MATLAB EXPO
JAPAN
2020年9月28日—10月2日 | オンライン

JPCERT CC®

セキュアコーディングを実践する
今よりセキュアなソフトウェアを実現するために

2020年10月1日(木)

JPCERTコーディネーションセンター
早期警戒グループ
戸田 洋三

本日の話題

- ✓自己紹介
- ✓解析ツールを活用しよう
- ✓解析ツールを盲信するな
- ✓解析ツールのクセを把握して上手に使いこなそう
- ✓まとめ

自己紹介

JPCERT/CC
早期警戒グループ
リードアナリスト　戸田 洋三

http://www.tomo.gr.jp/root/e9706.html

脆弱性情報分析, セキュアコーディング普及啓発活動……
に努めています

JPCERT/CC の活動

インシデント予防 ｜ インシデントの予測と捕捉 ｜ 発生したインシデントへの対応

脆弱性情報ハンドリング

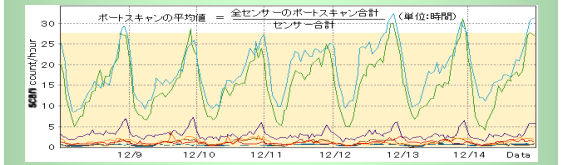
- 未公開の脆弱性関連情報を製品開発者へ提供し、対応依頼
- 関係機関と連携し、国際的に情報公開日を調整
- セキュアなコーディング手法の普及
- 制御システムに関する脆弱性関連情報の適切

JVN Japan Vulnerability Notes

情報収集・分析・発信

定点観測（TSUBAME）

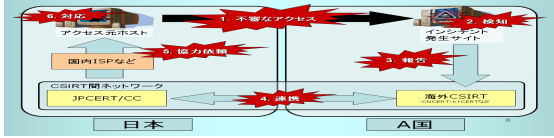
- ネットワークトラフィック情報の収集分析
- セキュリティ上の脅威情報の収集、分析、必要とする組織への提供



インシデントハンドリング

（インシデント対応調整支援）

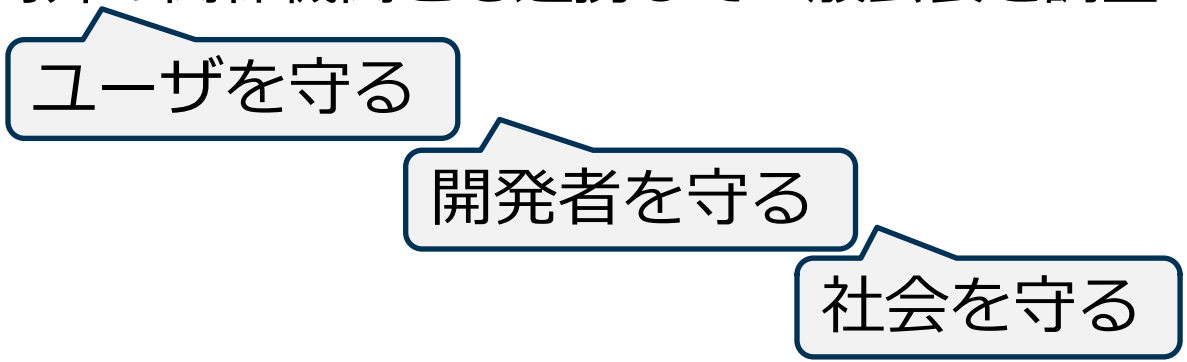
- マルウエアの接続先等の攻撃関連サイト等の閉鎖等による被害最小化
- 攻撃手法の分析支援による被害可能性の確認、拡散抑止
- 再発防止に向けた関係各関の情報交換及び情報共有



早期警戒情報	重要インフラ、重要情報インフラ事業者等の特定組織向け情報発信
CSIRT構築支援	海外のNational-CSIRTや企業内のセキュリティ対応組織の構築・運用支援
制御システムセキュリティ	制御システムに関するインシデントハンドリング、情報収集・分析発信
アーティファクト分析	マルウエア（不正プログラム）等の攻撃手法の分析、解析
国内外関係者との連携	日本シーサート協議会、フィッシング対策協議会の事務局運営等
国際連携	各種業務を円滑に行うための海外関係機関との連携

脆弱性情報ハンドリングとは？

- 脆弱性情報ハンドリング
  - ソフトウェア/ハードウエアシステム等における脆弱性情報の取扱い
  - 未公開の脆弱性情報を製品開発者に連絡し対応を依頼
  - 海外の関係機関とも連携して一般公表を調整



