A security assessment study and trial of TriCore-powered automotive ECUs

CODEBLUE 2014.12.18-19

Takahiro Matsuki (FFRI)
Dennis Kengo Oka (ETAS)

Today's Talk

- Introduction
- About ECU Software
- Overview of TriCore
- Investigation and Confirmation of Attack Methods
- Demo
- Summary and Future Plans

Introduction– Motivation

- Previous research has shown that vehicle ECUs can be targeted by attackers through injection of messages on the CAN bus – what about ECU software?
- ECU microcontroller architecture is different from traditional PC architecture; therefore, traditional software attacks do not work?
- ECU microcontrollers have specific security countermeasures preventing software attacks?
- If software is vulnerable, by adjusting traditional software attacks to ECU microcontroller architecture, it is possible to execute attacks?

Knowledge Required for Security Research of ECUs

- ECU Hardware and Software Configuration
  - ECU functions and microcontrollers, bus, I/O Interface
  - What microcontrollers are used

- ECU Microcontroller Architecture
  - Program Execution Method
  - Instruction Execution Flow, Register, Memory Layout

- ECU Software Execution Environment and Development Environment
  - Library, Compiler
  - Content of the code generated by development tools
  - Reverse engineering methods of the program files

About ECU Software

Vehicle ECUs

- Basically every modern production car has a multitude of electronic control units to provide safety- as well as comfort functions.
- Average vehicle has up to 70 ECUs
- >20,000,000 lines of source code
- Electronic components are estimated to cost about 50% of the automotive production costs by 2015
- 50% infotainment, 30% powertrain and transmission, 10% chassis control, 10% body and comfort

Examples of ECUs

- Engine control unit
  - Fuel amount and mixture, air and fuel delivery timing, valve timing, ignition timing, emission control, etc…
- Transmission control unit
  - Gear change, shift lock, shift solenoids, pressure control solenoids, etc…
- Body control unit
  - Central locking, immobilizer system, power windows, climate control, etc…
- ABS/ESP control unit
  - Regulating brake pressure, traction control, cornering brake control, etc…

ECU Basic Functionality

- Typical feedback control system
- 1. Monitor input. e.g. timer, sensors, CAN
- 2. Calculate or lookup appropriate response
- 3. Generate corresponding outputs

About ECU Software

- Fully custom, proprietary software
- Unix-based proprietary software
- Standardized software. e.g. AUTOSAR

Overview of TriCore

About TriCore

- Microcontroller for Vehicle ECUs
- Manufactured and sold by Infineon
  - Spin off of Semiconductor Unit from the German manufacturer Siemens AG
- ECUs With TriCore
  - Bosch EDC17 & MED17, Siemens
- Car Manufacturers Using ECUs with TriCore
  - Audi, BMW, Citroen, Ford, Honda, Hyundai, Mercedes-Benz, Nissan, Opel, Peugeot, Porsche, Renault, Seat, Toyota, Volkswagen, Volvo

Overview of the Architecture and Features

- Command Set
  - 32 bit RISC Architecture

- Unique Register Configuration
  - Completely separated address and data registers
  - A0~A16, D0~D16

- Model Number and Specifications of the Microcontroller Used in this Research
  - TC1797 (AUDO Future)
    - TriCore Architecture 1.3.1
    - Clock 180 MHz

Research Method (1)

- Research of Open Information and Specification Documents on the Web
  - Official User's Manual
    - Most reliable information source
    - Focused on memory related charts/diagrams
    - Keyword search for security related terms
      - security, protection, password
  - TriCore Architecture Overview
    - Summarized material of the User's Manual

Research Method(2)

- Searched Research Papers
  - TriCore Emulator
    - Porting TriCore to QEMU
  - Porting Linux Kernel to TriCore
- Searched for Information/Tools for Software Developers of ECU Software
  - Development Environment TASKING VX-toolset for TriCore (Evaluation Edition)
    - Compiler, IDE with simulator
  - Evaluation Board Infineon Starter Kit TC1797
  - FlexECU development platform
- Reverse Engineering of the Binary
  - Possible to Disassemble using IDA Pro
    - File Format is ELF for Siemens TriCore

Investigation and Confirmation of Attack Methods

Possible Vulnerabilities in ECU Software

- Non-Memory Corruption Vulnerabilities
  - Access Control Issues
  - Encryption Strength Issues
  - Inappropriate Authentication
  - Conflictions
  - Certificate / Password Management Issues
  - etc.

- Memory Corruption Vulnerabilities
  - Buffer Overflow
  - Integer Overflow
  - Use After Free
  - Null Pointer Dereferences
  - Format String Bugs
  - etc.

Since it was difficult to obtain and analyze actual ECU Software, we hypothesized about the possibility of memory corruption vulnerabilities.

