iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 6 **Issue:** III **Month of publication:** March 2018

**DOI:** http://doi.org/10.22214/ijraset.2018.3486

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 6 Issue III, March 2018- Available at www.ijraset.com*

# A Novel Approach to Explain the Detection of Memory Errors and Execution on Different Application Using Dr Memory.

Yashaswini J[1], Tripathi Ashish Ashok[2]
[1, 2] *School of computer science and engineering, VIT University, Vellore, India*

***Abstract*: *In a program there are many bugs residing in the source code and one of the most likely bug is the errors related to memory. This memory related errors are identified first and fixed. Memory errors include uninitialized memory, unaddressable memory etc. Memory leaks are considered as an error which exists because of allocation of memory by dynamic programming or run time of program become unreachable. Also, these leaks play an important role in delays of response time and also cause application failure. This paper will show the tool names "Dr Memory "which is a memory debugging tool and also the analysis of results showing different errors getting from this debugging tool. The plus point of the tool is to get the execution and result without disturbing the ROM. It is the first tool which is compatible with both windows and Linux.***
***Keywords: Dr Memory, Memory errors.***

## I. INTRODUCTION

The memory checking tools are there for the identification of type of errors in the program. These errors relate to memory are the main course of the delays in program execution and response time, which results in unacceptable time consumption. The memory related errors are counted as the bugs in the program as this cause the interrupts in execution. Many preventive measures are applied while programming and it is only possible because of symptoms for errors. Some memory related errors can also cause crashes of program while it is in execution. This will make user frustrated as their program get crashed in the middle of the execution and they may face loss of data of the program. We can't rely on the counter measure problem because it will be done only when we set some symptoms of errors. That is why we need some powerful tool which will detect the type of error, the program may face to that also in systematic manner. There are many memory checking tools in the today's date. Tools like valgrind, memcheck [1], purity, Intel parallel Inspector and Insure ++. These are tools which detects the memory related errors like references of freed memory, uninitialized memory, calls which are invalid like calling of free () twice for same memory and also leaks in memory. It is difficult to build this tool as it is not easy to implement and also it has complex codes.

### *A. Three major Challenges faced while Implementing a tool for Memory Related Errors*

1) *Performance:* This challenge is faced all the time while implementing any type of Program. But here this is most important challenge we have to pass through as the tool will execute and request for more process for the scanning of the whole program. This will help us to track the allocation of memory.
2) *Accuracy:* whatever result we are getting form the tool should detect the real errors that should not give false positives and should display only those which are violating errors. If we get the errors which in deed are not violating can cause lot of problems like loss of important information regarding the program. Now to acquire accuracy in tool we have to get the full scanning which will cause slower performance.
3) *Compatibility*: This challenge is most required for the usage of tool as the program should be implemented like that it can execute on low level operating system and also accepts its architecture. There is more demand for Windows operating system compared to UNIX, then also the most popular tool MemCheck [1] runs only on UNIX.

Tools execute their program in user mode because the memory references created by the kernel cannot be scanned by the kernel. Due to this, the tool should scan the application memory, which gets affected by the calling of their individual function. Now, to overcome there should be thorough knowledge of parameters or number of arguments passed in the function call. This is not easy to be done for the Windows operating system, as this is proprietary system and also it contains multiple heap libraries which make it more complex.

This paper shows the memory checking tool *Dr Memory* that overcome the above listed challenges with analysis of the result getting from the execution of memory checking tool.

## II. LITRATURE REVIEW

Nowadays the most extensively used memory checking tool is MemCheck [1], which is built on Valgrind [2] which is a dynamic instrumentation platform. Dr. Memory [3] has some properties such as shadow combination rules and propagation which are almost similar to MemCheck[1]. But MemCheck[1] supports only Linux platforms. It uses JIT compilation technique which includes dynamic recompilation. It detects the memory that you should not access, undefined values, memory leaks, overlapping pointers in memory and related functions, incorrect release of heap memory, etc. Valgrind [2] has tools associated with it like MemCheck [1], Addr sanitiser[4], Massif[4] etc.

