3546 4000 5162 504F 6061 0070
0x002100  frequency
0x002100  009D
0x002102  set_angle
0x002102  003D
0x002104  rx_index
0x002104  0EF8
0x002106  RXBUF
0x002106  4303
0x002108  index
0x002108  FDB4 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002178  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0021E8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002258  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0022C8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002338  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0023A8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002418  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002488  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0024F8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002568  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 C392 0130 C232 4303 D3D2 0224 D3D2 0222
0x0025D8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002648  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0026B8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002728  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002798  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002808  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002878  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0028E8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002958  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0029C8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002A38  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002AA8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002B18  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002B88  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002BF8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002C68  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002CD8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002D48  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002DB8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002E28  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002E98  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002F08  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002F78  3FFE 3FFE 3FFE 3FFE
0x002F80  _stack
0x002F80  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002FF0  3FFE 3FFE 3FFE 003D 009D 000D 258A 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
0x003060  3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
```

The vulnerability

What the attacker is after

The Inserted Malware

A NOP Sled is used so that the exact ISR return address isn't needed.

UART ISR Return Address

MONTANA STATE UNIVERSITY

MSP430 Attack – How it looks in data memory…

```
0x002000  decode_array
0x002000  3800 3728 0018 3427 3908 0000 0017 0D24 0078 3629 0000 3500 0007 0000 1314 127D 6968 0000 6A19 0026 3A00 0000 0000 0E25 7677 0000 6B00 0000 0004 0003 6C6D 0102
0x002070  7574 0000 0073 0000 5E5D 005C 5F72 0071 4748 4400 0049 1D00 4600 4500 0000 2223 0079 007A 004A 1E00 0006 007B 0000 117C 0000 4300 001A 1C1B 3B00 0000 0000 0F00
0x0020E0  3D64 4265 3E00 3100 3C63 2500 0000 3000 524D 4145 3546 4000 5162 504F 6061 0070
0x002100  frequency
0x002100  009D
0x002102  set_angle
0x002102  003D
0x002104  rx_index
0x002104  0EF8
0x002106  RXBUF
0x002106  4303
0x002108  index
0x002108  FDB4 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002178  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0021E8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002258  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0022C8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002338  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0023A8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002418  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002488  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x0024F8  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303
0x002568  4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 4303 C392 0130 C232 4303 D3D2 0224 D3D2 022
0x0025D8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002648  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0026B8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002728  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002798  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002808  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002878  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0028E8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002958  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x0029C8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002A38  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002AA8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002B18  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002B88  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002BF8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002C68  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002CD8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002D48  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002DB8  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002E28  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002E98  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002F08  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002F78  3FFE 3FFE 3FFE 3FFE
0x002F80  _stack
0x002F80  3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE 3FFE
0x002FF0  3FFE 3FFE 3FFE 003D 009D 000D 258A 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
0x003060  3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF
```

The vulnerability

What the attacker is after

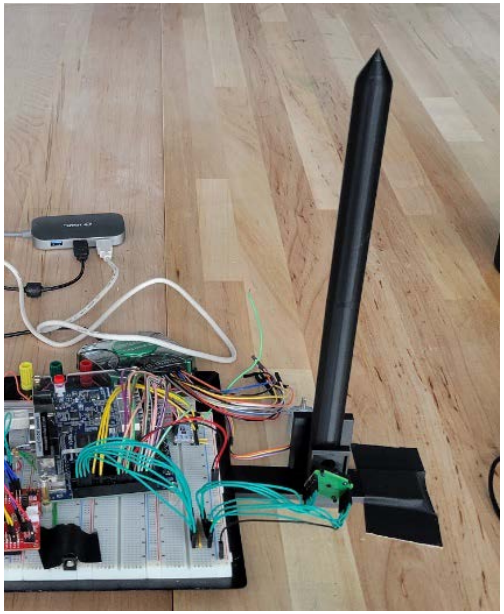
The Inserted Malware

A NOP Sled is used so that the exact ISR return address isn't needed.

