UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Lucrare de licență

Aplicație pentru menținerea securității serverelor de baze de date

propusă de

Stoica Tedy

Sesiunea: iulie, 2024

Coordonator științific

Conf. Dr. Andrei Panu

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Aplicație pentru menținerea securității serverelor de baze de date

Stoica Tedy

Sesiunea: iulie, 2024

Coordonator științific

Conf. Dr. Andrei Panu

Anexa II

Avizat,
Îndrumător Lucrare de Licenţă
Conf. Dr. Andrei Panu,
Data _____ Semnătura ____

DECLARAȚIE privind autenticitatea conţinutului lucrării de licenţă

Subsemnatul(a) Stoica Tedy domiciliat(ă) în România, Iași, Iași, Str. Tăietoare, nr. 16, sc. TR.1, et.1, ap.3, născut(ă) la data de 14.04.2003, identificat(ă) prin CNP 5030414226720, absolvent(ă) al(a) **Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de Informatică**, specializarea Informatică, promoţia 2021-2024, declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă/diplomă/disertaţie/absolvire cu titlul: **Aplicație pentru menținerea securității serverelor de baze de date** elaborată sub îndrumarea dlui Conf. Dr. Panu Andrei, este autentică, îmi aparține şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea autenticității, consimțind inclusiv la introducerea conţinutului său într-o bază de date în acest scop. Declar că lucrarea de faţă are exact acelaşi conţinut cu lucrarea în format electronic pe care profesorul îndrumător a verificat-o prin intermediul software-ului de detectare a plagiatului.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diplomă sau de disertaţie şi în acest sens declar pe proprie răspundere că lucrarea de faţă nu a fost copiată, ci reprezintă rodul cercetării pe care am întreprins-o.

Data _____ Semnătură student

Anexa III

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „**Aplicație pentru menținerea securității serverelor de baze de date**”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent Stoica Tedy

(semnătura în original)

Cuprins

Introducere

În noua eră digitală, menținerea securității aplicațiilor devine din ce în ce mai importantă, aceasta menținându-ne viețile personale private de ochii altor persoane. Însă, menținerea securității aplicațiilor este doar un pas necesar în necesarul de efort pe care trebuie făcut în scopul securității. Un alt aspect important care devine din ce în ce mai vizibil este securitatea precară pe care oamenii o fac legat de bazele de date folosite, întrucât aceștia pot lăsa expuse porturile serverelor de baze de date la internet, folosi parole slabe sau chiar introduce parolele bazelor de date folosite în interiorul codului, fără a folosi variabile de sistem sau alte tehnici asemănătoare. Această lucrare de licență dorește să rezolve ultimele probleme menționate, prin scanarea configurațiilor bazelor de date folosite de o aplicație și prin monitorizarea permanentă a secretelor care se află în locații precum repertorii Git, imagini Docker sau chiar liste publice de parole.

Docker este o platformă digitală folosită din ce în ce mai frecvent datorită ușurinței pe care o oferă pentru a ajuta aplicațiile să fie mai scalabile și mai ușor de manevrat cu ajutorul containerelor. Containerele sunt instanțe ale imaginilor Docker, care pot fi rulate independent una de alta pe același sistem. Acestea pornesc de la anumite imagini Docker, care reprezintă un “șablon” al aplicației. Acestea sunt construite prin intermediul unor automatizări care pot executa comenzi de la o imagine de start. Aceste comenzi pot introduce în mod involuntar secrete în imaginile Docker, care vor putea fi căutate sau chiar utilizate de actori malițioși. Din acest motiv, această aplicație propune scanarea imaginilor Docker pentru secrete.

O altă sursă de secrete comună este Git, întrucât este cea mai utilizată aplicație pentru management-ul codului sursă al aplicațiilor. Datorită ușurinței cu care se pot realiza commit-uri și immutabilității commit-urilor Git[9], este foarte posibil ca pe parcursul aplicațiilor care sunt dezvoltate pe lungă durată să fi avut în vreunul dintre commit-uri scăpate secretele care vor fi folosite pentru autentificarea cu bazele de date. Din acest motiv, propunem și această sursă de secrete pentru aplicația pe care urmează să o dezvoltăm.

Ca și baze de date pe care această aplicație le urmărește, au fost incluse cele mai populare baze de date open-source, și anume PostgreSQL, MySQL, MongoDB și Redis. PostgreSQL și MySQL sunt baze de date relaționale, bazate pe dialectul SQL și au împreună aproximativ 60% [16, 20] din totalul cotei de piață a bazelor de date relaționale. MongoDB este o baza de date nerelațională care deține 44% din totalul cotei de piață a bazelor de date nerelaționale. Redis este o baza de date cheie-valoare folosită în general pentru caching, acesta având 90% din cota acestei piețe.

Principalul mod de organizare la nivel de aplicație vor fi organizațiile, care vor avea mai multe proiecte. Fiecare organizație are o listă de utilizatori, dintre care fiecare are un nivel de acces. Pentru fiecare organizație există un singur deținător (“Owner”), acesta putând să editeze rolul oricărui utilizator din organizație, inclusiv să ofere rolul de deținător altui utilizator, pierzându-și rolul de deținător între timp. Mai există și rolul de Administrator, care are aceleași permisii ca rolul de deținător, însă acesta nu poate oferi altor persoane rolul de deținător. Ultimul rol care mai există este rolul de vizualizator(“Viewer”), care nu permite adăugarea altor utilizatori/baze de date/surse, însă care poate rula scanări pe bazele de date și sursele existente.

Fiecare organizație poate avea mai multe proiecte, care sunt create de Administratorii sau Deținătorii organizației. Fiecare proiect reprezintă locația unde se vor crea o grupare de una sau mai multe repertorii Git, imagini Docker sau baze de date. Aceste grupări reprezintă nivelul la care se vor realiza scanările, credențialele din aceste surse fiind folosite doar la nivelul acestor baze de date.

În următoarele capitole vom prezenta o aplicație scrisă în limbajul de programare Golang care oferă administratorilor o modalitate de a scana într-un mod automatizat existența vulnerabilităților în sistemele de baze de date menționate mai sus, existența unor credențiale valide în interiorul repertoriilor Git sau a imaginilor Docker, dar și existența utilizatorilor cu parole ușor de compromis folosind atacuri de tip forță brută. Aplicația va fi scalabilă, fiind implementată folosind o arhitectură distribuită. Aceasta va suporta și un mod de funcționare la “distanță”, prin intermediul unui worker extern, care va prelua comenzi de la backend-ul central și va executa scanările necesare. În plus, în cadrul acestei licențe va mai fi prezentată și o modalitate de integrare a licenței în interiorul sistemelor CI (Continuous Integration). Aplicația mai oferă și o interfață intuitivă, realizată folosind framework-ului Svelte.

Contribuții

Pe parcursul acestei licențe am analizat metodele de criptare a parolelor în mai multe baze de date, metodele prin care un utilizator ar putea accesa aceste parole criptate, dar și configurațiile puse la dispoziție de aceste sisteme de baze de date. În cadrul acestei licențe am ales mai multe dintre cele mai importante configurații care pot afecta securitatea în mod negativ, și am creat modalități prin care să se poată anunța administratorii acestora. Fiecare bază de date deține o modalitate (sau mai multe) de a verifica că o parolă este asociată unui utilizator, iar aplicația descrisă de noi implementează toate aceste modalități în interiorul limbajului Go, fără a mai fi nevoie astfel de implementările originale ale bazelor de date.

În plus, am analizat modalitatea de stocare a obiectelor din repertoriile Git, și am implementat o soluție care permite parcurgerea tututor commit-urilor și analizarea individuală a fiecăruia. În plus, soluția abordată oferă și o eficiență mare pe repertoriile mari, întrucât analiza se face doar pe liniile modificare în interiorul unui commit, evitându-se astfel risipa energiei. Această abordate oferă posibilitatea scanării secretelor, chiar și dacă acestea au fost șterse la un moment dat prin intermediul altui commit.

O altă contribuție importantă este analizarea modalității de stocare a imaginilor Docker, și implementarea unei soluții care va putea verifica prezența secretelor chiar și în interiorul imaginilor în care acestea au fost șterse explicit, folosind imutabilitatea straturilor imaginilor Docker.

Acest proiect de licență a conținut și studierea API-ului de la National Vulnerability Database (NVD), pentru a putea extrage vulnerabilitățile asociate unei versiuni de bază de date, dar și implementarea unei soluții de actualizare automată a vulnerabilităților, și de folosire a acestor informații în interiorul scanărilor efectuate[19].

În cadrul acestei licențe a fost realizată și proiectarea unei soluții bazate pe o arhitectură distribuită, care va suporta un număr foarte mare de proiecte și utilizatori. Implementarea acestei soluții s-a realizat folosind limbajul de programare Go, pentru partea de backend, worker extern și de Continuous Integration. Pentru partea de frontend s-a folosit Svelte, un framework scris în Javascript.

În plus, s-a realizat și o extensie pentru librăria `authboss`[25], care este una dintre cele mai populare librării de autentificare pentru Go, pentru a lăsa utilizatorii să se conecteze folosind protocolul WebAuthn, care este redenumit și ca Passkey. Extensia a

fost implementată într-un alt repertoriu Git față de restul proiectului de licență (https://github.com/Tedyst/authbosswebauthn).

Capitolul 1. Descrierea aplicației de menținere a securității bazelor de date

O aplicație de menținere a securității bazelor de date ar trebui să fie capabilă să realizeze scanarea automată a mai multor surse de secrete, precum repertorii Git, imagini Docker, sau chiar baze de date, pentru a putea identifica și notifica administratorii în cazul în care au fost introduse credențiale valide. Aceasta ar trebui să fie capabilă să realizeze scanarea bazelor de date pentru a identifica configurații care pot produce vulnerabilități, precum și să realizeze scanarea versiunilor bazelor de date folosite pentru a identifica vulnerabilități cunoscute. În plus, ar trebui să fie capabilă să realizeze verificarea combinațiilor nume/parolă asupra bazelor de date, precum și să folosească workeri separați de aplicația principală pentru a putea accesa baze de date din spatele firewall-urilor. În plus, ar trebui să folosească o metodă pentru a opri build-urile aplicațiilor clienților în cazul în care s-au introdus credențiale valide, precum și să notifice Administratorii acestora în cazul în care au fost introduse credențiale valide.

1.1. Problema adresată

În ultimii ani s-a observat o creștere seminficativă a numărului de atacuri asupra bazelor de date, acestea fiind de obicei considerate un punct slab al aplicațiilor.
Acestea sunt atacate în mod indirect prin metode precum SQL Injection[23], care oferă atacatorilor acces la toate datele din baza de date, până la atacuri de forță brută care, dacă reușite, vor oferi atacatorilor acces la întreaga bază de date, chiar dacă aplicația care folosea baza de date era sigură. Din cauza că din ce în ce mai multe aplicații folosesc utilitare de tipul ORM, care simplifică codul și elimină posibilitatea erorilor de tipul SQL Injection, a început să crească atenția atacatorilor asupra bazelor de date din spatele aplicațiilor, întrucât acestea au rămas una dintre puținele căi prin care se poate obține accesul total la datele unei aplicații.

Alte tipuri de atacuri care se pot realiza asupra bazelor de date sunt exploit-urile, care pot sau pot sa nu fie folosibile din internet, dacă nu există deja o pereche de credențiale valide. Aceste exploit-uri sunt de fapt bug-uri la nivelul codului bazelor de date, și deși rare, sunt extrem de rele din punct de vedere al securității.
O altă modalitate comună prin care se pot întâmpla atacuri asupra bazelor de date sunt cele de tipul “hardcoding”, în care credențialele folosite de aplicație sunt menționate direct în cod sau publicate într-o altă formă pe internet (fișiere de configurare, imagini cu parolele, imagini Docker etc.).

1.2. Soluția propusă

Propunem în licența descrisă mai jos o aplicație care va scana în mod automat și continuu mai multe repertorii Git și imagini Docker (care vor fi adăugate înainte de un administrator al aplicației), și va monitoriza bazele de date folosite de aplicație astfel încât să nu existe nici o pereche de credențiale valide uitată. În plus, se va monitoriza în mod activ și versiunile bazelor de date, întrucât acestea pot fi foarte vechi, sau pot conține vulnerabilități recente (0-day exploits). Aplicația va fi realizată în limbajul Golang, un limbaj de programare creat de Google în 2009, care este asemănător cu C, și care ușurează scrierea programelor concurente și eficiente.

Aplicația va suporta un mecanism de forță brută prin care, dacă o bază de date suportă, se vor face verificări de combinații de username/parolă fără să fie necesară încercarea conectării la baza de date cu fiecare combinație, întrucât ar dura extrem de mult. Pentru a realiza acest lucru, va fi necesară oferirea aplicației noastre a unor credențiale mai elevate, care vor putea lua direct hash-ul parolelor pentru toți utilizatorii bazelor de date și cu care se va putea realiza verificarea într-o manieră mult mai rapidă și care nu poate conduce la erori.

Erorile pe care le evită această metodă sunt activarea unor mecanisme de prevenție a forței brute, precum “fail2ban”, care va bloca adresa IP a worker-ului în cazul în care va fi greșit parola de mai multe ori, sau chiar blocarea bazei de date care trebuia să fie scanată datorită numărului mare de conexiuni care vor fi încercate simultan, și care va conduce la degradarea performanței aplicațiilor clienților.

Aplicația mai suportă un mecanism prin care, dacă la nivelul unui proiect, s-a verificat pentru un anumit utilizator o pereche de parola, la următoarea scanare parola nu va mai fi verificată, scăpând astfel de o serie inutilă de calcule, și reducând timpul necesar unei scanări în mod considerabil.

Aplicația propusă include și o listă de credențiale cunoscute (din listele de parole “RockYou” sau “Cain-And-Abel”), prin care se poate verifica existența unor parole slabe pentru oricare din conturile existente din bazele de date scanate.

Pentru a verifica existența unor vulnerabilități în bazele de date, aplicația va monitoriza flux-ul de vulnerabilități oferit de NVD (Grupul organizațional care oferă clasificările de tip CVE - Common Vulnerabilities and Exposures), și va verifica în mod activ ca să nu existe vulnerabilități existente pentru versiunea curentă a bazelor de date scanate.

Aplicația va avea și un API care folosește paradigma REST, și care va fi documentat folosind OpenAPI.

1.3. Funcționalitățile aplicației

Pentru a începe utilizarea aplicației, utilizatorul va trebui să se înregistreze folosind interfața grafică. După ce acesta se va înregistra, va fi întâmpinat de o pagină asemănătoare cu figura 1.1.

