OWASP Top Ten Web Application Security Risks

1. Injection napadi:

   Injection napadi se odnose na klasu napada koji omogućavaju napadačima da daju nepouzdane podatke prilikom unosa koje prevodilac obrađuje kao dio naredbe ili upita čime se mijenja tok izvršenja programa (dolazi do izvršavanja neautorizovanih instrukcija pomoću kojih se pristupa neautorizovanim podacima). Injection napadi obuhvataju sljedeće napade: SQL, NoSql, XXE, OS, LOG, Xpath, ...

   - SQL napadi: Napadači koriste ranjivost web aplikacija za pristup bazi podataka. Na taj način moguće je ugroziti sigurnost web aplikacije koja koristi SQL upite iz podataka unešenih od strane korisnika. Napadač ovaj napad koristi za izmjenu konstrukcije pozadinskog SQL upita. Ako napadač uspije da izmijeni upit posljedica do koje može doći jeste neautorizovano preuzimanje kontrole nad podacima u bazi podataka. Neki od nizova znakova koje napadač može unijeti su: 105 OR 1=1 što predstavlja sljedeći upit: SELECT * FROM SOME_TABLE WHERE id = 105 OR 1=1.
     - Rješenje: JPA Hibernate u sebi sadrzi validator koji sprečava napade ako se upiti pišu na pravi način. Naime, u slučaju naših aplikacija koristimo parametrizovane upite. Lijepljenje stringova kako bi se formirao konačan upit dovodi do pogodne situacije u kojima se može umetnuti neki sql izraz putem forme. Upotreba pravilno kreiranih metoda za sql upite: findById(Long id), findByName(String name),.. omogućava zaštitu od ovakvih napada. U ovom slučaju je potrebna relevantna validacija parametara koji se šalju, id, name, itd. Umetnuti podaci će se na adekvatan način izbjeći zahvaljujuci JDBC drajveru tako da se zbog upotrebe parametrizovanih upita sadržaji koriste kao podaci a ne kao kod.
   - LOG napadi: Aplikacije često koriste log fajlove u kojima se čuvaju istorije događaja ili transakcija što omogućava prikupljanje statistike, kasniji pregled ali i omogućava otklanjanje grešaka koje postoje u sistemu. U zavisnosti od prirode aplikacije pomenute radnje se mogu automatizovati ili ručno obavljati. Pisanje nevažećeg unosa u datoteke koje vode evidenciju događaja omogućava napadaču da krivotvori unose u fajlove. Ranjivosti se javljaju kada podaci u aplikaciju dolaze od nepouzdanog izvora ili kada se podaci upisuju u log fajlove. Ukoliko su log napadi uspješni otvara se mogućnost ubrizgavanja novih/lažnih dnevnika za vođenje evidencije ali i XSS napada pri čemu se napadači nadaju da će krivotvoreni fajlovi biti pregledati pri provjeri ranjivosti aplikacije.
     - Rješenje: Upotreba Logger klase iz slf4j biblioteke. Ovim se omogućava evidentiranje događaja koji mogu da budu opasni: neuspješno logovanje, itd. Takođe, ovo smo iskoristili i za evidenciju drugih radnji pri čemu se mogu pratiti druge aktivnosti korisnika aplikacije što omogućava poboljšavanje pojedinih funkcionalnosti ako se za tim javi potreba.

2. Broken Authentication:

   Funkcije aplikacije koje se odnose na autentifikaciju i upravljanje sesijom često se pogrešno implementiraju, omogućavajući napadačima da kompromituju lozinke, ključeve i tokene sesija ili da iskoriste nedostatke implementacije kako bi privremeno ili trajno preuzeli identitet drugih korisnika. Upotreba REST arhitekture koja je stateless je olakštavajuća okolnost u sistemu. Mogući problemi su:

   - Upotreba slabih lozinki: rješenje - lozinka sadrži minimalno 10 karaktera (podržani su velika i mala slova, cifre i specijalni karakteri), pored toga, lozinka ne sme da sadrži username

