

Kulgram, Basic Debugging dengan GDB

kali ini saya sedikit sharing hal yang agak basic sebenarnya, karena kulgram2 sebelumnya materinya agak sepertinya advance bagi pemula. saya akan sharing kulgram tentang basic debugging dengan GDB

Reverse engineering itu soal membongkar dan menganalisa program. Ada 2 metode analisa yang dapat dilakukan, static analisis dan dynamic analysis Static analisis adalah menganalisa program tanpa mengeksekusi program. Dalam reverse engineering, static analisa dilakukan dengan membaca hasil disassembly, memahami struktur dari program, bisa juga dengan membaca source code (jika memang dimiliki) itu termasuk static analisis.

sementara dynamic analisis adalah menganalisa program dengan mengeksekusinya.

kita akan melihat kelakuan suatu program selagi dia berjalan, dengan begini kita bisa tau apa yang dia lakukan sebenarnya. Program yang di obfuscate akan sulit dilakukan static analisis, program yang diobfuscate sudah diacak sedemikian rupa sehingga sulit dibaca (dengan static analis). Dengan dynamic analisis kita bisa membaca perilaku program tersebut, misalnya sebuah fungsi yang diobfuscate bisa kita pahami dengan cara membaca parameter dan return value dari fungsinya saja

hal yang paling umum dalam melakukan dynamic analisis yaitu dengan teknik debugging, tool yang digunakan adalah debugger

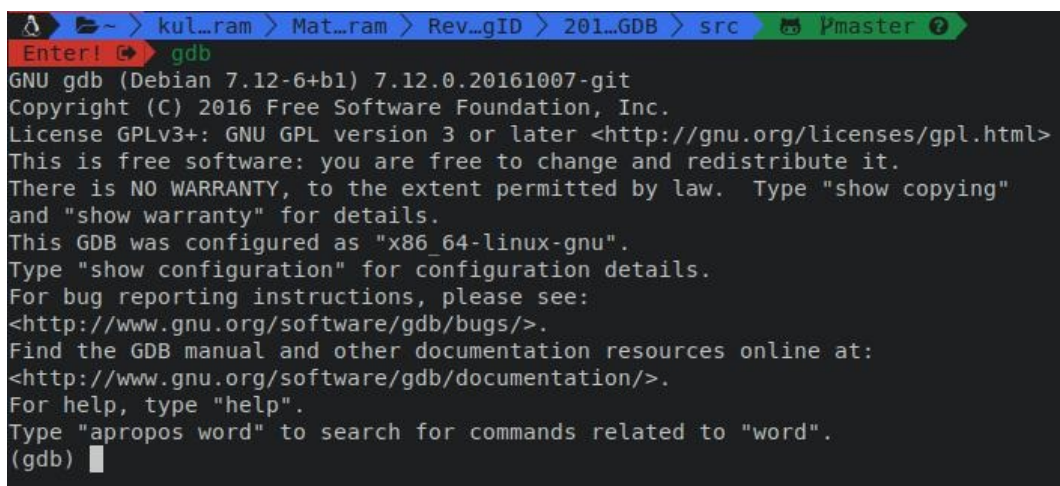
dalam reverse engineering, debugging merupakan salah satu hal yang penting. Dengan debugging kita dapat melihat perilaku program yang sedang berjalan, melihat kondisi program pada waktu tertentu, mempause program, melihat status register dll.

ada banyak macam tool yang bisa digunakan dalam debugging, salah satunya adalah gdb. GDB merupakan tool debugging yang umum digunakan di system UNIX, tapi masih banyak sebenarnya yang lebih baik dari gdb. disini saya akan membahas gdb karena gdb adalah tool yang sering saya gunakan untuk debugging dan juga saya udah terbiasa dengan perintah gdb. gdb tetap powerfull untuk debugging. mungkin saya akan membahas tool debugger lainnya dilain waktu

Oke, sekarang cek gdbnya udah terinstall atau belum, bisa cek nya melalui command `gdb --help`, jika belum terinstall gunakan command `sudo apt-get install libc6-dbg gdb` untuk menginstallnya.