UART ISR Return Address

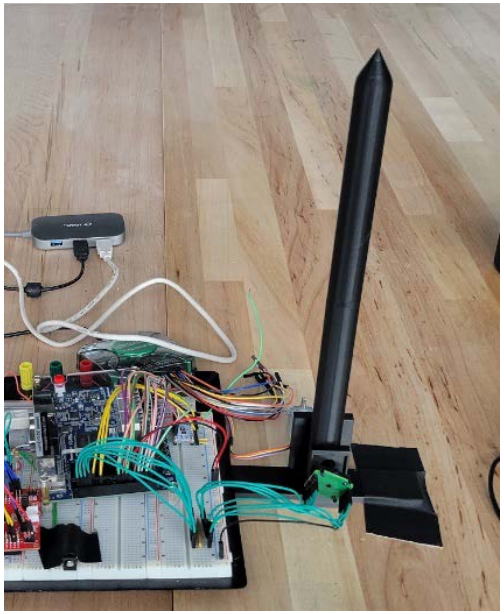
Demonstration Under Attack

The Same Attack is Made on CyberShield



Demonstration Under Attack

The Same Attack is Made on CyberShield

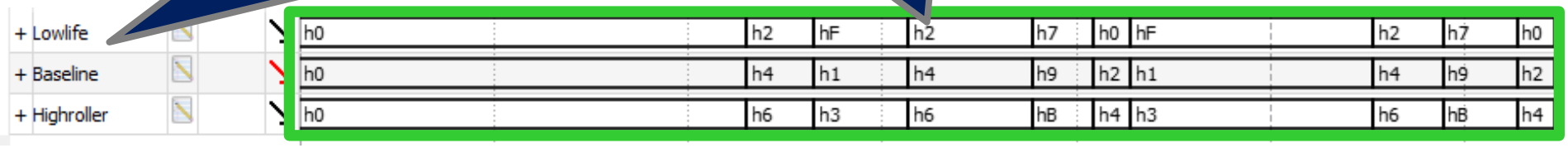


Demonstration Under Attack

The Same Attack is Made on CyberShield

We can see how CyberShield Responds by Measuring the Instruction Registers in the CPU with a Logic Analyzer.

All Opcodes are Different by Design

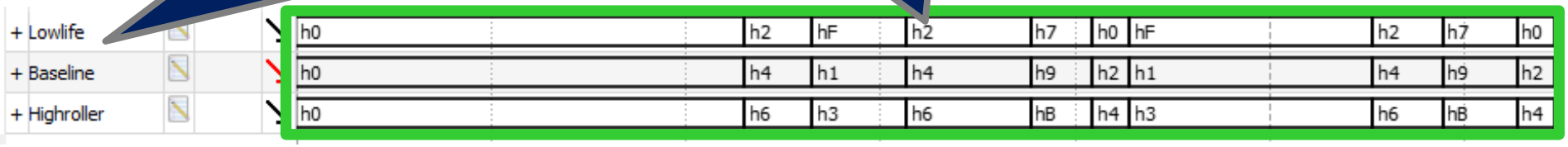


Demonstration Under Attack

The Same Attack is Made on CyberShield

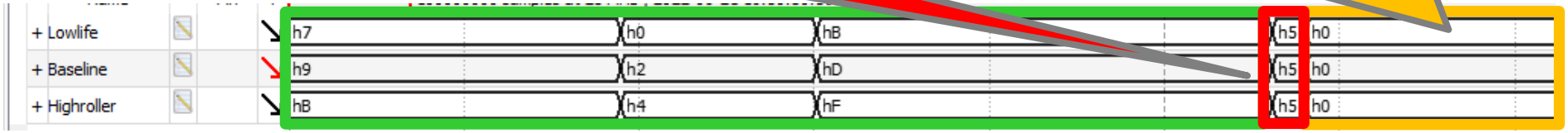
We can see how CyberShield Responds by Measuring the Instruction Registers in the CPU with a Logic Analyzer.

All Opcodes are Different by Design



The attack is detected when all three CPUs see the same Opcode.

CyberShield Halts Operation and Initiates a Recovery Procedure.



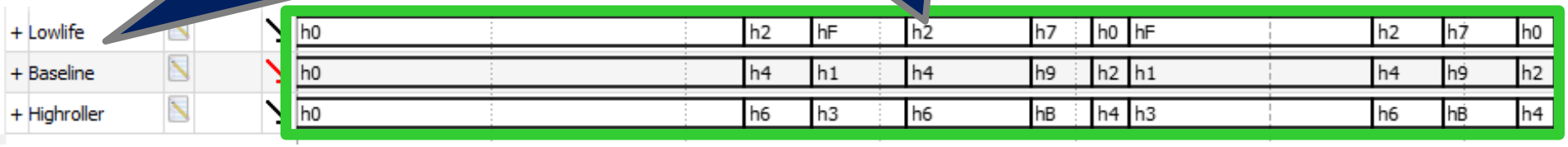
Demonstration Under Attack

MONTANA STATE UNIVERSITY

The Same Attack is Made on CyberShield

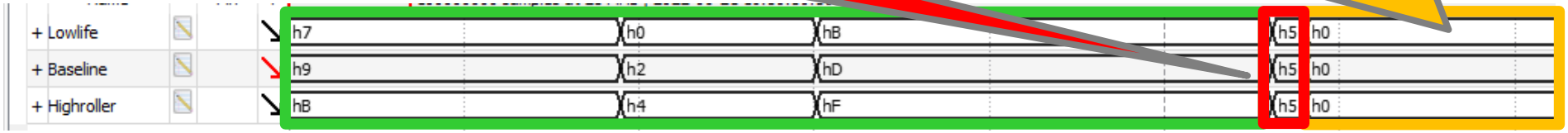
We can see how CyberShield Responds by Measuring the Instruction Registers in the CPU with a Logic Analyzer.

