hackerearth

Penetration Testing Report

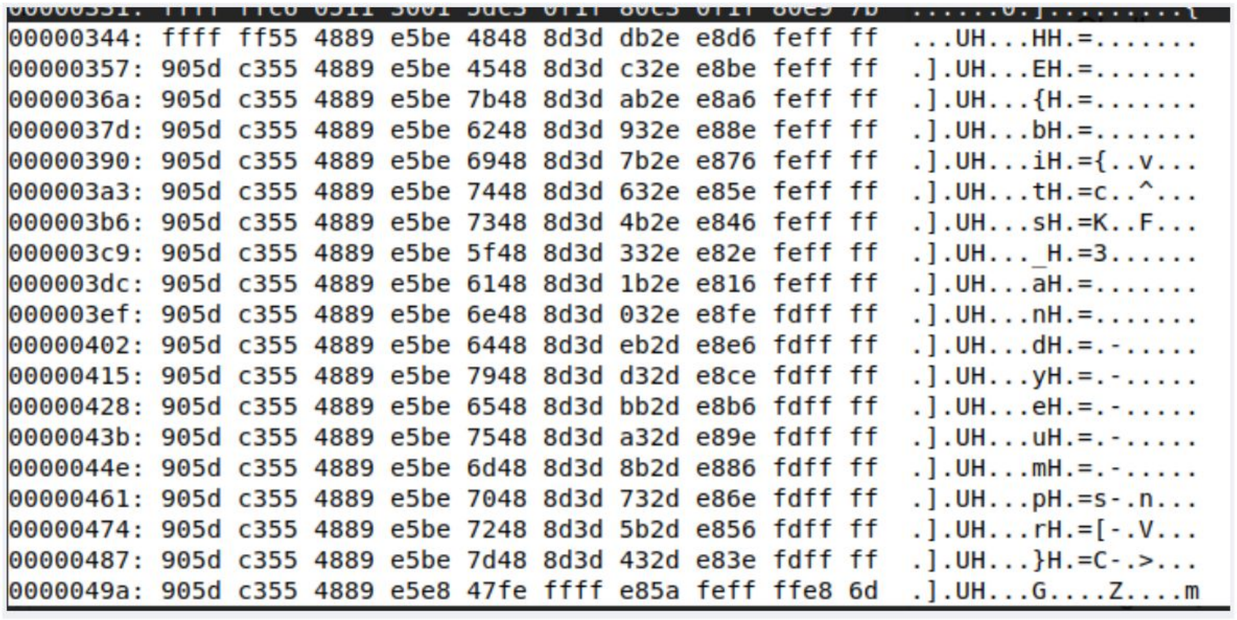
For

“NightMare :-)”

S.NO.	Title	#
1.	Challenge Category	Reverse Engineering
2.	Challenge Related Files	N/A
3.	File Link / Target IP	N/A

