Crash Investigations

Investigating memory corruption issues
https://gitlab.zeuthen.desy.de/ers/crash_investigator

Davit Kalantaryan
DV-ERS
Zeuthen

HELMHOLTZ ASSOCIATION

DESY

Overview

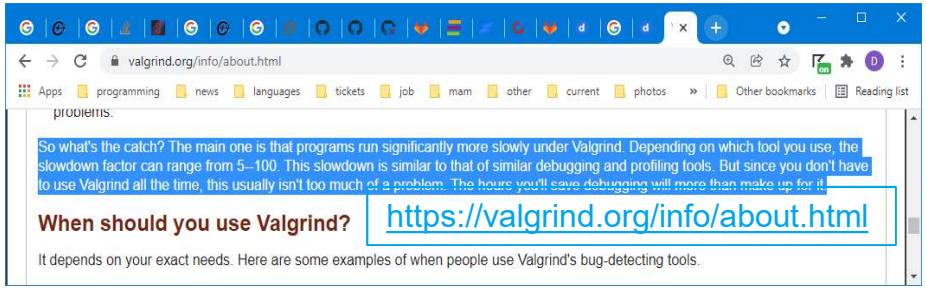
- Memory handling related crashes
- Tools for investigations of memory-related problems
  - valgrind (https://valgrind.org/):
    X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android, X86/Android (4.0 and later), MIPS32/Android, X86/FreeBSD, AMD64/FreeBSD, X86/Darwin and AMD64/Darwin (Mac OS X 10.12).
  - deleaker (https://www.deleaker.com/):
    Windows
- Crash investigator (https://gitlab.zeuthen.desy.de/ers/crash_investigator)
  - Reasons to implement this (cases when Valgrind will not help or will behave poorly)
  - How it is implemented (the idea behind)
  - How to use it
  - How to extend it
- Demo

Memory related issues and crashes

- Meaning of colors in the below list
  - Green: possible to investigate with the newly created crash investigator.
  - Yellow: theoretically possible to investigate but not implemented.
  - Red: not possible to investigate.
  - All cases are possible to investigate by Valgrind (even red cases)!
- Memory leak: this will not lead to a crash but after a long run there will not be available memory for the proper functionality of an application.
- Deallocation of non-existing memory:
- Double free issue: most probably this will immediately crash the application.
- Bad realloc issues: this kind of issue will happen when bad memory (not allocated memory/ already freed memory/ memory created with incompatible to realloc method) is provided as the first argument to the realloc call.
- Allocation - deallocation mismatch: this is the case when memory is allocated using one method and memory is freed by the other method (for example new/free, new[]/delete, etc.). Similar errors will lead to crashes in some cases. Most crashes will happen immediately after corruption.
- Memory overrun issue: this is a very hard detectable issue because they crash applications not immediately and not always. They crash applications when the memory behind allocated memory is responsible for sensitive data. Usually, the crash happens not during the bad call with overrun, but later when the mentioned sensitive memory is accessed.

How Valgrind works. Reasons to have something else

- In the previous slide, there was a note that Valgrind works for all mentioned cases. Then the question arises why do we need something else?
  - It slows down application 5-100 times.
  - It starts application on virtual environment and this can hide some issues that is there because of concurrency.
  - Installation is needed and I'm not sure if we will have permission to install Valgrind on our server hosts.



  - Extendibility: Valgrind mentions the possibility to extend, but I think it is very complicated and one should have very well knowledge of low-level programming (including GNU assembly) for handling Valgrind code. To clone and play with Valgrind code one can use the command: 'git clone git://sourceware.org/git/valgrind.git'.
  - Runtime extendibility !!!
  - Windows!!! (for Windows proof of concept app is done, but complete system is not prepared)
- Valgrind starts applications in the virtual environment where all instructions of applications are trapped by Valgrind, then analyzed and only after that the instructions executed.
  - This is the reason of slowing down the application dramatically.
  - This allows to investigate memory overflow issues by analyzing read/write instructions and by comparing memory size with read/write size in the instruction.

    So if we have suspension that crash is there because of memory overflow Valgrid should be used (or Valgrind also should be used alongside with other tools).

