

Symbolic Execution

Terima kasih buat yang udah hadir di kulgram kali ini

Hari ini kita akan bahas tentang Symbolic Execution dan Angr. Sekarang lumayan nge-hits terutama digunakan pas ctf ataupun menyelesaikan crackme challenge

Mungkin ada yang pernah dengar tentang angr, atau pernah coba tentang angr. Angr adalah framework untuk binary analysis. Ada banyak sekali kegunaannya, tapi kita batasi buat symbolic execution.

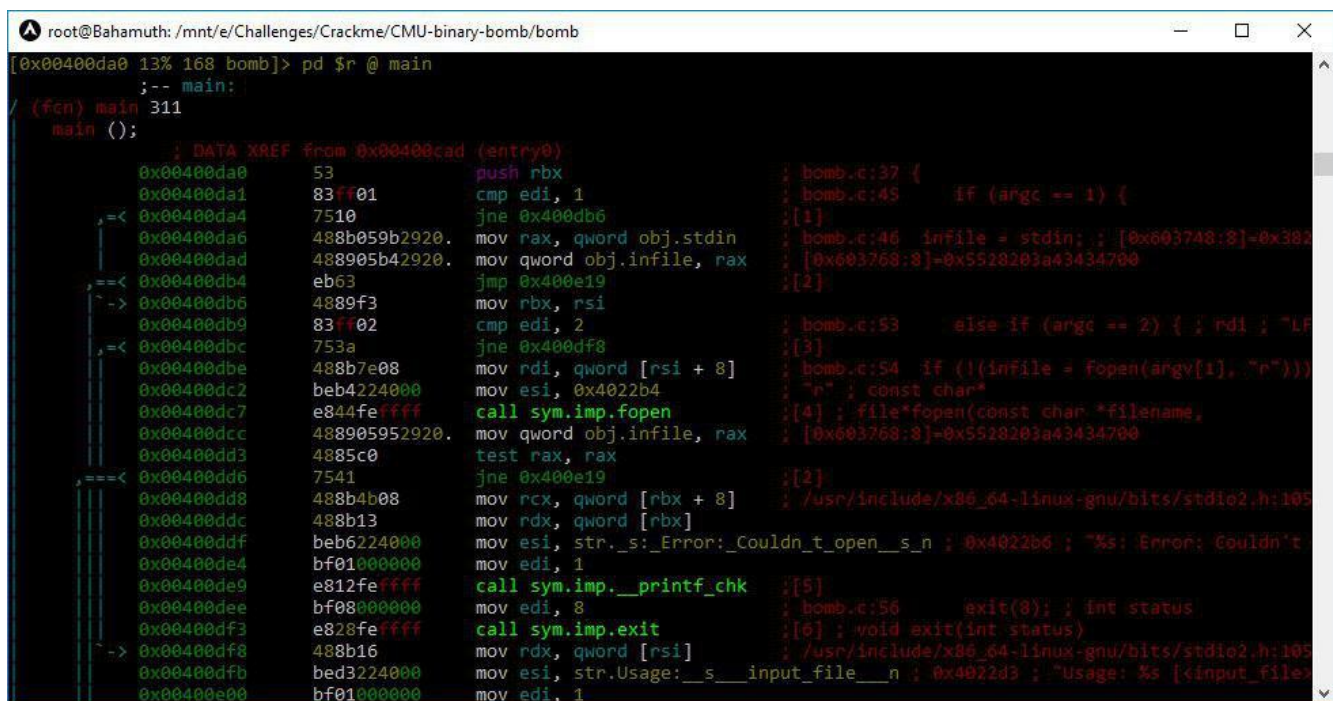
Kuharap sih udah pada install jadi pas sesi setelah ini bisa langsung coba.

angr menggunakan python dan karena sifatnya yang scriptable itu dia banyak dipake oleh orang-orang. Writeup penyelesaian kasus dengan angr pun sudah mulai banyak.

tapi sebelum kita bahas angr, kita bahas dulu apa itu symbolic execution

Dahulu sekali, program itu tak terlalu kompleks seperti sekarang. Jaman dulu, RE ya sebagian besar tentang disassembly -> baca kode -> lakukan perubahan. Kalau perlu analisis kelakuannya dengan cara debug atau dijalankan.

