LDRA

Taint analysis, functional safety, cybersecurity, and you

テイント解析、機能安全、サイバーセキュリティ

Taint analysis, functional safety, cybersecurity, and you

Mark Pitchford has over 30 years' experience in software development for engineering applications. He has worked on many significant industrial and commercial projects in development and management, both in the UK and internationally. Since 2001, he has worked with development teams looking to achieve compliant software development in safety- and security-critical environments, working with standards such as DO-178, IEC 61508, ISO 26262, IIRA, and RAMI 4.0. Mark earned his Bachelor of Science degree at Trent University, Nottingham, and he has been a Chartered Engineer for over 20 years. He now works as Technical Specialist with LDRA Software Technology. https://www.edn.com/author/mark-pitchford/

Mark Pitchford は、エンジニアリングアプリケーション向けのソフトウェア開発で 30 年以上の経験があります。英国および海外で、多くの重要な産業および商業プロジェクトの開発と管理に携わってきました。2001 年以降、セーフティクリティカル、セキュリティクリティカルな環境で規格準拠のソフトウェア開発を目指す開発チームと協力し、DO-178、IEC 61508、ISO 26262、IIRA、RAMI 4.0 などの規格に取り組んできました。Mark はノッティンガムのトレント大学で理学士号を取得し、20 年以上にわたり公認エンジニアとして活躍しています。現在は、LDRA Software Technology でテクニカルスペシャリストとして働いています。

本講演のオリジナル版　https://ldra.com/events/taint-analysis-webinar-2/

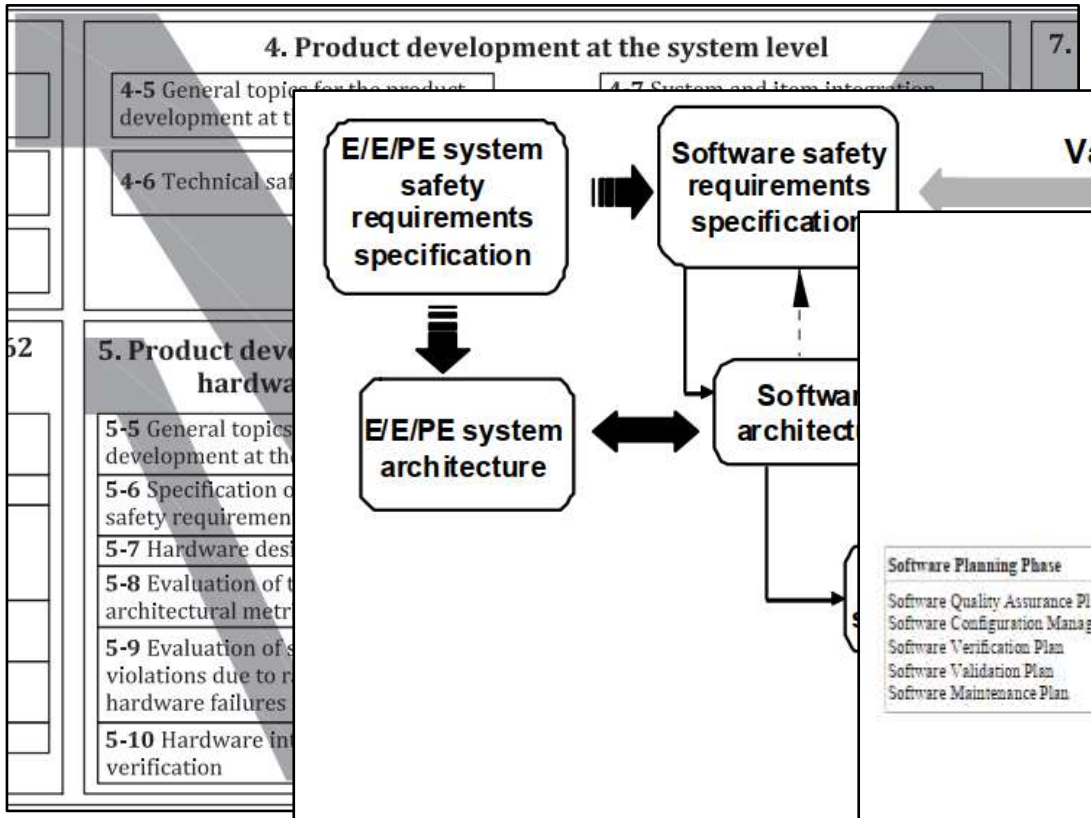
Agenda

1. Functional safety & cybersecurity
2. CWE, CVE, and the automotive software stack
3. Taint considerations at design time
4. Taint considerations at coding time
5. Summary and conclusions

機能安全規格

航空宇宙	DO-178C (First published 1992)
産業機器	IEC 61508 (First published 1998, Updated 2010)
鉄道	EN 5012X (First published 2001)
原子力	IEC 61513 (First published 2001)
自動車	ISO 26262 (Published 2011)
医療機器	IEC 62304 (First published 2006)
プロセス	IEC 61511 (First published 2003)
…	

機能安全規格とV 字モデル



ISO 26262
(自動車)

IEC 61508 (産業機器)

EN 50128
(鉄道システム)

セキュリティ分野で行われてきた従来の取り組み

Reactive

- ガイドライン無し
- リスク軽減無し

- 信頼性が無い
- 悪意あるロジックに対して不十分
- 回復力が無い

- パフォーマンステスト
- 侵入テスト
- 負荷テスト
- 機能テスト

セキュリティ分野で行われてきた従来の取り組み

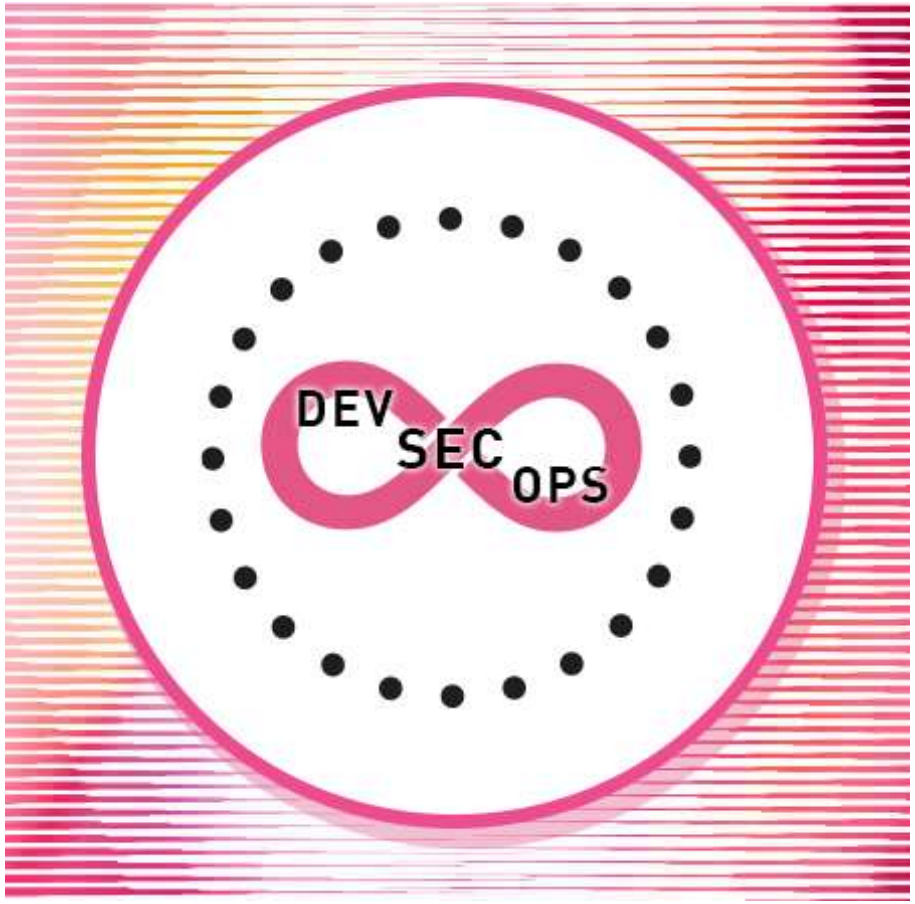
Reactive

Coding | Executable

- ガイドライン無し
- リスク軽減無し

- 信頼性が無い
- 悪意あるロジックに対して不十分
- 回復力が無い

シフトレフト：セキュリティを開発プロセスの早期段階に取り込む



https://www.checkpoint.com/cyber-hub/cloud-security/what-is-shift-left-security/

What is Shift Left Security?

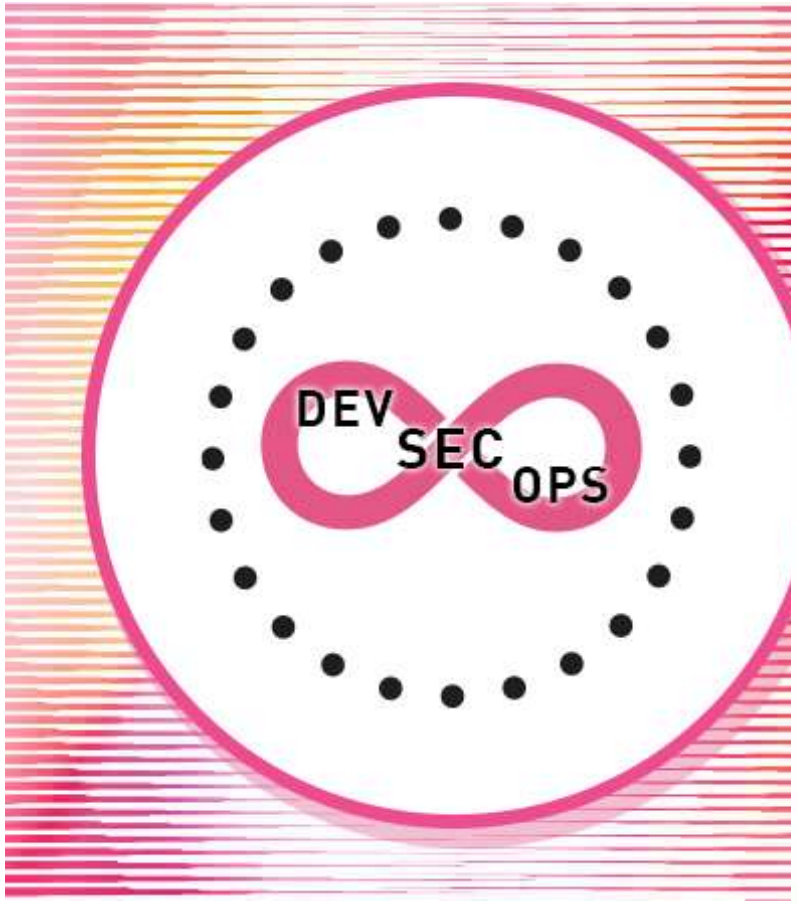
Shift left refers to moving security sooner in the development process. Graphing the process of application development, with time as the X axis, the process begins with recognition of a need that a technology or service will fulfill, whether it's an application being developed for sale to paying customers or for internal use. As the solution moved through the stages of conception, design, develop, build, and test, security was often a final step, prior to depl
outs . And
this
Addi t the
proc ing it
on a

「以前は、セキュリティはリリースされる前にアプリケーションの外側に単にラップされるだけで、それは必然的に時間を費やしていた」

「後から追加するよりも、プロセス全体を通じてセキュリティをより緊密に統合することで安全性は向上する」

Shift left is the way to remedy these problems.

