**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 primer01.js.

Korisnike treba obrazovati i dati im do znanja da ne treba da klikću na neproverene linkove koji im stižu u e-mailu ili u poruci kako ne bi na apsurdan način doveli našu aplikaciju u stanje u kome mora da se brani od napadača. Seminare i slične skupove, koji bi obrazovali naše korisnike, treba organizovati što češće kako bi korisnici bili obavešteni o najnovijim informacijama koji se tiču bezbednosti na internetu, kao i iz razloga što korisnici koji nisu informatički najobrazovaniji moraju se podsećati šta ne bi trebali da rade inače će u jednom trenutku svakako napraviti grešku.

**9. Using components with known vulnerabilities**

Na osnovu analize **OWASP dependency-check** ustanovljeno je da naša aplikacija ima nekoliko problema.

* Prvi problem je bio sa httpclient-4.1.1.jar fajlom koji je imao propust da ignoriše podešavanja http.socket.timeout prilikom rukovanja sa SSL-om i dozvoljava napadačima da izvrše Denial of Servise (DoS) napad preko neimenovanih vektora. Ovaj problem je rešen u novijoj verziji httpclient jar fajla koji smo i mi ubacili u našu aplikaciju. Koristimo httpclient-4.5.2.jar fajl koji je zahtevao i ubacivanje novijeg httpcore jar fajla tako da smo ubacili i httpcore-4.4.4.jar fajl.
* Drugi problem se odnosio isto na httpclient-4.1.1.jar fajl koji nije na ispravan način proveravao da li hostname servera odgovara domen imenu subjekta iz X.509 sertifikata i omogućavalo je napadaču koji prisluškuje saobraćaj da vara SSL servera preko CN stringa. Problem je rešen na identičan način kao i prethodni iz razloga što je ovaj propust popravljen u novijoj verziji jar fajla.
* Treći problem se odnosio na jersey-apache-client4-1.17.jar i imao je dva propusta identična onim koji su imali propusti httpclient jar fajla. Ovaj problem je rešen uz pomoć httpclient-4.5.2.jar fajla koji pokriva ovaj propust kao i ubacivanjem novije verzije jersey-apache-client4-1.19.1.jar fajla.
* Četvrti problem je bio propust koji je imao xalan-2.7.1.jar fajl.Problem je bio sa TransformerFactory koji je na nepravilan ograničavao pristup određenim osobinama kada je omogućen FEATURE\_SECURE\_PROCESSING i na taj način dozvoljavao napadačima da zaobiđu očekivana ograničenja i opterećenja proizvoljne klase ili pristup spoljašnjim resursima uz pomoć :
  + Crafred
  + Xalan:content-header
  + Xalan:entities
  + Xslt:content-header
  + Xslt:entites
  + Java osobinama koje su vezane za XSLT 1.0

Ovaj problem je rešen dodavanjem novije verzije xalan-2.7.2.jar fajla u kojem je ovaj propust rešen.

**8. Cross-site request forgery**

Korisnik prilikom logovanja na aplikaciju dobija svoj JWT (JSON Web Token) koji se smešta u Local Storage korisnikovog pretraživača. Prilikom sledećeg pristupa aplikaciji korisnik šalje njegov JWT i na osnovu njega se proverava da li ima pristup nekom od resursa aplikacije, ako nema JWT da pošalje odbija se njegov zahtev i šalje se na login stranicu aplikacije. Kao dodatna zaštita implementiran je i HTTPS mehanizam koji omogućava kriptovanje zahteva i smanjuje mogućnost da se ukrade JWT. Ako se korisnik izloguje sa sistema njemu se briše JWT iz Local Storage i ukoliko ponovo pokuša da pristupi aplikaciji moraće ponovo da se uloguje kako bi dobio novi JWT koji se generiše samo za njega. JWT token se pravi u UserController klasi u metodi checkUser.

Problem kod ovoga je što ako neko drugi koristi korisnikov računar može da pristupi i tokenu ukoliko se korisnik nije izlogovao sa aplikacije i time izbrisao i token iz svog pretraživača. 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**

Odrađena je autorizacija bazirana na RBAC standardu, na osnovu tri korisničke grupe (odbornik, građanin, predsednik skupštine). Na početku svake metode kontrolera se proverava da li korisnik ima permisiju za obavljanje te metode.

