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 în producție

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

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

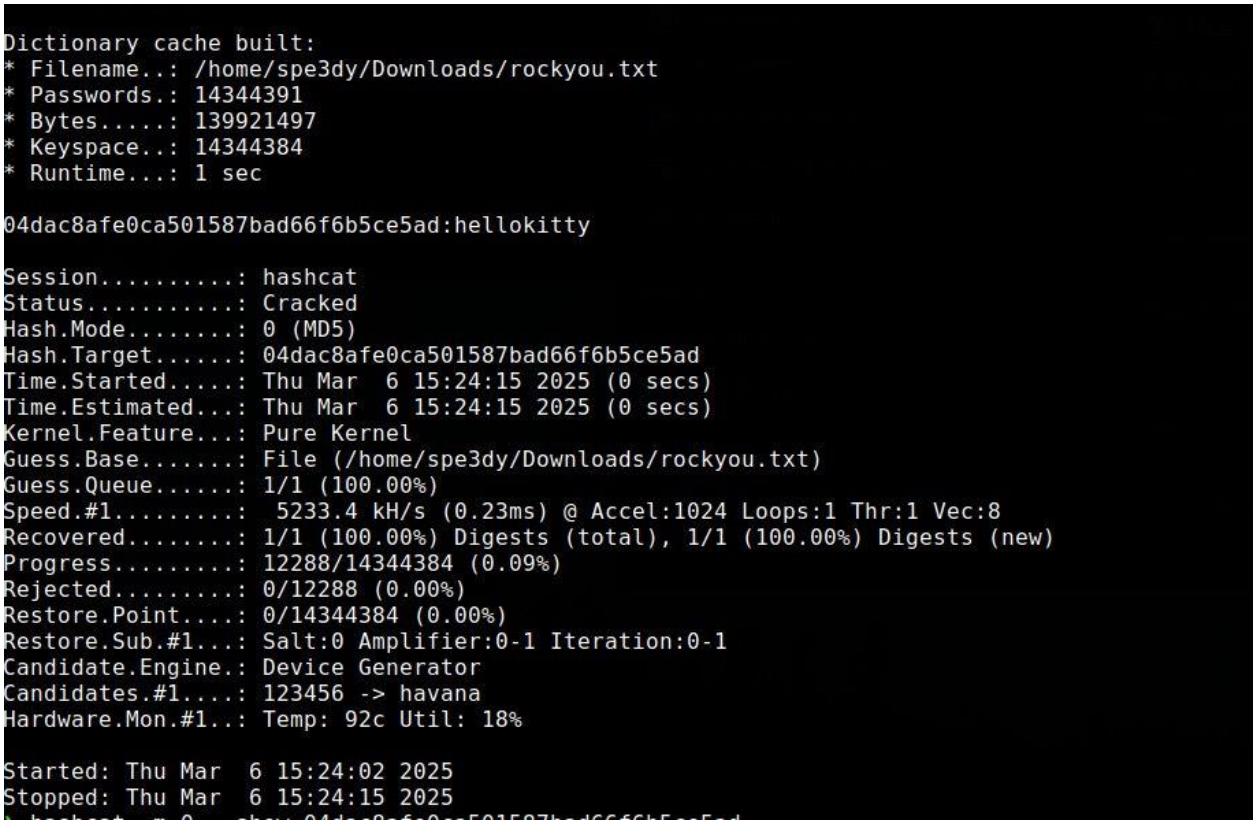

Figura 1.1 – Aplicația “Hashcat”

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]

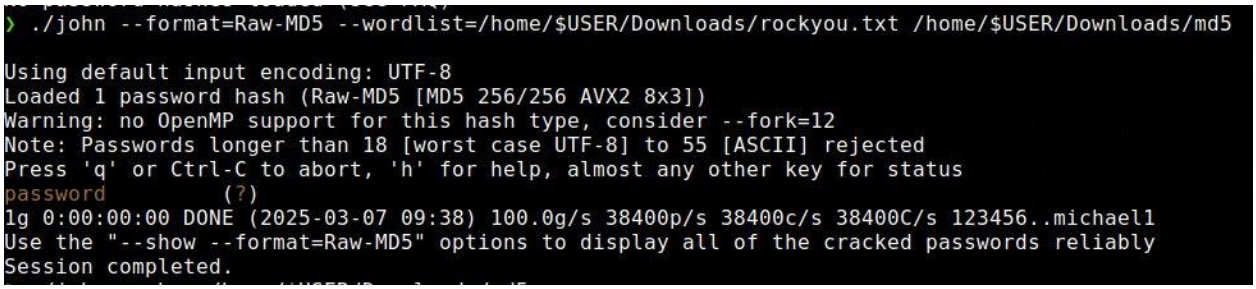

Figura 1.2 – Aplicația "John The Ripper"