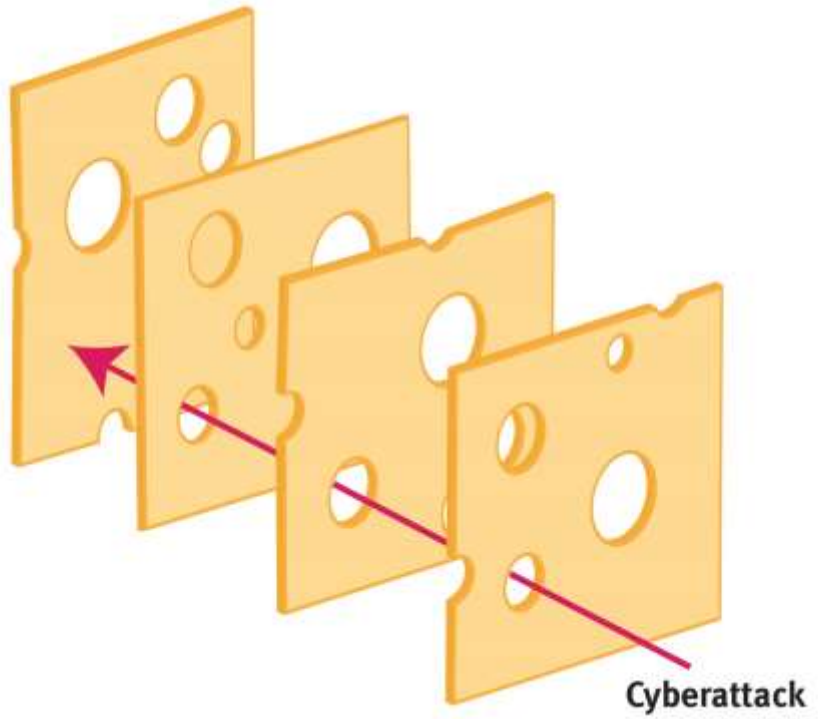
シフトレフト



https://www.checkpoint.com/cyber-hub/cyber-security/what-is-shift-left-security/

スイスチーズモデルとサイバーセキュリティ

サイバーセキュリティは、アーキテクチャ、設計、開発プロセスのあらゆる部分における警戒に依存する



- 「正しい」イメージがロードされていることを確認するセキュアブート
- システムの重要な部分を保護するドメイン分離
- 脆弱性を最小限に抑えるための MILS（最小権限）設計原則
- 攻撃対象領域の最小化
- セキュアコーディング
- セキュリティに重点を置いたテスト

これらのテクニックはどれも絶対確実というわけではない。しかし、これらを組み合わせることで、サイバー攻撃が成功するリスクを最小限に抑えることができる。セキュアコーディングは、この多層防御アプローチの重要な部分となる

セキュアコーディングとは?

- セキュアコーディングには様々な定義がある
- 多くの権威ある団体（CWE、OWASP など）は、一連のルールによってセキュアコーディングの概念を参照している
- 以下の定義は、本講演のテーマでもある、機能安全とセキュリティを同時に解決するといった目的に即している

  - セキュアコーディングとは、最高のセキュリティ規格とベストプラクティスに準拠したコード設計の原則を指す
  - セキュアコーディングはセキュリティを優先し、既知および未知の脆弱性からコードを保護する
  - セキュアコーディングでは、コードのセキュリティの責任はセキュリティチームではなく開発者に委ねられる

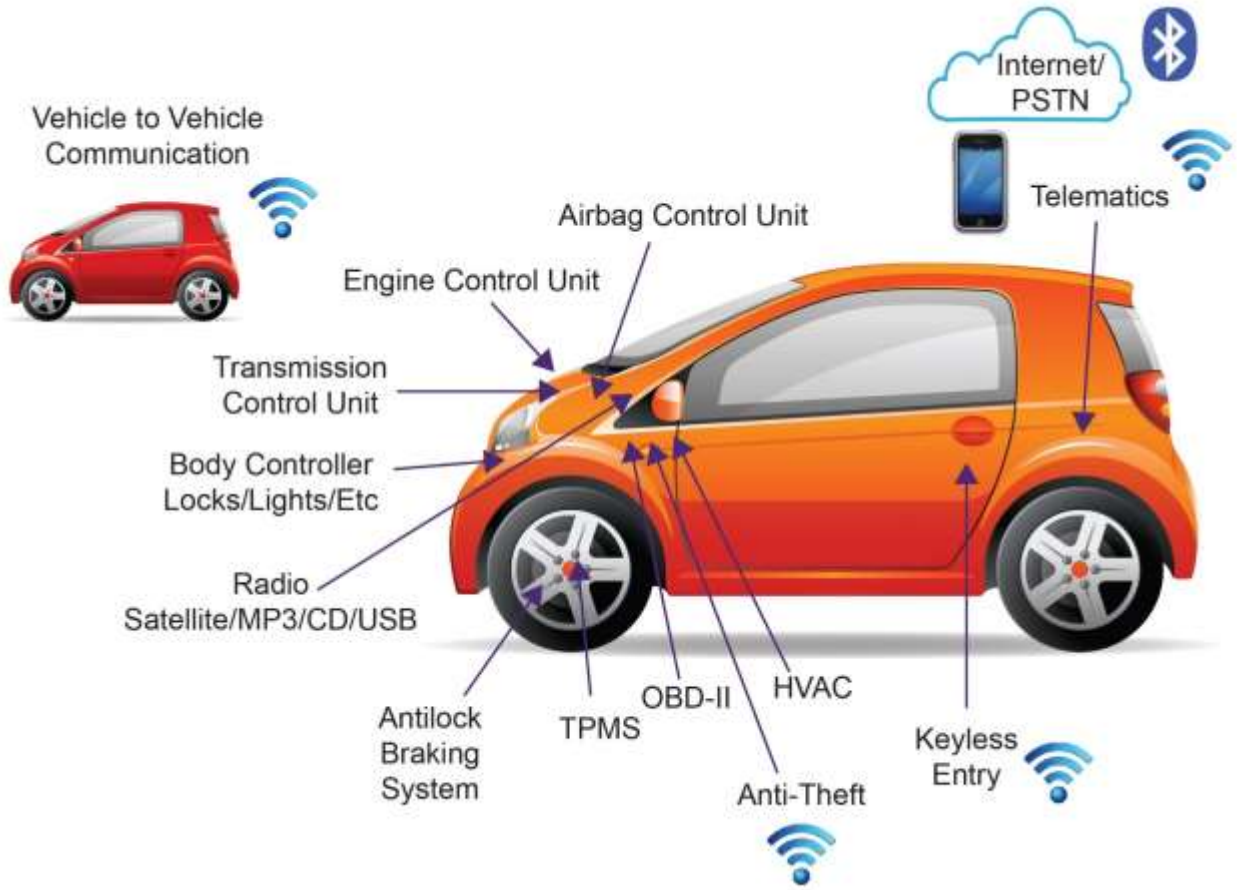
https://nordvpn.com/cybersecurity/glossary/secure-coding