Consideration of Memory Corruption Vulnerabilities and Possible Attacks

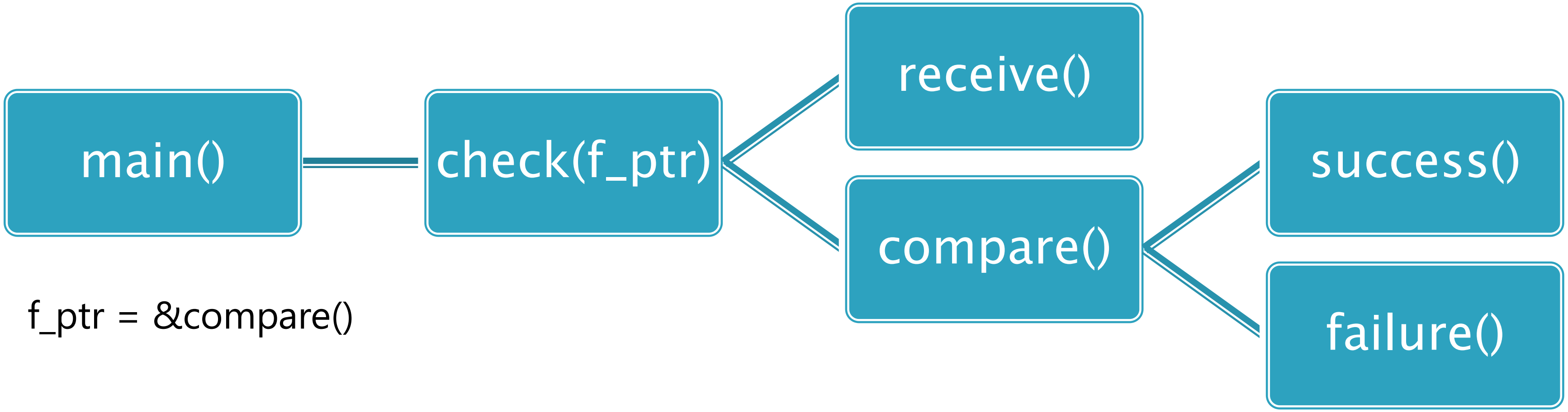
- Buffer Overflow
  - Stack Overflow
  - Heap Overflow
- Integer Overflow
  - Hypothesized that integer overflows can cause of heap overflows
- Format String Bug
  - Possible to overwrite an arbitrary value in an arbitrary address, hypothesized that attacks are possible
- Use After Free
  - Implied attacks are possible because C++ code is executable with TriCore
- Null Pointer Dereference
  - Trap occurs by access to memory address zero, hypothesized attacks are possible

Possibility of Buffer Overflow Attacks

- Stack Overflows
  - For TriCore, unlike x86 and others, the return address is saved in the address register (A11) instead of the stack, therefore overwriting the return address using a stack overflow is not possible.
- Heap Overflows
  - Will examine TriCore's heap management in the near future

Overwriting Function Pointers by Stack Overflows (1)

- Possibilities of a Stack Overflow Attack
  - If the buffer and function pointer exist on the stack in the following code flow, it is possible to change the execution flow of the program by a stack overflow



f_ptr = &compare()

Overwriting Function Pointers by Stack Overflows (2)

- Example Code

```
failure() { ...
}

success() { ....
}

compare() { ...
}

receive() {
    // receive input value
    input = ...
    char buffer[10];
    strcpy( buffer, input );
}
```

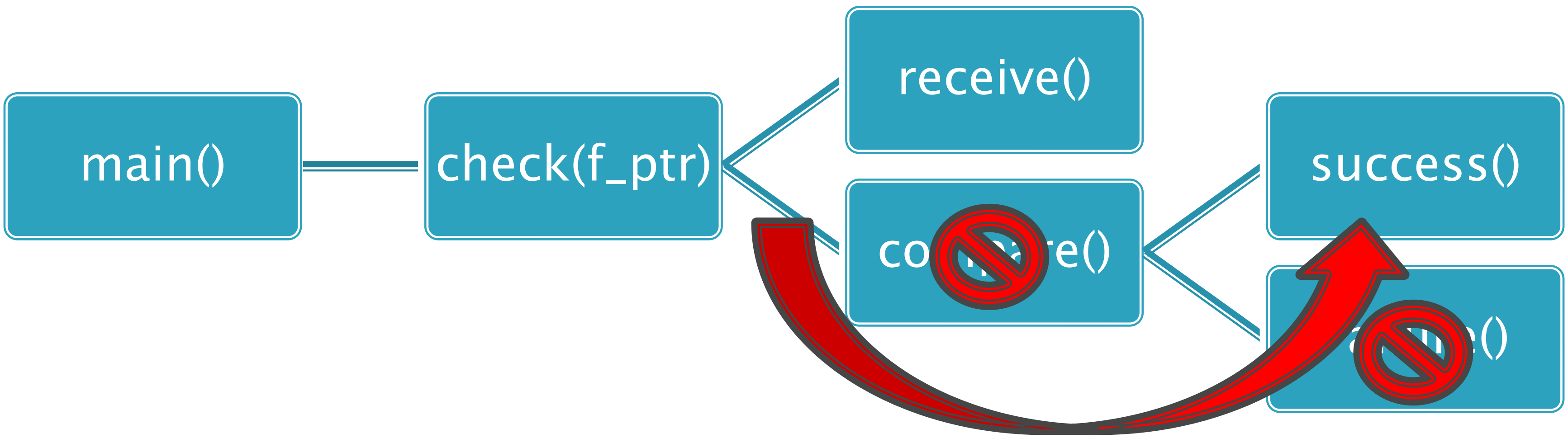
```
check( f_ptr ) {
// call receive() to receive incoming values
      receive();
// call compare() using a function pointer
      f_ptr();
}

main () {
      function_ptr = &compare;
      ...
      check( function_ptr );
}
```

