

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

JPCERT/CC®
Japan Computer Emergency Response Team Coordination Center
JPCERT コーディネーションセンター

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/a9706.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

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

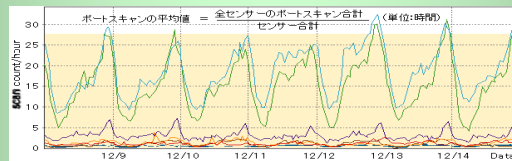


インシデントの予測と捕捉

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

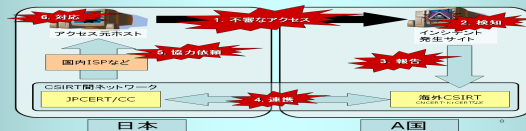


発生したインシデントへの対応

インシデントハンドリング

(インシデント対応調整支援)

- ▶ マルウェ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

マルウェア (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/ハードウェ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

ユーザを守る

開発者を守る

社会を守る

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

ソフトウェ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

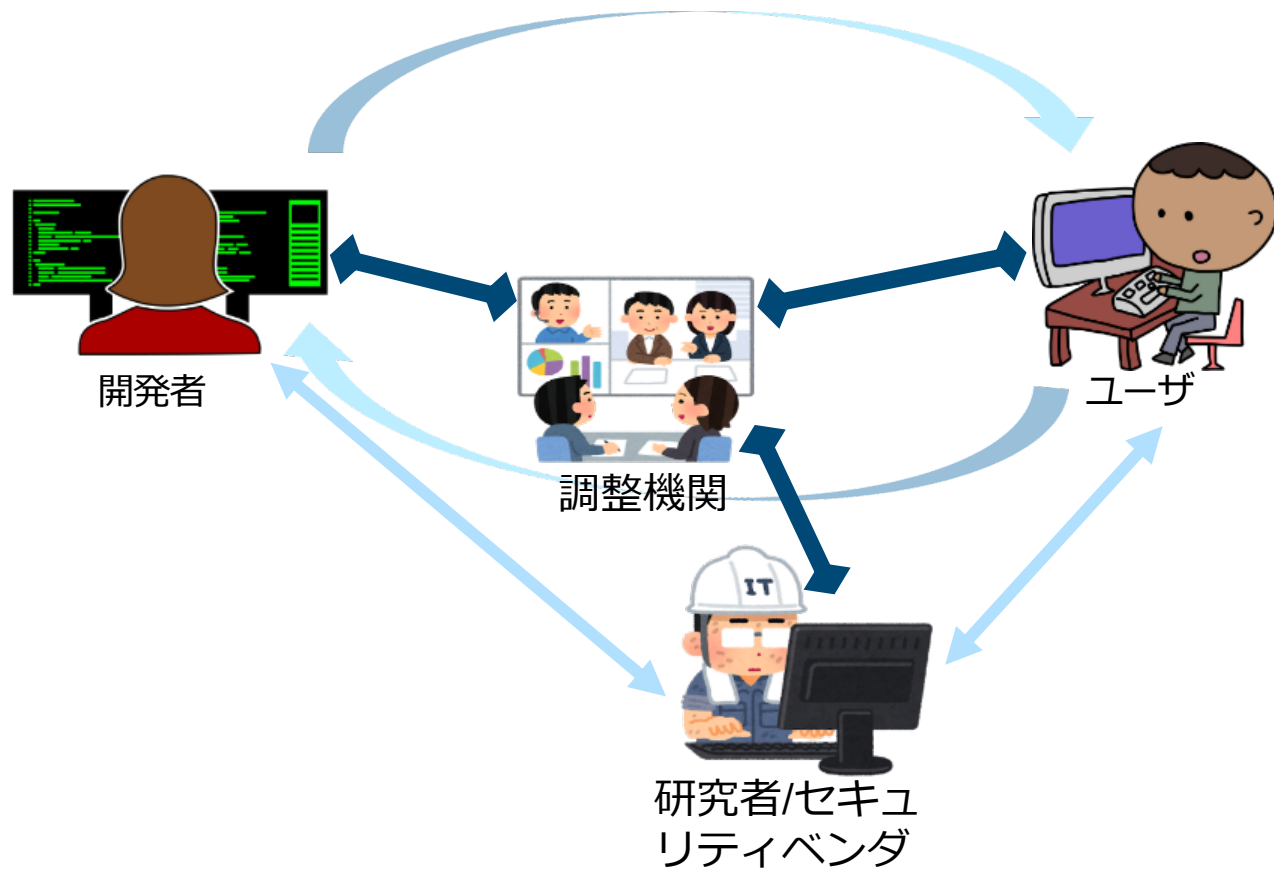
情報セキュリティ早期警戒パートナー シップガイドライン



<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/>



JVN Japan Vulnerability Notes

最終 Eng

(完了) ネットワークメンテナンスのお知らせ[2020/11/2]

新着リスト RSS

JVNVU#98542645:	InterScan Web Security シリーズ製品に複数の脆弱性	[2020/08/25 14:00]
JVN#50890770:	Apache Struts 2 にサービス運用妨害 (DoS) の脆弱性	[2020/08/25 12:00]