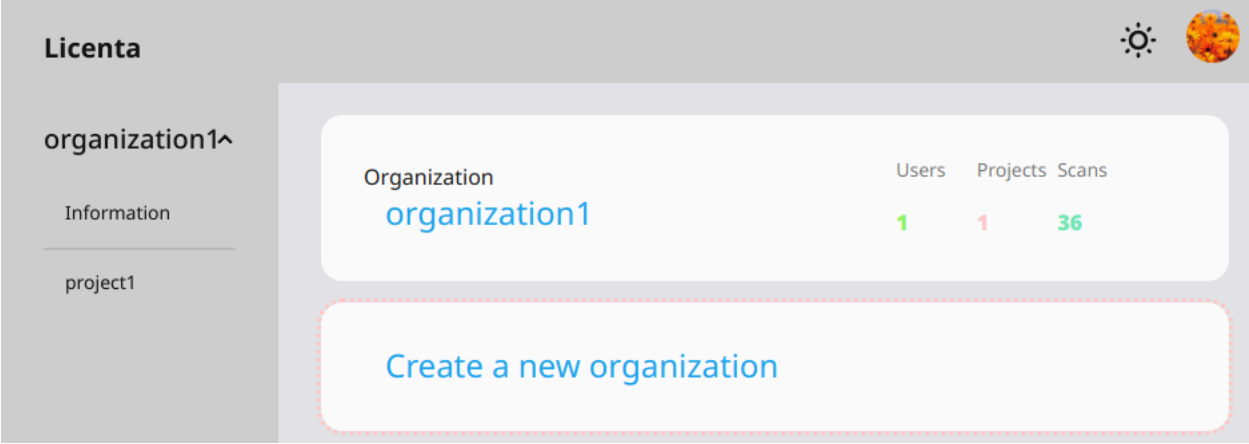

Figura 1.1. Interfața principală

În interiorul acestei pagini, utilizatorul va putea crea o nouă organizație sau accesa una deja existentă. Lista cu organizații și cu proiecte va fi tot timpul vizibilă în bara din stânga, sau folosind meniul de tip hamburger pe telefoane. După ce utilizatorul va selecta organizația pe care dorește să o acceseze, acesta va fi întâmpinat cu o pagină asemănătoare figurii 1.2. În cadrul acestei pagini, utilizatorul va putea crea un nou proiect folosind butonul “Create a new project”, sau accesa pagina specifică unui proiect apăsând pe acesta. În dreptul fiecărui proiect este un buton care va permite ștergerea definitivă a acestuia.

În partea de jos a acestei pagini, se află un buton care permite ștergerea permanentă a organizației. Sub acel buton se află o modalitate prin care Administratorii pe organizație pot vedea, crea și șterge workerii asociați organizației. Fiecare worker are propriul său token, care permite autentificarea de la distanță a acestuia, acest token fiind practic cheia secretă a acestuia. În partea din dreapta, se află toți utilizatorii invitați în cadrul organizației, dar și posibilitatea de a invita persoane noi. Fiecare utilizator are unul dintre cele câteva roluri posibile, și anume “Owner”, “Admin”, “Viewer”. În dreptul fiecărui utilizator se află un buton care permite editarea rolului utilizatorului de către persoane care au permisie mai mare decât el. Spre exemplu, un utilizator cu permisia de “Admin” nu poate fi editat decât de către “Owner”, și poate edita doar utilizatori cu permisia de “Admin” sau “Viewer”. Nu se poate oferi o permisie mai mare decât cea pe care o are utilizatorul care oferă permisia, iar dacă un utilizator își oferă permisia de

“Owner”, acesta va fi retrogradat la “Admin” în mod automat. Interfața prin care se efectuează aceste operații este descrisă în figura 1.3.

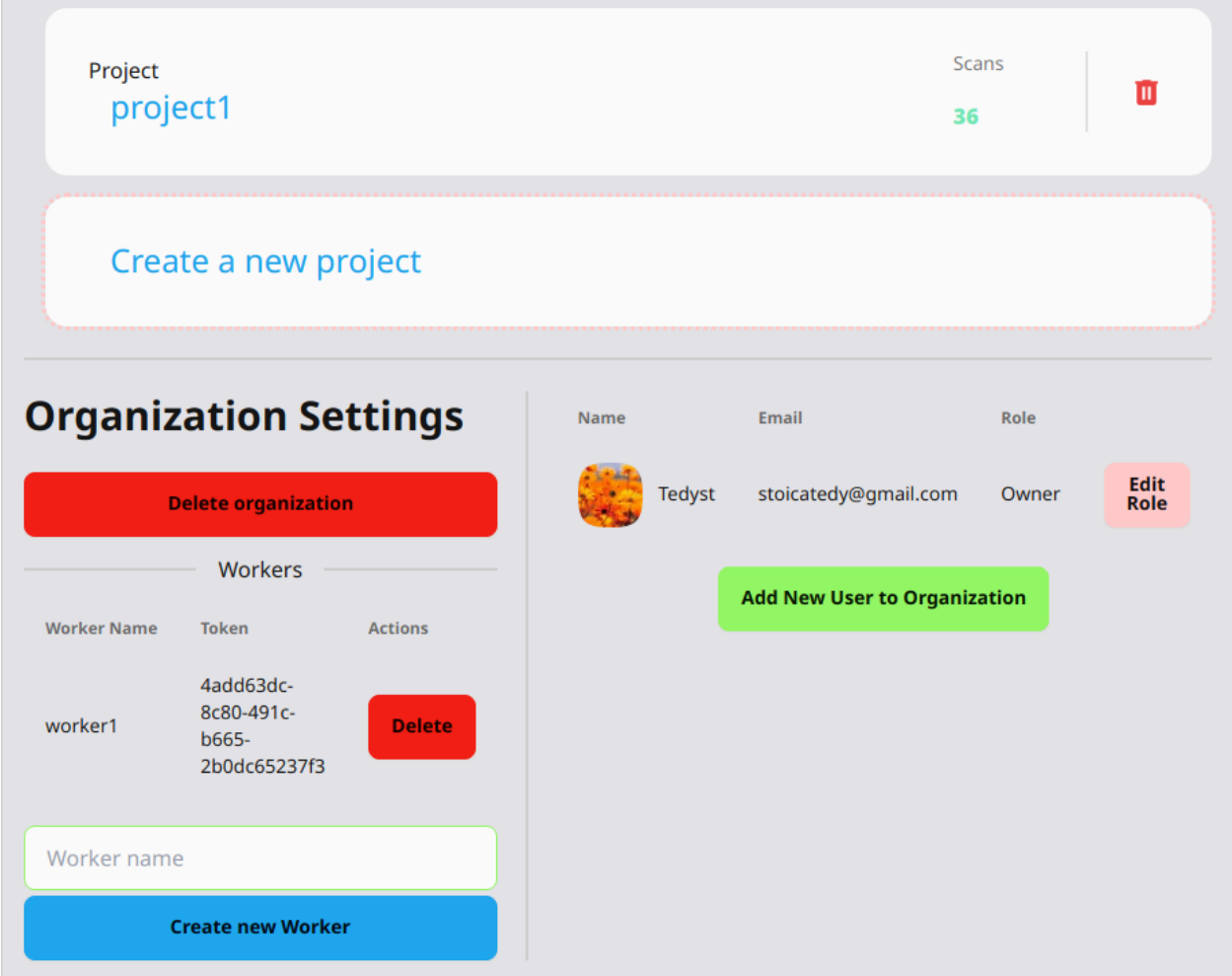

Figura 1.2. Interfața organizațiilor

După ce un proiect a fost selectat, se va afișa o pagină în care se pot modifica toate detaliile specifice acelui proiect, acest lucru fiind prezentat în figura 1.4. În imaginea prezentată, se pot observa trei secțiuni principale, și anume “Project Actions”, “Scanned Databases”, respectiv “Secret Sources”.

În cadrul secțiunii de Project actions, putem adăuga baze de date, care pot fi bazele de date Redis, MongoDB, PostgreSQL sau MySQL, surse de scanare, care pot fi repertoriii Git sau imagini Docker. Tot în cadrul acestei opțiuni putem activa sau dezactiva scanarea folosind workeri externi pentru acest proiect, dar și cere o nouă scanare care să se execute în acest moment.

În cadrul secțiunilor de “Scanned Databases”, respectiv “Secret Sources”, se pot observa bazele de date, respectiv sursele de scanare asociate proiectului curent.

Fiecare dintre acestea poate fi editată sau ștearsă, folosind butoanele din dreapta acestora. Sursele de secrete mai au și posibilitatea vizualizării a id-urilor de commit sau strat Docker care au fost scanate, dar și de vizualizarea secretelor identificate în fiecare, asemeni imaginii 1.5.

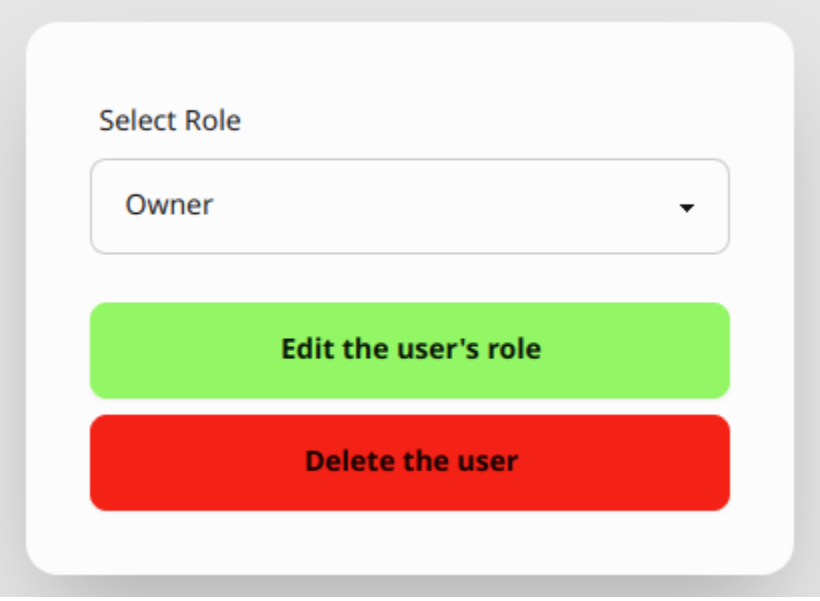

Figura 1.3. Modificarea permisiunilor utilizatorilor unei organizații

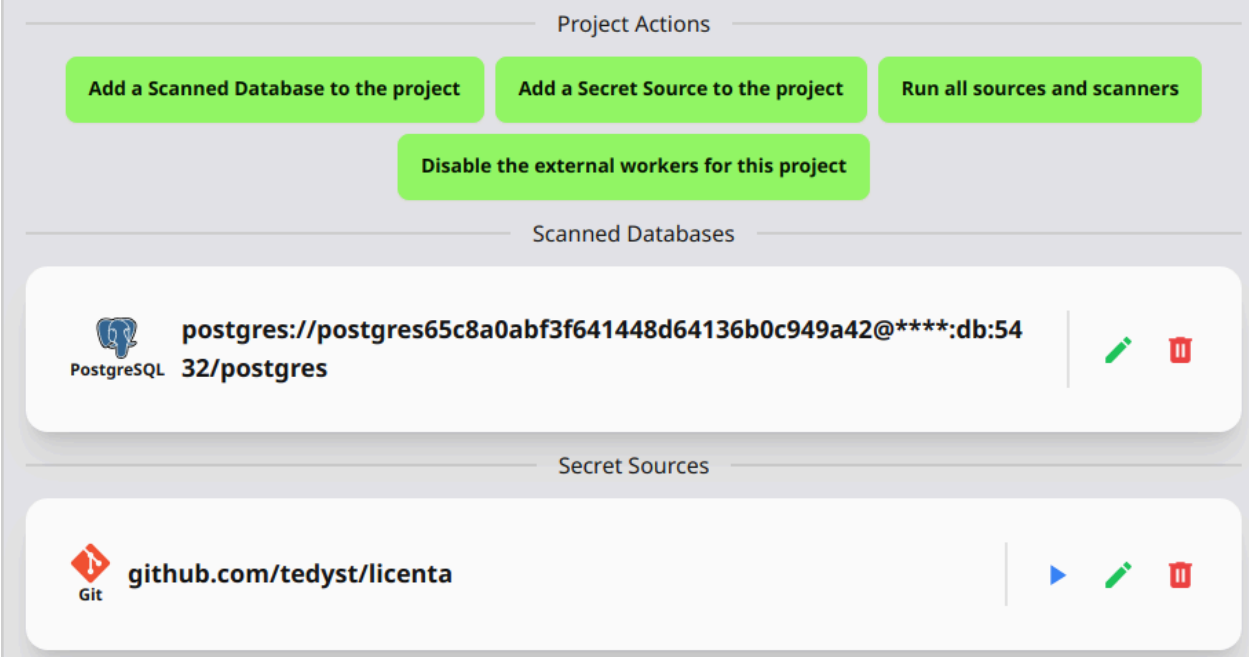

Figura 1.4. Acțiunile, sursele și bazele de date specifice unui proiect

Figura 1.5. Un secret găsit în interiorul unui repertoriu Git

Sub secțiunile prezentate, se află o listă cu toate scanările efectuate în cadrul unui proiect. Această secție este împărțită în mai multe grupuri de scanări (“Scan Groups”). Fiecare grup reprezintă o cerere de scanare a unui proiect și conține câte o scanare pentru fiecare bază de date sau sursă de scanare. Sursele de scanare sunt primele care sunt rulate, acestea oferind scanărilor de baze de date credențialele necesare verificării credențialelor. În imaginea 1.6. se poate observa statusul curent al două scanări, una al unui repertoriu Git, care s-a terminat și care a găsit unul sau mai multe secrete, dar și al unei baze de date de tip Postgres, care încă nu s-a terminat, și care încă nu a generat informații.

Fiecare scanare conține mai multe informații referitoare la modul în care aceasta a derulat, acestea fiind accesibile prin intermediul butonului din dreapta acesteia. După ce un utilizator apasă pe acel buton, va fi întâmpinat de o pagină asemănătoare figurii 1.7. În această pagină, se vor prezenta în partea de sus statusul scanării de tip forță brută, care va afișa numărul de parole încercate și numărul de parole totale, această metrică fiind actualizată în timp real pe parcursul scanării, o dată per secundă. În secția

de mai jos se află rezultatele, care arată pașii executați de fiecare scanare în parte, dar și severitatea informațiilor găsite. Severitatea maximă a acestor informații va decide severitatea totală a scanării.

