GRAYHAT

McAfee

Fuzzing and finding vulnerabilities using AFL/WinAFL

Hardik Shah

@hardik05

McAfee

Fuzzing with WinAFL

- What it is?
- Instrumentation with DynamoRIO
- Fuzzing Strategies
- Using WinAFL
  - Hands on: Compile Sample C program using Visual Studio.
  - Hands on: Find Offset of Fuzz Function.
  - Hands on: Run Winafl in debug mode to check everything is working fine.
  - Hands on: Fuzz using WinAFL.
  - Hands on: Analyzing the crashes and finding root cause.
- Fuzzing real world programs.
  - Hands on: Write a harness/test program to read/parse MDB files.
  - Hands on: Analyze MDB files.
  - Hands on: Fuzz it using WinAFL.
- CVE-2018-8423 and CVE-2019-0576 Analysis
- Conclusion

What it is?

- Challenges in Fuzzing closed source programs
  - No source code, only dll, exes
    - Gdi32.dll,gdiplus.dll, msrd3x40.dll
- WinAFL
  - Windows port of AFL(American Fuzzy Lop)
  - Maintained by Ivan Fratric.
  - Uses **DynamoRIO**(drrun.exe) for instrumentation/code coverage
  - afl-fuzz.exe communicates with winafl.dll which act as a client for drrun.exe
  - Need to write a program called harness.
    - Sample program which calls the functions from the DLLs which handles file parsing.
    - Harness should have a function which accept the file name as input, opens it and then do further processing.
    - winafl will use this function for in memory fuzzing.
- Sample Fuzz function.-→

```
int FuzzMe(TCHAR* name)
{
	CoInitialize(NULL);
	_DBEnginePtr dbe;
	HRESULT hr = dbe.CreateInstance("DAO.DBEngine.120");
	if (hr==0x0)
	{
		DatabasePtr db = dbe->OpenDatabase(name);
		db->Close();
		db = NULL;
		dbe = NULL;
	}
	else
	{
		printf ("not able to createInstance,%0X", hr);
	}
	CoUninitialize();
	return 0;
}
```

McAfee

DynamoRIO + WinAFL

Instrumentation using DynamoRIO

- afl-fuzz.exe is the process responsible for fuzzing as on linux.
- winafl.dll is a client dll which handles the instrumentation from dynamorio, pre_fuzz, post_fuzz handlers etc.
- Both communicates over IPC.
- Registers even callbacks for various events like basic blocks, module load, unload etc.
- Checks it module is target module, then search for the fuzz function using symbol/address.
- Call pre_fuzz handler to save the state
- Execute the function and monitor for the crashes, update coverage map.
- Call post_fuzz handler and restore the stage.
- Loop till number of iterations.

```
drmgr_init();
drx_init();
drreg_init(&ops);
drwrap_init();

options_init(id, argc, argv);

dr_register_exit_event(event_exit);

drmgr_register_exception_event(onexception);

if(options.coverage_kind == COVERAGE_BB) {
    drmgr_register_bb_instrumentation_event(NULL, instrument_bb_coverage, NULL);
} else if(options.coverage_kind == COVERAGE_EDGE) {
    drmgr_register_bb_instrumentation_event(NULL, instrument_edge_coverage, NULL);
}

drmgr_register_module_load_event(event_module_load);
```

```
if(options.fuzz_module[0]) {
    if(strcmp(module_name, options.fuzz_module) == 0) {
        if(options.fuzz_offset) {
            to_wrap = info->start + options.fuzz_offset;
        } else {
            //first try exported symbols
            to_wrap = (app_pc)dr_get_proc_address(info->handle, options.fuzz_method);
            if(!to_wrap) {
                //if that fails, try with the symbol access library
                drsym_init(0);
                drsym_lookup_symbol(info->full_path, options.fuzz_method, (size_t *)(&to_wrap), 0)
                drsym_exit();
                DR_ASSERT_MSG(to_wrap, "Can't find specified method in fuzz module");
                to_wrap += (size_t)info->start;
```

McAfee

Fuzzing Strategies

Bitflip – flips a bit i.e. 1 becomes 0, 0 becomes 1

- 1/1,2/1,4/1,8/8 ….32/8

Byte Flip – flips a byte

Arithmetic –random arithmetic like plus/minus

Havoc – random things with bit/bytes/addition/subtraction

Dictionary – user provided dictionary or auto discovered tokens.