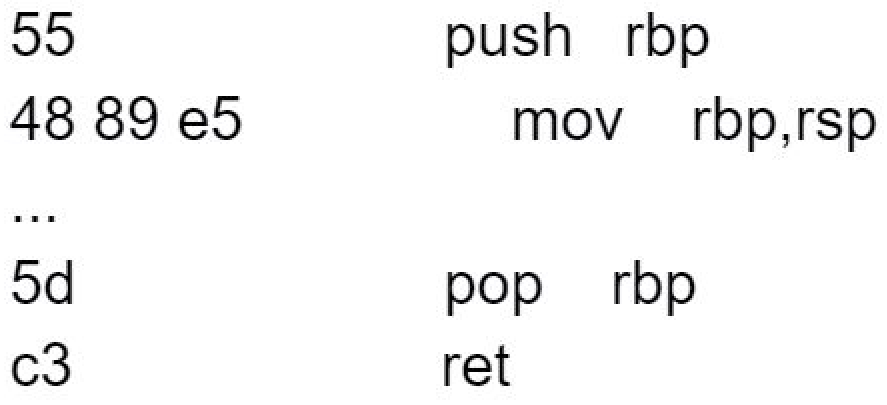
PROCEDURE

1. **We are provided with a core dump. Examining the flavor-text and the dump, we notice that the dump has no null bytes; we conjecture that they have been stripped out.**
2. **xxd -c 19 bianry this will give a deduplicated flag in column**



```
00000344: ffff ff55 4889 e5be 4848 8d3d db2e e8d6 feff ff  ...UH...HH.=.......
00000357: 905d c355 4889 e5be 4548 8d3d c32e e8be feff ff  .].UH...EH.=.......
0000036a: 905d c355 4889 e5be 7b48 8d3d ab2e e8a6 feff ff  .].UH...{H.=.......
0000037d: 905d c355 4889 e5be 6248 8d3d 932e e88e feff ff  .].UH...bH.=.......
00000390: 905d c355 4889 e5be 6948 8d3d 7b2e e876 feff ff  .].UH...iH.={..v...
000003a3: 905d c355 4889 e5be 7448 8d3d 632e e85e feff ff  .].UH...tH.=c..^...
000003b6: 905d c355 4889 e5be 7348 8d3d 4b2e e846 feff ff  .].UH...sH.=K..F...
000003c9: 905d c355 4889 e5be 5f48 8d3d 332e e82e feff ff  .].UH..._H.=3......
000003dc: 905d c355 4889 e5be 6148 8d3d 1b2e e816 feff ff  .].UH...aH.=.......
000003ef: 905d c355 4889 e5be 6e48 8d3d 032e e8fe fdff ff  .].UH...nH.=.......
00000402: 905d c355 4889 e5be 6448 8d3d eb2d e8e6 fdff ff  .].UH...dH.=.-.....
00000415: 905d c355 4889 e5be 7948 8d3d d32d e8ce fdff ff  .].UH...yH.=.-.....
00000428: 905d c355 4889 e5be 6548 8d3d bb2d e8b6 fdff ff  .].UH...eH.=.-.....
0000043b: 905d c355 4889 e5be 7548 8d3d a32d e89e fdff ff  .].UH...uH.=.-.....
0000044e: 905d c355 4889 e5be 6d48 8d3d 8b2d e886 fdff ff  .].UH...mH.=.-.....
00000461: 905d c355 4889 e5be 7048 8d3d 732d e86e fdff ff  .].UH...pH.=s-.n...
00000474: 905d c355 4889 e5be 7248 8d3d 5b2d e856 fdff ff  .].UH...rH.=[-.V...
00000487: 905d c355 4889 e5be 7d48 8d3d 432d e83e fdff ff  .].UH...}H.=C-.>...
0000049a: 905d c355 4889 e5e8 47fe ffff e85a feff ffe8 6d  .].UH...G....Z....m
```

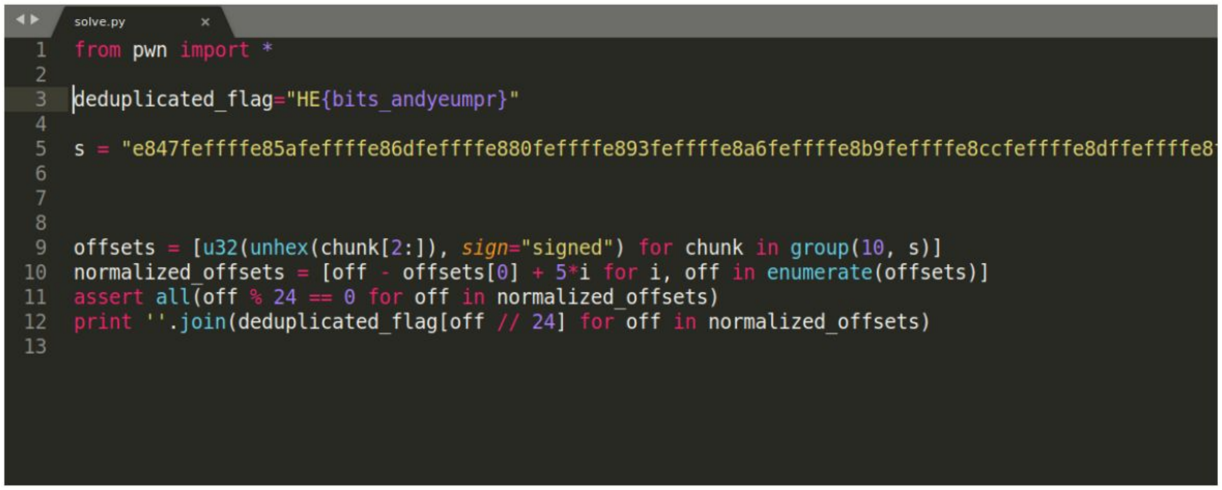
3. **So now we know that flag exists somewhere in binary how to extract that**
4. **deduplicated_flag="HE{bits_andyeumpr}"**
5. **Let's do one thing: extract all such function calls.**
6. **20-byte chunks that have the outline of a function definition and each hold a character:**



```
55              push   rbp
48 89 e5          mov    rbp,rsp
...
5d              pop    rbp
c3              ret
```

7. **A typical function call will be**
   e8 b1 fe ff ff   call   0xfffffffffffffec6
8. **So extract all such call which are tied together**

9. **All calls are e847feffff,e85afeffff,e86dfeffff,e880feffff,e893feffff,e8a6feffff,e8b9feffff, e8ccfeffff,e8dffeffff,e8f2feffff,e805ffffff,e8b8feffff,e853feffff,e80effffff,e879feffff,e81cffffff,e887feffff,e89afeffff,e8ddfeffff,e820ffffff,e833ffffff,e846ffffff,e881feffff,e894feffff,e8a7feffff,e8bafeffff,e86dfeffff,e850feffff,e833feffff,e836ffffff,e811feffff,e814ffffff,e83fffffff**
10. **Now let's rebuild all calls into functions.**


solve.py

```python
from pwn import *

deduplicated_flag="HE{bits_andyeumpr}"

s = "e847feffffe85afeffffe86dfeffffe880feffffe893feffffe8a6feffffe8b9feffffe8ccfeffffe8dffeffffe8



offsets = [u32(unhex(chunk[2:]), sign="signed") for chunk in group(10, s)]
normalized_offsets = [off - offsets[0] + 5*i for i, off in enumerate(offsets)]
assert all(off % 24 == 0 for off in normalized_offsets)
print ''.join(deduplicated_flag[off // 24] for off in normalized_offsets)

```

11. **Run this given code and it will print the flag. {s is all the function calls }**

Flags:

S.No.	Flag - No.	Flag
1.	Flag 1	HE{bits_and_bytes_dump_and_strip}