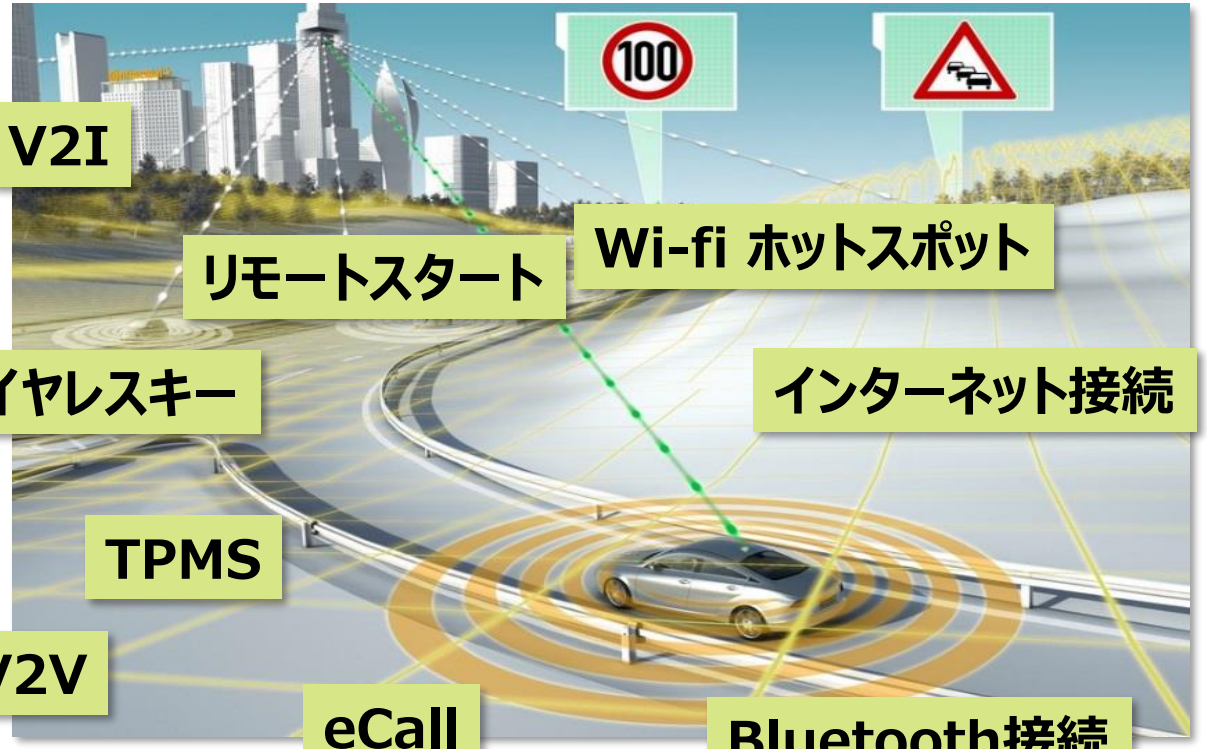
# ソフトウェアバグを根絶する静的コード解析
## ～組込みシステムのセキュリティ脆弱性の検出と安全性を証明～

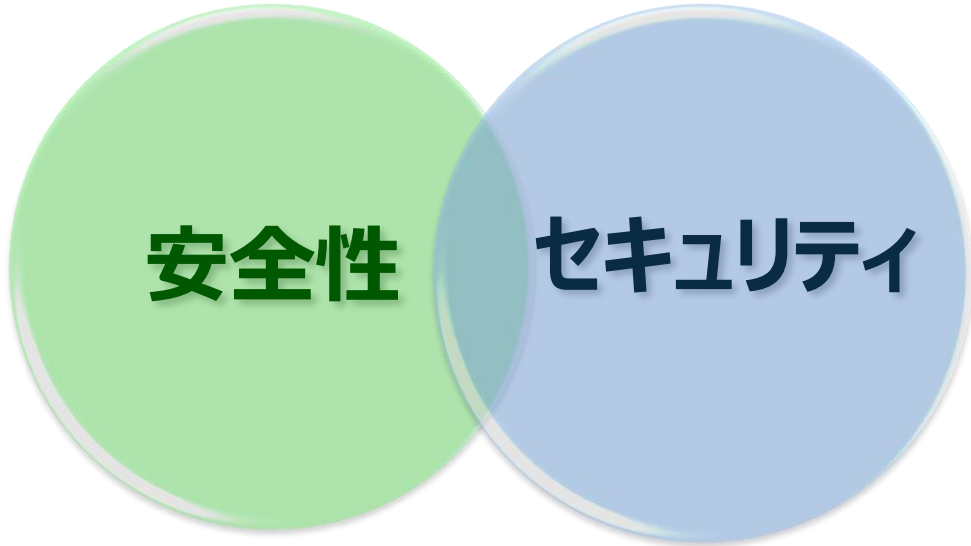
**MathWorks Japan**
**アプリケーションエンジニアリング部 (制御)**
**アプリケーションエンジニア**
**田中 康博**

# 自動運転のサイバーセキュリティ
## ソフトウェアは安全性だけではなくセキュリティも重要

車載通信 & V2X通信




- 通信の増加
- インターネット上のノード
- サイバー攻撃の脅威にさらされる



# 自動運転のサイバーセキュリティ (2)
## コードレベルでセキュリティを作り込むことが重要

### ソフトウェアの脆弱性：

**Jeepハック後に140万台をリコール**

出典：2015年
http://www.blogcdn.com/www.autoblog.com/media/2013/02/2014-jeep-cherokee-1.jpg