機能安全プロセスとセキュアコーディング

CWE, CVE, and the automotive software stack

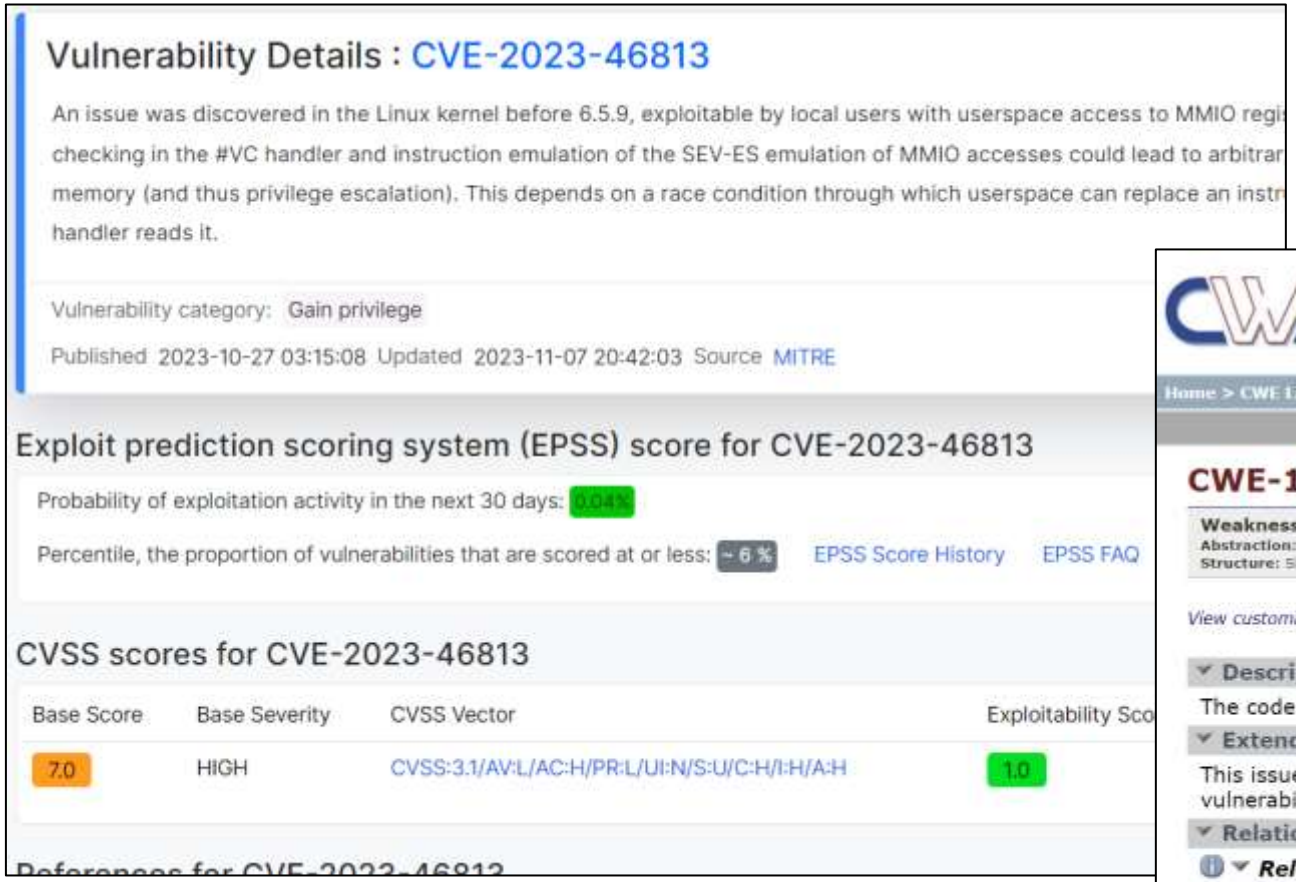
セキュリティの必要性の出現

- 他の業界と同様に、機能安全（ISO 26262）が確立されてきたが、、



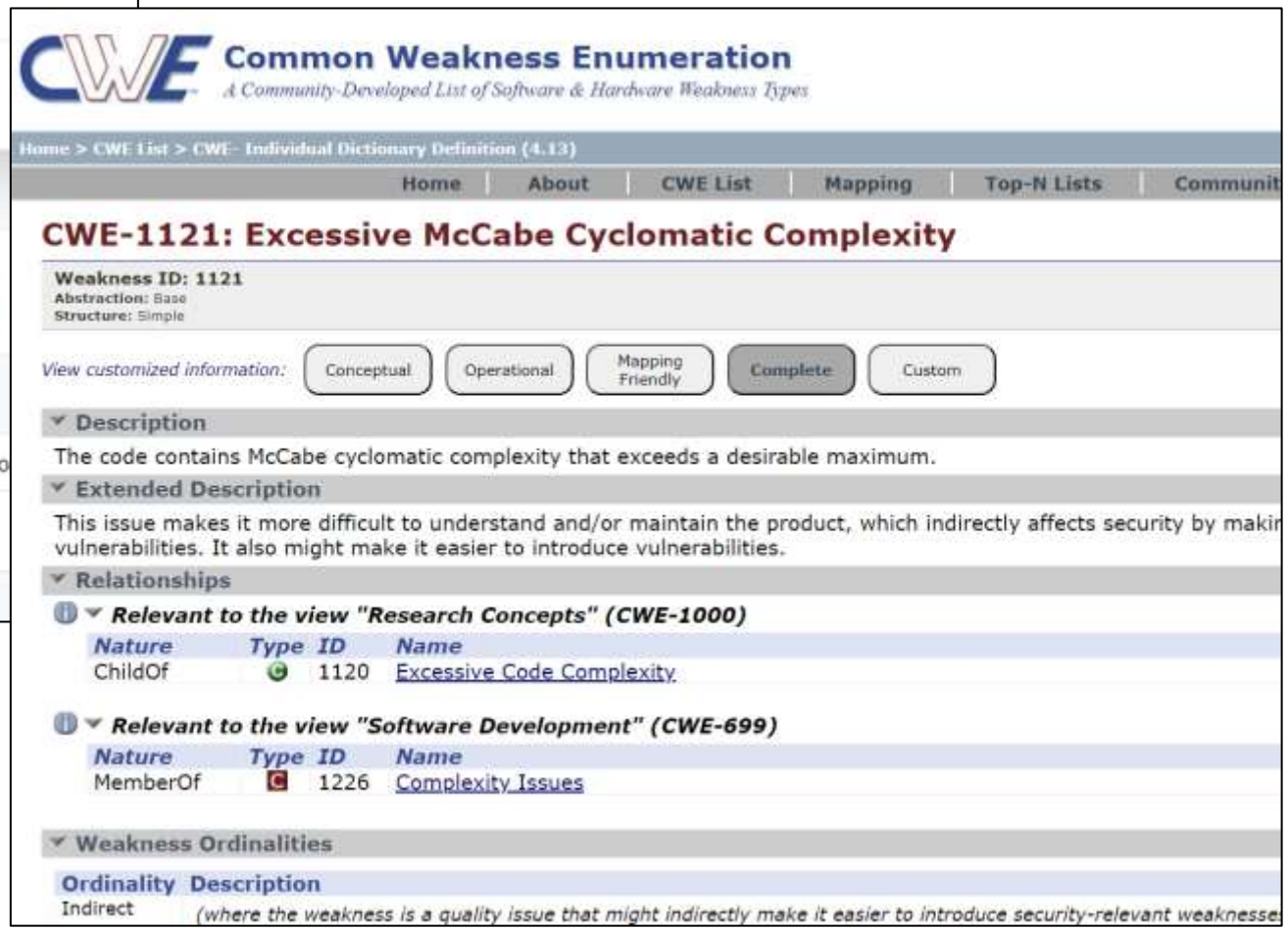
コネクテッドカーは、静的で固定されたデバイス固有のアプリケーションから、クラウド接続への移行により、監視・モニタリング、アップグレード、機能強化、そして付加価値を容易に提供できるようになる一方で、セキュリティの脆弱性が課題になる

CVE & CWE –脆弱性（Vulnerabilities）と 弱点（Weaknesses）



これは CVE の**脆弱性**の例で、特定の製品またはパッケージ（この場合は Linux カーネル）の特定の問題の説明

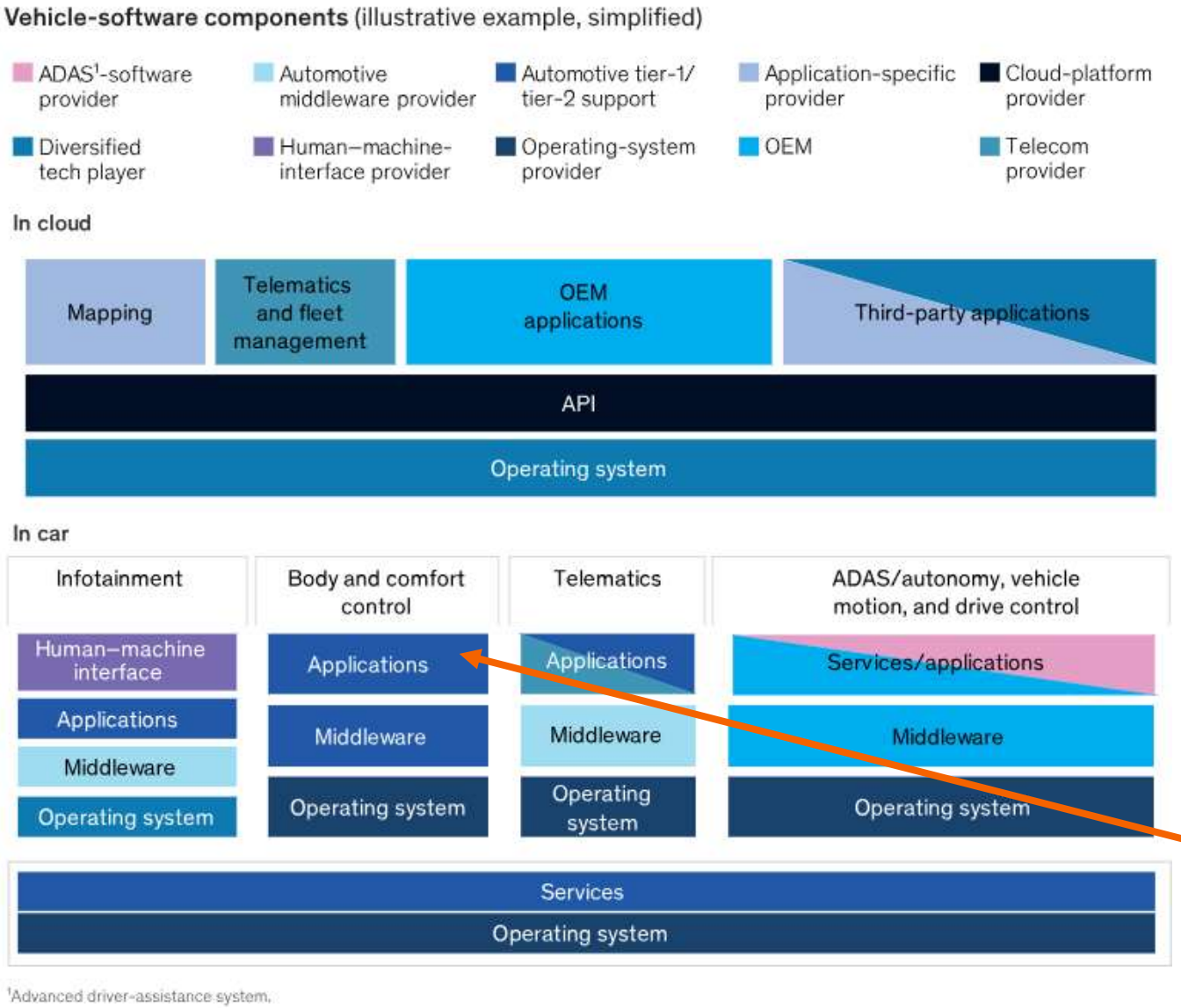
https://www.cvedetails.com/cve/CVE-2023-46813/?q=CVE-2023-46813



これは CWE での**弱点**の例で、脆弱性を招かないようにするために回避すべき状況の説明

https://cwe.mitre.org/data/definitions/1121.html

隠れた部分のコードサイズが重要、、



[1]Advanced driver-assistance system.

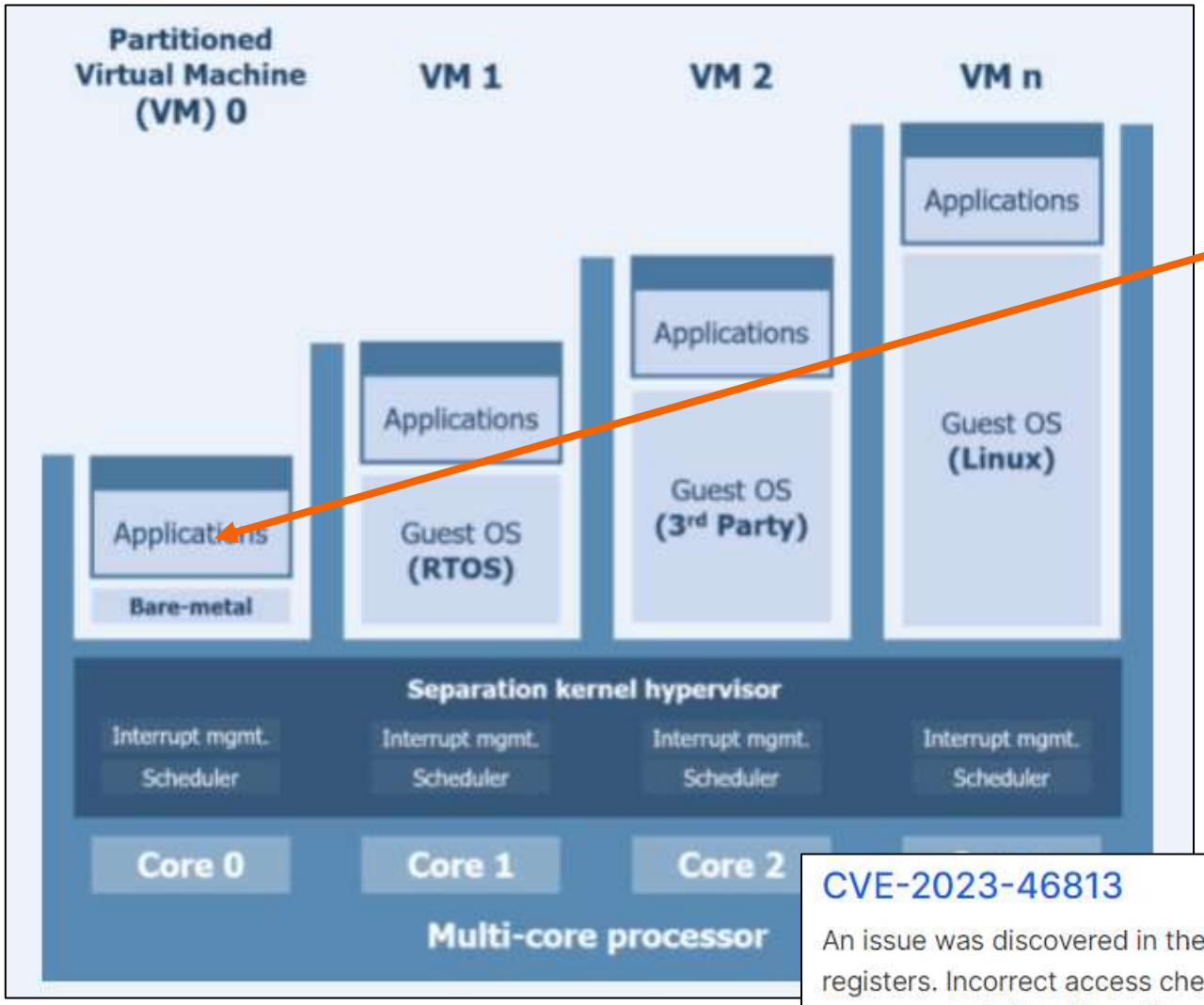
https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-case-for-an-end-to-end-automotive-software-platform

- LDRA の解析ツールは、新たに開発されるコードに主眼を置いている
- Manfred Broy 教授によると、現代の高級車には「おそらく 1 億行近くのソフトウェア コードが含まれている」

  https://www.motorauthority.com/news/1026505_modern-luxury-vehicles-claimed-to-feature-more-software-than-a-fighter-jet/

- ただ、それらは新しいモデルごとにゼロから実装されるわけではない
  - コードベースの大部分はレガシーコード、つまり COTS、またはオープンソース
  - それだけでなく、サードパーティのコードの多くも、独自のレガシーコード、COTS、またはオープンソースも使用
- この矢印部分のアプリケーションは、全体からすると小さな一部であるが、、

典型的な事例！



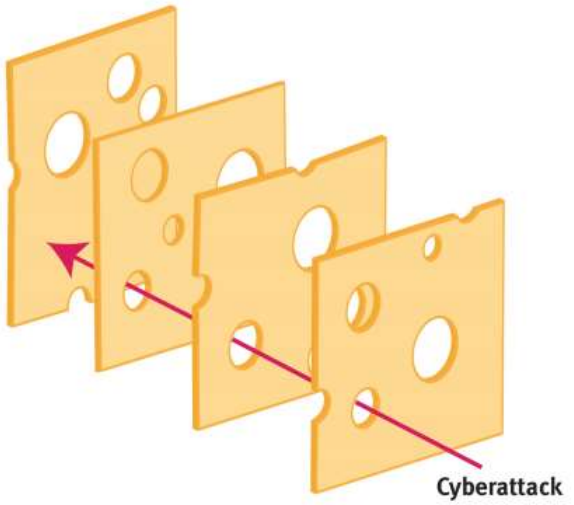
- 図はハイパーバイザーを展開する典型例
- 各OS はオープンソースか市販製品
- ここで Buildroot や Yocto(またはその他) を使用して、小さなアプリケーション用に最小限の Linux OS を作成したと仮定
  - 殆どの部分はLinux カーネル
  - cvedetails.com で、「linux kernel」で検索すると、46,600 件の結果が表示される
  - 一例として、実行可能なコード内に、この脆弱性が含まれてるかどうかを、どうすればわかるか?

