CS6008: CRYPTOGRAPHY & NETWORK SECURITY
ASSIGNMENT 1: MODULE 1
FINDING PASSWORDS EXECUTABLES USING GDB

NAME : ANUSREE V
REG NO : 2019103507
BATCH : R
DATE : 05-03-2022

AIM: Finding the password stored in an Executable binary file using GDB

TOOLS INVOLVED:

- GDB
- GCC
- Windows OS

PROBLEM DESCRIPTION:

A binary code where the user entered password is compared with an actual password given which has to be debugged and disassembled through gdb. And then by making use of various gdb commands which give access to memory locations, stack contents etc, we must try to find what the actual password is and then use the actual password to get access.

INPUT:

Input is the executable in binary form.

OUTPUT:

Trying to debug the binary code through gdb and finding what the right password is.

SCREENSHOTS:

➔ Program:

```c
#include<stdio.h>
#include<string.h>

void checkPassword(char[]);
int main(){
    char user_psw[100];
    printf("Enter password\n\n");
    fgets(user_psw,99,stdin);
    user_psw[strlen(user_psw)-1]='\0';
    checkPassword(user_psw);
}

void checkPassword(char user_psw[]){
    char password[] = "anusreev";
    if(strcmp(user_psw,password)==0){
        printf("Correct password. Authentication successful\n\n");
    }
    else{
        printf("Wrong password. Authentication failed\n\n");
    }
}
```

The program code for eg.c takes in user input for the password, performs verification against the actual password and prints whether the authentication passes or fails. As seen here, the correct password for user authentication is “anusreev”.

➔ The executable is opened in debugging mode with the GDB – GNU Debugger. This is to reverse engineer the code and attempt to find the password.

➔ Info functions - Prints all the functions currently loaded for the program:

```
PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto> gcc -o eg eg.c
PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto> gdb eg -q
Reading symbols from C:\Users\Anusree\Documents\Sem\Sem 6\Crypto\eg.exe...done.
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x00000000  __deregister_frame_info
0x00000000  __register_frame_info
0x00401290  _mingw32_init_mainargs
0x004012d0  mainCRTStartup
0x004012f0  WinMainCRTStartup
0x00401310  atexit
0x00401320  _onexit
0x00401330  __gcc_register_frame
0x004013e0  __gcc_deregister_frame
0x00401410  main
0x0040146e  checkPassword
0x004014d0  _setargv
0x00401870  __cpu_features_init
0x00401980  __do_global_dtors
0x004019c0  __do_global_ctors
0x00401a20  __main
0x00401a90  __dyn_tls_init@12
0x00401b40  __tlregdtor
0x00401bb0  ___w64_mingwthr_add_key_dtor
0x00401c30  ___w64_mingwthr_remove_key_dtor
0x00401cc0  __mingw_TLScallback
0x00401ec0  _pei386_runtime_relocator
0x004020b0  fesetenv
0x00402130  __mingw_aligned_free
0x00402130  __mingw_free
0x00403100  __mingw_glob
0x00403200  __mingw_globfree
0x00403260  __mingw_dirname
0x00403830  __mingw_opendir
0x00403a30  __mingw_readdir
0x00403a80  __mingw_closedir
0x00403ad0  __mingw_rewinddir
0x00403b30  __mingw_telldir
0x00403b60  __mingw_seekdir
0x00403be0  __mingw_memalign_base
0x00403ca0  __mingw_realloc
0x00403d20  __mingw_memalign_realloc
0x00403e2c  stricoll
```

```
0x00403e34  strdup
0x00403e3c  wcstombs
0x00403e44  vfprintf
0x00403e4c  tolower
0x00403e54  strncmp
0x00403e5c  strlen
0x00403f44  GetProcAddress@8
0x00403f4c  GetModuleHandleA@4
0x00403f54  GetLastError@0
0x00403f5c  GetCommandLineA@0
0x00403f64  FreeLibrary@4
0x00403f6c  FindNextFileA@8
0x00403f74  FindFirstFileA@8
0x00403f7c  FindClose@4
0x00403f84  ExitProcess@4
0x00403f8c  EnterCriticalSection@4
0x00403f94  DeleteCriticalSection@4
0x00403fa0  register_frame_ctor
0x00403fb0  _CTOR_LIST__
0x00403fb0  __CTOR_LIST__
0x00403fbc  _DTOR_LIST__
0x00403fbc  __DTOR_LIST__
(gdb)
```