- Over/insert/over(autodetected)

Interest - replace content in original file with interesting values

- 0xff,0x7f etc – 8/8,16/8..

Splice – split and combine two files to get a new file.

Ref: https://github.com/googleprojectzero/winafl/blob/master/afl_docs/technical_details.txt

What we have learned So far?

What is WinAFL?
How it works?
Fuzzing strategies

McAfee

Hands On

Hands on: Lets Fuzz Simple C program with WinAFL

- **Compiling:**
  - Compiling Vulnerable C program using visual studio/use precompiled binaries.
- **Enable Page Heap:**
  - Enable: **gflags /p /enable readfile.exe**
  - Verify: **gflags /p**
- **Fuzz Function**:
  - Finding the offset of the ProcessImage Function.
    - Use WinDBG
    - x ReadFile!ProcessImage
- **Making sure program is getting instrumented properly:**
    - C:\fuzzingwork\DynamoRIO-Windows-7.1.0-1\bin32\drrun.exe -c winafl.dll -debug -target_module readfile.exe -target_offset 0x10a0 -fuzz_iterations 10 -nargs 1 – readfile.exe 1.img

Drrun.exe -> DynamoRIO Binary.
-c winafl.dll -> Client for drrun.
-debug -> Winafl will generate debug log.

-target_module -> Name of the module which will be instrumented.
-target_offset -> Offset of the fuzz function for in memory fuzzing.
-fuzz_iterations -> Loop counter for fuzz function.
-nargs -> Number of argument function expects.

McAfee

Hands on: Running WinAFL and Fuzzing

Command line syntax

AFL-FUZZ.EXE AFL_ARGS -- INSTRUMENTATION_OPTIONS -- PROGRAM_NAME @@

```
afl-fuzz.exe -i in -o Out -t 5000+ -D C:\Fuzzing\DynamoRIO-Windows-7.1.0-1\bin32 -- -coverage_module readfile.exe -target_module readfile.exe -target_offset 0x10a0 -fuzz_iterations 5000 -call_convention thiscall -nargs 1 -covtype edge -- readfile.exe @@
```

1. afl-fuzz.exe arguments

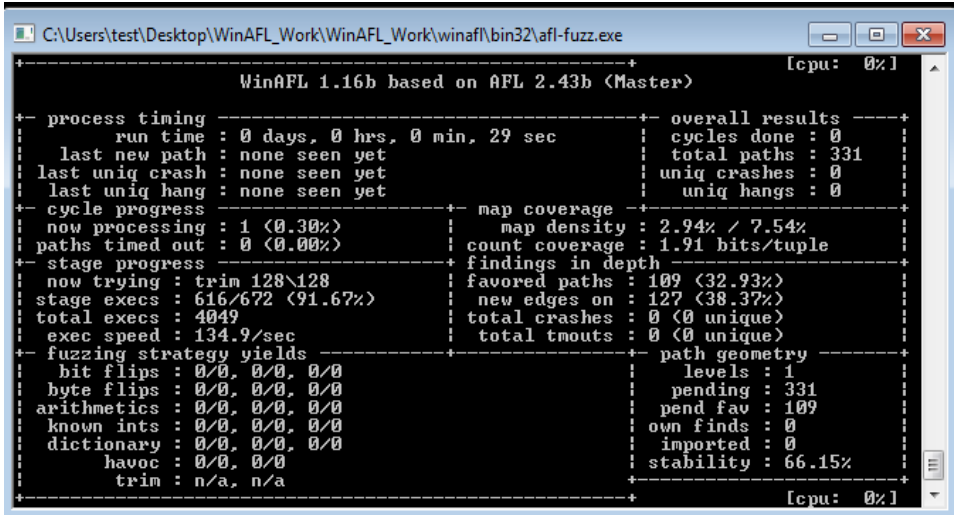
-M – Master Instance
-S – Slave instance
-i -> input directory
-o -> output directory
-D -> DynamoRIO bin32/64 directory path
-t -> timeout