**セキュリティ強化後も
ハッキング可能だった**


出典：2017年
http://techon.nikkeibp.co.jp/atcl/column/15/425482/072800280/?P=3

### セキュリティ対策を困難にする要因:

- H/WとS/Wのスペック
- セキュリティに対する知識
- セキュリティレベルの差

**コードのセキュリティを
何で検証するのか？**

# Polyspace® – ソフトウェアのコード静的解析ソリューション
## コードのセキュリティと安全性の確保は静的解析が有効な手段

**Polyspace Bug Finder™**
による**セキュリティ脆弱性の検出**


**Polyspace Code Prover™**
による**安全性の証明**

# アジェンダ

1．サイバー攻撃の事例紹介

2．セキュリティ脆弱性の検出

3．ソフトウェアの安全性の証明

4．ユーザー事例 & まとめ

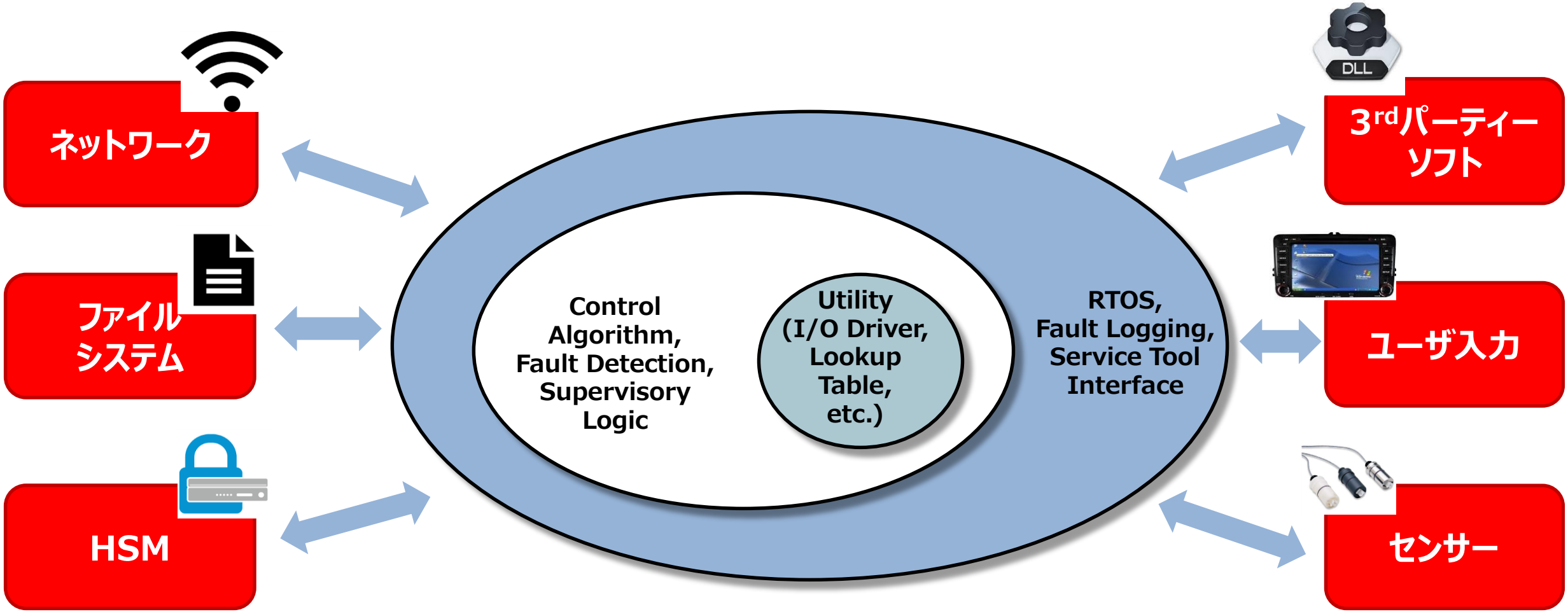
# Jeepハック

Miller (left) and Valasek demonstrated the rest of their attacks on the Jeep drove it around an empty parking lot. WHITNEY CURTIS FOR WIRED

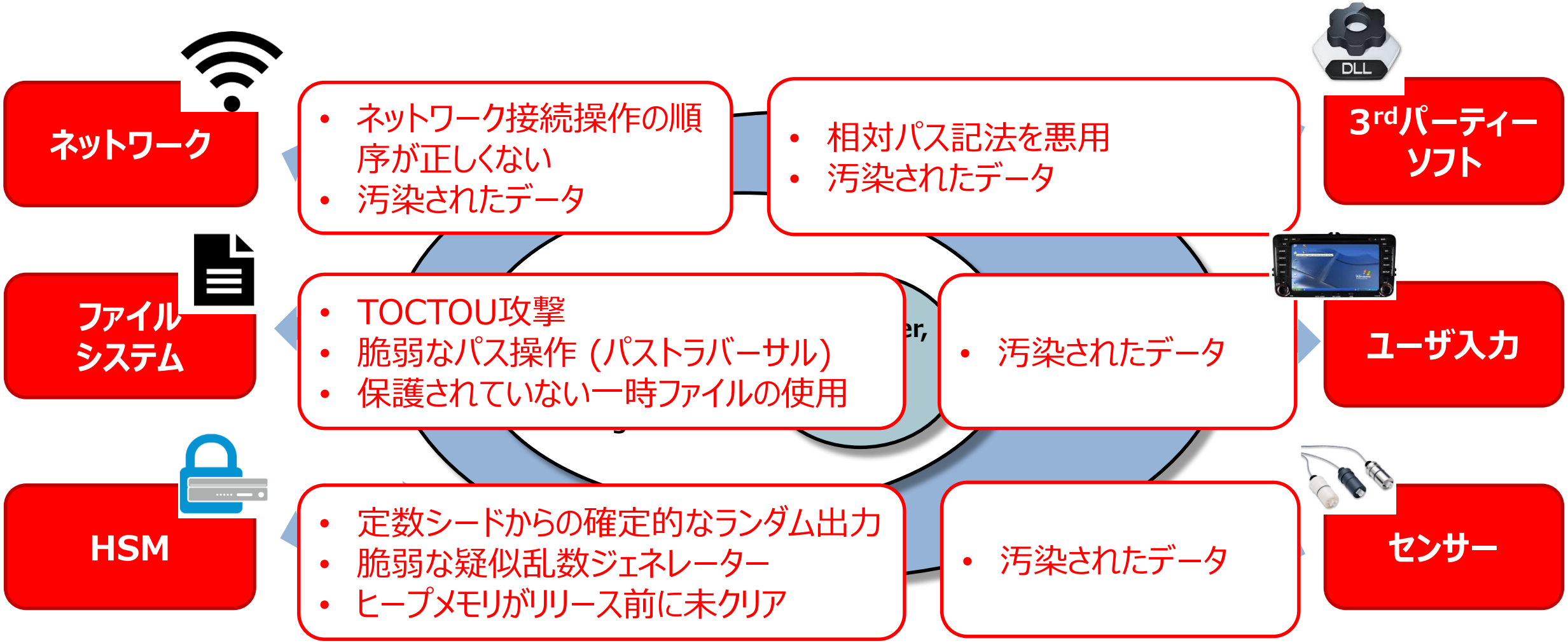
## HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

出典: 2015年 https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/

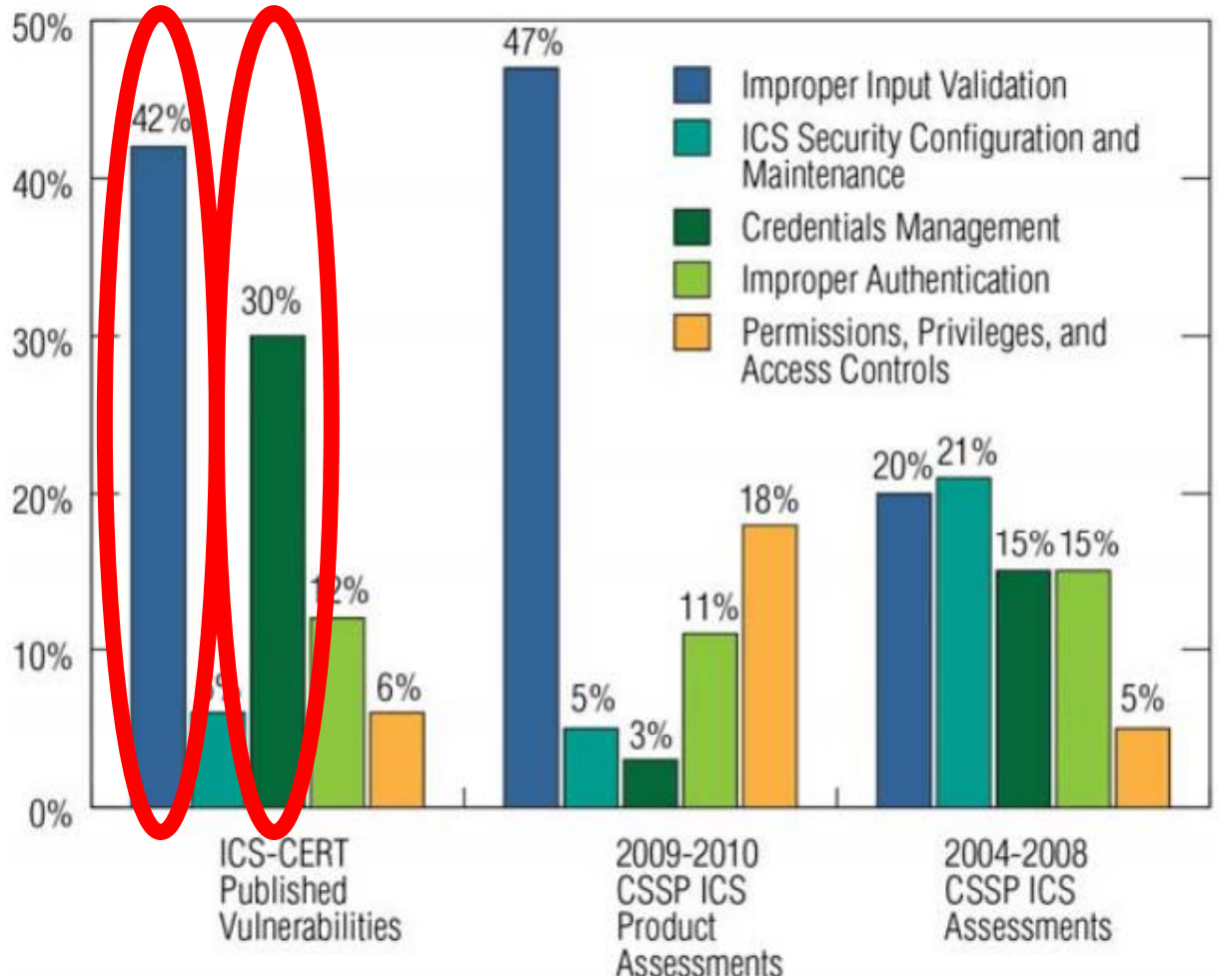
# 組込みソフトウェアのセキュリティ – 外部インターフェース
## 外部インターフェースを介して攻撃されるリスクが高まっている