6.5.9 より前の Linux カーネルで、ユーザー空間から MMIO レジスタにアクセスできるローカル ユーザーが悪用できる問題が発見されました。#VC ハンドラーでのアクセス チェックが正しくなく、、、

CVE-2023-46813

An issue was discovered in the Linux kernel before 6.5.9, exploitable by local users with userspace access to MMIO registers. Incorrect access checking in the #VC handler and instruction emulation of the SEV-ES emulation of MMIO accesses could lead to arbitrary write access to kernel memory (and thus privilege escalation). This depends on a race condition through which userspace can replace an instruction before the #VC handler reads it.

Swiss cheese…



- バイナリスキャナー　BlackBerry Jarvis や Mend.io など
- 実行可能ファイルのパターンマッチングを行って、コードに危険な脆弱性が含まれているかどうかを確認する
- もし脆弱性が見つかれば、それを軽減するか、別のビルドを使用する
- ソースコードが無い場合、それが唯一の選択肢
- ただし、このようなスキャナーは既知の**脆弱性**検出にのみ有効

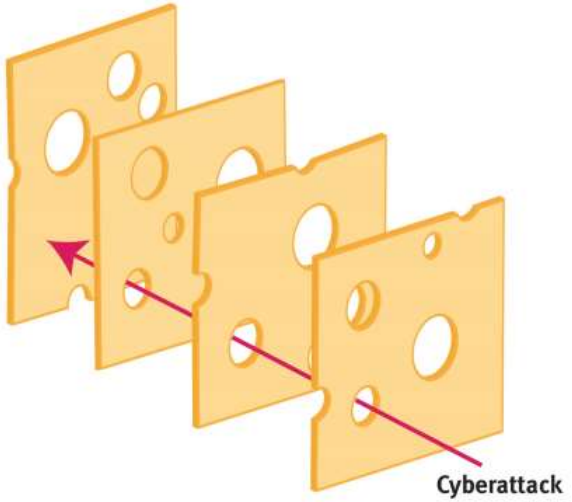
UNCOVER SOFTWARE VULNERABILITIES ACROSS YOUR COMPLEX SUPPLY CHAIN

It's challenging to understand software composition and vulnerability exposure of embedded systems—especially in industries such as automotive, medical equipment, and aerospace and defense, where you need to navigate complex supply chains and stringent regulatory requirements.

BlackBerry Jarvis scans binary images or files you upload and generates reports that include graphical views of third-party files, third-party licenses and groupings of detected vulnerabilities by severity.

directinsight.co.uk/blackberry-jarvis-static-binary-checker/

Swiss cheese…

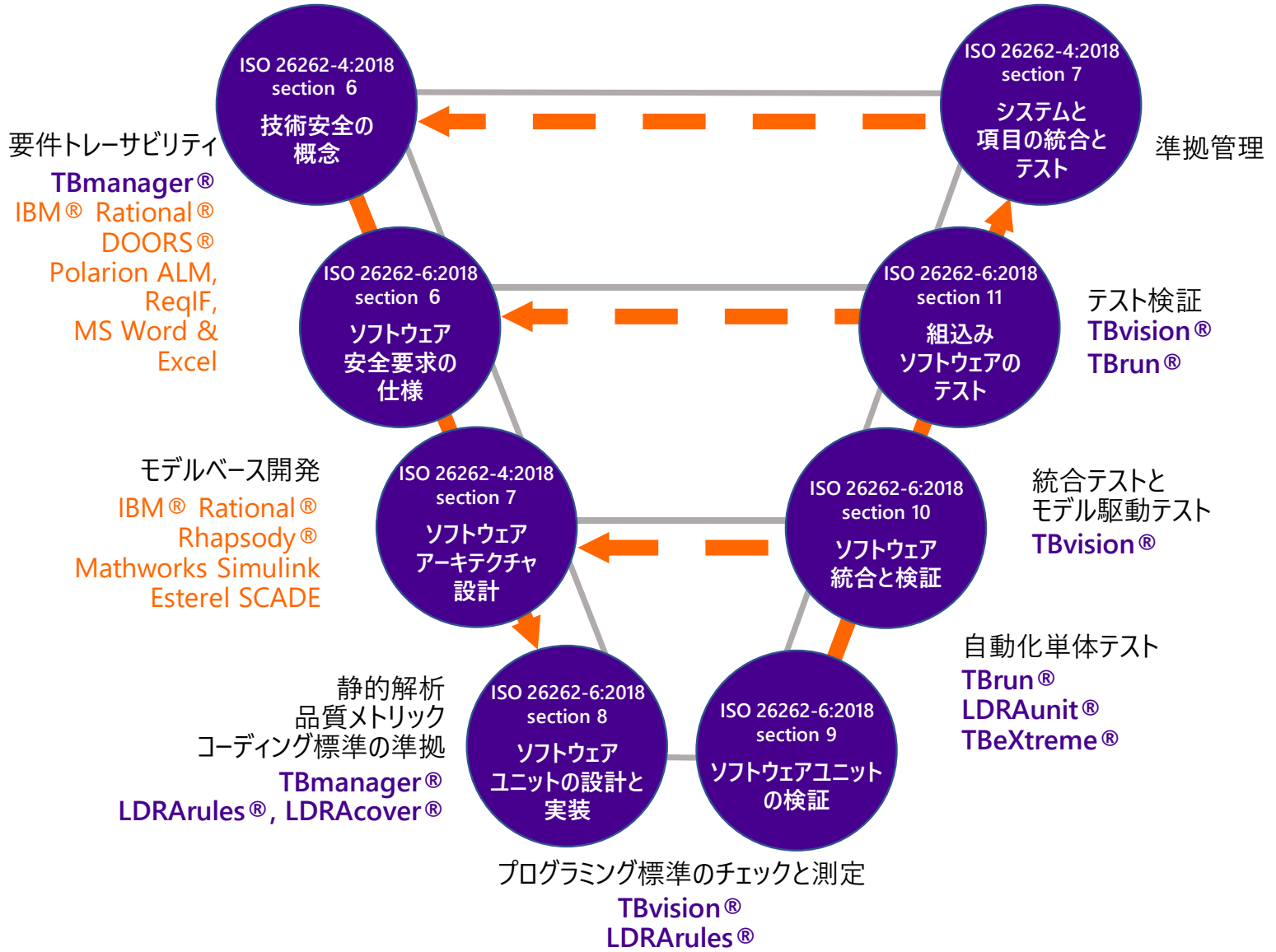


- このような脆弱性は、コード内の特定の弱い構造、または間違いの結果である可能性が高い
- 改めて、**脆弱性**は、**弱点**により招かれた症状
- CVE(Common Vulnerabilities and Exposures) と CWE(Common Weakness Enumeration) は、それぞれ**脆弱性**と**弱点**のリスト
- これらは関連して、どちらもサイバーセキュリティの分野でセキュリティ関連の問題を分類および特定するために使用されるが、目的は異なる
- LDRAの解析ツールでは、新しく実装されるコードの**弱点**を排除することを目指す。弱点がなければ、脆弱性も存在しない
- これはバイナリ脆弱性スキャナーに関連するが、異なるもので、スイスチーズモデルでいうと 2 枚の防御壁となる

Taint considerations

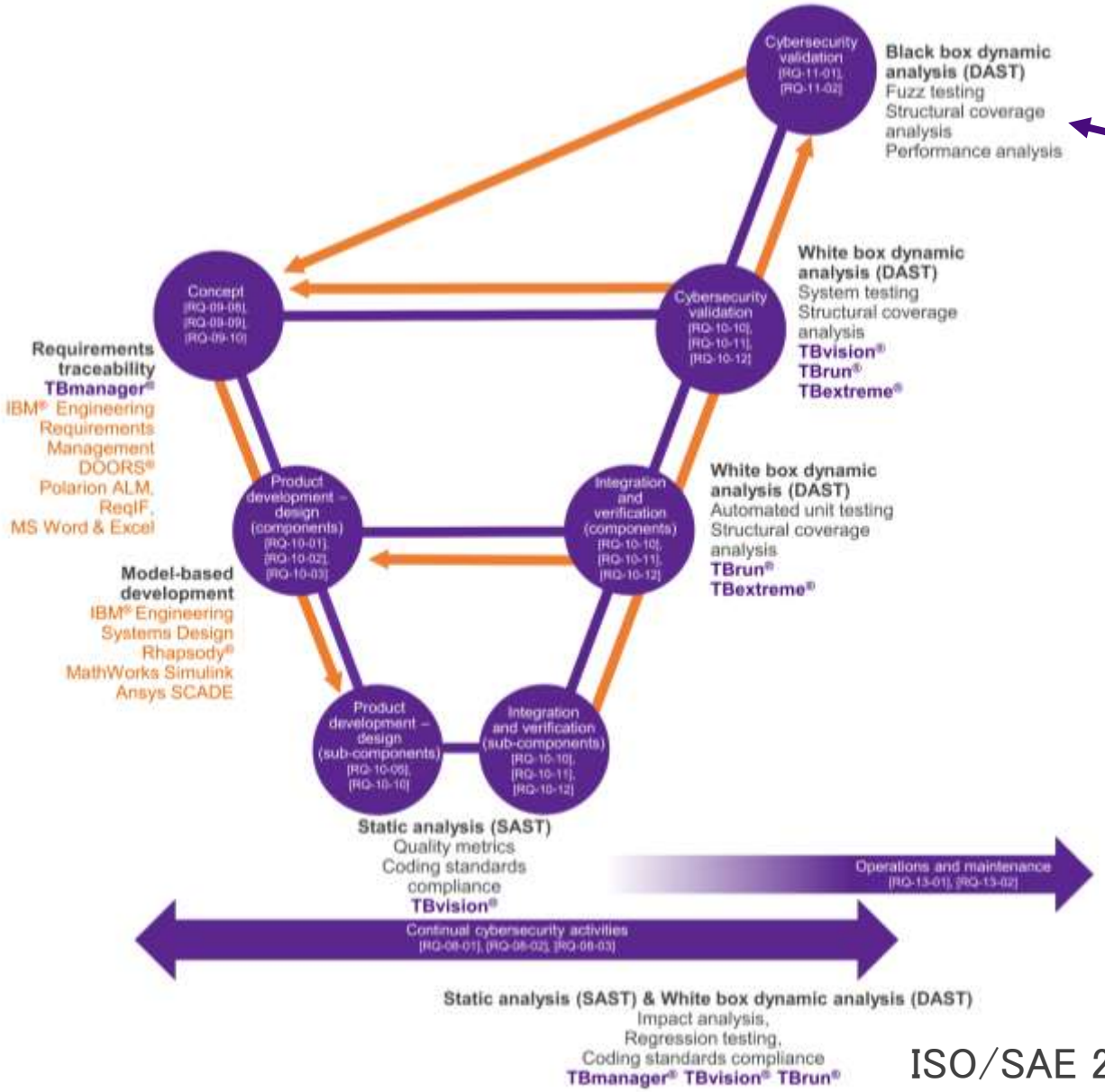
at design time

ISO 26262 に準拠したソフトウェア開発プロセス



ISO 26262 第2版ソフトウェア開発プロセスガイドラインと自動化ツールチェーンの機能の対応付け

ISO/SAE 21434 に準拠したプロセス



- ISO/SAE 21434 のセクションとの関係を示すように修正された V 字モデル
- コネクテッドシステムでの継続的なサポートの必要性が強調される
- ISO/SAE 21434 にはソフトウェア開発者向けの詳細なガイダンスが欠けるが、国際規格が与えるアドバイスは妥当

