Static analysis tool C-STAT

Ensuring code quality through static analysis

Catch issues before they become problems with C-STAT, the static analysis tool fully integrated into the IAR Embedded Workbench IDE. Effortlessly ensure your code complies with MISRA standards and hundreds of security checks from CWE and CERT, all without leaving your development environment.

Key Highlight Features

- Analysis of C and C++ code
- Includes more than 1000 checks in total, some comply with rules as defined by MISRA C:2023, MISRA C:2012, MISRA C++:2008 and MISRA C:2004
- More than 250 checks mapping to issues covered by CWE, SANS Top25 and OWASP
- Checks compliance with the coding standard CERT C for secure coding
- Fully integrated with the IAR Embedded Workbench IDE and the command line IAR Build Tools
- Comprehensive and detailed error information
- Fast execution
- Available for most IAR Embedded Workbench and IAR Build Tools products
- Also available as TÜV SÜD certified version for selected IAR functional safety editions

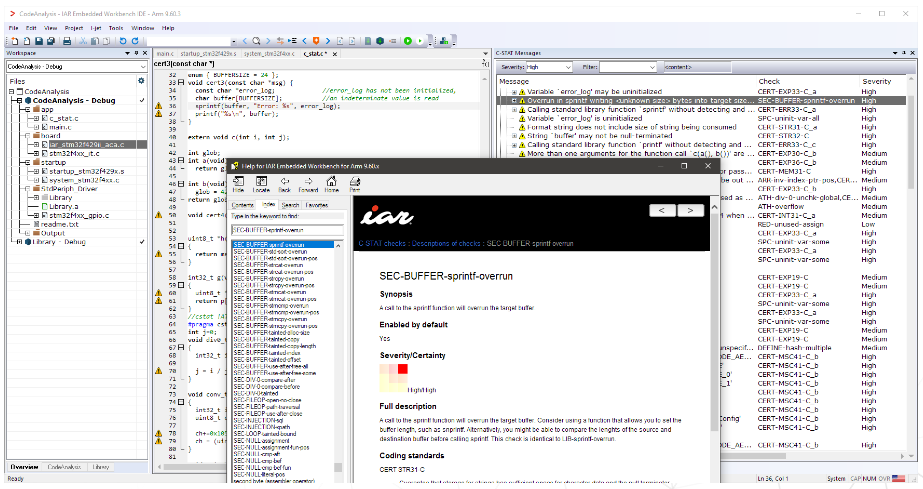
C-STAT checks code compliance with industry standards MISRA, CWE and CERT C/C++

C-STAT checks compliance with MISRA rules, CERT C/C++ Secure Coding Standards, and CWE-defined weaknesses, simplifying adherence to these standards with actionable insights and industry-aligned outputs.

MISRA (the Motor Industry Software Reliability Association) rules, initially developed for the automotive industry, are now widely used across embedded systems. MISRA-C:2023 builds on MISRA-C:2012 (and amendments) with enhanced rules for modern safety and security challenges. The rules are classified as mandatory, required, or advisory, covering areas like compiler differences, avoiding unsafe functions, limiting code complexity, and ensuring maintainability.

CWE (the Common Weakness Enumeration) is a community-developed dictionary of software weaknesses, detailing potential consequences, mitigations, code samples, and references to help manage vulnerabilities.

CERT provides secure coding rules for C and C++ to prevent coding and design errors that can lead to vulnerabilities. Each guideline includes a title, description, non-compliant code example, and compliant solutions, aimed at eliminating insecure practices and undefined behaviors.



```
enum { BUFFERSIZE = 24 };
void cert3(const char *msg) {
  const char *error_log;             //error_log has not been initialized,
  char buffer[BUFFERSIZE];           //an indeterminate value is read
  sprintf(buffer, "Error: %s", error_log);
  printf("%s\n", buffer);
}

extern void c(int i, int j);

int glob;
```

