MINISTERUL EDUCAŢIEI ŞI CERCETĂRII AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Ingineria Software și Automatică

Programul de studii: Securitatea informațională

Sistem integrat pentru cracking-ul hash-urilor

Practica de licență

Student(ă): _____ **Chihai Adrian, SI-211**

Coordonator întreprindere: _____ **Efros Viorel, manager de proiect TIC**

Coordonator de licență: _____ **Masiutin Maxim, asist.univ.**

Coordonator universitate: _____ **Bulai Rodica, lector universitar**

Chişinău 2025

UNIVERSITATEA TEHNICĂ A MOLDOVEI

FACULTATEA CALCULATOARE, INFORMATICĂ ȘI MICROELECTRONICĂ

DEPARTAMENTUL INGINERIA SOFTWARE ȘI AUTOMATICĂ

SECURITATEA INFORMAȚIONALĂ

AVIZ

la teza de licență

Titlul:_____

Studentul(a) _____Chihai Adrian_____ grupa ____SI_211_____

1. Actualitatea temei:

_____
_____
_____

2. Caracteristica tezei delicență:

_____
_____
_____

3. Analiza prototipului:

_____
_____
_____

4. Estimarea rezultatelor obținute:

_____
_____
_____

5. Corectitudinea materialului expus:

_____
_____
_____

6. Calitatea materialului grafic:

_____
_____

7. Valoarea practică a tezei:

_____
_____

8. Observații și recomandări:

_____
_____

9. Caracteristica studentului și titlul conferit:

_____
_____
_____

Lucrarea în formă electronica corespunde originalului prezentat către susținere publică.

Coordonatorul tezei de licență lector universitar, Bulai Rodica _____

(titlul științifico-didactic, titlul științific, semnătura, data, numele, prenumele)

CUPRINS

INTRODUCERE

În perioada curentă în care securitatea datelor este o preocupare majoră, strategiile de securizare a informațiilor și de recunoaștere a vulnerabilităților sunt esențiale pentru securitatea clienților și a organizațiilor. Odată cu dezvoltarea tehnologică atacurile cibernetice devin din ce în ce mai frecvente și periculoase, iar strategiile de asigurare necesită o îmbunătățire constantă. Printre tehnicile utilizate în securitatea cibernetică cea de brute force este utilizată atât în scopuri de protecție, cât și pentru testarea și analizarea vulnerabilităților din cadrele de date.

Această teză vizează investigarea în detaliu a procedurilor de brute force legate de spargerea hash-urilor, analizând viabilitatea și adecvarea acestor algoritmi în scenarii diferite. Hash-urile sunt o componentă crucială de asigurare a informațiilor, fiind utilizate pe scară largă pentru verificarea integrității fișierelor, autentificarea și protecția datelor. În cazul în care pentru hashing sunt folosiți algoritmi slabi sau dacă folosim parole compromise putem deveni victima unui astfel de atac. Prin intermediul acestei aplicații, utilizatorii vor putea testa astfel de atacuri și verifica nivelul de securitate al algoritmilor de hashing folosiți, precum și al credentialelor proprii. Pentru realizarea acestor simulări de atac, aplicația va oferi suport pentru diferiți algoritmi de hashing, printre care MD5, SHA-1 și SHA-256, unii dintre cei mai cunoscuți și care servesc drept fundație pentru alți algoritmii avansați.

Această lucrare nu se limitează doar la evidențierea riscurilor asociate atacurilor brute-force, ci își propune să ofere o soluție accesibilă și practică pentru testarea acestor tehnici. Prin dezvoltarea acestei aplicații, dorim să creăm un laborator interactiv, intuitiv și ușor de utilizat, conceput special pentru cei care nu sunt familiarizați cu aplicațiile de tip CLI (Command Line Interface). Astfel, utilizatorii vor putea explora și aplica conceptele de securitate cibernetică într-un mediu prietenos, care facilitează înțelegerea și experimentarea fără a necesita cunoștințe avansate de programare sau utilizare a terminalului.

1 ANALIZA ȘI DEFINIREA DOMENIULUI DE STUDIU

În contextul securității cibernetice, hashing-ul reprezintă o funcție matematică unidirecțională care transformă datele într-un șir de caractere cu o lungime fixă, numit hash. Acest proces este ireversibil, ceea ce înseamnă că informația originală nu poate fi recuperată din hash-ul generat

Hashing-ul este utilizat în numeroase aplicații de securitate, incluzând stocarea parolelor, semnăturile digitale și verificarea integrității fișierelor și documentelor. Prin conversia conținutului în valori unice, hashing-ul asigură protecția datelor împotriva accesului neautorizat și a alterărilor accidentale sau intenționate. Un exemplu comun este utilizarea hashing-ului în autentificarea utilizatorilor, unde sistemele compară hash-ul parolei introduse cu cel stocat în baza de date pentru a permite sau bloca accesul. [1]

Pentru a obține un nivel ridicat de securitate, hashing-ul se bazează pe trei componente esențiale:

- Cheia de intrare (input key) – datele originale care urmează să fie procesate.
- Funcția hash – algoritmul matematic care convertește datele într-o valoare unică.
- Tabela de hash (hash table) – structură de date care stochează valorile hash pentru referință și căutare rapidă.

Unul dintre avantajele principale ale hashing-ului este capacitatea sa de a garanta integritatea și securitatea datelor. De exemplu, în procesul de validare a fișierelor descărcate, se creează un hash unic, numit checksum, care servește la verificarea autenticității și integrității fișierului. Checksum-ul este o valoare numerică unică obținută din conținutul fișierului printr-un algoritm de hashing, iar orice modificare, chiar și un singur caracter, va genera un checksum diferit. La final, checksum-ul este recalculat și comparat cu valoarea inițială. Dacă valorile se potrivesc, acest lucru sugerează că fișierul a rămas neatins. O valoare diferită a checksum-ului final semnalează modificarea fișierului astfel, putem identifica o posibilă corupere sau o modificare, fie accidentală, fie deliberată, a datelor. [1]

Una dintre cele mai mari provocări este coliziunea – situația în care două valori de intrare diferite generează același hash. Pentru a minimiza acest risc, algoritmii moderni, cum ar fi SHA-256 și SHA-3, sunt proiectați pentru a reduce probabilitatea acestor coliziuni. [2] Ei folosesc diferse metode care minimizează probabilitatea întâlnirii acestui eveniment:

- Folosirea sării (Salting) - adăugarea unui șir aleatoriu la final înainte de face operațiunea de hash pentru a preveni coliziunile și atacurile de tip ”rainbow table”.
- Hashing iterativ (Key stretching) - aplicarea repetată a funcției de hash crește securitatea și reduce coliziunile în cazul parolelor.
- Verificare prin HMAC - pentru autentificare, utilizarea HMAC (Hash-based Message Authentication Code) ajută la evitarea coliziunilor și la protecția integrității mesajelor

În comparație cu criptarea hashing-ul este unidirecțional astfel textul, fișierul asupra căruia este aplicată operația de hashing nu poate fi decodificat în datele inițiale. Datorită acestui fapt hashing-ul este potrivit pentru a fi utilizat la autentificare și verificare. Comparativ cu criptarea care conține o cheie care poate decripta mesajul, o face perfectă spre folosire în protectiția și transmiterea datelor confidențiale.

În concluzie, hashing-ul este o tehnologie esențială pentru securitatea cibernetică, fiind utilizat în gestionarea parolelor, autentificare, protecția datelor și verificarea integrității fișierelor. Alegerea unui algoritm de hashing adecvat este esențială pentru a preveni atacurile cibernetice și pentru a asigura confidențialitatea și securitatea datelor digitale

1.1 Analiza și cercetarea soluțiilor existente

La moment platformele cele mai cunoscute care se ocupă de acest lucru sunt: “HashCat”, “Jhon the Ripper”, “Cain & Abel”, “OphCrack” .

Hashcat este un instrument open-source și este unul dintre cele mai puternice și eficiente pentru atacuri brute-force și dictionary attack asupra diverselor tipuri de hash-uri. Acesta este optimizat pentru procesare GPU, permițând efectuarea unor atacuri rapide și eficiente asupra unor baze mari de date de hash-uri, datorită vitezei procesoarelor grafice. Hashcat suportă multiple moduri de atac, inclusiv atacuri de dicționar, combinare, hibrid și chiar atacuri bazate pe reguli personalizabile. Hashcat dispune de următoarele funcții. [3]

- poate fi utilizat pe mai multe sisteme de operare
- multi-platform (suport OpenCL și CUDA)
- peste 150 algoritmi implementați

Commented [1]: Posibil abreviere

- utilizare redusă a resurselor
- sistem de benchmarking incorporat
- poate efectua dictionary attacks la peste 30 de protocoale

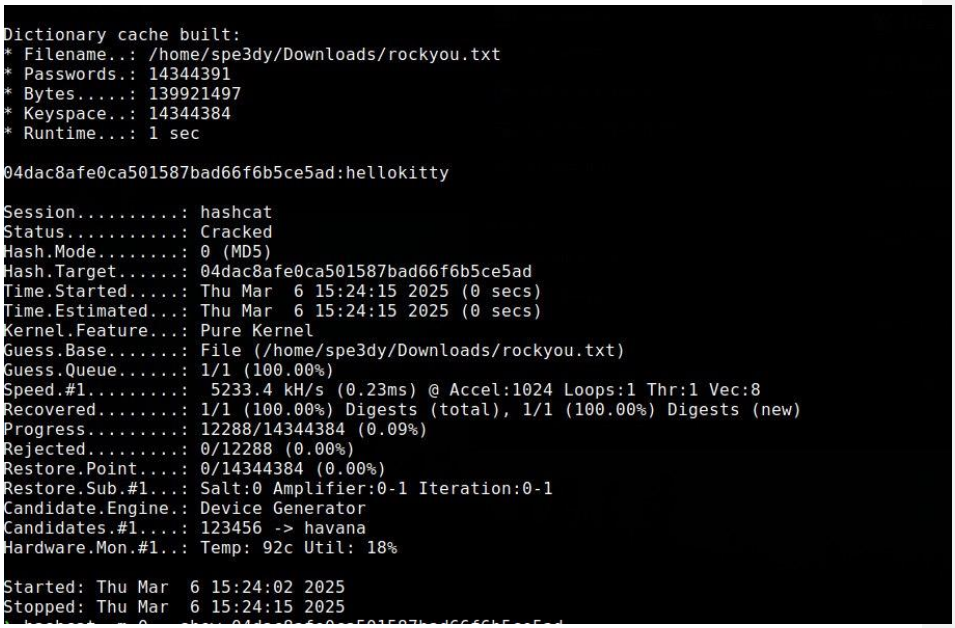

Figura 1.1 – Aplicația “Hashcat”

John the Ripper este un instrument versatil destinat testării parolelor și hash-urilor. Acesta poate funcționa atât în mod single-user, unde încearcă să ghicească parolele bazându-se pe modele comune, cât și în mod dictionary attack sau brute-force. John the Ripper este popular datorită capacității sale de a lucra pe multiple platforme și de a suporta mai multe formate de hash, inclusiv NTLM, LM, SHA, MD5 și multe altele. În plus, oferă posibilitatea de a personaliza atacurile prin reguli complexe. John the Ripper este optimizat pentru a utiliza eficient resursele sistemului, inclusiv suport pentru accelerare GPU și SIMD, ceea ce îmbunătățește semnificativ viteza de procesare.[4]

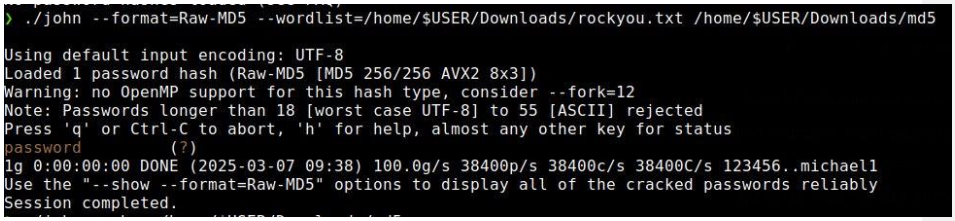

Figura 1.2 – Aplicația "John The Ripper"

Cain & Abel este un instrument multifuncțional de securitate ofensivă, utilizat nu doar pentru cracking-ul hash-urilor, ci și pentru sniffing-ul traficului de rețea, analiza protocolului VoIP și recuperarea parolelor ascunse sub asteriscuri în aplicațiile Windows. Cain & Abel este util în scenarii de testare a securității unde se analizează vulnerabilitățile din rețele și parolele utilizate în medii nesecurizate. Este instrumentul cel mai bun pentru atacurile de tip MITM, dar problema este că poate fi folosit doar windows și poate fi mai complicat pentru utilizatorii începători.[3] Funcțiile de care dispune acest instrument sunt.

