OWASP TOP 10

1. **Injection**
    - *Code Injection*
        - Novije verzije Spring Boot-a nas štite od ovog napada.
    - *SQL Injection*
        - Hibernate, Spring i JPA nas štite od većine SQL napada. JPA prilikom kreiranja upita koristi *prepared statements* koje nas najbolje štite od ovog napada. Uz to, MySQL nas štiti od ovog napada tako što ne dozvoljava izvršavanje *multi-statement*-a.
        - Dodali smo validacije za sve DTO klase i URL parametre kako bismo sprečili napadača da unese neku naredbu u obično *input* polje i da naj način izrvši *injection* napad.
        - Prilikom ispisa grešaka vodili smo računa da ne prikazujemo korisniku nikakve informacije o arhitekturi naše baze.
    - *CRLF Injection*
        - Response Splitting
            - Stara ranjivost od koje nas štite sve novije verzije Java-e.
        - Log Injection
            - Upotrebili smo OWASP Java Encoder Project, kao preporučenu biblioteku za sprečavanje Injection i XSS napada. Pomoću njene *forHtml()* funckije enkodiramo log poruke pre upisa u log fajlove.
    - *LDAP Injection*
        - Od ove vrste napada smo se zaštitili definisanjem regularnih izraza i upotrebom Spring-ovih anotacaija za validaciju ulaznih podataka.
2. **Broken Authentication**
    - Prilikom prijave na sistem i promene lozinke, korisnik će biti blokiran u slučaju da sa iste IP adrese 3 puta unese pogrešnu lozinku.
    - Pri svakom definisanju lozinke od strane korisnika proveramo da li se ta lozinka nalazi na NIST-ovoj listi nebezbednih lozinki. Ako se nalazi, zahtevamo od korisnika da izabere neku drugu lozinku.
    - Sama šifra je validna ako sadrži najmanje 10 karaktera, od kojih mora biti po barem jedno veliko i malo slovo, broj i neki od specijalnih karaktera.
    - Lozinka se u bazi čuva *hash*-ovana i *salt*-ovana. Za to smo koristili Bcrypt (12 rundi prema *best practices*), koji prilikom kreiranja *hash*-ovanja koristi *salt*. Prilikom prijavljivanja na sistem, funkcija koja vrši poređenje šifre u bazi za unetom šifrom izvlači *salt* iz šifre u bazi i pomoću tog *salt*-a generiše *hash* unete šifre. Korisnik će biti uspešno ulogovan samo ako se izgenerisani *hash* i *hash* u bazi poklapaju.
    - Korisnik dobija mejl sa linkom za aktivaciju naloga. Aktivacioni link u sebi sadrži token koji je validan 3h od momenta registracije. Sve dok ne aktivira nalog putem primljenog linka, korisnik nije u mogućnosti da pristupi sistemu.
    - Prilikom resetovanja lozinke, korisnik prvo unosi svoju e-mail adresu na koju mu potom stiže mejl. U slučaju da korisnik unese nepostojeći mejl, biće mu

prikazana poruka da je mejl poslat, iako zapravo nije, čime se štite poverljivi podaci u sistemu. Mejl sadrži link sa tokenom koji je validan narednih 45 minuta. Klikom na link, otvara se forma za unos nove šifre. Nova šifra se zajendo sa tokenom šalje na server. Na njemu se vrši provera da li taj token postoji u bazi, da li je validan i da li pripada korisniku koji je resetovanje šifre i zahtevao.

- Trajanje JWT-a je podešeno na nisku vrednost (15 mininuta), pri čemu se odjavljivanjem kao i vremenskim istekom token invalidira.

3. **Sensitive Data Exposure**
    - U osetljive podatke u našoj aplikaciji spada korisnička lozinka i zbog toga smo je u bazi čuvali hešovanu. Za to smo koristili Bcrypt (12 rundi prema *best practices*), koji prilikom kreiranja *hash*-ovanja koristi *salt*. *Salt* predstavlja dodatni vid zaštite, jer dodavanjem nasumičnih karaktera obezbeđuje da dve identične lozinke nemaju isti *hash*.
    - Kredencijali za plaćanje kao što su: PAN broj, Security Code, Card Holder Name, Merchant id, Merchant password, Client id, Client Secret i Merchant token takođe predstavljaju osetljive podatke koje je potrebno zaštiti. Prilikom čuvanja ovih podataka koristili smo simetrično šifrovanje pri čemu smo obezbedili da samo osoba koja poseduje šifru može da dešifruje ove podatke. Simetrično šifrovanje smo vršili korišćenjem *AES/ECB/PKCS5Padding* algoritma.
    - Umesto HTTP protokola, u kom postoji rizik da će podaci biti otkriveni, koristili smo HTTPS protokol i digitalne sertifikate. Sertifikati upotrebljeni u okviru sistema se generišu pomoću RSA algoritma sa 4096 *key-size* za *certificate authority*, odnosno 2048 key-size za *end-entity* generisane sertifikate.
4. **XML External Entities (XXE)**
    - Razmena podataka u XML formatu je izbegnuta upotrebom JSON formata kao trenutno najrasprostranjenijeg standarda za razmenu podataka.
5. **Broken Access Control**
    - Primenili smo Role Based Access Control kako bismo kontrolisali pristup *endpoint*-ima svih mikroservisa. Prilikom implementacije RBAC-a ispoštovali smo preporuke sa OWASP Procative Controls.
    - Na frontend-u, koji je React aplikacija, kreirali smo *guard*-ove sa kojima smo onemogućili pristup stranicama svim korisnicima koji nemaju odgovarajuću permisiju.
    - Sistem bismo dodatno ograničili prilikom postavljanja u produkciju tako što bismo:
        - Za osetljive funkcije, poput pristupa administrativnim stranicama, dodatno ograničili pristup na osnovu IP adrese. Na taj način bi samo korisnici iz određene mreže mogli da pristupe odgovarajućim resursima.
        - Pomoću ACL-a dodatno ograničili pristup osetljivim fajlovima (kao što su log i kofiguracioni fajlovi), tako što kažemo operativnom sistemu koji korisnici imaju pristup konkretnom fajlu.
6. **Security Misconfiguration**

- Obzirom da operativni sistemi servera inicijalno nisu konfigurisani da budu sigurni, potrebno je izvršiti takvu konfiguraciju. Prilikom postavljanja u produkciju onemogućili bismo *debug*-ovanje i izlistavanje direktorijuma (*directory listing*), kako bi korisnicima sprečili pristup osetljivim podacima i izvršavanje koda.
- Takođe, neophodno je periodično vršiti proveru ranjivosti, kako bismo na vreme sprečili potencijalne napade.
- *Default* korisničke naloge za potrebe razvijanja aplikacije je neophodno izbrisati iz sistema prilikom produkcije.

7. **Cross-Site Scripting (XSS)**
    - Browseri nas štite od najtrivijalnih XSS napada.
    - React biblioteka nas od ovog napada štiti tako što tretira sve podatke kao *untrusted*, odnosno *escape*-uje i sanitizuje ih. Takav pristup dosta otežava napadaču da izvrši ovaj napad.

        „By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks."
8. **Insecure Deseralization**
    - Nastaje kao posledica deserijalizacije neproverenih (*untrusted*) podataka. Da bismo to sprečili, pomoću Spring-ovih anotacija i regularnih izraza smo validirali sve podatke koji stignu na server.
    - Takođe, upotreba formata za razmenu podataka, kao što je JSON, smanjuje mogućnost za ovu vrstu napada, jer se podaci prenose u tekstualnom obliku, a potom deserijalizuju upotrebom proverenih deserijalizatora.
9. **Using Components with Known Vulnerabilities**
    - Pustili smo OWASP Dependecy Checker nad svim mikrosevrisima kako bismo bili svesni svih ranjivosti biblioteka koje smo koristili. Zatim smo potražili detaljnije informacije o tim ranjivostima i pokušali smo da ih otklonimo.
    - Nakon postavljanja u produkciju, periodično bi trebalo pokretati alate kao što je OWASP Dependecy Checker kako bismo bili u toku sa novijim verzijama *dependency*-ja koje otkalanju ranjivosti prethodnih, ali i kako bismo na vreme otkrili pojave novih ranjivosti koje potencijalno mogu biti iskorišćene na našem sistemu.
10. **Insufficient Logging & Monitoring**
    - Logovali smo sve podatke o prijavljivanju na sistem, promeni i resetovanju lozinke i kontroli pristupa kako bismo mogli da identifikujemo sumnjive pristupe. Za svaki log fajl se pravi *backup* nakon što fajl dostigne veličinu od 10MB, što u našem slučaju podrazumeva oko 100.000 log poruka po fajlu.
    - Prilikom postavljanja u produkciju neophodno je usvojiti plan reakcije i oporavka od incidenta. Takođe, neophodno je da redovno *backup*-ujemo sve log fajlove, jer ako se čuvaju samo lokalno postoji mogućnost da će usled pada sistema nestati ili će ih napadači izbrisati, čime se direktno narušava

princip neporecivosti. Pored toga, bilo bi dobro i omogućiti da aplikacija detektuje trenutno aktivne napade.