TriCore 車載 ECU に対する セキュリティアセスメントの検討と試行

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

CODEBLUE 2014.12.18-19

Takahiro Matsuki (FFRI)
Dennis Kengo Oka (ETAS)

目次

- はじめに
- ECUソフトウェアについて
- TriCore の概要
- 攻撃手法の検討と試行
- デモ
- まとめと今後の方針

はじめに – 目的

- 自動車セキュリティの先行研究で、車載 ECU は CAN バスへのメッセージのインジェクションによって攻撃者のターゲットになることが示されている。ECU ソフトウェアは攻撃対象になるだろうか?
- ECU マイコンのアーキテクチャは従来の PC アーキテクチャとは異なるため、従来のソフトウェア脆弱性攻撃は有効ではない?
- ECU マイコンにはソフトウェアへの脆弱性攻撃を防ぐ特定のセキュリティ対策があるのだろうか?
- ECU ソフトウェアが脆弱な場合、従来の脆弱性攻撃を ECU マイコンアーキテクチャに適合させることによって、攻撃を実行することは可能?

ECUのセキュリティ研究に必要と考えられる知識

- ECUのハードウェアおよびソフトウェア構成
  - ECUの機能とマイコン、バス、I/Oインタフェース
  - どんなマイコンが使われているのか
- ECUのマイコンアーキテクチャ
  - プログラム実行方式
  - 命令実行フロー、レジスタ、メモリレイアウト
- ECUソフトウェア実行環境および開発環境
  - ライブラリ、コンパイラ
  - 開発ツールによって生成されるコードの内容
  - プログラムファイルのリバースエンジニアリング手法

ECUソフトウェアについて

車載 ECU

- 基本的に現在の車は、快適機能だけでなく、安全性を提供するためにも電子制御ユニットを多数有している
- 平均 70 個の ECU を搭載している
- ECU ソフトウェアのソースコードは2000万行以上
- 2015年までに自動車の生産コストの50%程度が電子部品になる
- 50%がインフォテインメント、30%がパワートレイン及びトランスミッション、10%がシャーシ制御、10%のボディと快適機能

ECU の例

- エンジン制御ユニット
  - 燃料の量と混合、空気および燃料の送出タイミング、バルブタイミング、点火タイミング、エミッション制御…
- トランスミッション制御ユニット
  - 変速、シフトロック、シフトソレノイド、圧力制御ソレノイド…
- ボディ制御ユニット
  - 集中ロック、イモビライザーシステム、パワーウィンドウ、空調制御…
- ABS/ESP 制御ユニット
  - ブレーキ圧調整、トラクションコントロール、コーナリングブレーキ制御…

ECU の基本機能

- 典型的なフィードバック制御システム
- 1. 入力の監視、例えば、タイマー、センサー、CAN
- 2. 計算あるいは適切な対応の探索
- 3. 対応する出力の生成

ECUソフトウェアについて

- 完全にカスタムなプロプライエタリなソフトウェア
- Unixベースの独自のソフトウェア
- 標準化されたソフトウェア、例えば、AUTOSAR

TriCoreの概要

TriCore について

- 自動車ECU用マイコン
- 製造販売 Infineon
  - 独シーメンスの半導体部門のスピンアウト
- TriCore を搭載しているECUの例
  - Bosch EDC17 & MED17, Siemens
- TriCore 搭載 ECU の採用車種がある自動車メーカ
  - Audi, BMW, Citroen, Ford, Honda, Hyundai, Mercedes-Benz, Nissan, Opel, Peugeot, Porsche, Renault, Seat, Toyota, Volkswagen, Volvo

アーキテクチャの概要と特徴

- 命令セット
  - 32ビット RISC アーキテクチャ
- 独特なレジスタ構成
  - アドレス用レジスタとデータ用レジスタが完全に分離
  - A0~A16, D0~D16
- 今回調査したマイコンの型番とスペック
  - TC1797 (AUDO Future)
    - TriCore アーキテクチャ 1.3.1
    - クロック 180 MHz

調査方法(1)

- Web 上の公開情報・仕様書を探す
  - 公式 User's Manual
    - 最も信頼性の高い情報源
    - メモリ関連の図表に注目
    - セキュリティに関係するキーワードを検索
      - security, protection, password
  - TriCore Architecture Overview
    - User's Manual の要約資料

調査方法(2)

- 論文を探す
  - TriCoreエミュレータ
    - QEMUにTriCoreをポーティング
  - Linux Kernel を TriCoreにポーティング
- ECU ソフトウェア開発者向け情報・ツールを探す
  - 開発環境 TASKING VX-toolset for TriCore 評価版
    - コンパイラ、シミュレータ付き IDE
  - 評価ボード Infineon Starter Kit TC1797
  - FlexECU development platform
- バイナリをリバースエンジニアリングする
  - IDA Pro で逆アセンブル可能
    - ファイルフォーマットは
      ELF for Siemens TriCore

攻撃手法と検討と試行

ECUソフトウェアに考えられる脆弱性

- 非メモリ破壊脆弱性
  - アクセス制御の問題
  - 暗号強度の問題
  - 不適切な認証
  - 競合状態
  - 証明書・パスワード管理の問題
  - etc.
- メモリ破壊脆弱性
  - バッファオーバーフロー
  - 整数オーバーフロー
  - Use After Free
  - Null Pointer Dereference
  - フォーマットストリングバグ
  - etc.