C-STAT Messages — Severity: High

Message	Check	Severity
Variable `error_log' may be uninitialized	CERT-EXP33-C_a	High
Overrun in sprintf writing <unknown size> bytes into target size...	SEC-BUFFER-sprintf-overrun	High
Calling standard library function `sprintf' without detecting and ...	CERT-ERR33-C_a	High
Variable `error_log' is uninitialized	SPC-uninit-var-all	High
Format string does not include size of string being consumed	CERT-STR31-C_a	High
String `buffer' may not be null-terminated	CERT-STR32-C	High
Calling standard library function `printf' without detecting and ...	CERT-ERR33-C_c	High
More than one arguments for the function call `c(a(), b())' are ...	CERT-EXP30-C_b	Medium

Help for IAR Embedded Workbench for Arm 9.60.x

C-STAT checks : Descriptions of checks : SEC-BUFFER-sprintf-overrun

SEC-BUFFER-sprintf-overrun

Synopsis

A call to the sprintf function will overrun the target buffer.

Enabled by default

Yes

Severity/Certainty

High/High

Full description

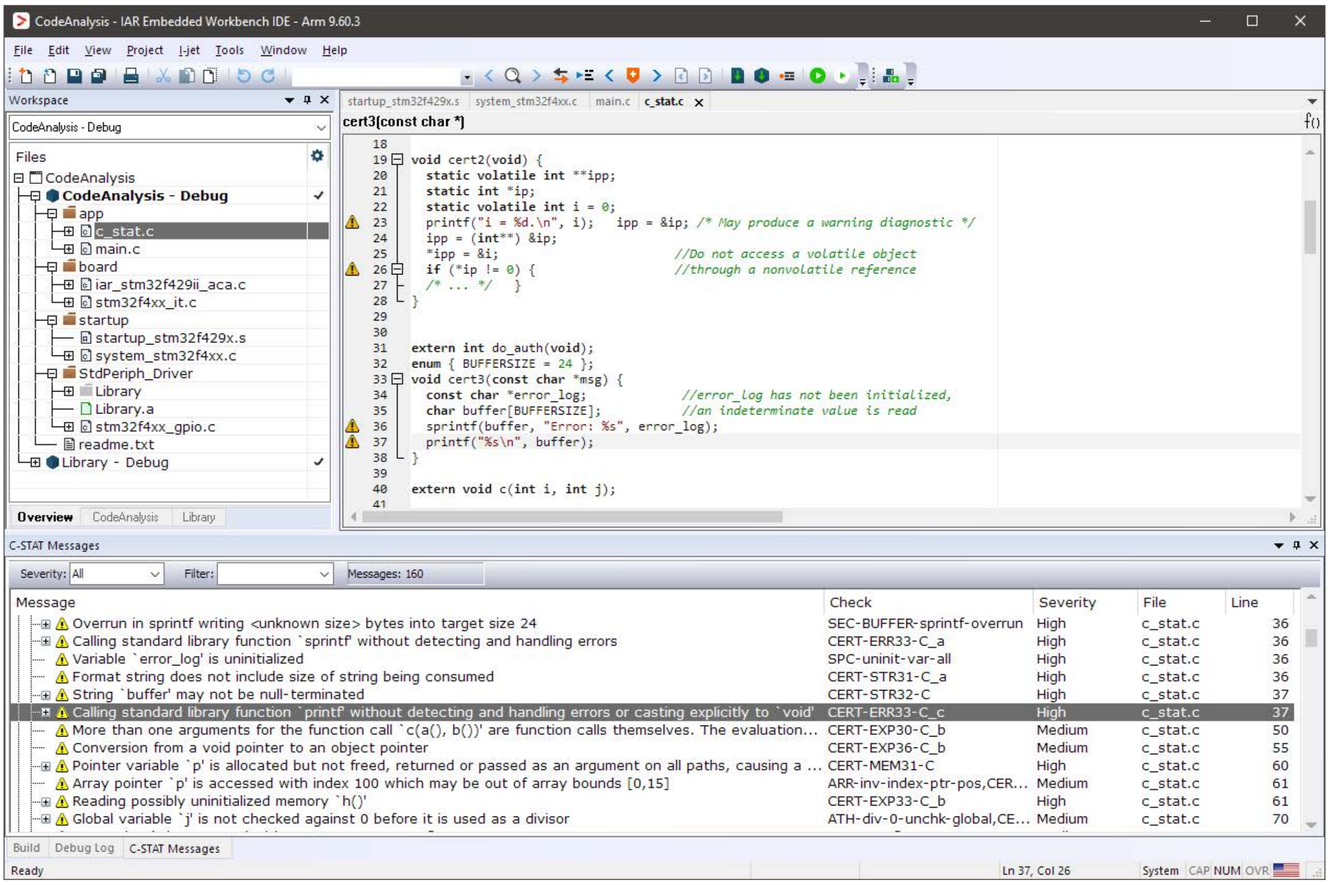
A call to the sprintf function will overrun the target buffer. Consider using a function that allows you to set the buffer length, such as snprintf. Alternatively, you might be able to compare the lengths of the source and destination buffer before calling sprintf. This check is identical to LIB-sprintf-overrun.

