Coverity® CERT Report

ALPR_ORIG
original

Enterprise	LG Electronics
Division	Development Team
Coverity Connect Project	ALPR_ORIG_CERT_CPP
Prepared For	cmu studio project
Prepared By	Team2 AhnLab
Prepared On	Jul 4, 2022 5:53 PM

Coverity® CERT Report

Executive Summary

This report collects results of Coverity CERT analyses for a specific project. The results are aggregated and displayed in ways that give the reader insight into the CERT compliance of software in that project and of its readiness for release.

The intended audiences of this report are software decision-makers and developers. Because the contents might be considered sensitive, it is not intended for external release.

To review detailed code-level findings, developers can click this link to the Coverity Connect platform (http://cim.lge.com:5500/reports#p10080) to see source code annotated with CERT violations and associated details.

Lines of Code Inspected: 707879

Compliance Scorecard

This project is **not consistent with CERT compliance**:

- L1 violation count: 1457
- L2 violation count: 3831
- L3 violation count: 3191

The Target Compliance Level is Fully Compliant: Complies with all of the L1, L2, and L3 rules. No violations are allowed.

Violations By Classification

The following tables display issue occurrences by classification that were found by Coverity Static Analysis. Only the occurrences of CERT issues are included in the counts.

Outstanding

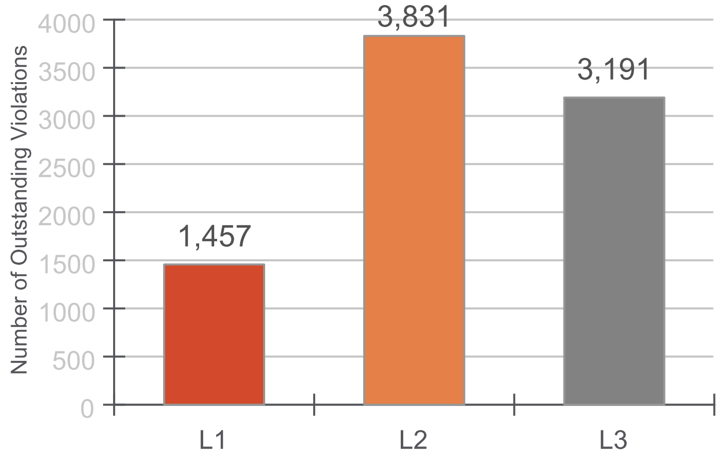
True Positive	0
Unclassified	8,479

Dismissed

False Positive	0
Has Deviation	0

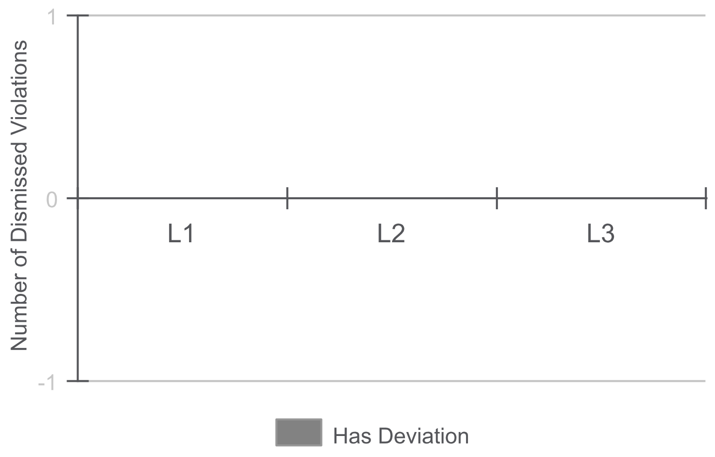
Outstanding Violations By CERT Level

A total of 8479 outstanding violations were found. Each violation has a CERT level associated with its rule. The chart below shows the number of occurrences of each of the level.