nanti juga kita akan menggunakan gcc untuk compile program, cek gccnya dengan `gcc --help`, untuk nginstallnya `sudo apt-get install build-essential`

jika kalian menjalankan perintah gdb, maka akan outputnya akan seperti ini, dan kita masuk ke prompt gdb



```
Enter! > kulram > Matram > Rev_gID > 201_GDB > src > Pmaster >
gdb
GNU gdb (Debian 7.12-6+b1) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

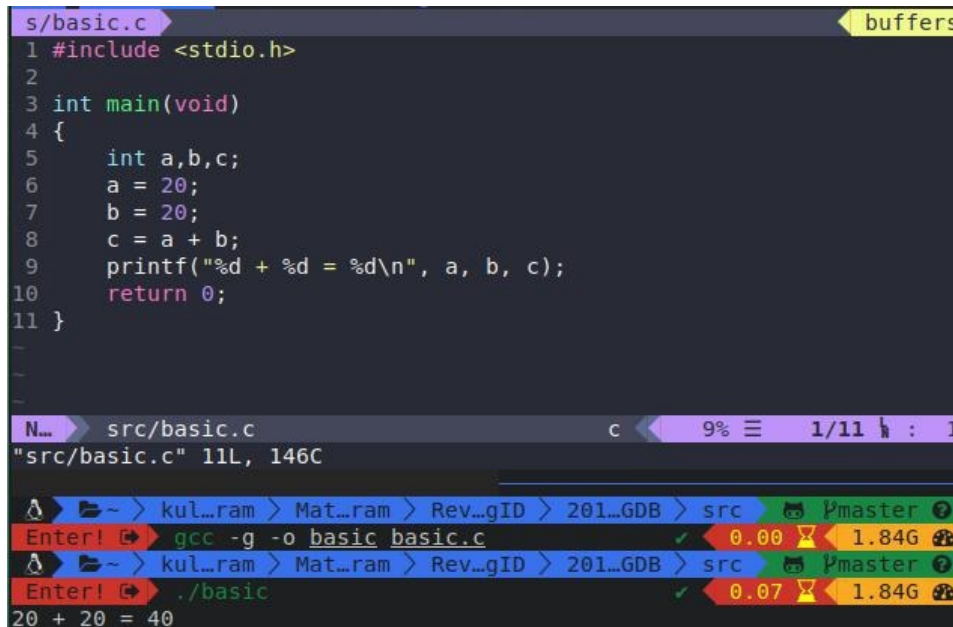
untuk permulaan, saya mempunyai simple program dengan bahasa c, yang kita jadikan sebagai program yang akan kita debug

seperti ini script sederhana kita,

```
#include <stdio.h>

int main(void)
{
    int a,b,c;
    a = 20;
    b = 20;
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

simpan dengan `namafile.c` , compile dengan `gcc -g -o namafile namafile.c` *namafile bisa diganti apapun



```
s/basic.c buffers
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a,b,c;
6     a = 20;
7     b = 20;
8     c = a + b;
9     printf("%d + %d = %d\n", a, b, c);
10    return 0;
11 }

N... src/basic.c c 9% 1/11 : 1
"src/basic.c" 11L, 146C

Enter! > gcc -g -o basic basic.c ✓ 0.00 1.84G
Enter! > ./basic ✓ 0.07 1.84G
20 + 20 = 40
```

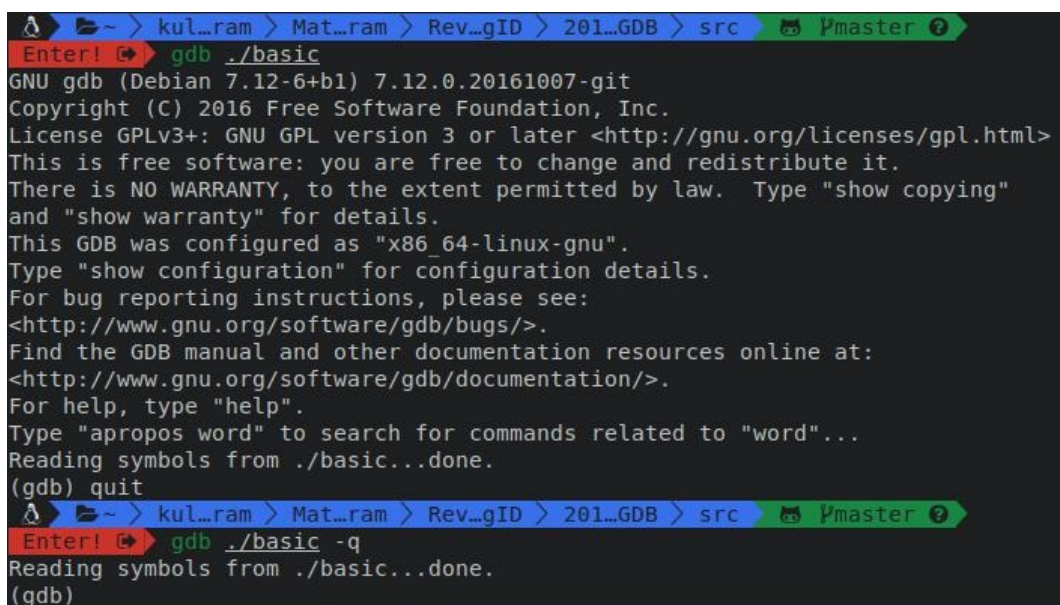
opsi -g pada gcc berfungsi untuk memberikan informasi debugging pada program