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 pe 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

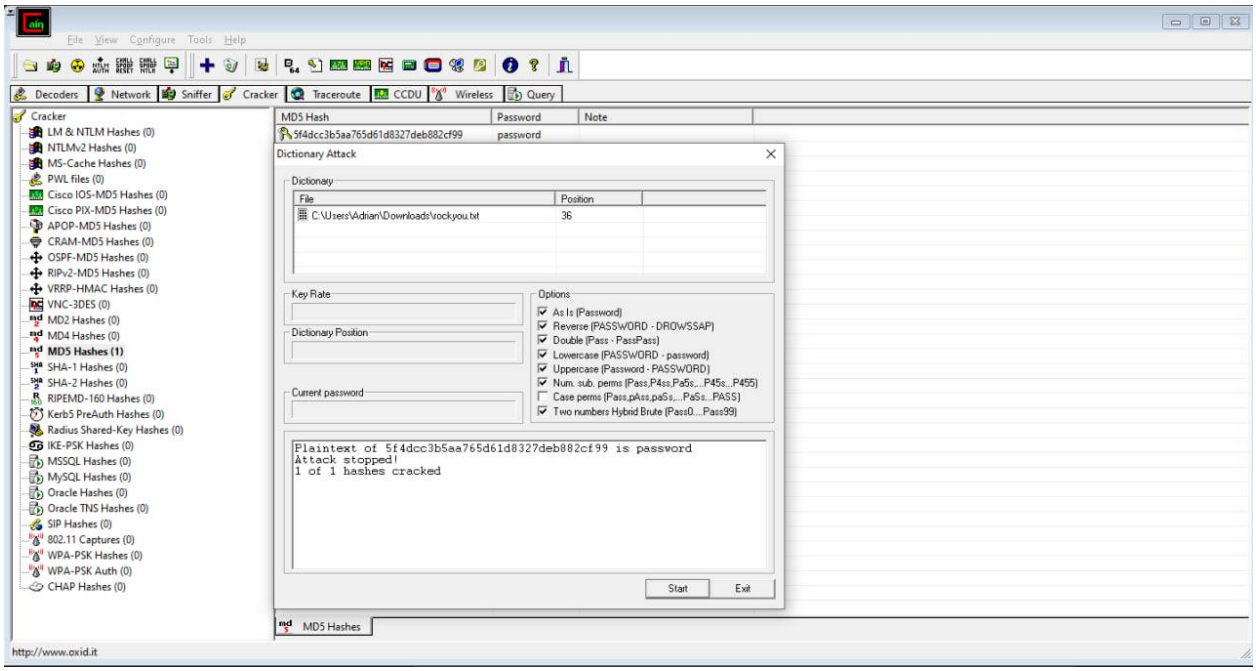

Figura 1.3 – Aplicația “Cain & Abel”