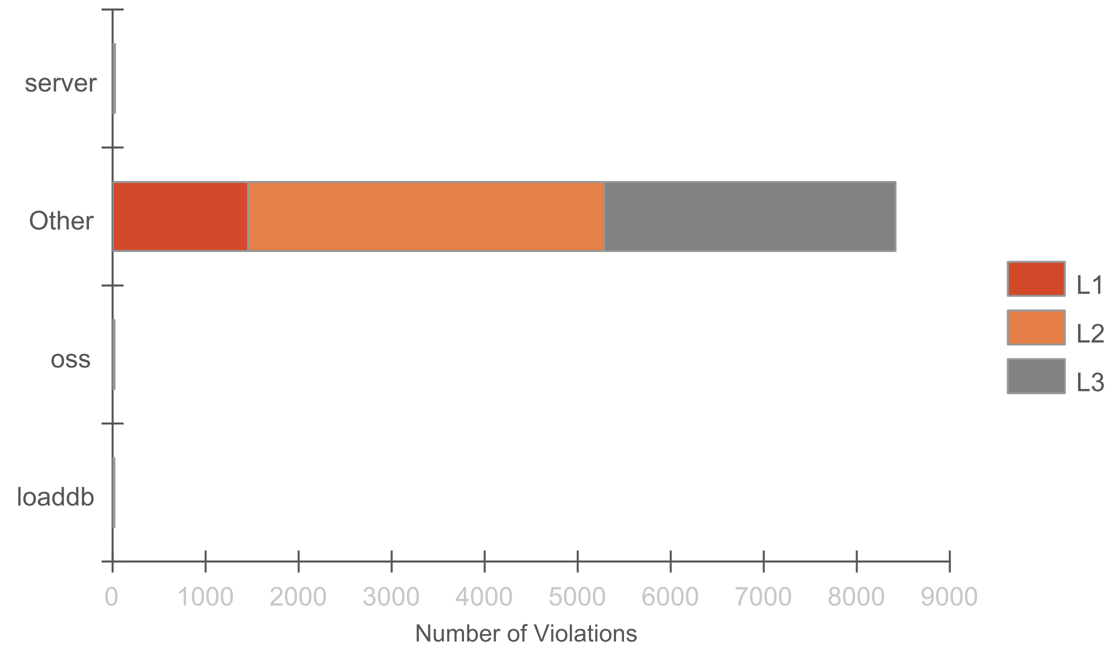
Dismissed Violations By CERT Level

A total of 0 dismissed violations were found. Each violation has a CERT level associated with its rule. The chart below shows the number of occurrences by classification for each level.