日本における脆弱性情報流通の取り組み

経済産業省告示

ソフトウエア等脆弱性関連情報取扱規程

受付機関: IPA
IPA セキュリティセンター

調整機関: JPCERT/CC
JPCERT/CC 早期警戒グループ

https://www.meti.go.jp/policy/netsecurity/

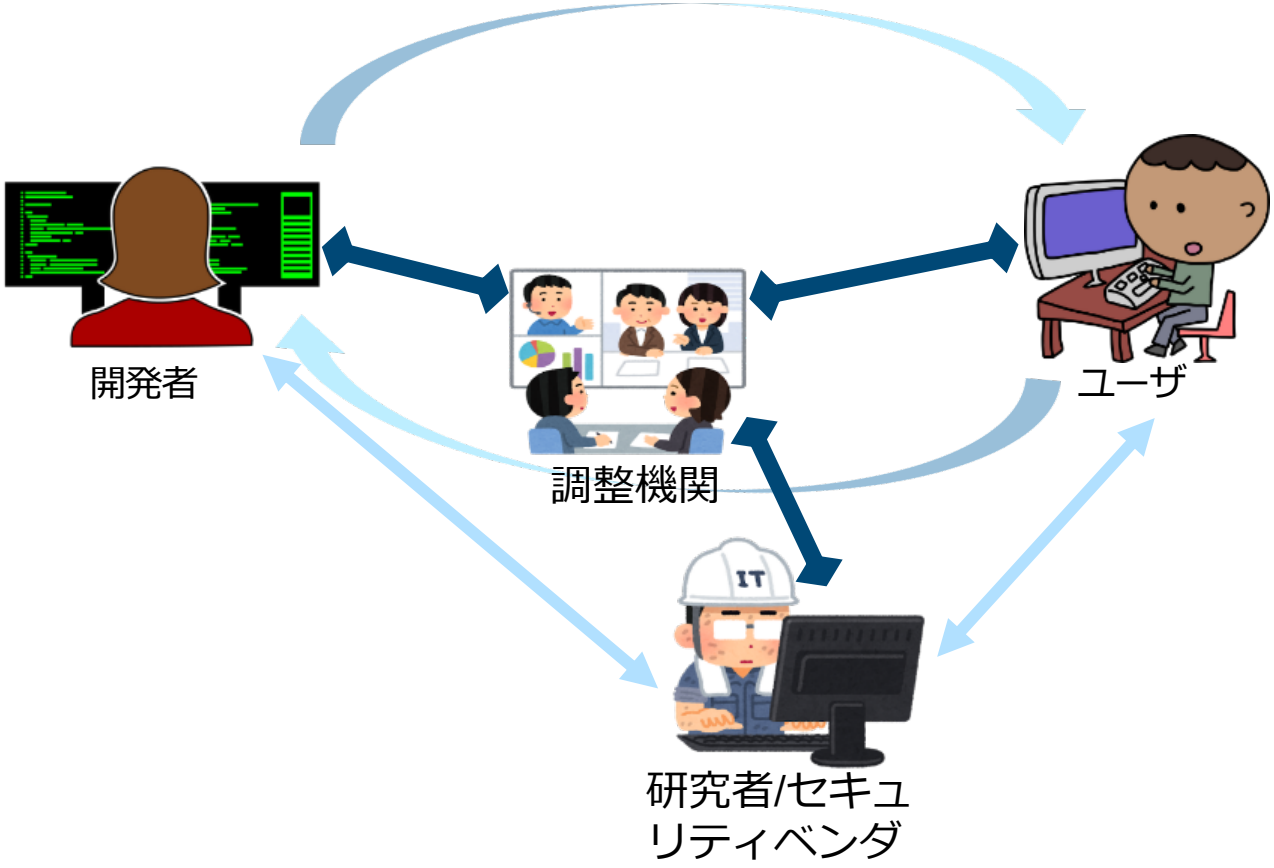
情報セキュリティ早期警戒パートナーシップ

情報セキュリティ早期警戒パートナーシップガイドライン

https://www.jpcert.or.jp/press/vh/

https://www.jpcert.or.jp/press/2004/0708.txt

脆弱性対応に関わるステークホルダ



JVN.JP: Japan Vulnerability Notes

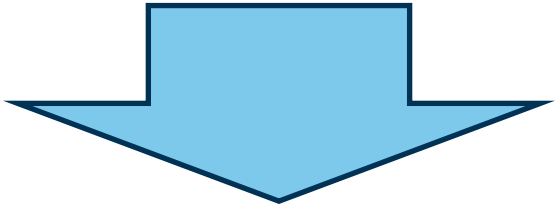
https://jvn.jp/

http://jvndb.jvn.jp/index.html

脆弱性対応からセキュアコーディングへ

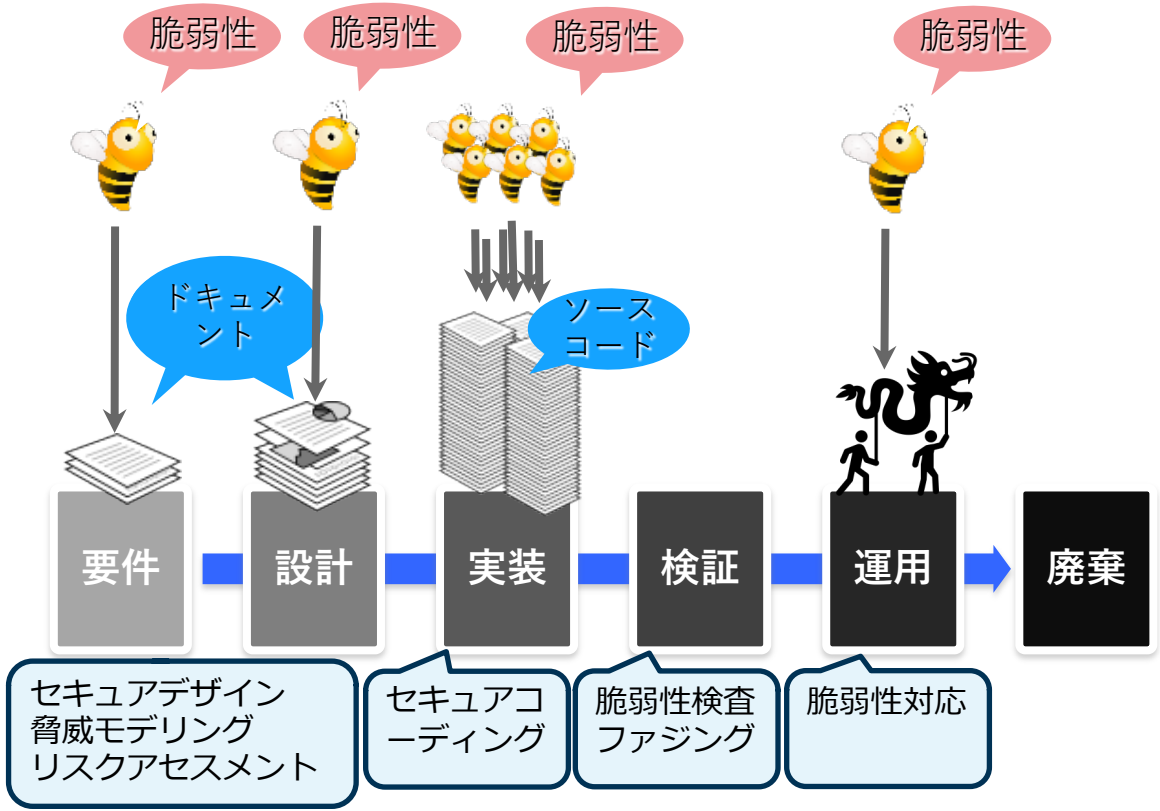
ここまでは製品リリース後に脆弱性が発見された
場合の話 (事後対応)