Overwriting Function Pointers by Stack Overflows (3)

- Memory Layout

Address	Variable
0x8000 010C	check()
0x8000 013C	receive()
0x8000 0170	compare()
0x8000 017C	success()
0x8000 0188	failure()
…	…
0xD000 8FC8	buffer[]
0xD000 8FE0	function_ptr

Example – Changing Flow



check() calls f_ptr() which now points to success() (0x8000 017C) instead of compare() (0x8000 0170)

Overwriting Function Pointers by Stack Overflows (4)

- Issues
  - If compiler optimization is “on”, the function pointer will be stored in the address register
  - Unclear whether there are similar code patterns in actual ECU software

Considering Attacks Possibilities Using TriCore's Control Mechanism

Considering Attack Possibilities Using TriCore's Control Mechanism

- Preconditions
  - It is possible to overwrite data by using memory corruption vulnerabilities
  - Under the above condition, considered ways to execute arbitrary code

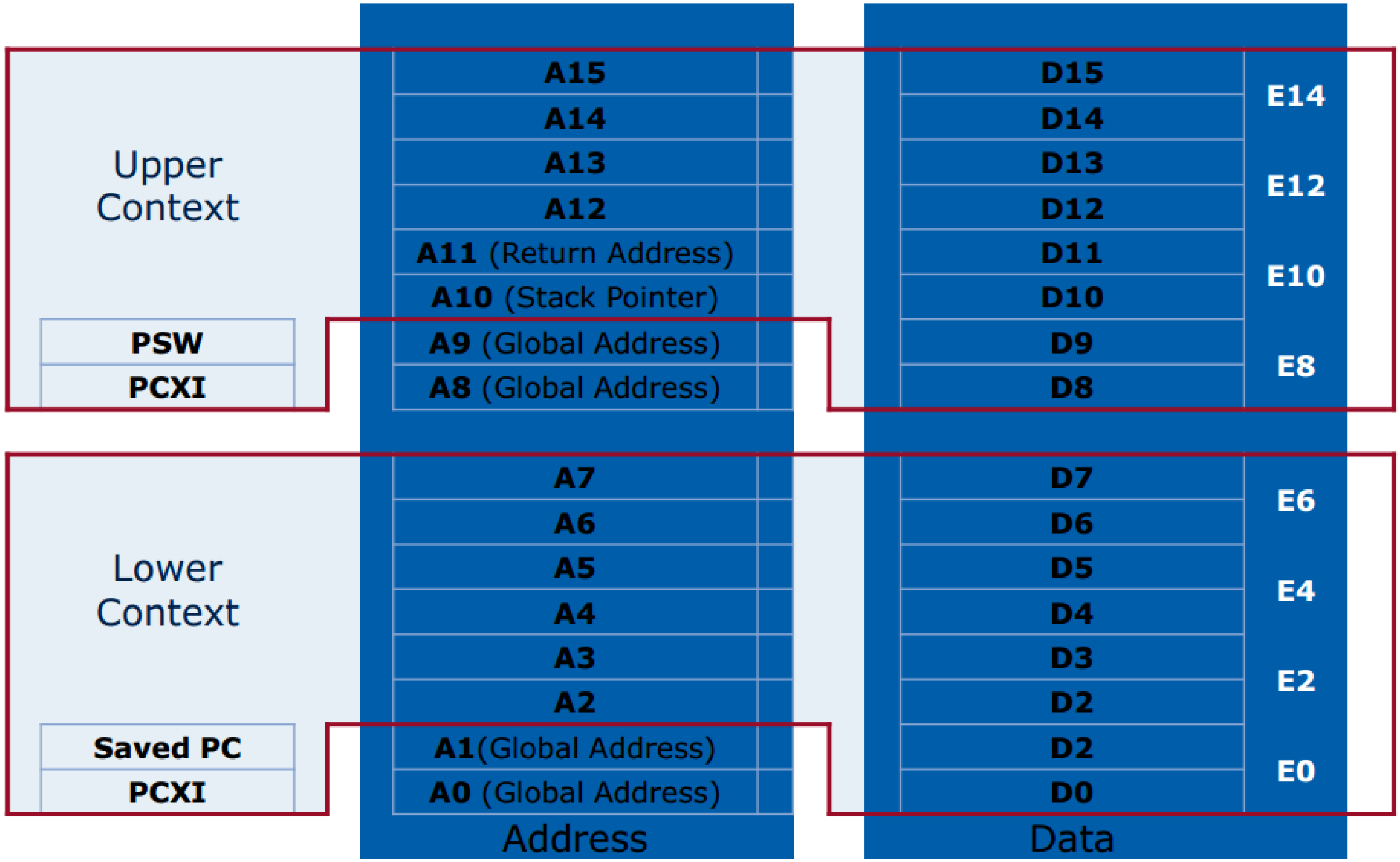
- TriCore's Control Mechanism
  - Context Management Mechanism
  - Interrupt/Trap Mechanism