ISO/SAE 21434 のライフサイクルを修正 V 字モデルで表現

機能安全プロセスの多くがセキュアコーディングに

要件トレーサビリティ

規格が定めるオブジェクティブの達成

コーディング規約への準拠

コードの複雑性の解析

データフロー、コントロールフロー解析

構造化カバレッジ解析

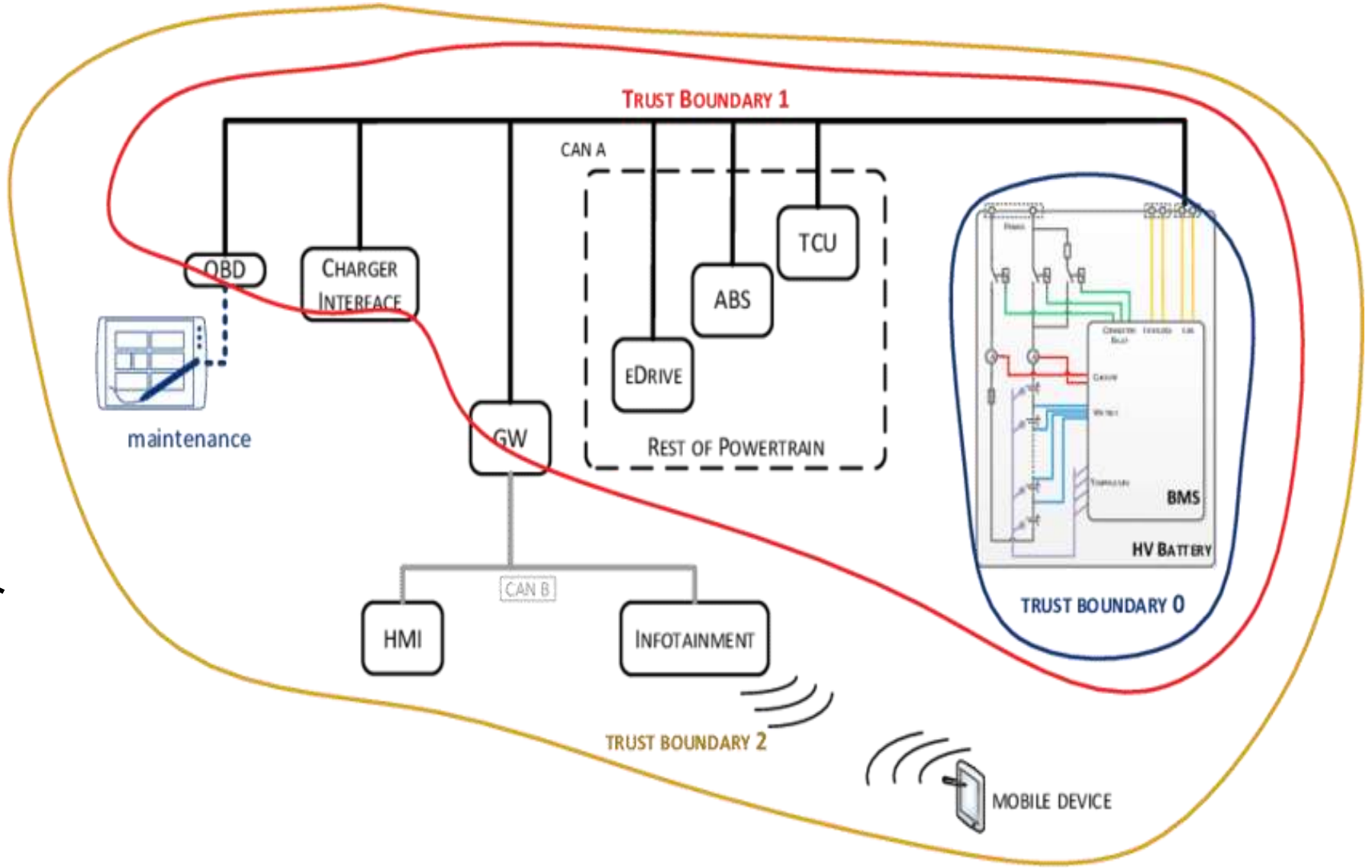
ターゲットレベルでのテスト実行

ツール認定

規格準拠の管理

ISO/SAE 21434 § 8.5 脆弱性解析

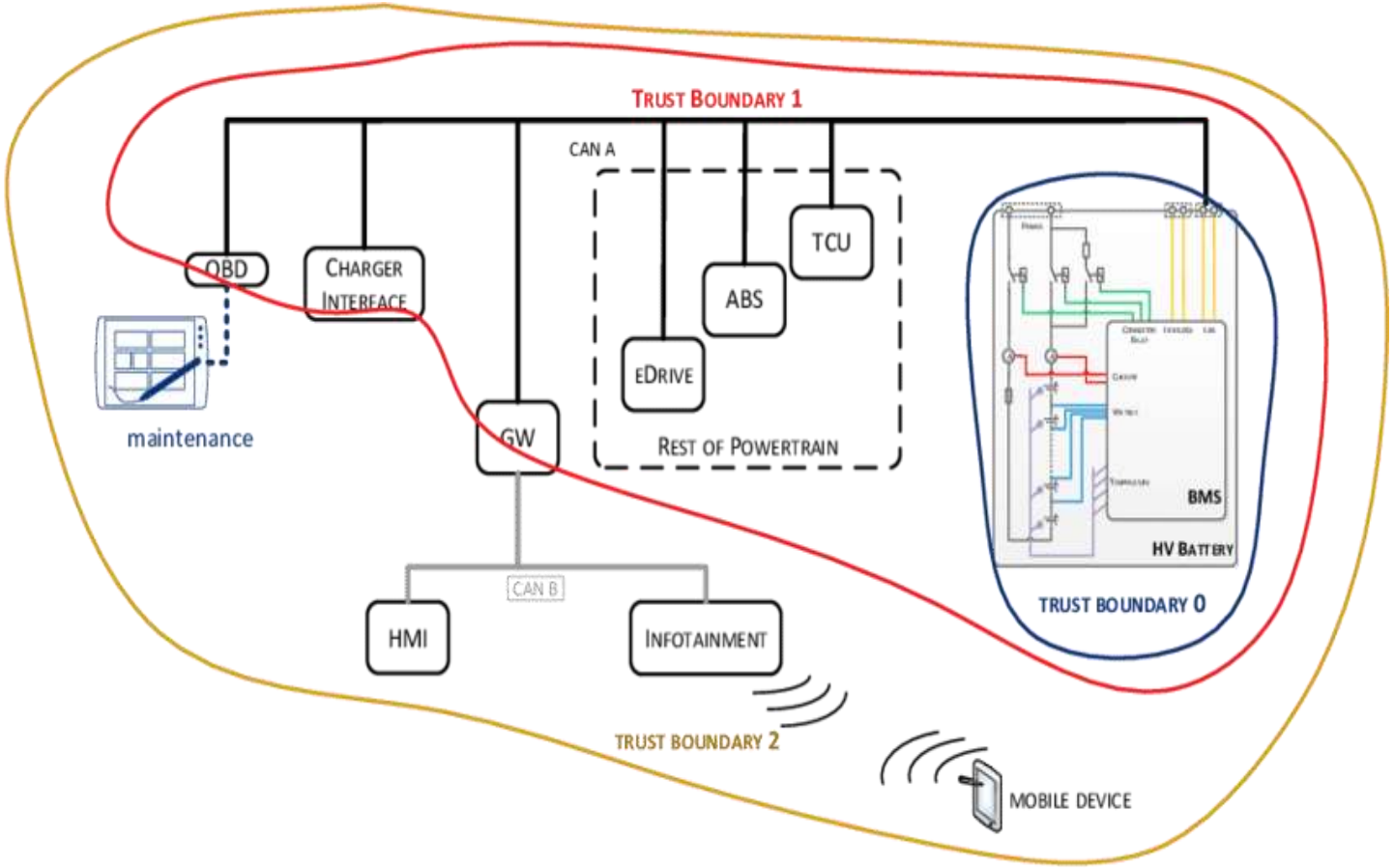
- サイバーセキュリティに機能安全プロセスを採用する際には、主要な差別化要因を見失わないことが重要
- 脆弱性解析はそのような差別化要因の１つ
- 信頼境界の特定は、ソフトウェア脆弱性解析（SVA）の一般的なアプローチ



https://www.researchgate.net/figure/Trust-boundary-Layers-of-Use-Case_fig2_317546519

設計時の信頼境界の識別

1. ソフトウェア脆弱性解析（SVA）の最初のステップは、アプリケーションを分解してデータを分析し、エントリポイントと終了ポイントを制御する
2. 信頼境界の交差を識別する
3. 「脅威モデル」で交差を文書化する
4. システムの内訳に基づいて脅威を分類する
   - Common Vulnerability Scoring System (CVSS) および/または Common Weakness Scoring System (CWSS) の使用を検討する