Tapi semakin lama program semakin kompleks. Kalau kita menelusuri alur sebuah program hanya berbekal baca teks doang, maka kita bisa tersesat. Karena itu solusi yang brilian adalah membagi kode-kode itu menjadi seperti graph



```
root@Bahamuth: /mnt/e/Challenges/Crackme/CMU-binary-bomb/bomb
[0x00400da0 13% 168 bomb]> pd $r @ main
;-- main:
; (fcn) main 311
main ();
; DATA XREF from 0x00400cad (entry0)
0x00400da0 53 push rbx
0x00400da1 83ff01 cmp edi, 1
0x00400da4 7510 jne 0x400db6
0x00400da6 488b059b2920. mov rax, qword obj.stdin
0x00400dad 488905b42920. mov qword obj.infile, rax
0x00400db4 eb63 jmp 0x400e19
0x00400db6 4889f3 mov rbx, rsi
0x00400db9 83ff02 cmp edi, 2
0x00400dbc 753a jne 0x400df8
0x00400dbe 488b7e08 mov rdi, qword [rsi + 8]
0x00400dc2 beb4224000 mov esi, 0x4022b4
0x00400dc7 e844feffff call sym.imp.fopen
0x00400dcc 488905952920. mov qword obj.infile, rax
0x00400dd3 4885c0 test rax, rax
0x00400dd6 7541 jne 0x400e19
0x00400dd8 488b4b08 mov rcx, qword [rbx + 8]
0x00400ddc 488b13 mov rdx, qword [rbx]
0x00400ddf beb6224000 mov esi, str._Error:_Couldn_t_open_s_n ; 0x4022b6 ; "%s: Error: Couldn't
0x00400de4 bf01000000 mov edi, 1
0x00400de9 e812feffff call sym.imp._printf_chk ;[5]
0x00400dee bf08000000 mov edi, 8
0x00400df3 e828feffff call sym.imp.exit ;[0] ; void exit(int status)
0x00400df8 488b16 mov rdx, qword [rsi]
0x00400dfb bed3224000 mov esi, str.Usage:_s__input_file__n ; 0x4022d3 ; "Usage: %s [<input_file>
0x00400e00 bf01000000 mov edi, 1
```

Contoh misalnya gambar diatas ini, ini adalah challenge dari CMU Binary bomb dibuka dengan radare2, membaca teks aja butuh waktu untuk memahami alur dari sebuah binary.

Kalau kita liat definisinya, symbolic execution adalah teknik analisis program untuk menentukan input apa yang dapat menyebabkan suatu bagian program tereksekusi.

Misal kita ada kode C seperti ini:

```
int f()
{
// ...
scanf("%d", &y);
x = y * 2 + 135;
z = x / 2;
if (z == 15)
{
printf("Flag: %s", read_flag_from_file());
}
else
{
printf("Bukan!");
}
}
```

Kita ingin menampilkan flag. Untuk menampilkan bagian itu ada kondisi yang harus terpenuhi yaitu : $z == 15$
z diperoleh dari $x / 2$
x diperoleh dari $y * 2 + 135$
y diperoleh dari inputan pengguna.

Inputan pengguna bisa sembarang, tapi kira-kira bilangan apa yang bisa memenuhi syarat di atas sehingga flag bisa didapatkan? Kita tidak tau (bisa dihitung manual sih) tapi begitulah ide dasarnya dari symbolic execution.

Jadi di sana ada dua cabang kan, 1 blok yang kita inginkan, dan 1 lagi adalah blok yang kita hindari. Symbolic execution akan mencoba kemungkinan-kemungkinan nilai yang bisa dimasukkan, mencoba mencari input apa yang bisa digunakan untuk mencapai flag dan menghindari kata bukan.