- folosit pentru web-cracking
- ARP spoofing
- posibilitatea de înregistra conversații VoIP
- acesta poate sparge diverse forme de hașuri LM și NT, hash-uri IOS și PIX hash-uri RADIUS, parole RDP, MD2, MD4, MD5, SHA-1, SHA-2, RIPEMD-160, Kerberos 5, MSSQL, MySQL, Oracle și SIP

Commented [2]: Abreviere

Commented [3]: Abreviere

Commented [4]: Abreviere

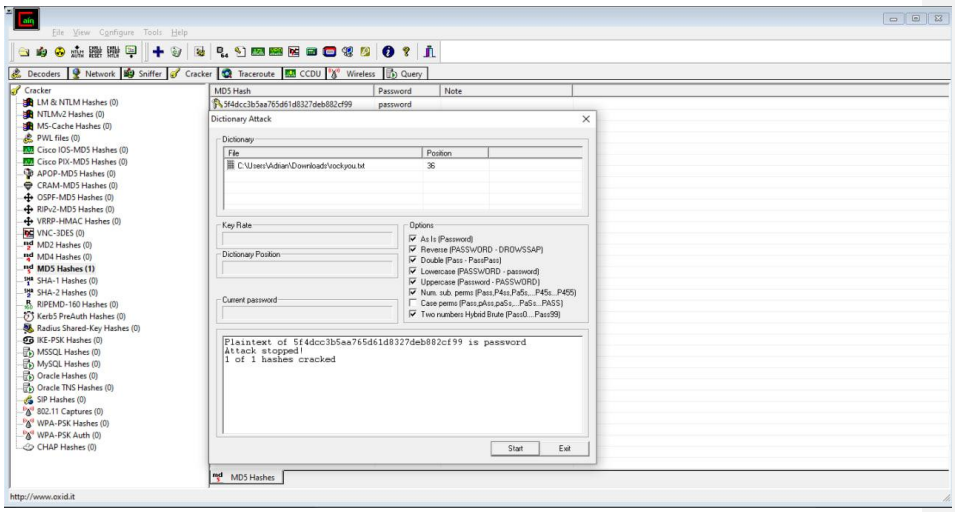

Figura 1.3 – Aplicația "Cain & Abel"

Ophcrack este un software specializat în cracking-ul hash-urilor LM și NTLM utilizate de sistemele de operare Windows. Acesta folosește o metodă eficientă bazată pe rainbow tables, ceea ce permite spargerea rapidă a parolelor slab securizate. Ophcrack este preferat pentru recuperarea parolelor din sistemele Windows, deoarece poate funcționa independent de sistemul de operare și poate descifra parole chiar și fără acces direct la sistemul țintă. Acest instrument este suficient de rapid și ușor pentru un utilizator care sparge parole pentru prima dată cu cunoștințe de bază de Windows și poate sparge majoritatea parole în câteva minute, pe majoritatea computerelor.[3] Acesta are următoarele funcții.

- disponibil pe mai multe sisteme de operare ca Windows, Linux/Unix, Mac OS
- poate sparge hashuri de tim LM și NTLM
- atac combinatoric
- atac brute-force
- atac direct

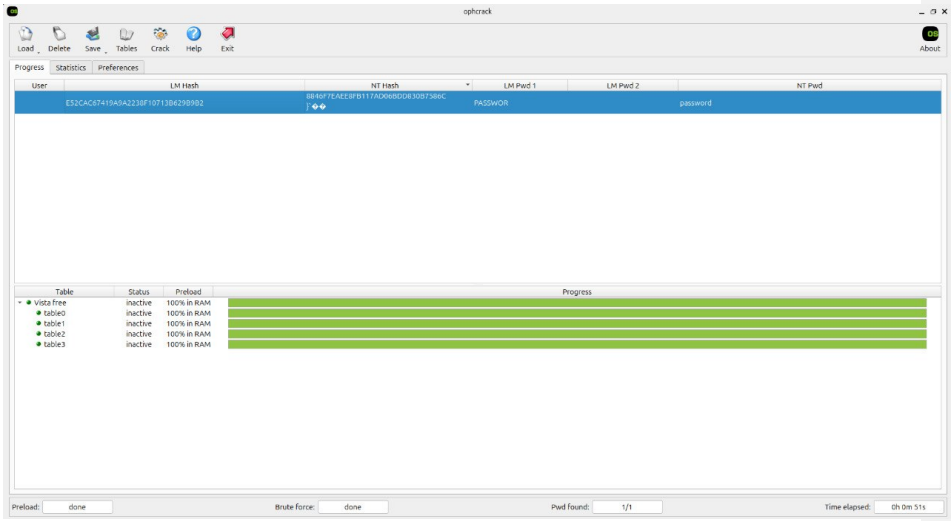

Figura 1.4 – Aplicația "Ophcrack"

1.2 Definirea scopului și obiectivul proiectului

Scopul acestui proiect este de a dezvolta o aplicație interactivă destinată testării și analizei vulnerabilităților algoritmilor de hashing prin atacuri brute-force. Aplicația este concepută pentru utilizatorilor interesați de securitatea cibernetică și are ca obiectiv principal oferirea unui mediu de testare și învățare accesibil, care combină atât aspectele teoretice, cât și aplicațiile practice ale atacurilor asupra algoritmilor de hashing.

Din cauza unor posibile limitări hardware, aplicația va rula exclusiv pe procesor (CPU), fără sup pentru accelerare GPU. Această restricție poate afecta viteza și eficiența atacurilor brute-force, deoarece GPU-urile sunt semnificativ mai rapide în procesarea masivă a datelor. Ca urmare, timpul necesar pentru spargerea unui hash va fi mai mare comparativ cu soluțiile care utilizează accelerare hardware. Totuși, aplicația va optimiza execuția pe CPU pentru a maximiza performanța în limitele resurselor disponibile.

Commented [5]: De adaugat abreviere

Obiectivele principale ale aplicației includ:

- Realizarea unei interfețe intuitive pentru a oferi o experiență bună utilizatorilor
- Identificarea tipului de hash utilizat.
- Generarea hash-urilor pentru diverse parole și texte introduse de utilizator.
- Verificarea unui hash în wordlist-uri compromise, care conțin posibile credențiale expuse, facilitând astfel testarea securității acestora.
- Încărcarea wordlist-urilor suplimentare pentru a oferi utilizatorilor o gamă mai largă de credențiale și date suplimentare.
- Efectuarea atacurilor de tip brute-force, încercând descifrarea unui hash prin testarea sistematică a combinațiilor posibile.
- Realizarea unui raport după finisarea atacului brute-force cu referire la timpul total de execuție, numărul de combinații testate, rata medie de testare pe secundă (hashes per second) și rezultatul final (dacă hash-ul a fost spart sau nu)
- Vizionarea în timp real a hash-ului testat și a rezultatului curent, oferind utilizatorilor posibilitatea de a monitoriza parolele încercate și de a analiza dinamica procesului de cracking.
- Posibilitatea de a opri, pune pe pauză și relua atacul fără a pierde progresul testării, permițând utilizatorilor să gestioneze eficient resursele hardware.

2 PROIECTAREA ȘI ARHITECTURA SISTEMUL INFORMATIC

În timpul proiectării arhitecturii unui sistem informatic se definesc structuri fundamentale, soluții software și scalabile. Proiectarea și arhitectura sistemului destinat aplicației de brute-force pentru cracking-ul hash-urilor va facilita înțelegerea cerințelor funcționale și non-funcționale. Pentru modelarea sistemului informatic se vor utiliza tehnici avansate care vor simula entitățile și relațiile dintre ele, vor reprezenta fluxul datelor și vor defini structurile software necesare pentru o implementare optimizată și eficientă. Proiectarea unui sistem informatic constă în arhitectura sistemului, care cuprinde entitățile, modulele, interfețele

Pentru proiectarea unui sistem informatic sunt disponibile mai multe opțiuni

- Metodele structurate, ele implică o abordare sistematică, divizând sistemul în module independente pentru a facilita înțelegerea și întreținerea.[5]

- Metode orientate pe obiect, ele concentrează pe modelarea sistemului în jurul obiectelor, care combină datele și comportamentul, promovând reutilizarea codului și întreținerea eficientă.[5]

- Metodele iterative și incrementale, ele presupun dezvoltarea sistemului în etape succesive, permițând testarea și evaluarea constantă a componentelor, reducând riscurile și identificând problemele mai devreme în procesul de dezvoltare.[5]

- Metode funcționale, ele se axează pe utilizarea funcțiilor matematice pure pentru a modela comportamentul sistemului, eliminând efectele secundare și mutabilitatea stărilor, ceea ce crește predictibilitatea și fiabilitatea sistemului.[5]

În plus, utilizarea unui limbaj de modelare precum UML (Unified Modeling Language) permite o reprezentare clară și standardizată a structurii și comportamentului sistemului. Diagramele UML contribuie la o mai bună comunicare între dezvoltatori, utilizatori și alte părți interesate, facilitând atât înțelegerea, cât și implementarea soluției.

2.1 Cerințele funcționale ale sistemului

Pentru a putea realiza un sistem informatic, este necesară stabilirea inițială a cerințelor funcționale. Acestea definesc acțiunile pe care sistemul trebuie să le îndeplinească pentru a asigura funcționalitatea dorită. Printre cerințele funcționale ale sistemului se numără: introducerea unui hash și selectarea metodei de brute-force, detectarea automată a tipului de hash introdus, posibilitatea de a încărca un wordlist sau un rainbow table pentru optimizarea procesului, vizualizarea progresului în timp real și salvarea unui raport detaliat la finalizarea sau oprirea procesului.

Fiecare dintre aceste funcționalități a fost detaliată, subliniind modul în care utilizatorii vor interacționa cu sistemul și cum se va realiza procesarea datelor. Introducerea unui hash este punctul de start al aplicației, permițând utilizatorilor să introducă manual un hash pe care doresc să-l spargă. În ceea ce privește determinarea tipului de hash, aplicația va analiza formatul și lungimea acestuia pentru a identifica algoritmul utilizat la generarea sa. Acest proces ajută utilizatorii să aleagă cea mai eficientă metodă de atac, optimizând timpul de procesare.

Un aspect esențial al sistemului este posibilitatea de a încărca wordlist-uri sau rainbow tables. Această funcționalitate permite utilizatorilor să utilizeze seturi de date preexistente pentru a accelera procesul de spargere a hash-urilor, în loc să genereze combinații complet aleatorii. De asemenea, aplicația trebuie să suporte atât wordlist-uri standard, cât și personalizate, oferind utilizatorilor flexibilitate în strategia de atac.

Un alt element important este vizualizarea progresului în timp real. Utilizatorii trebuie să poată urmări informații despre numărul de combinații testate, rata de procesare (hash-uri/secundă), timpul estimat pentru finalizarea procesului și rezultatele intermediare. Aceste date vor fi afișate într-o interfață grafică intuitivă, facilitând monitorizarea și ajustarea parametrilor de atac, dacă este necesar.

După finalizarea sau oprirea procesului, utilizatorii vor putea generea un raport detaliat. Acest raport trebuie să includă hash-ul inițial, parola descoperită (dacă a fost identificată), metoda utilizată, durata totală a procesului și alte statistici relevante. În cazul în care procesul a fost întrerupt, raportul trebuie să reflecte progresul realizat până în acel moment, astfel încât utilizatorii să poată relua atacul de unde a fost întrerupt.

Pentru a asigura o experiență de utilizare optimă, toate aceste funcționalități vor fi integrate într-o interfață de utilizator (UI) intuitivă și ușor de navigat. O interfață bine concepută, bazată pe principii de design centrate pe utilizator, facilitează accesul rapid la informații și îmbunătățește satisfacția utilizatorilor. Pentru a atinge acest obiectiv, este esențial să se acorde atenție detaliilor și să se pună în practică principii de design bine stabilite.[6] Astfel, utilizatorii vor putea interacționa eficient cu sistemul, beneficiind de o experiență plăcută și productivă.

2.2 Cerințele non-funcționale ale sistemului

Cerințele nefuncționale se referă la atributele calitative ale unui sistem software, influențând aspecte precum performanța, securitatea, scalabilitatea și experiența generală a utilizatorului. Aceste cerințe, deși nu descriu funcționalități specifice, sunt esențiale pentru asigurarea eficienței și fiabilității sistemului astfel ele pot impune anumite restricții sau contrângeri asupra sistemului.[7]

Pentru ca sistemul să fie performant trebuie să fie capabil să proceseze hash-urile într-un mod eficient, pentru acest lucru e nevoie de optimizarea algoritmilor pentru a minimiza timpul de procesare. Optimizarea algoritmilor eficientizează și gestionarea resurselor hardware pentru a nu supraîncărca sistemul utilizatorului. De asemenea, scalabilitatea este crucială pentru a permite sistemului să se adapteze la volume mari de date, cum ar fi wordlist-urile sau tabelele rainbow de dimensiuni considerabile. Aceste aspecte sunt fundamentale în proiectarea și implementarea unui sistem informatic robust și eficient.

Pentru a asigura fiabilitatea și disponibilitatea sistemului, este esențial ca acesta să funcționeze continuu și fără erori, menținând o disponibilitate ridicată. Implementarea mecanismelor de backup și recuperare este crucială pentru prevenirea pierderii datelor și pentru asigurarea continuității operațiunilor. În plus, sistemul trebuie să fie rezilient la diverse tipuri de atacuri sau defecțiuni, menținându-și funcționalitatea chiar și în condiții adverse.

