

ISIS - Crash #1489

Dangling Pointer in Conversion of QString to char *

2013-02-08 09:20 AM - Kris Becker

Status: Closed	
Priority: Block	
Assignee: Steven Lambright	
Category: API	
Target version: N/A	
Impact:	Software Version:
Description	
<p>There are numerous occasions where QStrings are converted to char * pointers in our ISIS API and main applications. Many of those conversions use a shortened code sequence using the QString::toAscii() method. This method creates a QByteArray that has methods to get at the null terminated raw char * pointer. Much of the code looks like this:</p>	
<pre>QString sTableFile = QString(QString(cFileTable_Copy[i][0]).toAscii().data());</pre>	
<p>The issue is that the QString::toAscii() method may result in a conversion of some sort and, therefore, a QByteArray object is created holding the new string. The QByteArray::data() method returns the pointer to the converted data and immediately the QByteArray is destroyed resulting in the dangling pointer problem (the object is destroyed before the data is accessed via the pointer).</p>	
<p>This issue is confirmed by this Qt FAQ: http://qt-project.org/faq/answer/how_can_i_convert_a_qstring_to_char_and_vice_versa. Although the FAQ shows the use of the QString::toLocal8Bit() method, all those types of methods return the same QByteArray constructed object. See also the QString documentation at http://qt-project.org/doc/qt-4.8/qstring.html.</p>	
<p>The FAQ shows the correct way to code this by declaring a QByteArray explicitly and then using the QByteArray::data() method.</p>	
<p>A quick scan of our Beta ISIS system shows that there are 188 source files and 5 include headers that will have to be visually inspected to determine if the usage/context of QString::toAscii() results in a dangling pointer. I did not scan the system for use of the other QString methods that result in the same behavior, so that will also need to be performed as well.</p>	
<p>This is a very subtle but serious bug and for that reason, I put a block status on the entire ISIS system (this really should be fixed before the next release). It doesn't appear to cause frequent problems, but nonetheless, I happen to discover the problem using this same code when calling a third party package (GEOS) with a very large WTB hexadecimal string. It threw an error because the memory pointing to the string had changed that included erroneous non-hexadecimal characters.</p>	
Related issues:	
Related to ISIS - Recommendation #1051: Refactor iString	Closed

History

#1 - 2013-02-08 12:56 PM - Tammy Becker

Triaged high based on reporter's description

#2 - 2013-02-11 03:38 PM - Steven Lambright

The code example above is valid code; the Qt FAQ referenced explicitly shows a primitive string on the left hand side (which would be invalid). The code in the ticket description has a class type of QString, which will copy the temporary's data before the temporary is destroyed.

Some references about temporary variable lifetimes:

<http://bytes.com/topic/c/answers/546970-scope-temporary-variables>

<http://stackoverflow.com/questions/584824/guaranteed-lifetime-of-temporary-in-c>

<http://msdn.microsoft.com/en-us/library/a8kfxa78%28v=vs.71%29.aspx>

<http://publib.boulder.ibm.com/infocenter/lnxpcomp/v8v101/index.jsp?topic=%2Fcom.ibm.xlcpp8l.doc%2Flanguage%2Fref%2Fcp1r382.htm>

I haven't found any references disputing these explanations of temporary object lifetimes.

This is an example of code that demonstrates the problem we ran into long ago with STL strings:

```
string str("Hello");
const char *tmp = (str + " World").c_str();
cout << tmp << endl;
```

The fix of this problem looks like:

```
string str("Hello");  
cout << (str + " World").c_str() << endl;
```

OR

```
string str("Hello");  
string tmp = str + " World";  
cout << tmp.c_str() << endl;
```

I also have a test program available at: /home/slambright/misc/constructor_destructor/test (compiled currently on Linux, feel free to copy and play with this)

#3 - 2013-02-11 05:04 PM - Kris Becker

Can you confirm that of the 190+ occurrences of those constructs none of them are used improperly?

#4 - 2013-02-12 10:14 AM - Steven Lambright

Yes.

#5 - 2013-02-12 01:48 PM - Kris Becker

Ok, thanks.

My apologies for the false alarm. I'll be more careful/thorough next time.