## Heširanje lozinki

Heširanje je ključni korak u zaštiti poverljivih podataka, kao što su lozinke. Neke od najboljih praksiza heširanje lozinki uključuje korišćenje jakih, standardnih algoritama kao i implementaciju dodatnih mera kako bi se obezbedila njihova poverljivost. Danas, neki od najčešće korišćenih algoritama za heširanje jesu *Argon2id* i *bcrypt.* Aspekti kao što su bezbednost, performanse i lakoća korišćenja mogu uticati na izbor algoritma u zavisnosti od toga šta nam je prioritet. Iako *bcrypt* može biti preferiran zbog boljih performansi i šire podrške, kada je u pitanju bezbednost, *Argon2id* se izdvaja kao bolji izbor. *Argon2id*, iako je sporiji, namerno je dizajniran da bude memorijski i računarski intenzivan, što ga čini otpornijim na *brute force* napade.

Prilikom implementacije, važno je obratiti pažnju na konfiguraciju i adekvatan odabir parametara. Preporučena praksa uključuje podešavanje parametara poput minimalnog broja iteracija, minimalne veličine memorijskih zahteva i stepen paralelizma na odgovarajuće vrednosti koje odgovaraju specifičnim potrebama za sigurnošću. Neke od preporučenih vrednosti jesu:

* Memorija: 46 MiB, Broj iteracija: 1, Stepen paralelizma: 1
* Memorija: 19 MiB, Broj iteracija: 2, Stepen paralelizma: 1
* Memorija: 12 MiB, Broj iteracija: 3, Stepen paralelizma: 1
* Memorija: 9 MiB, Broj iteracija: 4, Stepen paralelizma: 1
* Memorija: 7 MiB, Broj iteracija: 5, Stepen paralelizma: 1

Prilikom implementacije *Argon2id* algoritma, preporučuje se korišćenje kriptografskih biblioteka i okvira, poput *libsodium*. Ova biblioteka se aktivno održava i ažurira, što osigurava primenu najnovijih bezbednosnih ispravki i unapređenja. Stoga, poslednja verzija *libsodium* za implementaciju *Argon2id* algoritma nema ozbiljnih ranjivosti i sigurna je za upotrebu.

Na osnovu gore navedenih stavki, sledeći zahtevi čine osnovu za bezbednu primenu mehanizma heširanja:

* Upotreba *Argon2id* algoritam za heširanje lozinke, koji pruža visok nivo bezbednosti.
* Konfiguracija parametara poput troškova vremena, troškova memorije i paralelizam na odgovarajući način kako bi se postigla ravnoteža između bezbednosti i performansi.
* Implementiacija *Argon2id* algoritmakoristeći pouzdan provajder kao što je *libsodium* ili druga dobro prihvaćena kriptografska biblioteka.
* Redovno praćenje bezbednosnih saveta i ažuriranja od provajdera kako bi se brzo rešili svi eventualni propusti.
* Osigurati da implementacija prati najbolje prakse za skladištenje lozinki, uključujući sigurno slanje i heširanje lozinki pre nego što se sačuvaju u bazi podataka.
* Primena praksi bezbednog kodiranja i sprovođenje redovnih bezbednosnih revizija radi identifikacije i adresiranja potencijalnih ranjivosti u implementaciji.

Pridržavajući se ovih zahteva, dizajniran mehanizam heširanja može efikasno zaštititi poverljivost korisničkih lozinki, istovremeno smanjujući potencijalne bezbednosne rizike.

## Mehanizam revizije (auditing)

Primarni cilj rukovanja greškama i revizije je da pruži korisne informacije za korisnika, administratore i timove za reagovanje na incidente. Cilj nije stvaranje ogromnih količina logova, već visokokvalitetnih logova, sa više korisnih nego praznih informacija [1]. Postoji nekoliko osnovnih zahteva koji svaki mehanizam za logovanje treba da ispuni, kroz koje ćemo u daljem tekstu da prođemo.