JVNVU#92184689:	Apache HTTP Web Server 2.4 における複数の脆弱性に対するアップデート	[2020/08/24 16:30]
JVNVU#92773731:	Diebold Nixdorf ProCash 2100xe USB ATM における暗号化の欠如に関する脆弱性	[2020/08/24 15:00]
JVNVU#99081767:	NCR SelfServ ATM BNA における複数の脆弱性	[2020/08/24 15:00]
JVNVU#93330893:	NCR SelfServ ATM における複数の脆弱性	[2020/08/24 15:00]
JVNTA#98870234:	産業用ロボット制御用のソフトウェアシステムにおける問題	[2020/08/21 18:45]
JVNVU#98069467:	ISC BIND 9 に複数の脆弱性	[2020/08/21 17:15]
JVNVU#96372881:	Philips 製 SureSigns VS4 における複数の脆弱性	[2020/08/21 17:15]
JVN#88315581:	Exment における複数のクロスサイトスクリプティングの脆弱性	[2020/08/21 12:00]
JVNTA#95473801:	トレーラーとトラクターヘッド間の電力線通信ネットワークにおける情報漏えいの脆弱性	[2020/08/20 16:30]
JVNVU#96514651:	Siemens 製品に複数の脆弱性	[2020/08/13 15:30]
JVNVU#99606488:	Intel 製品に複数の脆弱性	[2020/08/13 11:00]
JVNVU#90099158:	Schneider Electric 製 APC Easy UPS On-Line Software にパストラバサリの脆弱性	[2020/08/12 17:00]

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



JVN iPedia 脆弱性対策情報データベース

活用ガイド | [JVN iPedia English Version](#)

JVN iPedia ようこそ

JVNに掲載される脆弱性対策情報のほか、国内外問わず日々公開される脆弱性対策情報のデータベースです。

ご利用されている製品の脆弱性対策情報の収集にご活用ください。具体的な活用方法等は [こちら](#)。

脆弱性対策情報データベース検索

お知らせ

■アンケートにご協力ください
IPAが公開しているツールや資料の品質向上のため、アンケートにご協力をお願い致します。
下記リンクのアンケートフォームよりご回答ください。

[アンケートフォームはこちら](#)