### *A. Now We Shall Look Into Some Of The Existing Memory Error Detection Tool*

1) *Leak Tracer [5]:* It only detects the memory leaks. It was designed in competition valgrind [2] and labium.
2) *MemCheck [1]:* Including this, there are many tools in Valgrind. Uninitialized memory [6], memory leaks [7] are detected by this tool.
3) *Windows Leaks Detector [8]:* This tool will scan for the errors from the memory while the application is executing. No interference is done in the application.
4) *Purify [13]:* First marketable tool. Memory leaks and detection of use after free error are detected by this tool. This approach is also used in MemCheck[1] and Dr Memory.
5) *Insure++ [12]:* It supports inserting instrumentations at any place which include prior to the compile time and at link time and at run time. It reports uninitialized memory [6].

Some of the other memory error detecting tools are DUMA [9], Deleaker [10], BCHECK [11], Electric Fence [14], MemWatch [14] etc.

For some of the tools like Leak Tracer, application must free all of its memory prior to exiting, though it have data whose lifetime is process lifetime. Reachability based Leak detection [15] uses scanning of memory to find orphaned memory allocation which can't be accessed. This type of leak detection is used in modern tools, which includes our Dr. Memory too.

## III.TOOL OVERVIEW

A technique used in Dr Memory is memory shadowing. It is used for the tracking of a data of target application. Target application is an application that runs on the Dr Memory tool for the memory related errors. This technique is used to increase a computers speed by using high speed RAM memory in place of slower ROM memory. RAM is about three times as fast as ROM. The concept of memory shadowing is to copy every bite of applications memory along with its meta data from ROM to RAM memory. This will allow us to analyze those data without disturbing the current execution of that application. It can identify the errors from that shadowed data.

### *A. This technique Indicates Three States*

1) *Unaddressable:* Memory which are not accessible by the application.
2) *Uninitialized:* Memory which are not written and unable to read but can be addressable.
3) *Defined:* Memory which are both addressable and defined.

Also, this tool considered the invalid memory pages as unaddressable and those memory which are top of stack are also considered as unaddressable. As shown in the Fig. 1, the memory on the heap which is allocated outside the malloc block is also unaddressable.

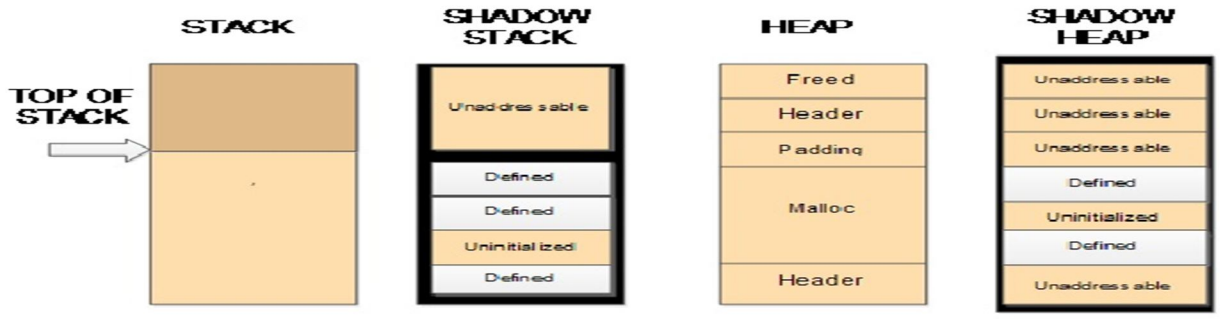



Figure 1: Shadowed memory (stack and heap) of the target application

For a quick explanation of a memory shadowing technique, there are codes to control hardware devices such as keyboards, is normally executed in a special Rom chip called the BIOS ROM. However, this chip is slower than the general-purpose Ram that comprises main memory that is RAM. Therefore, PC's BIOS code is copied into RAM, when the computer boots. This said as shadow RAM.

### *A. Memory Tracking*

It can be said as heap tracking because of steps taken according to windows platform. As there are multiple heap libraries to choose from. Two approaches are taken for the modifications and monitoring in heap library calls.