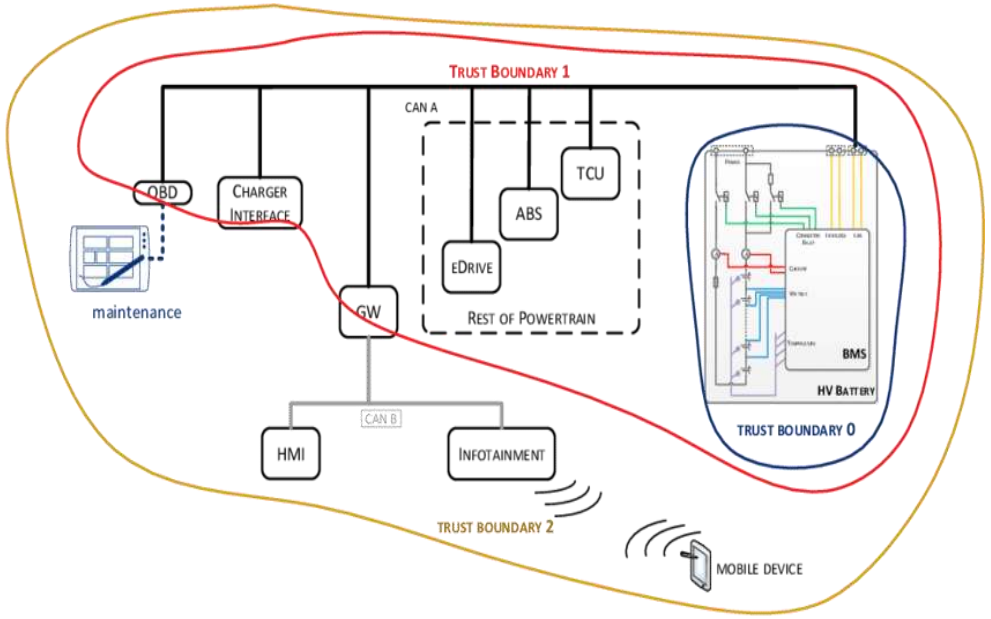
Taint considerations

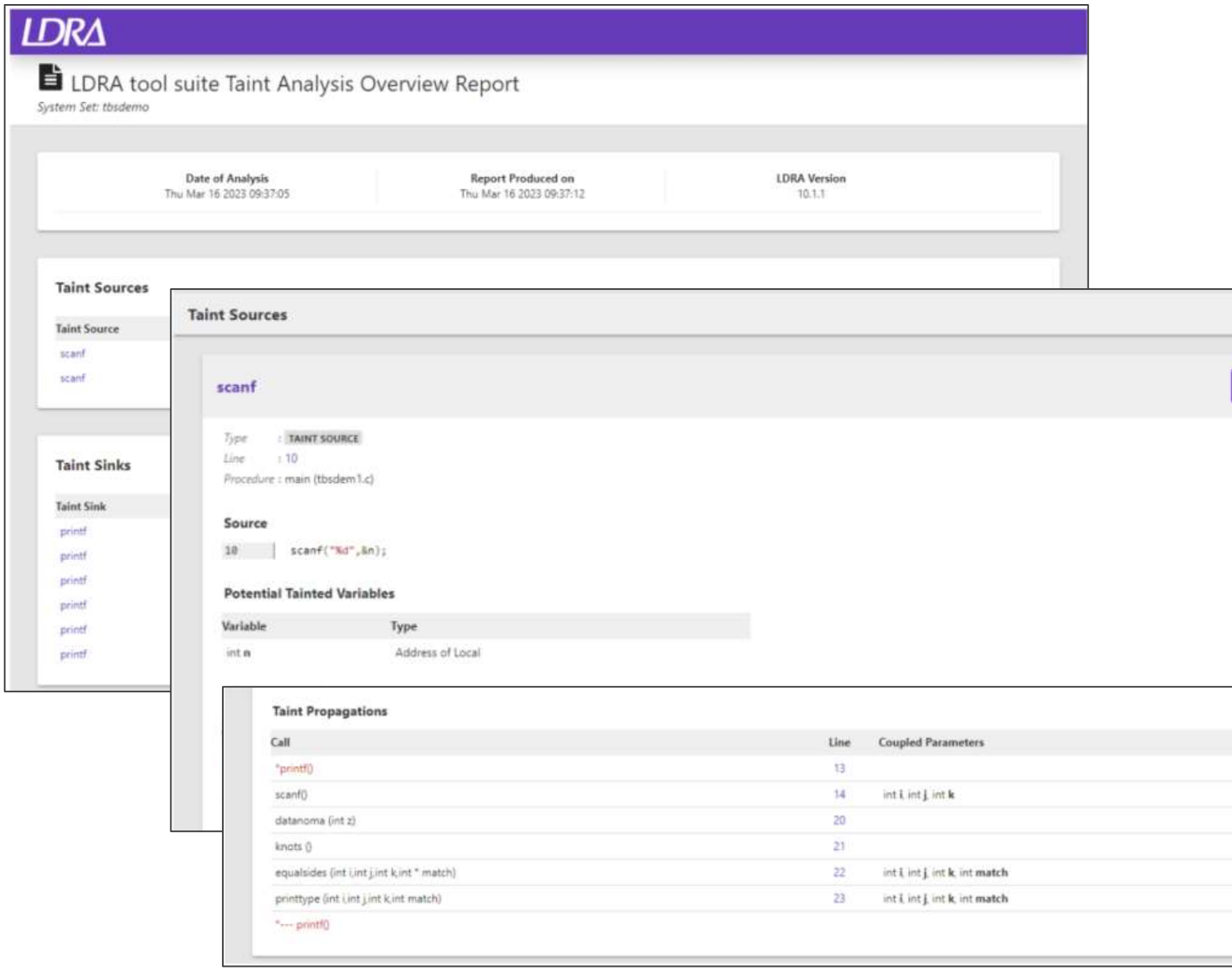
at coding time

開発およびテスト時の信頼境界

5. テイント解析を開発プロセスに即して考えると...

6. 汚染されたデータに対して適切な考慮が払われているかどうか、また信頼境界分析から得られた要件が正しく満たされているかどうかを確認するために使用できる





静的および動的攻撃ベクトル

- 静的攻撃ベクトルは、動作状態に関係なく脆弱性をターゲットにする
  - 設計または実装の欠陥から生じる
  - 検出には、脆弱性スキャンとコード解析、つまり静的解析
  - 予防には、パッチ適用、システム強化、およびセキュアコーディング規約に重点が置かれる

静的および動的攻撃ベクトル

- 動的攻撃ベクトルは、システムの動作中に発生する脆弱性を悪用する
  - SQL インジェクションやコマンド インジェクションなどの攻撃では、実行中のアプリケーションのコンテキスト内で悪意のあるコードまたはコマンドが実行される
  - DoS 攻撃では通常、システムまたはネットワークに過剰なトラフィックやリソース要求を送りつけ、その容量を圧迫して正常な動作を妨害する

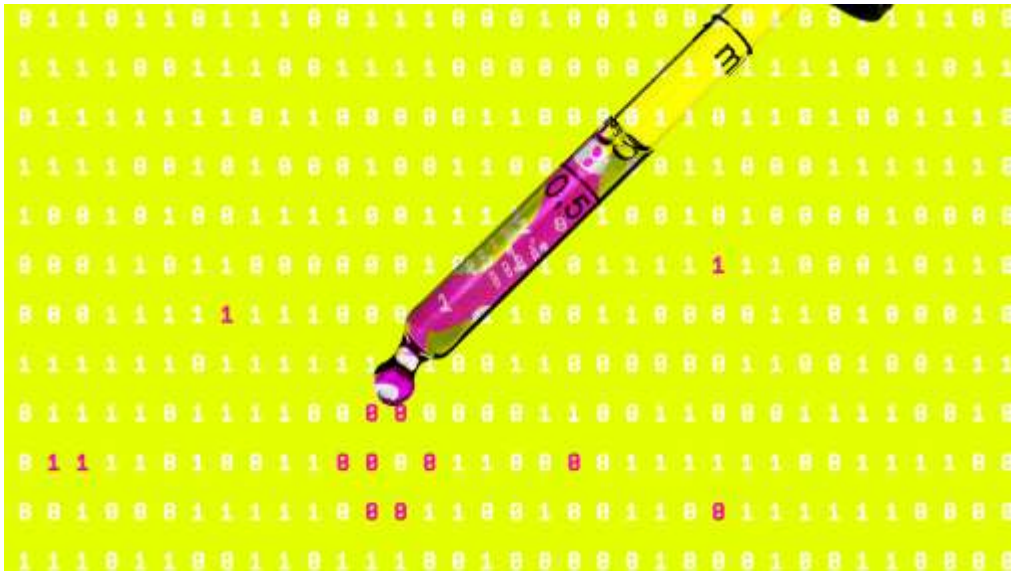
- 動的解析は、このようなアクションを模倣し、攻撃中のパフォーマンスの低下やサービスの中断を防ぐためのコードの能力を分析する手段を提供する

Test Case View

Test Case	Regression P / F	Procedure
Tc 1	PASS	addProduct
Tc 2	PASS	countProducts
Tc 3	PASS	endSession
Tc 4	PASS	generateTicket
Tc 5	PASS	identifyProduct
Tc 6	FAIL	removeLastProduct
Tc 7	PASS	startSession
Tc 8	PASS	Cashregister_barcode
Tc 9	FAIL	Cashregister_cancel
Tc 10	PASS	Cashregister_code
Tc 11	PASS	Cashregister_end
Tc 12	PASS	Cashregister_key
Tc 13	PASS	Cashregister_start
Tc 14	PASS	Productdatabase_getProduct

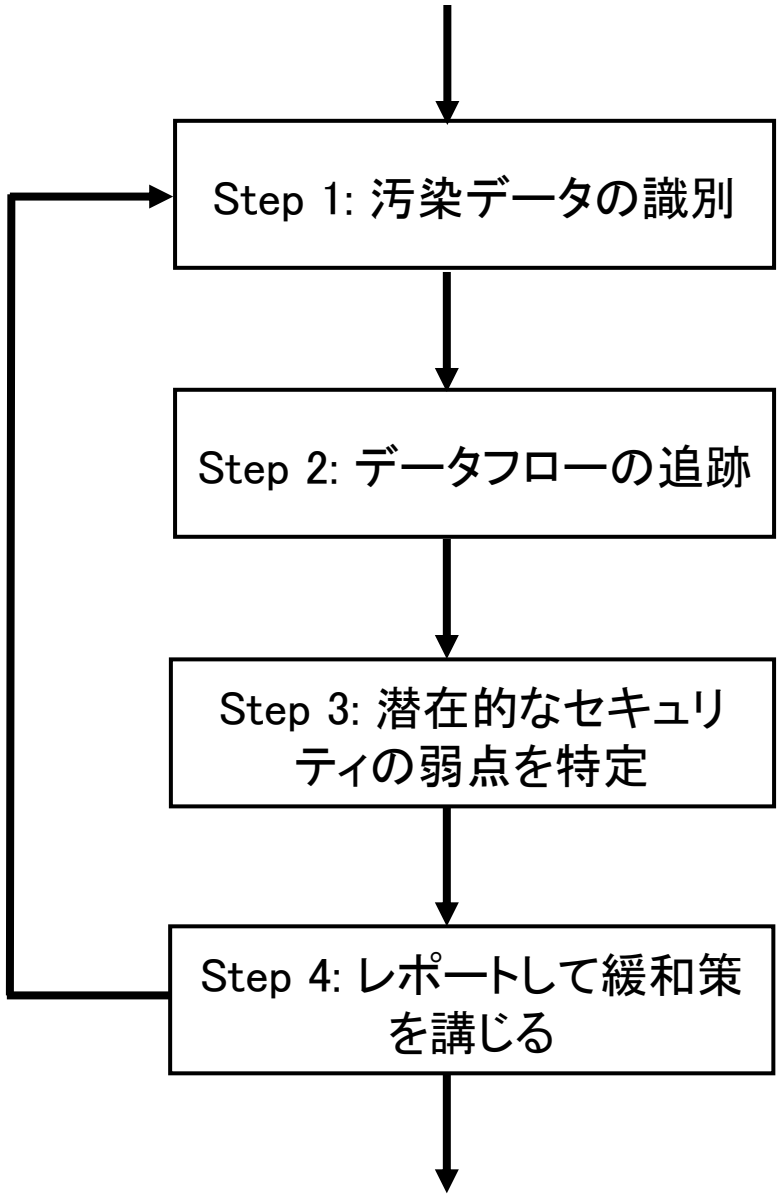
テイント解析: 攻撃ベクトルに対する防御

テイント解析は、プログラムを通過する際に潜在的に安全でない、または信頼できないデータを識別して追跡するためにソフトウェア セキュリティで使用される手法で、これには4つのステップがある



1. **汚染データの識別**: ユーザー入力、ネットワークから受信したデータ、ハイパーバイザーからの通信、または攻撃者の影響を受ける可能性のあるデータなど、信頼できないソースまたは外部ソースから生成されたデータを「汚染された」としてマークする
2. **データフローの追跡**: プログラムまたはシステム内を移動する汚染されたデータの流れを追跡する。これには、汚染されたデータがコード内でどのように操作、処理、および使用されるかを追跡することが含まれる
3. **潜在的なセキュリティの弱点を特定**: たとえば、汚染されたデータが適切なサニタイズなしでデータベースクエリで使用されると、SQL インジェクションの脆弱性につながる可能性がある。汚染されたデータがユーザーインターフェイスに漏洩すると、機密情報の漏洩につながる可能性がある
4. **レポートして緩和策を講じる**: 開発者がリスクを軽減するための適切な対策を講じることができるように、脆弱性を特定する。これには、コードの修正、入力検証の追加、データサニタイズ手順の改善などが含まれる