実際のECUソフトウェアの入手および解析が困難であったため、
メモリ破壊脆弱性が存在すると仮定し、理論的な攻撃手法を検討

メモリ破壊脆弱性と攻撃可能性の検討

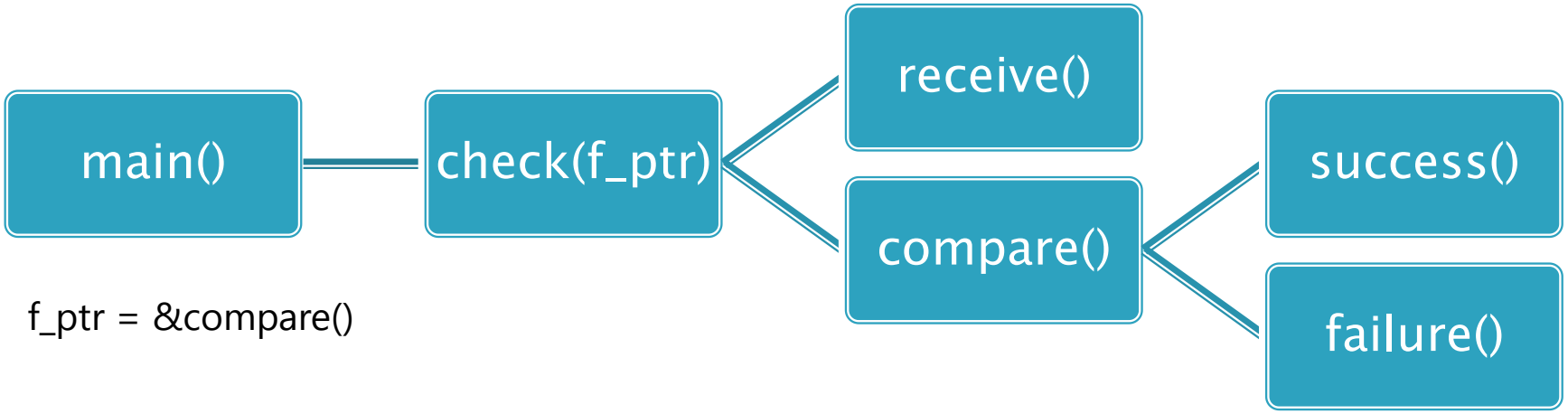
- バッファオーバーフロー
  - スタックオーバーフロー
  - ヒープオーバーフロー
- 整数オーバーフロー
  - ヒープオーバーフローの原因となると推測
- フォーマットストリングバグ
  - 任意のアドレスに任意の値を書き込み可、攻撃可能と推測
- Use After Free
  - TriCoreでC++コードも実行可能なため攻撃可能と推測
- Null Pointer Dereference
  - 0番地へのアクセスでトラップが発生するため攻撃不可と推測

バッファオーバーフローの攻撃可能性

- スタックオーバーフロー
  - TriCore は x86 等と異なり、リターンアドレスがスタックではなく、アドレスレジスタ(A11)に保存されるためスタックオーバーフローによるリターンアドレスの上書きは不可
- ヒープオーバーフロー
  - TriCoreのヒープ管理については今後調査予定

スタックオーバーフローによる関数ポインタ上書き(1)

- スタックオーバーフローによる攻撃可能性を検討
  - 以下のようなコードフローでスタック上にバッファと関数ポインタが存在する場合、スタックオーバーフローによってプログラムの実行フローを変更できる