Japi bagaimana cara penelusurannya? brute force setiap block kode yang ada di antara input dan tujuan? Nggak

Ada ralat dikit, enaknya itu 135 diganti 136

Jika diperhatikan lagi, sebenarnya untuk mencapai blok itu ada kondisi yang harus terpenuhi, yaitu kita harus mencari nilai y yang memenuhi persamaan ini:

$$\begin{aligned}x &= y * 2 + 136 \\z &= x / 2 \\z &= 15\end{aligned}$$

Jika disederhanakan menjadi

$$\begin{aligned}(y * 2 + 136) / 2 &= 15 \\ \text{atau} \\ y * 2 + 136 &= 30 \\ y &= (30 - 136) / 2 = 73\end{aligned}$$

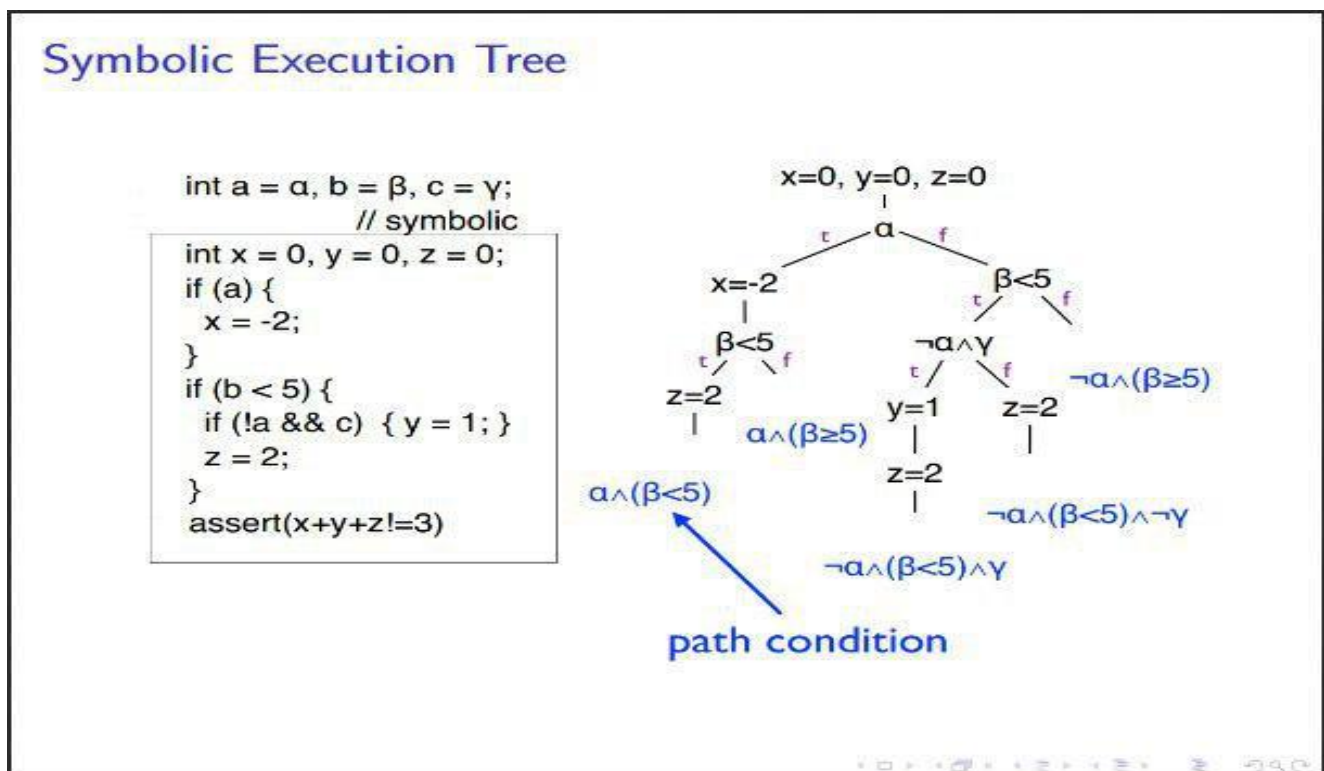
Jika kita memasukkan 73 maka kita akan mencapai blok flag.

Kondisi-kondisi ini disebut sebagai constraint atau kondisi yang membatasi solusi input yang ada.

Untuk menelusuri dari awal hingga blok akhir, symbolic execution bekerja dengan cara mencari kondisi-kondisi apa saja yang membatasi dan kemudian memecahkan kondisi itu untuk mencari nilai yang tepat.

Batasan ini bisa diciptakan oleh symbolic execution sendiri atau kita tetapkan. Misal jika kita tau bahwa input hanya printable character, maka kita bisa berikan constraint bahwa nilai per byte memiliki rentang tertentu.

Contoh lain:



Picture diambil dari slide bahan ajar suatu institusi pendidikan di luar.