3. Program arguments followed by @@

2. DynamoRIO Instrumentation options

-coverage_module -> name of dll/exe which needs to be instrumented.
-target_module -> Name of the harness executable.
-target_method or –target_offset -> Function name of offset which process input files.
-call_convention – Function calling convention i.e. thiscall,stdcall,cdecl,fastcall etc.

Hands on: WinAFL Status Screen



Ref: https://github.com/google/AFL/blob/master/docs/status_screen.txt

McAfee

Crash Analysis

Hands on: Analyzing the crashes and finding root cause.

- Using Visual Studio
  - Debugging the source code with crash file and tracking the execution flow, finding root cause.
- Using windbg
  - Debugging the executable with crash file and finding root cause.

What we have learned So far?

How to make sure everything is working properly for winafl.
How to fuzz a simple program using winafl.
How to do crash analysis on windows to find root cause.

Fuzzing Real World Software

McAfee

Fuzzing MDB Files.

- **Harness to fuzz MDB file.**
  - Uses CDaoDatabase object.
  - Open MDB file.
  - Close it.
  - Parsing code resides in msrd3x40.dll
  - db.open will call functions from this dll.
  - If malformed structure,no proper checks than our program will crash.
  - Resulted in 13 CVEs.
- **Fuzzing**
  - Compile the c code.
  - Collect some MDB files from google.
  - Fuzz it using winafl.

```
#include <afx.h>
#include <afxdao.h>

int FuzzMe(TCHAR* name)
{
    CDaoDatabase db;
    //printf("parsing file:%s", (char*)name);

    try
    {
        if (name != NULL) {
            db.Open(name);
            db.Close();
            return(0);
        }

    }
    return(0);
}
int _tmain(int argc, TCHAR* argv[])
{
    if (argc < 2) {
        printf("Usage: %s <DB file>\n", (char *)argv[0]);
        return 0;
    }
    FuzzMe(argv[1]);
    return 0;
}
```

McAfee

CVE-2018-8423

- Ref: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8423

CVE-2018-8423 | Microsoft JET Database Engine Remote Code Execution Vulnerability

Security Vulnerability

Published: 10/09/2018
MITRE CVE-2018-8423

A remote code execution vulnerability exists in the Microsoft JET Database Engine.

An attacker who successfully exploited this vulnerability could take control of an affected system. An attacker could then install programs; view, change, or delete data; or create new accounts with full user rights. Users whose accounts are configured to have fewer user rights on the system could be less impacted than users who operate with administrative user rights.

To exploit the vulnerability, a user must open/import a specially crafted Microsoft JET Database Engine file. In an email attack scenario, an attacker could exploit the vulnerability by sending a specially crafted file to the user, and then convince the user to open the file.

The security update addresses the vulnerability by modifying how the Microsoft JET Database Engine handles objects in memory.

- Out of Bound Write Vulnerability.
- Vulnerability in Processing of MDB Files.
- MDB file are database files, used by MS Access.
  - Contains database structure like table, fields, indexes.
  - Old and proprietary file format, but many people have documented it over the years:
    - Ref: http://jabakobob.net/mdb/
    - Follow a page structure.
      - First Page, Table Defination Page, Data Page, OLE Fields. Table Properties
      - Makes it an easy fuzz target.