そもそも脆弱性を作り込まないでほしい…

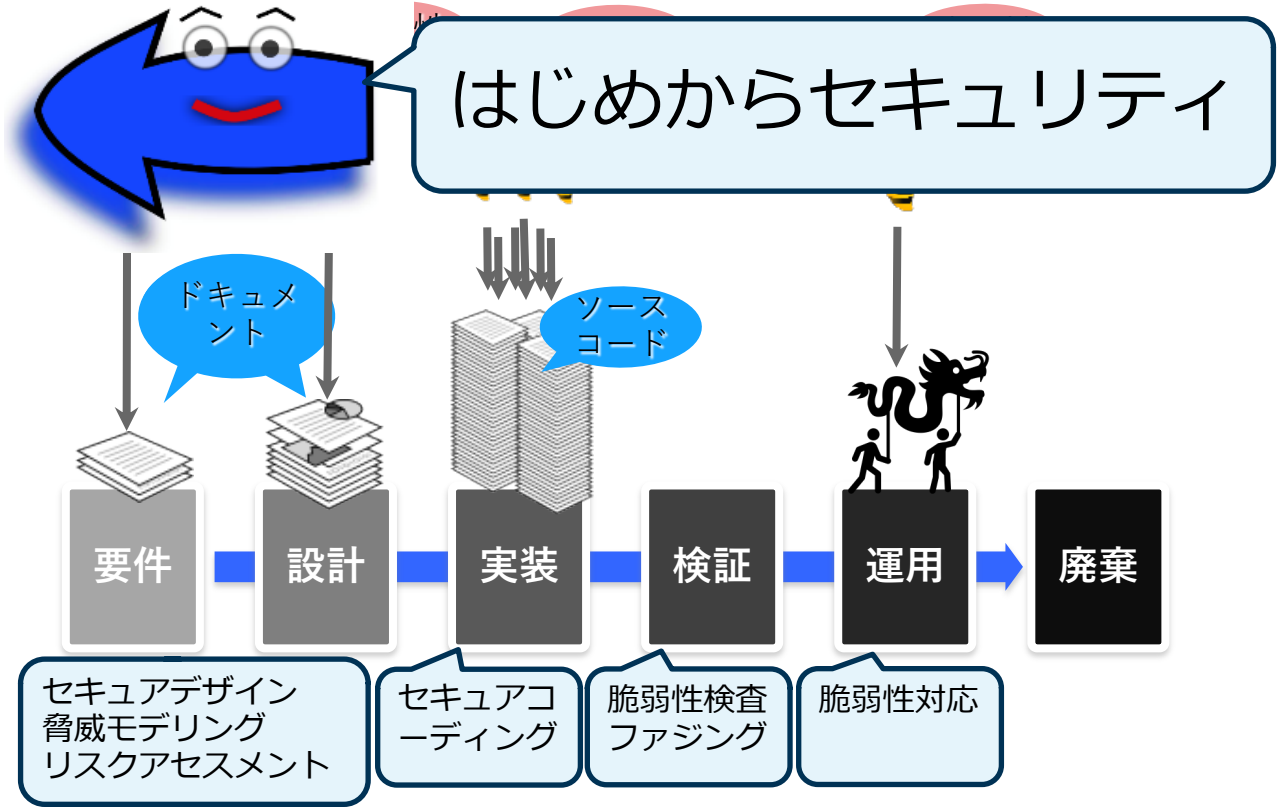


セキュアコーディング

製品開発の初期段階から脆弱性対応を考慮すべき



製品開発の初期段階から脆弱性対応を考慮すべき



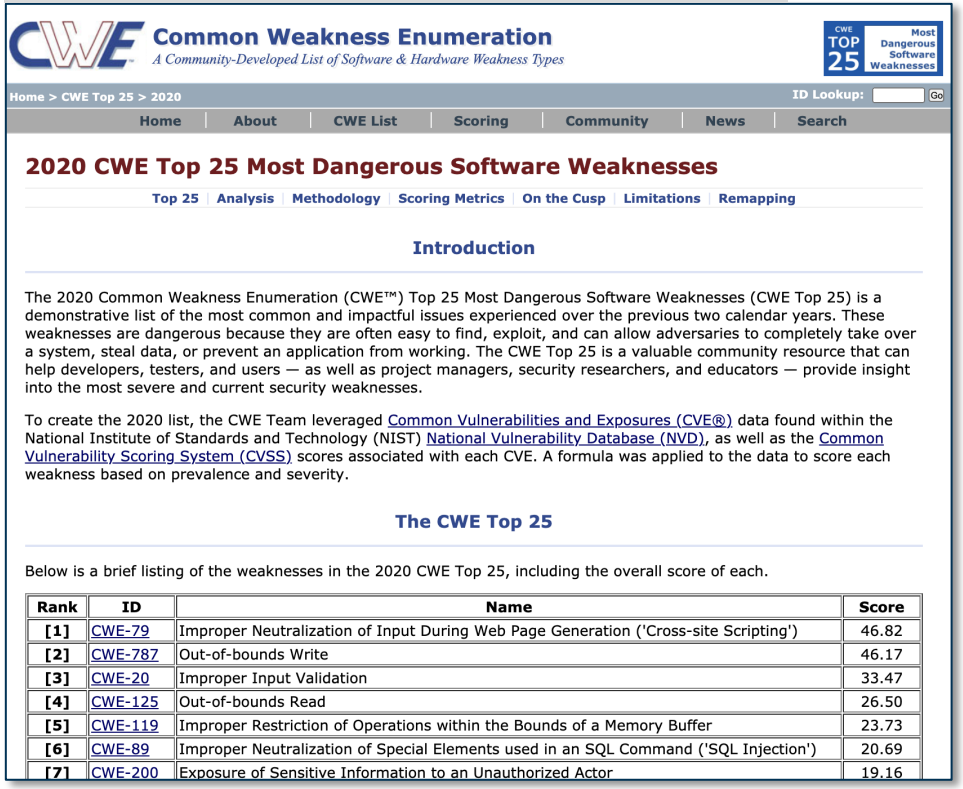
セキュアコーディングの理解と実践

理解する
- 脆弱性対応の重要性
- 脆弱性の原因となるコーディングパターンを知る

実践する
- 問題となるコーディングパターンを避け、セキュアなコードを書く
- コードがセキュアであることを確認する

セキュアコーディングを理解する

https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html



Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

CWE TOP 25 Most Dangerous Software Weaknesses

Home > CWE Top 25 > 2020 ID Lookup: Go

Home | About | CWE List | Scoring | Community | News | Search

2020 CWE Top 25 Most Dangerous Software Weaknesses

Top 25 | Analysis | Methodology | Scoring Metrics | On the Cusp | Limitations | Remapping

Introduction

The 2020 Common Weakness Enumeration (CWE™) Top 25 Most Dangerous Software Weaknesses (CWE Top 25) is a demonstrative list of the most common and impactful issues experienced over the previous two calendar years. These weaknesses are dangerous because they are often easy to find, exploit, and can allow adversaries to completely take over a system, steal data, or prevent an application from working. The CWE Top 25 is a valuable community resource that can help developers, testers, and users — as well as project managers, security researchers, and educators — provide insight into the most severe and current security weaknesses.

To create the 2020 list, the CWE Team leveraged Common Vulnerabilities and Exposures (CVE®) data found within the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), as well as the Common Vulnerability Scoring System (CVSS) scores associated with each CVE. A formula was applied to the data to score each weakness based on prevalence and severity.