Wrapping and replacing are those approaches. Wrapping will allow the original code to execute but also adds a prologue and epilogue where the modification of return value and their arguments is allowed. Replacing will use a custom function and will not execute any of the original library function.

The tool MemCheck [1], replaces heap library function and this tool operates on a range of UNIX operating system. But, in Windows there should be wrapping instead of replacing. In wrapping, we have to depend on known subset of API, while in replacing we need to emulate every nuance of the entire API. However, wrapping has advantages over replacing. Wrapping will keep the layout of heap safely and attaching to an application partway through execution, where the application has already allocated heap objects. A heap is an area of preserved computer main storage that program process can use to store data in some variable amount that won't be known until the program is running. A program may accept different amount of input from one or more, processing and then do the processing on all input data at once. This heap are there in windows in multiple libraries.

Along will wrapping there should be transparency that it should not confuse us and also not to crash the program application. This transparent wrapping is not taken by Valgrind and Pin, as they add a frame on the top of the stack.

There are three methods explored and can be used for wrapping functions transparency and robustly. There will be two processes while calling a function pre-hook and post-hook. Pre-hook means before the call and post-hook means after the call. In pre-hook, there will be some address of the function to which it has to go, and we just have to locate the address and invoke at that point. While post-hook is difficult because we have to locate return paths from the function.

Techniques to locate return path share:

*1)* Analyze the code first and build a control flow graph from that entry point. So, through this we can able to get the return paths easily. then also it is not easy to analyze an entire function.

*2)* The second one will keep an eye on the call site as we know the return point at the call instruction. post-hook can be set up after identifying the call target. And the pothook will be set for the invoked function after the call. There are two types of calls: direct calla and indirect calls. In Indirect call, the flow goes through a memory location and target cannot be analyzed statically.

*3)* Dr Memory uses this technique, as this is more dynamic and reactive approach. In this technique after getting inside the target function, return address is obtained and recorded for post-hook when it is reachable late. If the instruction of that address is already executed, then it will be flushed from the cache.

Recursion counter is used to identify in which layer they are operating currently. As in layered heap call, we need to be careful to only modify the arguments for the outer layer.

### *B. Error types Related to dr Memory*

After executing target application by Dr Memory, the reports will be listed with errors and will be created in text file.

The error report includes:

*1)* Error numbers

*2)* Type of error

*3)* Address and size of the memory access

*4)* Location of error where it is detected which is done by thread identifier

*5)* Unaddressable access

Any write or read of a memory location which is not allocated is taken as an "unaddressable access" by Dr Memory. Accessing to a memory address which is invalid is an unaddressable access.

```
Error #1: UNADDRESSABLE ACCESS beyond heap bounds: reading 0x000a720b-0x000a720c 1 byte(s)
# 0 main                              [e:\derek\drmemory\git\src\tests\malloc.c:96]
Note: @0:00:01.500 in thread 3808
Note: refers to 1 byte(s) beyond last valid byte in prior malloc
Note: prev lower malloc:  0x000a7208-0x000a720b
Note: instruction: mov    0x03(%edx) -> %al
```

Figure 2: Occurring of unaddressable access in a heap region providing the target address which is inside a freed region.

*C. Uninitialised read*

As shown in Fig. 3, it is reported when an application reads addressable memory which is not written from the time it was allocated When only a section of region is uninitialized, Dr Memory will report that section region. So that it will be easier to track down the problem. As shown in the Fig.3, it will also show which section of the socket call discovered this error.

```
Error #2: UNINITIALIZED READ: reading 0xffbae108-0xffbae114 12 byte(s) within 0xffbae100-0xffbae114
Elapsed time = 0:00:00.214 in thread 19298
system call socketcall setsockopt args
    <system call>
0x08049a65 <my-socket-test+0x1a65> my-socket-test!main
    ??:0
0x0092dbb6 <libc.so.6+0x16bb6> libc.so.6<nosyms>!__libc_start_main
    ??:0
0x080489b1 <my-socket-test+0x9b1> my-socket-test!_start
    ??:0
```

Figure 3: Example of uninitialized read which is occurred when structures or buffers passed to system calls

