



Tema 3 - Exploit ELFs, not elves

- Deadline: 10.01.2020 23:55
- Data publicării: 08.12.2019
- Responsabili:
 -  Mihai Dumitru
 -  George Muraru
- Updates:
 - 10.12.2019 - clarificare format flag
 - 10.12.2019 - modificare cerințe upload
 - 10.12.2019 - clarificare shellcode ASLR
 - 17.12.2019 - VM e opțional

Enunț

Anul acesta, Moș Crăciun s-a hotărât să împartă binare copiilor din satul UPB. Deoarece nu poate vedea în viitor, Moșul a scris 2 binare per persoană - unul pentru cazul în care am fost cuminți, iar altul pentru cazul în care am fost pe *naughty list*.

1. Analiza binarului - 20p

Chiar înainte de a compila binarele, Grinch s-a strecurat în lăcașul moșului și a adăugat o vulnerabilitate în surse. Deoarece Moșul nu înțelege instrucțiunile assembly și deoarece mai este puțin timp până la Craciun, acesta vă cere ajutorul.

Codul sursă a fost șters “din greșeală”, iar aplicația a rămas în mare parte nedocumentată. De asemenea, etichetele din cadrul programului au fost șterpelite de spiridușii obraznici ai Moșului.

La acest task, trebuie să analizați binarul **nice** și să descoperiți la ce adresă se găsește funcția vulnerabilă.

Scrieți în fișierul README adresa funcției vulnerabile și de ce este aceasta vulnerabilă.

2. Spargerea binarului - 40p

Moșul vă mulțumește pentru ca l-ați ajutat să identifice zona buclucașă din cod, însă acum dorește să înțeleagă și de ce acea zonă era problematică (tot pentru binarul **nice**).

Pentru acest task, trebuie sa generați un payload (va fi citit de la stdin) care va face programul să printeze un flag de forma: **NICE_FLAG{<sir_de_caractere>}** (dacă un flag de această formă se regăsește în outputul programului, înseamnă că ați reușit).


3. Spargerea binarului v2 - 40p

Pentru acest task, Moș-ul are nevoie de voi pentru a gasi vulnerabilitatea, iar mai apoi a sparge binarele copiilor obraznici (în binarul **naughty**).

În mod similar cu subpunctele anterioare, trebuie să identificați vulnerabilitatea și să o documentați în README. De asemenea, trebuie să furnizați un payload care să va ofere flag-ul, care este sub forma: **NAUGHTY_FLAG{<sir_de_caractere>}** (dacă un flag de această formă se regăsește în outputul programului, înseamnă că ați reușit).

4. Bonus - Shellcode - 20p

Generați un payload astfel încât să obțineți un shell folosind binarul **naughty**. Payloadul va fi consumat de la stdin.

Este suficient ca payloadul să funcționeze pentru cazuri cu  ASLR dezactivat. Pe scurt, ASLR este un mecanism de securitate care asigură că, la fiecare rulare, zonele de memorie ale procesului vor fi plasate la adrese random, greu de ghicit.

Pentru a dezactiva ASLR, puteți rula:

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Chiar și fără ASLR, adresele de stivă sunt destul de impredictibile și ar putea diferi de la o mașină la alta, deci este posibil ca payloadul vostru să nu funcționeze direct pe altă mașină. De aceea, important este să documentați în README logica din spatele payloadului.

Setup

Pentru dezvoltarea temei puteți folosi mașina virtuală de Linux de 32 de biți descrisă în secțiunea **Mașini virtuale** din pagina de resurse.

Trimitere și notare

Pentru descărcarea temei trebuie să intrați  aici.

Introduceți user-ul de cs.curs, iar după veți obține binarele pe care trebuie să le utilizați în rezolvarea temei.

Tema va trebui încărcată pe  vmchecker. Arhiva încărcată va fi o arhivă .zip care trebuie să conțină:

- Fișierul **README** care va conține explicații despre cum se sparg binarele **nice** și **naughty**, precum și flagurile găsite.
- Fișierele **nice_payload** și **naughty_payload** - vor conține payloadurile folosite pentru a obține flagurile din binarele respective.
- Fișierul **naughty_shellcode** care va conține payload-ul pentru a obține un shell - pentru binarul **naughty**
- Binarele **nice** și **naughty**

VMchecker-ul verifică doar existența fișierului **README** și a fișerelor de payload - nu se validează dacă corectitudinea flagurilor.

Verificarea corectitudinii celor 2 flag-uri va fi făcută de către asistenți (dacă ați obținut un flag de forma NICE_FLAG{<șir de caractere>} (respectiv NAUGHTY_FLAG{<șir de caractere>}) sigur e cel corect, nu există flaguri “false”).

Precizări suplimentare

- Este **interzisă** atacarea infrastructurii

Resurse ajutătoare

- Tool-uri disponibile care vă pot ajuta: gdb, IDA, radare, objdump, ghidra, pwntools, etc.
- **Laborator 9**: Analiza statică și dinamică a programelor.

Search

- Anunțuri
- Bune practici
- Calendar
- Catalog
- Feed RSS
- IOCLA Need to Know
- Reguli și notare
- Resurse utile

Cursuri

- Curs 01: Introducere
- Curs 01 - 02: Arhitectura sistemelor de calcul
- Curs 02 - 03: Arhitectura x86
- Curs 04: Reprezentarea datelor în sistemele de calcul
- Curs 05: Reprezentarea datelor în sistemele de calcul - C2
- Curs 06 - 07: Setul de instructiuni
- Curs 07: Declararea datelor
- Curs 08 - 09: Moduri de adresare
- Curs 09: Stiva
- Curs 10 - 11: Funcții
- Curs 12: C + asm
- Curs 13: Unelte, utilitare
- Curs 13 - 15: Buffer overflows, securitate
- Curs 16 - 17: Optimizări
- Curs 18 - 19: Virgulă flotantă

Laboratoare

- Laborator 01: Introducere
- Laborator 02: Toolchain
- Laborator 03: First baby steps
- Laborator 04: Rolul registrelor, adresare directă și bazată
- Laborator 05: Structuri, vectori. Operatii pe șiruri
- Laborator 06: Lucrul cu stiva
- Laborator 07: Apeluri de funcții
- Laborator 08: Interacțiunea C-assembly
- Laborator 09: Analiza statică și dinamică a programelor. GDB
- Laborator 10: Gestiunea bufferelor. Buffer overflow
- Laborator 11: Optimizări
- Laborator 12: Calcul în virgulă mobilă

Teme

- Tema 1 - Prefix AST
- Tema 2 - Stegano
- Tema 3 - Exploit ELFs, not elves

Table of Contents

- Tema 3 - Exploit ELFs, not elves
 - Enunț
 - 1. Analiza binarului - 20p
 - 2. Spargerea binarului - 40p
 - 3. Spargerea binarului v2 - 40p
 - 4. Bonus - Shellcode - 20p
 - Setup
 - Trimitere și notare
 - Precizări suplimentare
 - Resurse ajutătoare