Attack Methods Using the Context Management Mechanism

Overview of Context of TriCore

- About Context
  - The register value is CSA (Context Save Area)
    Saves and restores in TriCore's unique memory space
- Types of Context
  - 2 Types: Upper context and Lower context
  - Upper context
    - call command, interrupt, automatically saves when trapped
  - Lower context
    - Explicitly saved by using a dedicated command, used for passing parameters

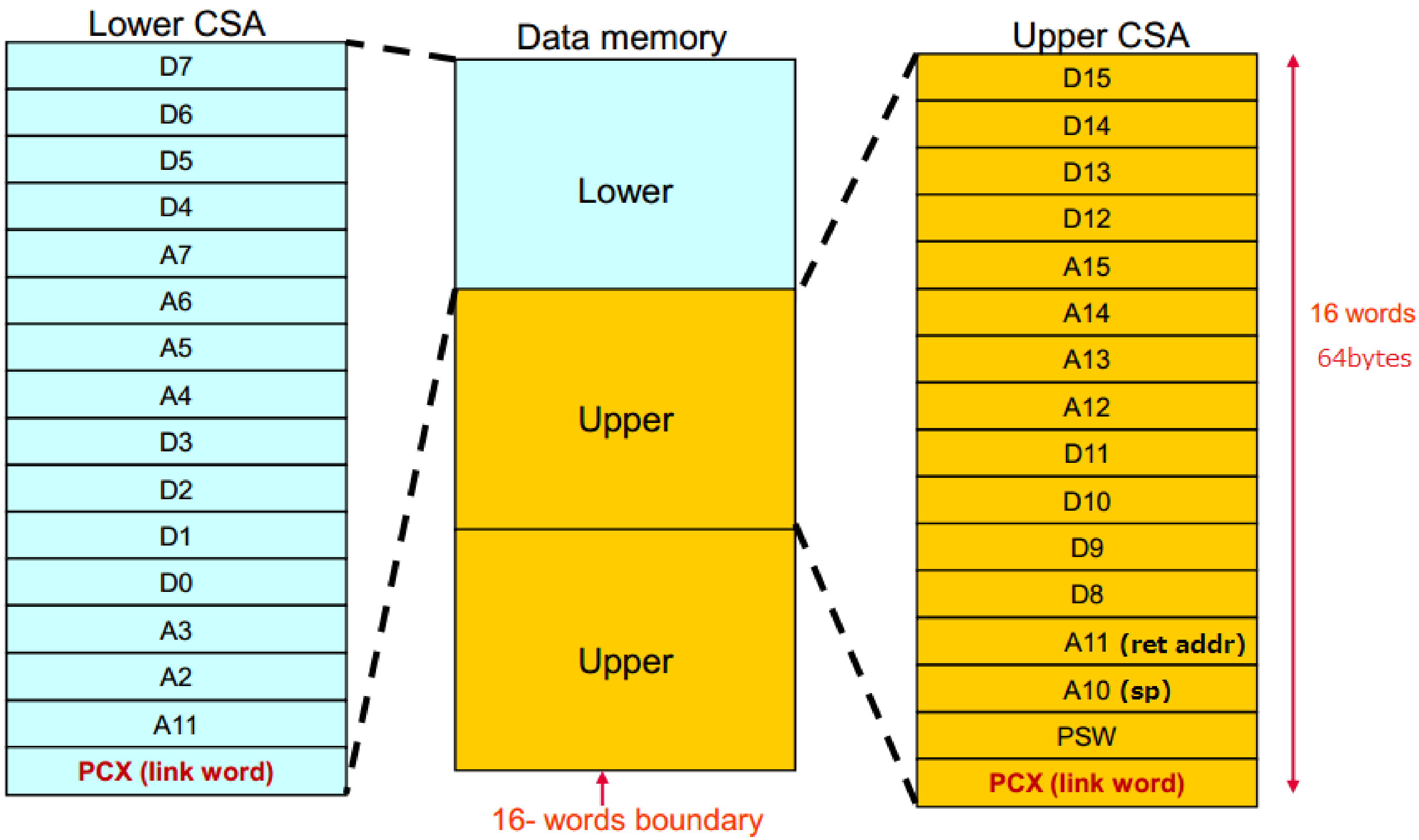
Registers Saved in the CSA



Reference: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

CSA Configuration

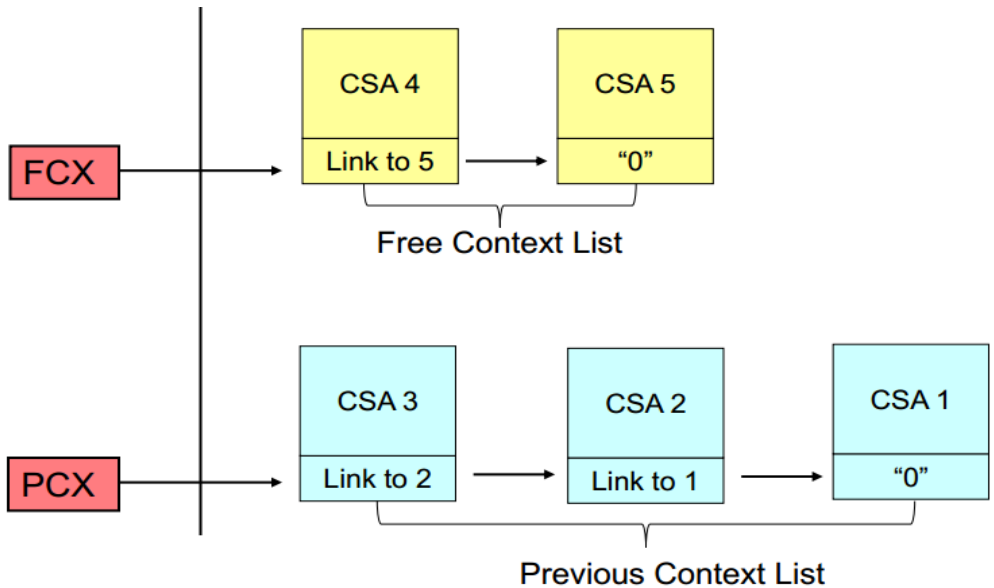
- Context Save areas can hold 1 upper or 1 lower context.
- CSA are aligned on a 16-word boundary.



Reference: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

CSA Management

- CSA is Managed by Link Lists
  - Used CSA List (PCX) , Unused CSA List (FCX)
  - Pointer to the first element of each list is PCX, stored in FCX register
    - However, needs to be converted because it is not a raw address



Reference: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

Code Execution Methods Using Context

- Method 1 : CSA Overwriting
  - By overwriting any return address saved in the CSA using a memory corruption vulnerability, it is possible to run code of an arbitrary address

