Subject: Re: CALL_EXTERNAL puzzle (still) ?
Posted by steinhh on Fri, 04 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

> Thanks for your comments, but ... after some additional testing, I still
> maintain that something is not clear about CALL_EXTERNAL. Firstly,
> regardless of the unconventional setup of my program, it *worked*. What
> bothered me was that I didn't understand why (i.e. it worked for reasons that
> would not be clear if one reads the code).

Hmm - I agree that this is extremely confusing - having
the code work for some plain wrong reason... It's only a freak
thing, as far as I can see. Your original example caused
the following output on my machine (AlphaServer, Digital Unix).

```
IDL> example
de-referenced pointer: 961081302
*IDL____Value of argc: 1
                        Segmentation fault
```

As far as I can see, your original IDL program sends (through argv[0]) a
pointer to an IDL_VARIABLE structure containing an IDL_TYP_PTR variable
(see "Type Codes" in external development guide).

My guess is that out of "bad luck", your string was
stored in the memory locations right after this IDL_VARIABLE
structure, and that the actual values in the IDL_VARIABLE
itself came out as non-printable characters. I.e., your
C pointer "filename" pointed to a memory area containing:

```
  [IDL_VARIABLE(as byte values)] [STRING TEXT]
  00 01 01 02 02 01 04 05 02 02  "This is the file name"
```

and when you printf'ed it, it looked like you were doing
the right thing, because what appeared on the screen was
"This is the file name".

>   However, why can't I pass a pointer?

Good question - I would have liked to be able to exploit
pointers inside external code, but RSI has (at the moment)
specifically forbidden that:

    Direct access to pointer and object reference heap
    variables (types IDL_TYP_PTR and IDL_TYP_OBJREF,
    respectively) is not allowed.

(External dev. guide, "Heap Variables")

So, although the pointer is passed, you cannot do anything
with it. Please, RSI? Why not some function like

```
 IDL_VPTR IDL_GetHeapVariable(HVID)
```

> And if I want to pass a pointer, and
> print the value of another pointer just before the CALL_EXTERNAL, why is the
> wrong one passed?

These are how your "bad luck" arose, apparently. Probably the
print statement caused the text to be stored right after the
space where the first pointer was located? Expect *no*
consistency from platform to platform, or IDL version to IDL
version!

> Any further comments?

I would recomment starting to use the "export.h" file that
defines the IDL_VARIABLE data type, and always accepting
parameters to external code by reference, not value.
This means you'll always get pointers to IDL_VARIABLE
structures, and you can do consistency checks on them etc. Sure,
it adds another "level" of access to get to the data, but
believe me, that's fine: More chances to *crash* things
instead of having spurious "Gosh, this worked, I must be
doing the right thing" experiences.

And then, with some experience of this, the jump isn't too big
(okay, it *is* scary!) to go on to Callable IDL (allows access
to IDL routines from within your external code, very neat!)
and LINKIMAGE, (or the Callable IDL equivalent IDL_AddSystemRoutine,
or even dynamically loadable modules -- hmmm, just discovered
those in the online docs thanks to your question!).

Regards,

Stein Vidar
(Hmh! - I should have chosen to do a PhD *on* IDL, not *with* IDL)