JVN iPediaで注目されている脆弱性

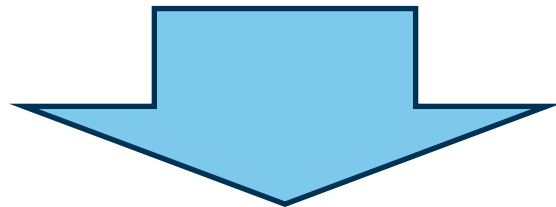
集計期間：2020/08/16 - 2020/08/22

- [JVND-2020-007506](#)
「NGINX Controller におけるクロスサイトリクエストフォージェリの脆弱性」
- [JVND-2020-007553](#)
「rails におけるクロスサイトリクエストフォージェリの脆弱性」
- [JVND-2020-007552](#)
「Helodesk におけるハードコードされた認証情報の使用に関する脆弱性」

脆弱性対応からセキュアコーディングへ

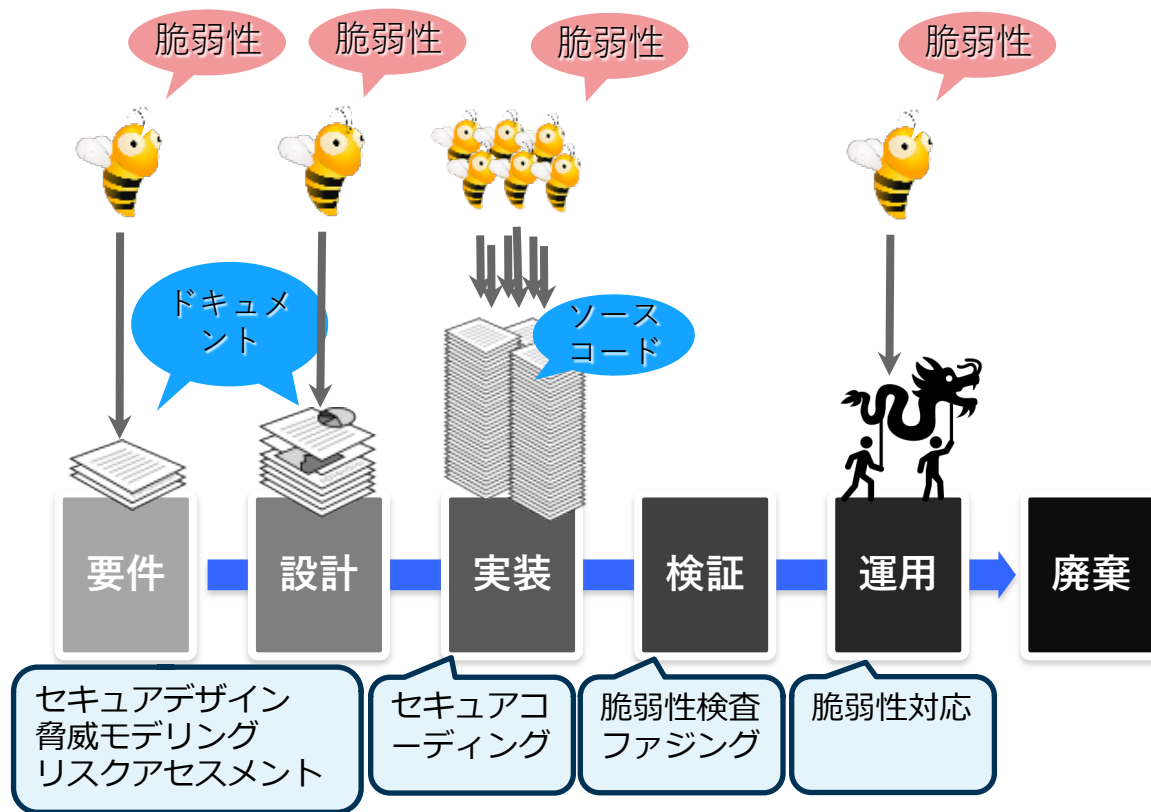
ここまでは製品リリース後に脆弱性が発見された
場合の話 (事後対応)

そもそも脆弱性を作り込まないでほしい...

