OWASP Top Ten gap analiza

***A1 – Injection***

Injection propusti nastaju kada aplikacija šalje neproverene podatke interpreteru. Podaci koje napadač šalje mogu dovesti do toga da interpreter izvrši neželjene komande ili da napadač pristupi podacima bez odgovarajuće autorizacije. SQL Injection (SQLi) predstavlja jednu vrstu injection napada gde napadač “ubrizgava” malicioznu SQL naredbu, koja direktno radi sa DB serverom web aplikacije. Može da dovede do gubitka I integriteta podataka, DOS-a, a ponekad I do preuzimanja čitavog server.

Kako smo mi rešili: Iskoristili smo biblioteku (<https://github.com/rkpunjal/sql-injection-safe>) koja vrši validaciju unesenih vrednosti atributa, prilikom pokušaja izmene ili kreiranja novog entiteta u bazi. Intercepter proverava da li unesena vrednost atributa sadrži neku od od predefinisanih SQL naredbi. Ako da, server ne omogućava da se novi entitet sačuva/izmeni u bazi podataka.

Kako bi trebalo rešiti/dopuniti rešenje: Koristiti prepared statement-e. Preporučuje se i OWASP-ov API (https://github.com/ESAPI/esapi-java-legacy) koji pruža mnoge rutine u rešavanju ovog problema.

***A2 – Neadekvatna autentifikacija i upravljanje sesijom***

Izrada mehanizma autentifikacije I upravljanja sesijom nije jednostavna, stoga se poprilično i zapostavlja (na primer kod logovanja, session timeout-a, remember me..). Ako su funkcije vezane za autentifikaciju i upravljanje sesijom nekorektno implementirane, napadač može kompromitovati lozinke, ključeve ili tokene sesije, ili iskoristiti druge implementacione mane u cilju preuzimanja identiteta korisnika privremeno ili trajno. Samim tim, napadač ima mogućnost da radi sve što bi I žrtva mogla (zato su nalozi privilegovanih korisnika česta meta napada)

Kako bi trebalo rešiti/dopuniti rešenje: pogledati OWASP-ov standard Application Security Verification, koji propisuje kako se rizici iz područja autentifikacije I upravljanja sesijom smanjuju. Dobar primer je OWASP-ov API (ESAPI Authenticator) koji se smatra dobrim primerom koji bi trebalo koristiti. Koristiti dobro iskonfigurisan HTTPS.

***A3 – Cross-Site-Scripting (XSS)***

XSS je tip computer security slabosti koje se pronalazi kod web aplikacija. XSS napad omogućava da napadač ubaci JavaScript kod na web stranicu koju ujedno posmatraju I ostali koristici iste aplikacije. Postoje dve vrste XSS propusta: jedni se pojavljuju na serverskoj a drugi na klijentskoj strain. Ovim napadom, napadači mogu da izvrše maliciozne skripte u žrtvinom web čitatu, I tako preuzmu korisničku sesiju, preusmere korisnika na maliciozan sajt, umetnu neželjen sadržaj u web čitatač…

Kako smo mi rešili: iskoristili smo Hibernate-ov validator na serverskoj strain (SafeHtml koji ide zajedno sa Javinim HTML parserom <https://jsoup.org/>). Ovakva validacija proverava da li pokusavamo da u bazu podataka ubacimo neki HTML sadržaj. Ukoliko to validator prepozna, prijavljuje grešku I ne dozvoljava da se željeni entitet sačuva/izmeni u bazi.

Kako bi trebalo rešiti/dopuniti rešenje: Ovakva Whitelist-a na serverskoj strain nije kompletna zaštita, jer u nekim aplikacijama će biti omogućeno unošenje specijalnih karaktera poput < I >. Potrebno je I na klijentskoj strani omogućiti pravilno escape-ovanje svih nepoverljivih podataka baziranih na HTML kontekstu. OWASP-ov AntiSamy ili Java HTML Sanitizer Project predstavljaju poprilično dobre biblioteke u sprečavanju ovakvih napada.

