

„Memory-Debugging“ in C and C++

Hayati Aygün
(Dipl. Informatiker)

eMail:
hayati.ayguen@epost.de

Memory debugging?

- Source of error:
 - Reading „unwanted“ values
 - Writing to „wrong“ addresses
- Effect:
 - The wished values doesn't get evaluated
 - The program code may get modified, what may lead to crashes
- Problem:
 - Source and Effect don't need any implication !
 - Effect seems kind of „random“
- Our Wish: HALT in Debugger

Example 1

```
void Welcome()  
{  
    char acName[10];  
    printf("Please enter your name: ");  
    scanf("%s", acName);  
    printf("Hello %s\n", acName);  
}
```

What's wrong here?

Solution

- Function mostly works without problems
- Entering names with more than 10 characters leads to undefined behaviour
- When entering much bigger names, the function cannot return to its caller.
Cause the return address on the Stack gets overwritten

Example 2

```
int main( int argc, char** argv ) {
    const char *src = "Hello World\n";
    char *dest = (char*)malloc( sizeof(char)
                                * strlen( src ) );

    if (!dest)        return 1;
    strcpy( dest, src );
    return 0;
}
```

And what's wrong here?

Solution

- Error not so obvious this time
- Memory reservation should have length `strlen() + 1`
- This error is very unlikely to show any effect!

Buffer Overflow und similar

Extract from Stack:

<u>Address</u>	<u>Content</u>
0x1000	functions return address
0x9990	int Field[4] ; // local array

Stack grows downwards (local variable addresses of called functions are below those of caller)

so Field[5] == return address !

Effect: in „Nirvana“ after return!

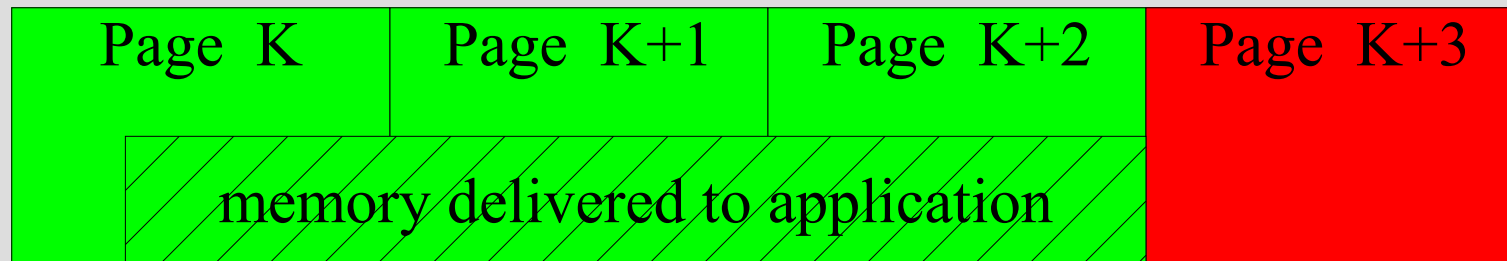
Ways out?

- Using secure high level languages?
No alternative at low resources (CPU, RAM)
- Compuware BoundsChecker (Windows)
Instrumentation at compilation
- valgrind (Linux)
Emulation of the x86 processor
- Electric Fence / EFence (Linux + Windows)
Utilization of Memory Management Unit:
access protection on memory pages