### Mehanizam za logovanje mora biti pouzdan, mora obezbediti dostupnost i integritet log datoteka

Prvi i glavni način kako ovo obezbediti jeste da ne pišemo sami svoje logove. Korišćenje console.log() funkcije kao i direktno pisanje u .txt fajlove treba uveliko izbegavati. Danas postoji veliki broj logging biblioteka koje se vrlo lako i pouzdano mogu integrisati u naše sisteme, poput *Log4j* za Javu, *Winston* za *NodeJS* ili *stdlib logger* za *Ruby on Rails*. Ovi alati se mogu kombinovati kako bi se implementirao robustan mehanizam za logovanje koji zadovoljava sve zahteve, uključujući pouzdanost, dostupnost i integritet log datoteka. Na primer, možete koristiti *Winston* ili *Bunyan* za logovanje u *Node.js* aplikaciji, a zatim *Elasticsearch*, *Logstash* i *Kibana* za centralizovano skladištenje, analizu i vizualizaciju logova. U ovakvom sličaju se logovi čuvaju na serverima ili u oblaku, čime se osigurava integritet i dostupnost podataka. Ovi alati takođe često podržavaju enkripciju podataka kako bi se osigurala bezbednost logova.

*Elasticsearch* posebno ima mehanizme za replikaciju podataka radi osiguravanja redundancije i visoke dostupnosti. Ovaj alat isto omogućava enkripciju podataka u pokretu (u toku prenosa) i miru (u mirovanju), čime osigurava bezbednost podataka.

Kod ovakvog rešenja još jedna bitna stavka bi bila da izbegavate da budete vezani za bilo kog određenog *vendor*-a. Organizujte svoju strategiju logovanja na takav način da, ukoliko se ukaže potreba, postane jednostavno zameniti biblioteku ili okvir za logovanje nekim drugim.

### Stavke log datoteke ne smeju sadržati osetljive podatke

Jako je bitno da obratimo pažnju na to koji se sve podaci loguju. Moramo osigurati da se nikada nikada ne čuvaju sledeći podaci:

* + - Lozinke
    - Brojevi kreditnih kartica
    - Brojevi socijalnog osiguranja
    - Identifikatori sesije Informacije koje je korisnik isključio
    - Tokeni za autorizaciju
    - PII (lični podaci koji mogu da identifikuju osobe, kao što su lična imena)

Ako smo se odlučili da koristimo neke od biblioteka za logovanje, možemo ovaj zahtev ispuniti tako što se oni konfigurišu da koriste filtere kako bi se automatski detektovali osetljivi podaci i izbrisali iz log poruka pre nego što budu zapisani. Ovo se može postići korišćenjem alata poput Logstash-a, koji podržava filtere za obradu log poruka.

U slučaju da koristimo biblioteku koja ne podržava direktno filtriranje osetljivih podataka ili izričito imamo potrebu za čuvanjem tih podataka, moramo obezbediti tokenizaciju ili enkripciju kako bismo sačuvali privatnost podataka.

### Svi događaji za koje su akteri bitni moraju biti zapisani, sa dovoljno informacija kako akteri ne bi mogli da poriču odgovornost (non-repudiation)

Pre nego što se akteru dozvoli pristup sistemu, potrebno je sprovesti autentifikaciju kako bi se potvrdilo da su identiteti korisnika tačni i pouzdani. To se obično postiže korišćenjem jednokratnih lozinki, biometrijskih podataka, sertifikata, ili drugih autentifikacionih metoda. Nakon autentifikacije, potrebno je primeniti stroge pristupne kontrole kako bi se odredilo šta svaki akter može i ne može da radi u sistemu. Ovo se postiže dodeljivanjem odgovarajućih privilegija na osnovu uloge ili identiteta korisnika.