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

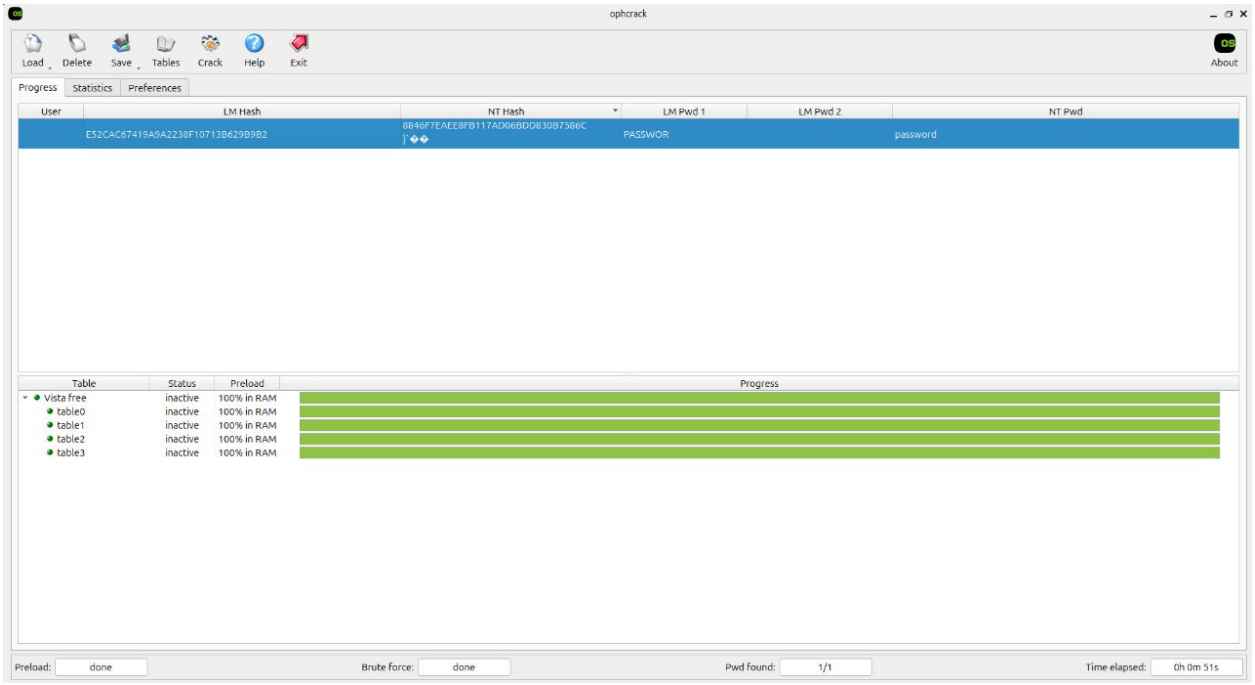

Figura 1.4 – Aplicația "Ophcrack"

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 suport 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.

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 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.

2.1.1 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. Acestea oferă o reprezentare vizuală a tuturor scenariilor în care utilizatorul interacționează 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.

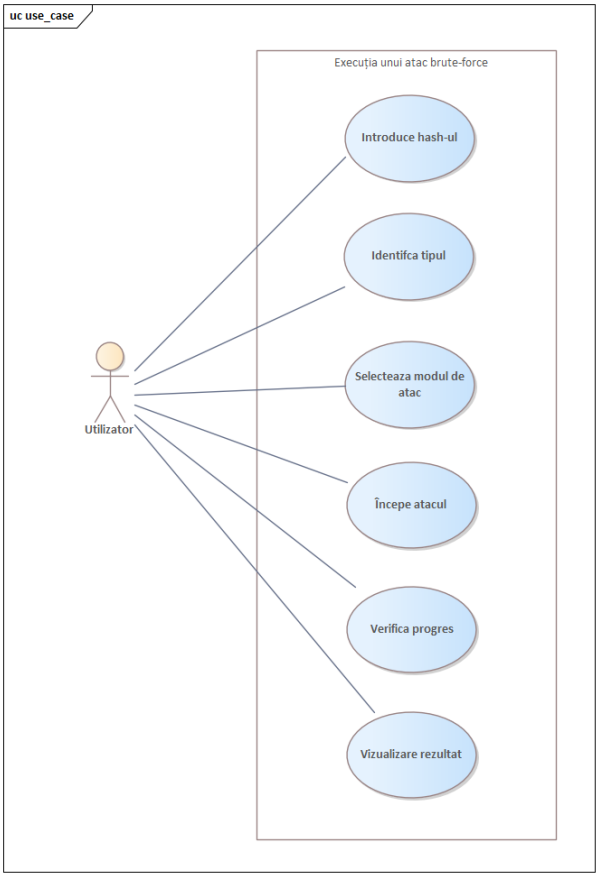

Figura 2.1 – Funcționalul general al sistemului

Diagrama de mai sus reprezintă diagrama cazurilor de utilizare 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.

2.1.2 Fluxul sistemului

Fluxul unui sistem informatic reprezintă succesiunea logică a proceselor și interacțiunilor dintre utilizator și componentele aplicației. În cadrul aplicației dezvoltate, fluxul de execuție al unui atac brute-force este esențial pentru înțelegerea modului în care utilizatorul interacționează cu sistemul și cum acesta procesează fiecare etapă a atacului.

Diagrama de activitate joacă un rol crucial în reprezentarea vizuală a acestui flux, evidențiind pașii principali ai procesului, deciziile critice și ramificațiile posibile în funcție de diverși factori, cum ar fi tipul de hash detectat sau metoda de atac selectată. Prin utilizarea unei astfel de diagrame, se poate optimiza arhitectura sistemului, se pot identifica eventuale blocaje și se poate asigura o execuție eficientă a operațiunilor. Totodată, aceasta facilitează înțelegerea logicii aplicației atât pentru dezvoltatori, cât și pentru utilizatori sau alți factori implicați în proiect.