➔ Disassembling the main function

```
(gdb) disas main
Dump of assembler code for function main:
   0x00401410 <+0>:     push   %ebp
   0x00401411 <+1>:     mov    %esp,%ebp
   0x00401413 <+3>:     and    $0xfffffff0,%esp
   0x00401416 <+6>:     add    $0xffffff80,%esp
   0x00401419 <+9>:     call   0x401a20 <__main>
   0x0040141e <+14>:    movl   $0x405044,(%esp)
   0x00401425 <+21>:    call   0x403e7c <puts>
   0x0040142a <+26>:    mov    0x4081b4,%eax
   0x0040142f <+31>:    mov    %eax,0x8(%esp)
   0x00401433 <+35>:    movl   $0x63,0x4(%esp)
   0x0040143b <+43>:    lea    0x1c(%esp),%eax
   0x0040143f <+47>:    mov    %eax,(%esp)
   0x00401442 <+50>:    call   0x403eac <fgets>
   0x00401447 <+55>:    lea    0x1c(%esp),%eax
   0x0040144b <+59>:    mov    %eax,(%esp)
   0x0040144e <+62>:    call   0x403e5c <strlen>
   0x00401453 <+67>:    sub    $0x1,%eax
   0x00401456 <+70>:    movb   $0x0,0x1c(%esp,%eax,1)
   0x0040145b <+75>:    lea    0x1c(%esp),%eax
   0x0040145f <+79>:    mov    %eax,(%esp)
   0x00401462 <+82>:    call   0x40146e <checkPassword>
   0x00401467 <+87>:    mov    $0x0,%eax
   0x0040146c <+92>:    leave
   0x0040146d <+93>:    ret
End of assembler dump.
(gdb)
```

➔ Disassembling checkPassword function

```
(gdb) disas checkPassword
Dump of assembler code for function checkPassword:
   0x0040146e <+0>:     push   %ebp
   0x0040146f <+1>:     mov    %esp,%ebp
   0x00401471 <+3>:     sub    $0x28,%esp
   0x00401474 <+6>:     movl   $0x616d6548,-0x15(%ebp)
   0x0040147b <+13>:    movl   $0x3332315f,-0x11(%ebp)
   0x00401482 <+20>:    movl   $0x3738395f,-0xd(%ebp)
   0x00401489 <+27>:    movb   $0x0,-0x9(%ebp)
   0x0040148d <+31>:    mov    0x8(%ebp),%eax
   0x00401490 <+34>:    mov    %eax,(%esp)
   0x00401493 <+37>:    call   0x403e5c <strlen>
   0x00401498 <+42>:    mov    %eax,0x8(%esp)
   0x0040149c <+46>:    lea    -0x15(%ebp),%eax
   0x0040149f <+49>:    mov    %eax,0x4(%esp)
   0x004014a3 <+53>:    mov    0x8(%ebp),%eax
   0x004014a6 <+56>:    mov    %eax,(%esp)
   0x004014a9 <+59>:    call   0x403e54 <strncmp>
   0x004014ae <+64>:    test   %eax,%eax
   0x004014b0 <+66>:    jne    0x4014c0 <checkPassword+82>
   0x004014b2 <+68>:    movl   $0x405054,(%esp)
   0x004014b9 <+75>:    call   0x403e7c <puts>
   0x004014be <+80>:    jmp    0x4014cc <checkPassword+94>
   0x004014c0 <+82>:    movl   $0x405084,(%esp)
   0x004014c7 <+89>:    call   0x403e7c <puts>
   0x004014cc <+94>:    nop
   0x004014cd <+95>:    leave
   0x004014ce <+96>:    ret
   0x004014cf <+97>:    nop
End of assembler dump.
(gdb)
```

➔ Disassembling strcmp function

```
(gdb) disas strcmp
Dump of assembler code for function strcmp:
   0x00403e54 <+0>:     jmp    *0x4081fc
   0x00403e5a <+6>:     nop
   0x00403e5b <+7>:     nop
End of assembler dump.
(gdb)
```

➔ Breakpoint is now placed in the call instruction to strcmp (0x403e54) to halt the execution of the program at that point and study the state and content of the registers.

```
(gdb) break strcmp
Breakpoint 1 at 0x403e54
(gdb)
```

➔ The code runs till the first breakpoint which is strcmp function

```
(gdb) run
Starting program: C:\Users\Anusree\Documents\Sem\Sem 6\Crypto/eg.exe
[New Thread 22100.0x43b0]
[New Thread 22100.0x26e0]
Enter password

anusreev

Breakpoint 1, 0x00403e54 in strcmp ()
(gdb)
```