- Method 2 : CSA Injection
  - By overwriting a Link word of the CSA using a memory corruption vulnerability, it is possible to restore crafted Upper context (including return address) and run arbitrary code.

CSA Overwrites on a Simulator

- Code on the right is the result of execution without augments
  func1
  func2
  func3
- Rewrite the return address (*ret) within func2 saved in the CSA to func3 address (0x80000360)
- When returned to func1, the A11 register value restores to func3 (0x80000360)
- Jump to func3 on func1 return

```c
#include <stdio.h>

int func3()
{
    printf("func3\n");
    return 0;
}
int func2()
{
    unsigned int *ret;
    printf("func2\n");
    ret=0xD0004F4C;
    *ret=0x80000360;
    return 0;
}
int func1()
{
    printf("func1\n");
    func2();
    return 0;
}
int main( int argc, char** argv)
{
    func1();
    if (argc == 1)
    {
        func3();
    }
}
```

CSA Overwrites on a Simulator

- CSA overwrite using an evaluation board was possible in the same way as the simulator
  - There are memory protections to prevent CSA overwrites by default.
  - May be possible to exploit on actual ECU Software

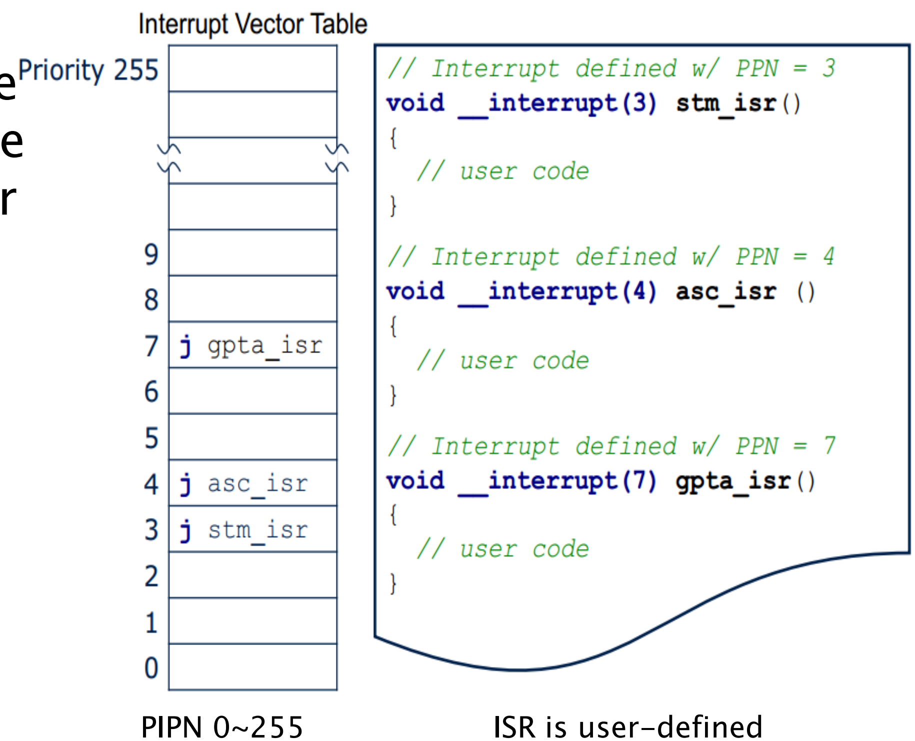
Attack Methods Using the Interrupt and Trap Mechanisms