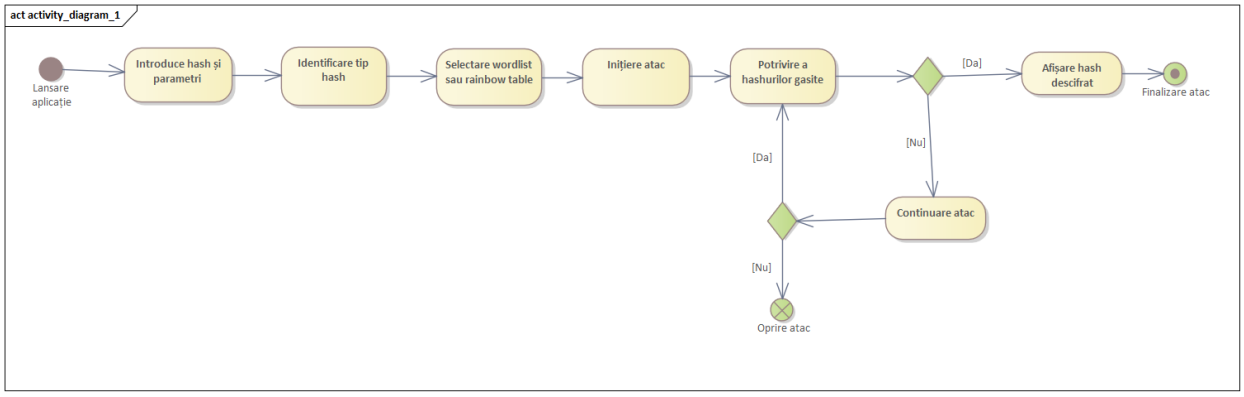

Figura 2.2 – Procesul de cracking

Diagrama din figura 2.2 prezintă fluxul execuției unui atac brute-force pentru spargerea unui hash. Procesul începe cu lansarea aplicației, urmată de introducerea hash-ului și identificarea tipului acestuia. Se selectează metoda de atac, fie prin wordlist, fie prin rainbow tables, apoi se inițiază atacul.

Dacă hash-ul este găsit, acesta este afișat, iar procesul se încheie. În caz contrar, utilizatorul poate alege să continue atacul sau să-l oprească. Această diagramă evidențiază clar succesiunea pașilor și punctele de decizie, optimizând execuția procesului.

2.1.3 Stările sistemului

În orice sistem informatic, gestionarea corectă a stărilor reprezintă un aspect esențial pentru asigurarea unei funcționări eficiente și robuste. Un sistem software poate trece prin diferite stări în funcție de acțiunile utilizatorului, de condițiile interne și de rezultatele proceselor desfășurate. În cazul aplicației dezvoltate, care implementează atacuri brute-force pentru spargerea hash-urilor, este esențial să înțelegem cum se modifică stările în timpul execuției și ce factori determină tranzițiile dintre acestea.

Diagrama de stări este un instrument fundamental în analiza și proiectarea sistemelor software, deoarece oferă o reprezentare vizuală clară a diferitelor stări prin care trece sistemul și a tranzițiilor dintre ele. Aceasta ajută atât în procesul de implementare, cât și în depanare, optimizare și scalare a aplicației. Prin definirea corectă a stărilor și a condițiilor de tranziție, se pot preveni blocajele și erorile, asigurându-se o execuție fluidă și previzibilă a funcționalităților.