The CWE Top 25

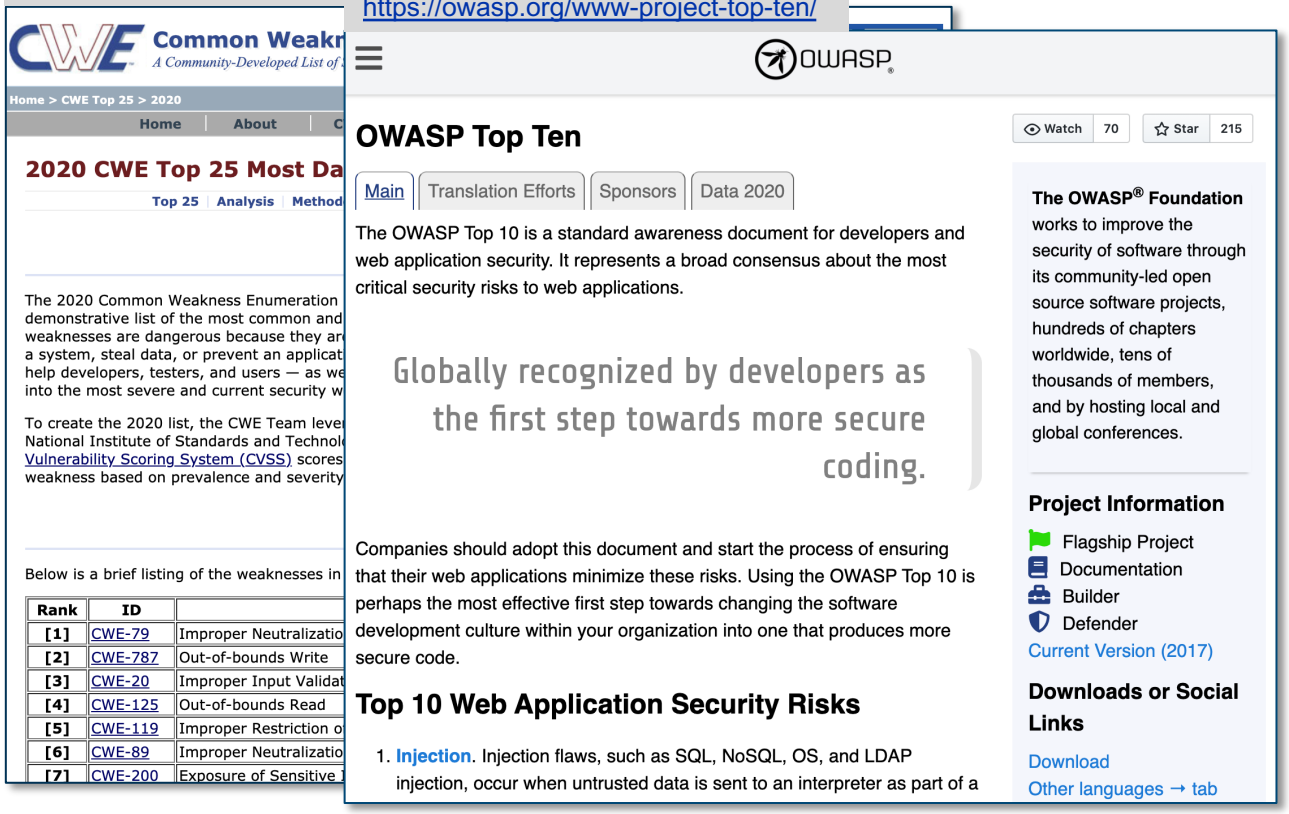
Below is a brief listing of the weaknesses in the 2020 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16