*D. Invalid heap Arguments*

This error is considered when a pointer which is not pointing to any valid address and that pointer is passed to free( ) as shown in Fig.4.

```
Error #3: INVALID HEAP ARGUMENT: free 0x00001234
Elapsed time = 0:00:00.180 in thread 21848
# 0 malloc!main                                  [/home/bruening/drmemory/git/src/tests/malloc.c:164]
# 1 libc.so.6!__libc_start_main                  [/build/buildd/eglibc-2.11.1/csu/libc-start.c:226]
# 2 malloc!_start
```

Figure 4: Problem arise immediately as an address 0x00001234 is an invalid heap address

*E. Memory Leas*

Scanning of leaks is classified in three categories by Dr Memory:

*1)* The application that can reach the memory: This is actually not considered as leak. Many of the application do not free the memory after use whose lifetime is same as the process lifetime. Therefore, it is not reflected as error by Dr Memory.

*2)* The application that can defiantly reach the memory: This defines a leak by Dr Memory. Here there is no way to free the memory as it lost all its references.

*3)* The application which reaches the memory only through pointers to the mid of allocation: This is possible leak in Dr Memory.

Further categorizing, there will be two types of leaks: Direct and Indirect leaks. An Indirect leak is an object of heap which is accessible by pointers to its beginning address, but these leaks are initiated in leaked objects. As shown in Fig.5, Dr. Memory will also reports number of leaks, possible leaks and accessible allocation along with call stack. Leaks and possible leak will be listed in the report by default. But reachable allocation are not shown by default to reduce the overhead.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       5 unique,     5 total,    574 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~       5 unique,     8 total,    205 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
```

Figure 5: Example lines from the output result where errors are ignored

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 6 Issue III, March 2018- Available at www.ijraset.com*

If use –show reachable option, we can able to see still reachable allocations as shown in Fig.6.

```
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       5 unique,     5 total,    574 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~       5 unique,     8 total,    205 byte(s) of still-reachable allocation(s)
~~Dr.M~~ NO ERRORS IGNORED
```

Figure 6: Example of output result where no errors are ignored.

```
Error #129: GDI USAGE ERROR: DC 0x180119d5 that contains selected object being deleted
# 0 system call NtGdiDeleteObjectApp
# 1 GDI32.dll!DeleteDC                          +0x150    (0x759f9fe1 <GDI32.dll+0x9fe1>)
# 2 GDI32.dll!DeleteDC                          +0xd      (0x759f9e9e <GDI32.dll+0x9e9e>)
# 3 MFC80U.DLL!?                                +0x0      (0x71966e2e <MFC80U.DLL+0x56e2e>)
# 4 Edraw.exe!?                                 +0x0      (0x0041d156 <Edraw.exe+0x1d156>)
# 5 Edraw.exe!?                                 +0x0      (0x0041de49 <Edraw.exe+0x1de49>)
```

Figure 7: GD error found in target application.

*F. Warning*

In this type, we are not getting any errors, but we will be warned for condition which has unusual memory relation. This will be helpful for an application developer to know about the unusual memory related conditions. For example, as shown in Fig.8.

```
Error #8: WARNING: heap allocation failed
@0:00:01.500 in thread 3748
# 0 suppress.exe!warning_test1                  [e:\derek\drmemory\git\src\tests\suppress.c:179]
# 1 suppress.exe!test                           [e:\derek\drmemory\git\src\tests\suppress.c:282]
# 2 suppress.exe!main                           [e:\derek\drmemory\git\src\tests\suppress.c:297]
# 3 suppress.exe!__tmainCRTStartup              [f:\sp\vctools\crt_bld\self_x86\crt\src\crt0.c:327]
# 4 KERNEL32.dll!BaseProcessStart
```

Figure 8: Example for warning message of the target application

## IV. EXPERIMENTAL ANALYSIS

*A. Example1.c*

We have executed 3 different programs containing C source code. These three programs have different errors as shown further in the paper.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
      char *x = malloc(128*sizeof(char));
      char *y = malloc(128*sizeof(char));
      y = x;
      free(x);
      free(y);
      return 0;
}
```