Bitno je implementirati mehanizam za logovanje koji će beležiti sve bitne događaje koji se odvijaju u sistemu, uključujući aktivnosti aktera. Ove informacije treba da budu dovoljne da se potvrdi akcija koju je izveo svaki akter. Kada akter izvrši određenu akciju u sistemu, moguće je koristiti digitalne potpise kako bi se potvrdilo da je ta akcija izvršena od strane određenog aktera. Digitalni potpis se može generisati korišćenjem privatnog ključa aktera i validirati korišćenjem javnog ključa. Kada akter izvrši određenu akciju u sistemu, može se automatski generisati potvrda ili obaveštenje koje se šalje akteru kao dokaz da je akcija uspešno izvršena. Ovo može biti u obliku e-pošte, SMS poruke ili digitalnog dokumenta, i time obezbediti non-repudation.

Za dodatnu sigurnost i non-repudiation, može se razmotriti korišćenje blockchain tehnologije. Blockchain omogućava decentralizovano skladištenje podataka i transparentnost transakcija, što može dodatno otežati poricanje odgovornosti.

### 2.4 Log datoteke moraju pružiti informacije potrebne za razrešavanje problema

Svaki zapis u log datoteci treba da sadrži dovoljno informacija o događaju kako bi se omogućilo razrešavanje problema. To uključuje:

* + - vremenske oznake
    - identifikaciju aktera
    - vrstu događaja
    - parametre ili podatke relevantne za događaj i ostale relevantne informacije

U slučaju da koristimo već postojeću biblioteku za logovanje, verovatno nam ona pruža označavanje nivoa (*level*) logovanja. Radi bržeg razrešavanja i pronalaženja rešenja problema, korisno je koristiti adekvatan nivo logovanja.

* + - Nivo TRAGOVA: trebalo bi da se koristi samo tokom razvoja za praćenje grešaka, ali nikada u finalnoj verziji sistema
    - Nivo DEBUGOVANJA: na ovom nivou se evidentira sve što se dešava u programu. Ovo se uglavnom koristi tokom otklanjanja grešaka. U slučaju finalne verzijebi trebali ostati samo najsmisleniji unosi.
    - INFO nivo: evidentiraju se na ovom nivou sve radnje koje su vođene korisnikom ili specifične za sistem (tj. redovno zakazane operacije...)
    - Nivo NAPOMENA: ovo će biti nivo na kojem će program raditi u produkciji. Zabeležite na ovom nivou sve značajne događaje koji se ne smatraju greškom.
    - Nivo UPOZORENJA: na ovom nivou se evidentiraju svi događaji koji potencijalno mogu postati greška. Na primer, ako je jedan poziv baze podataka trajao duže od unapred definisanog vremena, ili ako je keš memorija blizu kapaciteta
    - Nivo GREŠKE: evidentiraju se sve greške na ovom nivou. To mogu biti API pozivi koji vraćaju greške ili interne greške.
    - FATALNI nivo: koristi se veoma retko, ovo ne bi trebalo da se dešava mnogo u finalnom programu. Obično evidentiranje na ovom nivou označava kraj programa.

### 2.5 Mehanizam za logovanje mora stremiti ka tome da su logovi uredni, da je “pretrpanost” minimalizovana

Ima puno načina kako možemo obezbediti urednost logova. Jedan od načina jeste tako što koristimo rotiranje log datoteka na osnovu određenih kriterijuma kao što su veličina datoteke ili starost pomaže u održavanju urednosti logova. Na primer, možemo konfigurisati alate poput logrotate-a da rotiraju log datoteke svakodnevno ili kada dostignu određenu veličinu. Arhiviranje starih log datoteka isto može pomoći u održavanju urednosti trenutnih logova. Potrebno je premestiti starije log datoteke na odvojeno skladište ili ih komprimujte radi uštede prostora, ali ih i dalje zadržite dostupnim za buduću analizu i referencu. Pored arhiviranja, takođe bi bilo korisno implementirati automatizovano čišćenje starih logova nakon određenog vremenskog perioda može sprečiti akumulaciju nepotrebnih podataka. Preporučeni periodi zadržavanja logova su sledeći:

1. Mala preduzeća ili startapi:

* Operativni logovi: 30-90 dana
* Sigurnosni logovi: 90-180 dana
* Pravni i regulatorni logovi: 1-3 godine (u zavisnosti od relevantnih propisa)

1. Srednje velika preduzeća:

* Operativni logovi: 90-180 dana
* Sigurnosni logovi: 180 dana - 2 godine
* Pravni i regulatorni logovi: 3-7 godina

1. Velike korporacije:

* Operativni logovi: 180-365 dana
* Sigurnosni logovi: 1-3 godine
* Pravni i regulatorni logovi: 7+ godina

### 2.6 Stavke log datoteke moraju precizno iskazati vreme nastanka

Većina log biblioteka automatski uključuje vremenski pečat (timestamp) za svaki zapis u log datoteci. Ovo omogućava precizno iskazivanje vremena nastanka svakog događaja i olakšava analizu logova. Timestamp obično obuhvata datum i vreme kada je događaj zabeležen, uključujući informacije poput godine, meseca, dana, sata, minuta i sekundi.

Na primer, u Winston biblioteci za logovanje u Node.js, timestamp se obično dodaje automatski na svaki zapis koristeći unapred definisan format. Možete prilagoditi format timestampa prema svojim potrebama.

## Dodatne bezbednosne kontrole

U ovom poglavlju će se revidirati bezbednosne kontrole koje su iskorištene na projektu rađenom iz predmeta „Informaciona bezbednost“. Backend projekta je rađen pomoću Spring Boot tehnologije.

### Heširanje lozinki

Za heširanje lozinki u ovom projektu je korišten PasswordEncoder interfejs koji je sastavni dio Spring Security paketa. Spring Security nudi različite implementacije datog interfejsa kao što su: bCrypt, Scrypt, Argon2 i drugi. U ovom projektu je konkretno korištena BcryptPasswordEncoder implementacija. Prilikom implementacije moguće je podesiti velićinu salt vrijednosti koju enkoder koristi kao i „cost“ faktor koji određuje koliko je zahtevan sam proces heširanja. „Cost“ faktor predstavlja eksponent broja dva koji može biti od 4 do 31. Korišteni su uobičajeni parametri u samom projektu, odnosno „cost“ faktor je 10, a dužina salt vrijednosti je 16 bajtova. Treba napomenuti da BcryptPasswordEncoder ne podržava promjenu veličine salt vrijednost, stoga bi bila potreba samostalna implementacija, iako podrazumijevana dužina se smatra dovoljno „sigurna“ za većinu aplikacija, Konkretna implementacija je korištena zbog široke podrške, ali bi i prelazak na Argon2 implementaciju bio jednostavan, jer je podržan od strane Spring Security-a.

### Mehanizmi revizije

Biblioteka koja je korištena u projektu je Log4j logging biblioteka koja predstavlja standard za aplikacije pisane u Java programskom jeziku. Sam kod je pisan tako da se izbjegne usko sprezanje sa samom bibliotekom. Biblioteka ne podržava automatsko filtriranje sadržaja koji se loguje, ali je moguće podesiti filtere koji prihvatati ili odbijati logove na osnovu sadržaja. U samom projektu nije podešeno automatsko filtriranje, ali je vođeno računa da se nikako ne otkrivaju osjetljivi podaci, a kako sama aplikacija ne sadrži neke osjetljive podatke nije bilo potrebe za enkripcijom logova. Prilikom svakog loga zapisuje se i akter koji je napravio određenu akciju u sistemu u vidu njegove email adrese što nije dobra praksa i ne preporučuje se logovanje PII podataka. Iako je potrebno zabilježiti identitet aktera, treba razmotriti drugi način kojim bi se mogao utvrditi identitet bez otkrivanja osjetljivih podataka. Nivoi logovanja koji su podržani su nivoi greške, info i debugovanja. Nedostatak drugih nivoa može dovesti do nečitkosti logova, a samim tim i do sporijeg razrešenja problema. Organizacija log datoteka nije realizovana u sklopu projekta. Potrebno je dodati mehanizme za brisanje starih logova kao i njihovo arhiviranje. Vremenski pečat je automatski uključen u svaki log od strane biblioteke i to u ISO 8601 formatu. Moguće je konfigurisati i druge formate, ali se preporučuje navedeni format zbog svoje univerzalnosti i kompatibilnosti.