Software Assurance Tips

A product of the Software Assurance Tips Team[1]

Kevin Keen

Monday 3rd April, 2023

1 Coverity BAD_CAST

Updated Wednesday 11th February, 2026

Recently we came across a finding in a Coverity scan that was a bit puzzling at first. The finding was labeled BAD_CAST. The supporting details didn't help much either. It simply indicated a conversion between incompatible types. There are probably several different reasons Coverity might flag this issue. The line of code in question this time was attempting to start a new thread by calling pthread_create. As is expected with pthread_create, they were attempting to pass a pointer to a function that would be called as the thread started. That much is normal when using pthreads. Listing 1 illustrates a fairly normal use of pthread_create using a non member function pointer as the callback.

```
#include <pthread.h>
#include <iostream>

using namespace std;

void* non_member_function(void* foo) {
        cout << "In non_member_function!" << endl;
}

int main(int argc, char** argv) {
        pthread_t thread;

        pthread_create(&thread, NULL, &non_member_function, NULL);

        pthread_join(thread, NULL);

        return 0;
}
```

Listing 1: Typical Use of a Function Poiter

Where things went wrong was that the pointer being passed was a reinterpret_cast of the address of a member function. Listing 2 shows an example of attempting to pass a member function pointer to pthread_create. Don't do this!!!

```
#include <pthread.h>
#include <iostream>

using namespace std;

class Example {
        public:
                void* member_function(void* foo);
};

void* Example::member_function(void* foo) {
        cout << "In member_function!" << endl;
}

int main(int argc, char** argv) {
        pthread_t thread;

        pthread_create(&thread, NULL, reinterpret_cast<void*(*)(void*)>(&Example::member_function, NU

        pthread_join(thread, NULL);

        return 0;
}
```
Listing 2: Improper Use of a Pointer to a Member Function

In C++, function pointers differ from pointers to member functions. A pointer to a member function might actually be represented by a data structure containing extra information, especially if it were pointing at a virtual function.[2] In fact, there is so much difference between normal function pointers and pointers to member functions that there is even special syntax for declaring a pointer to a member function. Listing 3 shows an example of declaring and using a pointer to a member function. Listing 4 shows another example, improved with the use of a `typedef`.

```
#include <iostream>

using namespace std;

class Example {
        public:
                void* member_function(void* foo);
};

void* Example::member_function(void* foo) {
        cout << "In member_function!" << endl;
}

int main(int argc, char** argv) {
        Example myExample;

        void* (Example::*memFuncPtr)(void*) = &Example::member_function;
        (myExample.*memFuncPtr)(NULL);

        return 0;
}
```
Listing 3: Example of Declaring and Using a Pointer to a Member Function

```
#include <iostream>

using namespace std;

class Example {
        public:
                void* member_function(void* foo);
};

void* Example::member_function(void* foo) {
        cout << "In member_function!" << endl;
}

int main(int argc, char** argv) {
        Example myExample;

        typedef void* (Example::*memFuncPtr)(void*);
        memFuncPtr myPtr = &Example::member_function;
        (myExample.*myPtr)(NULL);

        return 0;
}
```
Listing 4: Use of Typedef with Pointer to Member Function to Improve Readability

A non-static member function can't really exist on its own. It is part of a class and must exist as part of that class. Trying to invoke it apart from the class being instantiated doesn't make much sense (unless the function is declared static[2]). Note that in the example code in Listings 3 and 4, we had to use an object (`myExample`) in order to invoke the member function through the pointer.

Good compilers will give a warning if you try to cast a pointer to a member function into a regular function pointer, but will normally allow you to do so anyway. Even ***IF*** it happens to work as expected when tested, doing this is strictly in the land of undefined behavior. It is not guaranteed to work, not portable, and could break in the most spectacular fashion at any point in the future.

If you find yourself in this situation, take heart! There is a solution! Simply have a non-member function as an intermediate step. That intermediate function can take a pointer or reference to the desired object and call the member method. And because it is a non-member function itself, it is safe to create a function pointer that can be passed to functions like `pthread_create`. Problem solved!

```
#include <pthread.h>
#include <iostream>

using namespace std;

class Example {
        public:
                void* member_function(void* foo);
};

void* Example::member_function(void* foo) {
        cout << "In member_function!" << endl;
}

void* intermediate_func(void* objPtr) {
        Example* examplePtr = (Example*)objPtr;
        examplePtr->member_function(NULL);
}

int main(int argc, char** argv) {
        Example myExample;

        pthread_t thread;

        pthread_create(&thread, NULL, &intermediate_func, myExample);

        pthread_join(thread, NULL);

        return 0;
}
```

Listing 5: Using an Intermediate Function

References

[1] Jon Hood, ed. SwATips. `https://www.SwATips.com/`.

[2] Standard C++ Foundation. “Pointers to Members”. In: (). URL: `https://isocpp.org/wiki/faq/pointers-to-members`.