Overview of the Interrupt Mechanism

- When an interruption occurs, the Interrupt Vector Table (IVT) is referred, and the Interrupt Service Routine (ISR) corresponding to the Pending Interrupt Priority Number (PIPN) is executed

- IVT Start Address
  - BIV Register (Begin Interrupt Vector)

- Addresses of entry point of each ISR
  - BIV | (ICR.PIPN << 5)
    - ICR (Interrupt Control Register)

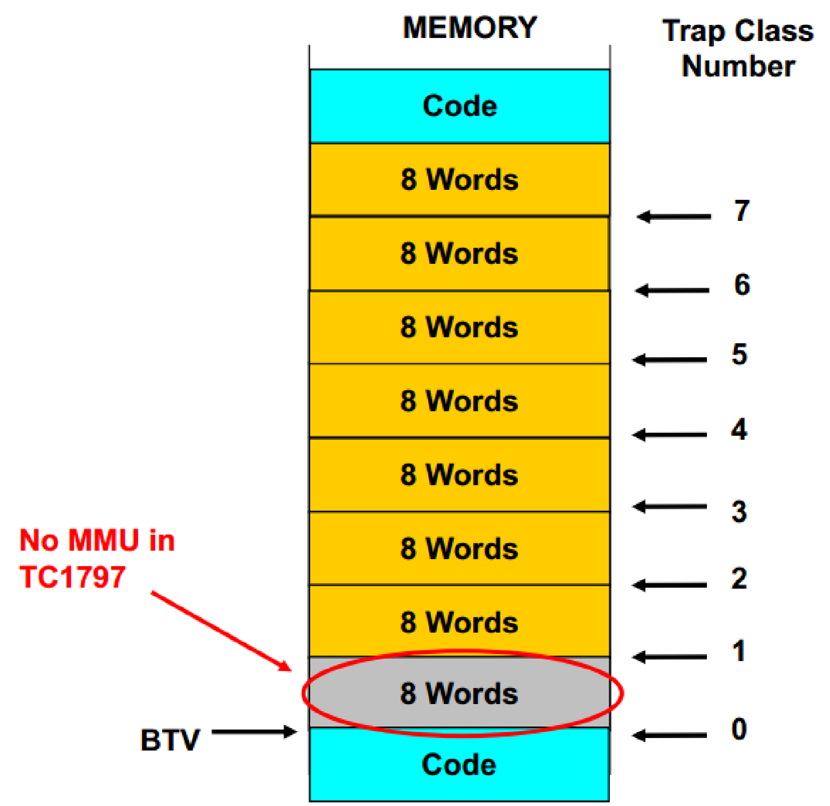


Reference: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

Overview of the Trap Mechanism

- A mechanism used when an exception occurs. It is trapped and runs a specific process
  - Causes of Traps
    - Command exceptions, unauthorized memory access, etc…

- When a trap occurs, the Trap Vector Table is referred, and the Trap Service Routine corresponding to the Trap Class Number (TCN) is executed.



Reference: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

Code Execution Method By Overwriting the Interrupt/Trap Vector Table

- Method 1: Overwrite the IVT
  - By overwriting the jump code to the ISR in the IVT, when a certain interrupt occurs, run arbitrary code
- Method 2: Overwrite the TVT
  - By overwriting the TSR code in the TVT , when a certain trap occurs, run arbitrary code