バッファオーバーフローによる関数ポインタ上書き（2）

- コードの例

```
failure() { ...
}

success() { ....
}

compare() { ...
}

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

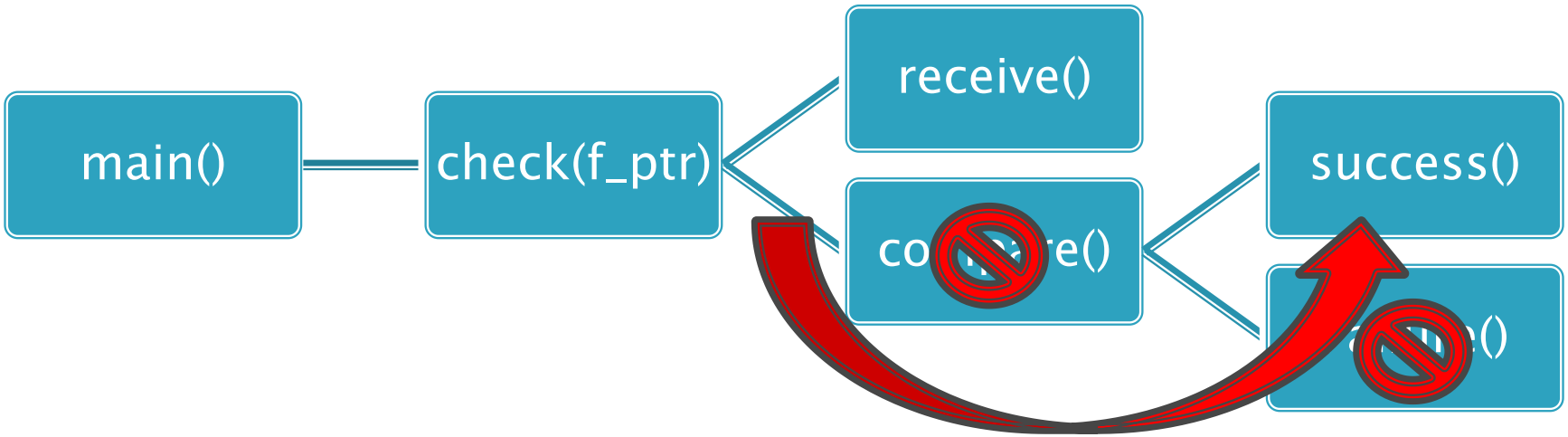
```
check( f_ptr ) {
// call receive() to receive incoming value
      receive();
// call compare() using function pointer
      f_ptr();
}

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

バッファオーバーフローによる関数ポインタ上書き(3)

- メモリレイアウト

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

コードフロー変更の例



check() が、compare() の代わりに success() を指す f_ptr() を呼び出す

バッファオーバーフローによる関数ポインタ上書き(4)

- 問題点
  - コンパイラの最適化が有効な場合、関数ポインタがアドレスレジスタに格納される
  - 実際のECUソフトウェアで同様パターンのコードが存在するか不明

TriCore の制御機構を利用した攻撃手法の検討

TriCore の制御機構を利用した攻撃手法の検討

- 検討の前提
  - **メモリ破壊脆弱性で任意のアドレスのデータを書換え可能**
  - **その場合に任意のコードを実行する手法を検討**

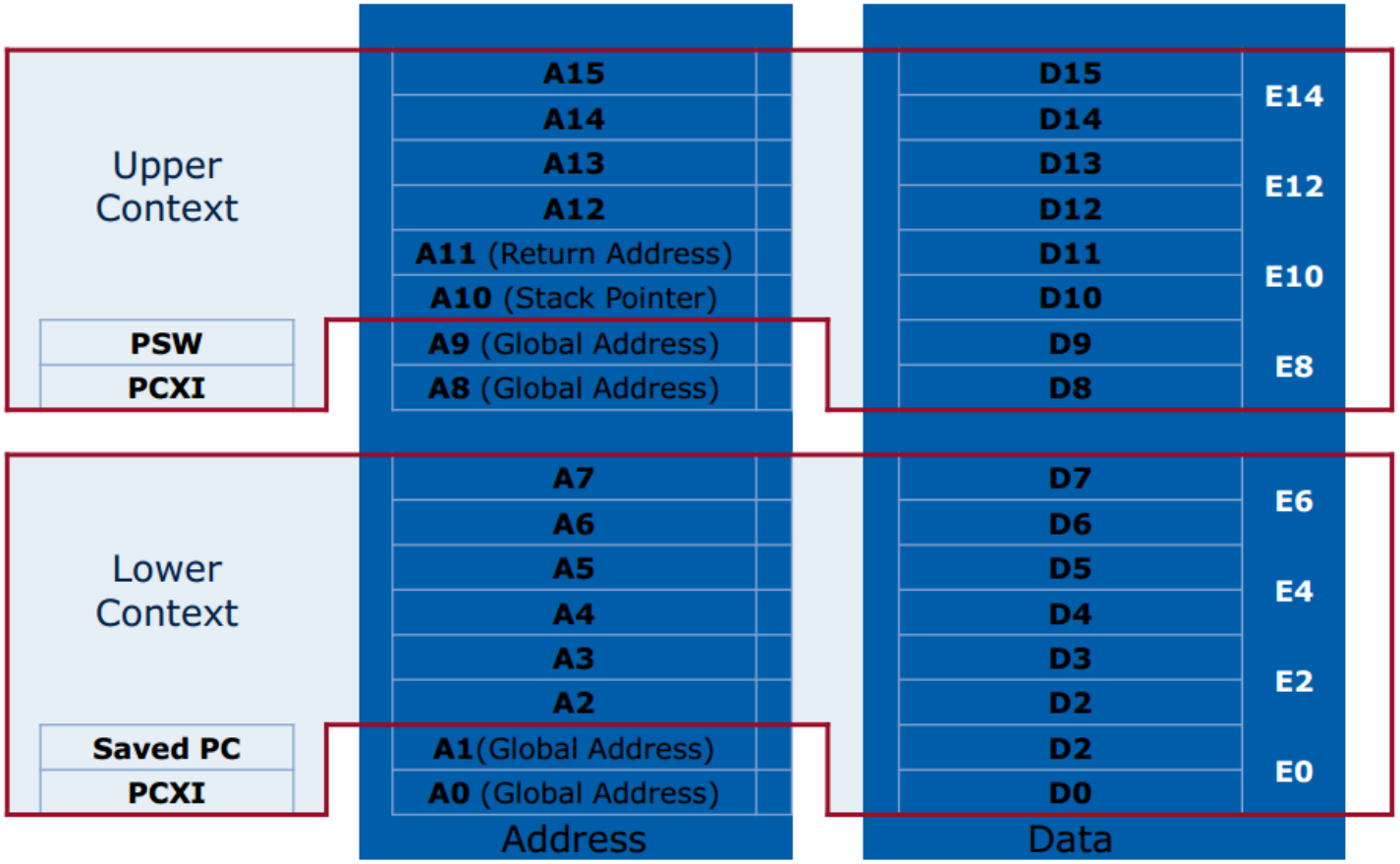
- TriCore の制御機構
  - コンテキスト管理機構
  - 割り込み・トラップ機構