Crash investigator - idea behind

In order to trap memory allocations functions and make some analyze the following can be done

- Overloading C++ operators new (https://en.cppreference.com/w/cpp/memory/new/operator_new) and delete (https://en.cppreference.com/w/cpp/memory/new/operator_delete).
  Sometimes only overwriting of these functions can help to detect problems. If the problem is there because of C functions (malloc and friends), then these overloads will not be helpful.
- Rewriting malloc, calloc, realloc and free.
  In case, if an application has its own malloc and friends function, then the application's functions will be called instead of Glibc functions. This will happen because these functions are defined as weak symbols and any new symbol with the same name(s) will be used instead. This is true only for Linux. For Windows hacking malloc is more complicated and a topic for a separate discussion.
- In order to load the analyzer library to the already compiled process address space without recompilation - the following techniques are used (https://en.wikipedia.org/wiki/DLL_injection)
  - Linux: LD_PRELOAD environment variable
  - Windows: DLL injection
    1. Using WriteProcessMemory+CreateRemoteThread or
    2. Using registry “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs”

Some cases of indirect double/free

Be aware double-free will not happen only in the case when one explicitly calls 2 times to function delete or free. Double free can happen also by calling other APIs that, in turn, will allocate/deallocate buffers!!! Sometimes the problem can be there because of concurrency!!!

- A wrong sequence of calling library functions internally allocating and deallocating buffers (for example fopen/fclose).
- Global buffers allocation/deallocation without proper synchronization.
- Realloc of the same address with the assumption that the old address is not deleted and valid after the realloc call. As you see - here crash can happen even without global visible pointer.

```
13  int main(int, char* [])
14  {
15      void* pMemory = malloc(10);
16      realloc(pMemory,100);
17      realloc(pMemory,200);  // this is a potential double free
18      return 0;
19  }
```

- Working with STL containers should be properly synchronized. I think it is obvious that containers should not be protected by some synchronization mechanism (for example mutex) in the implementation. So the containers should be protected by a developer.

Code will be analyzed during demo

https://gitlab.zeuthen.desy.de/ers/crash_investigator/-/blob/master/src/tests/main_double_free01_test.cpp

main_double_free01_test.cpp # main(int, char *[]): int

```cpp
static void Corruption01(){  // double free
    int* pnValue = new int;
    delete pnValue;
    delete pnValue;
}
static void Corruption02(){  // bad realloc 01 (wrong address)
    void* pnValue = realloc(reinterpret_cast<void*>(0x100),200);
    free(pnValue);
}
static void Corruption03(){  // bad realloc 02 (deallocated address)
    void* pnValue = malloc(200);
    free(pnValue);
    void* pnValueRe = realloc(pnValue,400);  // we have problem here
    free(pnValueRe);
}
static void Corruption04(){  // bad realloc 03 (allocated by new address)
    int* pnValue = new int;
    void* pnValueRe = realloc(pnValue,400);  // we have problem here
    free(pnValueRe);
}
static void Corruption05(){  // dealloc missmatch 01
    int* pnValue = new int[2];
    delete pnValue;
}
static void Corruption06(){  // dealloc missmatch 02
    int* pnValue = static_cast<int*>(malloc(sizeof(int)));
    delete pnValue;
}
static void Corruption07(){  // dealloc missmatch 03
    int* pnValue = new int;
    free(pnValue);
}
static void Corruption08(){  // indirect double free
    FILE* pFile = fopen("/tmp/aaa.txt","w");
    printf("pFile: %p\n",static_cast<void*>(pFile));
    fflush(stdout);
    if(pFile){
        fclose(pFile);
        fclose(pFile);
    }
}
```

callback.hpp #

```cpp
namespace crash_investigator {

enum class FailureAction : uint32_t{
    Unknown,
    MakeAction,
    DoNotMakeActionToPreventCrash,
    ExitApp,
};

enum class FailureType : uint32_t{
    Unknown,
    DoubleFree,
    BadReallocMemNotExist,
    BadReallocDeletedMem,
    FreeMissmatch,
};
```

Exit App is default behavior

Alongside these cases, the tool helps to detect bugs like in the function Corruption08.