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887*
*Volume 6 Issue III, March 2018- Available at www.ijraset.com*

*B. Output Result*

```
Dr. Memory version 1.10.1 build 3 built on Apr 10 2016 18:05:55
Dr. Memory results for pid 5972: "example1.exe"
Application cmdline: "C:\Users\-DIEGO\Desktop\programs\example1.exe"
Recorded 115 suppression(s) from default C:\Program Files (x86)\Dr. Memory\bin64\suppress-default.txt

Error #1: INVALID HEAP ARGUMENT to free 0x0000000000fa0320
# 0 replace_free                [d:\drmemory_package\common\alloc_replace.c:2706]
# 1 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 2 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 3 main                        [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
Note: @0:00:01.568 in thread 1948
Note: memory was previously freed here:
Note: # 0 replace_free                [d:\drmemory_package\common\alloc_replace.c:2706]
Note: # 1 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
Note: # 2 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
Note: # 3 main                        [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
```

1) *Description:* The above Example 1.c can be interpreted in the following way. Here, in the code there are two pointers x and y. Then the pointer x is assigned to pointer y (i.e y will get the address of x). After freeing those pointers then also the address of b will not be the same as it is getting the address of a. This leads to Invalid Heap Argument error. This is one of the error which is faced run time, there will be no compiling error for this. This code will lead to dangling pointer. Likewise, Dr. Memory can be applied to the other applications too.

*C. Example 2.c*

```
#include <stdlib.h>

int main()

{
    char *y = malloc(9);
    y [9] = 'a';
    return 0;
}
```

```
Dr. Memory version 1.10.1 build 3 built on Apr 10 2016 18:05:55
Dr. Memory results for pid 4204: "example2.exe"
Application cmdline: "C:\Users\-DIEGO\Desktop\example2.exe"
Recorded 115 suppression(s) from default C:\Program Files (x86)\Dr. Memory\bin64\suppress-default.txt

Error #1: UNADDRESSABLE ACCESS beyond heap bounds: writing 0x0000000001040329-0x000000000104032a 1 byte(s)
# 0 main                [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
Note: @0:00:01.533 in thread 5236
Note: refers to 0 byte(s) beyond last valid byte in prior malloc
Note: prev lower malloc:  0x0000000001040320-0x0000000001040329
Note: allocated here:
Note: # 0 replace_malloc                [d:\drmemory_package\common\alloc_replace.c:2576]
Note: # 1 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
Note: # 2 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
Note: # 3 mingw_onexit                 [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/atonexit.c:42]
Note: # 4 main                        [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
Note: instruction: mov    $0x61 -> (%rax)

Error #2: LEAK 9 direct bytes 0x0000000001040320-0x0000000001040329 + 0 indirect bytes
# 0 replace_malloc              [d:\drmemory_package\common\alloc_replace.c:2576]
# 1 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 2 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 3 mingw_onexit                 [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/atonexit.c:42]
# 4 main                         [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
```

Description: The above Example 2.c can be interpreted in the following way. The above result will tell us that the pointer y we have declared has allotted a space for 9 bytes as the char data type is of 1 byte. And we are giving a write operation to that memory which is outside the range. Therefore, we are getting the unaddressable access error. Also we will be warned for the invalid read of the variable y. This leads to leak in memory. We will also get the process id and thread which was used for execution. Likewise, Dr. Memory can be applied to the other applications too.

*D. Example3.c*

```
#include <stdlib.h>

int main ()

{
    char *i = malloc (100);
    return 0;
}
```

*1) Output result*

```
Dr. Memory version 1.10.1 build 3 built on Apr 10 2016 18:05:55
Dr. Memory results for pid 5164: "example3.exe"
Application cmdline: "C:\Users\-DIEGO\Desktop\programs\example3.exe"
Recorded 115 suppression(s) from default C:\Program Files (x86)\Dr. Memory\bin64\suppress-default.txt

Error #1: LEAK 100 direct bytes 0x0000000000fd0320-0x0000000000fd0384 + 0 indirect bytes
# 0 replace_malloc                [d:\drmemory_package\common\alloc_replace.c:2576]
# 1 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 2 ntdll.dll!RtlEncodePointer    +0x26     (0x00007ffb23013d07 <ntdll.dll+0x63d07>)
# 3 mingw_onexit                  [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/atonexit.c:42]
# 4 main                          [C:/crossdev/src/mingw-w64-v4-git/mingw-w64-crt/crt/crtexe.c:218]
```