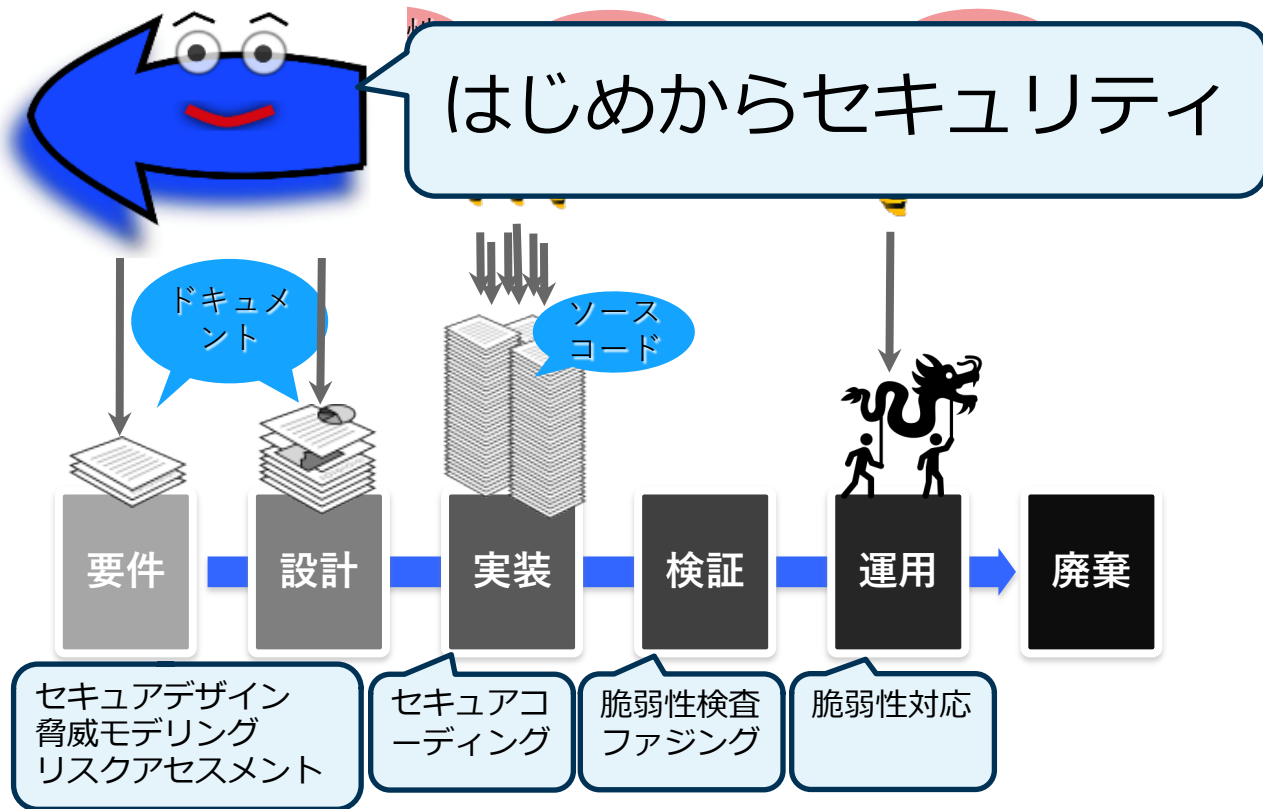


セキュアコーディング

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



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



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/>

The screenshot shows two overlapping web pages. The background page is the OWASP Top Ten website, featuring the OWASP logo and the title 'OWASP Top Ten'. It includes a navigation menu with 'Main', 'Translation Efforts', 'Sponsors', and 'Data 2020'. The main content area contains a paragraph explaining the OWASP Top 10 as a standard awareness document for developers and web application security. A large quote states: 'Globally recognized by developers as the first step towards more secure coding.' Below this, it says 'Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.' At the bottom, it lists 'Top 10 Web Application Security Risks' with the first item being '1. Injection. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a'. The foreground page is the '2020 CWE Top 25 Most Dangerous Weaknesses' page from MITRE. It features the CWE logo and the title '2020 CWE Top 25 Most Dangerous Weaknesses'. It includes a paragraph explaining the 2020 Common Weakness Enumeration as a demonstrative list of the most common and dangerous weaknesses. It also mentions that to create the 2020 list, the CWE Team leveraged the National Institute of Standards and Technology's Vulnerability Scoring System (CVSS) scores. Below this, it says 'Below is a brief listing of the weaknesses in' followed by a table of the top 7 weaknesses.