# 組込みソフトウェアのセキュリティ – 脆弱性
## 外部インターフェースが多ければ攻撃されるリスクは更に高まる



# 組込みソフトウェアのセキュリティ脆弱性




**最大の問題:**

- 不適切な入力確認
- クレデンシャル管理

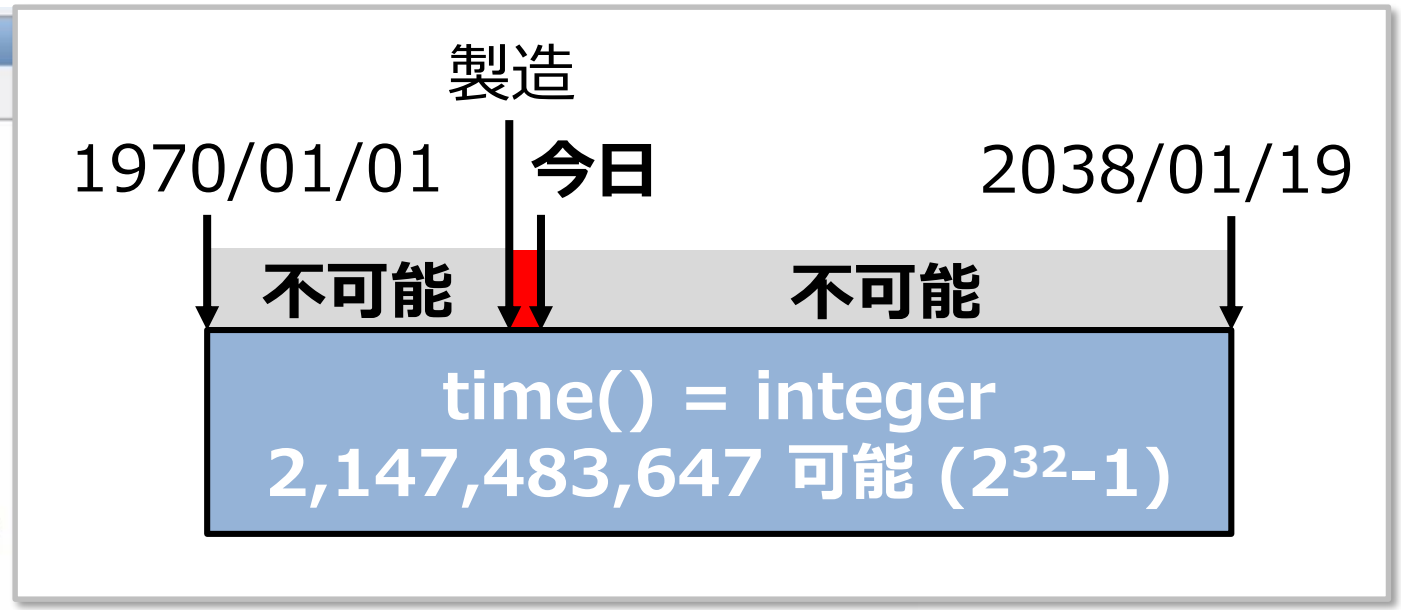
出典：2016年 *Industrial Control Systems (ICS) Cyber Emergency Response Team (ICS-CERT)*

# Jeepハック – 決定論的乱数発生器 (DRNG)

Source

wifi.c

```
21
22  char *get_password()
23  {
24          int c_max = 12;
25          int c_min = 8;
26          unsigned int t = time(NULL);
27          srand (t);
28          unsigned int len = (rand() % (c_max
29          char *password = malloc(len);
30          int v9 = 0;
31          do{
32                  unsigned int v10 = rand();
33                  int v11 = convert_byte_to_ascii_letter(v10 % 62);
34                  password[v9] = v11;
35                  v9++;
36          } while (len > v9);
37          return password;
38  }
```




**Polyspace Bug Finder**

# アジェンダ

1．サイバー攻撃の事例紹介

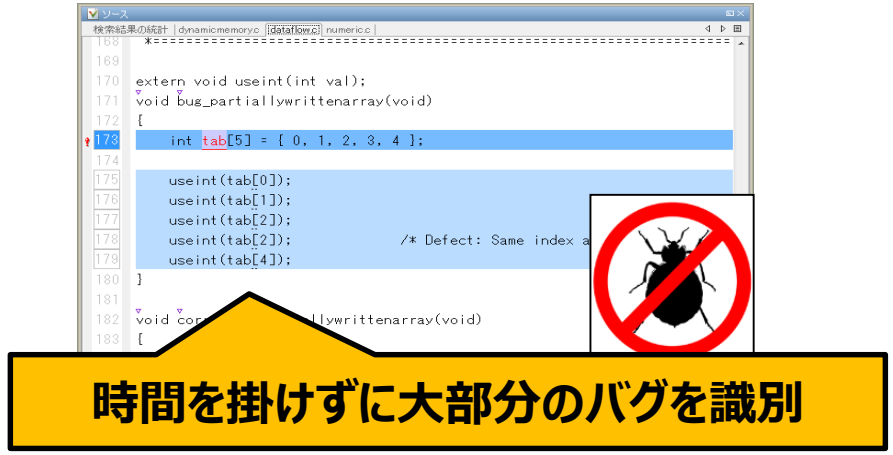
**2．セキュリティ脆弱性の検出**

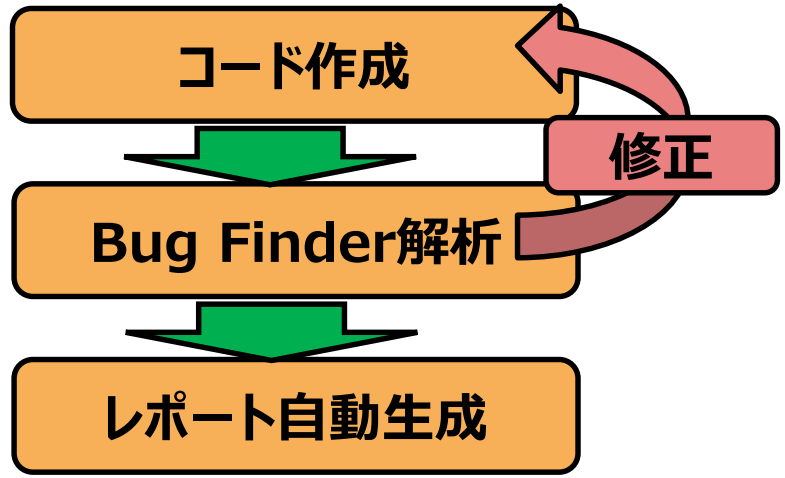
3．ソフトウェアの安全性の証明

4．ユーザー事例 & まとめ

# Polyspace Bug Finder のスピーディー解析
## 素早い欠陥検出・修正を可能とする！

- **セキュリティ脆弱性・バグ検出**
  - コード作成後、すぐに欠陥を検出・修正
- **コーディングルールチェック**
  - エラー予防と再利用性向上
  - MISRA準拠を確認
- **コードメトリックス解析**
  - コードの複雑度を測定