コンテキスト管理機構を用いた攻撃手法

TriCore のコンテキストの概要

- コンテキストとは
  - レジスタの値を CSA (Context Save Area) という TriCore 独自のメモリ領域に保存・復元する仕組み
- コンテキストの種類
  - Upper context と Lower context の2種類
  - Upper context
    - call 命令、割り込み、トラップ時に自動的に保存される
  - Lower context
    - 専用命令で明示的に保存、パラメータ渡しに利用

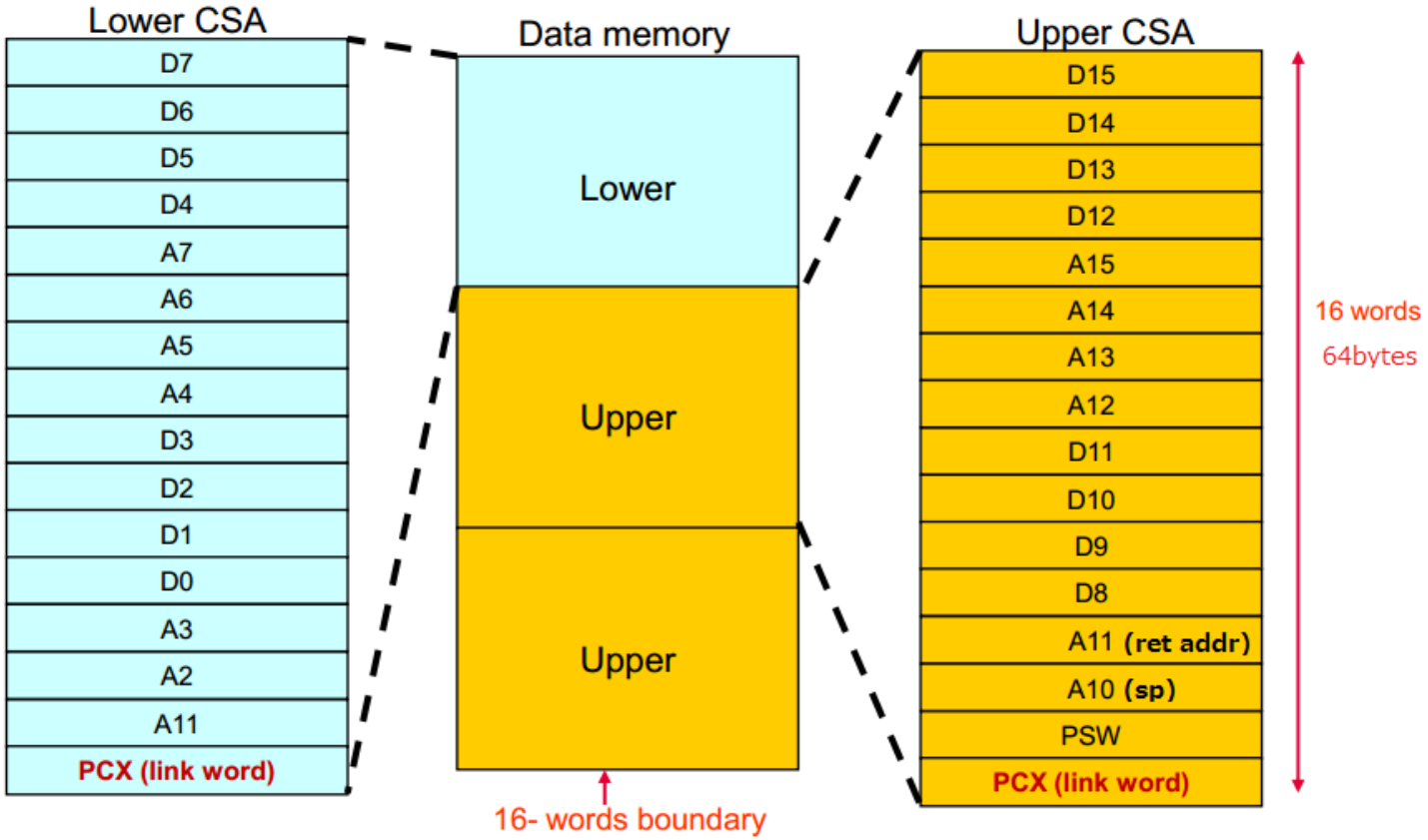
CSA に保存されるレジスタ



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

CSA の構成

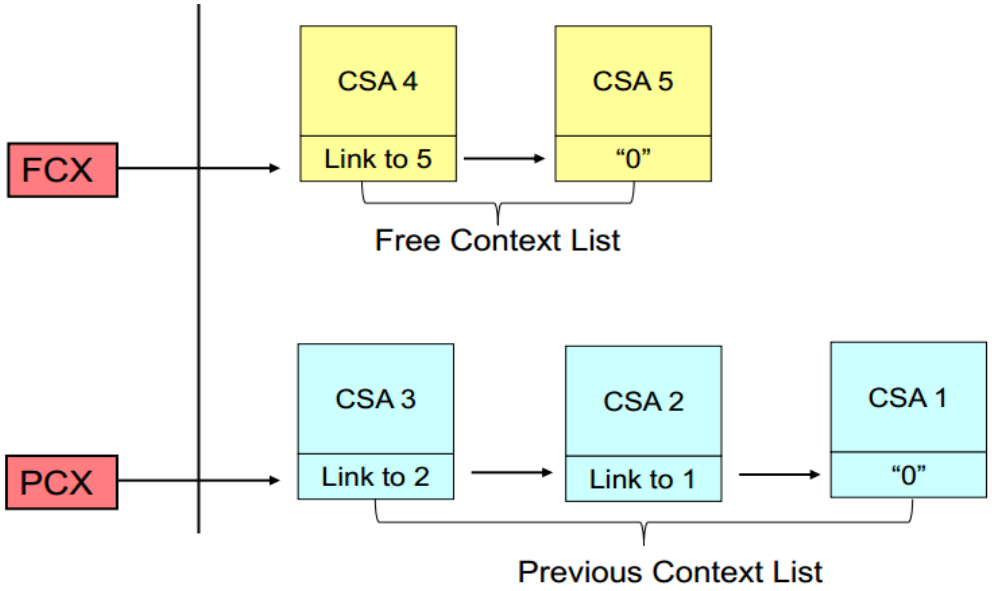
- Context Save areas can hold 1 upper or 1 lower context.
- CSA are aligned on a 16-word boundary.



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

CSA の管理

- CSA はリンクリストで管理される
  - 使用済み CSA リスト (PCX) 、未使用 CSA リスト (FCX)
  - 各リストの先頭要素へのポインタが PCX, FCX レジスタに格納
    - ただし、生のアドレスではないため変換が必要



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

コンテキストを用いたコード実行手法

- 手法1：CSA Overwriting
  - 何らかのメモリ破壊脆弱性で CSA に保存された Upper context 内のリターンアドレスを上書きすれば任意アドレスのコード実行が可能

- 手法2：CSA Injection
  - 何らかのメモリ破壊脆弱性で CSA の Link word を上書きすれば、リターンアドレスを含む細工された Upper context を復元させ、任意コードの実行が可能

シミュレータ上での CSA 上書きの試行

- 右記コードの実行結果
  func1
  func2
  func3

- func2 内で CSA に保存されたリターンアドレス(*ret)を func3 のアドレス(0x80000360)に書換え

- func1 に return すると A11 レジスタの値が func3 (0x80000360) に復元される

- func1 の return で func3にジャンプ

```
#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();
    }
}
```

評価ボードでの CSA 上書きの試行

- 評価ボードでもシミュレータと同様に CSA 上書きが可能であった
  - デフォルトでは CSA の書き換えを防止するメモリ保護はない
  - 実際のECUソフトウェアでの脆弱性攻撃にも適用できる可能性がある