テイント解析プロセス



テイント解析レポート

関数コール & フロー図

コーディング規約チェック

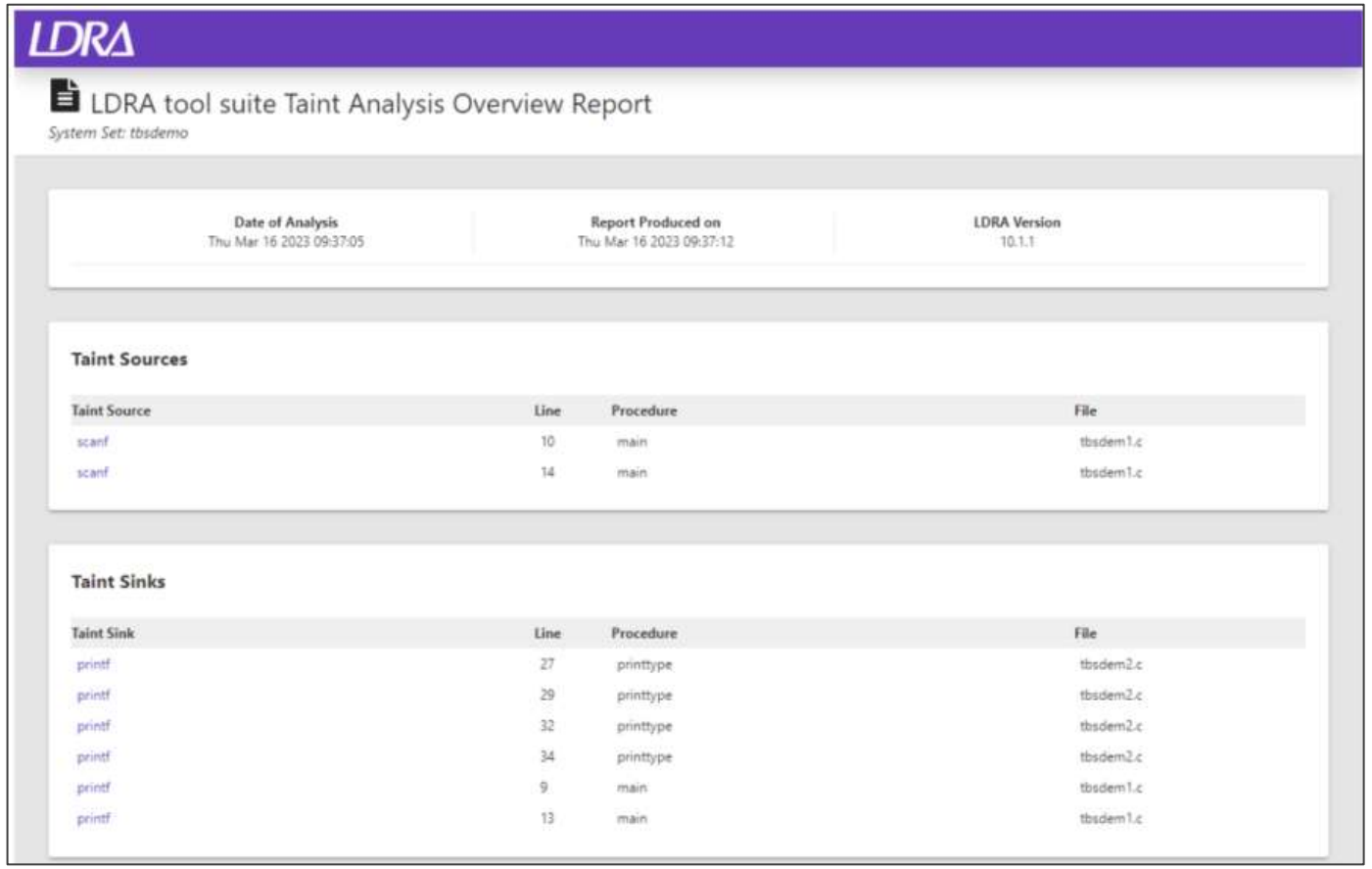
単体テストと堅牢性テスト

カバレッジ、ユニット テスト、セキュリティ レポート

Step 1: 汚染データの識別

静的解析により、以下のような、信頼できないソースや外部ソースからの、「汚染された」データを識別する手段を提供

- ユーザー入力
- ネットワークから受信したデータ
- パーティション間通信
- 攻撃者の影響を受ける可能性のあるその他のデータ



LDRA tool suite Taint Analysis Overview Report

System Set: tbsdemo

Date of Analysis	Report Produced on	LDRA Version
Thu Mar 16 2023 09:37:05	Thu Mar 16 2023 09:37:12	10.1.1

Taint Sources

Taint Source	Line	Procedure	File
scanf	10	main	tbsdem1.c
scanf	14	main	tbsdem1.c

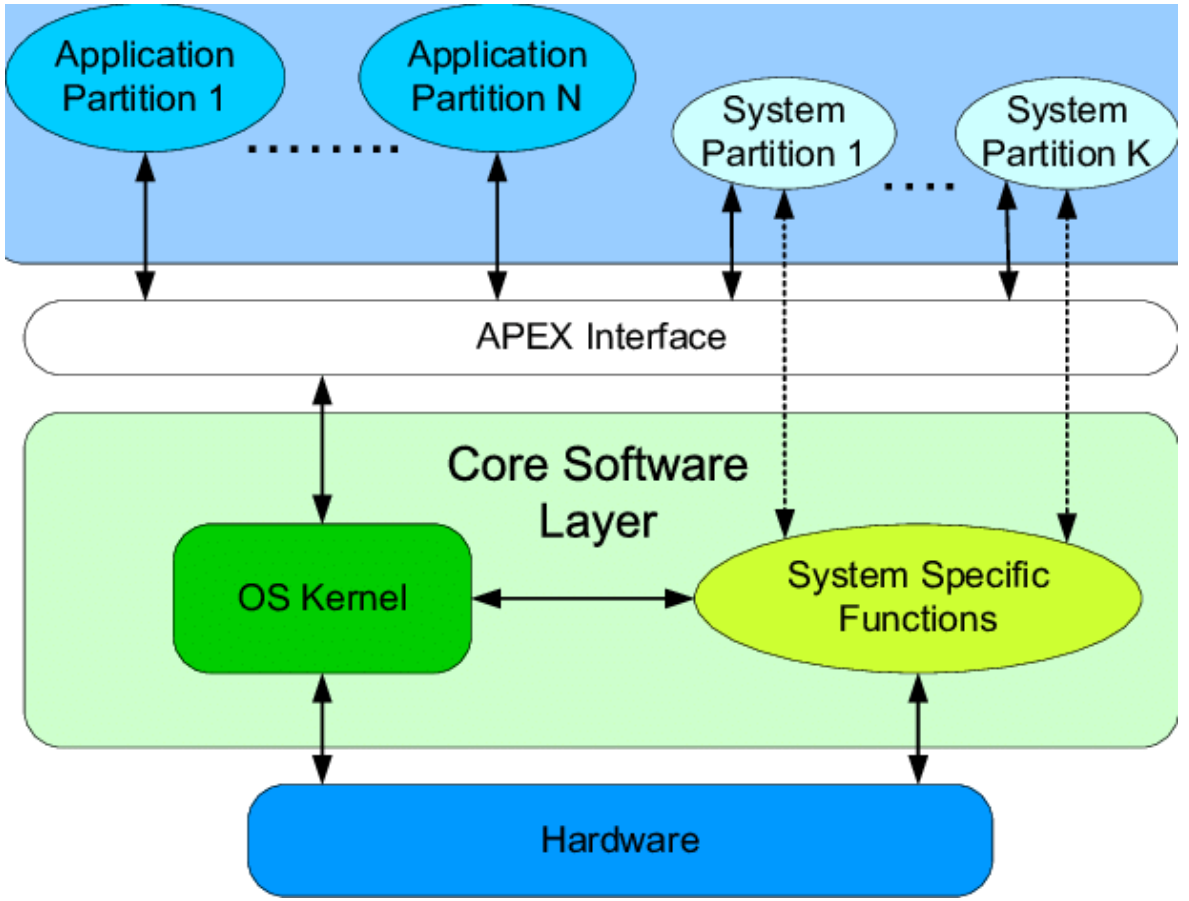
Taint Sinks

Taint Sink	Line	Procedure	File
printf	27	printtype	tbsdem2.c
printf	29	printtype	tbsdem2.c
printf	32	printtype	tbsdem2.c
printf	34	printtype	tbsdem2.c
printf	9	main	tbsdem1.c
printf	13	main	tbsdem1.c

テイント解析：データの汚染源と、そのデータの伝搬先をレポート

Step 1: 汚染データの識別 ARINC 653 での例

- ARINC 653 (Avionics Application Standard Software Interface) は、セーフティクリティカルな分割された航空電子機器の分割システムの標準
- これは、同じコンピュータ ボード上で実行されている ARINC パーティション間でメッセージを交換するために使用できるパーティション間通信 (IPC) メカニズムを定義する
- このようなパーティション間通信の関数呼び出しによってパーティションに入るすべてのデータは、汚染されていると見なす必要がある



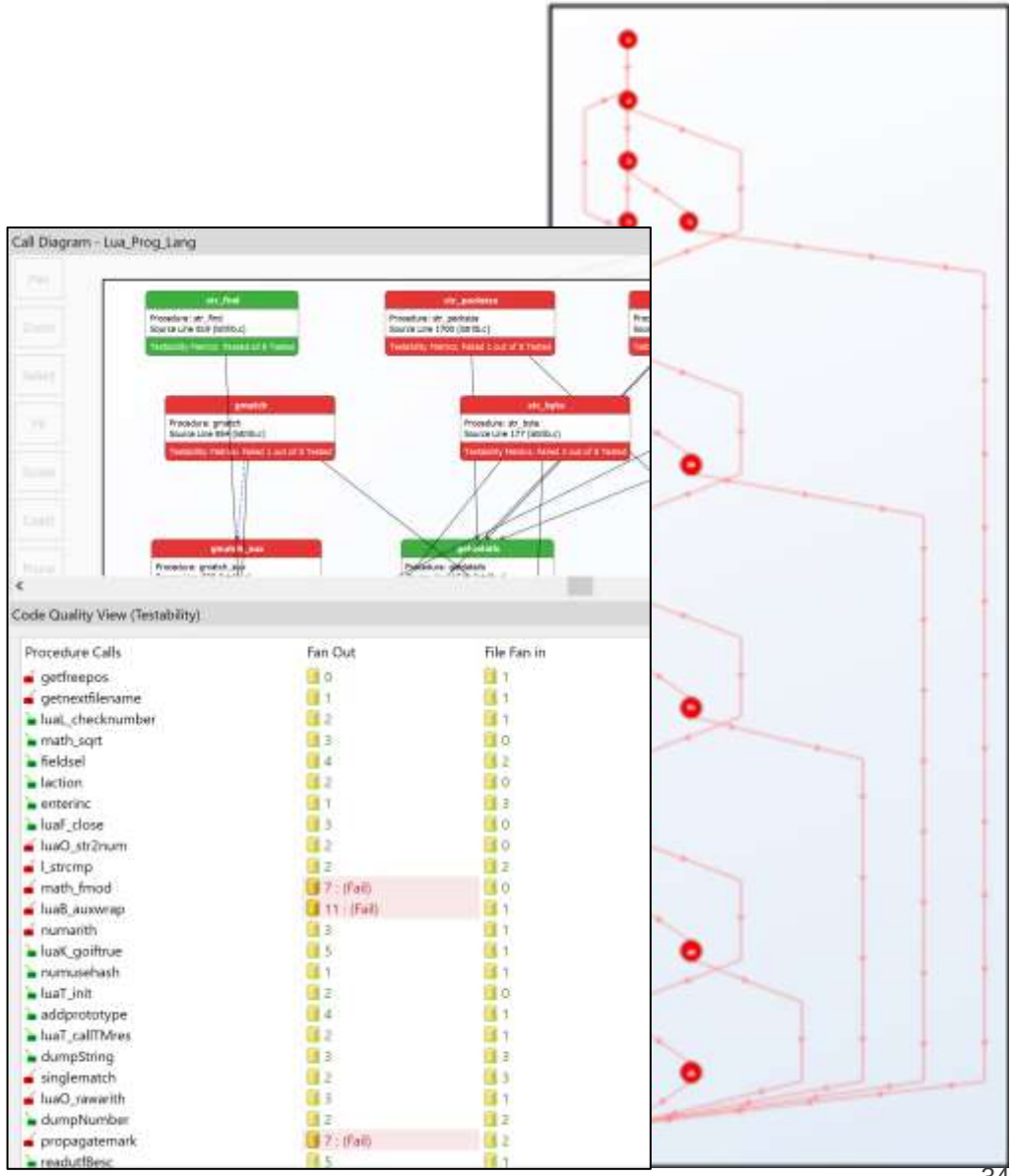
- ***CREATE_SAMPLING_PORT(PortID, MaxMessageSize, MaxMessages);***
- ***SEND_SAMPLING_MESSAGE(PortID, MessagePtr, MessageSize);***
- ***RECEIVE_SAMPLING_MESSAGE(PortID, MessagePtr, MessageSizePtr);***

