Lab5

joi, 9 noiembrie 2023 18:35

1.
Candidate 1:NU
- Nu există o logică de generare a unor numere aparent aleatoare.
- doar afișează repetitiv valoarea seed-ului și nu modifică seed-ul într-un mod care să genereze o secvență aparent aleatoare.
returneaza 0 (converge spre 0)

Candidate 2: NU
- modifică seed-ul prin înmulțirea acestuia cu 1.5 la fiecare iterație, dar nu este suficient de complicat sau nepredictibil pentru a fi considerat un PRG adecvat.
- nu există nicio logică pentru a produce o secvență de numere care să pară aleatoare.

Candidate 3:NU
- afișează rezultatul unei operații de shiftare asupra seed-ului, dar nu modifică seed-ul
- nu e o implementare corespunzătoare a unui PRG

2. secrets.py - MAI JOS

EX.3

Ce problemă identificați în următoarele secvențe de cod

Example Language: Java
```
private static final long SEED = 1234567890;
public int generateAccountID() {
Random random = new Random(SEED);
return random.nextInt();
}
```

Example Language: PHP
```
function generateSessionID($userID) {
srand($userID);
return rand();
}
```

Care este CWE ID asociat scenariilor de mai sus si problemei pe care acestea o ridică?
🕐 Ce se întâmplă dacă nu se folosește același seed de fiecare dată, dar spațiul seed-urilor posibile este mic? Puteți găsi un CWE ID corespunzător acestui caz?
🕐 Căutați atacul identificat la punctul precedent în [5]. Identificați și aici o mențiune la seed?

🕐 Găsiți alte utilizări defectuoase ale PRG explicate în alte CWE-uri. Există CVE-uri corespunzătoare acestora?
🕐 Căutați înregistrări CVE care se referă la vulnerabilități în legătură cu PRG. Câte ați identificat ca fiind definite în acest an?

În primul rând, există o problemă în modul în care seed-ul este folosit în ambele exemple. Seed-ul ar trebui să fie unul secret și imprevizibil pentru a asigura o generare cât mai aleatoare a numerelor. În ambele cazuri, seed-ul este static și poate fi ușor de prezis sau de aflat, ceea ce slăbește securitatea generatorului de numere pseudo-aleatoare (PRG).

1. **Java Example:**
   - Problema: Seed-ul este static (`SEED = 1234567890`), ceea ce înseamnă că, dacă cineva descoperă acest seed, poate prezice toate valorile generate de generatorul pseudo-aleator.
   - CWE ID: CWE-338 (Use of Cryptographically Weak PRNG)

2. **PHP Example:**
   - Problema: Seed-ul este derivat dintr-un userID, care poate fi prezentat ca fiind imprevizibil. Cu toate acestea, dacă userID-ul nu este suficient de aleatoriu sau spațiul seed-urilor posibile este mic, un atacator poate ghici sau enumera seed-urile posibile.
   - CWE ID: CWE-330 (Use of Insufficiently Random Values)

În ceea ce privește întrebările suplimentare:

- Dacă spațiul seed-urilor posibile este mic, poate apărea o problemă de predicție a seed-ului. Acest lucru ar putea fi asociat cu CWE-330 sau CWE-338, în funcție de circumstanțe.

- Există multe CWE-uri asociate utilizării defectuoase a PRG-urilor, iar multe dintre ele pot fi găsite în CWE-330 și CWE-338.

EX.2 CODUL

```
import secrets
import string

# Generare parola
def generate_strong_password():
    alphabet = string.ascii_letters + string.digits + '.!$@'
    password = ''.join(secrets.choice(alphabet) for _ in range(10))
    return password
```

```
# Exemplu:
strong_password = generate_strong_password()
print("Parola generata:", strong_password)


#Generare string URL-safe:

url_safe_string = secrets.token_urlsafe(32)

# Exemplu de utilizare:
print("String URL-safe generat:", url_safe_string)


#Generare token hexazecimal:

hex_token = secrets.token_hex(32)

# Exemplu:
print("Token hexazecimal generat:", hex_token)


 #Verificare securizată a identității:**
def secure_compare(s1, s2):
    return secrets.compare_digest(s1, s2)

# Exemplu de utilizare:
string1 = "abc"
string2 = "abc"
result = secure_compare(string1, string2)
print("Sunt secventele identice?", result)


#Generare cheie fluidă binară:

fluid_key = secrets.token_bytes(16)

# Exemplu:
print("Cheie fluida generata:", fluid_key)


# #Stocare securizată a parolelor:
# from passlib.hash import bcrypt

# def hash_password(password):
#     salt = bcrypt.gensalt()
#     hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)
#     return hashed_password

# # Exemplu:
# user_password = "parola_secreta"
# hashed_password = hash_password(user_password)
# print("Parola hashată:", hashed_password)
```