*2) Description*

The above Example 3.c can be interpreted in the following way. We can clearly see that memory leak is there and it is caused by a call of malloc in main. Memory leaks don't cause any outward problems until our memory is full and our call for malloc fails. That is why memory leaks are the most difficult bugs to find. The number of mallocs differ from the number of frees then it will cause the leak in memory, so it should not differ. Likewise, Dr Memory can be applied to the other applications too.

It can be summarized from the three examples that section 4.1 describes "Invalid heap memory", section 4.2 describes "unaddressable access error" and section 4.3 describes "Memory leak caused by malloc function". From the given three examples we can interpret some of the errors detected by the memory tool. Those three examples are on C language and we can analyze that the target application should be in the .exe format. The examples and description are given for their individual error result and also the error type. Still more errors can be detected by this tool as discussed in section 3.2. This tool can be applied to many Target applications to detect memory related errors which can be used for debugging those errors. And these errors will help developers to prevent or solve further in their target application.

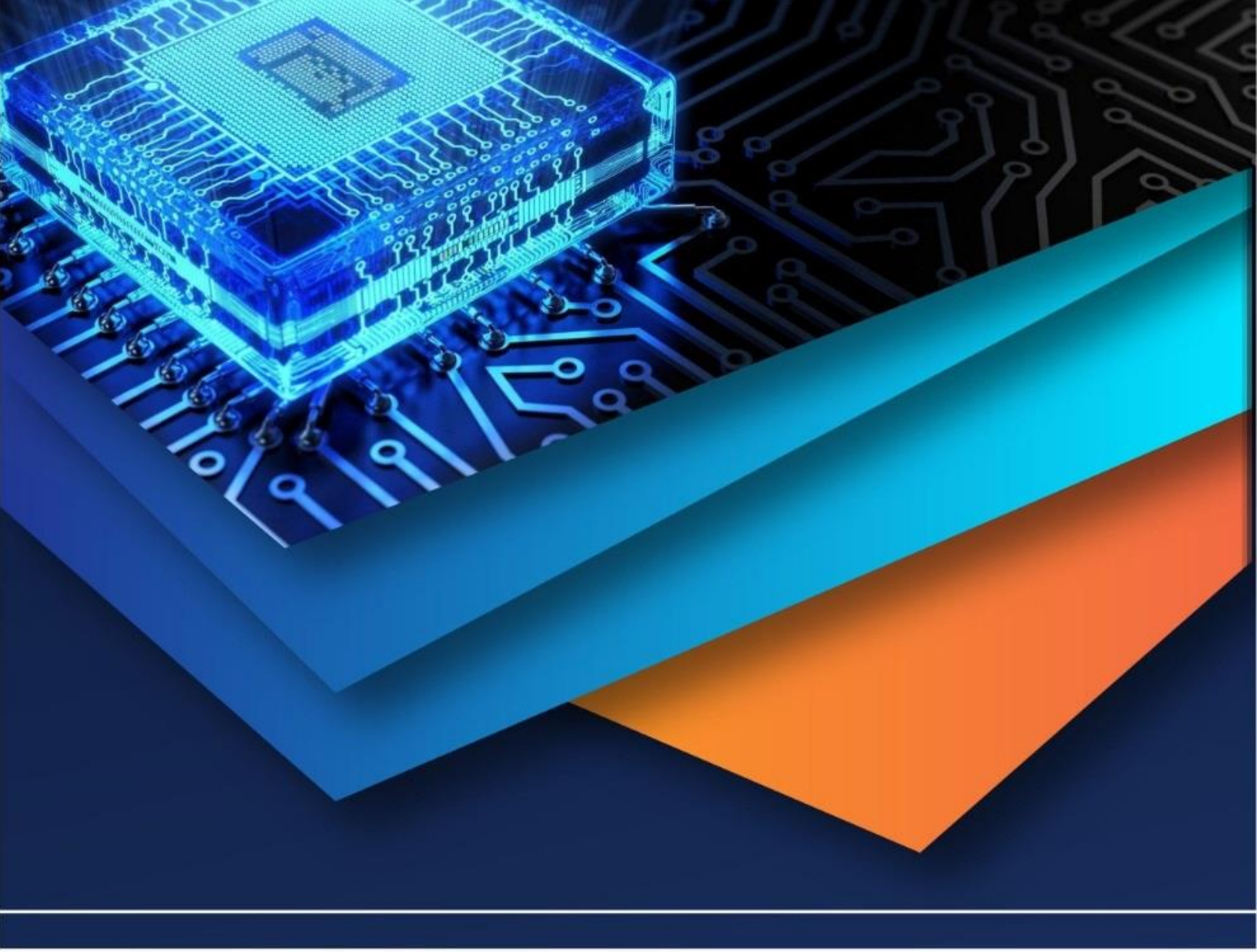
## V. CONCLUSION AND FUTURE WORK

This paper concludes the techniques used in the Dr Memory regarding the types of errors described in detail. Also executed different target application on Dr Memory which also varies with time consumption along with the type of errors. This Memory debugging tool is compatible on Windows and Unix platform. The output result in this paper is getting on the Windows platform, hence we are getting the GDI errors, and this GDI is errors are detected in Windows API.

As this tool is open source many future project ideas are using this memory debugging tool. We can able to know the thread and the process used last wrote a memory value which is used in debugging and many other applications. In some cases, multiple process creation is there in a target application but some of the user might want a single report per process instead of getting multiple