Sistemul trebuie să asigure un nivel ridicat de fiabilitate și disponibilitate, garantând o funcționare continuă și stabilă în mare parte din timp, fără erori critice care să afecteze utilizatorii. Pentru a preveni pierderea datelor în cazul unor întreruperi neașteptate, aplicația va include un

mecanism de salvare automată a progresului utilizatorului, permițând reluarea activităților din punctul în care au fost întrerupte, asigurând astfel o experiență fluidă și neîntreruptă.

Aplicația trebuie să fie proiectată astfel încât să permită integrarea unor noi algoritmi de brute-force, oferind flexibilitate în extinderea funcționalităților fără modificări majore în structura existentă. De asemenea, codul va fi organizat modular, asigurând o arhitectură clară și bine definită, ceea ce va facilita atât întreținerea, cât și actualizarea sistemului pe termen lung.

Pentru a asigura corectitudinea și funcționalitatea algoritmilor implementați, aplicația va include teste unitare și de integrare, verificând astfel atât componentele individuale, cât și interacțiunea dintre ele. De asemenea, testarea va fi realizată în diferite scenarii, simulând diverse condiții de utilizare, pentru a evalua eficiența tehnicilor brute-force și impactul acestora asupra performanței sistemului. Această abordare va permite identificarea și remedierea eventualelor erori înainte de implementarea finală, garantând o funcționare stabilă și optimizată.

Aplicația va fi dezvoltată și utilizată cu respectarea principiilor etice, evitând orice utilizare abuzivă sau ilegală a tehnicilor brute-force implementate. Scopul principal al sistemului este de a contribui la cercetare și securitate cibernetică, nu de fi folosită pentru posibile activități malițioase, iar utilizatorii vor fi informați cu privire la utilizarea corectă și responsabilă a software-ului.

2.3 Modelarea comportamentală a sistemului

Descrierea comportamentală a sistemului, conform standardului UML, se realizează prin diferite tipuri de diagrame care pun accentul pe aspecte specifice ale interacțiunii dintre componentele sale. Aceste diagrame oferă o perspectivă transparentă asupra modului în care funcționează aplicația, permițând echipei de dezvoltare să înțeleagă procesele interne și modul în care utilizatorii interacționează cu sistemul.

Pentru aplicația de spargere a hash-urilor prin forța brută, următoarele diagrame UML sunt utilizate pentru modelarea comportamentală:

- Diagrama cazurilor de utilizare (Use case diagram) - prezintă o imagine de ansamblu a sistemului, subliniind actorii și funcționalitățile principale.
- Diagrama de activități (Activity diagram) - descrie fluxurile de date și etapele necesare pentru finalizarea unui proces.
- Diagrama de stări (Statechart diagram) - prezintă tranzițiile și schimbările de stare ale componentelor sistemului.
- Diagrama de secvență (Sequence diagram) - reprezintă interacțiunile dintre componente într-o anumită secvență temporală.
- Diagrama de colaborare (Collaboration diagram) - ilustrează fluxurile de mesaje și conexiunile dintre componentele sistemului.

Aceste diagrame oferă o reprezentare cuprinzătoare și intuitivă a comportamentului sistemului, facilitând identificarea și optimizarea proceselor esențiale pentru funcționalitatea aplicației. Prin intermediul diagramelor comportamentale, dezvoltatorii pot vizualiza și analiza secvențele de acțiuni, interacțiunile între obiecte și evoluția stărilor acestora în timp.

2.3.1 Generalitățile sistemului informatic

Sistemul informatic dezvoltat pentru analiza și implementarea tehnicilor brute-force în procesul de cracking al hash-urilor urmărește să ofere o soluție eficientă și intuitivă pentru utilizatori. Acesta este structurat modular, incluzând o interfață grafică pentru gestionarea procesului, un motor de procesare dezvoltat în Golang pentru efectuarea atacurilor brute-force, precum și componente pentru administrarea listelor de parole și generarea rapoartelor.

Pentru a asigura o proiectare clară și o înțelegere detaliată a interacțiunii dintre utilizatori și sistem, utilizăm diagramele Use Case. Ea oferă o reprezentare vizuală a tuturor scenariilor în care utilizatorul poate interacționa cu sistemul, evidențiind acțiunile principale și fluxurile de execuție.

Diagramele Use Case sunt utile deoarece:

- Clarifică cerințele funcționale ale aplicației.

- Evidențiază interacțiunile dintre utilizatori și sistem, ajutând la definirea comportamentului general al aplicației.
- Simplifică procesul de comunicare între echipa de dezvoltare și utilizatorii finali, facilitând înțelegerea cerințelor și a funcționalităților.
- Ajută la testarea și validarea sistemului, oferind o bază pentru verificarea scenariilor de utilizare.

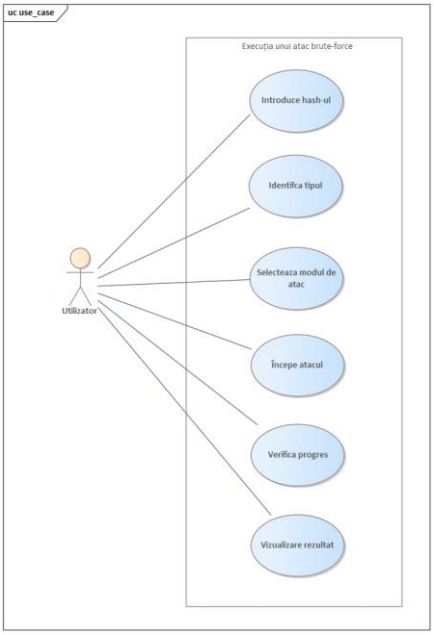

Figura 2.1 – Funcționalități generale ale sistemului

Diagrama din figura 2.1 reprezintă cazurile pentru funcționalitățile generale ale aplicației de cracking hash prin atac brute-force. Aceasta oferă o viziune de ansamblu asupra procesului prin care utilizatorul poate executa un astfel de atac.

Utilizatorul poate introduce un hash pe care dorește să-l spargă, iar sistemul identifică automat tipul acestuia pentru a selecta metoda adecvată. După identificarea tipului de hash, utilizatorul poate alege modul de atac, optând între diferite strategii, cum ar fi forța brută pură sau metode

bazate pe dicționar. Odată selectată metoda, utilizatorul inițiază atacul, iar sistemul începe procesul de spargere a hash-ului. Pe parcurs, utilizatorul are posibilitatea de a verifica progresul atacului în timp real, urmărind eventualele rezultate parțiale. La finalul procesului, aplicația afișează rezultatul atacului, indicând dacă hash-ul a fost spart sau dacă încercarea a eșuat.

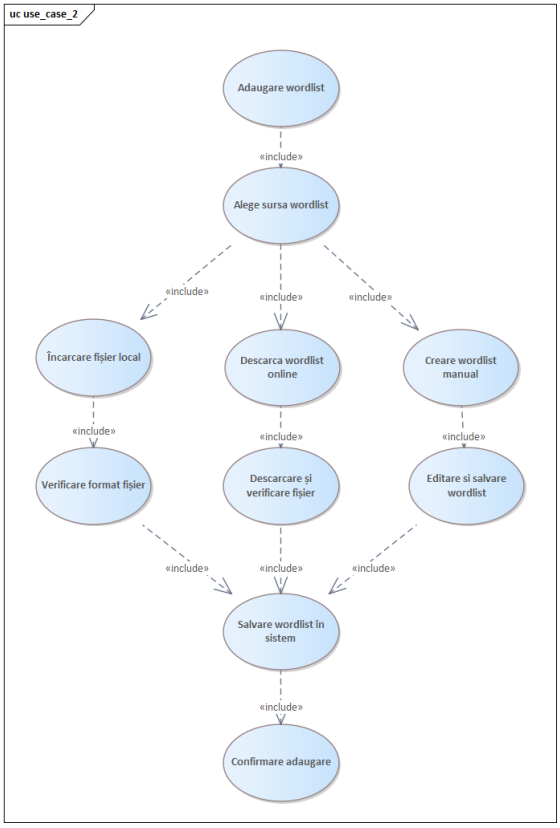

Figura 2.2 - Adăugarea unui wordlist

Diagrama din figura 2.2 reprezintă procesul de adăugare a unui wordlist în aplicație. Utilizatorul va avea avea 3 metode de adăugare a unui wordlist, ele sunt următoarele: încărcarea unui fișier

local, descărcarea unui wordlist online sau crearea manuală a unuia nou. În cazul încărcării unui fișier sau descărcării unui wordlist, sistemul verifică automat formatul acestuia pentru a se asigura că este compatibil. Dacă fișierul este valid, acesta este salvat și poate fi folosit în atacurile brute-force. În cazul în care utilizatorul optează pentru crearea manuală a unui wordlist, acesta poate edita și salva conținutul direct în aplicație. La final, sistemul confirmă adăugarea wordlist-ului, iar acesta devine utilizabil în procesul de spargere a hash-urilor.

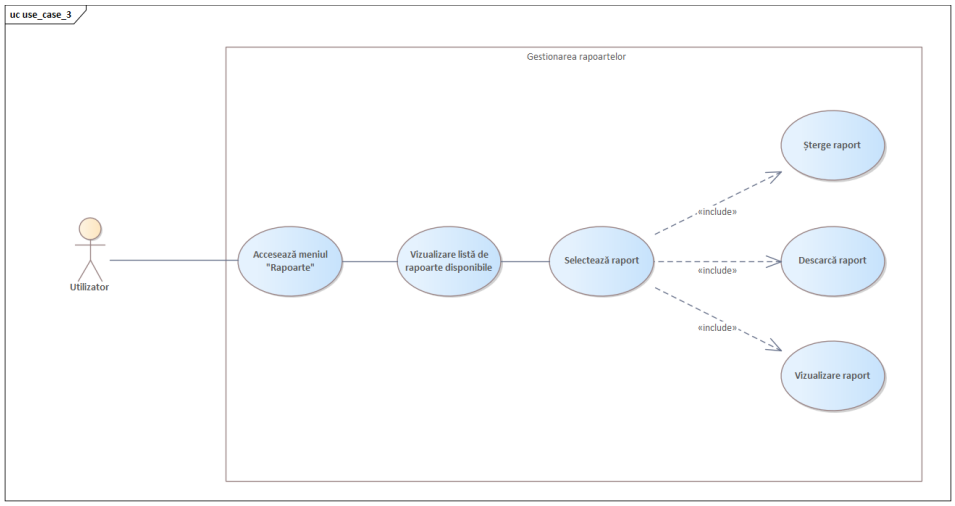

Figura 2.3 – Interacțiunea utilizatorului cu modulul de rapoarte

În diagrama 2.3 este ilustrată interacțiunea utilizatorului cu modulul de gestionare a rapoartelor din cadrul aplicației. Utilizatorul accesează secțiunea dedicată rapoartelor generate în sesiunile anterioare de cracking, moment în care sistemul afișează o listă cu documentele disponibile. Asupra fiecărui raport selectat, utilizatorul poate efectua una dintre cele trei acțiuni: „Vizualizare raport”, „Descarcă raport” sau „Șterge raport”.

2.3.2 Fluxul operațional al aplicației

În timpul dezvoltării aplicației, diagramele de activitate sunt importante pentru modelarea și înțelegerea fluxului operațional al sistemului. Aceste diagrame oferă o reprezentare vizuală a succesiunii de acțiuni și decizii din cadrul proceselor aplicației, facilitând o perspectivă clară asupra funcționării acesteia.

Importanța diagramelor de activitate în cadrul aplicației:

- Clarificarea fluxului de procese: Diagramele de activitate permit identificarea și reprezentarea secvențială a activităților, evidențiind tranzițiile și condițiile dintre acestea.

- Detectarea potențialelor blocaje sau ineficiențe: Prin analiza diagramelor de activitate, se pot identifica zonele susceptibile la blocaje sau ineficiențe în fluxul operațional, oferind oportunități de optimizare.

- Facilitarea comunicării între echipă: O reprezentare vizuală clară a proceselor ajută la colaborarea membrilor echipei de dezvoltare și la asigurarea unei înțelegeri comune a funcționalităților sistemului.

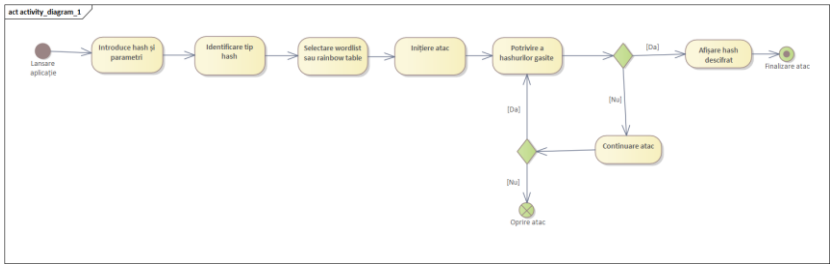

Figura 2.4 – Fluxul procesului de cracking

În figura 2.4 este reprezentat fluxul procesului pe care îl urmează aplicația pentru a face brute-force unui hash. Utilizatorul introduce un hash, iar sistemul identifică automat tipul de hash sau utilizatorul poate specifica tipul manual în parametri, acești pași sunt esențiali pentru alegerea metodei de atac.