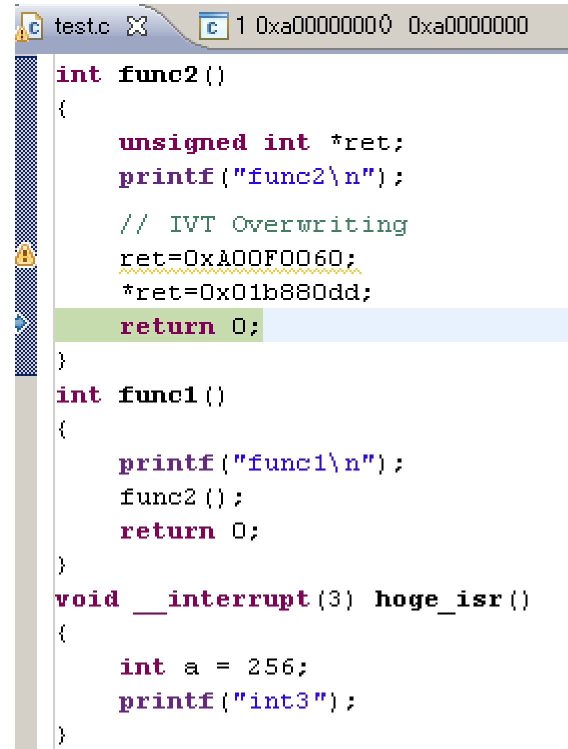
Overwrite of IVT/TVT on a Simulator

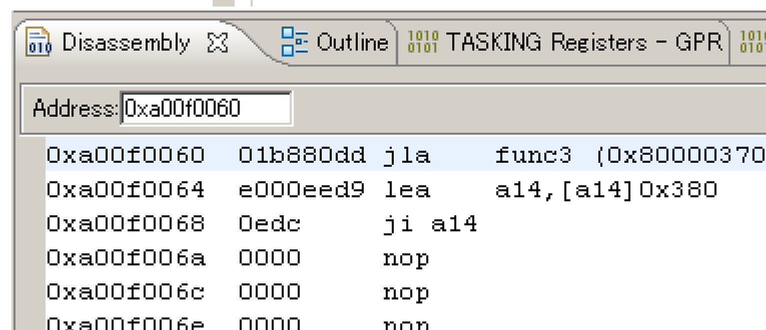
- BIV value 0xa00f0000

- Define a ISR as __interrupt(3) hoge_isr() , a jump code to hoge_isr() is allocated to 0xa00f0060 (0xa00f00+ 32*3) , making it possible to overwrite

- Overwrite possible on simulator
  - However, because whether an interrupt could be triggered intentionally is unknown, left untested
  - In the real project, the 0xA segment is mapped on the Flash memory, and may not be overwritable.

- The TVT is similarly overwritten





IVT/TVT Overwrite on an Evaluation Board

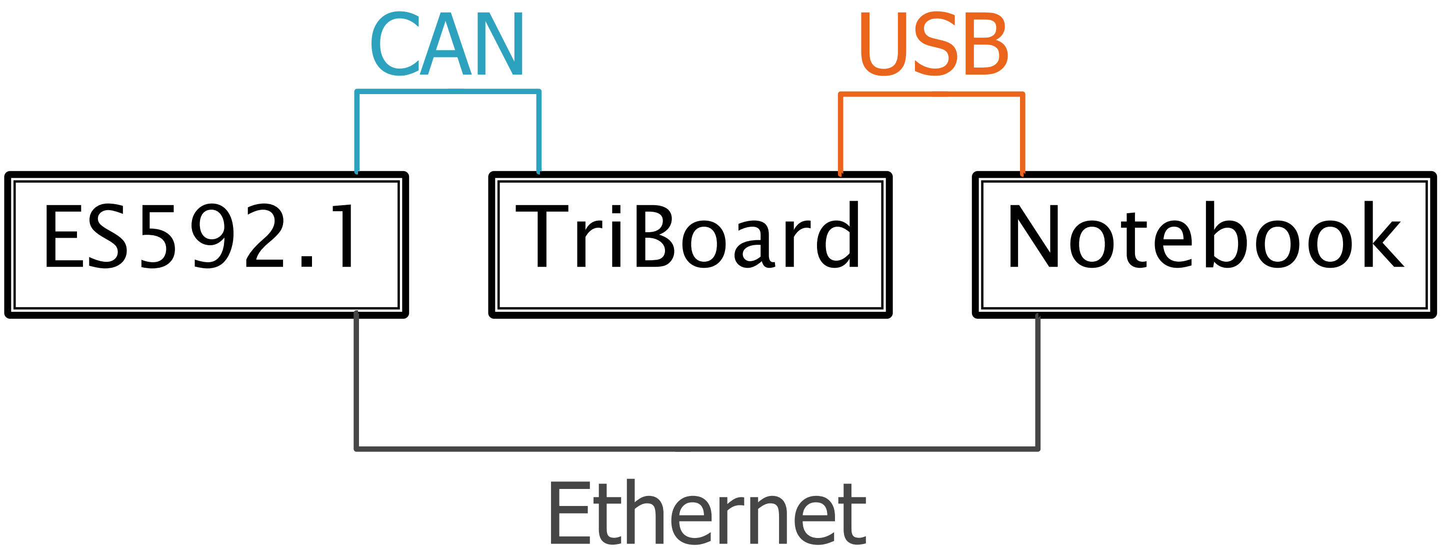
- The BIV and BTV values of the evaluation board are different from the simulator
  - BIT @ 0xF7E1FE20, BTV @ 0xF7E1FE24
- Protected
- Overwriting from the debugger possible
- Tried to overwrite the code by disabling the protection and a trap2 occurred
- Probably cannot exploit on actual ECU software

