EHU/UPV

Ingenieritza informatikoa

Buffer overflow

Egilea:
Iker Sandoval

2025/04/01

Índice

1. Sarrera

nformatika-segurtasunaren arloan, ahultasunak ustiatzea ezinbesteko praktika da sistemak nola konprometitzen diren eta, azken finean, nola babestu ikasteko. Sistema zaharretan edo behar bezala konfiguratu gabe daudenetan, eraso ohikoenetako bat buffer overflow delakoa da; horrek erasotzaileari kode arbitrarioa sartzeko eta exekutatzeko aukera ematen dio.

Aurkitu dugu exekutagarri bat eta exekutagarri horren bidez pila gainezkatzea eta shell bat lortzea, helburua izango da. Horren ostean, SUID bita erabiliko dugu pribilegio eskalada bat lortzeko.

Honen helburua izango da ikustea nola pribilegiodun exekutagarri bat izanda eta konfigurazio batzuk kenduta, ikustea nola posible den root bezala jartzea.

Horretarako, pila nola funtzionatzen den ondo jakin behar da.

2. Herramientak

- Linux bat x86-64 egiturarekin
- gdb

3. Analisia.

Hasteko, emandako kodea ulertu behar dugu. Honek, egiten duena da parametroz jasotzen du parametro bat eta 64 byteko buffer batera kopiatzen du. Kopiatzeko, strcpy erabiltzen du eta honek ez ditu bufferraren mugak egiaztatzen. Horresegatik, ustiapena hemendik hasiko da. Lortu nahi duguna da, pilan oferflow bat sortzea, memoriara pasatzeko.

3.1. Shell bat lortu

Itzulera helbidea lortu nahi dugu, horresegatik parametro bezala pythoneko print bat erabili da hainbat .ᴬ"karaktere erabilita. adibidez, .ᴬAAA". Probak eginez gero, jakin dugu 72 A erabilita, itzulera helbidearen desplazamendua jakingo dugu.

Behin hau jakinda, shellcode bat sortuko dugu. Horretarako 59. syscall erabiliko dugu. Behar diren parametroak prestatuz, kontuan izanda little-endian erabili behar dela. Kodea konpilatuko dugu nasm erabilita eta exekutatuko dugu ld erabilita.

Orain, badakigu 64 byteko buffer bat daukagula eta 72 byteko desplazamendua behar dugula. Buffer honen amaieran lortutako shell-aren hexadezimala idatziko dugu eta kalkulatuko digutu zenbat byte dituen. Gure kasuan, 31 dira. Buferra betetzeko, 64-shellkodearen luzera, NOP behar ditugu. Gure kasuan 33 dira.

Badaukagu 64 Byteko desplazamendua pilaren hasieratik, beste 8 behar ditugu eta horretarako beste NOP kate bat sartuko dugu pilan.

Momentu honentan nahi dugu pilan egotea:

- 33 NOP katea pilaren hasieran
- shellkodea
- 8 NOP katea behean
- kodearen hasieraren helbidea

Azkenengoz, kodearen hasiera bilatu nahi dugu. Horretarako .c programa bat sortuko dugu non emandako programa exekutatuko du execv bitartez.

C programa array bat izango du payloadekin eta argumentu bezala esleituko zaio emandako programario execv bitartez.

ASLR konfigurazioa kenduko dugu 0 bat idatziz hemen: /proc/sys/kernel/randomizevaspace

NX protekzioa kentzeko dmesg erabili dugu.

gcc-rekin konpilatzerako orduan -fno-stack-protector eta -z no-exec flag-ak erabiltzea gomendagarria da. Biano printzipioz, ez du eragininik sortutako programan, baina, aurkitutako programan bai, zeren eta lortutako helbideak aldakorrak izan daitezke. Gure kasuan helbide finko baten bila gabiltza.

Behin programa konpilatuta izanda, exekutatzerako orduan, ikusiko dugu segmentation fault, hau da helbidea ez delako egokia. Horretako, gdb erabiliko dugu hainbat A erabilita. Breakpoint bat jarriko dugu strcopy ostean, horrela kopia egingo da pilara eta hor rsp-ren balioak ikusiko ditugu. Ikusten baditugu A hauek hexadezimalean(41) ondo goaz, helbidea hartuko dugu eta payload-en falta den kodearen zatian idatziko dugu little-endian eta hexadezimalez.

Payload programa probatuko dugu. Gure kasuan, makefile bat sortu dugu errazago exekutatzeko. Exekutatzerakoan erroren bat ematen badu, helbideekin jolasten hasi behar gara. Shell bat ikusi arte.

3.2. SETUID bat lortu

Behin shell bat izanda, pribilegioen bila goaz. Pribilegio gehien dituenak root da eta bere UID 0 da. Orduan honen bila goaz, posible da user guztientzako erabiltzea baino helbidea bilatu behar da. Root en bila joateko asm programa bat sortu dugu setreuid(0,0) syscall erabiliko dugu.

Programa honela funtzionatzen du: xor rdi,rdi (rdi garbitzeko, 0 bezala jartzeko) bat egin dugu. Rsi,rax erregistroekin berbera egin dugu. Orain, rax eta lehengo bi argumentuak 0 bezala ditugu

setreuid syscall-en zenbakia 71 da. Orduan, balore hori esleituko dugu mov al, 0x71

Amaitzeko syscall exekutatuko dugu "0f05"jarriz.

setuid(getuid()) erabiltzen badugu bi syscall erabiliko ditugu eta batek bueltatzen duena besteak hartuko du.

4. Ondorioak

Praktika honek erakutsi digu nola funtzionatzen duten buffer overflow motako ahuleziek 64 biteko inguruneetan eta nola manipulatu pilak programa kalteberak kontrolatzeko. Gainera, SUID bitarekin eskubideak igotzeak sistema eragileetan pribilegioen kudeaketa zorrotza aplikatzearen garrantzia erakutsi du.

Ustiapenaren arrakasta shellcode egituratu batean eta RIPen desplazamendu zuzenaren kalkuluan oinarritu da. Gainera, inguruneko babes-neurriak gaitasungabetzeak ahultasuna errepikatzeko proba kontrolatuak egitea ahalbidetu du.

Praktikaren bidez ikasitakoak sistema errealetan ahuleziak identifikatzeko eta horiei aurre egiteko balioko du, baita programazio-praktika seguruak aplikatzeko ere. Babes-neurriak ezartzea eta pribilegioak behar bezala kudeatzea oinarrizko urratsak dira sistema seguruak izateko.

5. Github repositorioa

https://github.com/ikerSandoval003/Bufferoverflow.git