[File : basic]

[File : basic.c]


untuk yang males compile 🙄 bisa langsung download binary dan sourcenya. dan juga supaya programnya sama dengan yang saya punya, mungkin beberapa versi compiler akan ngehasilin binary yang berbeda kalo dilihat di dari disassemblynya

load program basic diatas ke gdb, dengan perintah `gdb ./basic`



```
Enter! > gdb ./basic
GNU gdb (Debian 7.12-6+b1) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./basic...done.
(gdb) quit
Enter! > gdb ./basic -q
Reading symbols from ./basic...done.
(gdb)
```

disini saya pakai perintah `gdb ./basic -q` supaya nggak nampilin output seperti perintah sebelumnya

```
kul...ram > Mat...ram > Rev...gID > 201...GDB > src > Pmaster ?
Enter!  gdb ./basic -q
Reading symbols from ./basic...done.
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

jika kalian ketik perintah `help`, maka akan banyak pilihan disana, `gdb` sangat terdokumentasi, jadi gak perlu takut hehe

```
(gdb) list main
1      #include <stdio.h>
2
3      int main(void)
4      {
5          int a,b,c;
6          a = 20;
7          b = 20;
8          c = a + b;
9          printf("%d + %d = %d\n", a, b, c);
10         return 0;
(gdb) █
```

perintah `list main` akan menampilkan source code dari fungsi `main`, perintah ini akan bekerja kalo pada saat dicompile menambahkan opsi `-g` pada `gcc`nya seperti yang dikatakan sebelumnya

dalam reverse engineering, mungkin binary tidak dicompile dengan opsi `-g`. jadi kita harus melihat disassemblynya. perintah `disassemble main` akan menampilkan hasil disassembly untuk fungsi `main`. perintah bisa disingkat menjadi `disas main`

```
(gdb) disas main
Dump of assembler code for function main:
0x00000000004004d7 <+0>:      push    %rbp
0x00000000004004d8 <+1>:      mov     %rsp,%rbp
0x00000000004004db <+4>:      sub     $0x10,%rsp
0x00000000004004df <+8>:      movl    $0x14,-0x4(%rbp)
0x00000000004004e6 <+15>:     movl    $0x14,-0x8(%rbp)
0x00000000004004ed <+22>:     mov     -0x4(%rbp),%edx
0x00000000004004f0 <+25>:     mov     -0x8(%rbp),%eax
0x00000000004004f3 <+28>:     add     %edx,%eax
0x00000000004004f5 <+30>:     mov     %eax,-0xc(%rbp)
0x00000000004004f8 <+33>:     mov     -0xc(%rbp),%ecx
0x00000000004004fb <+36>:     mov     -0x8(%rbp),%edx
0x00000000004004fe <+39>:     mov     -0x4(%rbp),%eax
0x0000000000400501 <+42>:     mov     %eax,%esi
0x0000000000400503 <+44>:     lea     0x9a(%rip),%rdi        # 0x4005a4
0x000000000040050a <+51>:     mov     $0x0,%eax
0x000000000040050f <+56>:     callq   0x4003f0 <printf@plt>
0x0000000000400514 <+61>:     mov     $0x0,%eax
0x0000000000400519 <+66>:     leaveq  %eax
0x000000000040051a <+67>:     retq
End of assembler dump.
(gdb)
```

untuk yang belum pernah liat assembly, mungkin keliatannya aneh wkww

secara default, gdb akan menampilkan assembly dengan syntax AT&T (syntax assembly ada 2, syntax intel dan AT&T). untuk mengubahnya ke syntax intel bisa menggunakan perintah `set disassembly-flavor intel`. kalo saya terbiasa dengan syntax intel