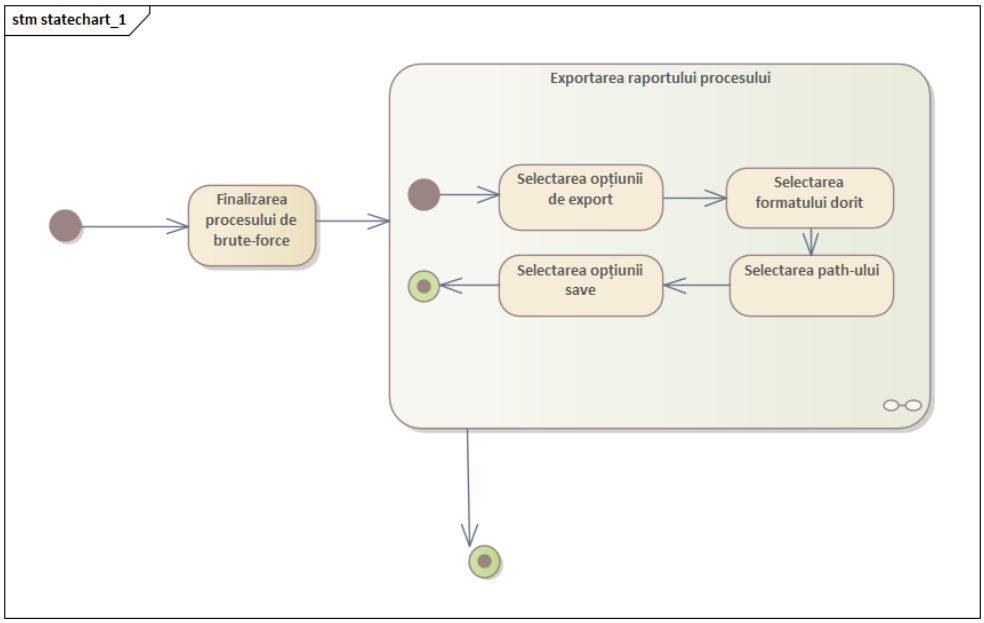

Figura 2.3 – Procesul de exportare a raportului

Diagrama din figura 2.1.3 ilustrează tranzițiile prin care trece sistemul în timpul execuției unui atac brute-force asupra unui hash. Procesul începe în starea Inactiv, unde sistemul așteaptă introducerea hash-ului și a parametrilor necesari. După primirea acestor date, aplicația trece în starea Prelucrare date, unde identifică tipul de hash și selectează metoda optimă de atac.

Odată stabilită metoda, sistemul intră în Executare atac, unde generează și testează posibile valori. Dacă hash-ul este descifrat, aplicația trece în starea Finalizare, afișând rezultatul și încheind procesul. Dacă atacul nu reușește, utilizatorul poate opta pentru Continuarea atacului, revenind la procesul de generare și verificare, sau poate decide oprirea atacului, ceea ce determină tranziția către starea Oprit, unde sistemul încheie toate operațiunile și revine la starea inițială.

Această diagramă evidențiază succesiunea logică a stărilor și punctele de decizie cheie, contribuind la optimizarea fluxului aplicației.

2.1.4 Descrierea utilizării sistemului

Utilizarea unui sistem informatic presupune interacțiunea utilizatorului cu diverse componente ale acestuia, fiecare având un rol bine definit în procesul de execuție. În cazul aplicației de atac brute-force pentru spargerea hash-urilor, utilizatorul inițiază procesul prin introducerea datelor necesare, iar sistemul răspunde executând diferiți pași până la obținerea unui rezultat.

Diagrama secvențială este un instrument esențial pentru înțelegerea modului în care componentele sistemului comunică între ele, prezentând interacțiunile într-o ordine cronologică. Aceasta permite identificarea pașilor cheie din utilizarea aplicației, evidențiind mesajele schimbate între utilizator și sistem. Printr-o astfel de reprezentare, se poate optimiza arhitectura aplicației, asigurând un flux logic clar și eficient al operațiunilor.

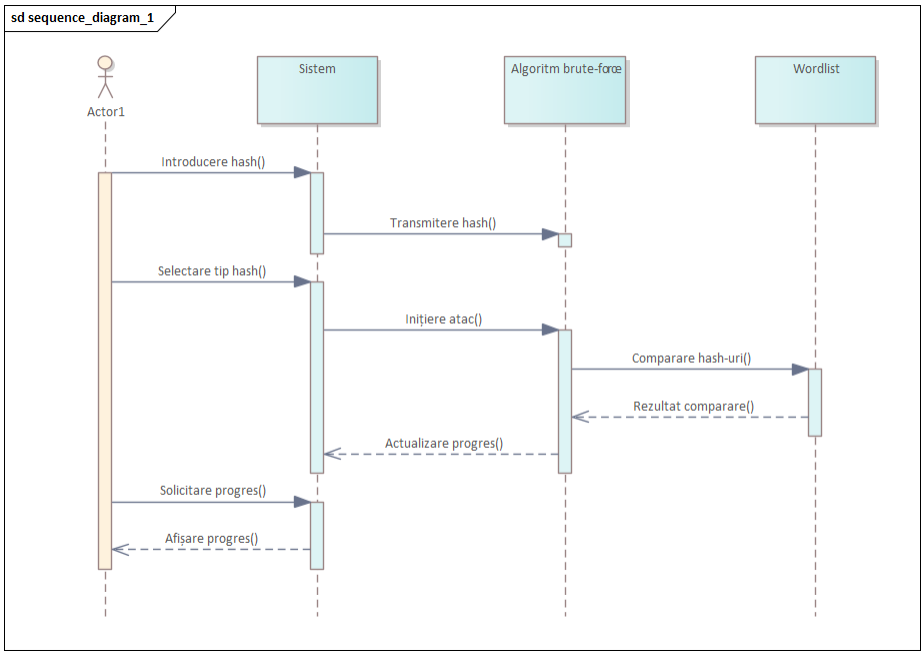

Figura 2.4 – Afișarea progresului procesului

Diagrama din figura 2.4 ilustrează interacțiunile dintre utilizator și sistem pe parcursul execuției unui atac brute-force asupra unui hash. Aceasta evidențiază succesiunea acțiunilor și schimbul de mesaje dintre componentele sistemului, reflectând fluxul logic al procesului.