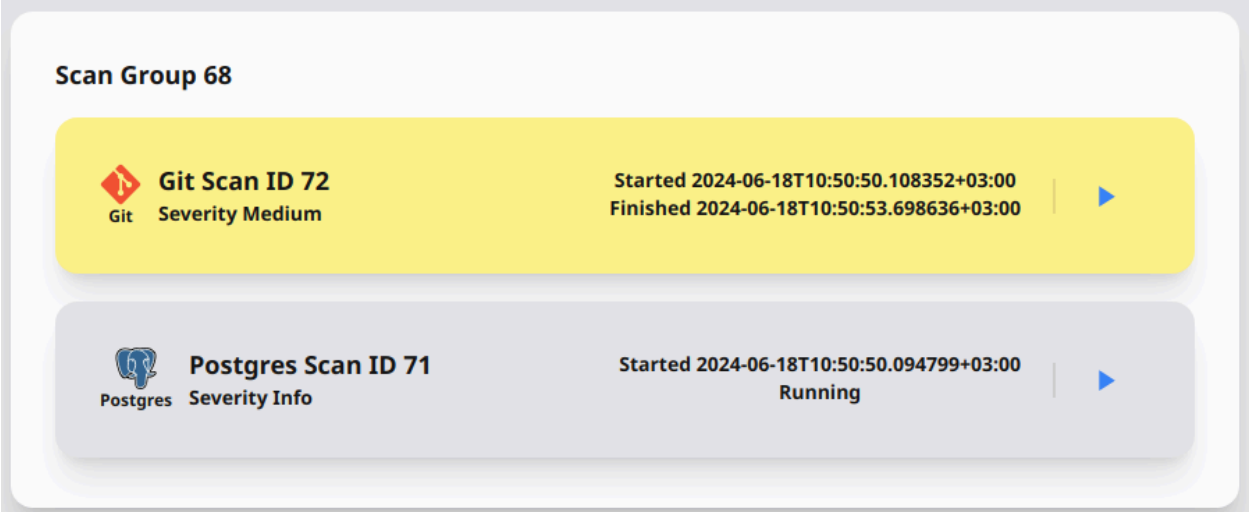

Figura 1.6. Un grup de scanări care conține o scanare a unui repetoriu Git și una a unei baze de date de tip Postgres

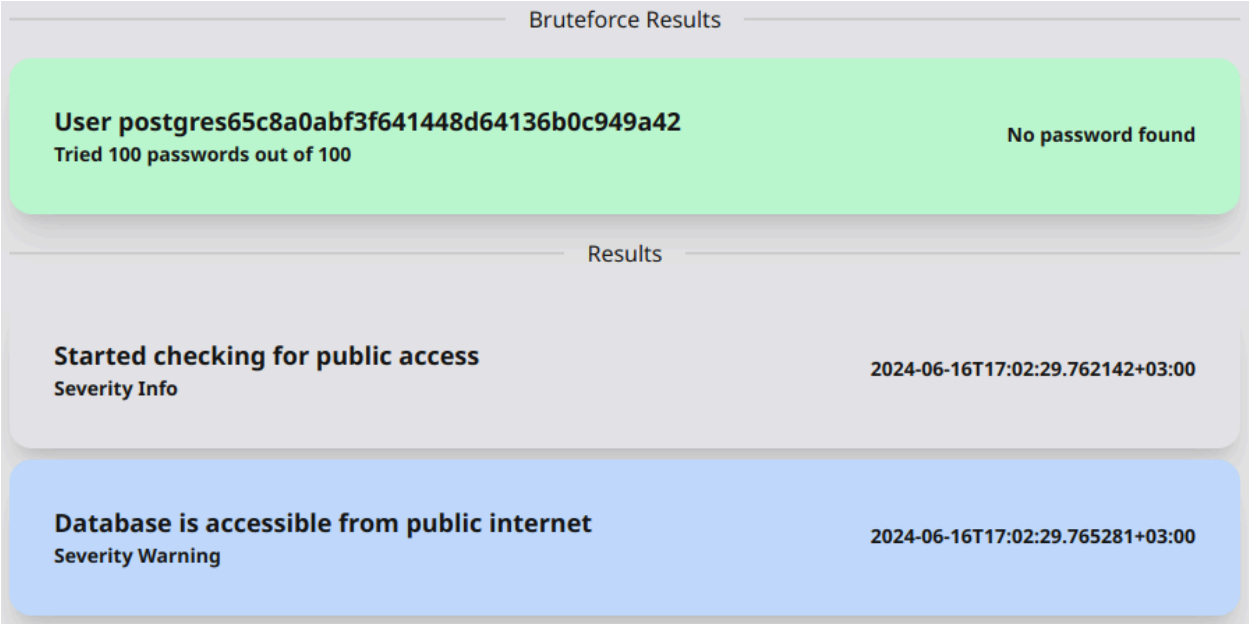

Figura 1.7. Rezultatele unei scanări de bază de date de tip Postgres

Un utilizator poate folosi și un worker extern, per organizație, pentru a putea scana și bazele de date care nu sunt expuse la internet. Pentru această facilitate, utilizatorul se va duce la pagina specifică organizației și va crea un nou worker extern, iar apoi îi va copia token-ul specific acestuia. Apoi, acesta va instala aplicația Go pe un calculator care are acces direct la baza de date, și va rula aplicația în acest mod:

`./licenta worker --worker-token token123123123`, unde token123123123 reprezintă token-ul copiat anterior. Utilizatorul va activa apoi funcționalitatea de worker extern pentru proiectul pe care dorește să îl scaneze, iar mai apoi va rula o scanare folosind aceeași modalitate prezentată anterior.

Pentru a folosi și funcționalitatea de Continuous Integration, pentru a anunța backend-ul că se dorește o nouă scanare pentru un anumit proiect, dar și pentru a verifica statusul scanării, se vor rula aceeiași pași ca la workerul extern, însă se va rula următoarea comandă: `./licenta ci --worker-token token123123123 --project 1`, unde token123123123 reprezintă worker token-ul pentru aplicația CI, iar 1 reprezintă id-ul proiectului, care se poate obține prin intermediul REST API-ului. Această aplicația va aștepta finalizarea tuturor scanărilor din grupul de scanări generat, și va returna un status de eroare dacă severitatea este mai mare decât un prag configurabil (este setat în mod inițial la severitatea Înaltă).

1.3.1. Extragerea datelor de autentificare pentru utilizatorii bazelor de date

Una dintre cele mai comune variante prin care se obține accesul la bazele de date ale aplicațiilor este prin intermediul credențialelor “uitate” prin diferite locații, precum repertorii Git, imagini Docker, etc. Aplicația prezentată dorește să elimine acest canal prin care bazele de date sunt compromise prin monitorizarea constantă a codului sursă și a imaginilor compilate pentru a fi siguri că nu au fost lăsate credențiale cel puțin necriptate folosind tool-uri precum `git-secret`[10].

Scanarea propriu-zisă se face luând fiecare fișier în parte și scanându-l după o serie de filtre regex, verificând fiecare linie în mod individual. Filtrele sunt capabile să și decodeze textul encodat folosind Base-64, în acest caz luându-se valoarea textului decodat. Deoarece aceste filtre au fost făcute să fie cât mai generice, pe rezultatele acestui modul se va aplica un alt mod de filtrare, care este împărțit în mai mulți pași:

- Dacă un secret este sigur să fie util (cum ar fi PostgreSQL connection strings), se returnează probabilitatea 1.
- Se folosește o listă de parole care să fie cu totul ignorate, pentru a mai reduce falsele pozitive.
- Se alege un multiplicator pentru probabilitate, care este de obicei 1, însă este 2 pentru secretele provenite din Base64.
- Se calculează entropia Shannon și se calculează probabilitatea folosind formula $1/(1 + e^{-0.2*(entropia-40)})$. Această formulă ne oferă un număr între 0 și 1, care estimează probabilitatea ca un cuvânt să fie generat aleatoriu folosind funcția logistică.

- Se folosește o listă de cuvinte care reduc probabilitatea (precum “password”, “hash”, etc.), iar dacă secretul nostru conține unul dintre aceste cuvinte, i se reduce probabilitatea cu `30%`.
- Se folosește o altă listă de cuvinte care cresc probabilitatea (precum “postgres”, “redis”, etc.) care cresc probabilitatea cu `100%`.
- Probabilitatea finală se limitează la `1`.
- Dacă probabilitatea finală este mai mare decât `0.7`, atunci secretul este considerat un secret adevărat.

Pentru fiecare pereche de secrete se menține și ultimele linii de dinaintea secretului, pentru a putea fi afișate în interfață. Scanarea secretelor se face folosind toate nucleele mașinii, folosindu-se tehnici de programare concurențială pentru realizarea unei scanări rapide.

1.3.1.1 Extragerea datelor de autentificare din repertoriile Git

Repertoriile Git sunt o colecție de commit-uri și de link-uri către acele commit-uri (refs), fiecare commit reprezentând schimbările efectuate de la unul sau mai multe parent commits. Schimbările sunt de două tipuri, `add` and `delete`. Operația de modificare a unei linii de cod este implementată ca ștergerea acelei linii și apoi adăugara liniei modificate.

Scannerul nostru descarcă întreg repertoriul Git în memorie, și apoi scanează toate commit-urile folosind un utilitar de tipul `git log` din librăria `go-git`[11]. Pentru fiecare commit se scanează în maniera prezentată mai sus doar schimbările adăugate de acel commit (față de toți părinții commit-ului), și apoi se salvează în baza de date toate secretele găsite.

1.3.1.2 Extragerea datelor de autentificare din imaginile Docker

Imaginile Docker sunt o colecție de arhive tar, combinate într-o altă ahrivă tar[14]. Fiecare arhivă tar reprezintă un `layer`, care este creat de câte o instrucțiune din Dockerfile, precum (FROM, ADD, COPY, RUN, etc.). Fiecare layer este format din o listă de fișiere care au fost modificate la nivelul acelei instrucțiuni, toate fișierele din acel layer suprascriind același fișier (dacă există), din layere-le de mai jos.

Scanner-ul nostru funcționează prin scanarea fiecărui layer în mod individual, pentru a elimina situațiile de tipul următor:

- Un layer adaugă fișierul `env.prod` folosind o instrucțiune de tipul `COPY . /app`
- Un alt layer compilează aplicația/modifică alte fișiere

- Un ultim layer execută o operație de tipul `RUN rm -rf /app`, care va șterge tot folderul `/app`

La finalul acestor operații, în imaginea Docker pare că fișierul original din `/app` nu mai există, însă acesta va exista pe disc de fiecare dată când această imagine va fi copiată/rulată. Acest lucru combinat cu faptul că imaginile Docker sunt făcute să fie distribuite, mai ales în contexte precum rularea aplicației pe platforme de tipul Kubernetes, pot compromite securitatea aplicației în mod involuntar.

Scanarea propriu-zisă a fișierelor se execută folosind tehnicile prezentate mai sus la capitolul 1.3.1, salvarea secretelor găsite făcându-se în baza de date a aplicației.

1.3.2. Scanarea configurațiilor care pot produce vulnerabilități în sistemele de baze de date

Un alt factor important în securitatea bazelor de date reprezintă configurarea corectă a bazelor de date în așa fel încât să nu permită conectări anonime, sau atacuri de tip Man-In-The-Middle. Cea mai importantă și comună configurație este utilizarea certificatelor SSL de către utilizatorii bazelor de date și de către bazele de date, întrucât informațiile care circulă nu vor putea fi interceptate de către actorii malițioși.

Fiecare configurație are un nivel de alertă care poate fi configurat la nivel individual, și care va ajuta la stabilirea severității întregii scanări.

1.3.2.1. Scanarea configurațiilor bazelor de date PostgreSQL

Pentru a scana configurațiile bazelor de date PostgreSQL, aplicația va avea nevoie de acces la tabela `pg_settings`, o tabelă specială unde se stochează toate configurațiile bazei de date[21]. Configurațiile scanate sunt cele care vizează utilizarea conexiunilor SSL, cele care ajută la verificarea intruziunilor (activarea de `log_connections` și `log_disconnections`, respectiv cele care ar putea conduce la coruperea datelor (precum dezactivarea `fsync`, `full_page_writes`).

O altă configurație importantă scanată reprezintă `password_encryption`, care stabilește algoritmul de criptare al parolelor noilor utilizatori ai bazei de date. Dacă acesta este `off` (adică parolele sunt stocate în mod plaintext), sau dacă este `md5` (adică parolele sunt hash-uite folosind algoritmul MD5), atunci se va crea o alertă de severitate Înaltă, întrucât acești algoritmi sunt foarte slabi. Configurația `listen_addresses` va da o alertă dacă baza de date nu este montată doar la IP-uri

interne (de tip localhost), întrucât asta ar putea duce la expunerea acesteia la internetul public.

1.3.2.2. Scanarea configurațiilor bazelor de date MySQL

Pentru a scana configurațiile bazelor de date MySQL se va verifica folosind comanda SQL `SHOW VARIABLES`[1], de unde se vor verifica specific aceste configurații:

- `ssl_key`, pentru a fi siguri că baza de date folosește criptarea datelor în tranzit folosind SSL
- `caching_sha2_password_digest_rounds` pentru a verifica că hash-uirea parolelor se face folosind un număr mare de runde, care ar reduce posibilitatea atacurilor de tip brute-force
- `debug` pentru a fi siguri că baza de date nu oferă atacatorilor informații în plus față de un mod `non-debug`
- `flush` pentru a fi siguri că baza de date nu va pierde informații în caz de restart

1.3.2.3. Scanarea configurațiilor bazelor de date MongoDB

Pentru scanarea configurațiilor se va folosi comanda `db.adminCommand({getParameter:"*"})`, care va extrage toți parametrii configurabili ai bazei de date MongoDB care este scanată. Printre configurațiile scanate se numără:

- `scramIterationCount`, care ar trebui să fie peste 10000
- `scramSHA256IterationCount`, care ar trebui să fie peste 15000
- `sslMode`, care nu ar trebui să fie disabled

1.3.2.4. Scanarea configurațiilor bazelor de date Redis

În modul de bază Redis nu folosește autentificare[6], fiind un candidat foarte bun pentru atacuri, dacă acesta poate fi accesat public sau din interiorul rețelelor controlate de atacatori. Pentru a se verifica configurațiile serverului Redis, se va folosi comanda `CONFIG GET`[2], care oferă utilizatorilor conectați informații despre fiecare configurație în parte.

Redis oferă și posibilitatea (care este activată în mod normal, chiar dacă este configurată autentificarea cu utilizator și/sau parolă) să se facă autentificare anonimă, care în general este un lucru nedorit de administratorii care nu știu de această funcționalitate. Aplicația noastră verifică dacă configurația `user default off` este pusă în fișierul de configurare, în caz contrat fiind creată o alertă de severitate Înaltă.

În plus se mai verifică și folosirea criptării TLS a traficului prin existența cheii `tls-key-file-pass`.