CVE-2018-8423 Analysis

PoC Availble : https://github.com/thezdi/PoC/tree/master/ZDI-18-1075
Contains two files-> Group1 [MDB file], poc.js [js file]
McAfee analysis - https://www.mcafee.com/blogs/other-blogs/mcafee-labs/jet-database-engine-flaw-may-lead-to-exploitation-analyzing-cve-2018-8423/

```
02 01 - table definition page identifier.
56 43 - VC
00 00 00 00 - next page
C4 02 00 00 - Table Definition Length
10 00 00 00 - Number of rows
00 00 00 00 - Autonumber
53 - Table Type / Flags? ==> system table
11 00 - Next Column Id
0B 00 - Variable Columns
11 00 - Column Count
02 00 00 00 - Index Count
02 00 00 00 - Real Index Count
00 06 00 00 - Row Page Map
01 06 00 00 - Free Space Page Map
```

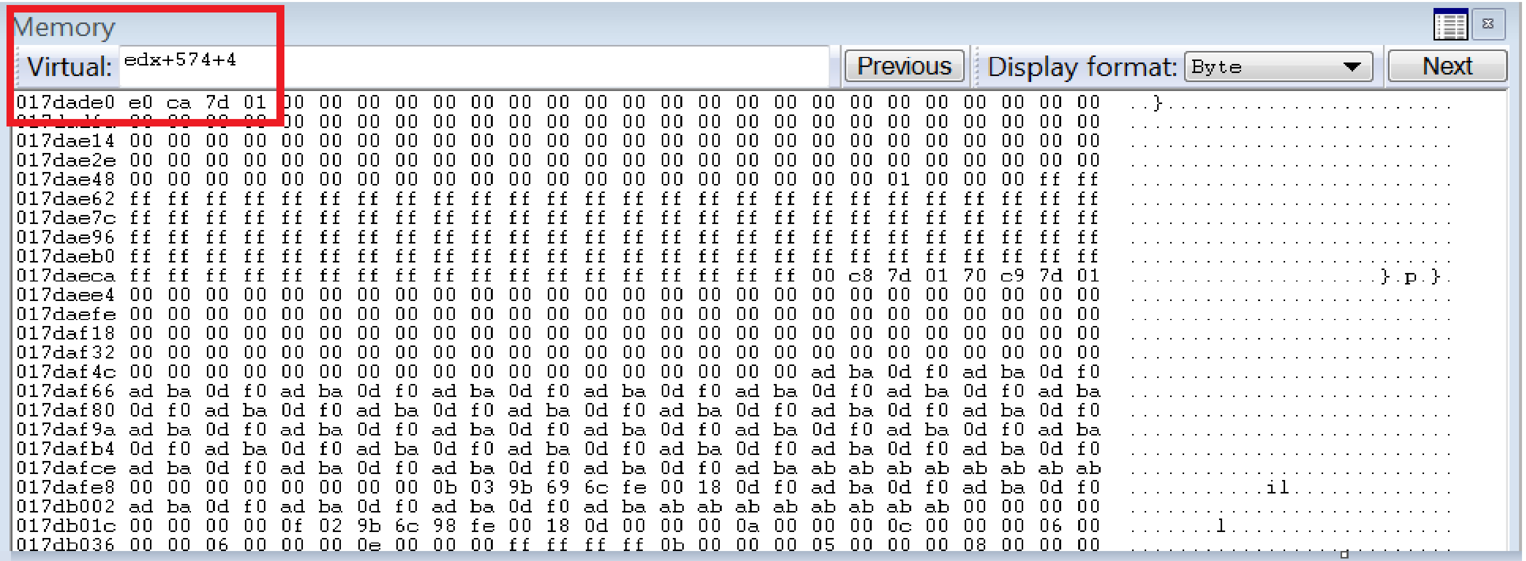
```
for every index (including those that aren't real):
index1:
01 00 00 00 - Index Number
01 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
01 - index_type

index2:
00 23 00 00 - Index Number
00 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
00 - index_type
```

```
Iterate for the number of num_idx
Index1
    02 - Length
    Id - Name(of size Length)
Index2
    0C - Length
    ParentIdName - Name(of size Length)
```

```
0:000> g
(bf0.aa8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01dcae60 ebx=06fb427c ecx=00002300 edx=01dca868 esi=01dccbf0 edi=00000001
eip=68f5f745 esp=001fcff8 ebp=06fb4258 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
msrd3x40!TblPage::CreateIndexes+0x175:
68f5f745 89b48a74050000  mov     dword ptr [edx+ecx*4+574h],esi ds:0023:01dd39dc=????????
```

McAfee

CVE-2018-8423 Analysis



ECX =1 , EDX+574 = 17DADDC start of the memory location for 32*4 = 128 bytes..