Coding standards

CERT STR31-C

Fully integrated into the IDE

C-STAT is fully integrated into the IAR Embedded Workbench IDE, making it as easy to use as standard build tools, with no need for complex setups. It checks your code against MISRA, CERT, and CWE standards, ensuring safety and security by referencing all relevant rules. Since these standards complement each other, C-STAT allows you to check against entire rulesets or individual rules, all thoroughly documented.



```
void cert2(void) {
  static volatile int **ipp;
  static int *ip;
  static volatile int i = 0;
  printf("i = %d.\n", i);   ipp = &ip; /* May produce a warning diagnostic */
  ipp = (int**) &ip;
  *ipp = &i;                           //Do not access a volatile object
  if (*ip != 0) {                      //through a nonvolatile reference
  /* ... */   }
}


extern int do_auth(void);
enum { BUFFERSIZE = 24 };
void cert3(const char *msg) {
  const char *error_log;                //error_log has not been initialized,
  char buffer[BUFFERSIZE];              //an indeterminate value is read
  sprintf(buffer, "Error: %s", error_log);
  printf("%s\n", buffer);
}

extern void c(int i, int j);
```

Message	Check	Severity	File	Line
Overrun in sprintf writing <unknown size> bytes into target size 24	SEC-BUFFER-sprintf-overrun	High	c_stat.c	36
Calling standard library function `sprintf' without detecting and handling errors	CERT-ERR33-C_a	High	c_stat.c	36
Variable `error_log' is uninitialized	SPC-uninit-var-all	High	c_stat.c	36
Format string does not include size of string being consumed	CERT-STR31-C_a	High	c_stat.c	36
String `buffer' may not be null-terminated	CERT-STR32-C	High	c_stat.c	37
Calling standard library function `printf' without detecting and handling errors or casting explicitly to `void'	CERT-ERR33-C_c	High	c_stat.c	37
More than one arguments for the function call `c(a(), b())' are function calls themselves. The evaluation...	CERT-EXP30-C_b	Medium	c_stat.c	50
Conversion from a void pointer to an object pointer	CERT-EXP36-C_b	Medium	c_stat.c	55
Pointer variable `p' is allocated but not freed, returned or passed as an argument on all paths, causing a ...	CERT-MEM31-C	High	c_stat.c	60
Array pointer `p' is accessed with index 100 which may be out of array bounds [0,15]	ARR-inv-index-ptr-pos,CER...	Medium	c_stat.c	61
Reading possibly uninitialized memory `h()'	CERT-EXP33-C_b	High	c_stat.c	61
Global variable `j' is not checked against 0 before it is used as a divisor	ATH-div-0-unchk-global,CE...	Medium	c_stat.c	70

IAR Embedded Workbench

IAR Embedded Workbench is a complete C/C++ development toolchain for embedded applications, offering high-quality code, optimized performance, and extensive debugging tools. The integration of C-STAT helps developers ensure code quality early, reducing errors and speeding up time to market.

Do you need to ensure your code is secure, high-quality, and standard-compliant?

We're here to support you, from project start to product life-cycle management.

Contact your local IAR team to get started: iar.com/contact!