***A4 – Neadekvatna kontrola pristupa***

U aplikacijama se za podatle vrlo često koriste stvarna imena ili ključevi objekata, dok se za funkcije često koriste nazivi funkcija I URL-ovi. Zbog ustaljenosti ovakvih propusta, funkcijama I podatcima se može lako pristupiti, tj mogu se lako pogoditi. Pripusti se dešavaju prilikom izrade projekta, gde aplikacije I API-ji ne verifikuju uvek da li korisnik ima dozvolu da pristupi željenom resursu, što dovodi do problema u kontroli pristupa. Napadači mogu iskoristiti sve ove prostupe, I samim tim mogu pristupiti resursima za koje nisu ovlašćeni (kao što su izmena prava pristupa, modifikovanje korisničkih naloga..).

Kako smo mi rešili: putem API-ja Spring Boot Security-a kao I implementacijom kontrole pristupa RBAC smanjili smo mogućnost napada na naš sistem. SpringBootSecurity čuva dokumente I od neautentifikovanog posetioca, dok RBAC reguliše koja uloga ima pristup kojoj funkcionalnosti u okviru koje tabele.

Kako bi trebalo rešiti/dopuniti rešenje: Koristiti indirektne reference na objekte po korisniku ili sesiji. OWASP-ov ESAPI uključuje i sekvencijalne i random acces mape referenci koje programeri mogu koristiti da eliminišu direktne reference na objekte.

***A5 – Neadekvatna bezbednosna konfiguracija***

Greške u bezbednosnoj konfiguraciji se mogu desiti na bilo kom nivou aplikacije, uključujući i platformu, web server, aplikativni server, bazu, framework, kao i kod. Automatizovani skeneri su korisni za detekciju delova koji nedostaju, neadekvatnih konfiguracija, korišćenja standardnih naloga, nepotrebnih servisa itd. Sigurna konfiguracija treba da bude definisana, implementirana i održavana jer standardna konfiguracija često nije sigurna. Osim toga, softver treba i ažurirati. Ukoliko bi došlo do uspešnog napada, čitav sistem bi mogao u potpunosti biti kompromitovan bez ikakvog znanja programera ili administrator o tome.

Kako bi trebalo rešiti: arhitektura aplikacije da bude takva da omogućava efikasnu I sigurnu separaciju između komponenti. Ponavljani proces hardening koji omogućava brzo I lako deploy-ovanje novog pravilno iskonfigurisanog okruženja. Okruženja za razvoj I produkciju treba da budu identično konfigurisana a process da bude automatizovan. Periodično pokretati skeniranja I revizije za detekciju propusta.

***A6 – Okrivanje osetljivih podataka***

Vrlo česta greška koja se javlja podrazumeva da se osetljivi podaci ne enkriptuju. Prilikom enkripcije, obično se generišu slabi ključevi, neadekvatno se upravlja istim, kao I upotreba slabih algoritama dovodi do velikih problema. Maliciozni napadači često koriste web aplikacije i API-je koji ne štite pravilno osetljive podatke koje mogu ukrasti I modifikovati. Osetljivi podaci (lični podaci, bankovski računi, brojevi kreditnih kartica..) zahtevaju dodatnu zaštitu, kao što je enkriptovanje prilikom skladištenja i transporta.

Kako smo mi rešili: Hashiranje password-a jBCrypt-om. Enkriptovanje jwtTokena(koji služi za komunikaciju klijent-server) HS512 algoritmom.

Kako bi trebalo rešiti/dopuniti rešenje: treba enktiptovati sve osetljive podatke u našem skladištu I prilikom transporta (npr. broj I stanje računa u banci predućeza). Oni podaci koji nisu od krucijalnog značaja, tj da se koriste svakognevno, ne trebaju se čuvati u sistemu. Obezbediti upotrebu jakih standardnih algoritama (prateći dobru praksu), kao I adekvatno upravljanje ključevima.

***A7 – Nedovoljna zaštita od napada***

Aplikacije i API-ji su izloženi napadima sve vreme. Većina aplikacija i API-ja detektuje pogrešan unos i jednostavno ga odbacuju, dozvoljavajući napadaču da iznova i iznova napada. Ovakve napade sprovodi zlonamerni korisnik koji pokušava da eksploatiše ranjivost Sistema. Otkrivanje i blokiranje manuelnih i automatizovanih napada je jedan od najefektnijih načina za povećanje niva bezbednosti. Nije dovoljno samo obezvediti osnovnu validaciju ulaznih podataka, već je potrebno obezbediti I automatsko detektovanje, logovanje, reagovanje na učestale pokušaje eksploatacije resursa.