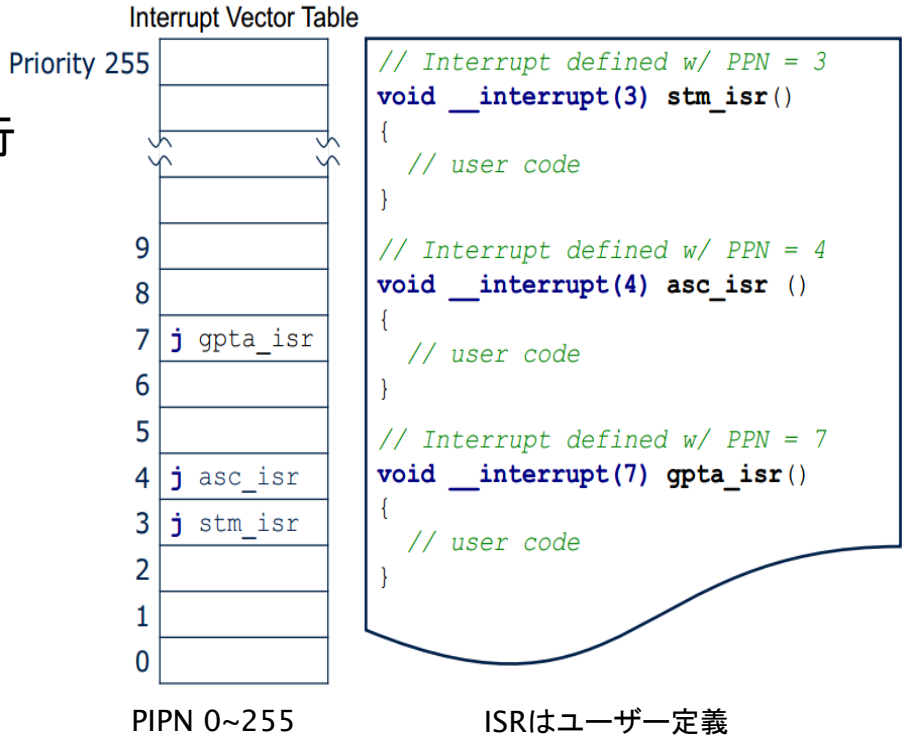
割り込みとトラップ機構を用いた攻撃手法

割り込み機構の概要

- 割り込みが発生すると割り込みベクタテーブル(IVT)が参照され、割り込み優先番号(PIPN)に対応する割り込みサービスルーチン(ISR)が実行される

- IVTの開始アドレス
  - BIVレジスタ(Begin Interrupt Vector)

- 各ISRのエントリポイントのアドレス
  - BIV | (ICR.PIPN << 5)
    - ICR (Interrupt Control Register)

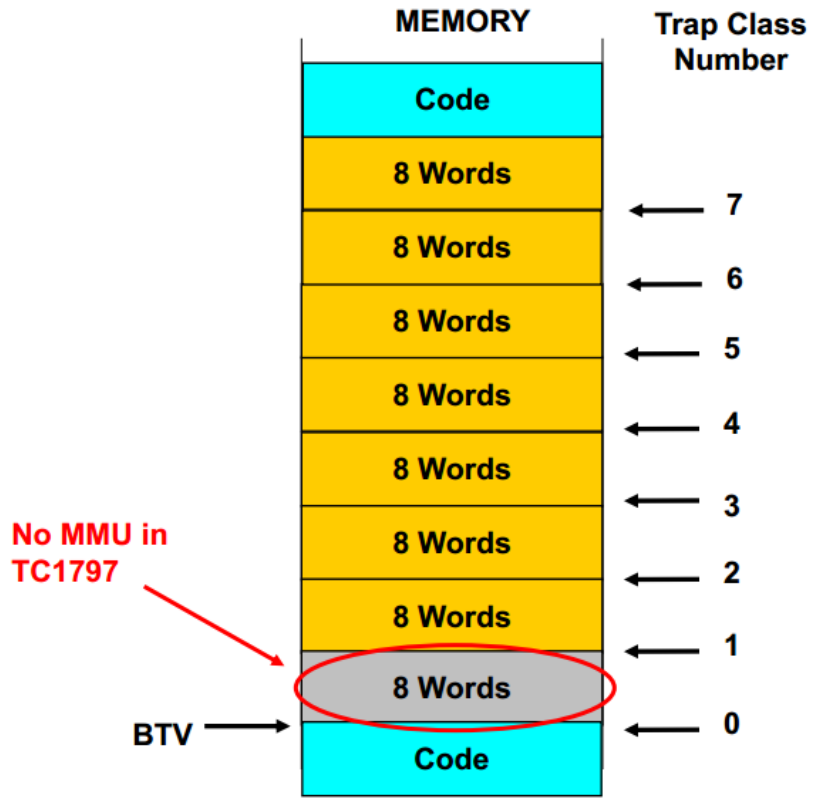


出典: Tricore Architeture Overview
http://www.infineon-ecosystem.org/download/schedule.php?act=detail&item=44

トラップ機構の概要

- 例外が発生した時それを捕捉し、特定の処理を行わせる機能
  - トラップ発生要因
    - 命令による例外、不正なメモリアクセス等

- トラップが発生するとトラップベクタテーブル（TVT）が参照され、トラップクラス番号(TCN)に対応するトラップサービスルーチン（TSR）が実行される



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

割り込み・トラップベクタテーブルの書き換えによるコード実行手法

- 手法1: IVT の書き換え
  - IVT にある ISR へのジャンプコードを書き換えることにより、特定の割り込みが発生した時に、任意のコードを実行
- 手法2: TVT 書き換え
  - TVT にある TSR コードを書き換えることにより、特定のトラップが発生した時に、任意のコードを実行