EDX+574+4xECX = address of First Index Name which is "Id", ECX should be 1 here.

ECX= 00002300

EDX+574+4xECX = address of second index name which is "ParentId", since 0x000002300 is too big(0x2300x4=0x8c00), it can not come inside allotted memory. It will try to write at memory location which is out of bound of the allocated buffer. This was the root cause of cve-2018-8423.

McAfee™

So it is fixed, right?

```
6a621115 0f8dab010000    jge     msrd3x40!TblPage::CreateIndexes+0x226 (6a6212c6)
6a62111b 85c9            test    ecx,ecx
6a62111d 0f88a3010000    js      msrd3x40!TblPage::CreateIndexes+0x226 (6a6212c6)
6a621123 8b08            mov     ecx,dword ptr [eax]
6a621125 81f9ff000000    cmp     ecx,0FFh
6a62112b 0f8f95010000    jg      msrd3x40!TblPage::CreateIndexes+0x226 (6a6212c6) [br=1]
6a621131 85c9            test    ecx,ecx
6a621133 0f888d010000    js      msrd3x40!TblPage::CreateIndexes+0x226 (6a6212c6)
6a621139 43              inc     ebx
6a62113a 83c014          add     eax,14h
```

```
0:000>
eax=06ce4258 ebx=00000001 ecx=00002300 edx=00000002 esi=00000002 edi=0019d724
eip=6a62112b esp=0019d598 ebp=06ce4214 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
msrd3x40!TblPage::CreateIndexes+0x8b:
6a62112b 0f8f95010000    jg      msrd3x40!TblPage::CreateIndexes+0x226 (6a6212c6) [br=1
```

McAfee

No, we found another issue!!, CVE-2019-0576

```
703de306 8dabf4050000    lea     ebp,[ebx+5F4h]
703de30c 8d642400        lea     esp,[esp]
703de313 8b8c8374050000  mov     ecx,dword ptr [ebx+eax*4+574h] ds:0023:01ac7834=????????
703de31a 3b7104          cmp     esi,dword ptr [ecx+4]
703de31f 8b09            mov     ecx,dword ptr [ecx]

0:000> g
(14b0.15f8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0055b7a8 ebx=00559420 ecx=703a2772 edx=0000000c esi=00000018 edi=00000000
eip=703de313 esp=0012d2bc ebp=00559a14 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
msrd3x40!Table::FindIndexFromName+0x43:
703de313 8b8c8374050000  mov     ecx,dword ptr [ebx+eax*4+574h] ds:0023:01ac7834=????????
```