Procesul începe când utilizatorul introduce hash-ul și parametrii, iar sistemul preia aceste date și le analizează pentru a identifica tipul de hash. După determinarea acestuia, sistemul notifică utilizatorul și îi permite să aleagă metoda de atac, fie prin wordlist, fie prin rainbow tables.

După selecția metodei, sistemul inițiază atacul, generând și verificând posibile valori. În timpul execuției, utilizatorul poate solicita verificarea progresului, iar sistemul răspunde cu actualizări despre stadiul curent al procesului. Dacă hash-ul este spart, aplicația returnează rezultatul final și procesul se încheie. În cazul în care atacul nu reușește, utilizatorul are opțiunea de a continua atacul sau de a-l opri.

2.2 Descrierea structurală a sistemului

Descrierea structurală a unui sistem informatic în conformitate cu standardul UML se realizează printr-o serie de diagrame specializate, fiecare evidențiind diferite aspecte ale organizării și interacțiunii componentelor sale. Aceste diagrame, precum diagrama de clasă și diagrama de componentă oferă o reprezentare statică a modului în care elementele sistemului sunt definite și conectate.

Diagramele structurale joacă un rol esențial în modelarea și înțelegerea arhitecturii unui sistem, punând accent pe relațiile dintre entități și pe organizarea componentelor software. Acestea nu doar că definesc structura statică a sistemului, dar evidențiază și relațiile de asociere, agregare și compoziție dintre elemente, contribuind astfel la o proiectare clară și coerentă.

Diagrama de clasă reprezintă una dintre cele mai importante diagrame structurale, deoarece definește entitățile sistemului, atributele și metodele acestora, precum și relațiile dintre clase. Aceasta oferă o imagine detaliată asupra modului în care obiectele interacționează și se asociază într-un model software.

Pe lângă diagrama de clasă, diagramele de obiecte ilustrează instanțele specifice ale claselor la un anumit moment în timp, furnizând o perspectivă asupra stării sistemului în execuție. Aceste diagrame ajută la înțelegerea relațiilor dintre obiectele create și modul în care acestea sunt utilizate în cadrul sistemului.

Diagrama de componentă prezintă organizarea modulelor software și a pachetelor din cadrul aplicației. Aceasta este esențială pentru gestionarea complexității sistemului și pentru clarificarea interfețelor dintre diferitele componente software.

Prin utilizarea acestor diagrame structurale, dezvoltatorii pot avea o înțelegere clară a arhitecturii sistemului, facilitând colaborarea în echipă și documentarea detaliată a proiectului. Aceste reprezentări vizuale contribuie la o proiectare bine definită, reducând ambiguitățile și asigurând o implementare coerentă a soluției software.

2.2.1 Descrierea structurii sistemului

Structura unui sistem software este esențială pentru înțelegerea modului în care componentele sale sunt organizate și interconectate. În cadrul modelării UML, diagrama de clasă este unul dintre cele mai importante instrumente utilizate pentru reprezentarea statică a arhitecturii unui sistem, oferind o viziune clară asupra entităților care îl compun, relațiilor dintre acestea și comportamentului lor.

Diagrama de clasă evidențiază principalele clase, atributele și metodele acestora, precum și asocierile dintre diferitele entități din sistem. Aceasta permite identificarea relațiilor de moștenire, agregare și compoziție, oferind astfel o perspectivă detaliată asupra organizării și interdependențelor dintre componentele software. Prin intermediul acestui tip de diagramă, se poate defini clar arhitectura sistemului, facilitând procesul de dezvoltare, întreținere și extindere a aplicației.

În cazul aplicației de atac brute-force asupra hash-urilor, diagrama de clasă descrie modul în care sunt structurate entitățile principale, cum ar fi gestionarea hash-urilor, metodele de atac, algoritmii utilizați și interfața cu utilizatorul. Această reprezentare statică ajută la clarificarea funcționalităților fiecărei componente și la optimizarea procesului de implementare.

Prin utilizarea diagramelor de clasă, echipa de dezvoltare poate obține o înțelegere profundă a arhitecturii sistemului, reducând riscul erorilor și asigurând o implementare coerentă și scalabilă.