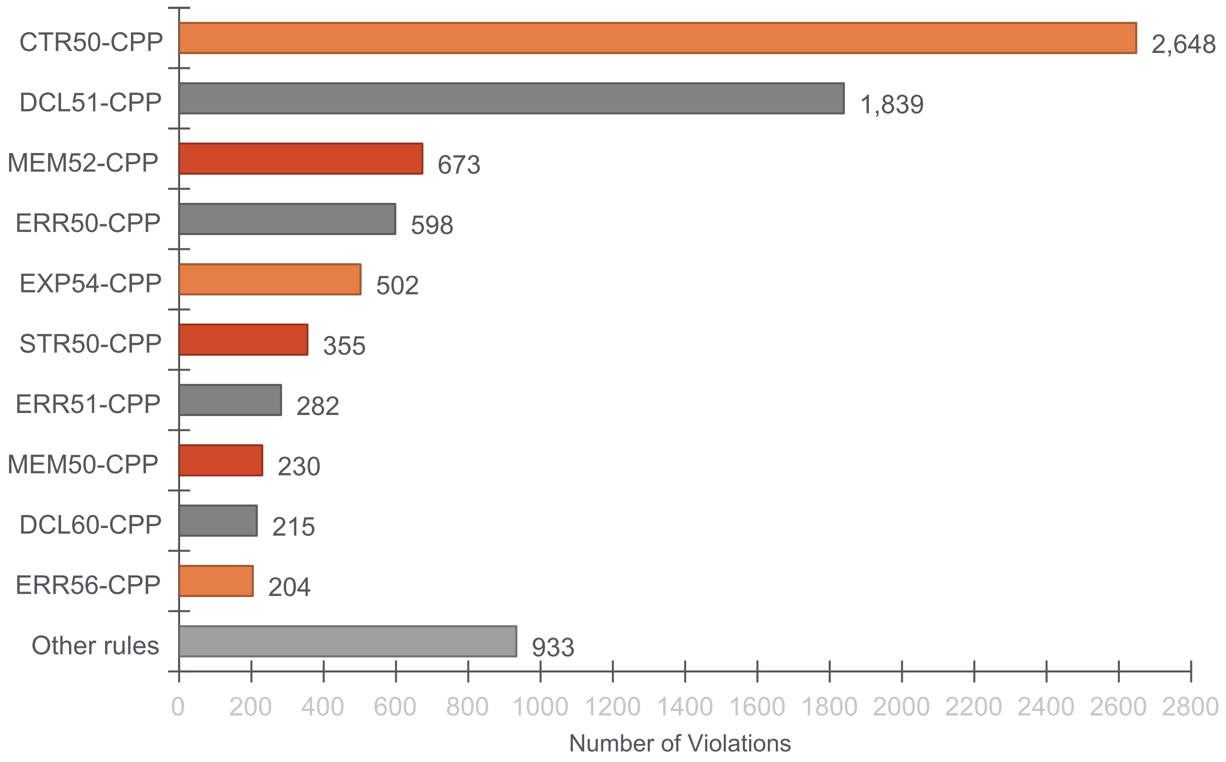
Detailed Results

Outstanding Violations By Component



Outstanding Violations By Rule

Violations are divided by rule. Bars are colored according to the rule's level.



Coverity® CERT Report

Standard: CERT C++ 2016

Rule ID	Description	Priority	Level	Supported	Enabled	Dismissed	Violations
CTR57-CPP	Provide a valid ordering predicate.	P2	L3	Yes	Yes	0	0
EXP52-CPP	Do not rely on side effects in unevaluated operands.	P3	L3	Yes	Yes	0	0
ERR57-CPP	Do not leak resources when handling exceptions.	P2	L3	Yes	Yes	0	31
STR53-CPP	Range check element access.	P6	L2	Yes	Yes	0	20
OOP51-CPP	Do not slice derived objects.	P4	L3	Yes	Yes	0	2
CON50-CPP	Do not destroy a mutex while it is locked.	P4	L3	Yes	Yes	0	0
MEM52-CPP	Detect and handle memory allocation errors.	P18	L1	Yes	Yes	0	673
EXP60-CPP	Do not pass a nonstandard-layout type object across execution boundaries.	P12	L1	Yes	Yes	0	10
DCL60-CPP	Obey the one-definition rule.	P3	L3	Yes	Yes	0	215
DCL53-CPP	Do not write syntactically ambiguous declarations.	P2	L3	Yes	Yes	0	0
EXP51-CPP	Do not delete an array through a pointer of the incorrect type.	P2	L3	Yes	Yes	0	0
ERR56-CPP	Guarantee exception safety.	P9	L2	Yes	Yes	0	204
FIO51-CPP	Close files when they are no longer needed.	P4	L3	Yes	Yes	0	2
DCL51-CPP	Do not declare or define a reserved identifier.	P3	L3	Yes	Yes	0	1839
CTR56-CPP	Do not use pointer arithmetic on polymorphic objects.	P9	L2	Yes	Yes	0	0
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime.	P18	L1	Yes	Yes	0	16
EXP61-CPP	A lambda object must not outlive any of its reference captured objects.	P6	L2	Yes	Yes	0	0
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor.	P9	L2	Yes	Yes	0	0
ERR55-CPP	Honor exception specifications.	P9	L2	Yes	Yes	0	89
DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope.	P6	L2	Yes	Yes	0	0
DCL50-CPP	Do not define a C-style variadic function.	P12	L1	Yes	Yes	0	14
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call.	P6	L2	Yes	Yes	0	0
MEM54-CPP	Provide placement new with properly aligned pointers to sufficient storage capacity.	P18	L1	Yes	Yes	0	3
DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary.	P1	L3	Yes	Yes	0	0
ERR59-CPP	Do not throw an exception across execution boundaries.	P12	L1	Yes	Yes	0	70
OOP53-CPP	Write constructor member initializers in the canonical order.	P4	L3	Yes	Yes	0	1
CTR50-CPP	Guarantee that container indices and iterators are within the valid range.	P9	L2	Yes	Yes	0	2648
STR51-CPP	Do not attempt to create a std::string from a null pointer.	P18	L1	Yes	Yes	0	8
ERR50-CPP	Do not abruptly terminate the program.	P4	L3	Yes	Yes	0	598
EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type.	P8	L2	Yes	Yes	0	203
DCL56-CPP	Avoid cycles during initialization of static objects.	P2	L3	Yes	Yes	0	0

