SonarCFamily for C/C++ Rule List

Function-like macros should not be invoked without all of their arguments	Bug	cwe, misra, preprocessor	This is a constraint error, but preprocessors have been known to ignore this problem. Each argument in a function-like macro must consist of at least one preprocessing token otherwise the behaviour is undefined. See MISRA C:2004, 19.8 - A function-like macro shall not be invoked without all of its arguments. MITRE, CWE-628 - Function Call with Incorrectly Specified Arguments

Stack allocated memory should not be freed	Bug	unpredictable	Stack allocated memory, like memory allocated with the functions alloca, _alloca, _malloca, __builtin_alloca, is automatically released at the end of the function, and should not be released with free. Explicitly free-ing such memory results in undefined behavior.
			
			Noncompliant Code Example
			
			`void fun() {`
			`  char *name = (char *) alloca(size);`
			`  // ...`
			`  free(name); // Noncompliant, memory allocated on the stack`
			`  char *name2 = "name";`
			`  // ...`
			`  free(name2); // Noncompliant, memory allocated on the stack`
			`}`
			
			Compliant Solution
			
			`void fun() {`
			`  char *name = (char *) alloca(size);`
			`  // ...`
			`  char *name2 = "name";`
			`  // ...`
			`}`

Closed resources should not be accessed	Bug	cert	Using the value of a pointer to a FILE object after the associated file is closed is undefined behavior.
			
			Noncompliant Code Example
			
			`void fun() {`
			`  FILE * pFile;`
			`  pFile = fopen(fileName, "w");`
			`if (condition) {`
			`  fclose(pFile);`
			`  // ...`
			`  }`
			`fclose(pFile); // Noncompliant, the file has already been closed`
			`}`
			
			Compliant Solution
			
			`void fun() {`
			`  FILE * pFile;`
			`  pFile = fopen(fileName, "w");`
			`if (condition) {`
			`  // ...`
			`  }`
			`fclose(pFile);`
			`}`
			
			See
			CERT, FIO46-C. - Do not access a closed file

Dynamically allocated memory should be released			Memory allocated dynamically with calloc(...), malloc(...), realloc(...) or new should be released when it's not needed anymore. Failure to do so will result in a memory leak that could bring the box to its knees.
			
			This rule raises an issue when memory is allocated and not freed in the same function. Allocated memory is ignored if a pointer to it is returned to the caller or stored in a structure that's external to the function.
			
			Noncompliant Code Example
			
			`int fun() {`
			`char* name = (char *) malloc (size);`
			`if (!name) {`
			`return 1;`
			`}`
			`// ...`
			`return 0; // Noncompliant, memory pointed by "name" has not been released`
			`}`
			
			Compliant Solution
			
			`int fun() {`
			`char* name = (char *) malloc (size);`
			`if (!name) {`
			`return 1;`
			`}`
			`// ...`
			`free(name);`
			`return 0;`
			`}`
			
			See
			MITRE, CWE-401 - Improper Release of Memory Before Removing Last Reference ('Memory Leak')
			MEM00-C. - Allocate and free memory in the same module, at the same level of abstraction
			CERT, MEM31-C. - Free dynamically allocated memory when no longer needed

Freed memory should not be used			Once a block of memory has been freed, it becomes available for other memory requests. Whether it's re-used immediately, some time later, or not at all is random, and may vary based on load. Because of that randomness, tests may pass when running locally, but the odds are that such code will fail spectacularly in production by returning strange values, executing unexpected code, or causing a program crash.
			
			Noncompliant Code Example
			
			`char *cp = malloc(sizeof(char)*10);`
			`// ...`
			`free(cp);`
			`cp[9] = 0; // Noncompliant`
			
			See
			
			MITRE, CWE-416 - Use After Free
			CERT, MEM30-C. - Do not access freed memory
			CERT, MEM50-CPP. - Do not access freed memory
			CERT, EXP54-CPP. - Do not access an object outside of its lifetime
