Subject: Possibly memory leak SSGM
Posted by Neijwiert on Thu, 01 May 2014 17:47:52 GMT
View Forum Message <> Reply to Message

So I was going trough some code and I found my way to the file gmlog.cpp

The logger keeps track of the current socket connections it has in a SimpleDynVecClass list. However this class does not use the delete operator on its contents when you call ::Delete.

So I found these 2 leaks:

- in void SSGMGameLog::Think()

where you gracefully close the sockets upon error or something else you call Connections.Delete(Connections[i]); this does not however delete the resources to Connections[i].

- in void SSGMGameLog::Shutdown()

you loop trough all connections and close the sockets, however no memory is released trough this process.

All objects inside this list are created with the new operator, however no subsequent delete operators are called.

I might be completly wrong at this, but anyway it's worth reporting i suppose...

EDIT:

Further explenation:

```
bool Delete(int index,bool allow_shrink = true)
 {
  if (index < ActiveCount-1)
  {
   memmove(&(Vector[index]),&(Vector[index+1]),(ActiveCount - index - 1) * sizeof(T));
  }
  ActiveCount--;
  if (allow_shrink)
  {
   Shrink();
  }
  return true;
 }
 bool Delete(T const & object,bool allow_shrink = true)
 {
  int id = Find_Index(object);
  if (id != -1)
  {
   return Delete(id,allow_shrink);
  }
  return false;
 }
```

these are the 2 overloaded functions inside SimpleDynVecClass. As you can see it only uses memmove which doesn't guarantee that the resources are overwritten. The code in gmlog.cpp calls the bottom Delete first, which then calls the top one if there's an index found.

Subject: Re: Possibly memory leak SSGM
Posted by danpaul88 on Fri, 02 May 2014 09:20:00 GMT
View Forum Message <> Reply to Message

SimpleDynVec stores all objects in a shared block of allocated memory, thus that usage of memmove is perfectly fine, the location it was moved from is inside the same block of allocated memory which will be reused when the next thing is added to the vector. It's a common pattern used to avoid allocating blocks of memory too frequently, std::vector does the same sort of thing. Don't confuse deleting something from the vector (removing the pointer to the struct from the vector) and freeing the memory allocated for the struct, they are two distinct operations which do different things.

You are however correct that the new'd instance of the Connection struct appears to be leaked on line 78 (Connections.Delete(Connections[i]);). Technically it is also leaked in Shutdown() but that is only called when the FDS is shutting down so it's not a massive problem although should be fixed for consistency.

Note it is NOT necessary to call Delete on the actual vector in shutdown() because the vector will erase itself when it goes out of scope (which occurs when the class instance which owns it is deconstructed), although it would perhaps be good practice to call Delete_All on the vector after looping through closing all the sockets just to avoid leaving dead sockets in the vector

---

Subject: Re: Possibly memory leak SSGM
Posted by Neijwiert on Fri, 02 May 2014 10:43:29 GMT
View Forum Message <> Reply to Message

danpaul88 wrote on Fri, 02 May 2014 02:20SimpleDynVec stores all objects in a shared block of allocated memory, thus that usage of memmove is perfectly fine, the location it was moved from is inside the same block of allocated memory which will be reused when the next thing is added to the vector. It's a common pattern used to avoid allocating blocks of memory too frequently, std::vector does the same sort of thing. Don't confuse deleting something from the vector (removing the pointer to the struct from the vector) and freeing the memory allocated for the struct, they are two distinct operations which do different things.

You are however correct that the new'd instance of the Connection struct appears to be leaked on line 78 (Connections.Delete(Connections[i]);). Technically it is also leaked in Shutdown() but that is only called when the FDS is shutting down so it's not a massive problem although should be fixed for consistency.

Note it is NOT necessary to call Delete on the actual vector in shutdown() because the vector will erase itself when it goes out of scope (which occurs when the class instance which owns it is deconstructed), although it would perhaps be good practice to call Delete_All on the vector after looping through closing all the sockets just to avoid leaving dead sockets in the vector

Ok was just making sure. Atleast I learn something from it

Subject: Re: Possibly memory leak SSGM
Posted by StealthEye on Fri, 02 May 2014 18:13:38 GMT
View Forum Message <> Reply to Message

To add to what was already said:

The main difference between (Dynamic)VectorClass and Simple(Dyn)VecClass is that the latter variant assumes that the type can be constructed, destructed, and copied like plain memory. No (copy/move) constructors and destructors are called. Therefore the Simple* version should only be used with simple types (e.g. primitives, pointers, but not usually classes).

But yeah, they are real leaks, and they should ideally be fixed. There are many more small leaks like these though, I'm sure. It's not that harmful.