**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. Što se tiče lozinki koji se čuvaju u bazi i dodatno su zaštićene metodom hash and salt. Heširanje je odrađeno u klasi UserServiceImpl u metodama hashPassword i generateSalt.

Formirani dokumenti koji se čuvaju u bazi, čuvaju se potpisani i enkriptovani, kao još jedan vid rešenja na dati sigurnosni propust. Za to su korišćeni jaki i noviji kriptografski algoritmi. Ovo je implementirano u aplikaciji IstorijskiArhiv u metodama koje se nalaze u IasgnsServiceImpl klasi.

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**

Postoje dva načina od ove vrste napada. Prvi je da se objekti indirektno referenciraju, a drugi da se objekti direktno referenciraju uz proveru prava pristupa. Naš način odbrane od ovog napada jeste taj da prilikom svakog pristupa određenom resursu, proveravamo da li je korisnik ulogovan i koja su njegova prava pristupa (dakle, drugi način).

U datoj aplikaciji, ovo je rešeno tako što je ulogovanom korisniku prikazano samo ono što je njemu predviđeno. Na primer, ulogovan građanin ne može uneti novi propis već može samo vršiti pretragu postojećih propisa, kao i pregled istih, odbornik ne može prihvatati akte itd. Naravno, prava pristupa se proveravaju prilikom svake akcije. Ukoliko ulogovani građanin pokuša da preko URL-a ode na određenu stranicu na kojoj nema prava pristupa, biće redirektovan.

**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(login, registracija, pretraga, unos novog propisa, amandmana…), tj. na samom pocetku REST metoda kontrolera za obradu ovakvih podataka.

**2) Broken authentication and session management**

Korisnici na sistem se prijavljuju upotrebom korisničkog imena tj. email-a i lozinke koje su uneli prilikom registracije. Kako bi se sprečilo korišćenje jednostavnih lozinki, korisnik pri registraciji mora da unese niz od najmanje osam karaktera. Takođe mora uneti bar jedno veliko slovo i bar jednu cifru. Prilikom registrovanja, korisnik mora dva puta uneti lozinku kako bi potvrdio da želi baš tu lozinku. Ovo je implementirano u kontroleru UserController u metodi registerUser.

Lozinke korisnika se čuvaju u XML bazi pri čemu su one transformisane upotrebom hash i salt metoda.

Implementirana web aplikacija za komunikaciju izmdu svog frontend-a i backend-a koristi HTTPS protokol tj. TLS čime je obezbeđena tajnost podataka koji se razmenjuju na datoj relaciji. Time je obezbeđena sigurnost korisničkog imena i lozinke koji se prenose prilikom registrovanja i prijavljivanja na sistem.

Radi identifikovanja prijavljenog korisnika se ne koristi sesija već JWT (Java Web Token) mehanizam. Time su izbegnuti problemi koje sesija za sobom nosi (upotrebom XSS-a se može ukrasti sesija). Token koji browser dobija od servera se čuva u samom browser-u (Local Storage). Taj token se prilikom odjavljivanja korisnika briše.

**1) Injection**

Opasnost od injection napada prilikom pretrage po metapodacima (datumu kreiranja i datumu usvajanja) je odklonjena pošto se na serverskoj strani vrši provera da li uneti podaci za datum kreiranja formiraju datum i da li uneti podaci za datum usvajanja formiraju validan datum.

Prilikom popunjavanja formi za kreiranje akta ili amandmana veoma lako može doći do injection napada, ako se naravno ne izvrši zaštita. Problem pretstavlja mogućnost korisnika da unese neke xml tagove i time promeni strukturu rezultujućeg dogumenta. U te tagove može ubaciti neku skriptu koja će se izvršiti prilikom preuzimanja (od strane korisnika) odgovarajućeg dokumenta iz baze podataka. Stoga bi XML dokumenat bilo potrebno validirati. Jedan od načina je uz pomoć odgovarajuće XML šeme pre slanja u bazu podataka. Međutim u aplikciji je implementirano upotrebom biblioteke Jsoup-1.9.2.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.