➔ Info about registers and their values until the breakpoint

```
(gdb) info registers
eax            0x61febc 6422204
ecx            0x61febc 6422204
edx            0x7efeff09       2130640649
ebx            0x3ea000 4104192
esp            0x61fe6c 0x61fe6c
ebp            0x61fe98 0x61fe98
esi            0x4012d0 4199120
edi            0x4012d0 4199120
eip            0x403e54 0x403e54 <strcmp>
eflags         0x202    [ IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x53     83
gs             0x2b     43
(gdb)
```

➔ Trying to read the values present in the registers:

After reaching breakpoint 1, the program halts in the instruction which performs a call to strcmp. strcmp is the current function being executed which means the arguments of strcmp must be on top of the stack. The arguments of strcmp must be the passwords being compared and hence accessing them

would help to find the password. The command ‘info registers’ provides information about the state of the registers at that point in program execution. The esp – stack pointer register is to be inspected.

To view the content of the addresses in string form, x/s is used.

```
(gdb) x/s 0x61febc
0x61febc:       "anusreev"
(gdb) x/s 0x7efefeff
0x7efefeff:     <Address 0x7efefeff out of bounds>
(gdb) x/s 0x261000
0x261000:       ""
(gdb) x/s 0x61fe6c
0x61fe6c:       "§\024@"
(gdb) x/s 0x61fe98
0x61fe98:       "(ÿa"
(gdb) x/s 0x4012d0
0x4012d0 <mainCRTStartup>:      "ƒì\034Ç\004$\001"
```

```
(gdb) x/s 0x403e54
0x403e54 <strcmp>:      "ÿ%ü\201@"
(gdb)
```

- ➔ Printing the stack content:
  Now the ‘x’ command is used to examine the memory.

x/20s $esp displays the top 20 chunks in the top of the stack by accessing the stack pointer register $esp.

```
(gdb) x/20s $esp
0x61fe6c:       "\230\024@"
0x61fe70:       "¾_a"
0x61fe74:       "╡_a"
0x61fe78:       "Æ_a"
0x61fe7c:       ""
0x61fe7d:       ""
0x61fe7e:       ""
0x61fe7f:       ""
0x61fe80:       "X_a"
0x61fe84:       "\001"
0x61fe86:       ""
0x61fe87:       "anusreev"
0x61fe90:       "\fk'H_ÿÿÿ(ÿa"
0x61fe9c:       "g\024@"
0x61fea0:       "¾_a"
0x61fea4:       "Y"
0x61fea6:       ""
0x61fea7:       ""
0x61fea8:       ""
0x61fea9:       "vRvt(µ"
(gdb)
```

➔ 2nd run:

```
(gdb) break strcmp
Breakpoint 1 at 0x403e54
(gdb) run
Starting program: C:\Users\Anusree\Documents\Sem\Sem 6\Crypto/eg.exe
[New Thread 9604.0x357c]
[New Thread 9604.0x1b84]
Enter password

anusree

Breakpoint 1, 0x00403e54 in strcmp ()
(gdb) info registers
eax            0x61febc 6422204
ecx            0x61febc 6422204
edx            0x7efefeff       2130640639
ebx            0x3aa000 3842048
esp            0x61fe6c 0x61fe6c
ebp            0x61fe98 0x61fe98
esi            0x4012d0 4199120
edi            0x4012d0 4199120
eip            0x403e54 0x403e54 <strcmp>
eflags         0x202    [ IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x53     83
gs             0x2b     43
(gdb)
```

```
(gdb) x/20s $esp
0x61fe6c:       "§\024@"
0x61fe70:       "¾_a"
0x61fe74:       "╡_a"
0x61fe78:       "\b"
0x61fe7a:       ""
0x61fe7b:       ""
0x61fe7c:       ""
0x61fe7d:       ""
0x61fe7e:       ""
0x61fe7f:       ""
0x61fe80:       "X_a"
0x61fe84:       "\001"
0x61fe86:       ""
0x61fe87:       "anusreev"
0x61fe90:       "Bƒu\v_ÿÿÿ(ÿa"
0x61fe9c:       "g\024@"
0x61fea0:       "¾_a"
0x61fea4:       "Z"
0x61fea6:       ""
0x61fea7:       ""
(gdb)
```

Since, the string “anusreev” (actual password) is present at the same address (0x61fe87) in both the runs, it could be the possible password.

➔ Checking if that's the right password

```
PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto> ./eg
Enter password

anusreev
Correct password. Authentication successful

PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto>
```

➔ For incorrect password:

```
PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto> ./eg
Enter password

anu
Wrong password. Authentication failed

PS C:\Users\Anusree\Documents\Sem\Sem 6\Crypto>
```