- ***CREATE_QUEUING_PORT(PortID, MaxMessageSize, MaxMessages, MaxMessageQueueSize);***
- ***SEND_QUEUING_MESSAGE(PortID, MessagePtr, MessageSize);***
- ***RECEIVE_QUEUING_MESSAGE(PortID, MessagePtr, MessageSizePtr);***

Step 2: データフローの追跡

テイント解析レポート

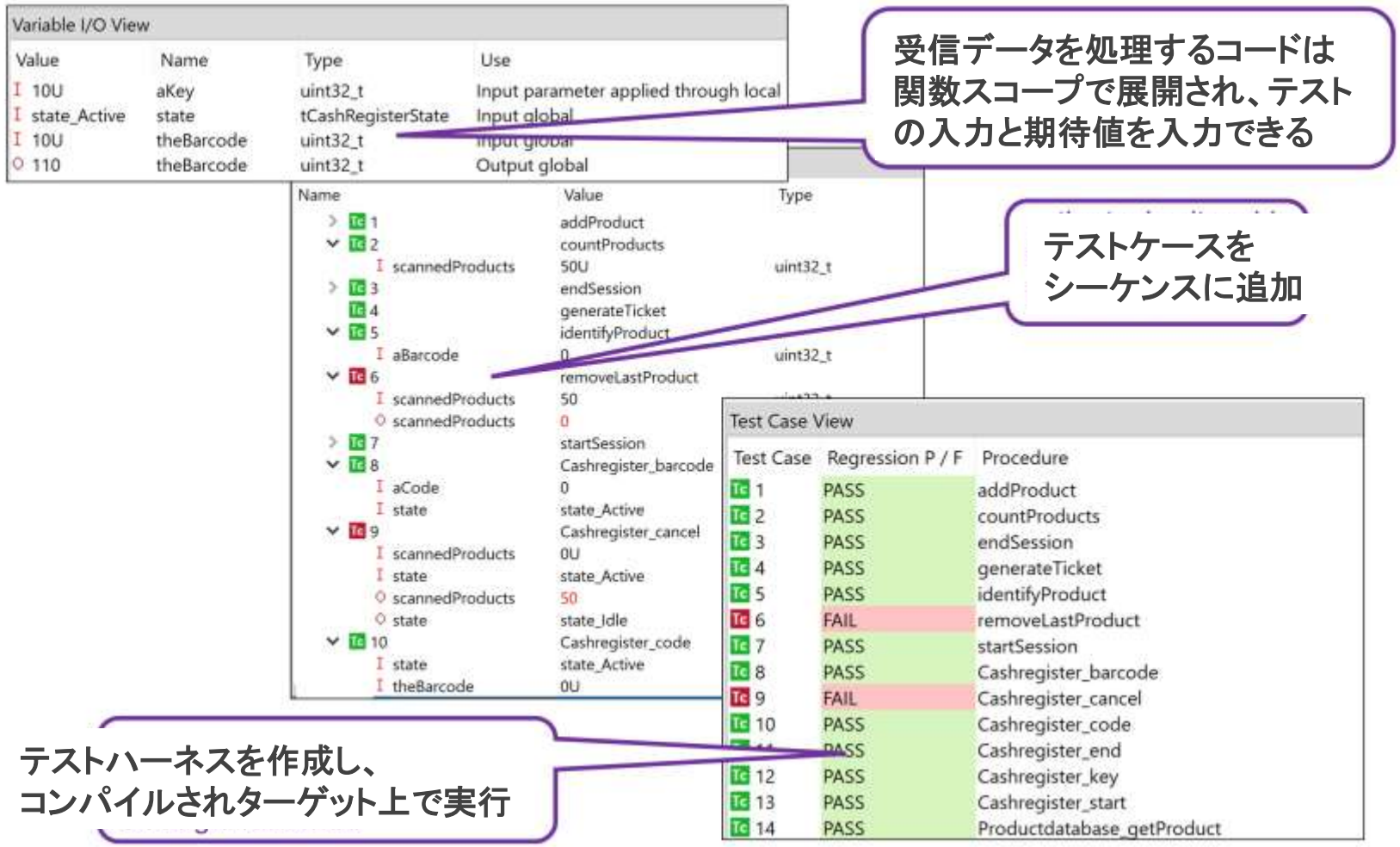
- 汚染されたデータが
  どこから入り、
  どこに伝播するかを
  レポートされている
- 関数の呼び出しとフロー
  の図は、その伝播の範囲
  を静的に示すために使用
  される



Step 2: データフローの追跡

静的解析のみに基づいてコードが何を実行するかを調査するだけでなく、それ以上の調査を行うことが重要

- 一部の静的解析ツールは、コードが実際に何を行うかを予測するが、あくまでも推定と近似値にすぎない
- 汚染源から始まる単体テストは、汚染された不正なデータがどのように処理されるかを示し、サポートメカニズムが機能し効果的であることの確認を提供する

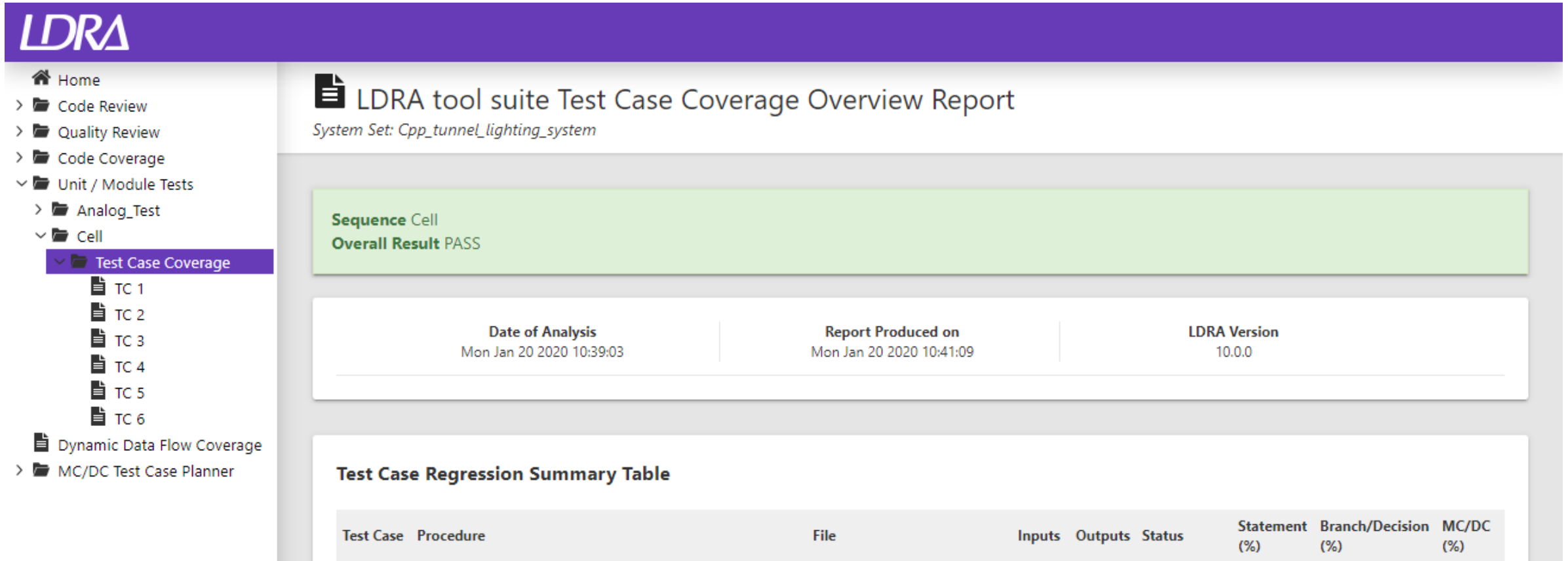


Step 3: 潜在的なセキュリティの弱点を特定

- 動的解析はここでも同様に重要で、
- 静的解析を通じてそれを実行する方法がわかる
  - セキュアコーディング規約だけを適用することも選択肢ではあるが、、

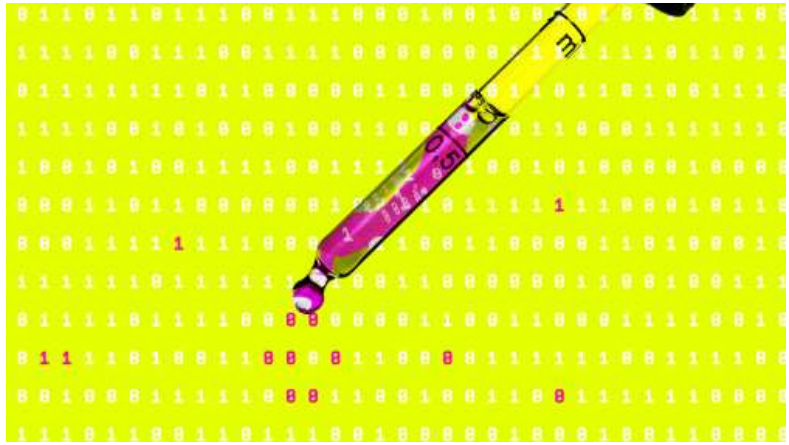
Step 4: レポートして緩和策を講じる

- もしデータがうまく処理されていない場合は、対処して再テストする
- すべてがデータを適切に処理していることを確認できれば、それを示す適切なエビデンスとして動的テストのカバレッジ解析結果を用いる



Summary

- 脆弱性分析は、機能安全とサイバーセキュリティ開発との重要な差別化要因
- 信頼境界分析は設計による緩和に役立つ
- テイント解析は、講じられた予防措置が十分であるかどうかをテストする
- 設定可能な汚染レポートは、環境（RTOS、ハイパーバイザー等）に応じて調整できる
- 数十年にわたってセーフティクリティカルなアプリケーションで実証されてきた技術を活用できることは朗報！



Need more information?

LDRA.com

LDRA Software Technology

LDRA Limited

@ldra_technology

LDRA Tools

info@ldra.com

FUJI SETSUBI

LDRA　スタンダード認証支援テストツール

https://www.fuji-setsu.co.jp/products/LDRA/

富士設備工業(株)電子機器事業部
https://www.fuji-setsu.co.jp