

Re: porting problems encountered

Source: <http://unix.derkeiler.com/Newsgroups/comp.os.vms/2003-12/0865.html>

From: RC Bryan (rcbryan_at_hotmail.com)

Date: 12/09/03

Date: 9 Dec 2003 09:30:28 -0800

```
> > char * sub(char *p)
> > {
> > int myP=p;
> > myP = (myP+15)& 0xfffff0; /* add and mask back down */
> > p=(char *)myP;
> > ...
> >
> > What exactly happens here ? Why are we doing this & 0xffff... ? If we
> > add my+1 then value would be myP + 1*sizeof(myP). Please correct me if
> > i am wrong .
```

You are wrong. This example is doing integer arithmetic on a pointer value. We are adding 15 to myP, an integer, not p, the pointer. If p had the value 0x4000F44, by adding 15 (decimal) (or 0xF hex) we get 0x4000f53. We then mask off the low order bits to get a 16 byte bounded space. 0x4000f53& 0xfffff0 == 0x4000f50. If we had not added 15, we would have 0x4000f40 which is a pointer to a space before the start of the p area. (Don't forget C lesson 1, & is bitwise and && is logical (the whole variable).) I actually saw this where we needed page bounded space which is particularly not portable. Page bounding or 16 byte bounding is the same idea except you can use 511 and 0xfffffe00 rather than 15 and 0xfffff0 (on some machines).

```
>
> > Using the struct from before:
> >
> > p=malloc(sizeof(mystruct)*2);
> > p+=sizeof(int);
> > intVal=((struct MYSTRUCT*)p)->val; /* val is long*/
> >
> > This will run with a warning on some platforms, crash on others and
> >
> > Is the problem due to assigning of long to int ?
```

No, it is due to trying to extract a long value from a value that is not properly bounded. 32 bit machines like to fetch 32 bit values

from addresses that end in 0,4,8, and c, 32 bit bounded addresses. 32 bit machines are fine with fetching 64 bit values from 32 bit bounded addresses since it requires two operations internally. On a 64 bit machine, they like to fetch 64 bit values from addresses that end in 0 and 8. On many machines, if you try to fetch values from improperly bounded addresses, you will get an exception. Interestingly, it depends upon the compiler and the O/S. I have seen things pass on, say Alpha-VMS and fail on Alpha-Tru64, the same hardware with a different operating system. Since this is a VMS newsgroup, I will note that one of the neat things about the original VAX was that you could fetch anything from any boundary. If I recall correctly, this included arbitrary bit fields, you were not even restricted to byte boundaries.

Here is another good one:

```
float a=1.0;
printf("%x", a);
```

This should give 0x3f800000 (for IEEE floating point) but on many machines it will give anything but.

The reason why has to do with the way the compiler passes arguments to subroutines. On many machines, it passes floating point arguments in floating point registers. Meanwhile, the printf subroutines does not know anything about the type of the argument and prints out what is in an integer register and you get garbage.

It took a lot of head scratching before I figured out what was going on the first time I saw this one. The solution is to have a:

```
union
{
  float f;
  int i;
}ftoi;
```

> *Thanks [RC] Bryan for the long post. Most it was useful for me . I have
> some doubts in understanding of the code.*

I am afraid to understand some of the hardware problems you have to have studied the machine code to see what is going on. Sometimes understanding THAT requires understanding what the hardware is doing. When I was in school, some of the guys would complain, "I am not going to be coding in assembly, why do I have to learn this?" The answer is that like it or not, at some point you will have bugs that can only be resolved by understanding what the machine is doing. (This went along with the Mechanical Engineering students who complained to me (another ME who just happened to work for Computer Services) "Why do I have to learn to program? I am an ME." I had a lot of sympathy for them.)

comp.os.vms: Re: porting problems encountered

Regards,
/RC Bryan