Coverity® CERT Report

Standard: CERT C++ 2016

Rule ID	Description	Priority	Level	Supported	Enabled	Dismissed	Violations
EXP53-CPP	Do not read uninitialized memory.	P12	L1	Yes	Yes	0	75
CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread.	P1	L3	Yes	Yes	0	0
ERR58-CPP	Handle all exceptions thrown before main() begins executing.	P9	L2	Yes	Yes	0	0
INT50-CPP	Do not cast to an out-of-range enumeration value.	P4	L3	Yes	Yes	0	46
OOP54-CPP	Gracefully handle self-copy assignment.	P2	L3	Yes	Yes	0	14
OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members.	P6	L2	Yes	Yes	0	6
ERR62-CPP	Detect errors when converting a string to a number.	P4	L3	Yes	Yes	0	18
CTR58-CPP	Predicate function objects should not be mutable.	P3	L3	Yes	Yes	0	0
EXP54-CPP	Do not access an object outside of its lifetime.	P6	L2	Yes	Yes	0	502
MEM55-CPP	Honor replacement dynamic storage management requirements.	P18	L1	Yes	Yes	0	0
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string.	P6	L2	Yes	Yes	0	0
ERR52-CPP	Do not use setjmp() or longjmp().	P4	L3	Yes	Yes	0	43
CON55-CPP	Preserve thread safety and liveness when using condition variables.	P2	L3	Yes	Yes	0	0
CTR52-CPP	Guarantee that library functions do not overflow.	P18	L1	Yes	Yes	0	2
ERR60-CPP	Exception objects must be nothrow copy constructible.	P4	L3	Yes	Yes	0	32
DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions.	P6	L3	Yes	Yes	0	20
ERR61-CPP	Catch exceptions by lvalue reference.	P3	L3	Yes	Yes	0	1
MEM56-CPP	Do not store an already-owned pointer value in an unrelated smart pointer.	P18	L1	Yes	Yes	0	0
MSC54-CPP	A signal handler must be a plain old function.	P6	L2	Yes	Yes	0	0
CTR53-CPP	Use valid iterator ranges.	P6	L2	Yes	Yes	0	0
OOP56-CPP	Honor replacement handler requirements.	P2	L3	Yes	Yes	0	0
EXP57-CPP	Do not cast or delete pointers to incomplete classes.	P4	L3	Yes	Yes	0	5
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container.	P6	L2	Yes	Yes	0	0
OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions.	P6	L2	Yes	Yes	0	13
CON53-CPP	Avoid deadlocks by locking in a predefined order.	P4	L3	Yes	Yes	0	0
DCL58-CPP	Do not modify the standard namespaces.	P6	L2	Yes	Yes	0	0
CON54-CPP	Wrap functions that can spuriously wake up in a loop.	P2	L3	Yes	Yes	0	0
DCL59-CPP	Do not define an unnamed namespace in a header file.	P4	L3	Yes	Yes	0	0
MSC53-CPP	Do not return from a function declared [[noreturn]].	P2	L3	Yes	Yes	0	0
ERR51-CPP	Handle all exceptions.	P4	L3	Yes	Yes	0	282
MEM57-CPP	Avoid using default operator new for over-aligned types.	P6	L2	Yes	Yes	0	0
STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator.	P18	L1	Yes	Yes	0	355

Coverity® **CERT Report**

Standard: CERT C++ 2016

Rule ID	Description	Priority	Level	Supported	Enabled	Dismissed	Violations
EXP56-CPP	Do not call a function with a mismatched language linkage.	P2	L3	Yes	Yes	0	27
OOP58-CPP	Copy operations must not mutate the source object.	P9	L2	Yes	Yes	0	2
EXP59-CPP	Use offsetof() on valid types and members.	P4	L3	Yes	Yes	0	0
EXP63-CPP	Do not rely on the value of a moved-from object.	P8	L2	Yes	Yes	0	0
CON52-CPP	Prevent data races when accessing fields from multiple threads.	P8	L2	Yes	Yes	0	0
EXP62-CPP	Do not access the bits of an object representation that are not part of the object's value representation.	P6	L2	Yes	Yes	0	0
CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow.	P18	L1	Yes	Yes	0	0
MSC50-CPP	Do not use std::rand() for generating pseudorandom numbers.	P6	L2	Yes	Yes	0	38
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived.	P18	L1	Yes	Yes	0	0
MSC52-CPP	Value-returning functions must return a value from all exit paths.	P8	L2	Yes	Yes	0	6
CON51-CPP	Ensure actively held locks are released on exceptional conditions.	P6	L2	Yes	Yes	0	1
EXP58-CPP	Pass an object of the correct type to va_start.	P4	L3	Yes	Yes	0	0
MEM50-CPP	Do not access freed memory.	P18	L1	Yes	Yes	0	230
EXP50-CPP	Do not depend on the order of evaluation for side effects.	P8	L2	Yes	Yes	0	53
MEM51-CPP	Properly deallocate dynamically allocated resources.	P18	L1	Yes	Yes	0	1
OOP50-CPP	Do not invoke virtual functions from constructors or destructors.	P2	L3	Yes	Yes	0	15
CTR54-CPP	Do not subtract iterators that do not refer to the same container.	P8	L2	Yes	Yes	0	46
DCL52-CPP	Never qualify a reference type with const or volatile.	P3	L3	Yes	Yes	0	0
MSC51-CPP	Ensure your random number generator is properly seeded.	P18	L1	Yes	Yes	0	0
ERR53-CPP	Do not reference base classes or class data members in a constructor or destructor function-try-block handler.	P2	L3	Yes	Yes	0	0
Totals						**0**	**8479**