```
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
0x00000000004004d7 <+0>:    push    rbp
0x00000000004004d8 <+1>:    mov     rbp, rsp
0x00000000004004db <+4>:    sub     rsp, 0x10
0x00000000004004df <+8>:    mov     DWORD PTR [rbp-0x4], 0x14
0x00000000004004e6 <+15>:   mov     DWORD PTR [rbp-0x8], 0x14
0x00000000004004ed <+22>:   mov     edx, DWORD PTR [rbp-0x4]
0x00000000004004f0 <+25>:   mov     eax, DWORD PTR [rbp-0x8]
0x00000000004004f3 <+28>:   add     eax, edx
0x00000000004004f5 <+30>:   mov     DWORD PTR [rbp-0xc], eax
0x00000000004004f8 <+33>:   mov     ecx, DWORD PTR [rbp-0xc]
0x00000000004004fb <+36>:   mov     edx, DWORD PTR [rbp-0x8]
0x00000000004004fe <+39>:   mov     eax, DWORD PTR [rbp-0x4]
0x0000000000400501 <+42>:   mov     esi, eax
0x0000000000400503 <+44>:   lea     rdi, [rip+0x9a]          # 0x4005a4
0x000000000040050a <+51>:   mov     eax, 0x0
0x000000000040050f <+56>:   call    0x4003f0 <printf@plt>
0x0000000000400514 <+61>:   mov     eax, 0x0
0x0000000000400519 <+66>:   leave
0x000000000040051a <+67>:   ret
End of assembler dump.
(gdb)
```

untuk yang pengen lebih tau tentang assembly, bisa dicek disini

<http://www.ilmuhacking.com/programming/belajar-assembly-di-linux/> dan disini

<https://github.com/d4em0n/tutorial-assembly>

untuk menjalankan program, gunakan perintah `run` bisa disingkat `r` aja

```
(gdb) run
Starting program: /home/ramdhan/kulgram/Materi-Kulgram/ReversingID/2017
1216 - Basic Debugging dengan GDB/src/basic
20 + 20 = 40
[Inferior 1 (process 24268) exited normally]
(gdb)
```

sekarang, kita akan mencoba breakpoint, breakpoint arti sederhana yaitu mempause program

```
(gdb) list main
1      #include <stdio.h>
2
3      int main(void)
4      {
5          int a,b,c;
6          a = 20;
7          b = 20;
8          c = a + b;
9          printf("%d + %d = %d\n", a, b, c);
10         return 0;
(gdb)
```

kita akan breakpoint pada baris 8, breakpoint di gdb menggunakan perintah `b nomor_baris` atau `b* alamat_intruksi`

```
(gdb) b 8
Breakpoint 1 at 0x4004ed: file basic.c, line 8.
(gdb) r
Starting program: /home/ramdhan/kulgram/Materi-Kulgram/ReversingID/2017
1216 - Basic Debugging dengan GDB/src/basic

Breakpoint 1, main () at basic.c:8
8          c = a + b;
(gdb)
```

setelah perintah breakpoint, lalu kita jalankan dengan perintah `run` atau `r`

proses program yang berjalan akan berhenti dibaris 8, ketika break kita bisa melakukan apapun program, contohnya menampilkan variable dan mengeset variable

```
(gdb) print a
$3 = 20
(gdb) print b
$4 = 20
```

untuk mengubah nilai variable, kita bisa menggunakan perintah set

kita akan mengubah variable a menjadi 50

```
(gdb) set variable a=50
```

untuk melanjutkan proses, kita bisa menggunakan perintah continue atau c

```
(gdb) set variable a=50
(gdb) c
Continuing.
50 + 20 = 70
[Inferior 1 (process 25806) exited normally]
(gdb)
```

variable a sudah berubah menjadi 50

contoh lain, masih dengan script yang sederhana

```
basic2.c
1 #include <stdio.h>
2
3 int add(int a, int b)
4 {
5     return a+b;
6 }
7
8 int main(int argc, char *argv[])
9 {
10     int x;
11     x = add(10, 20);
12     printf("%d\n", x);
13     return 0;
14 }
```

[File : basic2]

[File : basic2.c]

cara compilenya masih sama dengan sebelumnya gcc -g -o namafile namafile.c

load programnya ke gdb. kita bisa melihat source code program seperti yang dipraktekan sebelumnya

```
Enter! ➤ gdb ./basic2 -q
Reading symbols from ./basic2...done.
(gdb) list add
1     #include <stdio.h>
2
3     int add(int a, int b)
4     {
5         return a+b;
6     }
7
8     int main(int argc, char *argv[])
9     {
10        int x;
(gdb) list main
4     {
5         return a+b;
6     }
7
8     int main(int argc, char *argv[])
9     {
10        int x;
11        x = add(10, 20);
12        printf("%d\n", x);
13        return 0;
(gdb)
```

kita akan breakpoint sebelum fungsi add dipanggil. sebelumnya kita breakpoint dengan memasukkan nomor

baris sebagai argumen. sekarang kita bisa melakukan breakpoint menggunakan alamat intruksi sebagai argumen, nantinya program akan berhenti pada alamat intruksi tersebut