1.3.3. Scanarea versiunilor bazelor de date folosite asupra vulnerabilităților cunoscute

Pe parcursul existenței oricărei aplicații, în special cele care sunt folosite de foarte mult timp și de foarte multe persoane, este foarte posibilă apariția unor bug-uri, sau chiar vulnerabilități, care pot compromite securitatea persoanelor care folosesc acele aplicații. Acest lucru este valabil și pentru aplicațiile care oferă servicii de baze de date, oricât de populare ar fi acestea.

Din acest motiv, există echipe întregi care se ocupă de menținerea securității bazelor de date, precum Security Team de la PostgreSQL (security@postgresql.org). Acestea pot primi notificări despre vulnerabilități găsite de persoane de oriunde din lume, și pot aplica reparații pentru acele vulnerabilități, urmând ca după un timp dat (de obicei după ce s-a găsit o soluție pentru acea vulnerabilitate), să facă anunțuri publice de securitate.

Unul dintre cele mai folosite locuri unde se fac anunțuri de acest tip este NVD (National Vulnerability Database), o bază de date organizată de guvernul American, unde sunt incluse majoritatea produselor populare folosite de întreaga lume.

Această bază de date este publică, și oferă detalii despre fiecare vulnerabilitate cunoscută pentru produsele pe care aceasta le urmărește. Întrucât noi urmărim să scanăm doar baze de date foarte cunoscute, precum PostgreSQL, MySQL, Redis sau MongoDB, iar acestea își publică vulnerabilitățile majore către NVD, vom putea să folosim această listă de vulnerabilități să analizăm securitatea bazelor de date pe care le scanăm.

Pentru a putea scana versiunea unei baze de date, mai întâi va trebui descărcată întreaga bază de date NVD care are legătură cu bazele de date pe care vrem să le scanăm. Pentru a descărca doar ce ne interesează, folosim API-ul oferit de NVD și filtrăm după următoarele CPE (Common Platform Enumeration):

- `cpe:2.3:a:postgresql:postgresql`
- `cpe:2.3:a:oracle:mysql`

După ce descărcăm datele despre aceste vulnerabilități folosind un worker care rulează zilnic, vom putea filtra versiunile afectate (întrucât NVD ne oferă o listă cu toate versiunile afectate). Apoi, la fiecare scanare a unei baze de date, vom verifica versiunea

pe care o rulează să nu aibe vulnerabilități cunoscute, iar dacă are, vom crea o alertă cu severitate Înaltă.

1.3.4. Verificarea combinațiilor nume/parolă asupra bazelor de date

Unul dintre cele mai importante funcționalități ale aplicației reprezintă capabilitatea de a încerca combinațiile de username/parole găsite în repertoriile Git sau în imaginile Docker, dar și posibilitatea de a testa parole comun utilizate, din liste publice precum Cain-And-Abel sau Rockyou. Partea de bruteforce se realizează la nivelul worker-ilor care efectuează scanarea efectivă a bazelor de date, efectuând pentru fiecare bază de date o operație diferită explicată mai jos.

Fiecare parolă publică are asignat un identificator unic, monoton crescător, care servește ca identificator al parolelor care au fost verificate până în acel moment. Acest lucru se datorează faptului că noi lucrăm de obicei numai cu hash-uri de parole, dacă hash-ul parolei nu s-a schimbat, suntem siguri că parolele deja testate nu vor fi valide. Pentru a realiza această funcționalitate, vom salva toate hash-urile încercate alături de numele de utilizator pentru care s-a încercat decriptarea, id-ul ultimei parole verificate și proiectul pentru care s-a salvat parola.

La următoarea încercare de decriptare a aceleiași parole pentru același proiect, se va continua doar folosind parole neîncercate deja, sărind-use astfel peste majoritatea computațiilor. La fiecare parolă salvată se mai salvează și parola găsită, în cazul în care o vom găsi, și la următoarele computații se va sări complet peste partea de bruteforce, deoarece știm deja rezultatul căutării.

Pentru parolele care se află în repertoriile Git sau imaginile Docker, acestea vor fi asignate id-ul de -1 și vor fi scanate în același mod cu parolele de mai sus, cu excepția că la fiecare căutare toate parolele găsite în acest mod vor fi rescanate.

Partea de bruteforcing oferă utilizatorilor update-uri în mod regulat (la fiecare secundă), pentru a anunța progresul făcut în a verifica parolele. Progresul este salvat în baza de date și va putea fi verificat folosind API-ul, respectiv frontend-ul aplicației. Dacă o parolă este găsită, atunci se va crea o alertă de severitate Înaltă, pentru a notifica administratorii că se folosește o parolă nesigură.

1.3.4.1. Scanarea parolelor bazelor de date PostgreSQL

Parolele sunt preluate direct din baza de date folosind tabela `pg_catalog.pg_user`[3], Această tabelă conține informații despre toți utilizatorii bazei de date, alături de metoda de criptare și hash-ul parolelor acestora. Accesul la această

tabelă este motivul pentru care aplicația noastră are nevoie de acces de tip superuser la bazele de date Postgres scanate.

Parolele în postgres sunt stocate în 3 modalități diferite:

- `plaintext`, unde parola este salvată fără nici un fel de hashing
- `md5`, unde parola este salvată folosind algoritmul de hashing MD5
- `scramSHA256`, unde parola este salvată folosind algoritmul SCRAM SHA-256

Pentru parolele stocate folosind `plaintext`, pur și simplu este verificată prezența parolei în lista parolelor generice și în lista de secrete găsite în repertoriile Git sau în imaginile Docker.

Pentru parolele stocate folosind `md5`, se folosește librăria standard din Go pentru a se calcula hash-ul parolelor și se compară să fie același hash.
Pentru parolele stocate folosind `scramSHA256`, se extrage salt-ul parolei și numărul de iterații, se extrage `storedKey`, respectiv `serverKey` și apoi se folosește librăria `github.com/xdg-go/scram` pentru a crea un client SHA256 asemănător cu baza de date postgres, pe care va testa fiecare parolă individuală direct în memorie, asemănător cu conectarea la baza de date.

În plus, deoarece fiecare parolă are în componență și numele de utilizator pentru care a fost hash-uită, a fost necesară stocarea în tabela cu parole deja găsite și numele de utilizator pentru parola găsită.

1.3.4.2. Scanarea parolelor bazelor de date MySQL

Scanarea parolelor bazelor de date MySQL se realizează folosind tabela `mysql.user`, care conține informații despre utilizatorii și host-ul de pe care sunt autorizați să se conecteze. Această tabelă conține și hash-urile parolelor criptate folosind algoritmul `caching_sha2_password`. Acest algoritm este unul care în teorie este standard, însă implementarea MySQL este non-standard.

Acest lucru este cauzat de doi factori principali:

- Hash-urile bazei de date MySQL folosesc pentru stocarea salt-ului reprezentarea directă UTF-8 a acelor bytes, iar pentru reprezentarea hash-ului parolei se folosește encodarea base64
- MySQL folosește lungimea unui salt de 20 bytes, iar lungimea folosită de toate celelalte implementări (care sunt conforme standardului) este de 8-16 bytes.

Din acest motiv, nu a fost posibilă folosirea unei imeplentări deja existente a algoritmului de Hashing implementat în Go, fiind necesară reimplementarea acestuia. Implementarea realizată în cadrul acestui proiect de licență a fost bazat pe implementarea din C a algoritmului MySQL de criptare, prin translarea efectivă a tuturor operațiilor (https://github.com/mysql/mysql-server/blob/824e2b4064053f7daf17d7f3f84b7a3ed92e5fb4/mysys/crypt_genhash_impl.cc#L235)[18].

În plus, mysql nu permite extragerea fără modificare din cauza encoding-ului a hash-urilor, motiv pentru care a fost necesară o operație de encoding folosind hex înaintea prelucrării propriu-zise al parolei.
Extragerea a fost realizată folosind următorul SQL query: `SELECT CONCAT(host, ':', user), plugin, CONCAT('$mysql',LEFT(authentication_string,6),'$',INSERT(HEX(SUBSTR(authentication_string,8)),41,0,'$')) AS hash FROM mysql.user WHERE plugin = 'caching_sha2_password' AND authentication_string NOT LIKE '%INVALIDSALTANDPASSWORD%';`

1.3.4.3. Scanarea parolelor bazelor de date MongoDB

Pentru a obține acces la hash-urile utilizatorilor dintr-o bază de date MongoDB se va folosi comanda `db.getUsers({showCredentials: true})`[4]. Această comandă returnează toți utilizatorii și pentru fiecare dintre aceștia poate returna una sau două metode de hash-uire:

- Hash-uirea folosind `SCRAM-SHA-1`
- Hash-uirea folosind `SCRAM-SHA-256`

Pentru ambele tipuri de hash-uri se va calcula dacă acestea există printre parolele cu care urmează sa fie scanată baza de date folosind librăria `go-scram`, menționată anterior.

1.3.4.4. Scanarea parolelor bazelor de date Redis

Pentru a obține acces la toate hash-urile parolelor dintr-o bază de date Redis, se va folosi comanda `ACL LIST`[2], care afișează toți utilizatorii care se pot conecta la baza de date, împreună cu un hash SHA-256 care reprezintă parola acestora. Hash-ul pentru parolele de comparat se va calcula folosind librăria standard Golang `crypto`, restul făcându-se la fel ca la celelalte baze de date.

1.3.5. Folosirea de workeri separați de aplicația principală pentru a putea accesa baze de date din spatele firewall-urilor

Majoritatea companiilor folosesc aplicații care utilizează baze de date protejate de internet, întrucât prin acest mod ar face exploatarea mult mai grea a atacurilor demonstrate mai sus. Problema este că, dacă baza de date nu poate fi accesată prin internet, aceasta nu va putea fi scanată și verificată că respectă necesitățile de securitate.

Din acest motiv, aplicația prezentată aici suportă încă un mod de funcționare, și anume scanarea prin intermediul worker-ilor externi specifici per organizație. Fiecare organizație poate avea asignat unul sau mai mulți workeri, care pot prelua toate scanările care vor fi efectuate în cadrul proiectelor din acea organizație care au activată această funcționalitate. Workerii folosesc același cod pentru scanare ca aplicația principală, însă are beneficiul că poate fi rulat separat de acesta, chiar în interiorul zonei securizate de lângă baza de date care va fi scanată.

1.3.5.1. Metoda de comunicare dintre backend și workeri prin intermediul paradigmei REST

Pentru a comunica cu aplicația principală, backend-ul principal expune o interfață REST prin care workerii pot să efectueze operațiile necesare. Documentația interfeței este realizată folosind OpenAPI v3, pentru a putea ușura development-ul.

Pentru a se realiza atât partea de server și partea de client din interiorul worker-ului, s-a folosit librăria `openapi-codegen` pentru a genera automat cod în Golang pornind de la o specificație OpenAPI. Această metodă asigură corectitudinea implementărilor, dar și folosirea acelorași parametri în interiorul comunicării dintre backend și worker.

Workerii au permisia să execute modificări precum adăugarea de rezultate ale scanării, și să actualizeze statusul scanărilor doar pentru organizațiile pentru care au fost stabiliți.

1.3.6. Folosirea unei Continuous Integration app pentru a opri build-urile în cazul în care s-au introdus credențiale valide

Un alt caz comun în marile organizații este folosirea utilităților de tipul Continuous Integration care să ruleze pentru fiecare commit, sau pentru fiecare build. Acestea sunt în general folosite pentru a folosi aplicații de tip secret scanner, static code analyzers, sau chiar compilarea aplicațiilor. Acest proiect de licență conține o aplicație care se va utiliza în interiorul acestor pipeline-uri cu scopul de a rula o scanare completă pe un

proiect. Acest tool, anunță backend-ul prin intermediul unui endpoint REST, și este echivalentul unui request de scanare prin intermediul interfeței frontend. Această scanare presupune verificarea noilor commit-uri și/sau a imaginilor Docker și rescanarea tuturor bazelor de date ale proiectului. Utilitarul acesta va bloca până când scanarea va fi finalizată, chiar dacă ar fi eronată.

Dacă în timpul scanării se produce o alertă de gradul severitate Înaltă, atunci utilitatrul va returna codul de eroare 1, pentru a semnaliza tool-urilor de tip CI că s-a întâmplat un lucru grav, acestea oprind cu totul build-ul.

1.3.7. Notificarea Administratorilor în cazul în care au fost introduse credențiale valide sau există probleme de configurație

Dacă aplicația noastră găsește niște erori de configurație (prezentate mai sus) sau niște credențiale valide, aceasta va genera erori de severitate Înaltă. Dacă la finalul unei scanări există una sau mai multe alerte de această severitate, atunci se vor anunța pe email toți Administratorii sau Ownerul organizației în care se află proiectul, pentru a putea rezolva eroarea într-un timp cât mai util, astfel încât să nu ofere actorilor malițioși timp să atace baza de date.

1.3.8. Procesarea regulată a scanărilor și a actualizărilor bazei de date de vulnerabilități

Pentru a menține securitatea proiectelor, chiar și în momentul în care nu se rulează utilitarul de tip Continuous Integration, aplicația propune o modalitate de rulare automată, zilnică, a scanărilor pe toate proiectele introduse în aplicație. Această automatizare presupune existența unui Automatic Scheduler, care se va ocupa de preularea zilnică a noutăților publicate pe API-ul de la National Vulnerability Database, pentru a verifica dacă au existat noi vulnerabilități pentru versiunile deja existente, dar și de preluarea listei de proiecte înregistrate în aplicație și rularea automată a scanărilor pentru acestea. Această aplicație se ocupă doar de trimiterea spre execuție a acestor task-uri, acestea fiind urmate să fie procesate de workerii interni, sau de cei externi, după caz.

1.4. Soluții similare

Există mai multe aplicații similare cu soluția propusă în cadrul acestui proiect de licență, însă majoritatea au doar integrări cu bazele de date de tip “cloud”, fiind evitate și bazele de date care sunt rulate de organizații în mașini virtuale (VM-uri), sau care sunt rulate chiar pe mașini dedicate, în interiorul organizației (fizic). Din acest motiv,

majoritatea, deși scanează un număr mai mare de credențiale, nu pot verifica dacă acestea sunt chiar valide pentru bazele de date.
În plus, majoritatea tool-urilor nu scanează și erorile de configurație ale bazei de date, în general fiind presupus că administratorii de sistem sunt cei care actualizează aceste sisteme, și cei care le mențin siguranța configurațiilor.

1.4.1. GitHub Secret Scanner

GitHub Secret Scanner este soluția integrată în site-ul GitHub care scanează credențialele de la serviciile populare și le verifică valabilitatea prin intermediul API-urilor cu care acesta interacționează[5]. Această aplicație poate să revoce credențialele în mod automat, pentru secretele suportate.