- Oporavak naloga: za oporavak naloga se ne koriste knowledge-based-answers tehnike. Od korisnika se zahteva da unese mejl koji je povezan sa njegovim nalogom na servisu na koji mu stiže verifikacioni mejl sa jednokratnom lozinkom koju može da iskoristi u narednih 24h
- Prevelik broj neuspešnih prijava na sistem: sistem podržava logging broja neuspešnih logovanja kao i ip adresu i port sa kog je pokušano prijavljivanje. Nakon određenog broja neuspešnih pokušaja (5) korisniku se blokira nalog na određeno vreme
- heširanje lozinke i upotreba salt-a: Upotrebnom salt-a, načina čuvanja lozinki koji na postojeću lozinku dodaje slučajan string se otežavaju dictionary napadi. U aplikaciji je korišten PasswordEncoder koji koristi Bcrypt algoritam sa 10 rundi ponavljanja. Runde mogu da se povećaju na 12 kako bi se poboljšala nasumičnost algoritma. BCrypt interno generiše nasumičan salt pri svakom heširanju lozinke. String passworda je dužine 60, od čega prvih 23 karaktera predstavlja salt
- Nezaštićeni prenos kredencijala: rješenje – upotreba TLS odnosno HTTPS protokola,
- Sesija: upotreba jwt tokena koji se salje na klijentsku stranu i čuva u locale storage. Session ID nije vidljiv u URL-u i važi isključivo u okviru jedne sesije. Sesije se prekidaju logout metodom.

3. Sensitive Date Exposure:

   Slaba zaštita osjetljivih podataka može dovesti do zloupotrebe od strane napadača. Naime, napadači mogu da preuzmu ili modifikuju neke podatke koji su osjetljivi ili bitni za određeno poslovanje. U ove podatke se ubrajaju i lozinke, lični podaci, kreditne kartice, poslovne tajne i sl.

   - Rješenje: Veza između klijenta i servera je preko TLS, tj. HTTPS protokola. Osjetljivi podaci se ne čuvaju kao otvoreni tekst. Lozinke se heširaju ali i proširuju sa salt-om (slučajnom stringom). U script fajlovima putanje koje se koriste za dobaljanje resursa su učitane kao enviroment variable te nisu vidljive kao otvoreni tekst. Lozinke se ne vraćaju u browser, a korinička imena i drugi osetljivi podaci upakovani su u json web token.

4. XML External Entities (XXE):

   Svi podaci koji se razmjenjuju u SOAP protokolu se validiraju putem XML šeme (WSDL stil je wrapped-document/literal). Upotrebom JAXB parsera onemogućeni su eksterni entiteti pa je mala vjerovatnoća za XXE napade.

5. Broken Acess Control:

   Ograničenja u vezi sa onim šta autentifikovani korisnici mogu da rade. Napadači mogu da iskoriste ove mane za pristup funkcijama i/ili podacima za koje nisu autorizovani: pregled osjetljivih datoteka, izmjena podataka drugih korisnika, promjena prava pristupa, itd.

   - Neautorizovan pristup metoda: Upotreba RBAC modela za kontrolu pristupa. Kontrola pristupa je realizovana RBAC modelom koji reguliše uloge koje korisnici imaju kao i prava pristupa metodama na osnovu dodjeljenih privilegija koje su povezane sa ulogama. Jedna uloga moze da ima više privilegija, jedna privilegija(dozvola) moze da ima više uloga. Svaki korisnik sistema može da ima više uloga koje mu se dodeljuju prilikom registracije, a jedino administrator sistema ima dozvolu izmene prava pristupa. Autorizacija je sprovedena upotrebom spring security filter chaina koji proverava Authorization header iz njega na osnovu jwta nalazi korisnika sistema i

njegove permisije. Prolazak kroz filter chain nije moguć ukoliko u zahtevu ne postoji Authorization header ili ako je token u njemu nevalidan. Kontrola pristupa metodama se realizuje upotrebom @PreAuthorize anotacija koje definišu permisiju za svaku metodu kontrolera.