セキュアコーディングを理解する

https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

https://owasp.org/www-project-top-ten/

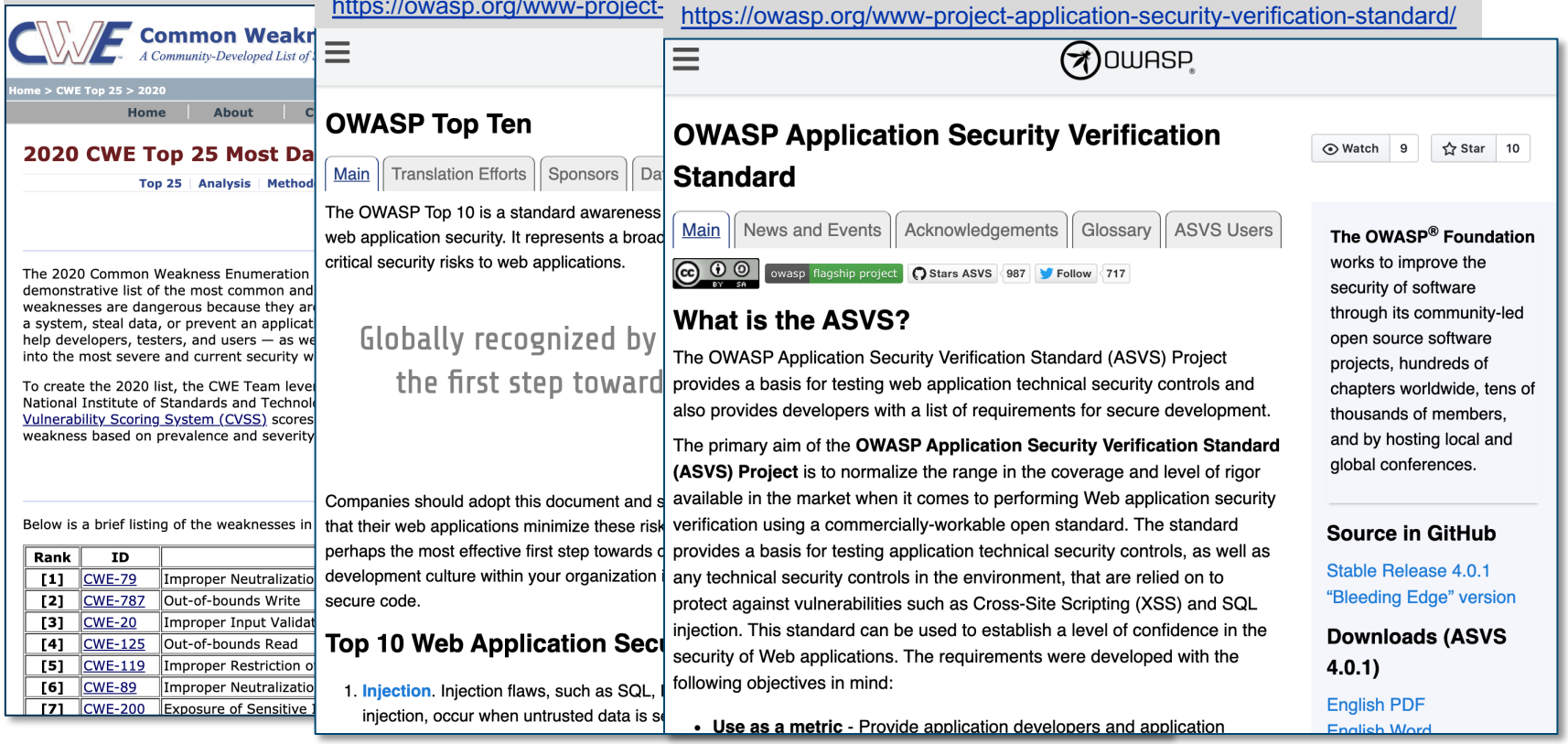


セキュアコーディングを理解する

https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

https://owasp.org/www-project-

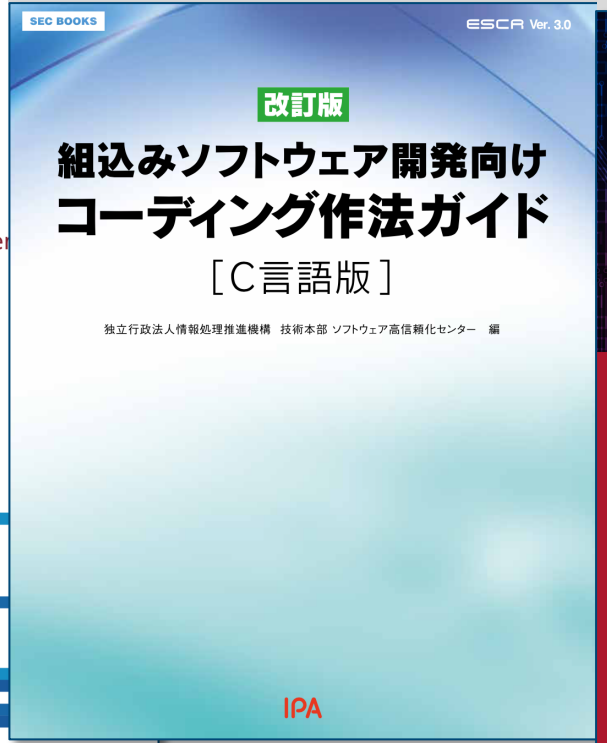
https://owasp.org/www-project-application-security-verification-standard/



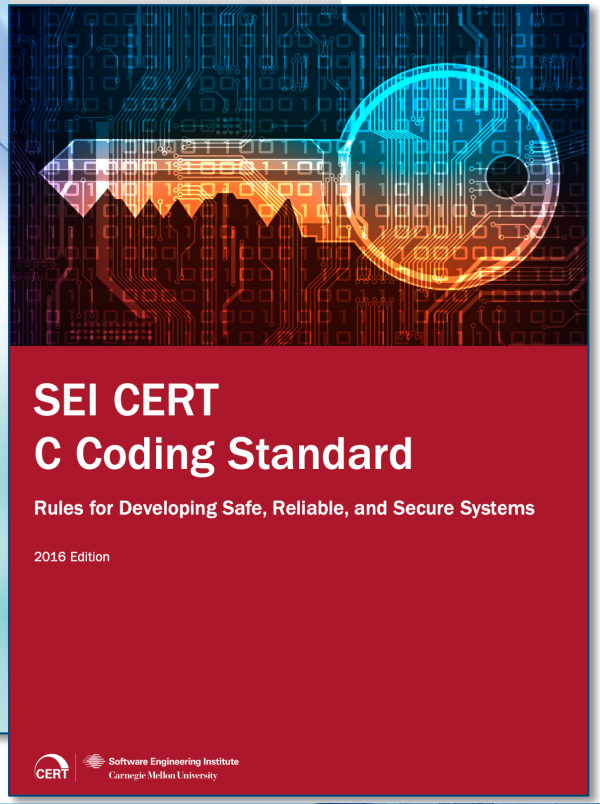
コーディングスタンダード

https://www.misra.org.uk/

https://www.ipa.go.jp/sec/publish/tn18-004.html



https://wiki.sei.cmu.edu/



セキュアコーディングの理解と実践

理解する
- 脆弱性対応の重要性
- 脆弱性の原因となるコーディングパターンを知る

実践する
- 問題となるコーディングパターンを避け、セキュアなコードを書く
- コードがセキュアであることを確認する

コンパイラやコード解析ツールを活用しよう!

コンパイラの解析機能が活躍した最近の例…

OpenSSL Security Advisory [21 April 2020]