După selecția strategiei (wordlist sau rainbow table), aplicația inițiază atacul și compară hash-urile generate cu cel introdus. Dacă se găsește o potrivire, parola este descifrată și afișată. În caz contrar, utilizatorul poate continua atacul sau opri procesul.

Fluxul se finalizează fie prin afișarea parolei descifrate, fie prin oprirea atacului în lipsa unui rezultat, oferind utilizatorului flexibilitate în alegerea metodei potrivite.

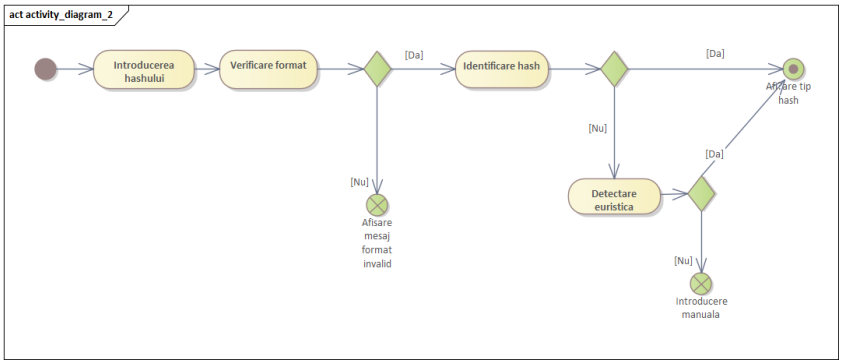

Figura 2.5 – Procesul de identifcare al unui hash

În figura 2.5 este ilustrat fluxul procesului de identificare a unui hash. Aplicația verifică automat dacă formatul hash-ului este valid, permițând continuarea procesului doar în cazul în care acesta este recunoscut. Dacă tipul hash-ului este identificat, utilizatorului îi este afișată informația corespunzătoare. În situația în care identificarea automată eșuează, sistemul recurge la o metodă de detectare euristică. Dacă nici această metodă nu oferă un rezultat clar, utilizatorul are posibilitatea de a introduce manual tipul hash-ului.

2.3.3 Modelarea comportamentului sistemului

Commented [6]: Revizuit introducere

În timpul dezvoltării aplicației, diagramele de stare sunt esențiale în modelarea și înțelegerea comportamentului dinamic al sistemului. Aceste diagrame oferă o reprezentare vizuală a diferitelor stări prin care poate trece o componentă a sistemului și modul în care tranzițiile între aceste stări sunt declanșate de evenimente specifice.

Importanța diagramelor de stare în cadrul aplicației:

- Clarificarea comportamentului sistemului: Diagramele de stare permit identificarea și reprezentarea secvențială a stărilor și tranzițiilor dintre acestea, evidențiind condițiile și evenimentele care determină schimbările de stare. Aceasta ajută la o înțelegere profundă a modului în care sistemul reacționează la diferite stimuli și situații.

- Identificarea comportamentelor complexe și a scenariilor de eroare: Prin analiza diagramelor de stare, se pot identifica comportamente complexe, bucle infinite sau stări neacoperite, oferind oportunități de optimizare și asigurând o funcționare robustă a sistemului.

- Facilitarea comunicării între membrii echipei: O reprezentare vizuală clară a comportamentului sistemului ajută la alinierea membrilor echipei de dezvoltare și la asigurarea unei înțelegeri comune a funcționalităților și reacțiilor sistemului în diverse situații.

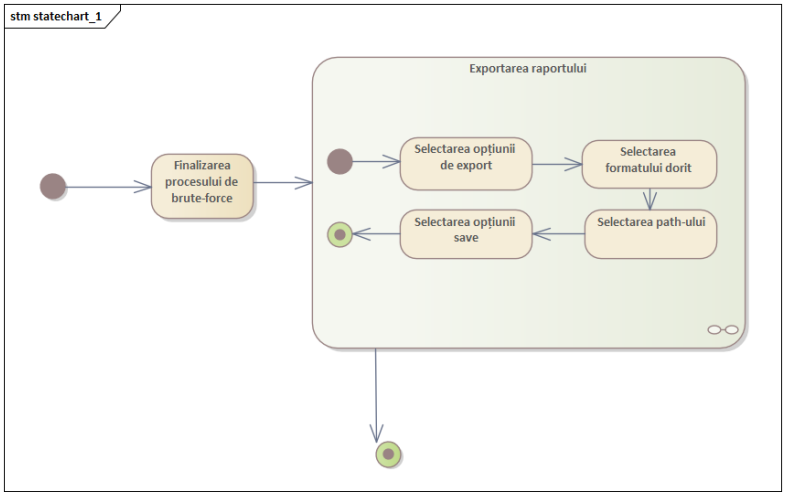

Figura 2.6 – Exportarea raportului

Diagrama din figura 2.6 modelează comportamentul sistemului în etapa de exportare a raportului după finalizarea unui atac brute-force. După încheierea procesului de cracking, utilizatorul selectează opțiunea de export și alege formatul dorit, urmat de specificarea locației unde fișierul va fi salvat. Diagrama evidențiază succesiunea stărilor interne ale sistemului și condițiile logice necesare pentru a ajunge la finalizarea operației. Totodată, subliniază modul secvențial în care se parcurg pașii de configurare a exportului.

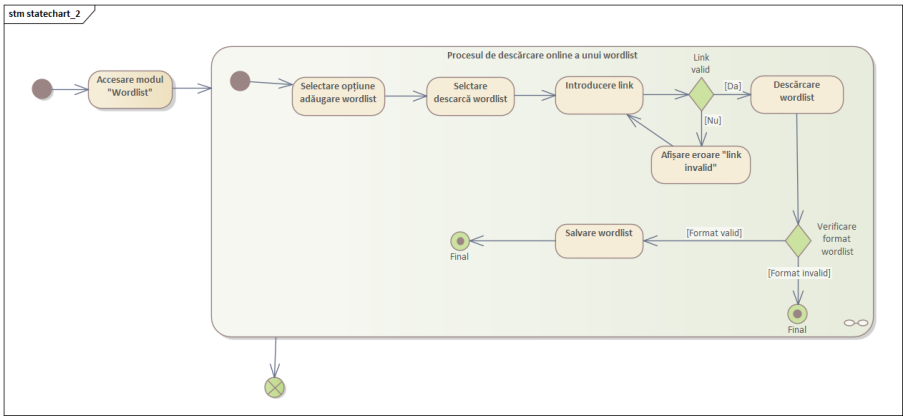

Figura 2.7 – Descărcarea și validarea unui wordlist din sursă online

Diagrama din figura 2.7 simulează fluxul de stări prin care trece aplicația atunci când utilizatorul dorește să adauge un wordlist dintr-o sursă online. După introducerea linkului, sistemul verifică validitatea acestuia, permițând continuarea doar dacă link-ul este valid. După descărcare, fișierul este verificat din punct de vedere al formatului, iar dacă este valid, este salvat în sistem. Diagrama evidențiază atât traseul ideal, cât și gestionarea cazurilor de eroare (link invalid, format greșit).

2.3.4 Descrierea fluxurilor de interacțiune utilizator–sistem

Diagramele de secvență oferă o reprezentare clară și detaliată a modului în care componentele unui sistem software interacționează în timp pentru a îndeplini anumite funcționalități. Acestea evidențiază ordinea mesajelor schimbate între obiecte, subliniind fluxurile de control și de date care apar între utilizator și subsistemele aplicației. Prin utilizarea diagramelor de secvență, se poate înțelege exact comportamentul sistemului în diverse scenarii de utilizare, contribuind astfel la validarea logicii de implementare și la facilitarea comunicării între membrii echipei de dezvoltare.[8] În plus, ele servesc ca documentație valoroasă pe termen lung, oferind o bază solidă pentru mentenanța și extinderea ulterioară a aplicației.[9]

În cadrul lucrării, diagramele de secvență sunt aplicate pentru a reprezenta procese precum inițierea și execuția unui atac de tip brute-force, exportul raportului după finalizarea atacului și identificarea automată a tipului de hash. Prin aceste reprezentări vizuale, se evidențiază succesiunea mesajelor transmise între obiecte și fluxul de control asociat fiecărui scenariu, contribuind la clarificarea cerințelor funcționale și la detectarea timpurie a eventualelor probleme de proiectare.

Astfel, utilizarea diagramelor de secvență în această teză nu doar că sprijină procesul de dezvoltare și documentare a aplicației, dar și îmbunătățește comunicarea în cadrul echipei de proiect, asigurând o înțelegere comună și detaliată a mecanismelor interne ale sistemului.

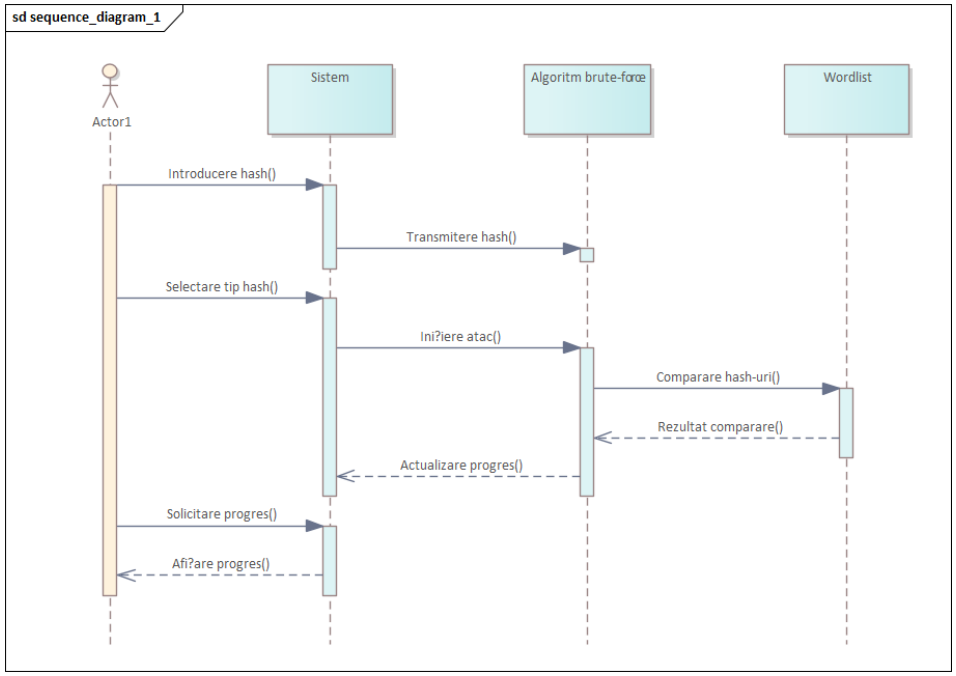

Figura 2.8 - Inițierea și execuția unui atac de tip brute-force

În figura 2.8 este simulat procesul de inițiere și execuție a unui atac de tip brute-force. Utilizatorul introduce hash-ul și selectează tipul acestuia, interacționând cu sistemul central care transmite datele către modulul de algoritm. Acesta realizează procesul de comparare folosind un wordlist și actualizează progresul, care este transmis înapoi spre sistem și afișat utilizatorului în timp real. Această diagramă evidențiază clar succesiunea pașilor implicați în atacul brute-force, subliniind interacțiunile dintre componentele sistemului și utilizator.

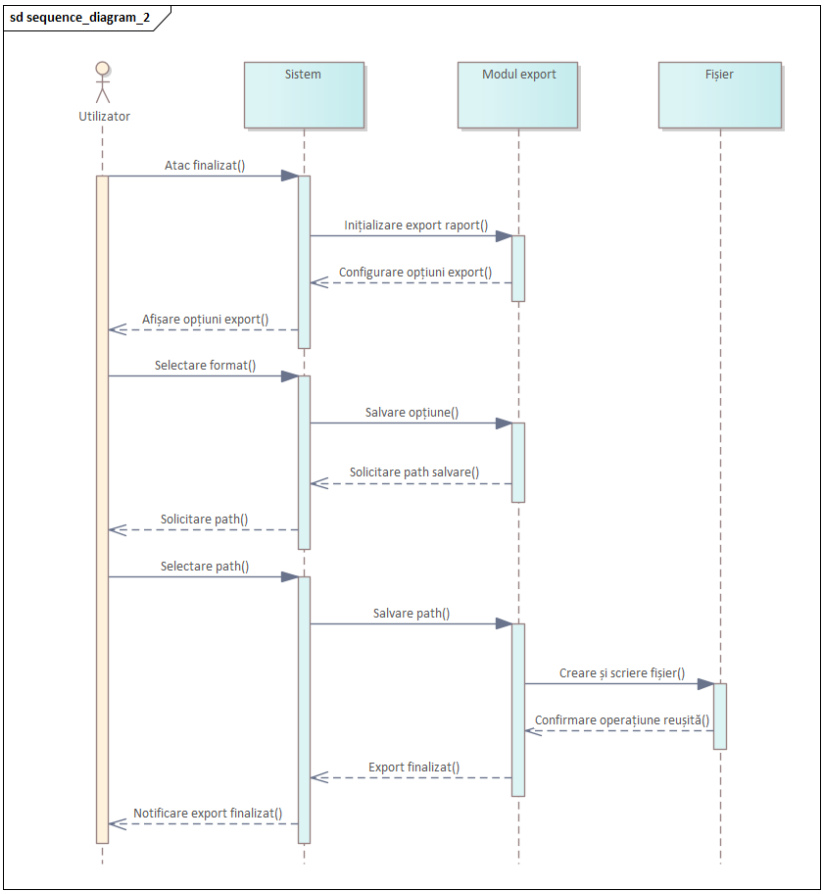

Figura 2.9 - Exportul raportului după finalizarea atacului

Figura 2.9 descrie secvența asociată exportului raportului după finalizarea atacului. Sistemul inițiază procesul de export prin comunicarea cu modulul dedicat, care, la rândul său, configurează opțiunile de export pe baza alegerii utilizatorului. După selectarea formatului și a locației de salvare, fișierul este generat, scris și confirmat ca fiind salvat cu succes, iar utilizatorul este notificat. Această diagramă subliniază importanța interacțiunii dintre utilizator și sistem în configurarea și finalizarea cu succes a procesului de export al raportului.