Di akhir ada assert ($x + y + z \neq 3$), apa x , y , dan z yang menyebabkan assert itu gagal?

x , y , dan z diset ketika kondisi-kondisi tertentu tercapai. Misal jika ada nilainya maka x diset -2. untuk mencapai assert itu bisa melalui beberapa jalur / path. Dilihat di sana ada banyak sekali kemungkinan path-nya. Berapa nilai a ? b ? c ? kita tidak tau. Maka kita simbolkan saja dengan sesuatu, misal α , β , γ .

Ketika kita berbicara simbol, kita gak peduli nilai konkretnya berapa itu a , b , c . Yang kita liat adalah bahwa ada nilai a , b , c . Udah.

Misal jalur yang kita ambil sekarang. adalah

- a false
- b bernilai kurang dari 5
- c bernilai bilangan positif.

Maka kalo ngeliat di tree kondisinya, $\sim\alpha \wedge (\beta < 5) \wedge \gamma$, ternyata $x + y + z$ bernilai 3 maka kondisi terpenuhi. Setelah ini, symbolic execution baru mencari kira-kira α , β , dan γ ini bisa bernilai apa saja sehingga kondisi ini terpenuhi.

nah itu tadi, symbolic execution intinya adalah simbol. Jadi kita menyimbolkan semua inputan. Jika kondisi tercapai maka kita tinggal mencari nilai-nilai yang mungkin.

aplikasinya seperti apa?

- mencari password yang sesuai. Password ini di-hardcoded tapi metode pengecekannya memusingkan. Yang pasti dia perlu input. Cari input ini apa kira-kira. Tapi karena bisa diketik berarti inputnya adalah semua karakter printable.
- ada sebuah bug buffer overflow. Input macam apa yang bisa diberikan untuk bisa terjadi buffer overflow?

Dsb

nah angr bisa digunakan untuk symbolic execution.

ada beberapa komponen di angr, kalau kalian perhatikan pas instalasi.

ada komponen untuk melakukan pemecahan constraint, pake z3 dari microsoft.

ada komponen untuk pembentukan graph (block-block kode) dari aplikasi

ada komponen untuk emulasi dan menjalankan kode (nggak hanya x86 aja, tapi juga arsitektur lain seperti arm)

dsb.

Emplimentasi Angr

Langsung aja contoh pertama

Skripnya seperti gambar dibawah ini

```
6 #include <stdio.h>
5 #include <string.h>
4 #include <stdlib.h>
3
2 int main(void)
1 {
7     char pass[14];
1     printf("Masukkan password : ");
2     fgets(pass, 13, stdin);
3     if(!strcmp(pass, "C0D3BR34K3R\n"))
4     {
5         printf("Password benar\n");
6     }
7     else
8     {
9         printf("Password salah\n");
10    }
11 }
```

Sumber code C

Tujuannya bukan nyari passwordnya ya :v, karena passwordnya udah ketauan disitu. tapi yang kita tau itu, gimana caranya menggunakan angr untuk mencari blok kode yg kita inginkan.

Disini angr akan mencari, apa seharusnya inputan kita agar eksekusi program sampai ke dalam blok if

Basicnya untuk meload angr, sepeti ini

```
import angr
```

```
p = angr.Project("./namabinary")
```

Untuk mengontrol eksekusi program kita menggunakan `angr.factory.path()`.

Dalam sekali eksekusi path dapat berjumlah lebih dari 1

Contohnya pada program diatas terdapat 1 percabangan

Ketika eksekusi program sampai pada `if()`, maka path akan berjumlah 2. yg 1 path yg berada diblok `if`, dan 1 lagi berada di blok `else`. masing2 path mempunyai state tersendiri. istilah state di angr adalah kondisi register, memory yg dipunyai masing2 path.