Rank	ID	Description
[1]	CWE-79	Improper Neutralization
[2]	CWE-787	Out-of-bounds Write
[3]	CWE-20	Improper Input Validation
[4]	CWE-125	Out-of-bounds Read
[5]	CWE-119	Improper Restriction of
[6]	CWE-89	Improper Neutralization
[7]	CWE-200	Exposure of Sensitive

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-owasp-top-ten/>

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

Common Weakness Enumeration

Home > CWE Top 25 > 2020

2020 CWE Top 25 Most Dangerous

The 2020 Common Weakness Enumeration demonstrative list of the most common and weaknesses are dangerous because they are a system, steal data, or prevent an application help developers, testers, and users — as well into the most severe and current security weaknesses.

To create the 2020 list, the CWE Team leveraged National Institute of Standards and Technology [Vulnerability Scoring System \(CVSS\)](#) scores weakness based on prevalence and severity.

Below is a brief listing of the weaknesses in this list.

Rank	ID	Description
[1]	CWE-79	Improper Neutralization
[2]	CWE-787	Out-of-bounds Write
[3]	CWE-20	Improper Input Validation
[4]	CWE-125	Out-of-bounds Read
[5]	CWE-119	Improper Restriction of
[6]	CWE-89	Improper Neutralization
[7]	CWE-200	Exposure of Sensitive

OWASP Top Ten

The OWASP Top 10 is a standard awareness web application security. It represents a broad critical security risks to web applications.

Globally recognized by the first step toward

Companies should adopt this document and so that their web applications minimize these risks perhaps the most effective first step towards a development culture within your organization secure code.

Top 10 Web Application Security

- Injection.** Injection flaws, such as SQL, injection, occur when untrusted data is sent

OWASP Application Security Verification Standard

The OWASP Application Security Verification Standard (ASVS) Project provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.

The primary aim of the **OWASP Application Security Verification Standard (ASVS) Project** is to normalize the range in the coverage and level of rigor available in the market when it comes to performing Web application security verification using a commercially-workable open standard. The standard provides a basis for testing application technical security controls, as well as any technical security controls in the environment, that are relied on to protect against vulnerabilities such as Cross-Site Scripting (XSS) and SQL injection. This standard can be used to establish a level of confidence in the security of Web applications. The requirements were developed with the following objectives in mind:

- **Use as a metric** - Provide application developers and application

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Source in GitHub

[Stable Release 4.0.1](#)

["Bleeding Edge" version](#)

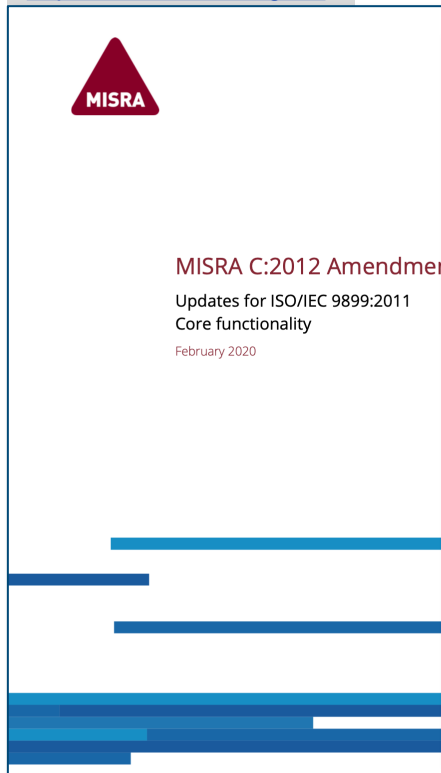
Downloads (ASVS 4.0.1)

