**OWASP TOP 10**

**10. Unvalidated redirects and forwards**

Jedan od načina zaštite od ove vrste napada na back-endu je izbegavanje redirect i forward mehanizama koji smo i mi usvojili. Na kontrolerima na back-endu (Spring) nemamo ni jedan redirect ili forward, return nam predstavljaju podaci koji se naknadno obradjuju na front-endu i prikazuju korisnicima.

Na front-endu (AngularJS) koristimo $state mehanizam koji nam omogućavamo da redirektujemo na osnovu naziva stanja, a ne zadavanjem URL koji bi omogućio napadaču da pošalje maliciozan string koji upućuje na maliciozan sajt. Ovo je implementirano u route.js fajlovima.

**9. Using components with known vulnerabilities**

Na osnovu analize **OWASP dependency-check** ustanovljeno je da naša aplikacija koristi zastarelu verziju Spring Boot-a, pa smo sa verzije 1.3.9. prebacili na 1.4.4. Uz nju su se update-ovali i ostali dependency.

Takodje je koriscena zastarela verzija AngularJS-a (1.5.8.) pa smo je update-ovali na 1.6.1.

**8. Cross-site request forgery**

Kao zaštita implementiran je i HTTPS mehanizam koji omogućava kriptovanje zahteva i smanjuje mogućnost da se ukradu podaci poslati na server, tj. podaci koji se cuvaju u Cookie-u.

Problem kod ovoga je što ako neko drugi koristi korisnikov računar i može da pristupi Cookie-u. Zbog pomenute pretnje moramo korisnika obučavati i skrenuti mu pažnju kako bi morao da se ponaša kako ne bi na naivan način ugrozio svoju bezbednost na internetu, kao i bezbednost same aplikacije.

**7. Missing function level access control**

Ova odbrana nije potrebna, jer aplikacija ne poseduje korisničke uloge(admin, korisnik itd.)

**6. Sensitive data exposure**

Ovi sigurnosni propusti, koji se javljaju prilikom izlaganja osetljivih podataka, rešeni su tako što je koršćen HTTPS (TLS) protokol, kako bi se obezbedila enkripcija podataka koji se šalju. Dodatna zaštita je omogućena za korisnikov JMBG i broj pasoša. Oni su enkriptovani pre nego što su stavljeni u cookie.

Postoje još neke mere zaštite, koje je neophodno uzeti u obzir, a nisu korišćene u ovoj aplikaciji. Na primer, neophodno je isključiti autocomlete u formama kod prikupljanja osetljivih informacija, takođe je neophodno i isključiti keširanje stranica koje sadrže osetljive informacije.

**5.Security misconfiguration**

Sa obzirom da nećemo postavljati našu aplikaciju da trči na nekom pravom serveru, tj. nećemo je stavljati u konkretnu upotrebu, ova tačka nije neophodna da se uradi.

**4. Insecure direct object references**

Naša aplikacija ne poseduje nikakve korisničke uloge, tj. ne radi sa njima, pa nema opasnosti od krađe poverljivih informacija. Takođe, ostale osetljive informacije se ne prikazuju u URL-u.

**3.Cross site scripting**

Ovaj problem je rešen koristeći [*JSOUP*](https://jsoup.org/cookbook/cleaning-html/whitelist-sanitizer) *API*, koji između ostalog štiti serversku stranu od XSS napada.

JSOPU poseduje svoju *Whitelist-u*, na osnovu koje dozvoljava određene HTML tagove.

Koristeći metodu **Jsoup**.clean(unsafe, **Whitelist**.basic());

gde je *“unsafe”* string koji potencijalno sadrži opasnu skriptu, rešavamo se bilo kakve opasnosti da se taj string nekako uvuče i izvrši u našoj aplikaciji.

Ova metoda se koristi svuda gde se dobijaju podaci sa klijenta, tj. na samom početku REST metoda kontrolera za obradu ovakvih podataka.

**2) Broken authentication and session management**

Naša aplikacija ne koristi sesiju niti bilo kakav oblik logovanja, tj. autentifikacije.

Jedina situacija gde se šalju poverljivi podaci je prilikom korišćenja PayPal-a. Međutim, PayPal ima svoje bezbedonosne mehanzime koji su pouzdani.

Implementirana web aplikacija za komunikaciju između svog frontend-a i backend-a koristi HTTPS protokol tj. TLS čime je obezbeđena tajnost podataka koji se razmenjuju na datoj relaciji.

**1) Injection**.

U aplikciji je odrađena zaštita validacijom na frontend strani prilikom svakog unosa bilo koje forme. Na server je upotrebom biblioteke Jsoup-1.10.1.jar i koričćenjem izraza oblika String cleanPostPayload = Jsoup.*clean*(postPayload, Whitelist.*basic*()); gde je cleanPostPayload prešićeni string iz promenjive postPayload. Ova metoda se poziva na početku svakog kontrolera koji prima neke podatke iz formi. Iz stringa će biti uklonjeni svi elementi koji mogu izazvati injection.