A short introduction to Valgrind

nct@ysagoon.com

08 November 2003

What is Valgrind ?

Valgrind is an an open-source memory debugger for x86-GNU/Linux that can detect :

- Use of uninitialised memory
- Reading/writing memory after it has been free'd
- Reading/writing off the end of malloc'd blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks – where pointers to malloc'd blocks are lost forever
- Passing of uninitialised and/or unaddressible memory to system calls
- Mismatched use of malloc/new/new [] vs free/delete/delete []
- Some misuses of the POSIX pthreads API

First example : deference of unassigned pointer

```
int main(int argc, char *argv[])
{
    int *a;
    *a = 0;
}
```

First example : deference of unassigned pointer

```
int main(int argc, char *argv[])
{
    int *a;
    *a = 0; <- a is an unassigned pointer !!
}
```

First example : deference of unassigned pointer

Source :

```
int main(int argc, char *argv[])
{
    int *a;
    *a = 0;
}
```

Valgrind's result :

```
Use of uninitialised value of size 4
    at 0x8048307: main (exemple1.c:4)
    by 0x8048264: (within /home/nct/data/articles/fs/valgrind/exemple1)
```

Second example : stack trash

```
int main(int argc, char *argv[])
{
    int array[10];

    array[4] = 0;
    array[-1] = 0;
}
```

Second example : stack trash

Source :

```
int main(int argc, char *argv[])
{
    int array[10];

    array[4] = 0;
    array[-1] = 0; <- we destroy the stack here
}
```

Valgrind's result :

```
Invalid write of size 4
    at 0x804830B: main (example2.c:6)
    by 0x8048264: (within /home/nct/data/articles/fs/valgrind/example2)
    Address 0xBFFFFA6C is just below %esp.  Possibly a bug in GCC/G++
    v 2.96 or 3.0.X.  To suppress, use: --workaround-gcc296-bugs=yes
```

Do not blame gcc here, it's the programmer's fault.

Third example : write on already freed memory

```
int main(int argc, char *argv[])
{
    char *buffer;
    buffer = (char *)malloc(256);
    free(buffer);
    buffer[0] = 0;
}
```

Third example : write on already freed memory

Source :

```
int main(int argc, char *argv[])
{
    char *buffer;
    buffer = (char *)malloc(256);
    free(buffer);
    buffer[0] = 0; <- we just freed this region
}
```

Valgrind's result :

```
Invalid write of size 1
    at 0x8048390: main (example3.c:6)
    by 0x40223A50: __libc_start_main (in /lib/libc-2.3.1.so)
    by 0x80482CC: (within /home/nct/data/articles/fs/valgrind/example3)
    Address 0x40EEC024 is 0 bytes inside a block of size 256 free'd
    at 0x4015D6A4: free (vg_clientfuncs.c:185)
    by 0x8048389: main (example3.c:5)
    by 0x40223A50: __libc_start_main (in /lib/libc-2.3.1.so)
    by 0x80482CC: (within /home/nct/data/articles/fs/valgrind/example3)
```

Fourth example : jump depends on uninitialised value

```
int main(int argc, char *argv[])
{
    int v;
    int i;
    if (v)
    {
        // do something
        i = 0;
    }
    else
    {
        // do something else
        i = 1;
    }
}
```

Fourth example : jump depends on uninitialised value

Source :

```
int main(int argc, char *argv[])
{
    int v;
    int i;
    if (v)
    { ... }
    else
    { ... }
}
```

Valgrind's result :

```
Conditional jump or move depends on uninitialised value(s)
    at 0x8048308: main (example2.c:5)
    by 0x40223A50: __libc_start_main (in /lib/libc-2.3.1.so)
    by 0x8048264: (within /home/nct/data/articles/fs/valgrind/example2)
```

Conclusion

All those four programs work, but they all do nasty things with the memory.

Valgrind can be very usefull, especially with complex projects involving lots of memory structure. For instance, it is actively used by KDE.

To get more informations about valgrind, go to its homepage :
`http://developer.kde.org/~sewardj/`
For sarge and sid :

```
apt-get install valgrind
```

For woody :

```
deb http://cmeerw.org/files/debian woody valgrind
apt-get install valgrind
```

Thanks for your attention, happy hacking ;-)