Soluția este activată de la început pentru toate repertoriile Git care există pe GitHub, și scanează mai multe lucruri precum:

- Descrierile și comentariile din “Issues”
- Descrierile, comentariile și titlurile din “Pull Requests”
- Descrierile, comentariile și titlurile din “Discussions”
- Conținutul commit-urilor din repertorii

Soluția are un număr mare de parteneri, care ajută aplicația să scaneze diferite tipuri de secrete. Fiecare partener își întreține unul sau mai multe regex-uri care sunt folosite pentru identificarea secretelor, acestea fiind mai apoi transmise înapoi partenerului prin intermediul unui API endpoint pentru a verifica dacă acestea chiar sunt valide. Dacă credențialele sunt valide, acestea sunt revocate în mod automat de parteneri. Această abordare identifică cel mai corect secretele valide, însă nu poate detecta credențialele bazelor de date decât în momentul în care baza de date este rulată folosind servicii de tipul Google Cloud SQL sau Amazon RDS.

În comparație cu aplicația noastră, această abordare nu ne permite scanarea bazelor de date rulate în afara Cloud, sau rulate în Cloud dar prin intermediul mașinilor virtuale configurate manual. În plus, aceasta nu reverifică existența credențialelor deja verificate, fiind posibil ca administratorii să schimbe parolele utilizatorilor bazelor de date în anumite parole deja existente în cadrul codului, fără a primi alerte. O altă deficiență a acestei abordări este lipsa verificării de complexitate a parolelor bazelor de date, întrucât aceasta nu oferă un mod de forță brută prin care se elimină posibilitatea utilizării parolelor slabe.

1.4.2. AWS CodeGuru Security

La fel ca și GitHub Secret Scanner, această aplicație poate să scaneze repertoriile Git, însă este limitată la a scana doar dacă acestea există în interiorul

aplicației “AWS Secrets Manager”[22]. Aceasta nu oferă posibilitatea de a revoca secrete, sau de a verifica dacă acestea sunt valide prin altă modalitate.

Aplicația poate scana doar un număr limitat de fișiere, precum `.go`, `.json`, `.py`, etc.. Aceasta nu este capabilă să scaneze în interiorul tuturor fișierelor care se pot găsi în interiorul unui repertori Git, precum aplicații compilate care conțin în interiorul acestora credențialele bazelor de date. În plus, aceasta poate găsi un număr limitat de secrete, și anume doar cele care se află în interiorul AWS Secrets Manager, sau alte câteva chei precum GitHub access token.

În comparație cu aplicația propusă, aceasta nu se poate conecta la bazele de date să verifice dacă credențialele sunt valide, și nu poate verifica dacă bazele de date scanate folosesc credențiale slabe sau au vulnerabilități cunoscute.

Capitolul 2. Arhitectura sistemului

Pentru a realiza o aplicație scalabilă și care să poată fi modificată rapid împotriva noilor vulnerabilități apărute, propunem o arhitectură bazată pe microservicii, care să se ocupe de lucruri precum actualizarea bazei de date de vulnerabilități, să se ocupe de scanarea propriu-zisă sau chiar să se ocupe de interfața publică.

Aplicația este împărțită în diferite componente, scrise în mai multe limbaje de programare:

- Partea de frontend este scrisă folosind Svelte, care se ocupă de oferirea unei interfețe intuitive, care poate fi folosită de pe orice dispozitiv, pentru toți utilizatorii aplicației. Aceasta oferă și posibilitatea de vizualizare a scanărilor și a rezultatelor acestor scanări, dar și verificarea, modificarea și crearea de noi conexiuni la baze de date sau locații din care să se extragă secrete;
- Partea de backend, scrisă în Golang care oferă un API prin paradigma REST pentru utilizatori, dar și care este folosit de către aplicația Svelte;
- Workerii, care rulează scanările publice, dar și scanările proiectelor care nu dețin worker propriu.
- Automatic Scheduler, care alocă la intervale regulate (o dată pe zi) o actualizare a bazei de date de vulnerabilități, dar și câte o scanare completă pentru fiecare proiect.

Pe lângă aceste componente, există alte două componente publice, pe care administratorii le pot rula separat de backend, și care oferă funcționalități pe care serverul principal nu le poate oferi singur:

- Workerii per proiect, care sunt asemănători workeri-lor generali, cu mica excepție că aceștia pot fi rulați în interiorul zonelor securizate de firewall-uri, aceștia rulând scanările în momentele când baza de date este protejată de internetul public;
- Partea de Continuous Integration, o aplicație scrisă tot în Golang, care anunță backend-ul de la distanță că se dorește o nouă scanare completă, și care va aștepta până când scanarea este finalizată, și va da anunța prin intermediul de return codes de rezultatul scanării.

2.1. Arhitectura generală

Aplicația constă în serviciile descrise mai sus, dar și în mai multe servicii care le leagă între ele. Aceste servicii, precum și legăturile principale între acestea au fost descrise în diagrama 2.1. Serviciile, precum și rolul acestora sunt:

- Nginx, un server HTTP este folosit de noi pentru a ruta în mod corect între endpoint-urile de tip /api/, care se duc către aplicația scrisă în Golang (API

Server), iar restul de endpoint-uri se vor duce către serverul Node, care rulează codul aplicației Svelte;

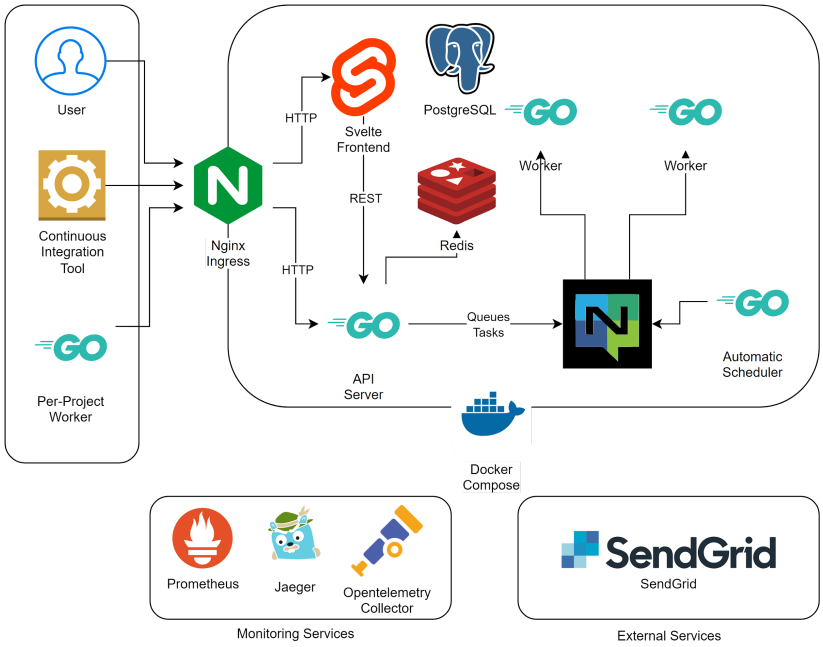

Figura 2.1. Arhitectura generală a sistemului

- NATS Server, un broker de task-uri scalabil[27], care nu consumă foarte multe resurse, este folosit pentru alocarea task-urilor de scanare/trimitere mailuri/actualizarea bazei de date de vulnerabilități, și este în mod principal alocat de API Server (în cazul în care se cere de la frontend/de la un CI tool), dar poate fi alocat și de către Automatic Scheduler. Toți workerii interni se conectează la acest broker de task-uri și procesează task-urile alocate de acesta;
- PostgreSQL, baza de date în care aplicația prezentată își stochează toate informațiile precum datele de conectare bazelor de date scanate, cheile workerilor și metodele de conectare ale utilizatorilor
- Redis, o bază de date bazată pe chei-valori, este folosită pentru caching, fiind în principal folosită la grăbirea verificării repetate a credențialelor transmise prin HTTP a userilor și workerilor externi, dar și a permisiilor acestora asupra proiectelor;
- SendGrid, un serviciu extern este folosit pentru trimiterea de mail-uri precum cele de resetare a parolei, sau de alertă în cazul în care există scanări cu severități Înalte;
- Opentelemetry Collector este folosit ca metodă de agregare a metricilor de la toți workerii interni și API servere, acesta trimițând datele la Jaeger[31] și Prometheus[32] pentru stocare[28]. Backend-ul, dar și workerii interni ai aplicației transmit date la acest serviciu pentru monitorizare;

- Jaeger este un serviciu care agregă activitățile aplicațiilor și oferă o interfață prin care se poate vedea punctul de plecare și toți pașii care s-au executat pentru acel punct de plecare. Spre exemplu, se poate observa request-ul de la care a pornit o scanare completă a unui proiect, și de la acel “trace” se pot observa toți pașii pe care i-a făcut întreg sistemul (prin ce worker a trecut, ce a executat acesta, SQL queries care s-au efectuat, etc.). Această metodă de urmărire a etapelor se cheamă “distributed tracing” și funcționează chiar și între mai multe servicii;
- Prometheus este o bază de date bazată pe timp, care stochează și oferă informații despre metricile aplicațiilor, precum numărul de mailuri trimise, utilizarea CPU a fiecărei aplicații individuale, timpul de răspuns al request-urilor HTTP, etc.

2.2. *Backend*-ul

Aplicația de backend este împărțită în mai multe module, toate comunicând între ele prin intermediul interfețelor pentru a ușura schimbarea implementărilor, dar și pentru a ușura testarea tuturor funcționalităților. Modulele sunt conectate între ele folosind dependency injection.

Cele mai importante module sunt:

- `api`, care oferă o interfață care utilizează paradigma REST API, pentru a fi accesibilă prin internet. Acest modul este compus din trei submodule, `auth` care se ocupă de autentificarea utilizatorilor, dar și de funcționalitățile precum recuperarea parolei, conectarea folosind chei de securitate, etc.; `authorization`, care se ocupă de permisiile utilizatorilor și API-ul propriu-zis. Acest API este generat de la o specificație OpenAPI, pentru a nu putea fi diferențe între implementarea aplicației și specificația acesteia;
- `bruteforce`, care se ocupă de partea de verificare a tuturor parolelor pentru un proiect, acesta fiind responsabil de utilizarea parolelor de la repertoriile Git, de la imaginile Docker, dar și de la listele de parole generice și simple. Acesta se mai ocupă și de notificarea la fiecare secundă de statusul scanării de parole;
- `extractors`, care definește modalitățile prin care trebuie scanate repertoriile Git, respectiv imaginile Docker. Acest modul se ocupă de paralelizarea acestor scanări, dar și de notificarea secretelor găsite. Acest modul definește și formatul secretelor găsite, prin submodulul `file`, care definește cum se scanează un fișier individual, acesta fiind folosit de cele două scanări menționate anterior;
- `nvd`, care se ocupă de actualizarea și verificarea vulnerabilităților versiunilor bazelor de date

- `scanner`, care definește modalitățile în care ar trebui scanate bazele de date, toate acestea satisfăcând o interfață comună, care poate fi extinsă foarte ușor;
- `saver`, care face conexiunea între scannere și obiectele din baza de date, acest modul fiind responsabil și de creearea obiectelor și actualizarea acestora în baza de date, bazat pe rezultatele oferite de scannerele individuale
- `tasks`, care oferă partea de workeri multipli interni, acesta fiind în mod principal o interfață prin care se pot rula task-uri de tip scanare, update al bazei de date de vulnerabilități, etc. fie în interiorul procesului care dorește task-ul, fie în interiorul altui worker, prin intermediul brokerului de task-uri NATS;
- `db`, care oferă funcții Golang generate de la fișiere SQL, care ajută la interacționarea eficientă și sigură cu baza de date PostgreSQL pentru a stoca date;
- `messages`, care oferă modulului de API modalitatea să urmărească cererile pentru scanare care sunt destinate workeri-lor externi. Aceste mesaje sunt rutate folosind topicele NATS.

Toate modulele au fost scrise folosind principiile Segregării interfețelor, pentru a ușura refolosirea acestora, dar și pentru a ajuta la testarea unitară a fiecărui modul individual. Fiecare modul exportă unul sau mai multe implementări, dar și interfețele asociate acelei implementări. Datorită implementării automate a interfețelor bazată doar pe nume din limbajul de programare Go, fiecare modul își prezintă interfața necesară funcționării acestuia, iar toate implementările care coincid acelei specificații pot fi folosite în mod automat.

În plus, pe parcursul întregii aplicații, dar și între servere, se transmit datele de context pentru a putea urmări traficul intern cauzat de fiecare cerere procesată de server. Pentru a se transmite contextul între module se folosește interfața `context.Context` din librăria standard Go, iar pentru transmiterea contextului între servere diferite s-a implementat un protocol bazat pe Protobuf care serializează contextul înaintea mesajului efectiv.

Protobuf este o metodă de serializare binară concepută de Google, care este agnostică de limbaj, și care pornește de la o definiție și generează în mod automat cod pentru fiecare limbaj de programare necesar. În cazul nostru, s-a folosit aplicația `protoc` pentru a genera cod Go care să proceseze și genereze mesaje Protobuf.

Protocolul conceput în cadrul acestei aplicații este conceput prin 3 părți individuale:

- Un număr formatat binar (4 octeți), care reprezintă lungimea antetului

- Un antet, de lungimea specificată anterior, care conține toate datele legate de Opentelemetry din contextul care a transmis mesajul. Aceste date au fost generate folosind `propagation.MapCarrier{}` din librăria `go.opentelemetry.io/otel/propagation`. Acest antet a fost serializat folosind Protobuf
- Mesajul efectiv, care a fost serializat tot folosind Protobuf

Aceste mesaje au fost transmise prin intermediul Serverului NATS de la producători la workerii interni, pentru a-i notifica pe aceșia de necesitatea pornirii scanărilor sau a altor acțiuni.

Modulul de `api` expune o singură funcție, care returnează un server bazat pe librăria `Gin` care expune toate funcționalitățile necesare aplicației prin intermediul paradigmei REST. Modulul necesită introducerea prin Dependency Injection a bazei de date folosite, a modulului folosit pentru autentificare, dar și a implementării de schimb de mesaje (“exchange”), respectiv modulul care implementează rularea task-urilor. În cadrul acestui modul se definesc și modalitatea prin care sunt expuse ID-urile traseelor Opentelemetry (“Opentelemetry traces”) clienților (prin intermediul antetelor HTTP), modalitatea de caching folosită, aspectele de securitate precum CORS Protection, dar și aspecte de afișarea informațiilor fiecărei cereri. Partea de procesare efectivă a cererilor se face prin implementarea unei implementări generate automat de la schema OpenAPI folosind `oapi-generate`, codul generat ocupându-se de validarea și parsarea cererilor. Specificația OpenAPI va fi anexată acestei lucrări.