[English PDF](#)

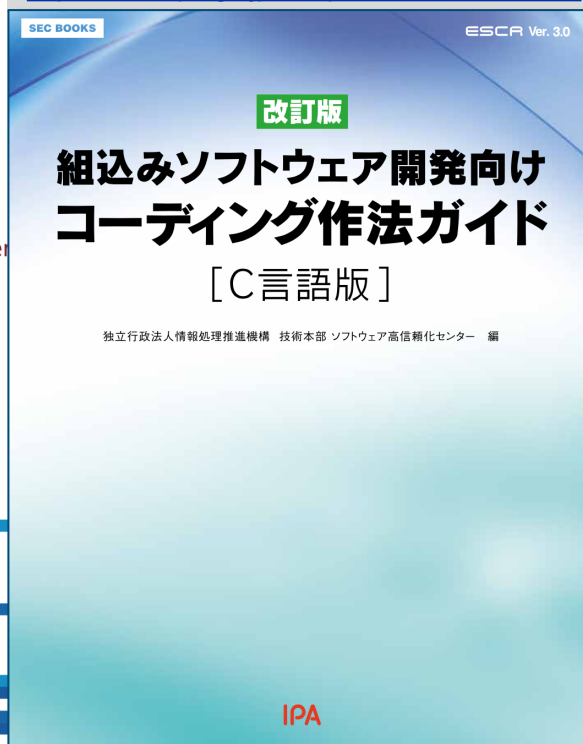
[English Word](#)

コーディングスタンダード

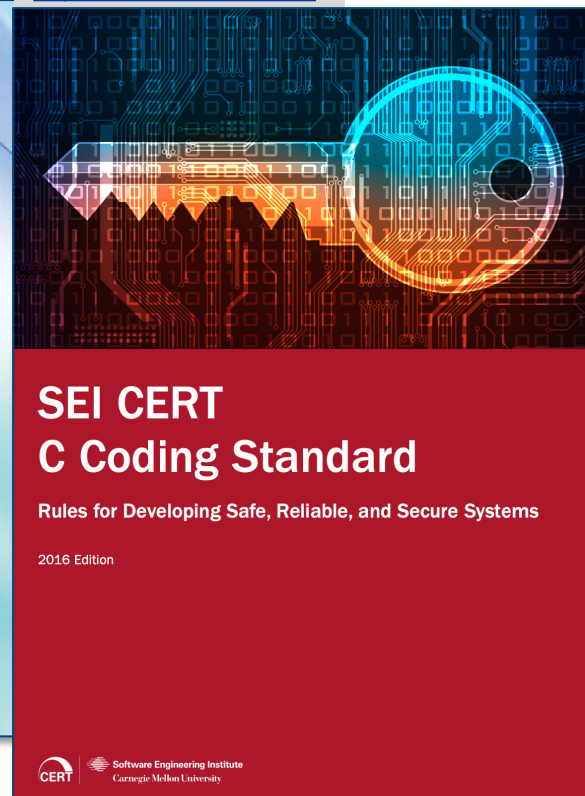
<https://www.misra.org.uk/>



<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.

Download

Mac OS X

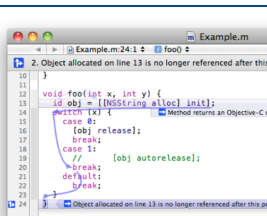
- Latest build (10.8+):
- [Release notes](#)
- This build can be used both from the command line and from within Xcode
- [Installation](#) and [usage](#)

Other Platforms

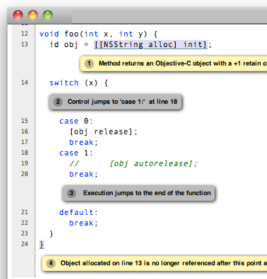
For other platforms, please follow the instructions for [building the analyzer](#) from source code.