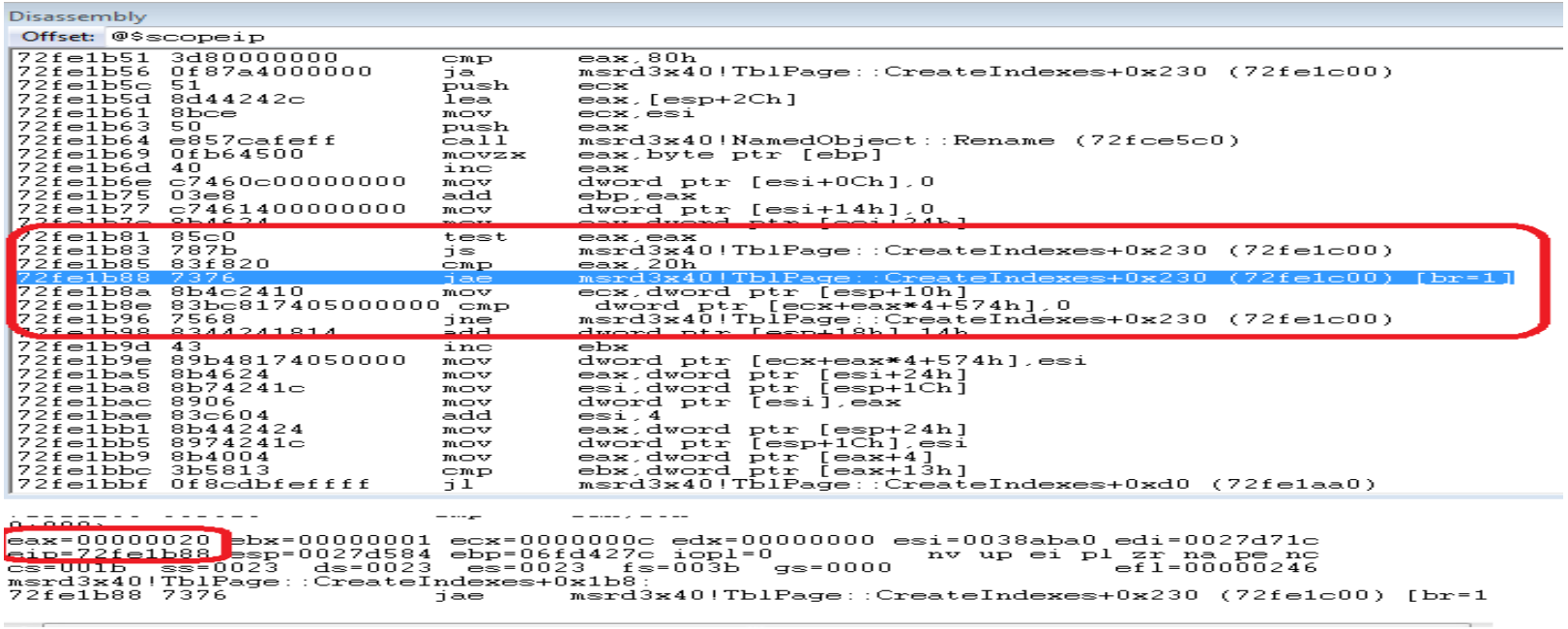
- Index number starts from 0.
- Memory is allocated for only 32 indexes[32x4=128 bytes].
- So index number can onlybe 0-31.
- If index number is 0x20 or 32, it will corrupt data at 128+4 bytes.
  This was the root cause of CVE-2019-0576.

Ref: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/analyzing-and-identifying-issues-with-the-microsoft-patch-for-cve-2018-8423/

```
for every index (including those that aren't real):
index1:
01 00 00 00 - Index Number
01 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
01 - index_type

index2:
20 00 00 00 - Index Number
00 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
00 - index_type

Iterate for the number of num_idx
Index1
    02 - Length
    Id - Name(of size Length)
Index2
    0C - Length
    ParentIdName - Name(of size Length)
```

McAfee

How it was finally fixed?



Crash triaging on windows.

- Manually its very difficult to analyze each and every crash.
- Automate using python + winappdbg.
- Write your own tools.
- Use bugid by skylined.
  - https://bugid.skylined.nl/

What we have learned So far?

How to fuzz real life program on windows?
cve-2018-8423/cve-2019-0576 root cause analysis.
Crash triaging on windows.

Conclusion

- Fuzzing is helpful in overall understanding of software security.
  - Helps to write better code.
  - Can help to find issues which can not be found using normal testing.
  - Helps in securing software.
  - Part of software development life cycle.
- Requires lot of hard work
  - Broken and non working stuff
  - Countless hours analyzing issues and fixing them
  - Multiple vendor follow-ups
  - rejections

But in the end its worth it ☺

McAfee

GRAYHAT

McAfee