Submodulul de autentificare, `auth` se ocupă de autorizarea fiecărui client, dar și de întreg procesul de înregistrare, autentificare, schimbare parolă, sau chiar de înregistrarea sau autentificarea folosind cod TOTP sau prin cheie Passkey. Acest modul folosește la bază librăria `authboss`, și implementează toate interfețele cerute de aceasta pentru funcționarea corectă a aplicației. În plus, acest modul se ocupă și de oferirea datelor despre utilizatorul care a făcut cererea modulului `api`.

Submodulul de autorizare, `authorization`, se ocupă de verificarea permisiilor utilizatorilor, fiind de obicei apelat după modulul de autentificare. Acesta verifică dacă utilizatorul care a făcut cererea are acces la proiectul/organizația cerută, dar și dacă are acces să facă operația cerută de acesta.

Modulul `brureforce` expune o modalitate prin care, dacă se oferă o implementare de tip `scanner.Scanner`, poate să extragă toți utilizatorii și să execute pentru fiecare din aceștia o verificare completă a parolelor generale, sau care se află în

proiectul din care provine scanarea. Acest modul este implementat folosind primitivele de concurență din librăria standard Go, pentru a facilita procesarea pe mai multe nuclee pentru fiecare utilizator. În plus, acest modul anunță prin intermediul apelării unei funcții `callback` despre statusul procesării cerute. Acest modul oferă și posibilitatea săririi peste calculele inutile, prin stocarea id-ului ultimei parole comune încercate, pentru a putea sări peste hash-urile care le știe deja că au fost încercare și nu au dat rezultatul căutat.

Modulul `cmd` oferă o interfață text pentru utilizarea aplicației de la liniile de comandă. Această interfață este implementată folosind librăria `github.com/spf13/cobra`. Credențialele și configurațiile sunt preluate de acest modul din linia de comandă sau direct din variabilele de sistem. Toate argumentele care pot fi folosite la linia de comandă pot fi luate direct de la variabilele de sistem, prin următoarea regulă: se folosește denumirea variabilei, folosind numai litere mari, la care se înlocuiește `-` cu `_`. În plus, acest modul oferă și posibilitatea efectuării migrării bazei de date folosind subcomanda `migrate`. Toate comenzile și subcomenzile au opțiunea de `--help`, care va afișa un mic tutorial despre cum trebuie utilizată comanda, dar și lucrurile necesare acesteia.

Modulul `tasks` include o interfață comună tuturor implementărilor care rulează task-uri, însă acesta include și două implementări a aceleiași interfețe: unul local și unul folosind serverul NATS. Cel local rulează task-urile folosind goroutines, pornindu-se astfel task-ul pe alt thread față de cel care s-a cerut efectuarea task-ului, iar cel care folosește serverul NATS trimite o notificare către serverul NATS că task-ul se dorește a fi efectuat. Modulul de NATS include și o implementare a unui worker NATS, care execută toate task-urile primite prin intermediul broker-ului central. Modulul de NATS folosește implementarea locală pentru executarea efectivă a task-urilor.

Pe parcursul acestei licențe am dezvoltat și o librărie pentru librăria de autentificare `authboss`, care este una dintre cele mai populare librării de autentificare pentru Go, pentru a lăsa utilizatorii să se conecteze folosind protocolul WebAuthn, care este redenumit și ca Passkey. Protocolul WebAuthn este un protocol care are la bază semnarea folosind chei publice, și constă în înlocuirea combinației username-parolă, în un singur pas, care este verificarea prezenței utilizatorului la calculator. Aplicațiile sau device-urile fizice care implementează protocolul sunt responsabile de menținerea credențialelor utilizatorilor. Cele mai populare implementări ale acestui protocol sunt cheile Yubikey[29], care oferă un dispozitiv fizic care atestă utilizatorul pe orice calculator, Windows Hello care atestă utilizatorul folosind metodele de conectare Windows (amprente, recunoaștere facială, etc.), sau telefoanele Android, care oferă

dispozitivelor Bluetooth din jur posibilitatea să ateste prezența utilizatorului prin protocolul CTAP2[26].

Fiecare bază de date are configurații diferite și alerte diferite, însă majoritatea au alerta “Database is accessible from the internet”. Aceasta alertă apare dacă un proiect are worker-ul separat activat și baza de date este accesibilă de la aplicația noastră principală. Această parte este prezentată în figura 2.3.

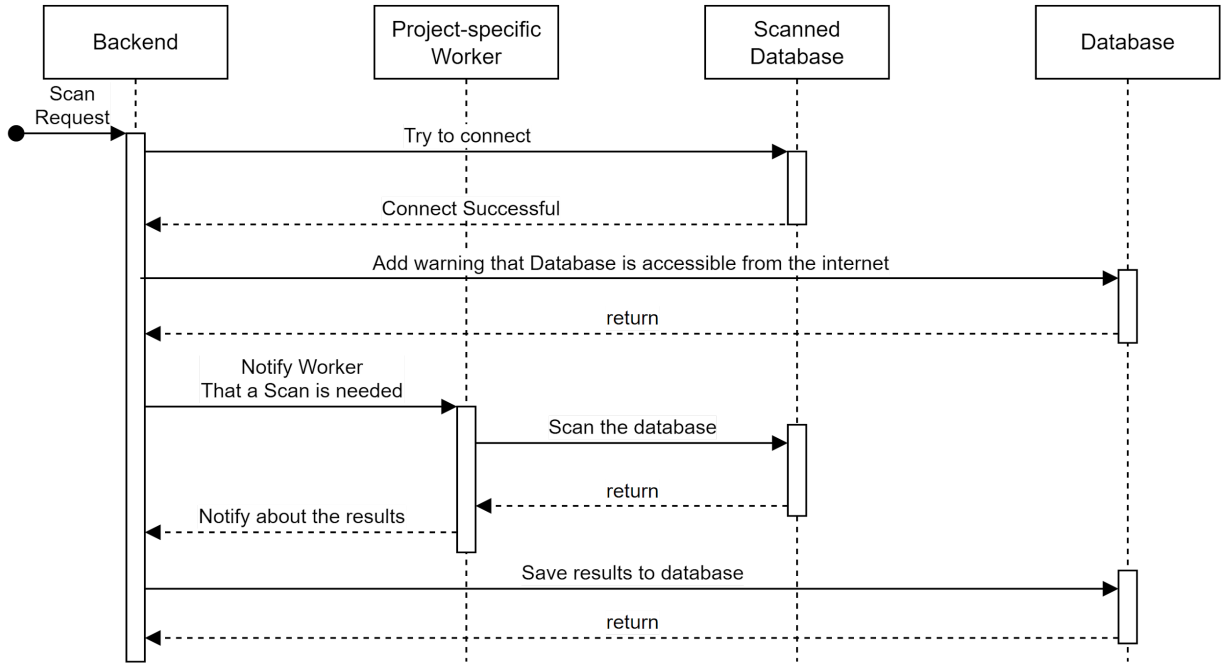

Figura 2.3. Utilizarea workerilor externi

Dacă proiectul nu are activată scanarea folosind workeri per-organizație, atunci această notificare va avea severitatea de Avertizare, pentru a nu influența pipeline-urile CI, care se opresc (returnează status diferit de 0) doar în momentul în care există o alertă cu severitate Înaltă.

2.2.1 Worker-ii externi, pentru o singură organizație

Workerii externi, care fiecare deservește unei singure organizații, folosesc modulele de `tasks`, `saver` și `scanner` pentru a rula scanările în mod asemănător workerilor interni ai proiectului, însă aceștia oferă o implementare diferită asupra modalității de stocare în baza de date, această implementare fiind de fapt un wrapper care apelează REST API-ul serverului principal, pentru a stoca datele propriu-zis. Modulul de `worker`, care implementează această funcționalitate, mai oferă și o

modalitate pentru a urmări cererile de la un API server principal, folosind HTTP Long Polling[15], pentru a ști când să se facă o scanare, dar și ce scanare să se efectueze.
Modulul de API folosește aici funcționalitatea oferită de modulul de messages, care ascultă mesajele publicate pe anumite topic-uri NATS specifice per proiect, și care are două modalități de a acționa:

- Dacă există un mesaj pe acel topic, atunci returnează mesajul direct, anunțând worker-ul extern că are un task de făcut;
- Dacă nu există nici un task, atunci așteaptă 10 secunde, și dacă nu, returnează un HTTP 204, întrucât nu a găsit nici un task de efectuat;

Ambele cazuri au fost prezentate în figura 2.4., unde prima cerere nu a returnat nici un task nou, însă a doua cerere a returnat un task nou de la serverul NATS.

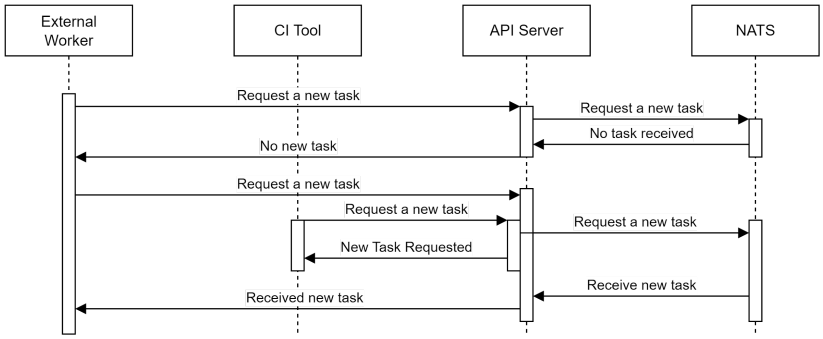

Figura 2.4. Utilizarea serverului NATS și a aplicației CI

S-a folosit HTTP Long poll deoarece HTTP este protocolul care poate trece cel mai ușor prin majoritatea firewall-urilor, și deoarece funcționalitatea simplă a acestuia este destul pentru funcționalitatea necesară acestei aplicații.

2.3. *Frontend*-ul

Frontend-ul este o aplicație realizată în Svelte, un framework JS care se axează pe performanță și simplicitate. Acesta a fost creat în 2016, și este privit ca unul dintre cele mai bune și simple framework-uri la momentul actual.

Aplicația Svelte folosește Server-Side Rendering pentru a genera paginile, aceasta consumând REST API-ul expus de backend. Aplicația frontend folosește și Client Hydrading pentru a genera o pagină interactivă folosind JavaScript, dacă utilizatorul are această funcționalitate activată în browser. Singura funcționalitate care depinde de JavaScript este conectarea folosind chei Webauthn, sau înregistrarea acestora. În rest, aplicația este complet funcțională folosind doar HTTP POST requests.

Aceste request-uri vor fi convertite de server-ul de frontend în JSON, care va fi trimis la backend pentru procesarea cererii.

La fiecare reîncărcare a paginii se vor descărca de la serverul principal unul sau mai multe endpoint-uri, care vor popula informațiile. Endpoint-urile vor fi așteptate folosind `Promise.all()`, pentru a facilita descărcarea în paralel a acestora. În plus, deoarece serverul Go este multi-threaded, acesta va procesa aceste cereri în mod paralel, grăbind astfel întreaga execuție.

Pentru a facilita o interfață ușor accesibilă, dar și intuitivă s-a folosit framework-ul CSS “DaisyUI”, care oferă niște clase Tailwind predefinite care sunt ușor de utilizat. Pentru restul necesar de CSS, s-a folosit Tailwind, care este o librărie de clase CSS predefinite care ajută la eliminarea fișierelor CSS externe, întreg-ul stil necesar unei componente fiind definit chiar în atributul `class`.

Aplicația frontend este împărțită în mai multe module, câte unul per pagină, fiecare modul având mai multe fișiere standard: `+page.svelte`, care conține codul HTML/CSS specific acestei pagini, `+page.server.ts`, care conține codul ce va fi acționat în momentul în care un utilizator trimite o cerere POST, dar și codul care va descărca datele utilizatorului de pe backend. Există și fișierele `+layout.svelte`, respectiv `+layout.server.ts`, care reprezintă codul care va fi executat de această pagină, dar și de către toate subpaginile directorului în care acestea se află.

Interfața dintre backend și frontend s-a realizat prin intermediul librăriei `openapi-fetch`, care a generat un client TypeScript care este compatibil cu specificația OpenAPI a backend-ului. Această abordare a asigurat corectitudinea implementării clientului, dar și verificarea automată a parametrilor primiți/trimiși de frontend.

Aplicația frontend folosește TypeScript pentru a genera informații despre tipul variabilelor JavaScript, astfel încât să ne putem asigura că aplicația nu va avea erori de sintaxă sau de tip în momentul folosirii.

În plus, pentru colectarea anonimă a datelor de utilizare a aplicației, s-a folosit Plausible, o alternativă focusată pe anonimitate la Google Analytics[30], care folosește un server hostat separat de instanța principală, și anume `plausible.tedyst.ro`.

Pentru stocarea informațiilor legate de tema curentă selectată de utilizator, se stochează în `localStorage` o cheie care are două valori: `light` sau `dark`, acestea reprezentând culoarea dorită de utilizator.

Pentru implementarea funcționalității de CORS la nivel de backend-frontend, înainte de fiecare cerere de tip POST care va fi transmisă la backend se va face o altă cerere, de tip OPTIONS, care oferă aplicației frontend un token CORS valid o singură dată, pentru utilizatorul curent conectat. Acest token va fi folosit în următoarea cerere, și va garanta că utilizatorul a dorit această cerere, acesta fiind transmis prin intermediul antetului `X-CSRF-Token`.

În plus, pentru implementarea funcționalității de CORS la nivel de client-frontend, se va folosi funcționalitatea nativă SvelteKit pentru a genera în mod automat token-urile necesare fiecărei cereri.

2.4. Stocarea datelor

Datele sunt stocate într-o bază de date PostgreSQL, iar datele sensibile (precum cheile de conectare pentru accesarea repertoriilor Git, imaginilor Docker sau parolele necesare conectarii la bazele de date sunt criptate la nivel de Postgres folosind funcțiile `pgp_sym_decrypt` și `pgp_sym_encrypt`. Criptarea se face folosind standardul OpenPGP[7, 8], iar cheia de criptare simetrica este o concatenare între o cheie generata aleatoriu per organizație, și o cheie separată, setată la nivel de server, care este configurată de o variabilă de de mediu înconjurător. Schema bazei de date este prezentată în figura 2.5. Cele mai importante tabele, precum și rolul acestora sunt:

- Tabelele `users`, `reset_password_tokens`, `totp_secret_tokens`, `remember_me_tokens` și `webauthn_credentials` se ocupă de autentificarea utilizatorilor, identificarea acestora după email, sau id, de autentificarea folosind coduri TOTP sau chei Webauthn, dar și de stocarea cheilor temporare de resetare a parolelor sau a credențialelor de tip “remember me”.
- Tabelele `git_repositores`, `git_commits`, `git_results`, respectiv `git_scans` se ocupă de stocarea credențialelor criptate ale repertoriilor Git scanate, de stocarea informațiilor despre fiecare commit, respectiv fiecare credențial găsit al fiecărui commit, dar și de legarea între rezultate și scanări.
- Tabelele `docker_images`, `docker_layers`, `docker_results`, și `docker_scans` stochează toate credențialele criptate ale imaginilor Docker scanate, toate informațiile despre fiecare strat Docker în parte, credențialele găsite și legăturile dintre fiecare rezultat și scanarea inițială.