**開発プロセスの上流で不具合を発見！**
**コードを統合前に多くの欠陥を修正！**
**コード開発の効率に繋がる！**

# セキュリティ脆弱性・バグ検出項目

## 数値
- ゼロ割、オーバーフロー
- 標準ライブラリ数学ルーチンの無効な使用
- …

## 動的メモリ
- メモリリーク
- 前に解放したポインターの使用
- …

## データフロー
- 読取りのない書込み
- 未初期化変数
- …

## リソース管理
- 読取り専用リソースに書込み
- 以前に閉じられたリソースを使用
- …

## 静的メモリ
- 配列の範囲外アクセス
- NULLポインター
- …

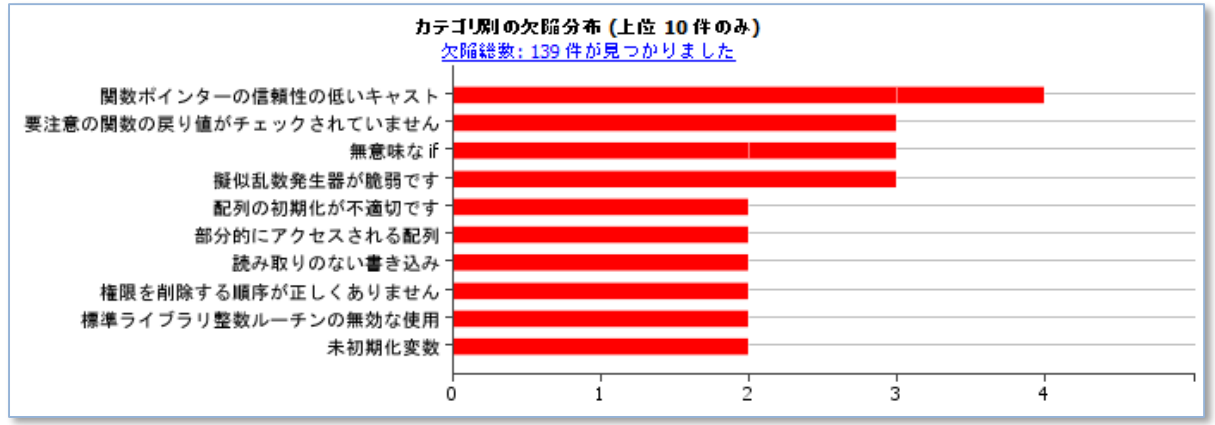
## セキュリティ
- パス操作が脆弱
- 疑似乱数発生器が脆弱
- …

## 同時実行
- データレース
- デッドロック
- …

## 適切な手法
- 未使用のパラメータ
- 値渡しの大きな引数
- …

## プログラミング
- 等号演算子による浮動小数点の比較
- 宣言の不一致
- …

## 汚染されたデータ
- 汚染された文字列形式
- 汚染されたサイズでのメモリの割り当て
- …




https://jp.mathworks.com/help/bugfinder/defect-reference.html

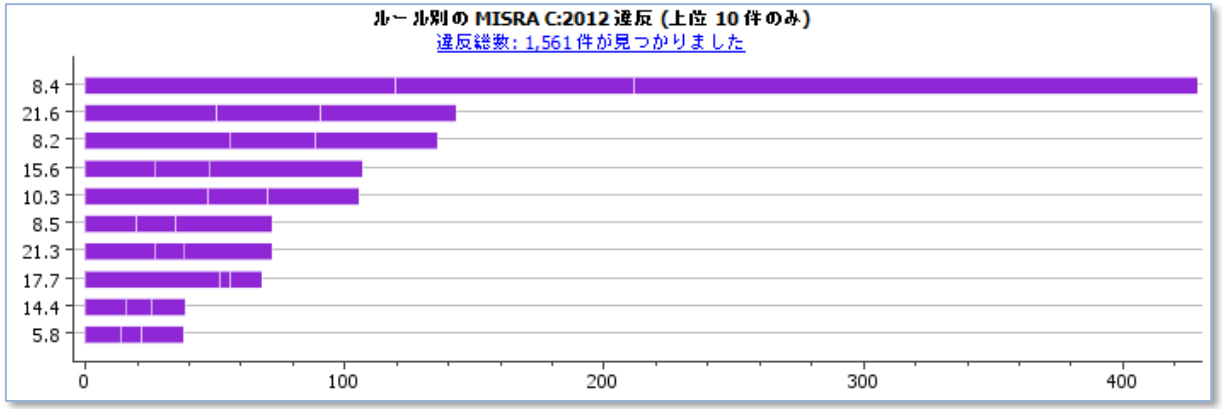
# サイバーセキュリティ – 業界活動と標準

## 自動車業界に見られるコーディング標準＆実践

- **CERT C**：セキュアコーディングスタンダード
- **ISO/IEC TS 17961**：Cセキュアコーディングルール
- **CWE**：共通脆弱性タイプ
- **MISRA-C:2012** Amendment 1：セキュリティ

# コーディングルールチェック

- **MISRA C® : 2004** チェッカー
- **MISRA C® : 2004 AC AGC** チェッカー
  - 自動生成コード用
- **MISRA C® : 2012** チェッカー
- **MISRA® C++** チェッカー
- **JSF®++** チェッカー




MISRA C:2012 Amendment 1

Additional security guidelines for MISRA C:2012

April 2016

- ✓ **新しいコーディング規約** に対応
  (全ての required と mandatory)
- ✓ **新しい指針 (directive) : 1**
- ✓ **新しいルール (rules) : 13**
- ✓ 既存のルール変更 [21.8]

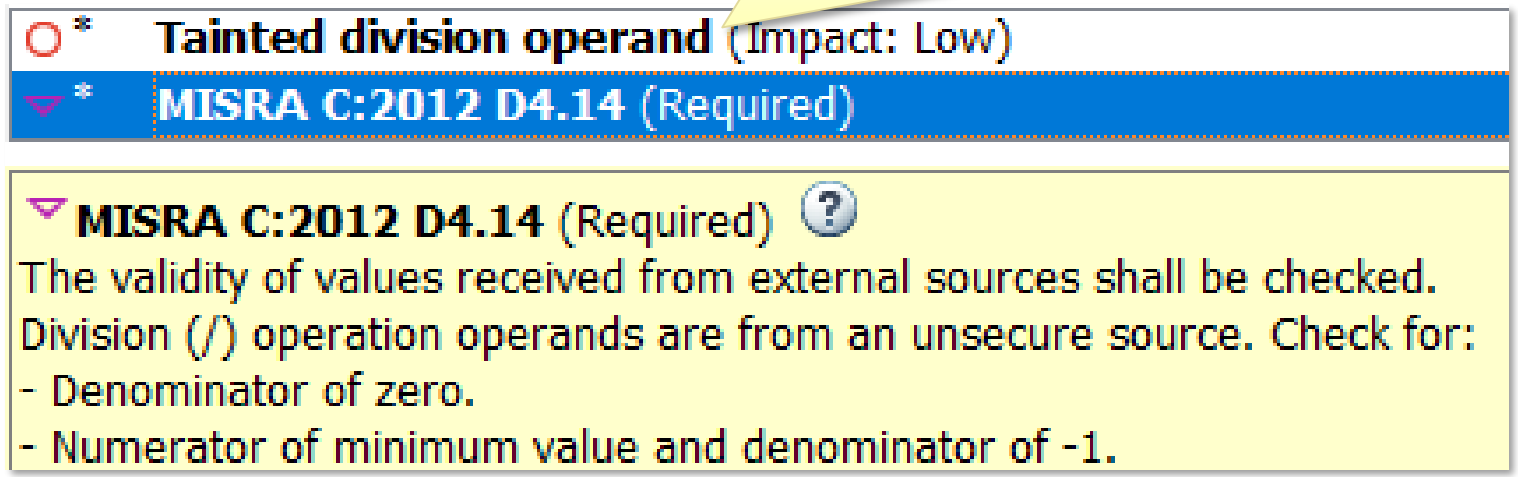
# MISRA-C:2012 改訂1 の新しい指針4.14の例

```
int corrected_taintedintdivision(int usernum, int userden) {
    int r = 0;

    if (userden!=0 && !(usernum=INT_MIN && userden==-1)) {
        r = usernum/userden;
    }

    print_int(r);
    return r;
}
```