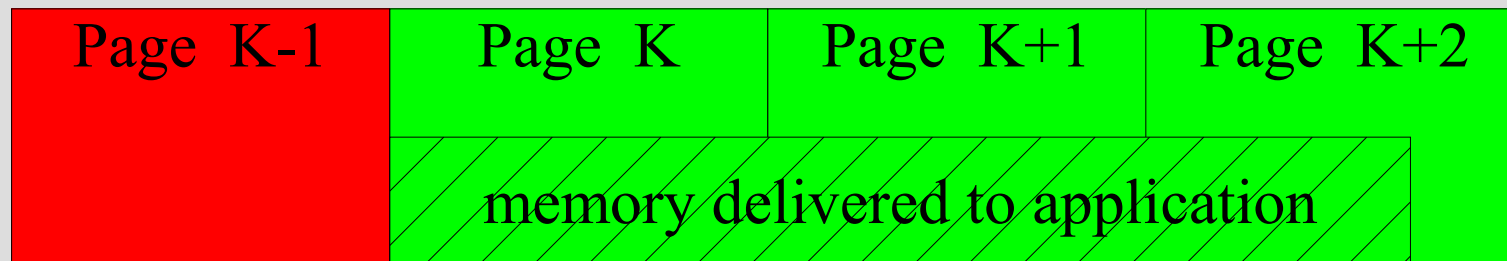
EFence – principles

- „Overloading“ malloc(), free() functions (by link-order) and C++ operators new, delete, ..
- Aligning memory begin/end to a page
- Protection of previous/next page using MMU
- Program halts exactly on command, causing the protection fault :-)
- MMU protects **only** dynamically reserved memory on heap :-)

Memory – Page borders



bzw.



Accessing green pages: O.K., red ones: K.O. !
page-size on intel x86 CPUs: **4096 Bytes**

Further facilities: protection after deallocation

Memory, deallocated with one of following

`free()` / `delete` / `delete[]`

functions may not get freed but protected – as long as there is enough memory

Further facilities: memory leak detection

- Especially at server processes with long runtime: processes with leaks need to get **restarted**
- Unproblematic detection of leaks, cause all allocation/deallocation is handled through E fence.
- Detection with special macros also within program parts, else at program termination
- Report of leaks with filename and line number of allocation

Further facilities: Check of deallocation function

Deallocation method incompatible to allocation method:

- malloc() / calloc() / realloc() / free()
- new / delete
- new[] / delete[]

Mixing these functions is NOT allowed!

Further facilities: memory initialization

- some compilers initialize memory to 0; especially when building debug versions!
 - Errors don't show up in Debug versions
 - Hard to track errors in Release versions :-(
- Initialization to values not equal to 0 (-1) stimulates error effects also in Debug versions

Protection without MMU

Not optimally supported for now :-)

BUT some tricky macros:

```
int aiTest[20], tmp;  
CA_DECLARE(aiTest, 20);  
for ( int j = 0; j < 100; ++j )  
    tmp = CA_REF( aiTest, j );
```

halts at Line CA_REF(aiTest, j) with j = 20

Macros without MMU

```
#define CA_DECLARE( NAME, SIZE )  
announces that pointer to NAME has size SIZE
```

```
#define CA_REF( N, IDX ) \  
N [ assert( (N ## _tmp = IDX) < N ## _size ), N ## _tmp ]  
Reference on index IDX of pointer N with checking
```

- Disatvantage: higher CPU load
- Disatvantage: need to modify existing code
- Advantage: EACH access on memory gets checked exactly – not only at page boundaries

Compilation?

- Utilization of all features require source-code modifications and also recompilation
- With Linux and its LD_PRELOAD mechanism any dynamically linked programs may get started with Efence. That way only the page protection is active.

Linux / Free Software ?!

- Development of Efence Library:
Bruce Perens 1987 – 1999
- Further development enabled by free software license (GPL):
Hayati Aygün 2002 – 2004
- Announcement through <http://freshmeat.net>
- Usage:
Version 2.0 / 2.2 from Bruce Perens is delivered with many Linux – Distributions

Todo's

- C++ conform behaviour of overloaded new/delete/... Operators
 - new_handler, exceptions, ..
- Protection of a functions / methods return address:
 - A C++ Class with Destructor on Stack?
 - Constructor protects one page
 - Destructor unprotects at return
- ???

Download

- Description:
<http://www.pf-lug.de/projekte/haya/efence.php>
- Download via cvs with CVSROOT=
„:pserver:anonymous
@ayguen.homeip.net
:/home/samba-shares/CVS_EFENCE“
und module = „efence“

End

Questions ?