- Izmena html-a ili putanje: osim rbac modela za kontrolu pristupa na backendu, rbac je sproveden i na front aplikaciji. Svaka komponenta i html stranica je autorizovana na osnovu role trenutno ulogovanog korisnika. Za sprovođenje autorizacije na frontu se koristi react biblioteka: react-router-role-authorization koja omogućava komponentama da naslede RoleAwareComponent klasu i na osnovu uloge korisnika prikažu komponentu ili ga redirektuju na početku stranicu
- Upotreba zastarelog tokena: nakon logout-a, token se invalidira i uklanja iz local storage-a, a na bekendu je omogućeno validiranje tokena u filteru pre pristupa bilo kojoj metodi, tako da je sprečena upotreba tokena koji je instekao ili je invalidiran
- Putem ACL je onemogućeno izmjena fajlova za logging i pristup konfiguracionim fajlovima je zabranjen

6. <u>Secutity Misconfiguration:</u>
   Security misconfiguration podrazumjeva aplikaciju u kojoj se koriste neažurirani softveri, sistem za upravljanje bazom podataka kao i neažurirane upotrebljene biblioteke. Pored navedenog uključuje se i otkrivanje previše informacija korisnicima pri rukovanju greškama. Uzimajući u obzir pomenute osobine treba voditi računa i o podizanju aplikacije na Internet. Tada je potrebno podesiti React da više ne bude u dev mode-u kakav je bio pri razvoju već da bude u production mode-u. Production mode onemogućava prikaz logova koji su postojali prilikom implementacije, koji bi mogli otkriti previše informacija korisnicima ali i zatvara debug port aplikacije.

7. <u>XSS (Cross-Site Scripting) napadi:</u>
   Ovi napadi se dešavaju kada se podaci unose u web aplikaciju putem nepouzdanog izvora, najčešće web zahtjeva ili su podaci uključeni u dinamički sadržaj koji se šalje web korisniku bez provjere zlonamjernog sadržaja. Zlonamjerni sadržaj se šalje web pregledaču najčešće u obliku JavaScript-a, ali takodje moze da zadrži HTML, Flash ili bilo koju drugu vrstu koda koji pregledač može izvršiti. Raznolikost ovih napada je skoro neograničena ali najviše se zasnivaju na prenošenju privatnih podataka poput kolačića ili drugih informacija o sesiji. Napadač preusmjerava 'žrtve' na web sadržaj koji on kontroliše.
   - Reflected xss: Napadi u kojima se maliciozna skripta ubacuje kao string i predstavlja deo korisnickog zahteva prema serveru. Server zatim ukljucuje taj string u odgovor koji se prikazuje u obliku error poruke, rezultata pretrage i sl. Reflektovani XSS napadi se isporučuju korisniku sa neke druge putanje: pomoću email poruke ili sa nekog drugog sajta.
     - ✓ Rešenje: Rad na React framework-u je dosta pomogao jer je kod React-a zaštita od HTML stringova po default-u prisutna. React automatski izbjegava variable stringa i to sprečava ubrizgavanje XSS napada kroz stringove sa zlonamjernim JavaScript-om. Međutim, bez potpunog oslanjana na ovo, radio se escaping parametara koji dolaze iz input polja formi kroz uklanjanje potencijalno opasnih znakova (<, >, ..)

- Stored xss: Napadi u kojima se ubacena maliciozna skripta trajno cuva na serveru (u bazi podataka, u polju komentara, kao post na forumu i sl...) i poziva se svaki put kada korisnik "pogodi" odgovarajucu funkcionalnost ili pri svakom ucitavanju stranice. Ovakav napad se smatra rizicnijim, izaziva vise stete vecem broju korisnika. Da bi se izvrsio stored XSS napad, potrebno je implementirati maliciozni kod u neko input polje (neki primeri su: forma, pretraga, komentar).
    - Na serverskoj strani koristili smo provjerenu klasu Encode koja je predložena od strane OWASP zajednice (import org.owaps.encoder.Encode) i njenu metodu forHtml (Encode.forHtml(String data)) koja vrši ecsaping karaktera pre čuvanja objekta i bazu, kako ne bi došlo do stored xss napada.

8. Insecure Deserialization:
   Iskorištavanje deserijalizacije je teško. Web aplikacije redovno koriste serijalizaciju i deserijalizaciju, a većina programskih jezika nudi izvorne funkcije za serijalizaciju podataka (naročito u uobičajene formate poput JSON i XML-a). Problem se javlja u postupku deserijalizacije. Većina programskih jezika nudi mogućnost prilagođavanja procesu deserijalizacije. Nažalost, često napadač može da zloupotrijebi funkcije deserijalizacije kada aplikacije deserijalizuje nepouzdane podatke koje napadač kontroliše. Ovo bi moglo da napadaču omogući da izvrši, između ostalih, i napad izvršavanja udaljenog koda. Aplikacije napisane Java jezikom su takođe podložne ranjivostima deserijalizacije. Mogućnost koja može da prevaziđe ove ranjivosti je:
   - Implementacija provjere intergitera kao što su digitani potpisi na bilo kom serijalizovanom objektu kako bi se spriječilo stvaranje zlonamjernih objekata ili neoplašćeni pristup podacima.