Coverity® **CERT Report**

Analysis Details

A Coverity project is a collection of one or more streams containing separately-analyzed snapshots. The latest snapshot in each stream is used when reporting results for a project. This section gives details about the streams and the analysis performed for each snapshot.

Stream	Snapshot Information					CERT Information		
	ID	**Analysis Date**	**Analysis Version**	**Analysis Command Line Options**	**Target**	**Configuration ID**	**Configuration Title**	**Standard**
ALPR_ORIG_	10455	2022-7-2 11:26:27 오후	2022.3.3	cov-analyze.exe --dir D: \securityspeciali \coverity \analysis \openalpr_origir --all --security --enable INTEGER_OVE --checker-option INTEGER_OVE --checker-option NULL_RETURN 0 --checker-option OVERRUN:repo --checker-option OVERRUN:stric --coding-standard-config D:\tool \ca202203\confi \coding-standards\cert-cpp\cert-cpp-all.config --strip-path D: \securityspeciali \cmu\Studio Project		867420	CERT-CPP All Rules	CERT C++ 2016

Coverity® **CERT Report**

Methodology

Introduction

This report aggregates the results of Coverity Static Analysis performed on a particular project (code base). Coverity Static Analysis is capable of finding quality defects, security vulnerabilities, and test violations through the process of scanning the output of a specially-compiled code base. The information in this report is specific to CERT vulnerabilities detected by Coverity Static Analysis.

The violations count in this report includes the count of all the Occurrences of each Coverity issue.

About Static Analysis

Static Analysis analyzes software code without executing the compiled program, for the purpose of finding logic errors or security vulnerabilities. Coverity's Static Analysis tools integrate with all major build systems and generate a high-fidelity representation of source code to provide full code path coverage, ensuring that every line of code and execution path is analyzed. Coverity Static Analysis supports the market-leading compilers for C, C++, Java, C#, Objective C, JavaScript, PHP, Python, and more.

About CERT

The CERT Division of the SEI (Software Engineering Institute) provides secure coding standards for commonly-used programming languages such as C, C++, Java and Perl, and the Android platform. These coding standards provide a set of rules and recommendations to develop safe, reliable, and secure systems.

Coverity Static Analysis finds violations based on the SEI CERT C Coding Standard.

Rules

CERT standards divide their compliance tests into a set of rules. Not all rules are amenable to being checked using Static Analysis. Those that are checkable with Static Analysis are further divided into those that the specific analysis tool actually can check and those that it cannot.

Each rule has an assigned priority, which is the product of three risk assessment values. These three values are assigned on a scale of 1 to 3 for severity, likelihood, and remediation cost, which are then multiplied together for each rule. This product provides a measure that can be used in prioritizing the rules. The priority has 10 possible values: 1, 2, 3, 4, 6, 8, 9, 12, 18, and 27. Each rule also has level, which divides the priority into one of three buckets: L3 (for 1, 2, 3, 4), L2 (for 6, 8, 9), and L1 (for 12, 18, 27).

The software team may begin remediation by implementing all rules at a particular level before proceeding to the lower-priority rules.

Compliance

Software that has been validated as complying with all Level 1 rules is considered to be L1 conforming. Software can be assessed as L1, L2, or fully conforming, depending on the set of rules by which the software has been validated.

CERT compliance requires that a software product have no defects or exploitable vulnerabilities.

Coverity Terminology

The table below compares CERT terminology to Coverity to help you understand more about the CERT violations in terms of Coverity issues.

CERT	Coverity
Violation	Occurrence
False Positive	False Positive
Has Deviation	Intentional
True Positive	Bug
Unclassified	Unclassified or Pending