```
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000004004eb <+0>:    push    rbp
0x0000000004004ec <+1>:    mov     rbp, rsp
0x0000000004004ef <+4>:    sub     rsp, 0x20
0x0000000004004f3 <+8>:    mov     DWORD PTR [rbp-0x14], edi
0x0000000004004f6 <+11>:   mov     QWORD PTR [rbp-0x20], rsi
0x0000000004004fa <+15>:   mov     esi, 0x14
0x0000000004004ff <+20>:   mov     edi, 0xa
0x000000000400504 <+25>:   call    0x4004d7 <add>
0x000000000400509 <+30>:   mov     DWORD PTR [rbp-0x4], eax
0x00000000040050c <+33>:   mov     eax, DWORD PTR [rbp-0x4]
0x00000000040050f <+36>:   mov     esi, eax
0x000000000400511 <+38>:   lea     rdi, [rip+0x9c]          # 0x4005b4
0x000000000400518 <+45>:   mov     eax, 0x0
0x00000000040051d <+50>:   call    0x4003f0 <printf@plt>
0x000000000400522 <+55>:   mov     eax, 0x0
0x000000000400527 <+60>:   leave
0x000000000400528 <+61>:   ret
End of assembler dump.
(gdb) b* 0x000000000400504
Breakpoint 1 at 0x400504: file basic2.c, line 11.
(gdb)
```

saya melakukan breakpoint pada alamat 0x000000000400504, jika dilihat disana, itu merupakan tempat intruksi call, intruksi call diatas digunakan untuk memanggil fungsi add

```
(gdb) r
Starting program: /home/ramdhan/kulgram/Materi-Kulgram/ReversingID/2017
Breakpoint 1, 0x000000000400504 in main (argc=1, argv=0x7fffffff718)
11      x = add(10, 20);
(gdb)
```

lalu kita jalankan

kalian perhatikan digambar ini, sebelum intruksi call, ada intruksi

```
mov esi, 0x14  -> artinya menyimpan nilai 0x14 (20 dalam desimal) ke register esi
mov edi, 0xa   -> artinya menyimpan nilai 0xa (10 dalam desimal) ke register edi
```

ternyata itu adalah argumen yang digunakn untuk memanggil fungsi add, argumen pertama akan disimpan di register edi, dan argumen ke 2 akan disimpan di register esi

```
(gdb) info registers
rax            0x4004eb 4195563
rbx            0x0      0
rcx            0x0      0
rdx            0x7fffffff728 140737488344872
rsi            0x14      20
rdi            0xa      10
rbp            0x7fffffff630 0x7fffffff630
rsp            0x7fffffff610 0x7fffffff610
r8             0x4005a0 4195744
r9             0x7ffff7de99e0 140737351948768
r10            0x4       4
r11            0x1       1
r12            0x400400 4195328
r13            0x7fffffff710 140737488344848
r14            0x0      0
r15            0x0      0
rip            0x400504 0x400504 <main+25>
eflags        0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

perintah `info registers` akan menampilkan semua register beserta nilainya, register rdi dan rsi berisi

argumen pertama dan kedua untuk memanggil fungsi add

```
(gdb) info reg edi
edi                0xa      10
(gdb) info reg esi
esi                0x14     20
```

info registers bisa disingkat info reg atau i r dan bisa diikuti dengan nama register yang ingin ditampilkan untuk mengganti nilai register, kita menggunakan perintah set seperti sebelumnya, tapi agak sedikit berbeda contohnya, kita akan mengganti nilai edi (yang berisi argumen pertama) menjadi 31317

```
(gdb) set $edi=31317
```

lalu kita lanjutkan prosesnya dengan perintah continue atau c saja

```
(gdb) set $edi=31317
(gdb) c
Continuing.
31337
[Inferior 1 (process 17313) exited normally]
(gdb)
```

oke, mungkin itu saja yang bisa saya bagikan, selanjutnya kalian bisa cari referensi lainnya diinternet, salah satunya disini <https://betterexplained.com/articles/debugging-with-gdb/> dan disini https://www.tutorialspoint.com/gnu_debugger/ masih jauh dan banyak yang harus dipelajari 😊, belajar bahasa C dan assembly akan sangat membantu untuk belajar reverse engineering. selebihnya bisa kalian tanyakan dan akan saya jawab semampunya. Terimakasih