All Opcodes are Different by Design



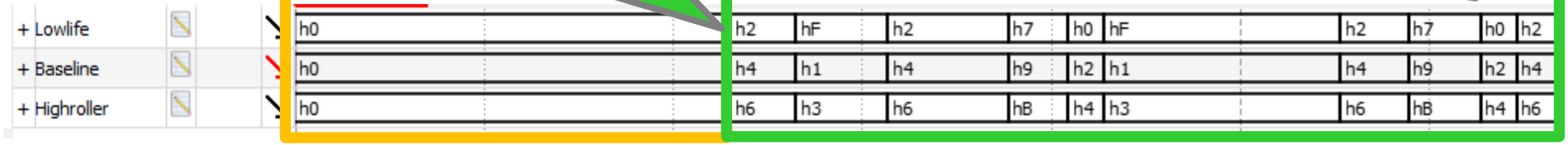
The attack is detected when all three CPUs see the same Opcode.

CyberShield Halts Operation and Initiates a Recovery Procedure.



After flushing out the malware, CyberShield resumes normal operation.

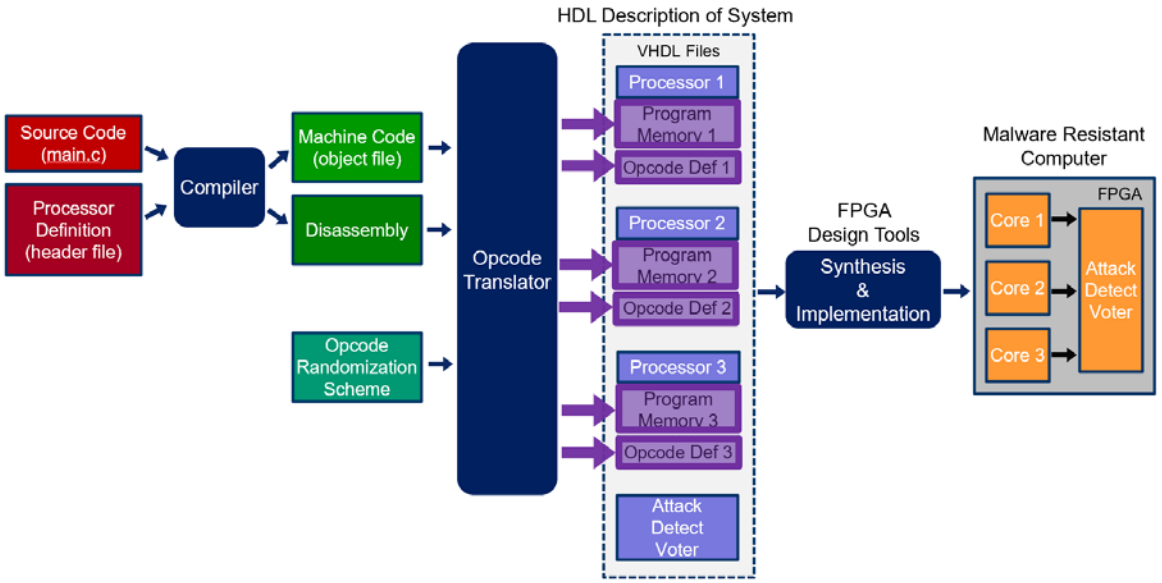
The rapid nature of hardware recovery allows low latency and the ability to operate-through-attack.



Conclusion

- CyberShield is an approach to defeating malware by introducing hardware diversity at the hardware level.
- This is enabled by real-time HDL generation at compile-time.
- A buffer insertion attack was used to test CyberShield.
- CyberShield was able to detect the malware, remove it, and continue operation while an MCU was not.

Questions


HDL Description of System
VHDL Files
Source Code (main.c)
Processor Definition (header file)
Compiler
Machine Code (object file)
Disassembly
Opcode Randomization Scheme
Opcode Translator
Processor 1
Program Memory 1
Opcode Def 1
Processor 2
Program Memory 2
Opcode Def 2
Processor 3
Program Memory 3
Opcode Def 3
Attack Detect Voter
FPGA Design Tools
Synthesis & Implementation
Malware Resistant Computer
FPGA
Core 1
Core 2
Core 3
Attack Detect Voter