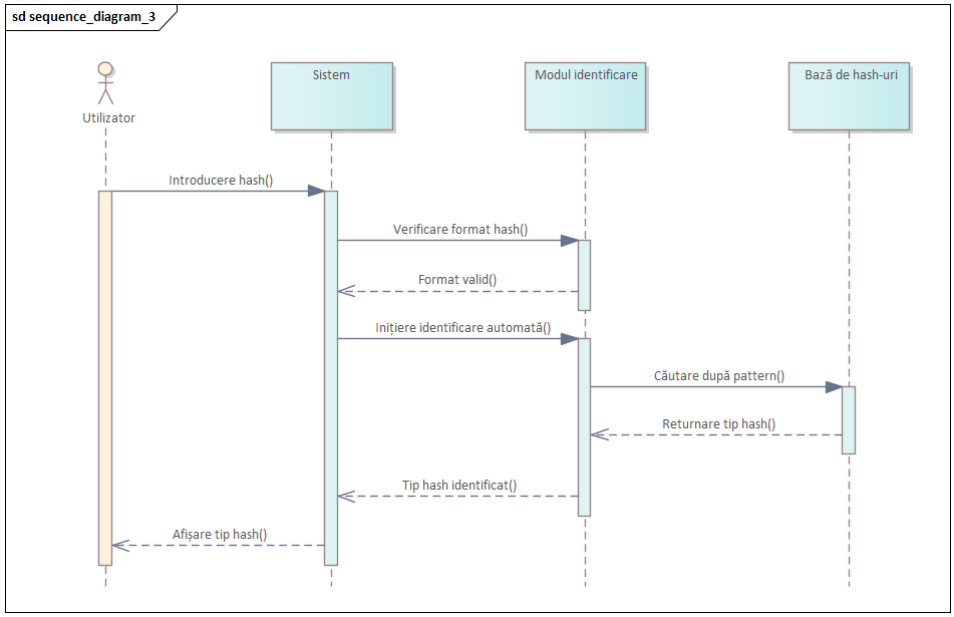

Figura 2.10 – Identificarea algorimtului de hash

Figura 2.10 prezintă interacțiunea dintre utilizator și sistem în procesul de identificare automată a tipului de hash. După introducerea valorii hash, sistemul apelează modulul de identificare care verifică formatul și, dacă este valid, inițiază o căutare euristică în baza de hash-uri pentru a determina tipul acestuia. În funcție de rezultat, sistemul comunică utilizatorului tipul identificat. Această diagramă evidențiază eficiența sistemului în automatizarea procesului de

identificare a tipului de hash, reducând astfel efortul utilizatorului și minimizând posibilitatea erorilor umane.

2.3.5 Utilizarea diagramelor de comunicare pentru evidențierea fluxurilor informaționale

Diagramele de comunicare (cunoscute și ca diagrame de colaborare) reprezintă un tip de diagramă UML utilizat pentru a evidenția interacțiunile dintre obiecte în cadrul unui sistem, punând accent pe schimbul de mesaje într-un context structural. Aceste diagrame sunt utile în analiza și proiectarea sistemelor software, deoarece permit înțelegerea clară a modului în care entitățile colaborează pentru a îndeplini o funcționalitate specifică [10].

În cadrul proiectului, au fost realizate două astfel de diagrame, fiecare reflectând un scenariu distinct de interacțiune între componentele sistemului dezvoltat. Aceste diagrame oferă o imagine clară a succesiunii mesajelor și a responsabilităților fiecărei entități, asigurând astfel o bază solidă pentru implementarea logicii aplicației.

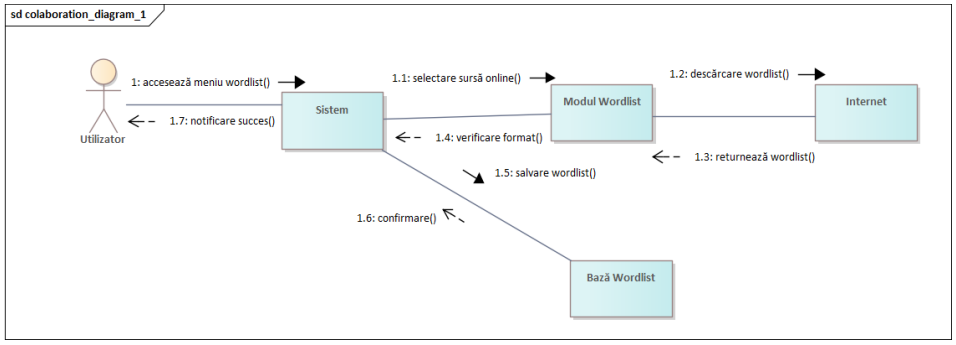

Figura 2.11 – Descărcarea și salvarea unui wordlist

În cadrul figurii 2.11 este prezintată diagrama de comunicare corespunzătoare procesului de selectare și salvare a unei liste de cuvinte (wordlist) dintr-o sursă online. Interacțiunea debutează cu inițiativa utilizatorului de a accesa meniul dedicat gestionării wordlist-urilor, moment în care sistemul declanșează o secvență de mesaje care implică Modulul Wordlist și o resursă externă din Internet. După ce lista este descărcată, aceasta este verificată din punct de vedere al formatului,

urmând ca, în cazul în care este validă, să fie salvată în baza de date. Procesul se încheie cu confirmarea operațiunii și notificarea utilizatorului cu privire la succesul acțiunii.

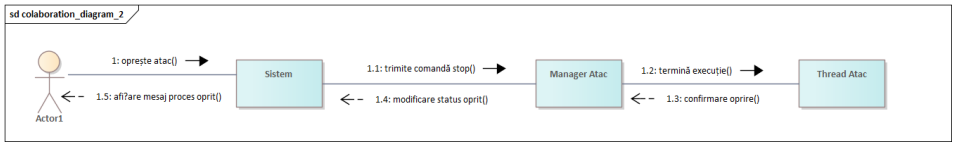

Figura 2.12 – Oprirea unui atac în execuție

Figura 2.12 ilustrează un scenariu distinct, cel al opririi unui atac activ. Actorul principal al sistemului inițiază oprirea printr-un mesaj transmis către sistem, care, la rândul său, propagă comanda de oprire către Managerul de Atac. Acesta comunică cu firul de execuție dedicat atacului, solicitând finalizarea acestuia. După confirmarea opririi, sistemul actualizează starea internă corespunzătoare și notifică utilizatorul cu un mesaj care atestă oprirea completă a procesului. Diagrama evidențiază succesiunea logică și clară a pașilor parcurși pentru întreruperea controlată a unui proces critic.

2.3.6 Utilizarea diagramelor de clasă pentru modelarea componentelor sistemului

Diagramele de clasă reprezintă un instrument fundamental în ingineria software, utilizat pentru a modela structura statică a unui sistem. Aceste diagrame ilustrează clasele componente ale sistemului, atributele și metodele acestora, precum și relațiile dintre ele, oferind o perspectivă clară asupra arhitecturii și interacțiunilor interne ale aplicației. O clasă într-o diagramă UML reprezintă o categorie de entități cu atribute și comportamente comune, servind drept șablon pentru crearea obiectelor cu caracteristici și funcționalități specifice[11]. Diagramele de clasă sunt utilizate pentru a modela structura (viziunea statică asupra) unui sistem, incluzând clase, interfețe, obiecte și relațiile dintre acestea[12].

În procesul de dezvoltare al sistemului, diagramele de clasă au fost importante în modelarea componentelor cheie, precum procesele de atac brute-force, mecanismele de export al rapoartelor și gestionarea listelor de cuvinte (wordlist-uri, rainbow tables). Aceste reprezentări au eficientizat

înțelegerea structurii și comportamentului fiecărei componente, simplificând etapele de proiectare și implementare.

În continuare, vom examina în detaliu diagramele de clasă relevante pentru sistemul dezvoltat, subliniind modul în care acestea reflectă arhitectura și funcționalitățile aplicației.

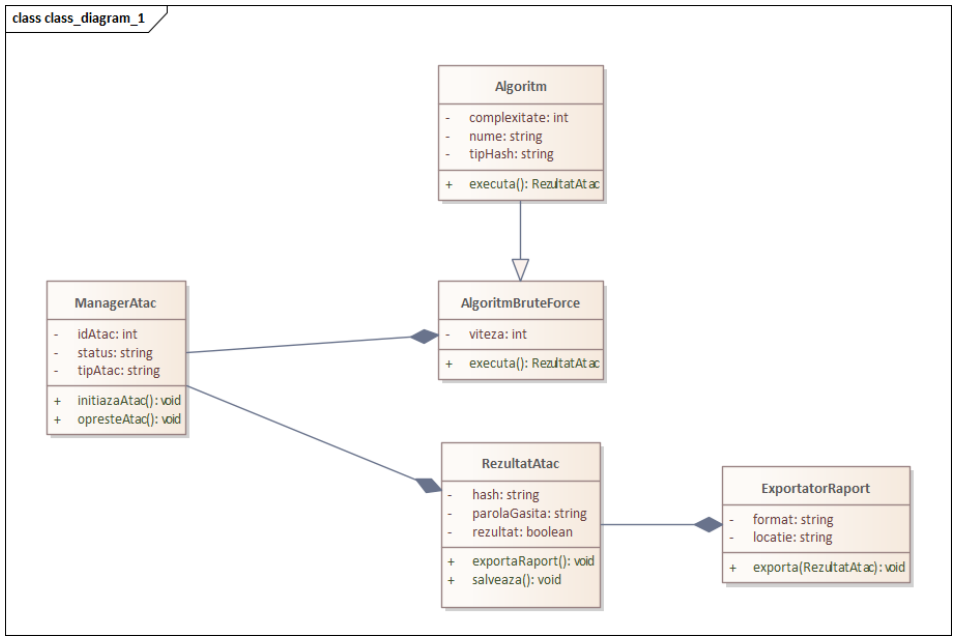

Figura 2.13 - Diagrama de clasă pentru procesul de atac de tip brute-force

Diagrama de clasă din figura 2.13 este specifică procesului de atac brute-force care evidențiază structura statică a componentelor implicate și relațiile dintre acestea. Managerul de atac coordonează execuția atacului, gestionând inițierea, oprirea și obținerea rezultatelor, menținând atribute precum identificatorul atacului, tipul și starea curentă. Algoritmul reprezintă o entitate abstractă ce definește structura generală a unui atac, specificând numele, tipul de hash și complexitatea asociată. Algoritmul de tip brute-force extinde această entitate abstractă, adăugând atributul de viteză și implementând metoda de execuție specifică acestui tip de atac. Rezultatul atacului stochează informațiile obținute, inclusiv hash-ul, parola identificată și succesul

operațiunii, oferind funcționalități pentru salvarea și exportul raportului. Exportatorul de raport gestionează procesul de export al rezultatelor într-un format și la o locație specificate, facilitând documentarea și analiza ulterioară a datelor.

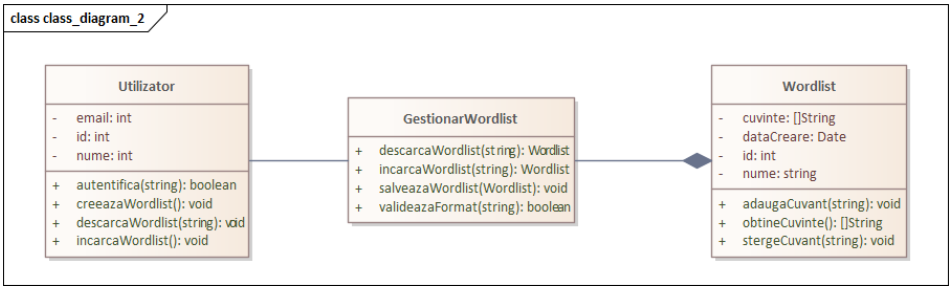

Figura 2.14 - Diagrama de clasă pentru procesul de export al raportului

În cadrul figurii 2.14 este reprezentată diagrama de clasă asociată procesului de export al raportului. Componentele și interacțiunile reprezentate în diagramă sunt necesare pentru generarea și salvarea rapoartelor rezultate în urma atacurilor. Utilizatorul inițiază procesul de export și vizualizează raportul generat, interacționând cu managerul de export, care este responsabil de întregul proces, inclusiv inițierea și verificarea stării exportului. Raportul conține detaliile relevante, cum ar fi conținutul și data creării, fiind generat de un component dedicat, generatorul de raport, care colectează informațiile din rezultatul atacului. Exportatorul de fișier gestionează salvarea raportului într-un la o locație specificată, asigurând accesibilitatea și integritatea datelor. Interacțiunile dintre aceste componente asigură un flux coerent și eficient al procesului de export, de la inițierea de către utilizator până la obținerea raportului final, gata pentru analiză și arhivare.