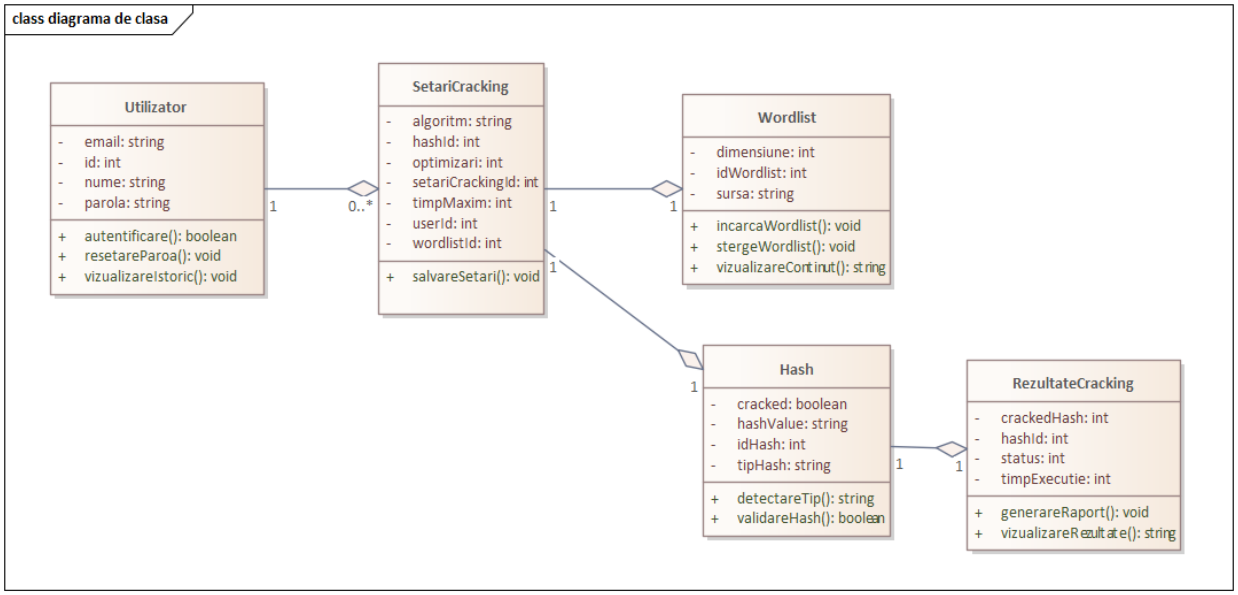

Figura 2.5 – Diagrama de clase a sistemului

Diagrama din figura 2.5 oferă o reprezentare structurală a principalelor entități din cadrul aplicației de cracking hash prin brute-force și relațiile dintre acestea.

Clasa Utilizator conține atributele de identificare, cum ar fi email, id, nume și parolă, și metodele necesare pentru autentificare, resetarea parolei și vizualizarea istoricului. Aceasta este asociată cu clasa SetariCracking, care gestionează parametrii unui atac, inclusiv algoritmul utilizat, optimizările, identificatorul hash-ului și timpul maxim de execuție.

Clasa Wordlist reprezintă listele de parole utilizate în atac, având atribute precum dimensiunea, sursa și identificatorul wordlist-ului, și metode pentru încărcarea, ștergerea și vizualizarea conținutului. Aceasta este conectată cu SetariCracking, indicând faptul că un atac poate folosi o anumită wordlist.

Clasa Hash gestionează informațiile despre hash-ul care trebuie spart, incluzând valoarea hash-ului, tipul acestuia și un indicator dacă a fost spart sau nu. Metodele sale permit detectarea tipului de hash și validarea acestuia.

După execuția atacului, rezultatele sunt stocate în clasa RezultateCracking, care conține hash-ul descifrat, timpul de execuție și statusul procesului. Aceasta dispune de metode pentru generarea unui raport și vizualizarea rezultatelor.

Diagrama reflectă clar structura și relațiile dintre componentele sistemului, evidențiind fluxul de date dintre utilizator, configurarea atacului, procesarea hash-ului și obținerea rezultatelor finale.

2.2.2 Relațiile dintre componenetele sistemului

Relațiile dintre componentele unui sistem software sunt esențiale pentru asigurarea unei funcționări coerente și eficiente. În cadrul unui sistem complex, diagramele de componentă joacă un rol important în reprezentarea structurii modulare și a interacțiunii dintre diferitele părți ale aplicației. Aceste diagrame ajută la vizualizarea modului în care sunt organizate componentele software, relațiile dintre ele și modul în care acestea comunică între ele.