汚染されたデータによる除算



# アジェンダ

1．サイバー攻撃の事例紹介

2．セキュリティ脆弱性の検出

**3．ソフトウェアの安全性の証明**

4．ユーザー事例 & まとめ

# バグを見つけることができますか？

ソース

検索結果の統計 | WhereAreTheErrors.c

```
1  int new_position(int sensor_pos1, int sensor_pos2)
2  {
3  int actuator_position;
4  int x, y, tmp, magnitude;
5
6  actuator_position = 2;  /* default*/
7  tmp = 0;                          /* values */
8  magnitude = sensor_pos1 / 100;
9  y = magnitude + 5;
10
11 while (actuator_position < 10)
12         {
13         actuator_position++;
14         tmp += sensor_pos2 / 100;
15         y += 3;
16         }
17 if ((3*magnitude + 100) > 43)
18         {
19         magnitude++;
20         x = actuator_position;
21         actuator_position = x / (x - y);
22         }
23 return actuator_position*magnitude;     /* value */
24 }
```

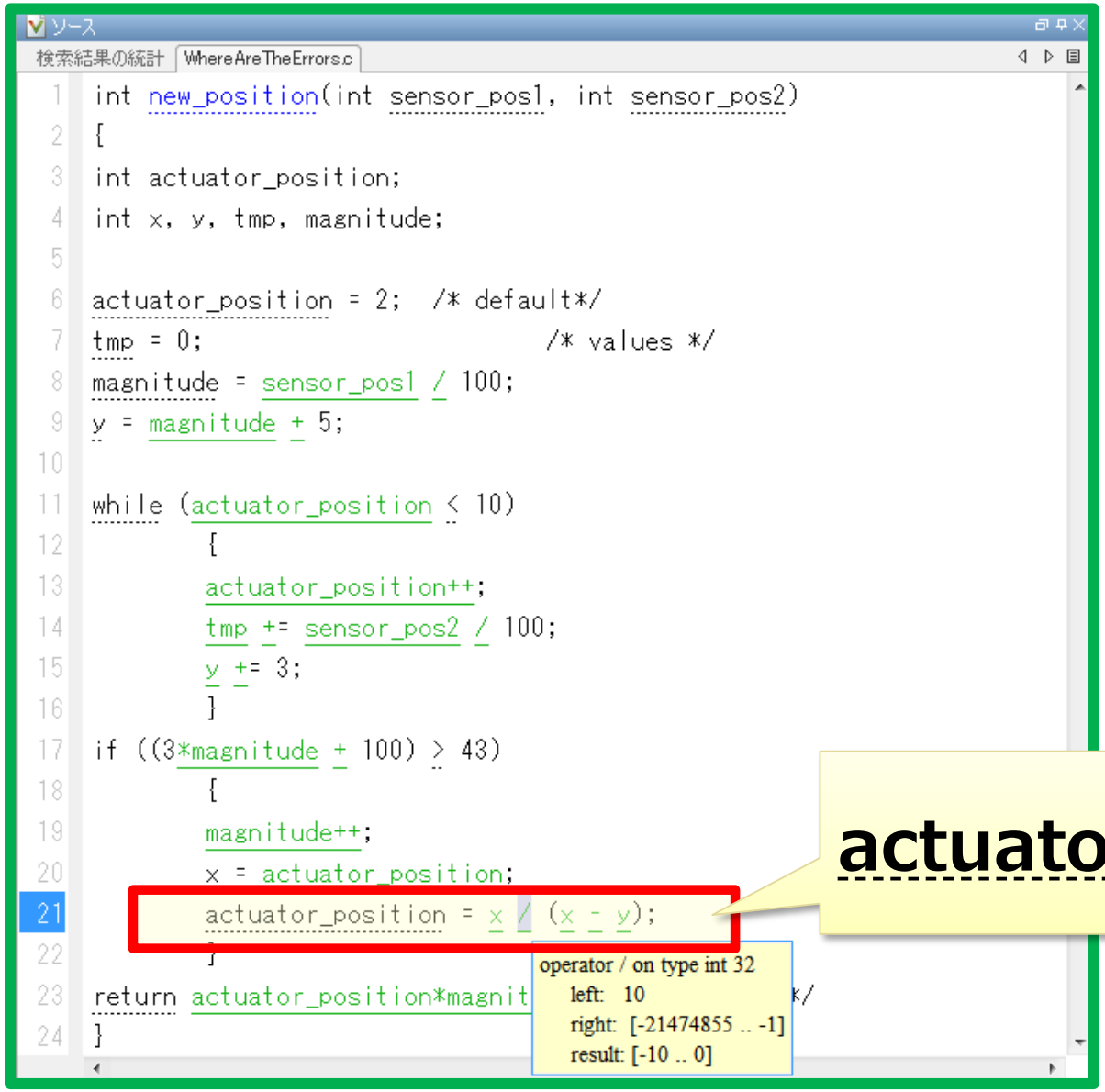
**ゼロ割**の可能性

**オーバーフロー**の可能性

**未初期化**の可能性

**actuator_position = x / (x - y);**

# Polyspace Code Prover の実行結果



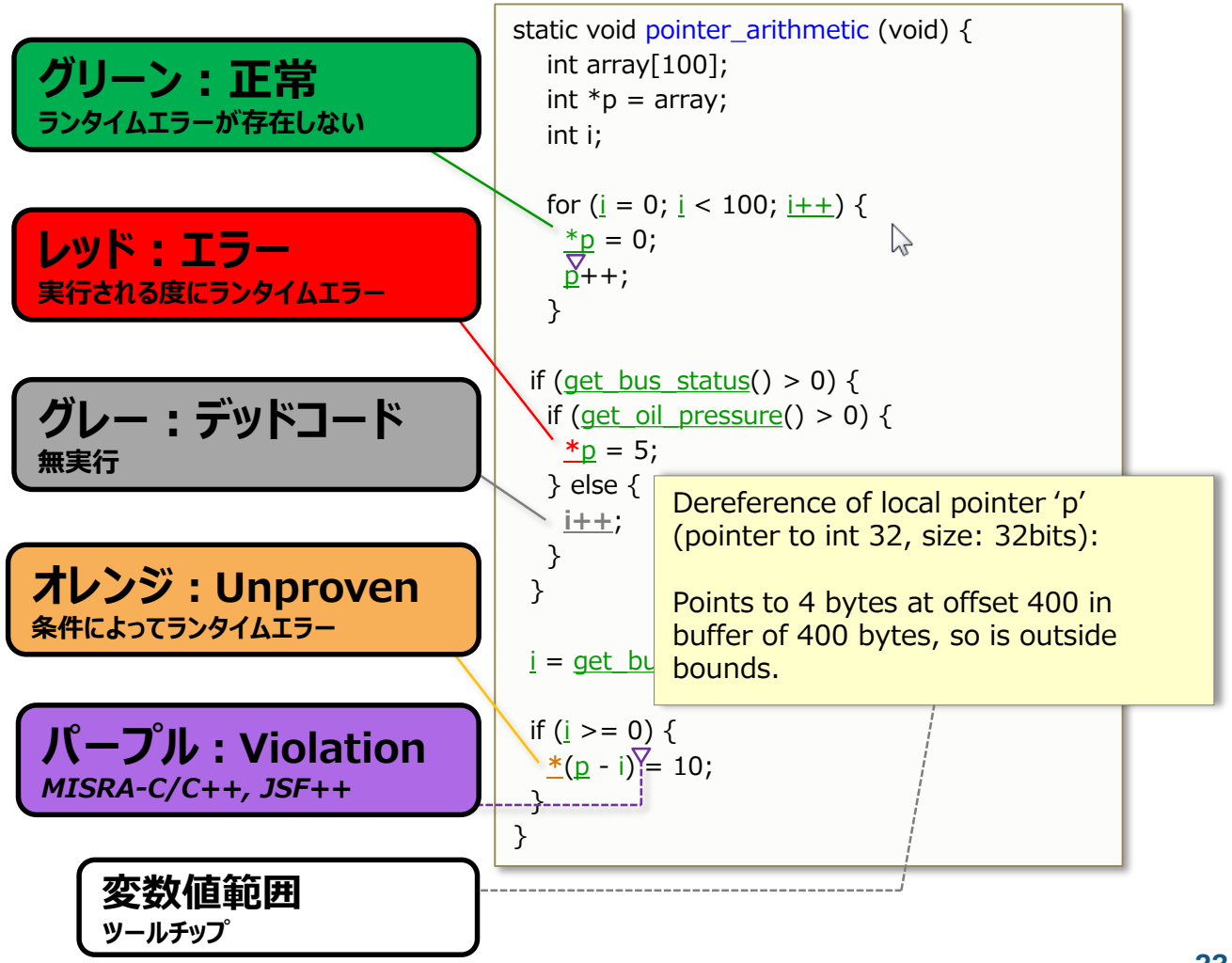

- ✓ あらゆる条件でコードの**安全性**を証明
- ✓ テストを行わずに網羅的に解析
  - **実行時エラー**
  - **条件による実行時エラー**
  - **到達不能なコード**