- Tabelele `nvd_cves`, `nvd_cpes` și `nvd_cve_cpes` stochează informații despre toate vulnerabilitățile preluate de la NVD, versiunile bazelor de date și legăturile dintre acestea.
- Tabelele `mongo_databases`, `mysql_databases`, `postgres_databases`, `redis_databases`, dar și `mongo_scans`, `mysql_scans`, `postgres_scans`, `redis_scans` stochează credențialele criptate ale serverelor de baze de date scanate, dar și legătura dintre acestea și scanarea inițială.
- Tabela `default_bruteforce_passwords` menține o listă cu parolele de bază verificate, aceasta fiind populată din liste de parole comune precum Rockyou.
- Tabelele `scans`, `scan_groups`, `scan_bruteforce_results` și `scan_results` fac legătura între grupurile de scanări, scanările efective, rezultatele scanării (care conțin și severitatea fiecărui rezultat), dar și statusul fiecărei scanări referitor la partea de forță brută.
- Tabela `bruteforced_passwords` menține o listă cu hash-urile parolelor care au incercat să fie găsite, dar și id-ul ultimei parole încercate din tabela `default_bruteforce_passwords`.
- Tabelele `workers`, `projects`, `organizations` și `organization_members` se ocupă de organizarea în organizații, proiecte, dar și de permisiile fiecărui utilizator sau worker extern.

2.5. Aspecte de securitate

Pentru prevenirea atacurilor de tip SQL Injection, se folosește librăria `sqlc`, care compilează linii SQL direct în cod Golang care este sigur de utilizat, și care nu permite prelucrarea directă a bazei de date prin comenzi SQL care ar putea fi injectate[24].

Pentru prevenirea atacurilor de tip Buffer Overflow/Return Oriented Programming, se folosește Golang, un limbaj sigur, care nu permite ieșirea din vectori. Limbajul este și garbage collected, așa că atacurile de tip use-after-free nu pot fi făcute.
Pentru prevenirea atacurilor bazei de date prezentate mai sus, se rulează baza de date într-un Docker container, într-o rețea privată separată de internet. Singurele aplicații care au acces la acea rețea sunt workerii și serverul care oferă API-ul. Datele din baza de date sunt criptate folosind OpenPGP cu o cheie simetrică. Cheia simetrică este generată per organizație și nu poate fi extrasă doar cu ajutorul bazei de date, fiind necesară și o altă cheie “SALT” oferită de variabilele de sistem. Aceste două chei sunt concatenate, și împreună generează cheia privată de criptare specifică unei organizații. Această abordare oferă siguranța că, în cazul în caze baza de date ar fi compromisă, dacă atacatorul nu ar compromite și aplicația principală (spre a obține cheia SALT din aplicație).

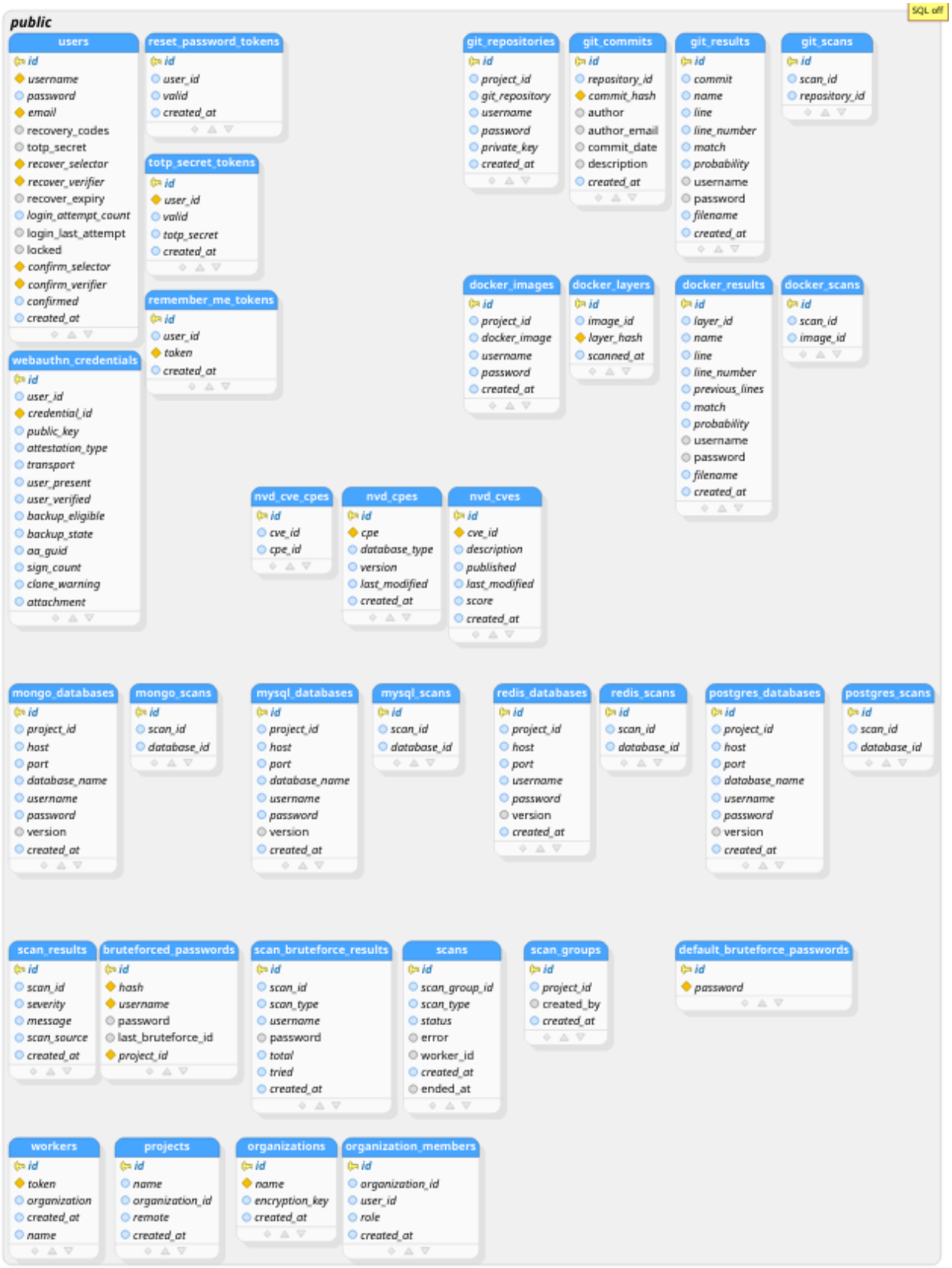

Figura 2.5. Arhitectura bazei de date

Un alt aspect legat de securitate este utilizarea comunicării criptate între Postgres și aplicațiile worker/API, prin folosirea unui certificat de autoritate semnat, care este incorporat în aplicații și marcat ca fiind de încredere. În acest mod se pot evita atacurile în cazul în care atacatorul reușește să obțină acces la traficul dintre cele două aplicații, traficul fiind complet criptat, nefiind posibilă interceptarea și modificarea acestuia prin folosirea unui MiTM (Man in the Middle) care să schimbe/scoată cu totul certificatul serverului Postgres.

Criptarea la nivel de bază de date se face folosind librării de încredere, și anume `pgcrypto`, care a fost verificată să implementeze algoritmii corect, fără vulnerabilități cunoscute[7].

La nivel de utilizatori ai aplicației, securitatea provine din necesitatea utilizării unor parole sigure (minim de 8 caractere, 1 număr, 1 simbol special), din posibilitatea utilizării a autentificării prin doi factori, dar și prin posibilitatea utilizării standardului WebAuthn[13], care oferă o variantă sigură de a te conecta folosind certificate publice oferite de dispozitive precum cheile Yubikey, telefoanele Android, dar și de alte aplicații software precum Windows Hello sau Bitwarden. Implementarea tehnologiei Webauthn a fost realizată folosind librăria `go-webauthn`[12], iar conectarea acestei librării de librăria `authboss` (care este folosită în această aplicație pentru autentificare) a fost realizată în cadrul realizării acestei licențe.

Capitolul 3. Scenarii de utilizare

3.1. Menținerea securității a bazelor de date care folosesc toate recomandările făcute de aplicație

Pentru cazul în care bazele de date au fost scanate de această aplicație, și nu au fost raportate erori, soluția propusă în această lucrare de licență va urmări ca în timp, dacă se vor efectua modificări de configurație la bazele de date scanate, atunci și acelea se vor fi monitorizate pentru a menține standardul de securitate. Spre exemplu, dacă în urma unei operații de migrare a bazei de date PostgreSQL se schimbă configurația `ssl` prin dezactivarea acesteia, aplicațiile care folosesc acea bază de date nu vor suferi modificări vizibile, iar administratorii ar putea să nu observe că această schimbare s-a propus. Aplicația propusă de noi ar observa această schimbare de configurație și ar alerta administratorii în termen de maxim o zi, sau chiar mai rapid, când se va efectua următoarea scanare manuală/de tip CI.

3.2. Schimbarea sau utilizarea unei parole slabe pentru unul sau mai mulți utilizatori ai bazelor de date

Pentru cazul în care un administrator al bazei de date setează parola unui utilizator să fie una destul de simplă, este o posibilitate mare ca aceasta să se afle în listele de parole generice importate de aplicația propusă în această lucrare de licență. În cazul în care aceasta chiar se află, atunci se va crea o alertă de severitate Înaltă la următoarea scanare (în maxim 24h), pentru a alerta că acel cont poate fi foarte ușor de compromis.

3.3. Introducerea unei perechi de username/password valide pentru o baza de date în repertoriile Git/imaginile Docker

Pentru cazul în care se introduce o parolă/o pereche de credențiale valide în interiorul repertoriilor Git, spre exemplu prin uitarea unui fișier de configurație, sau salvarea unui fișier legat de infrastructură, sau prin imaginile Docker, prin uitarea fișierelor de tipul `.dockerignore`, atunci se va observa în foarte puțin timp că acele credențiale sunt valide, astfel fiind alertați administratorii prin crearea unei alerte cu severitate Înaltă.

3.4. Aflarea în domeniu public a existenței unei vulnerabilități specifice versiunii de baze de date folosită

Pentru cazul în care se descoperă o vulnerabilitate nouă pentru una dintre bazele de date scanate de aplicația noastră, în maxim 24h se vor încărca datele despre acea vulnerabilitate în baza de date Postgres, urmând ca mai apoi să se lanseze scanări pentru toate proiectele înregistrate. Dacă unul sau mai multe proiecte folosesc baze de date cu versiuni afectate, atunci se vor crea alerte cu severitate Înaltă, fiind astfel alertați administatorii în vederea actualizării bazelor de date, pentru a repara vulnerabilitatea.

Concluzii și îmbunătățiri viitoare

În cadrul acestei lucrări am investigat aplicațiile care mențin securitatea bazelor de date, și am propus o soluție care poate monitoriza și alerta administratorii dacă au anumite configurații nesigure, sau dacă au uitat credențialele de la bazele de date într-un loc public. Obiectivele inițiale ale lucrării au fost de a realiza o aplicație care poate scana repertoriile Git publice și care poate verifica bazele de date public accesibile bazat pe aceste credențiale. Aceste obiective au fost atinse, și chiar extinse întrucât aplicația finală include mult mai multe funcționalități decât cele propuse inițial.

Unele limitări ale aplicației curente sunt numărul mici de baze de date scanate, care deși acoperă un procent mare din numărul de utilizatori, tot ar putea să nu fie îndeajuns. Printre bazele de date cele mai populare care nu au fost incluse în acest proiect sunt Cassandra, Oracle SQL, Microsoft SQL, si Elasticsearch. O altă limitare a proiectului este lipsa posibilității de a exclude anumite secrete per proiect, de a mări senzitivitatea cu care sunt detectate secretele sau chiar de a oferi diferite metode de extragere înafară de regex și base64.

O viitoare implementare a proiectului ar putea lua în vedere aceste cazuri speciale, și ar putea oferi utilizatorilor utilizarea de module, care să poată fi alese și configurate de aceștia în mod independent, per proiect.

Rezultatele acestei lucări au un domeniu mare de aplicabilitate, și ar putea contribui semnificativ la reducerea cazurilor în care atacatorii obțin direct acces la bazele de date ale aplicațiilor, prin intermediul vulnerabilităților sau credențialelor ușor de găsit.