What is Static Analysis?

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



Viewing static analyzer results



Viewing static analyzer results

<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>

Dashboard / SEI CERT C Coding Standard / 4 Back Matter

EE. Analyzers

Created by Barbara White, last modified on Dec 13, 2016

- Astrée
- Axivion Bauhaus Suite
- Clang
- CodeSonar
- Coverity
- Cppcheck
- ECLAIR
- EDG
- GCC
- Klocwork
- LDRA
- Parasoft
- Polyspace Bug Finder
- PRQA QA-C
- PVS-Studio
- Rose
- RuleChecker
- SonarQube C/C++ Plugin
- Splint
- TrustInSoft Analyzer

The information in the automated detection sections on this wiki may be

- provided by the vendors
- determined by CERT by informally evaluating the analyzer

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

Dashboard / SEI CERT C++ Coding Standard / 4 Back Matter

CC. Analyzers

Created by Sandy Shrum, last modified on Dec 13, 2016

- Axivion Bauhaus Suite
- Clang
- Coverity
- ECLAIR
- EDG
- GCC
- Klocwork
- LDRA
- Parasoft
- Polyspace Bug Finder
- PRQA QA-C++
- PVS-Studio
- Rose
- SonarQube C/C++ Plugin
- Splint

The information in the automated detection sections on this wiki may be

- provided by the vendors
- determined by CERT by informally evaluating the analyzer
- determined by CERT by reviewing the vendor documentation

Where possible, we try to reference the exact version of the tool for which the results were obtained. Because these tools evolve continuously, this information can rapidly become dated and obsolete.

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

Dashboard / SEI CERT Oracle Coding Standard for Java / 4 Back Matter

Rule or Rec. CC. Analyzers

Created by Barbara White, last modified on Dec 13, 2016

- CodeSonar
- Coverity
- Eclipse
- Findbugs
- Fortify
- Klocwork
- Parasoft
- Pmd
- SonarQube
- The Checker Framework
- ThreadSafe

The information in the automated detection sections on this wiki may be

- provided by the vendors
- determined by CERT by informally evaluating the analyzer
- determined by CERT by reviewing the vendor documentation

Where possible, we try to reference the exact version of the tool for which the results were obtained. Because these tools evolve continuously, this information can rapidly become dated and obsolete.

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
 - ー ツールはプログラム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?

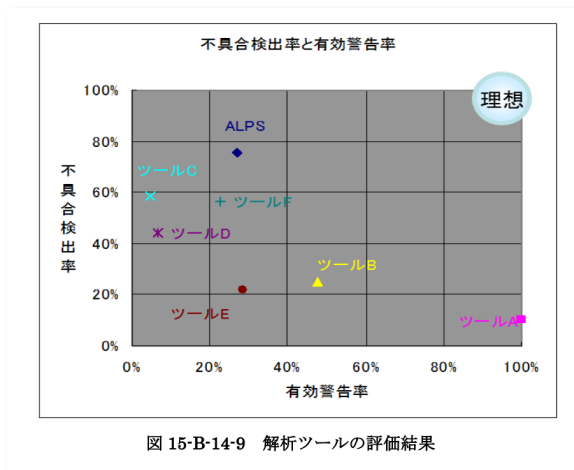
プロトコルやビジネスロジックの不備などに起因する脆弱性を検知するのは難しい

- 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る



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.jpccert.or.jp/>

— Email: pr@jpccert.or.jp

インシデントの報告

— <https://www.jpccert.or.jp/form/>

— Email: info@jpccert.or.jp

セキュアコーディングに関するお問い合わせ

— <https://www.jpccert.or.jp/securecoding/>

— Email: secure-coding@jpccert.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!