Kako smo mi rešili: Logovi koji će beležiti sve bitne komunikacije korisnika sa sistemom, poput prijave I odjave na sistem, formiranje, modifikacija I brisanje narudzbenica I faktura. Pored identifikatora korisnika koji je izazvao odabranu akciju, beleži se I akcija I vreme kad je izvršena akcija.

Kako bi trebalo rešiti/dopuniti rešenje: Automatski obrađivati zahteve ka sistemu, i obezbediti onemogućavanje ili monitoring korisničkih naloga koji se neadekvatno ponašaju.

***A8 – Cross-Site Request Forgery***

CSRF predstavlja maliciozni napad koji primorava web čitač ulogovane žrtve da pošalje falsifikovan HTTP zahtev ranjivoj web aplikaciji, uključujući i žrtvin cookie iz sesije, kao i sve ostale automatski uključene podatke o autentifikaciji (I takav zahtev će aplikacija smatrati legitimnim). CSRF koristi prednost činjenice da većina web aplikacija dozvoljava napadačima da predvide sve detalje određene akcije. Tako, napadač može da izvrši bilo koju funkcionalnost za koju žrtva ima ovlašćenje.

Napadači mogu prevariti žrtve da obave bilo koju izmenu koju je žrtva ovlašćena da izvrši, npr. ažuriranje podataka o nalogu, online kupovinu, logout, pa čak i login.

Kako smo mi rešili: Generisanje jwtTokena (https://github.com/jwtk/jjwt) prilikom prijave na sistem, koji se šalje sa svakim zahtevom (u header-u), I proverava se pred svaku akciju da li postoji prijavljen na sistem korisnik sa kredencijalima koji se mogu izvući iz tokena na serverskoj strani. Takođe, token ima I vremenski rok važenja.

Kako bi trebalo rešiti/dopuniti rešenje: koristiti CSRF Guard ili OWASP-ov ESAPI API koji uključuju metode koje programeri mogu da koriste da bi sprečili CSRF ranjivost. Još jedan vid zaštite predstavlja CAPTCHA mehanizam, koji zahteva od korisnika da se prilikom spavkog zahteva ponovo autentifikuje I dokaže da je korisnik.

***A9 – Korišćenje komponenti sa poznatim ranjivostima***

Neke komponente koje koriste aplikacije kao što su framework ili biblioteke pokreću se sa istim privilegijama kao i sama aplikacija. Te komponente često imaju bezbednosne propuste, pa ako se ranjivost komponente iskoristi, takav napad može olakšati gubljenje podataka ili preuzimanje servera. Većina aplikacija ima ove probleme, jer se timovi ne fokusiraju da komponente koje koriste aplikacije butu ažurne.

Kako smo mi rešili: Umesto ubacivanja u naš projekat nekih gotovih komponenti, koristili smo ručno napisane komponente. Koristili smo najnovije verzije biblioteka, koje otklanjaju bezbednosne propuste ranijih verzija.

Kako bi trebalo rešiti/dopuniti rešenje: Iskoristiti OWASP-ov API dependency check da se proveri da li neka od korišćenih zavisnosti ima neku slabost. Gde god je to moguće, razmisliti o dodavanju sigurnosnih „wrapper-a“ oko komponenti kako bi se onemogućile neiskorišćene funkcionalnosti i/ili osigurali slabi ili ugroženi aspekti komponenti.

***A10 – Nezaštićeni API-ji***

Savremene web aplikacije sve češće uključuju bogate klijentske aplikacije i API-je, kao što je JavaScript u web čitačima i mobilnim aplikacijama, koji se na bekendu povezuju sa nekom vrstom API-ja (SOAP/XML, REST/JSON, RPC, GWT itd.). API-ji (mikroservisi, servisi, endpoint-ovi) mogu biti podložni čitavom spektru napada. Ukoliko uspešno dođe do napada, može doći do krađe podataka, njihove izmene i uništavanje, neovlaščeni pristup čitavoj aplikaciji i kompletno preuzimanje servera.

Kako bi trebalo rešiti: Osigurati da postoji sigurna komunikacija između klijenta i API-ja (da su svi kredencijali ključevi I tokeni osigurni). Implementirati šemu za kontrolu pristupa koja štiti API od neispravnih poziva, uključujući i neovlašćene funkcije i reference podataka.