În contextul aplicației de cracking hash prin brute-force, diagrama de componentă reflectă modul în care sunt distribuite și interconectate principalele module ale aplicației, astfel încât procesul de

atac și generare a rezultatelor să fie realizat eficient. Fiecare componentă îndeplinește o funcție specifică, iar prin intermediul acestei diagrame, se poate înțelege cum se realizează comunicarea și coordonarea între aceste funcționalități.

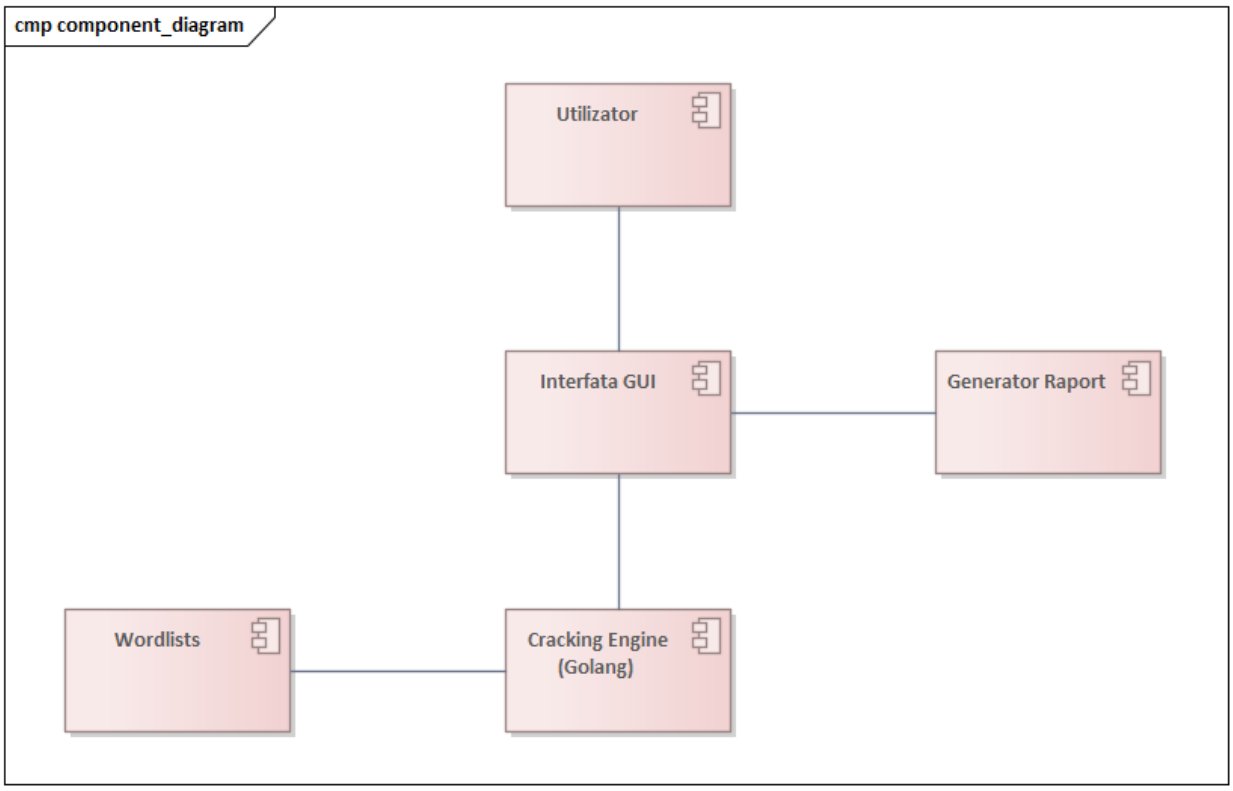

Figura 2.6 – Componentele sistemului

Diagrama de componentă ilustrează relațiile între principalele module ale sistemului. La centrul aplicației se află componenta Interfața GUI, care servește drept punct de interacțiune principal între utilizator și aplicație. Aceasta este conectată cu Utilizatorul, care interacționează cu interfața pentru a introduce datele necesare și a vizualiza rezultatele.

Componenta Interfața GUI este, de asemenea, conectată cu Generatorul de Raport, care se ocupă cu crearea unui raport final ce sumarizează rezultatele obținute în urma atacului. Mai mult, Cracking Engine (Golang) este componenta principală responsabilă de procesul de spargere a hash-urilor, iar aceasta este conectată cu Wordlists, care conține liste de parole folosite pentru atacuri. Această componentă joacă un rol important în optimizarea și alegerea parolelor pentru atacul brute-force.

Diagrama subliniază modul în care componentele sunt interconectate și cum informațiile circulă între ele pentru a asigura execuția corectă a aplicației, de la interacțiunea utilizatorului până la procesarea datelor și generarea rapoartelor.

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