シミュレータ上での IVT・TVT 上書きの試行

- BIVの値 0xa00f0000
- __interrupt(3) hoge_isr() というISRを定義すると、0xa00f0060 (0xa00f00+ 32*3) にhoge_isr() へのジャンプコードが配置されるため、これを書き換え
- シミュレータでは書き換え可能
  - しかし、割り込みを意図的に発生させられるどうか不明のため、未検証
  - 0xA セグメントは、実機では Flash メモリにマップされ書き換え不可の可能性あり
- TVTも同様に書き換え可能

test.c | 1 0xa0000000 0xa0000000

```
int func2()
{
    unsigned int *ret;
    printf("func2\n");
    // IVT Overwriting
    ret=0xA00F0060;
    *ret=0x01b880dd;
    return 0;
}
int func1()
{
    printf("func1\n");
    func2();
    return 0;
}
void __interrupt(3) hoge_isr()
{
    int a = 256;
    printf("int3");
}
```

Disassembly | Outline | TASKING Registers - GPR

Address: 0xa00f0060

```
0xa00f0060  01b880dd jla     func3 (0x80000370)
0xa00f0064  e000eed9 lea     a14,[a14]0x380
0xa00f0068  0edc     ji a14
0xa00f006a  0000     nop
0xa00f006c  0000     nop
0xa00f006e  0000     nop
```

評価ボードでの IVT･TVT 上書きの試行

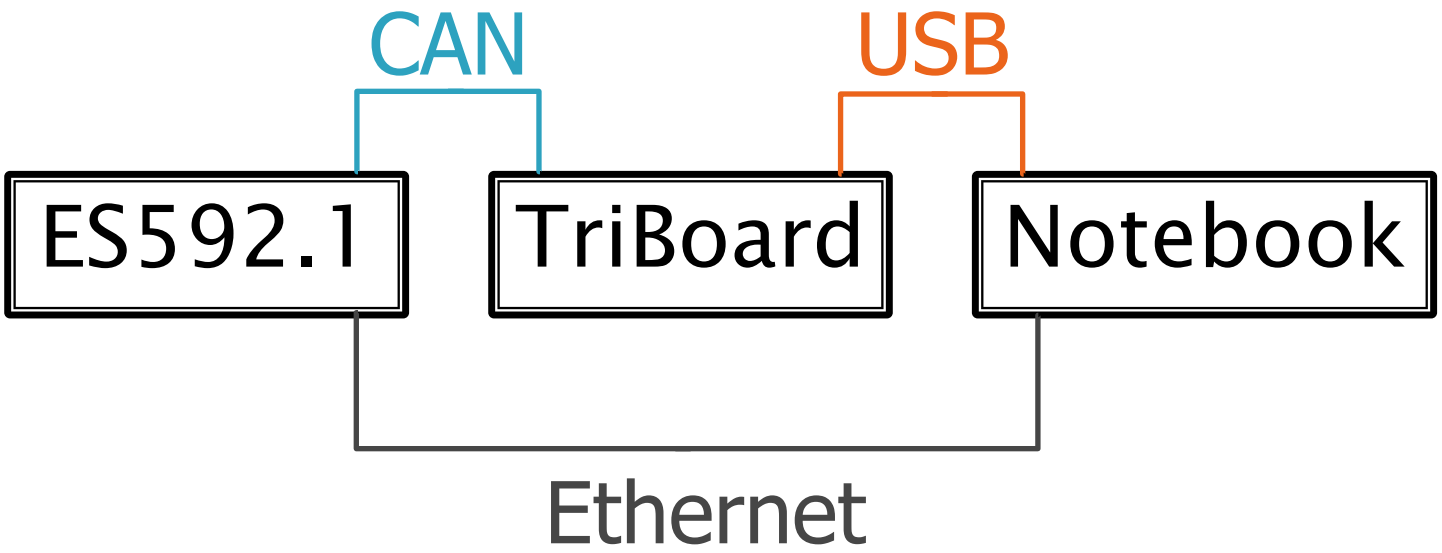
- 評価ボードでは BIV と BTV の値がシミュレータと異なる
  - BIT @ 0xF7E1FE20, BTV @ 0xF7E1FE24
- プロテクトがかかっている
- デバッガから書き換え可能
- プロテクトを無効にしてコードから書き換えを試行したがトラップ2が発生
- 実際のECUソフトウェアでの脆弱性攻撃にも適用できないと考えられる