Bibliografie

1. “3306 - Pentesting Mysql | HackTricks | HackTricks.” *Hacktricks.xyz*, 2018. URL: book.hacktricks.xyz/network-services-pentesting/pentesting-mysql (ultima accesare: iunie 2024)
2. “6379 - Pentesting Redis | HackTricks | HackTricks.” *Hacktricks.xyz*, 2018, book.hacktricks.xyz/network-services-pentesting/6379-pentesting-redis. (ultima accesare: iunie 2024)
3. “5432,5433 - Pentesting Postgresql | HackTricks | HackTricks.” *Hacktricks.xyz*, 2018, book.hacktricks.xyz/network-services-pentesting/pentesting-postgresql. (ultima accesare: iunie 2024)
4. “27017,27018 - Pentesting MongoDB | HackTricks | HackTricks.” *Hacktricks.xyz*, 2018, book.hacktricks.xyz/network-services-pentesting/27017-27018-mongodb. (ultima accesare: iunie 2024)
5. “About Secret Scanning - GitHub Docs.” *GitHub Docs*, 2024, docs.github.com/en/code-security/secret-scanning/about-secret-scanning. (ultima accesare: iunie 2024)
6. Docs. “Docs.” *Redis.io*, 2024, redis.io/docs/latest/. (ultima accesare: iunie 2024)
7. “F.28. Pgcrypto — Cryptographic Functions.” *PostgreSQL Documentation*, 9 May 2024, www.postgresql.org/docs/current/pgcrypto.html. (ultima accesare: iunie 2024)
8. Finney, Hal, et al. “RFC 4880: OpenPGP Message Format.” *IETF Datatracker*, 2024, datatracker.ietf.org/doc/html/rfc4880. (ultima accesare: iunie 2024)

9. “Git - Reference.” *Git-Scm.com*, 2024, www.git-scm.com/docs. (ultima accesare: iunie 2024)
10. “Git-Secret.” *Git-Secret*, 2024, sobolevn.me/git-secret/. (ultima accesare: iunie 2024)
11. “Go-Git/Go-Git: A Highly Extensible Git Implementation in Pure Go.” *GitHub*, 31 Mar. 2024, github.com/go-git/go-git. (ultima accesare: iunie 2024)
12. “Go-Webauthn/Webauthn: Webauthn/FIDO2 Library in Golang.” *GitHub*, 13 Mar. 2024, github.com/go-webauthn/webauthn. (ultima accesare: iunie 2024)
13. “Guide to Web Authentication.” *Guide to Web Authentication*, 2019, webauthn.guide/#webauthn-api. (ultima accesare: iunie 2024)
14. “Layers.” *Docker Documentation*, 2024, docs.docker.com/build/guide/layers/. (ultima accesare: iunie 2024)
15. Mohammad Hoseini Rad. “Long-Polling with Golang! - Mohammad Hoseini Rad - Medium.” *Medium*, Medium, 23 Feb. 2023, medium.com/@mhrlife/long-polling-with-golang-158f73474cbc. (ultima accesare: iunie 2024)
16. “MySQL - Market Share, Competitor Insights in Relational Databases.” *6sense*, 2024, 6sense.com/tech/relational-databases/mysql-market-share. (ultima accesare: iunie 2024)
17. “MySQL :: MySQL 8.4 Reference Manual.” *Mysql.com*, 2024, dev.mysql.com/doc/refman/8.4/en/. (ultima accesare: iunie 2024)
18. “Mysql-Server/Mysys/Crypt_genhash_impl.cc at 824e2b4064053f7daf17d7f3f84b7a3ed92e5fb4 · Mysql/Mysql-Server.” *GitHub*,

2024,
github.com/mysql/mysql-server/blob/824e2b4064053f7daf17d7f3f84b7a3ed92e5fb4/mysys/crypt_genhash_impl.cc#L235. (ultima accesare: iunie 2024)

19. “NVD - Home.” *Nist.gov*, 2024, nvd.nist.gov/. (ultima accesare: iunie 2024)
20. “PostgreSQL - Market Share, Competitor Insights in Relational Databases.” *6sense*, 2024, 6sense.com/tech/relational-databases/postgresql-market-share. (ultima accesare: iunie 2024)
21. “PostgreSQL: Documentation.” *Postgresql.org*, 2024, www.postgresql.org/docs/. (ultima accesare: iunie 2024)
22. “Secrets Detection - Amazon CodeGuru Security.” *Amazon.com*, 2024, docs.aws.amazon.com/codeguru/latest/security-ug/secrets-detection.html. (ultima accesare: iunie 2024)
23. “SQL Injection | OWASP Foundation.” *Owasp.org*, 2024, owasp.org/www-community/attacks/SQL_Injection#. (ultima accesare: iunie 2024)
24. “Sqlc Documentation — Sqlc 1.26.0 Documentation.” *Sqlc.dev*, 2023, docs.sqlc.dev/en/latest/. (ultima accesare: iunie 2024)
25. “Volatiletech/Authboss: The Boss of Http Auth.” *GitHub*, 2024, github.com/volatiletech/authboss. (ultima accesare: iunie 2024)
26. “What Is CTAP?” *Yubico*, 13 Dec. 2023, www.yubico.com/resources/glossary/ctap/. (ultima accesare: iunie 2024)
27. “What Is NATS | NATS Docs.” *Nats.io*, 14 Dec. 2021, docs.nats.io/nats-concepts/what-is-nats. (ultima accesare: iunie 2024)

28. What is OpenTelemetry. “What Is OpenTelemetry?” *OpenTelemetry*, 3 June 2024, opentelemetry.io/docs/what-is-opentelemetry/. (ultima accesare: iunie 2024)
29. “Yubico Home.” *Yubico*, 23 May 2024, www.yubico.com/. (ultima accesare: iunie 2024)
30. “Plausible Analytics | Simple, privacy-friendly Google Analytics alternative” Plausible, 23 May 2024, plausible.io/. (ultima accesare: iunie 2024)
31. “Jaeger: Open Source, Distributed Tracing Platform.” *Jaegertracing.io*, 2023, www.jaegertracing.io/. (ultima accesare: iunie 2024)
32. Prometheus. “Prometheus - Monitoring System & Time Series Database.” *Prometheus.io*, 2014, prometheus.io/. (ultima accesare: iunie 2024)

Anexa 1 - Specificația OpenAPI

Prima anexă conține documentația completă a API-ului dezvoltat pentru această lucrare de licență. Documentația este generată folosind specificațiile OpenAPI și include toate endpoint-urile, metodele HTTP, parametrii de intrare și răspunsurile posibile. Datorită faptului că specificația este foarte mare (aproximativ 4000 de linii), aceasta nu a putut fi inclusă complet în această lucrare. Specificația întreagă poate fi găsită la următoarea adresă: https://drive.google.com/file/d/1xlQdvAiUVcg63NNPVuaEXJt9ZnG_covj/view?usp=sharing

În plus, vom include rezumatele rutelor descrise prin OpenAPI cu ajutorul mai multor screenshot-uri ale aplicației Swagger, care a interpretat specificația proiectului:

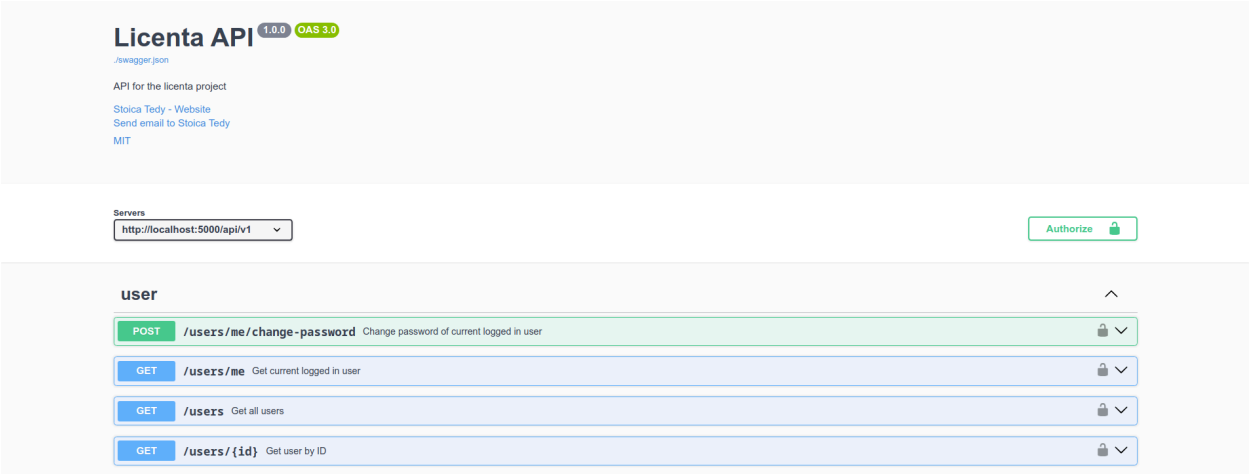

Figura 4.1. Descrierea specificației rutelor din categoria “user”

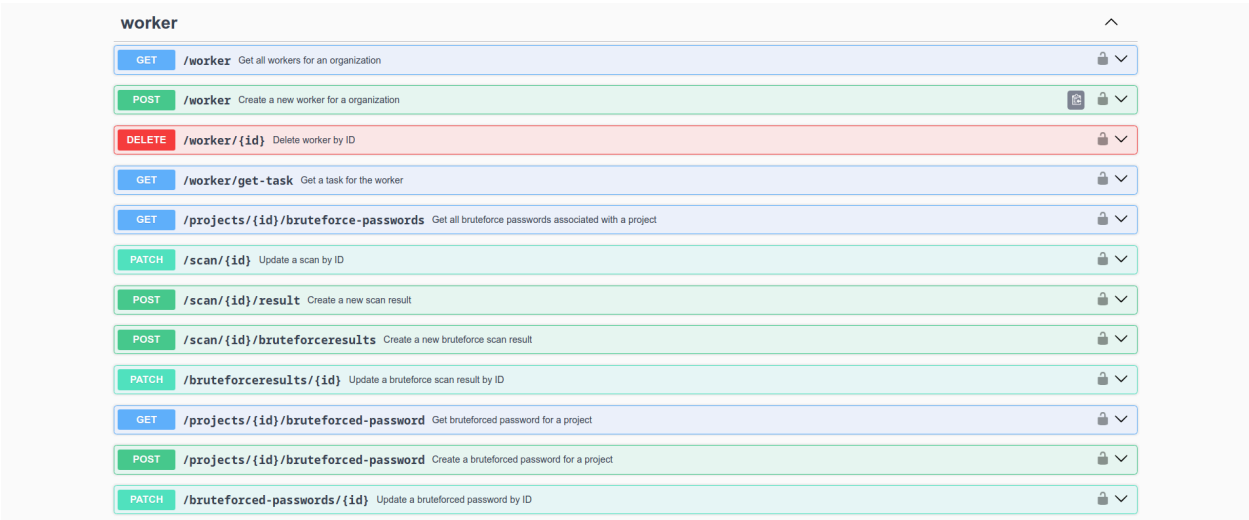

Figura 4.2. Descrierea specificației rutelor din categoria “worker”

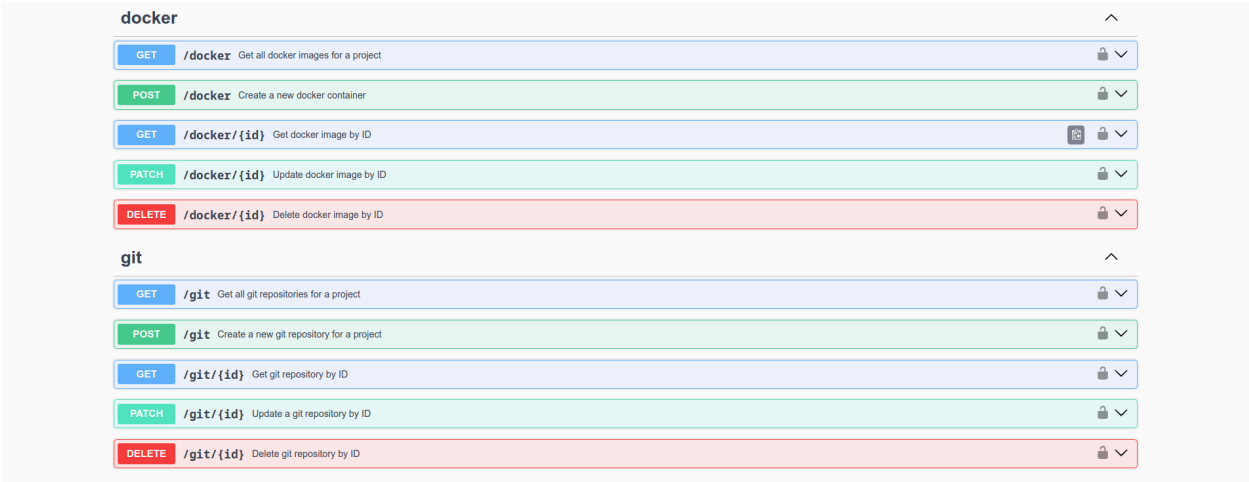

Figura 4.3. Descrierea specificației rutelor din categoriile “git” și “docker

Figura 4.4. Descrierea specificației rutelor din categoriile “project” și “scanner”

Figura 4.5. Descrierea specificației rutelor din categoriile “cve”, “postgres” și “mongo”

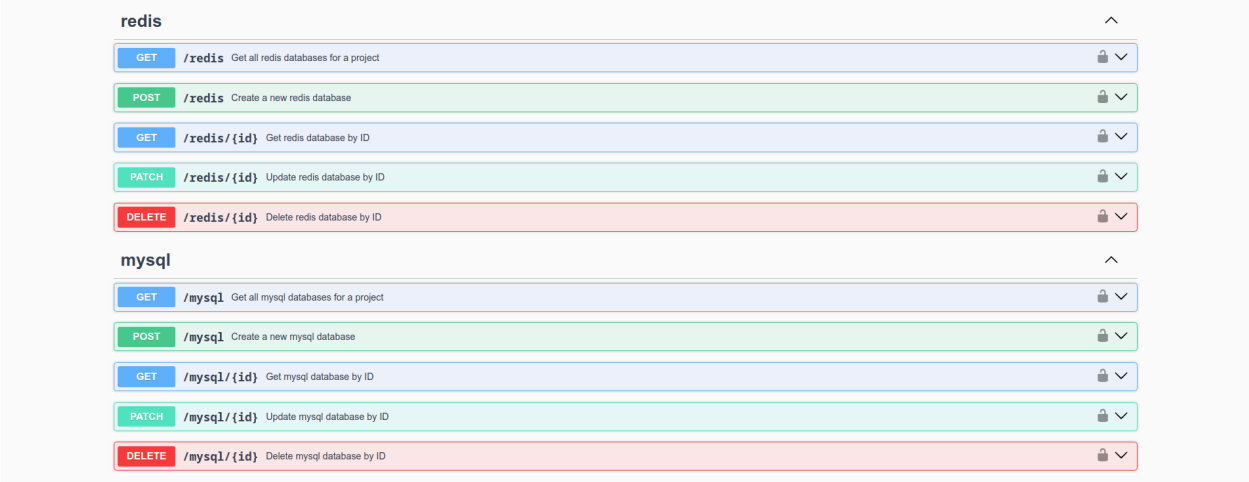

Figura 4.6. Descrierea specificației rutelor din categoriile “redis” și “mysql”

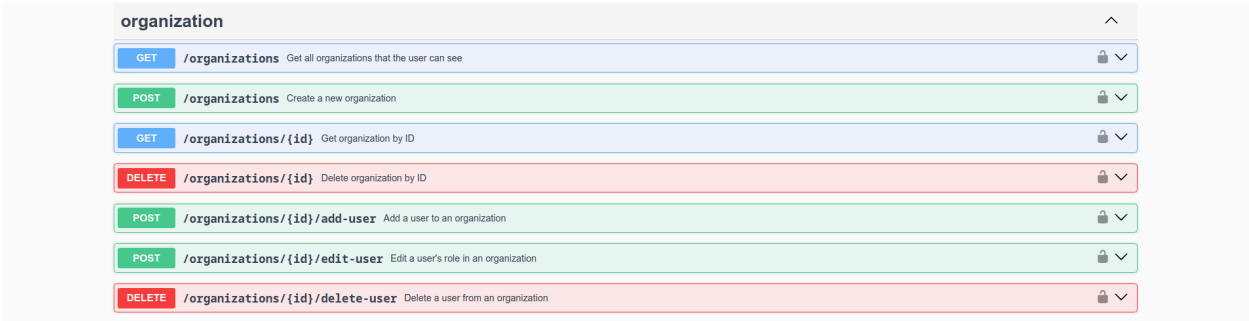

Figura 4.7. Descrierea specificației rutelor din categoria “organization”