Sama jika terjadi if bercabang (nested if), path akan berjumlah menjadi 4. masing2 path berada di blok program if atau else

```
11 import angr
10 import logging
9 logging.getLogger('angr.engines').setLevel('DEBUG')
8 strcmp = angr.Project("./strcmp") # Meload binary
7 p = strcmp.factory.path()
6
5 # melakukan step sampai branch
4 p.step()
3 while len(p.successors) == 1:
2     p = p.successors[0]
1     p.step()
12 □
1 b_left = p.successors[0] # Branch left adalah blok if (jika password benar)
2 b_right = p.successors[1] # Branch right adalah blok else (jika password salah)
3
4 print b_left.state.posix.dumps(0) # Tampilkan data yg berada di stdin (fd=0)
```

sumber code angr untuk strcmp

Nah diatas, adalah skrip angr yg digunakan untuk mendapatkan apa inputan kita, jika path berada didalam blok if

Setelah mendapatkan path di baris ke 5. selanjutnya kita memulai menjalankan path saat ini dengan p.step()

Baris 1 dimulai dari atas ya :v

Selanjutnya ada while loop. itu digunakan untuk melakukan path.step() selama jumlah path sama dengan 1

Karna setelah path menemui percabangan, path akan berjumlah 2. dan path yg ingin masuk ke blok if, akan mencari apa inputan yg tepat, agar kondisi diblok if terpenuhi.

Path.succesors akan berisi path2 yg aktif saat ini

kita akan mengambil path yg berada di blok if.

di p.successors[0]

Selanjutnya kita tampilkan. apa nilai dari input, yg menyebabkan blok if dieksekusi

```
ramdhan ~ > angr > kulgram > python solve_strcmp.py
WARNING | 2017-08-19 18:49:57,072 | claripy | Claripy is setting the
C0D3BR34K3R
```


ini cuman dasar, bagaimana kita menggunakan path yg ada di angr. selebihnya kalian ngoprek sendiri yak ^_^.

skrg kita ke contoh selanjutnya

ada contoh lagi nih

```
12 #include <stdio.h>
11 #include <stdlib.h>
10 #include <assert.h>
9
8 int basic(int x, int y, int z){
7     if((x%2) != (y*z)%2){
6         z = y*x+z;
5         if(z%2 == 0){
4             z -= x;
3             y += z;
2         }
1     }else{
13         z = z*x+y;
1         if(z%2 == 1){
2             z += x;
3             y -= z;
4         }
5     }
6     return y+z-x;
7 }
8 int main(int argc, char* argv[]){
9     int x = atoi(argv[1]);
10    int y = atoi(argv[2]);
11    int z = atoi(argv[3]);
12    printf("%d\n", basic(x, y, z));
13    assert(basic(x, y, z) == 31337);
14 }
```

sumber code C

Tujuan kita adalah, mencari nilai x, y, dan z yg dilewatkan kedalam fungsi basic dengan syarat return value dari fungsi harus bernilai 31337 ?

Karena kita hanya perlu melakukannya pada fungsi 'basic', kita bisa menggunakan `angr.factory.callable(addr)` dimana `addr` adalah alamat dari fungsi yg ingin kita panggil. `addr` akan kita isi dengan alamat dari fungsi basic

Selanjutnya yg kita perlukan adalah. symbolic variable dengan nama x, y, dan z dengan besar 32bit

Untuk membuat symbolic variable bisa kita lakukan di angr.

Begini skripnya, saya akan jelaskan satu2


```

20 import angr
19 from socket import ntohl # Untuk konversi big endian ke little endian 32bit
18 from ctypes import c_int
17 from time import time
16
15 """Mengubah unsigned int menjadi signed int"""
14 def signed(x):
13     return c_int(x).value
12
11 prog = angr.Project("./basic", load_options={"auto_load_libs":False})
10 basic_addr = prog.loader.main_bin.get_symbol('basic').addr # Mendapatkan alamat dari fungsi basic
9 basic = prog.factory.callable(basic_addr) # Alamat dari fungsi basic()
8 s = prog.factory.entry_state()
7
6 """ Membuat symbolic variable x, y dan z dengan besar 32bit"""
5 x = s.se.BVS('x', 32)
4 y = s.se.BVS('y', 32)
3 z = s.se.BVS('z', 32)
2 """ solver """
1 s.add_constraints(basic(x, y, z) == 31337)
21
1 """ Mengambil hasil dan menkonversinya menjadi signed little endian 32bit """
2 nilai_x = signed(ntohl(s.se.any_int(x)))
3 nilai_y = signed(ntohl(s.se.any_int(y)))
4 nilai_z = signed(ntohl(s.se.any_int(z)))
5
6 print "x = {0}".format(nilai_x)
7 print "y = {0}".format(nilai_y)
8 print "z = {0}".format(nilai_z)

```

s.se.BVS('x', 32) akan membuat symbolic variable bernama x, dan dilanjutkan dengan y dan z

s adalah kondisi awal program, diambil dari prog.factory.entry_state.