2.3.7 Modelarea arhitecturii sistemului prin diagrame de componente

Diagramele de componente oferă numeroase avantaje esențiale pentru gestionarea eficientă a proiectelor. În primul rând, acestea asigură o claritate structurală prin reprezentarea vizuală a componentelor și a relațiilor dintre ele, facilitând astfel înțelegerea arhitecturii sistemului de către toți membrii echipei de proiect. În al doilea rând, diagramele de componente permit identificarea

și gestionarea dependențelor dintre module, contribuind la detectarea potențialelor probleme legate de interdependențe și promovând un design modular robust. De asemenea, prin delimitarea clară a responsabilităților fiecărei componente, se încurajează reutilizarea acestora în diverse contexte, ceea ce sporește eficiența procesului de dezvoltare. Nu în ultimul rând, aceste diagrame servesc ca instrumente eficiente de comunicare între dezvoltatori, arhitecți și alte părți interesate, asigurând o înțelegere comună și coerentă a soluției propuse.

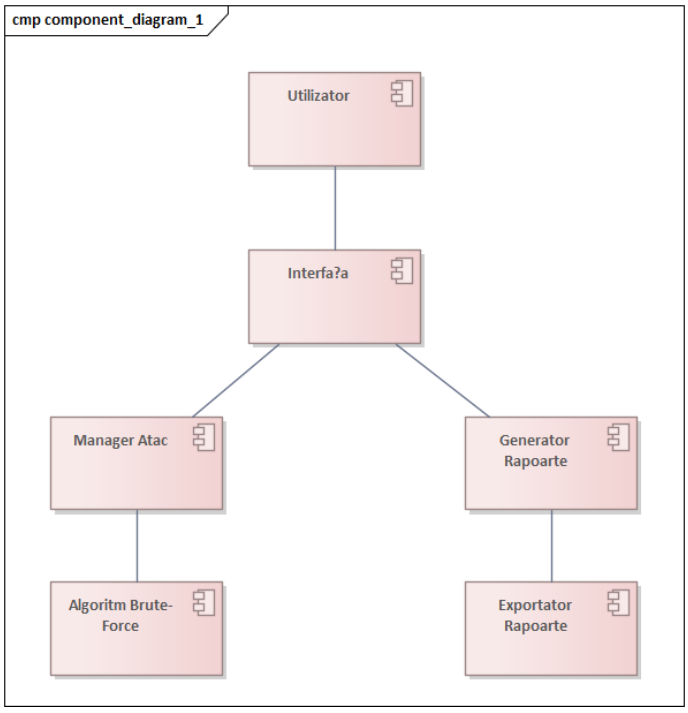

Figura 2.15 – Arhitectura sistemului de brute-force

Diagrama de componente din figura 2.15 este pentru procesul de atac brute-force și evidențiază interacțiunile dintre componentele sistemului. Utilizatorul pornește atacul prin intermediul Interfeței Utilizator, care transmite parametrii necesari către Managerul Atac. Acesta coordonează execuția Algoritmului Brute-Force, responsabil pentru generarea și testarea combinațiilor de parole. Rezultatele obținute sunt procesate de Generatorul de Rapoarte și

exportate de Exportatorul de Rapoarte, oferind utilizatorului acces la raportul final. Această diagramă clarifică arhitectura modulară a sistemului și relațiile dintre componente, facilitând înțelegerea și optimizarea procesului de atac.

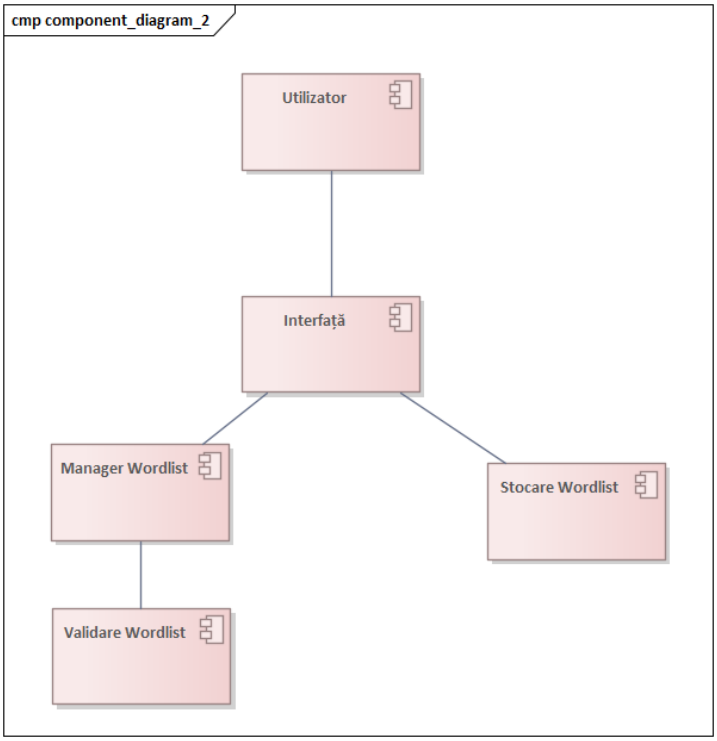

Figura 2.16 – Arhitectura pentru validarea wordlist-ului

Diagrama de componente prezentată în figura 2.16 prezintă procesul cd încărcare sau creare a unui wordlist, evidențiind relațiile și dependențele dintre principalele module implicate. Inițial, utilizatorul utilizează interfața pentru a introduce sau selecta un wordlist. Interfața transmite această informație către managerul de wordlist-uri, care e responsabilă de procesul de gestionare. Managerul trimite wordlist-ul către modulul de validare pentru a fi verificat. În cazul în care wordlist-ul este valid, managerul îl stochează în modulul de stocare a wordlist-urilor. Această

diagramă subliniază fluxul de date și interacțiunile necesare pentru asigurarea unui management eficient și sigur al wordlist-urilor.

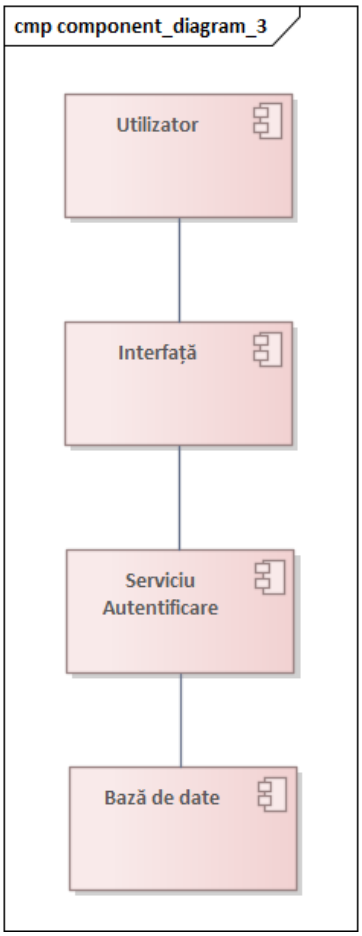

Figura 2.17 – Arhitectura sistemului de autentificare

În diagrama din figura 2.17 utilizatorul inițiază procesul de autentificare prin intermediul Interfeței Utilizator (UI), unde introduce credențialele necesare, precum numele de utilizator și

parola. Interfața Utilizator transmite aceste informații către serviciul de autentificare, componenta centrală responsabilă cu procesarea cererii de autentificare.

Serviciul de autentificare validează datele primite după care accesează Baza de Date a Utilizatorilor pentru a compara credențialele furnizate cu cele stocate. Dacă informațiile corespund, Managerul de Autentificare generează un token de sesiune și transmite confirmarea către Interfața Utilizator, semnalând succesul autentificării. În caz contrar, utilizatorul este notificat cu privire la eșecul autentificării și i se oferă opțiunea de a reîncerca.

2.3.8 Modelarea implementării aplicației

Diagramele de plasare facilitează înțelegerea și comunicarea modului în care componentele software sunt distribuite și interacționează în cadrul infrastructurii hardware a unui sistem. Aceste diagrame pot fi folosite pentru a vizualiza topologia hardware a unui sistem, pentru a descrie componentele hardware utilizate pentru implementare și pentru a arăta relațiile dintre acestea.

Unul dintre principalele avantaje ale diagramelor de plasare este de a evidenția interacțiunile dintre componentele software și hardware, oferind o perspectivă clară asupra arhitecturii sistemului. Această claritate contribuie la identificarea și rezolvarea potențialelor probleme legate de performanță, scalabilitate și securitate încă din fazele incipiente ale dezvoltării.

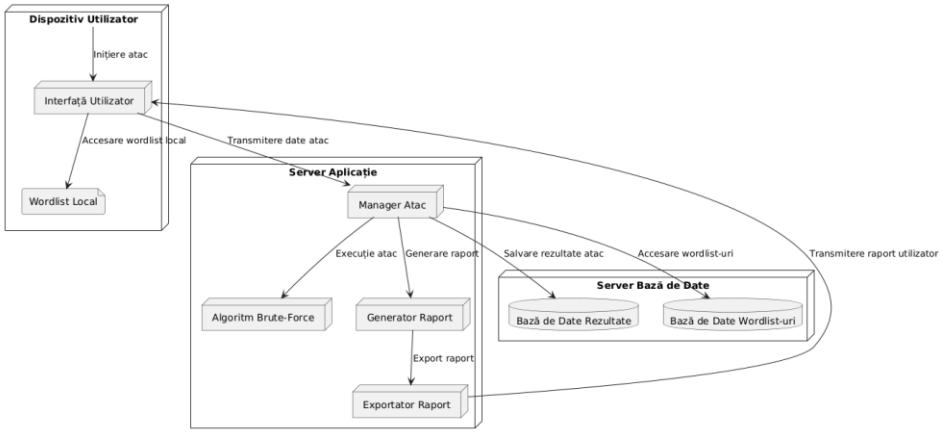

Figura 2.18 - Diagrama de plasare a sistemului de atac brute-force

În diagrama din figura 2.18, Dispozitivul Utilizatorului inlcude Interfața Utilizator și poate stoca un Wordlist Local. Utilizatorul inițiază atacul și interacționează cu interfața pentru a configura și lansa procesul. Serverul Aplicație conține componentele principale ale sistemului: Managerul Atac, Algoritmul Brute-Force, Generatorul Raport și Exportatorul Raport, care lucrează împreună pentru a executa atacul și a genera rapoartele corespunzătoare. Serverul Bază de Date stochează Baza de Date Rezultate, unde sunt salvate rezultatele atacurilor, și Baza de Date Wordlist-uri, care conține listele de parole utilizate în procesul de atac. Interacțiunile dintre aceste componente sunt reprezentate prin săgeți, indicând fluxul de date și procesele din cadrul sistemului.

3. Relizarea sistemului informatic

În cadrul acestui capitol este prezentat modul în care a fost implementat sistemul HashSentinel, o aplicație specializată în analizarea și descifrarea valorilor de tip hash prin tehnici brute-force și wordlist. Dezvoltarea acestui modul a fost realizată folosind un set specific de tehnologii care oferă atât flexibilitate, cât și performanță. Alegerea acestor tehnologii s-a bazat pe criterii precum eficiența în procesare, ușurința dezvoltării interfeței grafice și interoperabilitatea între limbaje.

Python a fost utilizat pentru crearea interfeței grafice (GUI) datorită simplității sale, a comunității largi de suport și a ecosistemului său bogat în biblioteci. Biblioteca Tkinter, inclusă în standardul Python, a fost aleasă pentru realizarea interfeței utilizatorului datorită integrării directe și a resurselor documentare disponibile. Tkinter permite dezvoltarea rapidă a aplicațiilor grafice, fiind potrivită pentru aplicații care nu necesită o interfață web complex[13].

Commented [7]: descriere

Golang (Go) a fost utilizat pentru partea de procesare efectivă a atacurilor brute-force, datorită performanței ridicate și a suportului nativ pentru programare concurentă prin goroutine-uri. Go este un limbaj compilat, cu un runtime eficient și o sintaxă curată, ceea ce îl face potrivit pentru aplicații care necesită rulare rapidă, utilizare eficientă a resurselor și execuție paralelă. Conform JetBrains, Go este considerat unul dintre cele mai performante și fiabile limbaje moderne, fiind folosit extensiv pentru dezvoltare de aplicații de sistem, rețea și securitate[14].