     Naša apliakcije podatke čuva u bazi podataka pri čemu se ne vrši serijalizacija/deserijalizacija podataka, stoga, bojazan od ovih napada ne postoji.

9. Using Components with Known Vulnerabilities:
   Ovi napadi se dešavaju kada razvojni tim ne zna verzije svih komponenti koje koristi (client & server side). Ovo uključuje i runtime okruženja, biblioteke koje se koriste ali i sistem za upravljanje bazama podataka. Do ovih napada dolazi kada programeri ne testiraju kompatibilnost ažuriranih ili nadograđenih biblioteka.
   - Za zaštitu od ovih ranjivosti potrebno je da se uklone sve nepotrebne funkcije, datoteke, dokumentacija,.. Pored ovoga, potrebno je voditi evidenciju verzija svih komponenti koje se koriste i na klijentskoj i serverskoj strani. Moguće je pretplatiti se na upozorenja putem mejla za bezbjedosne ranjivosti u vezi sa komponentama koje koristimo. Komponente koje koristimo treba da nabavimo samo sa zvaničnih izvora preko sigurnih veza pri čemu treba preferirati potpisane pakete kako bi se smanjila šansa za zloupotrebu tih komponenti koje koristimo. Za ovu aplikaciju napravljen je spisak korišćenih biblioteka i ostalih komponenti čije ranjivosti su testirane pomoću owasp dependency check alata. Xml šema sa testiranim bibliotekama se nalazi na kraju dokumenta. Dependency check nije detektovao ni jednu ranjivost postojećih biblioteka.

10. Insufficent Logging and Monitoring
    Dovoljno evidentiranje i nadgledanje aktivnosti koje se dešavaju u velikoj mjeri mogu da spriječe bezbednosne rizike koji se mogu javiti. Napadač najčešće provodi dosta vremena „proučavajući“ aplikaciju ili neki sistem kako bi pronašao ranjivosti. Ako sistem nema dovoljno evidentiranja i praćenja, napadač može da mirno traži greške i ranjivosti. To povećava šanse da uspješno pronađe i iskoristi ranjivost. Jako je bitno da postoji barem jedan

mehanizam za obavještavanje o napadu. OWAPS je predložio mjere koje omogužavaju zaštitu od ove ranjivosti. Ono što je implementirano u ovoj aplikaciji je sljedeće:

- Sve greške prilikom prijave treba da se evidentiraju sa dovoljno korisničkog konteksta kako bi se identifikovale sumnjive ili zlonamjerne radnje. Za ovo se koristila klasa Logger iz biblioteke slf4j pri čemu je moguće da aplikacija nastavi sa radom i prilikom otkaza Loggera. Implementirana su četiri tipa Loggera: prvi su info loggeri kod kojih se bilježe INFO aktivnosti, drugi su logovi gdje se bilježe ERROR aktivnosti, treći su namjenjeni za WARN, dok postoji poseban fajl u kom se sve čuva. Nakon dostignute određene veličine fajla (50MB), kreira se drugi koji se koristi. Broj arhiviranih fajlova je 30, za svaki tip, nakon čega se automatski brišu. Svako pristupanje aplikaciji (uspešno ili neuspešno se čuva u log fajlu sa ip adresom i portom sa koje je pokušan pristup). Osim ove, svaka izmena i/ili brisanje entiteta aplikacije (za našu aplikaciju je od ključnog značaja integritet oglasa i vozila koji mogu da se iznajmljuju) se upisuje u log fajl.
- Sve poruke unutar loga imaju naznačen nivo (TRACE, DEBUG, INFO, NOTICE, WARN, ERROR, FATAL)
- Vreme je predstavljeno pomoću UTC-a (Coordinated Universal Time) i offset-a da bi bilo čitljivo i developeru i osobi zaduženoj za monitoring

Format log zapisa:
2020-06-10 16:06:02.092 | INFO | [AdvertisementController] - |ADD AD| user: prodavac ad id: 1