C++ static code analysis: Appropriate memory de-allocation should be used

2 minutes

The same form that was used to create an object should always be used to delete it. Specifically, arrays should be deleted with `delete []` and objects should be deleted with `delete`. To do otherwise will cause segfaults (in the case of deleting an object with `delete []`) and memory leaks (in the case of deleting an array with `delete`).

This is also true when memory was allocated with `malloc`, or one of its variants, then it must be released using `free()` not `delete`. Similarly memory allocated by `new` can not be released using `free` instead of `delete`.

Noncompliant Code Example

```
string* _pString1 = new string;
string* _pString2 = new string[100];
char* _pChar = (char *) malloc(100);

delete [] _pString1; // Noncompliant; an object was declared but array deletion is attempted
delete _pString2;  // Noncompliant; an array was declared but an object (the first in the array) is deleted
delete _pChar; // Noncompliant
```

Compliant Solution

```
string* _pString1 = new string;
string* _pString2 = new string[100];
char* _pChar = (char *) malloc(100);

delete _pString1;
delete [] _pString2;
free(_pChar);
```

See

- CERT, MEM51-CPP. - Properly deallocate

dynamically allocated resources