Prin combinarea Python (Tkinter) pentru front-end și Go pentru back-end, aplicația HashSentinel reușește să îmbine accesibilitatea unei interfețe grafice prietenoase cu puterea de calcul necesară pentru operații intensive. Comunicarea între componente se realizează prin apeluri de sistem către un executabil Go precompilat, menținând separarea clară a responsabilităților și modularitatea sistemului.

3.1 Realizarea sistemului

Partea de backend a aplicației HashSentinel este responsabilă cu realizarea efectivă a procesului de spargere (cracking) a hash-urilor, utilizând tehnici brute-force și wordlist. Această componentă este implementată în limbajul de programare **Go**, ales datorită performanței ridicate

în procesarea paralelă și a modelului de concurență bazat pe goroutines, care este foarte eficient pentru sarcini intensive cum ar fi generarea și testarea rapidă a unui număr mare de parole.

Go permite realizarea unui executabil numit bruteforce_cli.exe, care este invocat de aplicația principală GUI. Acest executabil primește parametri precum:

- --hash: hash-ul care urmează să fie spart
- --type: tipul de hash (e.g., MD5, SHA1 etc.)
- --mode: tipul atacului (wordlist, bruteforce)
- --file: locația unui fișier wordlist
- --charset și --max-len: în cazul brute-force
- --threads: numărul de fire de execuție
- --ram: opțional, pentru a încărca wordlist-ul complet în memorie

Aceste opțiuni permit o flexibilitate ridicată în funcție de scopul testului sau al atacului de tip bruteforce.

Un avantaj major al utilizării Go este capacitatea sa de a gestiona foarte eficient execuții multi-threaded, fără complexitatea tradițională a managementului de thread-uri din alte limbaje. Folosirea canalelor (channels) și a goroutines asigură o procesare paralelă simplificată, ceea ce duce la o accelerare notabilă a procesului de cracking, în special în modul bruteforce sau wordlist cu citire din RAM.

Structura internă a componentelor backend include:

- Parsarea argumentelor din linia de comandă
- Identificarea hash-ului în caz de mod „auto”
- Selectarea algoritmului de spargere în funcție de tip
- Gestionarea atacului fie printr-o listă de parole (wordlist), fie generativ (bruteforce)

Optimizarea căutării prin utilizarea RAM-ului (la alegerea utilizatorului)

- Afișarea rezultatului final într-un format ușor de înțeles (inclusiv numărul de încercări și timpul)

Această arhitectură modulară a backendului permite integrarea ușoară cu diverse interfețe grafice, dar și rularea autonomă pentru testare.

Componenta de front-end a aplicației HashSentinel este responsabilă cu interacțiunea utilizatorului și este realizată în limbajul Python, utilizând biblioteca Tkinter. Alegerea Tkinter a fost motivată de mai mulți factori:

- este integrat nativ cu Python, fără a necesita instalarea de librării externe;
- permite crearea rapidă de interfețe grafice funcționale;
- oferă suficiente componente (widgets) pentru a construi o aplicație de tip GUI prietenoasă cu utilizatorul.
- Frontend-ul oferă o experiență interactivă, în care utilizatorul poate:
- introduce un hash de spart;
- specifica un *salt* opțional (înainte sau după hash);
- selecta tipul de atac (*wordlist* sau *brute-force*);
- încărca sau alege un fișier wordlist;
- opta pentru modul de citire din RAM;
- configura numărul de threaduri;
- seta charset-ul și lungimea maximă pentru atacurile brute-force;
- detecta automat tipul de hash;
- vizualiza rezultatul final într-o fereastră popup.

Aplicația construiește o comandă în funcție de opțiunile selectate de utilizator și o trimite către executabilul bruteforce_cli.exe, invocând procesul printr-un thread separat pentru a evita blocarea interfeței.

Exemple de funcționalități frontend:

- Selectarea wordlist-ului: Interfața permite atât încărcarea unui wordlist extern, cât și alegerea dintr-o listă de fișiere deja existente în directorul local. Acest lucru oferă utilizatorului flexibilitate și reutilizare.

- Detectarea tipului de hash: Aplicația permite identificarea automată a tipului de hash printr-un simplu click, apelând funcția --detect-only din backend. Rezultatul este afișat și setat automat în interfață.
- Controlul avansat al atacului: Utilizatorul poate alege între wordlist (cu sau fără RAM), și brute-force, cu opțiuni clare și intuitive. Schimbarea modului actualizează automat elementele vizibile în UI.
- Adăugare și creare wordlist: O funcționalitate suplimentară permite utilizatorului să creeze un wordlist personalizat, introducând cuvinte manual, salvate într-un fișier. Această opțiune este integrată printr-o fereastră separată, dedicată.

Interfața a fost concepută într-un stil modern, cu culori închise și elemente aliniate logic pentru a oferi o utilizare cât mai plăcută și eficientă.

3.2 Descrierea codului pe module

Un modul important al aplicației HashSentinel este mecanismul de detectare a tipului de hash pe baza expresiilor regulate (regex). Acesta este implementat în fișierul detect_wrapper.go din componenta Go a proiectului. Scopul acestui modul este să identifice, fără intervenție din partea utilizatorului, ce algoritm a fost folosit pentru a genera un anumit hash, pornind de la forma sa textuală.

Fragmentul de cod de mai jos definește structura de bază pentru fiecare regulă de detectare:

```
type HashRule struct {
      Name     string
      Regex    string
      compiled *regexp.Regexp
}
```

Fiecare HashRule conține:

- Name: numele algoritmului (ex. "MD5", "SHA256");
- Regex: expresia regulată care descrie forma specifică a hash-ului;
- compiled: versiunea pre-compilată a expresiei, folosită pentru verificări eficiente.

Lista completă a regulilor acoperă cele mai comune tipuri de hashuri, inclusiv MD5, SHA1, SHA256, SHA512, bcrypt, NTLM, hashuri MySQL și cele specifice unor platforme precum WordPress și Drupal:

```
var rules = []HashRule{
	{"MD5", "^[a-fA-F0-9]{32}$", nil},
	{"SHA1", "^[a-fA-F0-9]{40}$", nil},
	{"SHA256", "^[a-fA-F0-9]{64}$", nil},
	{"SHA512", "^[a-fA-F0-9]{128}$", nil},
	{"bcrypt", `^\$2[aby]?\$\d{2}\$[./A-Za-z0-9]{53}$`, nil},
	{"NTLM", "^[A-Fa-f0-9]{32}$", nil},
	{"MySQL323", "^\\*[A-Fa-f0-9]{40}$", nil},
	{"MySQLSHA1", "^[a-f0-9]{40}$", nil},
	{"WordPress", "^\\$P\\$[./0-9A-Za-z]{31}$", nil},
	{"Drupal7", "^\\$S\\$[./0-9A-Za-z]{52}$", nil},
}
```

În funcția DetectHash, toate expresiile sunt mai întâi compilate:

```
func compileRules() {
	for i := range rules {
		rules[i].compiled = regexp.MustCompile(rules[i].Regex)
	}
}
```

Ulterior, pentru fiecare regulă, aplicația verifică dacă hash-ul introdus se potrivește cu expresia respectivă:

```
func DetectHash(input string) string {
	compileRules()
	matches := []string{}

	for _, rule := range rules {
		if rule.compiled.MatchString(input) {
			matches = append(matches, rule.Name)
		}
	}
}
```

Dacă sunt identificate mai multe potriviri, acestea sunt afișate în consolă și este returnată prima opțiune ca default. În caz contrar, aplicația va indica faptul că nu a putut detecta hash-ul:

```
if len(matches) > 0 {
	fmt.Println("Detectare avansată:")
	for _, name := range matches {
		fmt.Println("-", name)
	}
	return matches[0]
}
```

```
fmt.Println("Fallback: necunoscut")
return "unknown"
```

Această abordare permite o detecție rapidă și fiabilă, eliminând necesitatea ca utilizatorul să cunoască detalii tehnice despre algoritmul folosit pentru generarea hash-ului. Detectarea se integrează cu interfața grafică, completând automat câmpul „Tip hash” după analiza hash-ului introdus.

Pentru spargerea hash-urilor sunt implementate două metode eficiente de spargere a hash-urilor folosind liste predefinite de parole (wordlist-uri): una care citește direct din fișier și alta care încarcă întreaga listă în memorie non volatilă.

Funcția CrackFromWordlist are rolul de a parcurge un fișier wordlist linie cu linie, testând fiecare cuvânt pentru a verifica dacă, odată hash-uit, corespunde valorii hash furnizate de utilizator.

```
lines := make(chan string, 1000)
results := make(chan string, 1)
```

creează două canale: unul pentru a distribui parolele către firele de execuție (concurente), și unul pentru a stoca eventualul rezultat.

Ulterior, sunt lansate n goroutines în funcție de parametrii:

```
for i := 0; i < threads; i++ {
    go func() {
        for word := range lines {
            if MatchHash(word, hashType, hash) {
                results <- word
                return
            }
        }
    }()
}
```

Fiecare goroutine consumă cuvinte din canalul lines, verificându-le împotriva hash-ului țintă prin funcția MatchHash. Dacă este identificată o potrivire, se trimite rezultatul în canalul results, iar celelalte fire sunt oprite.

Procesul de citire a wordlistului se realizează într-un thread separat:

```
scanner := bufio.NewScanner(file)
for scanner.Scan() {
    lines <- strings.TrimSpace(scanner.Text())
}
```

Această abordare reduce semnificativ consumul de memorie, deoarece fișierul nu este încărcat integral în RAM, ci procesat incremental.

În contrast, funcția CrackFromWordlistInRAM încarcă întregul conținut al fișierului în memorie, optimizând performanța în cazul în care fișierul este accesat frecvent sau când se dorește eliminarea latenței de la I/O.

Citirea fișierului are loc în întregime:

```
var allWords []string
for scanner.Scan() {
    allWords = append(allWords, scanner.Text())
}
```

Lista de parole este apoi împărțită în „chunk-uri”:

Commented [8]: explicare

```
chunkSize := (total + threads - 1) / threads
```

Acestea sunt procesate în paralel, fiecare fir prelucrând un segment diferit al wordlist-ului:

```
go func(words []string) {
    for _, word := range words {
        if MatchHash(word, hashType, hash) {
            result = word
            return
        }
    }
}()
```

În comparație cu metoda anterioară, aceasta are avantajul unei viteze mai mari de acces la date, însă poate consuma semnificativ mai multă memorie pentru wordlist-uri de dimensiuni mari.

Una dintre cele mai solicitante metode de spargere a unui hash este atacul de tip brute-force, care presupune încercarea tuturor combinațiilor posibile de caractere până la o lungime dată. Acest proces este complet automatizat și nu necesită o listă predefinită de parole, însă este extrem

de intensiv din punct de vedere computațional. În cadrul aplicației HashSentinel, această funcționalitate este implementată prin intermediul a două funcții backend: BruteForceRecursive și CrackBruteForceParallel.

Abordările brute-force sunt adesea considerate ineficiente în comparație cu metodele bazate pe dicționare (wordlists), însă în anumite scenarii — cum ar fi parole extrem de simple sau lipsa unui corpus de parole relevante — acestea rămân indispensabile. Studiile din domeniul criptanalizei arată că eficiența unui astfel de atac este direct influențată de lungimea parolei, complexitatea setului de caractere și puterea de procesare disponibilă[15].

Rolul parametrilor esențiali

Pentru a înțelege corect logica algoritmului, este importantă explicarea parametrilor charset, prefix, maxLen:

- charset: reprezintă setul de caractere permise în compunerea parolelor testate. De exemplu, un charset de tip "abc123" înseamnă că parolele generate pot conține doar literele a, b, c și cifrele 1, 2, 3. Cu cât charset este mai mare, cu atât spațiul de căutare crește exponențial. Alegerea unui charset optimizat pentru contextul hash-ului (ex. parole simple, parole alfanumerice, caractere speciale) influențează considerabil eficiența și durata atacului.
- prefix: este un șir de caractere deja format, care este extins recursiv în cadrul funcției BruteForceRecursive. În funcția paralelă CrackBruteForceParallel, fiecare fir de execuție pornește cu un prefix diferit, corespunzător unui caracter unic din charset, pentru a distribui în mod uniform efortul de căutare. Acest mecanism de împărțire a sarcinilor inițiale creează paralelism real, permițând ca fiecare fir să exploreze o ramură complet diferită a arborelui de combinații.
- maxLen: definește lungimea maximă a parolelor generate. Acest parametru este critic pentru a preveni recursivitatea infinită și pentru a menține procesul de căutare în limite rezonabile. De exemplu, un maxLen de 4 cu un charset de abc va genera $3^4 = 81$ de combinații, în timp ce o creștere la 8 va duce la peste 6.500 de combinații. Astfel, maxLen este o limitare impusă utilizatorului, atât pentru a controla durata execuției, cât și pentru a adapta procesul la nevoile specifice ale testului.