(https://www.openssl.org/news/secadv/20200421.txt)

Segmentation fault in SSL_check_chain (CVE-2020-1967)
==========
Severity: High
……..
This issue was found by Bernd Edlinger and reported to OpenSSL on 7th April 2020.
It was found using the new static analysis pass being implemented in GCC, -fanalyzer.
……..

参考: Static analysis in GCC 10
(https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10/)

イマドキのコンパイラの解析機能を活用しよう

コンパイラが出力するエラーや警告の内容を積極的に利用する。

コンパイルが通るからといって、警告を無視せず、適切な対応を行う心がけが大切。

イマドキのコンパイラの解析機能を活用しよう

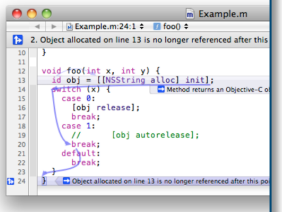
https://clang-analyzer.llvm.org/

Clang Static Analyzer

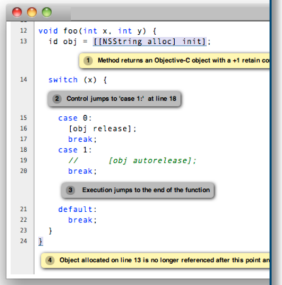
The Clang Static Analyzer is a source code analysis tool that finds bugs in C, C++, and Objective-C programs.