Setelah itu dengan add_constraints kita kasih tau ke angr, agar hasil dari basic(x, y, z) harus menghasilkan nilai 31337

Kita tidak tahu apa yg angr lakukan di belakang layar :p

Selanjutnya kita ambil nilai x, y, z dengan x.se.any_int yg selanjutnya kita convert ke little endian 32 bit

Selanjutnya kita ambil nilai x, y, z dengan x.se.any_int yg selanjutnya kita convert ke little endian 32 bit

```

ramdhan ~ > angr > kulgram > python basic.py
WARNING | 2017-08-19 18:46:25,957 | claripy | Claripy is setting the r
x = 54209272
y = 54240609
z = -1178382584
ramdhan ~ > angr > kulgram > ./basic 54209272 54240609 -1178382584
31337

```

Skrip diatas jika kita jalankan. kita sudah lolos dari fungsi assert

jika ada pernyataan saya yg salah mohon di koreksi ya :D

lanjut ya.. 1 contoh lagi

oiya ini saya berinama cari.c binarynya adalah 'cari' tinggal atur aja di skrip angr nya nanti

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

int main(int argc, char** argv){
    unsigned int cari[] = {71, 219, 1754, 7497, 24328, 111623,
                           413702, 1802245, 3407876, 28835843,
                           59768834, 398458881};

    int i;
    if(argc < 2){
        printf("Use : %s <password>\n", argv[0]);
        exit(-1);
    }
    if(strlen(argv[1]) != 12){
        SALAH:
        printf("Password salah\n");
        exit(-1);
    }
    for(i=0; i<12; i++){
        if((unsigned int)(argv[1][i] << (2*i) ^ (12-i)) != cari[i])
            goto SALAH;
    }
    printf("Password benar\n");
}
```

yg kita inginkan adalah eksekusi program meraih sampai printf("password benar") dan menghindari printf("password salah"),

Inputan kita berada di argv[1], jika dilihat di strlen inputan kita harus mempunyai panjang 12byte

Kita buat symbolic variable disini

```
8 import angr
7
6 cari = angr.Project("./cari", load_options={"auto_load_libs":False})
5
4 input_len = 12
3 flag = angr.claripy.BVS("flag", input_len*8)
2
1 cari_state = cari.factory.entry_state(args=["./cari", flag])
9 path_group = cari.factory.path_group(cari_state)
1
2 path_group.explore(find=0x4006eb, avoid=0x400691)
3 # '-----> ini alamat dimana password benar ditampilkan
4 # '-----> ini alamat dimana password salah ditampilkan
5 found = path_group.found[0]
6 flagnya = found.state.se.any_str(flag)
7
8 print flagnya
```

Dibaris 7 kita membuat symbolic variable bernama flag dengan panjang 12byte. dan kita buat sebagai

argumen, di baris 8

Di contoh 1 kita menggunakan path. disini kita menggunakan pathgroup, pathgroup sbenarnya adalah kumpulan dari path

Di path_group.explore tinggal kita masukkan alamat dimana program menampilkan "password benar" ke dalam find dan alamat yg kita hindari yg menampilkan password salah kedalam avoid

Anggap angr sudah sampai ke alamat yg kita inginkan, setelah itu kita ambil path yg ibaratnya sudah berhasil meraih diblok program yg menampilkan password benar

Selanjutnya kita ambil isi dari variable 'flag'

```
ramdhan ~ > angr > kulgram python solve_cari.py
WARNING | 2017-08-19 19:48:45,175 | claripy | Claripy is setting
K4mu men4n9
ramdhan ~ > angr > kulgram ./cari
Use : ./cari <password>
ramdhan ~ > angr > kulgram ./cari K4mu_men4n9_
Password benar
ramdhan ~ > angr > kulgram
```

Nah, mungkin disini ada yg mau nanya

Penjelasnya terlalu gajelas ya ? :v

Angr akan melakukan eksekusi dengan nilai2 symbolic, tanpa kita tahu apa seharusnya nilai tsb

Diakhir jika sudah sampai kedalam path yg kita inginkan, kita tampilkan nilai2 tsb