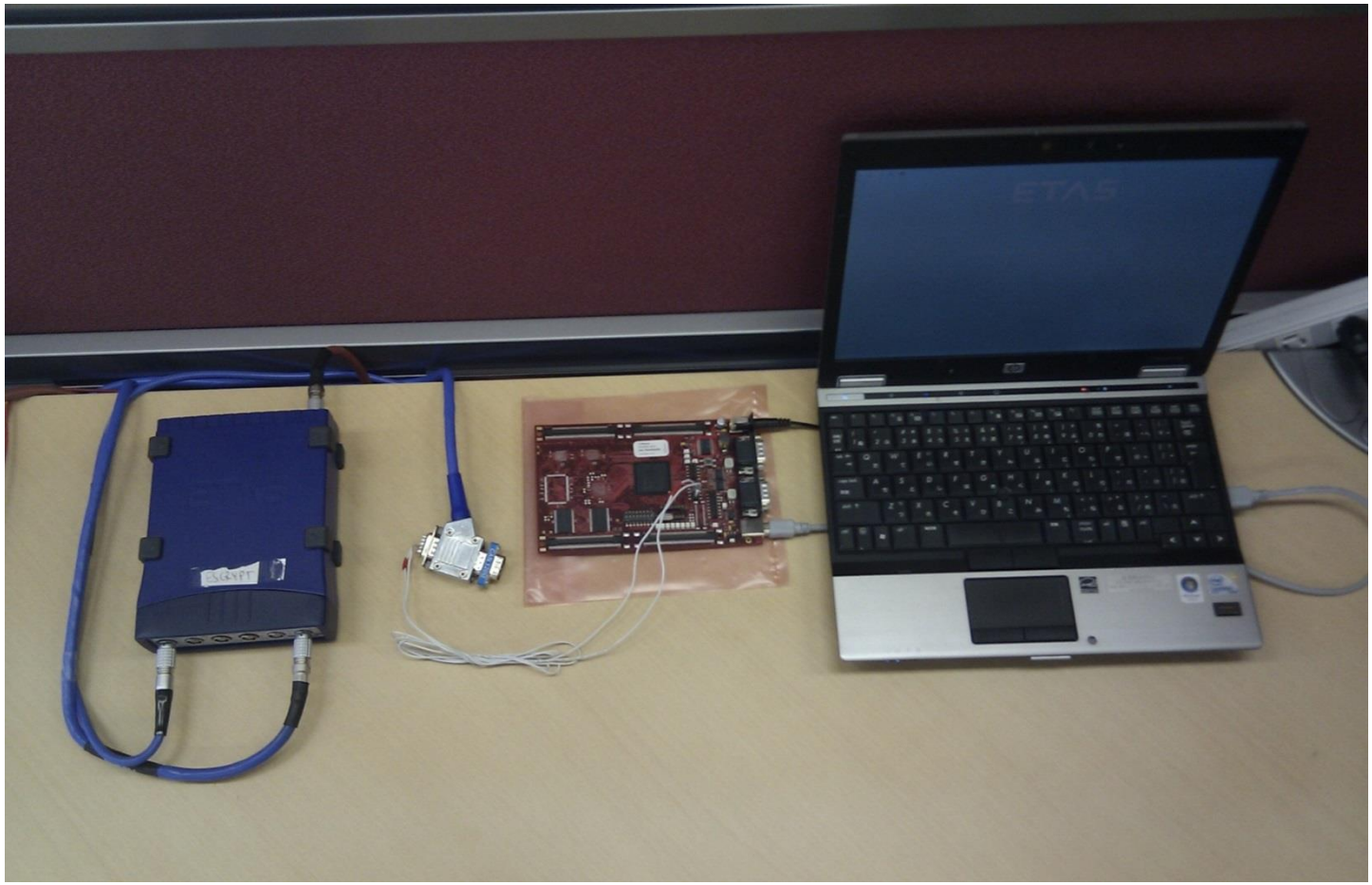
評価ボードを用いた検証

検証環境

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

メモリモニタリング手法




ETAS

デモ

まとめと今後の方針

まとめ

- ECU ソフトウェアにメモリ破壊脆弱性が存在する場合の攻撃手法を検討
- メモリ破壊脆弱性が存在する場合、任意コードを実行できる可能性はある
  - バッファオーバーフローが存在する場合、特定の条件で任意のコードを実行可能
  - CSAを改変することで任意のコードを実行可能
  - 割り込み・トラップベクタテーブルを改変することでの任意コード実行は不可能
- 脆弱なECUソフトウェアを作成し、攻撃のデモを実施
- 今回の調査は、理論的な攻撃手法の検討結果と脆弱なソフトウェアのサンプルを用いたデモのため、実際のECUソフトウェアの存在する脅威を指摘するものではない

今後の方針

- 追加調査
  - 今回調査したもの以外の脆弱性やアーキテクチャ固有の問題の調査
- 脅威の実証
  - ECUソフトウェアのリバースエンジニアリングを行い、メモリ破壊脆弱性が存在するかどうか調査
  - 実際の脆弱性を攻撃して、ECUの停止あるいは異常動作を引き起こすことが可能かどうか検証
- 対策検討
  - ECUソフトウェアの脆弱性対策手法の検討
  - プログラミングエラーに起因する脆弱性を効率的に発見する方法の検討

Thank you