I got an array with plenty of nuthin'

devblogs.microsoft.com/oldnewthing/20091218-00

December 18, 2009

Raymond Chen

A customer reported a memory leak in the function `PropVariantClear`:

> We found the following memory leak in the function `PropVariantClear`. Please fix it immediately because it causes our program to run out of memory.
>
> If the `PROPVARIANT`'s type is `VT_ARRAY`, then the corresponding `SAFEARRAY` is leaked and not cleaned up.
>
> ```
> SAFEARRAY* psa = SafeArrayCreateVector(VT_UNKNOWN, 0, 1);
> PROPVARIANT v;
> v.vt = VT_ARRAY;
> v.parray = psa;
> PropVariantClear(&v);
> // The psa is leaked
> ```
>
> Right now, we are temporarily working around this in our program by inserting code before all calls to `PropVariantClear` to free the `SAFEARRAY`, but this is clearly an unsatisfactory solution because it will merely result in double-free bugs once you fix the bug. Please give this defect your highest priority as it is holding up deployment of our system.

The `VT_ARRAY` value is not a variant type in and of itself; it is a type *modifier*. There are other type modifiers, such as `VT_VECTOR` and `VT_BYREF`. The thing about modifiers is that they need to *modify something*.

> The line `v.vt = VT_ARRAY` is incorrect. You have to say what you have a safe array *of*. In this case, you want `v.vt = VT_ARRAY | VT_UNKNOWN`. Once you change that, you'll find the memory leak is fixed.

The customer didn't believe this explanation.

> I find this doubtful for several reasons.
>
> 1. While this would explain why the `IUnknown`s in the `SAFEARRAY` are not released, it doesn't explain why the `SAFEARRAY` itself is leaked.
> 2. The `SAFEARRAY` already contains this information, so it should already know that destroying it entails releasing the `IUnknown` pointers.
> 3. If I manually call `SafeArrayDestroy`, then the `IUnknown`s are correctly released, confirming point 2.
> 4. The function `SafeArrayDestroy` is never called; that is the root cause of the problem.

The customer's mental model of `PropVariantDestroy` appeared to be that it should go something like this:

```
if (pvt->vt & VT_ARRAY) {
 switch (pvt->vt & VT_TYPEMASK) {
 ...
 case VT_UNKNOWN:
  ... release the IUnknowns in the SAFEARRAY...
  break;
 ...
 }
 InternalFree(pvt->psa->pvData);
 InternalFree(pvt->psa);
 return S_OK;
}
```

In fact what's really going on is that the value of `VT_ARRAY` is interpreted as `VT_ARRAY | VT_EMPTY`, because (1) `VT_ARRAY` is a modifier, so it has to modify something, and (2) the numeric value of zero happens to be equal to `VT_EMPTY`. In other words, you told OLE automation that your `PROPVARIANT` holds a `SAFEARRAY` filled with `VT_EMPTY`.

It also happens that a `SAFEARRAY` of `VT_EMPTY` is illegal. Only certain types can be placed in a `SAFEARRAY`, and `VT_EMPTY` is not one of them.

The call to `PropVariantClear` was returning the error `DISP_E_BADVARTYPE`. It was performing parameter validation and rejecting the property variant as invalid, because you can't have an array of nothing. The customer's response to this explanation was very terse.

> Tx. Interesting.

Raymond Chen

Follow