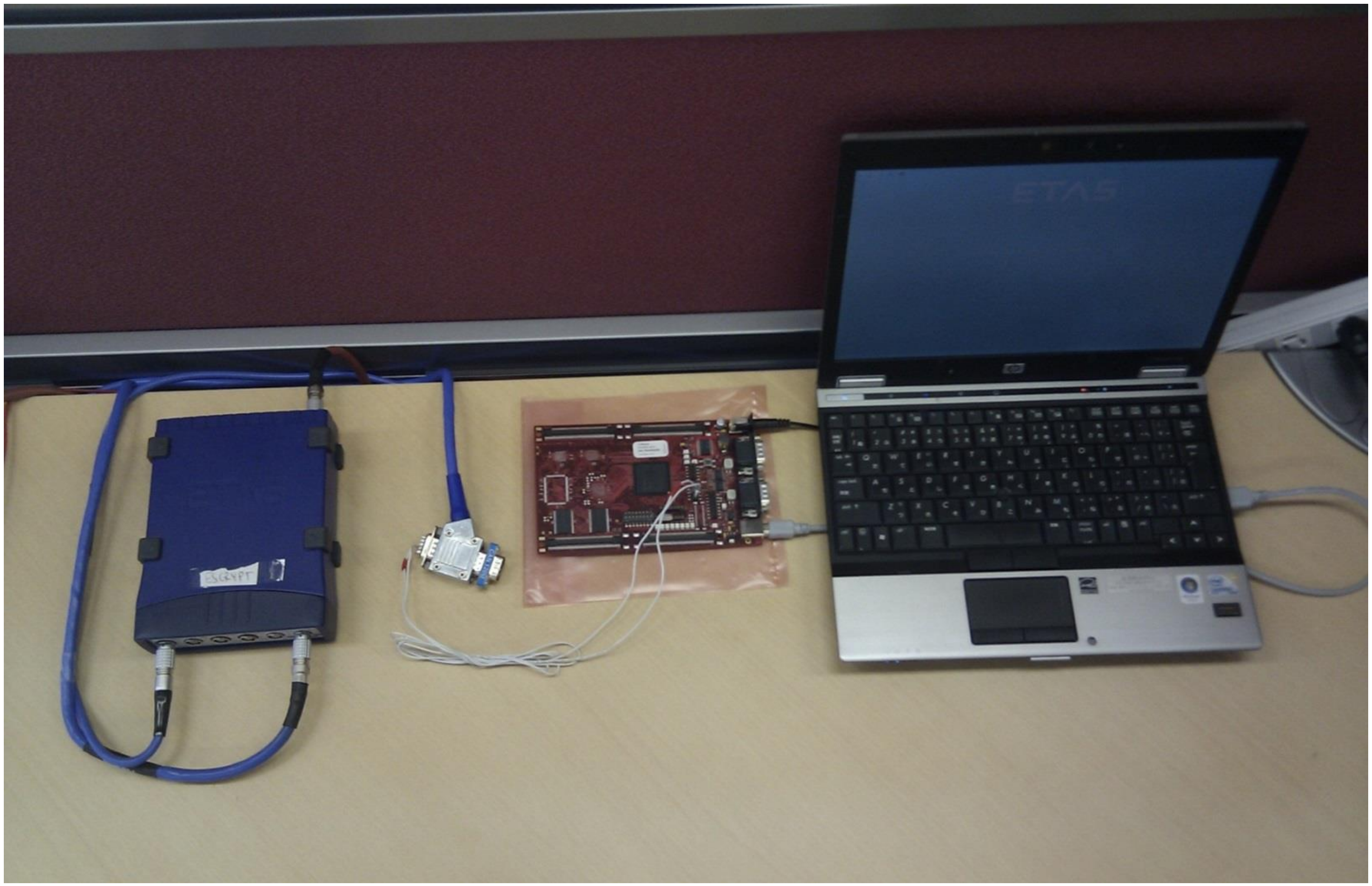
Verification on an Evaluation Board

Evaluation Environment

- HP EliteBook 2530p, Win7, Centrino2
  - HIGHTEC Free TriCore Entry Tool Chain
  - BUSMASTER
- Infineon TriBoard TC1797 V5.0
- ETAS ES592.1

Memory Monitoring Method




ETAS

Demo

Summary and Future Plans

Summary

- Considered attack methods on ECU software in which memory corruption vulnerabilities exist
- If memory corruption vulnerabilities exist, it may be possible to execute arbitrary code
  - If a buffer overflow exists, it is possible to execute arbitrary code under certain conditions
  - By altering the CSA, it is possible to execute arbitrary code
  - By altering the interrupt/trap vector tables, it is impossible to execute arbitrary code
- Created vulnerable ECU software and conducted an attack demo
- This research is a result of a study of logical attack methods and a demo conducted on a vulnerable software sample. This study does NOT indicate anything about existing threats on actual ECU software.

Future Plans

- Additional Research
  - Study other vulnerabilities and architecture specific issues
- Demonstrate the Threats
  - Reverse engineering of ECU software and investigate if memory corruption vulnerabilities exist
  - Attack actual vulnerabilities and verify if the ECUs stop or if anything abnormal occurs
- Consider Countermeasures
  - Consider countermeasures of ECU software vulnerabilities
  - Consider measures to efficiently discover vulnerabilities resulting from programming errors

Thank you