**actuator_position = x / (x - y);**

# Polyspace Code Prover でコードの正しさを証明
## 全ての実行パスの結果を証明する！

- **Quality（品質）**
  - ランタイムエラーの証明
  - 測定、向上、管理
- **Usage（使用方法）**
  - コンパイル、プログラム実行、テストケースは不要
  - 対応言語：C/C++/Ada
- **Process（プロセス）**
  - ランタイムエラーの早期検出
  - 自動生成コード、ハンドコードの解析可能
  - コードの信頼性を測定

**実機実験前にコード信頼性を確保して手戻りを削減**

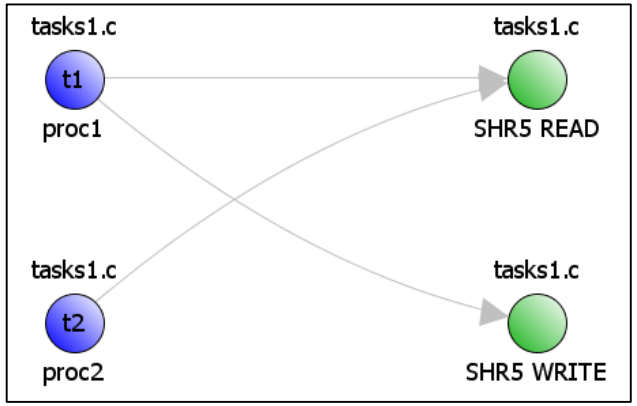


# Polyspace Code Prover – ソフトウェアの安全性の確保

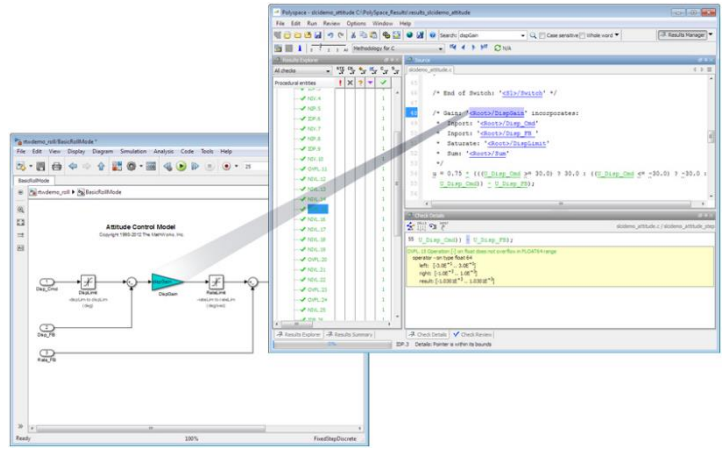
- **MISRAルール違反の安全性の証明**


- **機能安全規格に準拠**


- **グローバル変数の同時アクセスの特定**



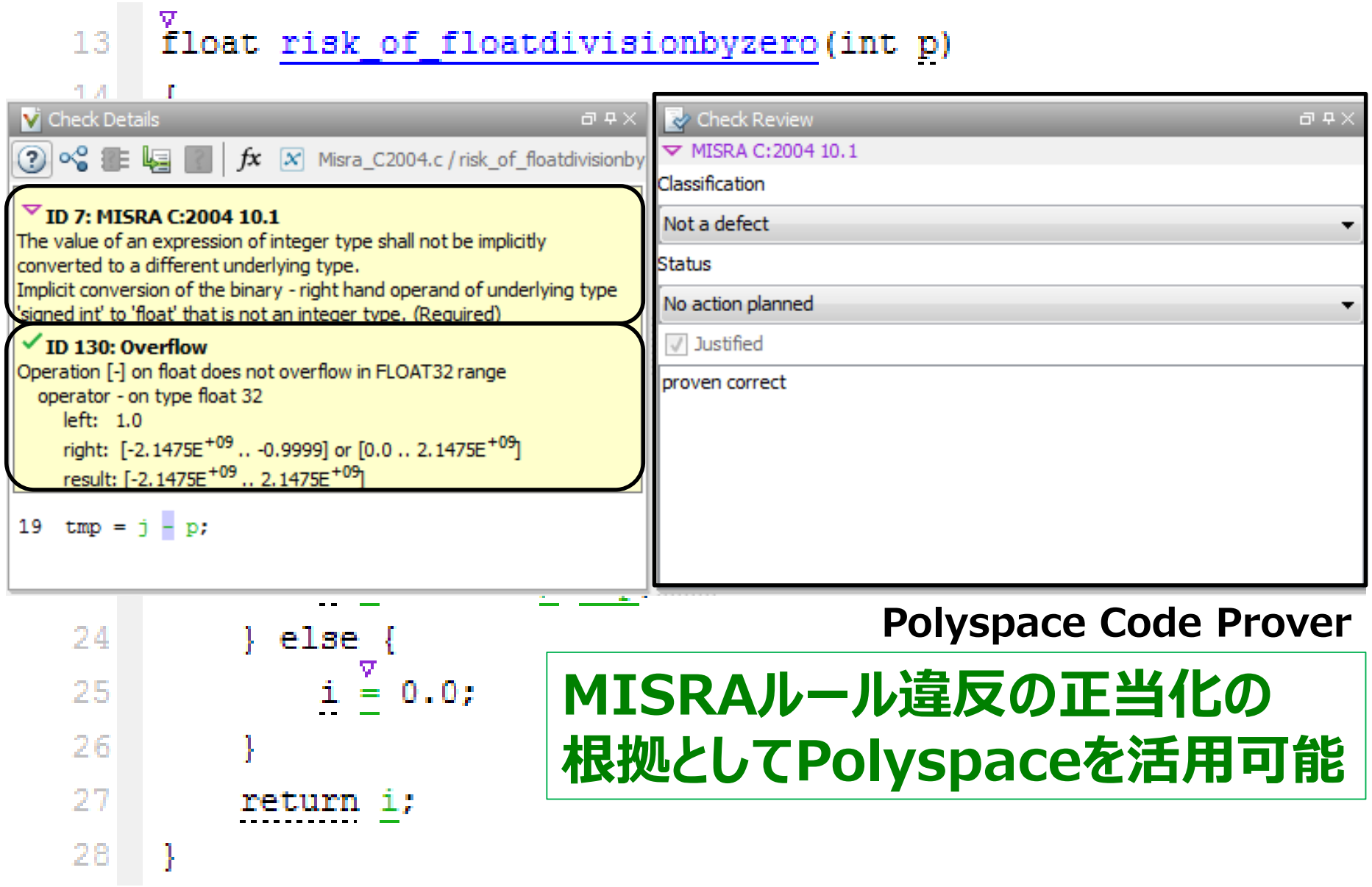

- **MBDとの統合による安全性の確保**



# MISRAルール違反の安全性の証明

**MISRA ルール違反**

**安全性の証明 (オーバーフロー無し)**




Polyspace Code Prover

**MISRAルール違反の正当化の根拠としてPolyspaceを活用可能**

# グローバル変数の同時アクセスの特定
# マルチタスク時のレビューを効率化

tasks1.c

```
29  static int SHR5 = 5;
```

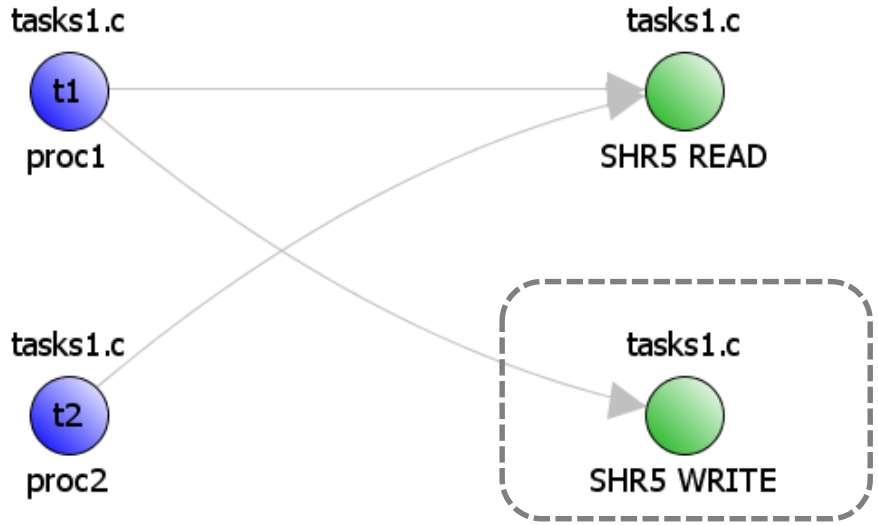
## 対話的なレビュー手法

保護された変数

Variable 'tasks1.SHR5' is shared among several tasks.
読み取り元タスク: proc1 proc2
書き込み元タスク: proc1

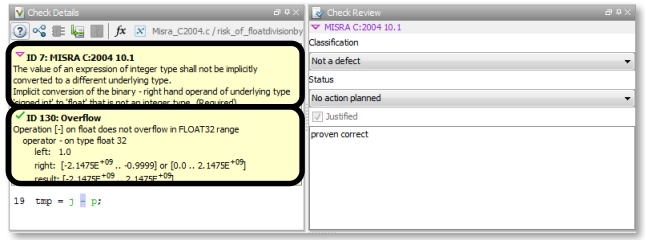
| | イベント | ファイル | スコープ | 行 |
|---|---|---|---|---|
| ◀ | 書き込み値: 5 | tasks1.c | _init_globals() | 29 |