Currently it can be run either from the command line or if you use macOS then within Xcode. When invoked from the command line, it is intended to be run in tandem with a build of a codebase.

The analyzer is 100% open source and is part of the Clang project. Like the rest of Clang, the analyzer is implemented as a C++ library that can be used by other tools and applications.



Viewing static analyzer rest



Viewing static analyzer results i

Download

Mac OS X

- Latest build (10.8+):
- Release notes
- This build can be used both from the command line and from within Xcode
- Installation and usage

Other Platforms

For other platforms, please follow the instructions for building the analyzer from source code.

What is Static Analysis?

The term "static analysis" is conflated, but here we use it to mean a collection of algorithms and techniques use order to automatically find bugs. The idea is similar in spirit to compiler warnings (which can be useful for findi take that idea a step further and find bugs that are traditionally found using run-time debugging techniques su

https://gcc.gnu.org/wiki/DavidMalcolm/StaticAnalyzer

Static Analyzer project

Status

Initial implementation is on gcc "master" branch for GCC 10.

Only C is supported for GCC 10 (I hope to eventually support C++, but it is out-of-scope for this release)

Internal documentation: prebuilt HTML

Git branch with some additional material: devel/analyzer (though this is now a long way behind "master" in other areas)

Bugs filed against "analyzer" component

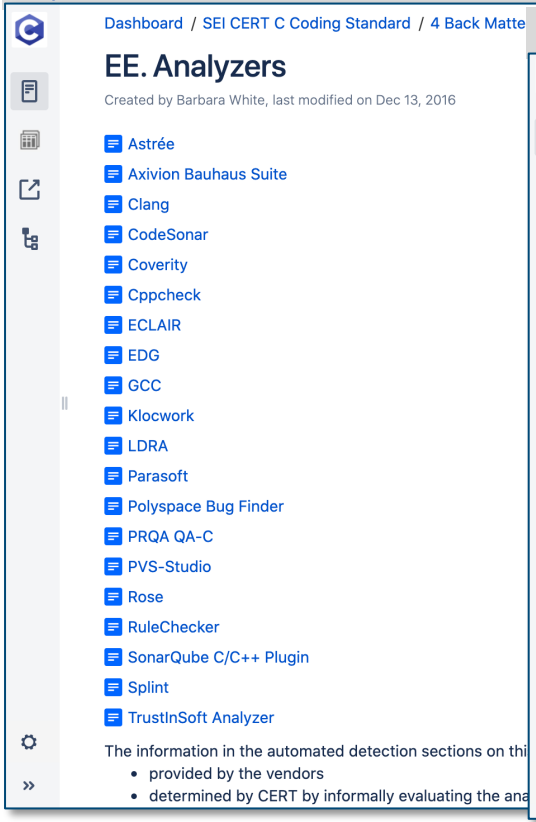
- open bugs
- all bugs (including closed)
- closed bugs
- chart
- tracker bug for reintroducing -Wanalyzer-use-of-uninitialized-value

History

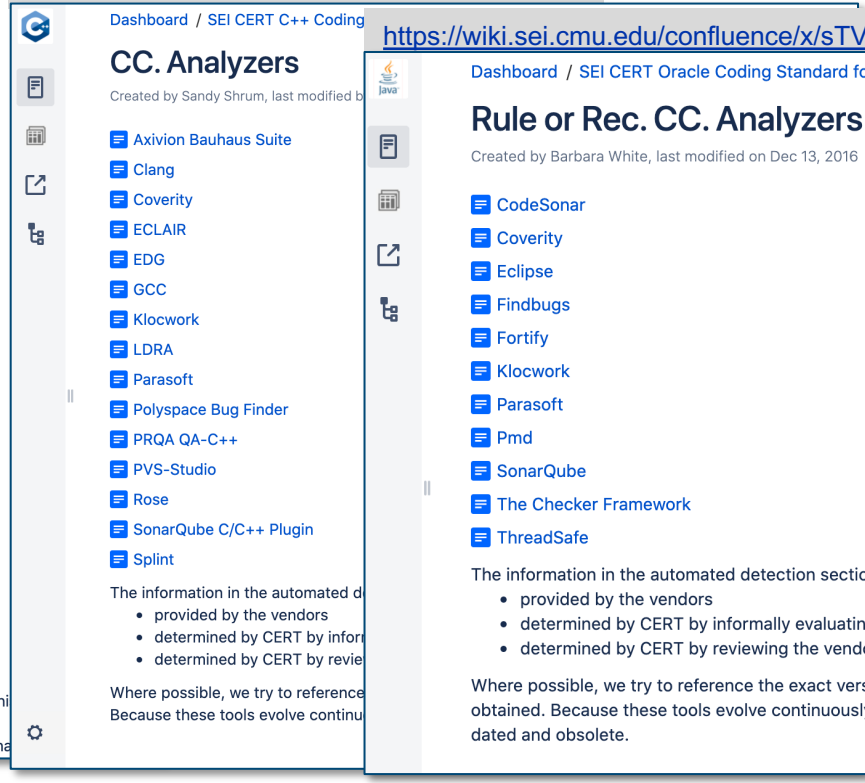
- 2020-08-13: Major rewrite of how state is tracked within the analyzer fixing numerous bugs and simplifying the implementation
- 2020-04-28: Removal of -Wanalyzer-use-of-uninitialized-value for GCC 10
- 2020-04-21: First CVE found using `-fanalyzer`, CVE-2020-1967
- 2020-03-26: Blog post: Static analysis in GCC 10
- 2020-01-15:
  - Updated branch from dmalcolm/analyzer to devel/analyzer
- 2020-01-14:

SEI CERT Coding Standards に対応状況を掲載しているツールたち

https://wiki.sei.cmu.edu/confluence/x/ctUxBQ



https://wiki.sei.cmu.edu/confluence/x/YXs-BQ



https://wiki.sei.cmu.edu/confluence/x/sTVGBQ

解析ツールを上手に使うための注意事項

- ✓ ツールの出力を正しく理解して適切に修正する
- ✓ 「未定義の動作」「未規定の動作」はどう扱ってる?
- ✓ 解析ツールの出力を疑う

注意事項: 解析ツールの出力の正しい理解

- コードレビューの作業自体は解析ツールに任せられるが, その結果を見て適切な判断をするのはヒトの作業.

- セキュアコーディングの理解がなければエラーメッセージの意味を理解して適切な修正を行うことはできない.

注意事項: 未定義、未規定、実装定義の扱い

- CやC++の言語仕様では「未定義の動作」「未規定の動作」「実装定義」とされている機能が含まれている.
- 「セキュア」なプログラムではこれらを避ける、あるいは意識してコーディングすることが重要
- 解析ツールが、このようなコードを適切に検出してくれるかどうか確認しておくべき

注意すべきコンパイラによる最適化

ある種の範囲チェックを破棄する C コンパイラの最適化の問題
(https://jvn.jp/vu/JVNVU162289/)

最適化処理において、ポインタ演算式を使った境界チェックのコードが「未定義動作」として省略され、オブジェクトコードに反映されなかった。

注意事項: 解析ツールの出力を疑う

- コード解析ツールのメッセージが常に正しいとは限らない
  - ツールはプログラマの意図までは汲み取ってくれない

- ときには、ツールのエラーメッセージを疑うことも重要

コード解析ツールを盲信してしまった例

JVNVU#925211: Debian および Ubuntu の OpenSSL パッケージに予測可能な乱数が生成される脆弱性
(https://jvn.jp/vu/JVNVU925211/)

- Debian の OpenSSL パッケージメンテナーが, Valgrind のデバッグエラーが出ないように2行コメントアウト.
- ssh-keygen の乱数生成機能に影響し, 32767個の鍵しか生成しなくなっていた
- 2年後にようやく問題が発見され修正.
- 参考: Lessons from the Debian/OpenSSL Fiasco (https://research.swtch.com/openssl)

ツールのクセを把握して使いこなそう

解析ツールにはそれぞれ得意不得意がある

- 検出しやすい / しにくい脆弱性
- エラーメッセージの分かりやすさ
- False positive はどのくらい?

プロトコルやビジネスロジックの不備などに起因する脆弱性を検知するのは難しい

- 最近だと UPnP に関する CallStranger(JVNTA#95827565) とか
- 古くは IPv6 Type0 ルーティングヘッダの問題(JVNVU#267289) とか…

解析ツールにはそれぞれ得意不得意がある

ツールの得意分野、不得意分野を知り、次の一歩へ:

- 解析ツールとマニュアルレビューの役割分担
- 異なる特徴の複数のツールを組み合わせて使う

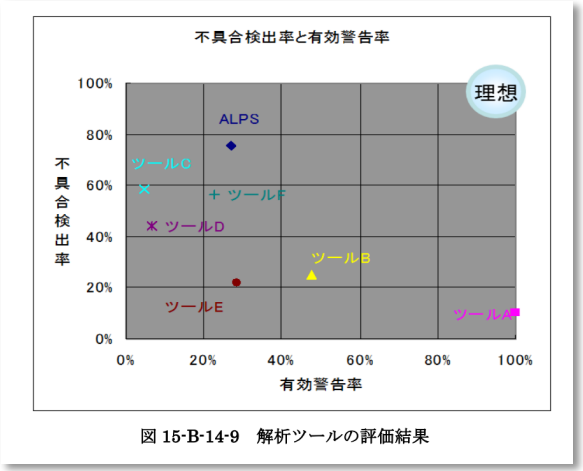


図 15-B-14-9　解析ツールの評価結果

IPA「先進的な設計・検証技術の適用事例報告書 2015年度版」から
https://www.ipa.go.jp/files/000049412.pdf

まとめ

- ✓ 脆弱性対応から予防的対応としてのセキュアコーディングへ

✓解析ツールを活用しよう

✓解析ツールを盲信するな

✓解析ツールのクセを把握して上手に使いこなそう

さらに追加の参考情報

- ❑ SAMATE – Software Assurance Metrics And Tool Evaluation (https://samate.nist.gov/Main_Page.html)
- ❑ Lessons from Building Static Analysis Tools at Google (https://cacm.acm.org/magazines/2018/4/226371-lessons-from-building-static-analysis-tools-at-google/fulltext)

お問合せはこちら…

JPCERTコーディネーションセンター

- https://www.jpcert.or.jp/
- Email: pr@jpcert.or.jp

インシデントの報告

- https://www.jpcert.or.jp/form/
- Email: info@jpcert.or.jp

セキュアコーディングに関するお問い合わせ

- https://www.jpcert.or.jp/securecoding/
- Email: secure-coding@jpcert.or.jp

※資料に記載の社名、製品名は各社の商標または登録商標です。

Thank you!