În funcția BruteForceRecursive, fiecare apel recursiv construiește o combinație de caractere pornind de la prefix, extins cu caractere din charset, până când este atinsă lungimea maxLen sau este găsită parola corectă:

```
if found.Load() || len(current) > maxLen {
    return "", false
}
```

În funcția CrackBruteForceParallel, acești parametri sunt utilizați pentru a inițializa mai multe fire de execuție independente, fiecare având un prefix de un caracter:

```
for _, c := range charset {
    go func(prefix string) {
        BruteForceRecursive(hash, hashType, charset, prefix, maxLen, ...)
    }(string(c))
}
```

4. Documentarea sistemului

Pentru a asigura o utilizare eficientă a aplicației HashSentinel, a fost realizat un proces de documentare care acoperă componentele software esențiale, cerințele de sistem, modul de configurare și scenariile de utilizare. Documentarea are scopul de a sprijini utilizatorul – fie el student, cercetător sau profesionist în domeniul securității informaționale – în înțelegerea completă a arhitecturii aplicației, precum și în exploatarea corectă a funcționalităților acesteia.

Aplicația HashSentinel a fost concepută pentru a funcționa pe platforme moderne și accesibile, oferind compatibilitate cu sisteme de operare populare și medii de dezvoltare actuale. În forma sa actuală, aplicația a fost testată pe sisteme Windows 10 și Windows 11, existând totodată posibilitatea de portare pe distribuții Linux compatibile, în special datorită utilizării limbajelor Go și Python.

Pentru o funcționare optimă, este necesară instalarea următoarelor componente:

- Python, versiunea 3.10 sau superioară, împreună cu biblioteca standard tkinter, utilizată pentru construirea interfeței grafice (aceasta este inclusă implicit în distribuțiile Python pentru Windows);
- Go (Golang), versiunea 1.21 sau mai recentă, indispensabilă pentru rularea și compilarea backend-ului aplicației;
- Un editor de cod modern, precum Visual Studio Code sau GoLand, este recomandat pentru modificarea surselor sau extinderea funcționalităților;
- Fișiere de tip wordlist (.txt), esențiale în cazul rulării unui atac de tip dicționar.

Structura proiectului este gândită într-o manieră clară și modulară, conținând principalele componente:

- Executabilul bruteforce_cli.exe, compilat din codul sursă scris în Go, responsabil pentru implementarea logicii de cracking;
- Directorul GUI/, care cuprinde interfața grafică dezvoltată în Python;
- Folderul wordlists/, destinat stocării fișierelor de tip dicționar utilizate în atacurile de tip wordlist.

Lansarea aplicației se face prin rularea fișierului principal main_sentinel_gui.py, fiind recomandată utilizarea unui mediu virtual Python pentru a izola dependințele și a preveni eventualele conflicte între pachetele instalate la nivelul sistemului.

Aplicația HashSentinel oferă o interfață grafică intuitivă și accesibilă, care maschează complexitatea proceselor de cracking hash din spatele unei logici robuste. Interfața permite utilizatorului să configureze, lanseze și monitorizeze în timp real procesele de spargere a hash-urilor, oferind în același timp flexibilitate și performanță.

Funcționalitățile cheie ale aplicației includ:

- Detecția automată a tipului hash: Utilizatorul poate opta pentru identificarea automată a algoritmului de hash (MD5, SHA1, SHA256 etc.), pe baza unui sistem intern de reguli regex implementate în backend.
- Atacuri de tip wordlist: Aplicația permite încărcarea unui fișier wordlist local, utilizat pentru testarea succesivă a fiecărei parole din listă. Utilizatorul poate alege între două moduri:
- citire secvențială de pe disc, optimă pentru sisteme cu memorie limitată;
- încărcare completă în RAM, care oferă timpi de execuție mai rapizi pentru fișiere de dimensiuni rezonabile.
- Atacuri de tip brute-force: În acest mod, parolele sunt generate automat pe baza unui set de caractere (charset) și a unei limite superioare pentru lungimea parolei (max-len). Această abordare este ideală în scenarii fără indicii despre parola originală.
- Configurabilitate completă: Utilizatorul poate seta numărul de threaduri de execuție, ceea ce influențează direct viteza procesului de cracking.
- Afișarea rezultatului în interfață: După finalizarea unui atac, aplicația oferă rezultatul (parola găsită), timpul de execuție și numărul total de parole testate.
- Sistem de jurnalizare (logging): Fiecare comandă transmisă către backend-ul Go este logată într-un format standardizat, incluzând momentul execuției și parametrii utilizați. Acest sistem facilitează trasabilitatea execuțiilor și analiza comparativă a performanței între diferite metode de atac.

Structura interfeței grafice este proiectată astfel încât să permită navigarea intuitivă între modulele aplicației (HashSentinel și PacketSentinel), oferind o experiență unificată și ergonomică.

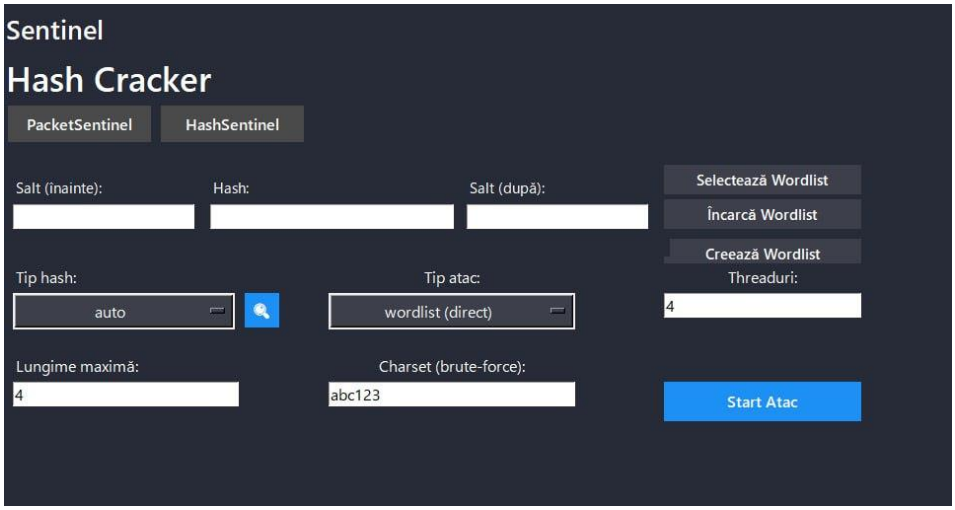

Figura 4.1 – Interfața aplicației

Pentru a asigura trasabilitatea și validarea procesului de testare, aplicația HashSentinel integrează un mecanism de jurnalizare a comenzilor transmise către componenta sa de tip linie de comandă, denumită bruteforce_cli.exe. Această componentă este responsabilă de execuția efectivă a procesului de spargere a hash-urilor și este invocată de interfața grafică printr-un apel extern. Fiecare execuție este însoțită de un mesaj de tip log în format [DEBUG HH:MM:SS], care permite identificarea exactă a parametrilor CLI utilizați și momentul inițierii operațiunii.

Sunt analizate trei exemple de loguri ale aplicației, fiecare ilustrând o metodologie distinctă de atac.

Atac de tip Wordlist cu încărcare în memorie (RAM)

```
[DEBUG 22:29:12] Comandă trimisă către Go:
z:\Licenta\GUI\PacketSentinel\bruteforce_cli.exe                        --hash
29c43922928308c873c3dfaf23f1631b --type md5 --mode wordlist --threads 4 --file
wordlists\rockyou.txt –ram
```

Această comandă reflectă un atac de tip wordlist asupra unui hash de tip MD5. Parametrul --file indică utilizarea unui fișier extern, rockyou.txt, recunoscut în comunitatea de securitate cibernetică pentru testele de penetrare. Prin adăugarea flagului --ram, fișierul este încărcat complet în memoria RAM, eliminând latențele cauzate de accesul la disc. Această abordare este benefică în cazul în care resursele hardware permit procesarea integrală în memorie, rezultând într-un timp de execuție redus pentru wordlist-uri de dimensiuni moderate.

Parametrii utilizați sunt:

- --hash: valoarea hash-ului de test.
- --type: specifică algoritmul hash (în acest caz, MD5).
- --mode: definește strategia de atac (wordlist).
- --threads: determină gradul de paralelizare.
- --ram: activează modul de procesare în memorie.

Atac de tip Wordlist cu citire directă de pe disc

```
[DEBUG 22:31:17] Comandă trimisă către Go:
z:\Licenta\GUI\PacketSentinel\bruteforce_cli.exe                        --hash
29c43922928308c873c3dfaf23f1631b --type md5 --mode wordlist --threads 4 --file
wordlists\rockyou.txt
```

Această comandă este similară celei anterioare, însă lipsește flagul --ram. Prin urmare, fiecare linie a fișierului rockyou.txt este citită secvențial direct de pe disc. Această metodă reduce consumul de memorie, fiind mai eficientă în cazul sistemelor cu resurse limitate. Totodată, ea poate fi mai lentă din cauza operațiunilor de I/O repetate, în special când se lucrează cu fișiere de mari dimensiuni.

Atac de tip Brute-Force cu charset personalizat

```
[DEBUG 22:35:38] Comandă trimisă către Go:
z:\Licenta\GUI\PacketSentinel\bruteforce_cli.exe                        --hash
29c43922928308c873c3dfaf23f1631b --type md5 --mode bruteforce --threads 4 --
charset abcdefghijklmnopqrstuvwxyz0123456789 --max-len 8
```

Acest log evidențiază execuția unui atac de tip brute-force, în care parolele sunt generate automat caracter cu caracter, utilizând setul definit de utilizator prin parametrul --charset. În acest caz, setul este compus din litere mici și cifre, ceea ce reflectă o strategie de spargere standard. Parametrul --

max-len restricționează lungimea maximă a parolei generate la 8 caractere, reducând spațiul de căutare și implicit timpul de execuție.

Parametri specifici metodei brute-force:

- --charset: definește alfabetul utilizat pentru generarea parolelor.
- --max-len: stabilește limita superioară a lungimii parolelor testate.

Această abordare este ideală în scenarii în care nu se cunosc informații despre parola originală și nici nu este disponibil un wordlist relevant.

BIBLIOGRAFIE

[1] "What Is Hashing in Cybersecurity? | CrowdStrike," CrowdStrike.com. Accessed: Mar. 04, 2025. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/data-protection/data-hashing/

[2] "https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf." Accessed: Mar. 04, 2025. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[3] V. L. Yisa, M. Baba, and E. T. Olaniyi, "A Review of Top Open Source Password Cracking Tools," 2016.

[4] "John the Ripper | Hackviser." Accessed: Mar. 05, 2025. [Online]. Available: https://hackviser.com/tactics/tools/john-the-ripper

[5] "Proiectarea Sistemelor Informatice - CUPRINS CAPITOLUL I 1 Proiectarea sistemelor - Studocu." Accessed: Mar. 07, 2025. [Online]. Available: https://www.studocu.com/ro/document/universitatea-romano-americana-din-bucuresti/proiectarea-sistemelor-informatice/proiectarea-sistemelor-informatice/3631008

[6] "UX (User Experience) – tot ce trebuie să știi despre proiectarea experienței utilizatorului - Copymate." Accessed: Mar. 12, 2025. [Online]. Available: https://copymate.app/ro/blog/multi/ux-user-experience-tot-ce-trebuie-sa-stii-despre-proiectarea-experientei-utilizatorului/

[7] "Ce este cerința nefuncțională în ingineria software?" Accessed: Mar. 13, 2025. [Online]. Available: https://www.guru99.com/ro/non-functional-requirement-type-example.html

[8] "Diagrame de interacțiune, colaborare și secvență cu exemple." Accessed: Mar. 24, 2025. [Online]. Available: https://www.guru99.com/ro/interaction-collaboration-sequence-diagrams-examples.html?utm_source=chatgpt.com

[9] D. L. Jisa, "Evaluarea si compararea instrumentelor CASE bazate pe UML," 2001.

[10] "Communication Diagram | Enterprise Architect User Guide." Accessed: Mar. 24, 2025. [Online]. Available: https://sparxsystems.com/enterprise_architect_user_guide/17.0/modeling_languages/communicationdiagram.html

[11] "Tutorial diagramă de clasă UML: Clasă abstractă cu exemple." Accessed: Mar. 25, 2025. [Online]. Available: https://www.guru99.com/ro/uml-class-diagram.html?utm_source=chatgpt.com

[12] "https://cadredidactice.ub.ro/sorinpopa/files/2018/10/L2_diagrama_de_clase.pdf?utm_source=chatgpt.com." Accessed: Mar. 25, 2025. [Online]. Available: https://cadredidactice.ub.ro/sorinpopa/files/2018/10/L2_diagrama_de_clase.pdf?utm_source=chatgpt.com

[13] "tkinter — Python interface to Tcl/Tk — Python 3.13.2 documentation." Accessed: Mar. 27, 2025. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[14] “Go Programming - The State of Developer Ecosystem in 2023 Infographic | JetBrains: Developer Tools for Professionals and Teams.” Accessed: Mar. 27, 2025. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2023/go/

[15] W. Stallings, “Cryptography and Network Security: Principles and Practice” - Capitolul 21.