| ◀ | 書き込み値: 28 | tasks1.c | proc1() | 103 |
| ▶ | 読み取り値: 5 | tasks1.c | proc1() | 103 |
| ▶ | 読み取り値: 5 or 28 | tasks1.c | proc2() | 112 |

## グラフィカルなレビュー手法

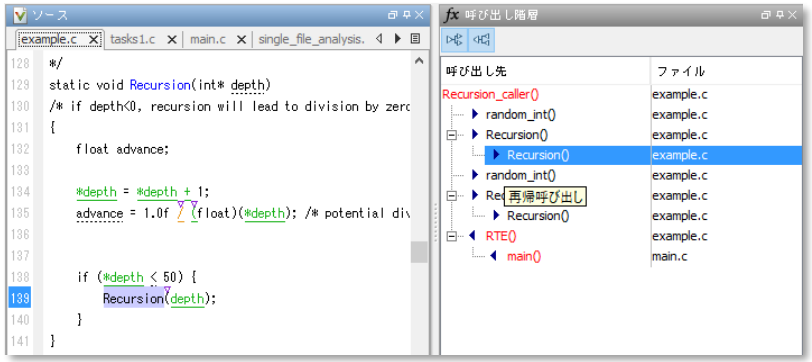



**グリーン**：同時アクセスから保護
グレー　：定義しているが未使用
**オレンジ**：競合の可能性あり

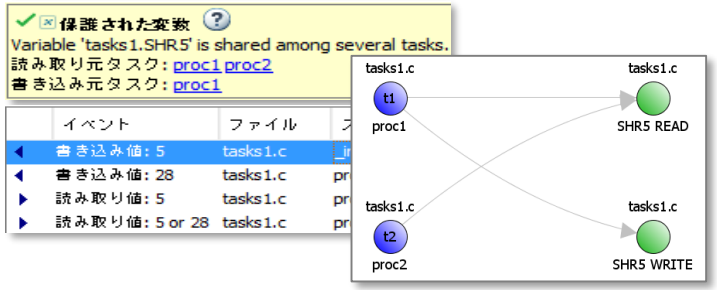
# Polyspace Code Prover – ISO26262適用




MISRAルール違反の安全性の証明




再帰呼び出しの識別




グローバル変数の同時アクセスの特定

**Table 8 – Design principles for software unit design and implementation ソフトウェアユニット設計と実装の基本原則**

| Methods | | ASIL (安全性要求レベル) | | | |
|---|---|---|---|---|---|
| | | A(低) | B | C | D(高) |
| 1g | No implicit type conversions 暗黙の型変換の禁止 | + | ++ | ++ | ++ |
| 1j | No recursions 再帰呼び出しの禁止 | + | + | ++ | ++ |

++：強く推奨、+：推奨

**Table 9 – Methods for the verification of software unit design and implementation ソフトウェアユニット設計の検証方法**

| Methods | | ASIL (安全性要求レベル) | | | |
|---|---|---|---|---|---|
| | | A(低) | B | C | D(高) |
| 1g | Data flow analysis データフロー解析 | + | + | ++ | ++ |

++：強く推奨、+：推奨

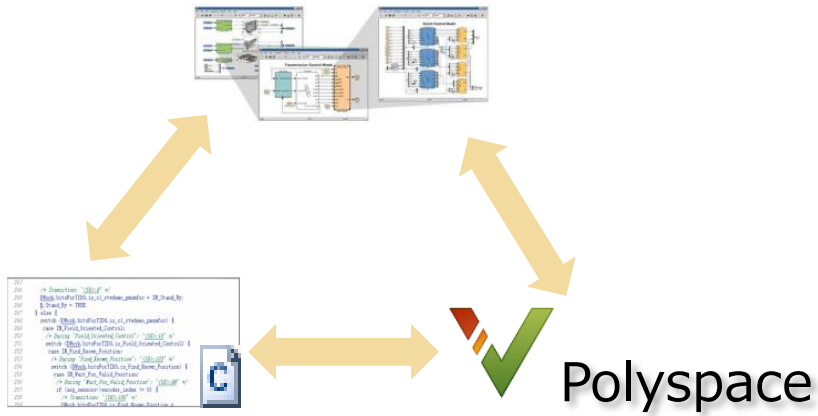
INTERNATIONAL STANDARD **ISO 26262-6**
First edition 2011-11-15