reports, so post processing can be added in the existing code. We can create exception for the required error that we know it's going to work it can be made such that .c or .sql files are taken as input directly. Dr. Memory identifies the line the leak occurred, but not the function call that returned the leaky pointer. This can be overcome by extending this tool in future. The error is not specifying from which function call error has occurred, the next outcome of the tool can overcome occur but we don't want that in the report, so annotation feature can be added which is not yet existing in Dr Memory. Dr. Memory takes only .exe files only as input, therefore as a part of future this loop hole.

## REFERENCES

[1] John Gyllenhaal, "Using 's MemCheck tool to find memory errors and leaks in MPI and serial applications on LINUX", https://computing.llnl.gov/code/memcheck/
[2] Valgrind TM Developers, "MemCheck a memory error detector," http://valgrind.org/docs/manual/mc manual.html
[3] Dr.Memory,"
[4] Florent Bruneau , "Memory part 5 – debugging_tools,"https://techtalk.intersec.com/2013/12/memory-part-5-debugging-tools/, pp.
[5] Leak Tracer,trace and analyse memory leaks in c++ programs," http://www.andreasen.org/LeakTracer/2003.
[6] "Reading uninitialized memory", http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.ide.userguide%2Ftopic%2Fmemory_uninit_mem_read_.html.
[7] "What are memory Leaks, " , https://msdn.microsoft.com/en-us/library/ms859408.aspx
[8] Sourceforge net production ,"Windows leak Detector" , http://winleak.sourceforge.net/
[9] "DUMA" ,http://duma.sourceforge.net/README.txt
[10] "DELEAKER" , http://www.deleaker.com/docs/introduction.html
[11] "BCHECK " , http://docs.oracle.com/cd/E18659_01/html/821-1380/blaio.html
[12] Parasoft, "Insure++" , http://www.parasoft.com/jsp/products/insure.jsp?itemId=63.
[13] R. Hastings and B. Joyce, "Purify: Fast detection of memory leaks and access errors" , in Proc. of the Winter USENIX Conference, Jan. 1992,pp. 125–136.
[14] "Memory debuggers for Linux coding," http://www.computerworld.com/article/3003957/linux/review-5-memory-debuggers-forlinux-coding.html ,pp. 1-2.
[15] "Reachability", https://gist.github.com/darkseed/1182373.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY