OWASP Top 10

1. Injection

Injection napad predstavlja ubacivanje zlonamjerong koda koji je napadaču daje sve informacije iz baze podataka. Ovaj tip napada smatra se glavnim problemom i najvećim bezbjednosnim rizikom web aplikacije u OWASP-u. Ovu vrstu napada moguće je realizovati kada se nepouzdani (nevalidirani, neprovjereni, nefiltrirani) podaci šalju interpreteru kao deo komande ili upita. Zlonamjerni podaci mogu poruzrokovati izvršavanje komandi nad bazom podataka (čitanje, izmjenu, brisanje) i na taj način pristupiti podacima bez ovlašćenja. Vrste Injection napada su: Code Injection, SQL Injection, CRLF Injection, LDAP Injection.

- ***Code Injection:*** Novije verzije Spring Boot-a nas same štite od ovog napada.

- ***SQL Injection:*** Napad se sastoji od upita putem ulaznih podataka od klijenta do aplikacije. Uspješna eksploatacija SQL Injection-a može čitati osjetljive podatke iz baze podataka, mijenjati podatke baze podataka (Insert / Update / Delete). Da bi se zastitili od ovog napada potrebno je koristiti ***prepared statement*** iz razloga ako bi dinamički gradili SQL upit kao na primer konkateniranjem stringova, korisnisnikov unos će se tretirati jednako kao ostatak SQL naredbi i ukoliko je u tom unosu maliciozni SQL upit, on će se izvršiti zajedno sa našim prvobitnim SQL upitom. Kod *prepared statementa* ako korisnik unese za neko polje maliciozan SQL upit, taj upit će korisnikov unos gledati kao najobičniji podatak, a ne kao naredba, time je napad spriječen. 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. White list input ***validation*** - korisnikov unos ne smije da služi za formiranje SQL upita (tako da on piše direktno koja tabela, koja kolona ili uslov za ORDER BY), već mi treba da mu izlistamo šta sme da koristi (na primjer u slučaju sortiranje kod ORDER BY, ne puštamo korisnika da unese asc ili desc, već da na osnovu njegovog unosa, mi naš string dodamo u SQL upit) Za svaki korisnikov unos moramo imati regularni izraz koji će ispitati da li je taj unos validan. Regularni izraz predsatvlja white list. Stoga rijesenje je validiranje korisnikovih unosa, dodavaje validacija za sve DTO klase i URL parametre kako bismo spriječili napadača da unese neku naredbu u obično *input* polje i da naj način izrvši *injection* napad (NotBlank za stringove, NotEmpty za nizove jer provjerava da li je null i da li je duzina nula, ne smijemo sprijeciti č,ć, američka slova..) Prilikom ispisa grešaka voditi 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 ESAPI, open source API.

- ***LDAP Injection:*** Zastita od ovog napada je definisanje regularnih izraza i upotreba Spring-ovih anotacaija za validaciju ulaznih podataka.

2. Broken Authentification

Funkcije aplikacije vezane za autentifikaciju i upravljanje sesijama često se pogrešno implementiraju, omogućavajući napadačima da kompromituju lozinke, ključeve ili tokene sesije, kao i da iskoriste druge nedostatke implementacije da privremeno ili trajno preuzmu identitet drugih korisnika.

- Pri svakom definisanju lozinke od strane korisnika provjeramo da li se ta lozinka nalazi na Black listi nebezbjednih lozinki. Ako se nalazi, zahtjevamo od korisnika da unese 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 nakon registracije na sistem, dobija mejl sa linkom za aktivaciju naloga. Aktivacioni link u sebi sadrži token koji je validan 24h od momenta registrovanja. 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.
- Takođe smo obezbijedili i dvostruku prijavu na sistem tj. *Two factor authentication*. Nakon uspješnog logovanja korisniku stiže na mejl kod (random string) koji važi 5 minuta i koji mora unijeti kako bi izvršio uspješno logovanje na sistem. Kod čuvamo u skladištu hešovan kako bismo bili sigurni da neko ko pristupa skladištu podaka ga neće moći zloupotrijebiti.

3. Sensitive Data Exposure

Prikazivanje, slanje i čuvanje osjetljivih podataka predstavlja opasnost za aplikaciju. Mnoge web aplikacije i API-ji ne štite ispravno podatke kao što su na primjer brojevi kreditnih kartica, lični i zdravstveni podaci. Napadači mogu da ukradu ili modifikuju takve, slabo zaštićene podatke kako bi izveli različite prevare, krađe identiteta ili druge zločine. U

slučaju da podaci nisu šifrovani, a komunikacija nije zaštićena, napadači mogu presresti i ukrasti ove podatke.

- U osjetljive 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*.
- Umesto HTTP protokola, u kom postoji rizik da će podaci biti otkriveni, koristili smo HTTPS protokol i digitalne sertifikate.

4. XML External Entities (XEE)

Do ove vrste napada može doći kada stariji ili loše iskonfigurisani parseri parsiraju unos koji sadrži referencu na spoljašnji entitet. Ovaj napad može prouzrokovati denial-of-service napad uključujući beskonačan fajl, skeniranje portova iz perspektive mašine gde se parser nalazi, kao i otkrivanje poverljivih podataka.

- Upotreba jednstavnijih formata podataka, poput JSON kao i izbjegavanje serijalizacije osjetljivih podataka.
- Provera XML poslatih fajlova i njihova validacija upotrebom šeme. Ažuriranja biblioteka za obradu XML dokumenata (Mi još uvijek nemamo xml dokumente u projektu).

5. Broken Access Control

Ograničenja o tome šta je autorizovanom korisniku dozvoljeno, često nisu pravilno primenjena. Napadači mogu iskoristiti ove nedostatke kako bi pristupili neovlašćenim funkcijama ili podacima, kao što je na primjer pristup nalozima drugih korisnika, pregled osjetljivih datoteka, izmjena podataka drugih korisnika, promena prava pristupa itd.

- Primjenili smo Role Based Access Control kako bismo kontrolisali pristup *endpoint*-ima svih mikroservisa. Svakom korisniku dozovoljeno je da pristupa samo onim delovima aplikacije za koje ima potrebne dozvole.
- Takođe, korisnik sa svakim zahtevom šalje i token koji dobija prilikom uspešnog logovanja. Token se poništava nakon što se korisnik izloguje.
- Bilježenje svakog neuspješanog pokušaja logovanja koristeći loging mehanizam.
- 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

Loša konfiguracija bezbednosnih kontrola je veoma česta. Do ove vrsta napada može doći zbog instaliranih i dozvoljenih komponenti koje nisu potrebne, kao i ako poruke o greškama sadrže previše informacije i otkirvaju informacije o unutrašnjosti sistema. Od ključnog je značaja definisati, implementirati i redovno održavati bezbednosne konfiguracije.

- Koristimo samo minimalan skup komponenti i biblioteka koje su neophodne i koje zaista koristimo u projektu.
- Rukovanje greškama ne otkriva previše, tako na primer prilikom logovanja u slučaju da korisnik ne unese dobru lozinku ili korisničko ime, neće dobiti tačne informacije šta je pogrešno, već samo poruku da se kredencijali ne poklapaju.
- Implementiran je globalni Exception handler, pa tako u slučaju da se desi neka greška koju nismo predvidjeli, korisnik neće biti u mogućnosti da pročita cijeli stack trace, već samo poruku da je došlo do greške i da pokuša kasnije.
- Takođe, neophodno je periodično vršiti provjeru ranjivosti, kako bismo na vrijeme spriječili potencijalne napade.

7. Cross-Site Scripting (XSS)

XSS napad se izvršava kada aplikacija sadrži nepouzdane podatke prilikom učitavanja nove web stranice, bez odgovarajuće provjere validnosti i escape-ovanja. XSS omogućava napadaču da izvrši zlonamjernu skriptu u pregledaču žrtve. To se obično radi umetanjem zlonamjernog koda u legitimnu web stranicu. Koristeći ovu tehniku, napadač bi mogao: izmijeniti sadržaj veb stranice, preusmjeriti korisnika na drugu, možda zlonamjernu web stranicu, pristupiti korisničkim sesijama i koristiti ove informacije za lažno predstavljanje korisnika, pristupiti kritičnim informacijama o korisnikovom sistemu, kao što su lokacija, web kamera, njihov sistem datoteka itd. Postoje tri vrsta XSS napada: Stored XSS, Reflected XSS i DOM XSS.

- Browseri nas štite od trivijalnih XSS napada.
- Ukoliko moramo da ubacujemo dinamički html potrebno je korisnikov sadržaj enkodovati. Izbjegavamo korišćenje innerHTML, umesto toga koristimo append.
- Izbjegavamo unos nesigurnih podataka kao sadržinu event handlera ili javascript koda. U ovom slučaju čak i ako javascrip enkriptujemo, opet će se izvršiti jer će browser na isti način interpretirati sadržaj.
- Ne ostavljamo nesigurne podatke unutar komentara.
- Kod JSON objekata obratili smo pažnju da je tip podataka (Content-Type) application/json a ne text/html.
- U HTTP headeru naveli smo da želimo da X-XSS-Protection bude uključen i da ukoliko se desi neki XSS napada da ga browser odmah prekine.
- http.headers().xssProtection().and().contentSecurityPolicy(„script-src’self“);
  Ovi headeri štite Api od snimjljenog XSS-a.

- Još jedan vid zaštite od XSS vezan za JavaScript i JQuery jeste to da nam je potreban vlastiti HTML koder zato što JavaScript ne pruža API za kodiranje HTML-a. Pretvoriti specijalne karaktere kao što su ?, &, /, <,> itd. u njihove odgovarajuće HTML ili URL kodirane ekvivalente. Na ovaj nacin izbjegavamo da korisnik unosi HTML context u JavaScript.

8. Insecure Deseralization

U aplikacijama može postojati bezbjednosni propust ako one deserijalizuju podatke od nepoznatih napadača. Napadi mogu da modifikuju logiku aplikacije ili da postignu izvršavanje koda preko klasa koje svoje ponašanje mijenjaju za vrijeme deserijalizacije.

- Ovaj napad nastaje kao posljedica deserijalizacije neprovjerenih (*untrusted*) podataka. Da bismo to spriječili, pomoću Spring-ovih anotacija i regularnih izraza smo validirali sve podatke koji stignu na server.
- Takođe, upotreba formata za razmenu podataka kao što su JSON i XML, smanjuje mogućnost za ovu vrstu napada, jer se podaci prenose u tekstualnom obliku, a potom deserijalizuju upotrebom provjerenih deserijalizatora.

9. Using Components with Known Vulnerabilities

U aplikacijama se koriste razne biblioteke kako bi ostvarili sve neophodne funkcionalnosti. Upotreba ovih komponenti donosi ranjivosti i rizik zbog neprovjerenosti korišćenih materijala i mogućnosti da imaju neku slabost koju napadač može da iskoristi za pristup osetljivim podacima.

- Pustili smo OWASP Dependecy Checker nad svim mikrosevrisima, kako bismo bili svjesni svih ranjivosti biblioteka koje smo koristili. Zatim smo potražili detaljnije informacije o tim ranjivostima i pokušali smo da ih otklonimo. Ažurirali smo starije verzije biblioteka na novije koje nemaju poznatih ranjivosti. Na frontu koristimo isključivo biblioteke koje su provjerene, uvezali smo linkove sa zvaničnih sajtova koje imaju digitalne potpise.
- 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 vrijeme otkrili pojave novih ranjivosti koje potencijalno mogu biti iskorišćene na našem sistemu.

10. Insufficient Logging & Monitoring

Nedostatak evidencije i praćenja aktivnosti napadačima pruža veću šansu za pristup aplikaciji i za njeno uništavanje. Nedostatak praćenja akcija prijave, neuspelih transakcija, neprijavljivanje greške i upozorenja na veliki broj grešaka uzrokuju da nije moguće u realnom vremenu otkriti napadača. Većina studija pokazuje da je vrijeme potrebno za otkrivanje ove vrste napada bilo preko 200 dana.

- Mikroservisi su konfiguraisani tako da log zapise u log fajlovima kreiraju na identičan način. Različiti tipovi logova su smješteni u odvojene fajlove što

doprinosi njihovoj boljoj preglednosti i lakšoj analizi. U log fajl se upisuju svi zahtjevi koji pristižu u aplikaciju, kao i odgovori na zahtjeve.

- Logovali smo sve podatke o prijavljivanju na sistem, registraciji, aktivaciji profila, resetovanju lozinke i verifikaciji profila kako bismo mogli da se identifikujemo sumnjive pristupe. Logovali smo takođe i kreiranje omiljenog post-a, kreiranje post-a, brisanje post-a i kreiranje story-ja.
- 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.