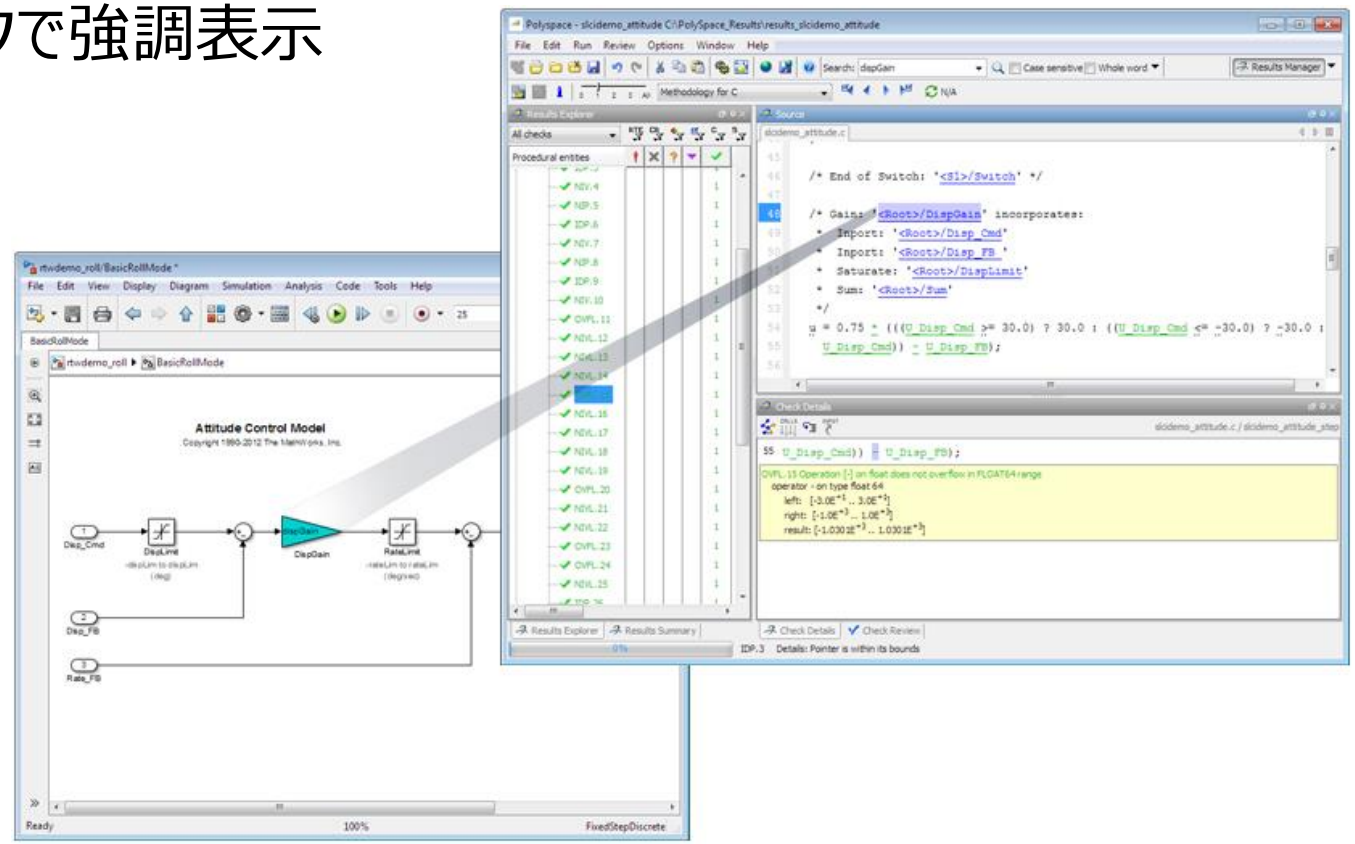
# 規格準拠に向けて
# Polyspace製品で目標規格達成をアシストします！

- コーディング規約
  - MISRA-C
  - MISRA-C++
  - JSF++
- ソフトウェアメトリクス
  - HIS Source Code Metrics
    - http://www.automotive-his.de/
- 認証規格
  - DO-178B
    - DO Qualification Kit
  - IEC 61508, ISO 26262, EN 50128
    - IEC Certification Kit

# モデルベースデザインとの統合
## PolyspaceのSimulink連携機能でモデル修正が容易

- Simulink®環境からPolyspace解析を実行
- Polyspace解析結果を該当ブロックで強調表示
- ブロックにPolyspaceの注釈を追加
- S-Functionコードの検証
- MISRA C のチェック



# アジェンダ

1．サイバー攻撃の事例紹介

2．セキュリティ脆弱性の検出

3．ソフトウェアの安全性の証明

**4．ユーザー事例 & まとめ**

# Polyspace ユーザー事例
# 医療、航空、自動車など様々な分野で利用されています

## Miracor Eliminates Run-Time Errors and Reduces Testing Time for Class III Medical Device Software

> ‘‘From a developer’s perspective, the main advantage of Polyspace Code Prover is a higher level of quality and correctness in the code. Polyspace Code Prover helps Miracor demonstrate this quality and correctness to the regulatory community, including the FDA, to prove that our device is safe.’’
> *- Lars Schiemanck, Miracor Medical Systems*

https://jp.mathworks.com/company/user_stories/miracor-eliminates-run-time-errors-and-reduces-testing-time-for-class-iii-medical-device-software.html

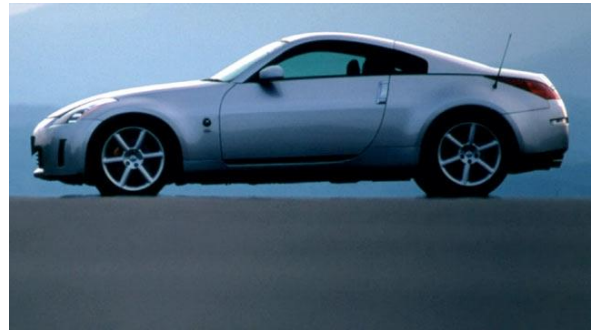
## Solar Impulse Develops Advanced Solar-Powered Airplane

> ‘‘From a developer’s perspective, the main advantage of Polyspace Code Prover is a higher level of quality and correctness in the code. Polyspace Code Prover helps Miracor demonstrate this quality and correctness to the regulatory community, including the FDA, to prove that our device is safe.’’
> *- Lars Schiemanck, Miracor Medical Systems*

https://jp.mathworks.com/company/user_stories/solar-impulse-develops-advanced-solar-powered-airplane.html

## 日産自動車
## ソフトウェアの信頼性を向上

> **「Polyspace 製品によって、高レベルなソフトウェアの信頼性を確保することができます。これは、業界内の他のツールにはできないことです。」**
> - 菊池光彦氏, 日産自動車



### 課題
ソフトウェア品質を向上させるために、発見が困難なランタイムエラーを特定する

### ソリューション
MathWorks のPolyspace製品を使用して、日産とサプライヤーのコードを包括的に解析する

### 結果
サプライヤーのバグを検出して評価
ソフトウェアの信頼性が向上
日産のサプライヤーが Polyspace 製品を採用

https://jp.mathworks.com/company/user_stories/nissan-increases-software-reliability.html

# Polyspace – まとめ
## 静的解析によるコード検証でセキュリティと安全性の確保

**Polyspace Bug Finder™ によるセキュリティ脆弱性の検出**

- ✓ セキュリティ脆弱性とバグの早期検出
- ✓ コーディングルールのチェック
- ✓ セキュリティ項目のドキュメンテーション & 推奨アクション

**Polyspace Code Prover™ による安全性の証明**

- ✓ あらゆる条件でコードの安全性を証明
- ✓ ソフトウェアの安全性の確保
  - MISRAルール違反の安全性の証明
  - グローバル変数の同時アクセスの特定
  - 機能安全規格のツール認証に対応
  - MBDとの統合による安全性の確保

# Polyspace – まとめ

## 静的解析によるコード検証でセキュリティと安全性の確保

**Polyspace Bug Finder™**
によるセキュリティ脆弱性の検出

**Polyspace Code Prover™**
による安全性の証明

# セキュリティと安全性の確保で ソフトウェアの品質を向上する Polyspace静的解析ソリューション

# Polyspace の有効活用に向けて
## 短期間で習得していただけるような教育カリキュラムをご提供します





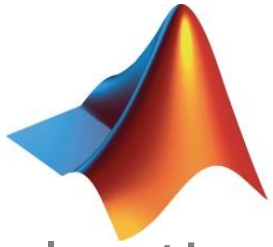
### ■ トレーニング サービス

- ➢ 定期 トレーニング
  - ✓ 東京、名古屋、大阪にて定期開催
  - ✓ 基礎コース(11)、応用コース(11)、専門コース(4)を ご提供
- ➢ オンサイト トレーニング
  - ✓ お客様サイトにて開催
  - ✓ ご要望に応じて3つのレベルでカリキュラムのカスタマイズが可能

### ■ コンサルティング サービス

- ➢ カスタム“Jumpstart”
  - ✓ 顧客モデルをベースにした短期集中型ツール導入サポート
- ➢ Advisory Service
  - ✓ 顧客Project に合わせた中長期アドバイザリサービス

# ご清聴ありがとうございました



MathWorks®

Accelerating the pace of engineering and science