OWASP Top Ten gap analiza

***A1 – Injection***

Injection propusti nastaju kada aplikacija šalje neproverene podatke interpreteru. Injection propusti su veoma rasprostranjeni, posebno u “legacy” aplikacijama i zastarelom kodu. Često se mogu pronaći u SQL, LDAP, Xpath ili NoSQL upitima, OS komandama, XML parserima, SMTP hederima, argumentima programa itd. Ovi propusti su jednostavni za otkrivanje prilikom analiziranja koda, ali ih je često teško otkriti putem testiranja. Skeneri i fuzzer-i mogu pomoći napadačima da otkriju injection propuste. Podaci koje napadač šalje mogu dovesti do toga da interpreter izvrši neželjene komande ili da napadač pristupi podacima bez odgovarajuće autorizacije.

*Tehnički uticaj napada*: ozbiljan. Injection napad može dovesti do gubitka integriteta podataka ili samih podataka, uskraćivanja pristupa (denial of access), a ponekad i do preuzimanja čitavog servera.

*Sprečavanje propusta:* zahteva separaciju nepoverljivih podataka od komandi i upita.

* Najpoželjnija opcija je korišćenje sigurnog API-ja koji izbegava potpunu upotrebu interpretera ili obezbeđuje parametrizovan interfejs. Ipak, treba biti oprezan sa API-jima, kao što su uskladištene procedure, koje su parametrizovane, ali i dalje mogu uvesti injekciju ispod haube.
* Ako parametrizovan API nije dostupan, treba pažljivo escape-ovati specijalne karaktere koristeći specifičnu escape sintaksu za dati interpreter. OWASP-ov ESAPI pruža mnoge od tih rutina.
* Pozitivna ili “white list” validacija ulaznih podataka se takođe preporučuje, ali nije potpuna odbrana s obzirom na to da mnoge aplikacije zahtevaju omogućavanje unosa specijalnih karaktera. Ako je potrebno unošenje specijalnih karaktera, samo gornja dva pristupa će obezbediti njihovu bezbednu upotrebu. OWASP-ov ESAPI ima proširivu biblioteku rutina za white list validaciju ulaznih podataka.

*Naša implementacija* koda koji štiti od SQL injection napada podrazumeva korišćenje “find” metoda visokog nivoa koje su preporučene za play framework.

*Link:* <https://www.playframework.com/documentation/1.3.x/security#sql>

*Link ka kodu (može se pogledati i bilo koja druga klasa kontrolera):* <https://github.com/impedansa/Poslovna-banka/blob/master/app/controllers/Zaposlenii.java>

Takođe, prilikom validiranja pristiglih XML dokumenata koristiće se XML šema.

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

Programeri često izrađuju autentifikaciju i upravljanje sesijom koje je prilagođeno potrebama projekta na kom rade, ali njihova ispravna izrada nije jednostavna. Kao rezultat toga, ove prilagođene šeme često imaju nedostatke u oblastima kao što su logovanje, upravljanje lozinkom, timeout-ovi, „remember me“, tajna pitanja, unapređivanje naloga itd. Pronalaženje takvih grešaka može biti teško jer je svaka implementacija jedinstvena. 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.

*Tehnički uticaj napada*: ozbiljan. Takvi propusti mogu dovesti do toga da neki ili čak svi nalozi budu napadnuti. Kada je napad uspešan, napadač može da radi sve što bi i žrtva mogla. Privilegovani nalozi su često meta napada.

*Sprečavanje propusta:* Primarna preporuka za organizaciju je da programerima obezbedi:

* Jedan set jakih ko *Sprečavanje napada* ntrola za autentifikaciju i upravljanje sesijom. Takve kontrole treba da teže sledećem:
* Da ispunjavaju sve zahteve vezane za autentifikaciju i upravljanje sesijom definisane u OWASP-ovom Application Security Verification standardu (ASVS) iz područja V2 (autentifikacija) i V3 (upravljanje sesijom).
* Da imaju jednostavan interfejs za programere. ESAPI Authenticator i User API se smatraju dobrim primerima koje bi trebalo emulirati, koristiti ili nadograđivati.

Veliki napori takođe treba da budu uloženi u izbegavanje XSS napada koji mogu biti iskorišćeni za krađu ID-ja korisničke sesije.

*Naša implementacija* autentifikacije je izvršena ugradnjom play secure modula u aplikaciju, koji se i preporučuje za korisnike ovog framework-a, dok je upravljanje sesijom izvršeno preko session objekta koji framework takođe pruža i koji je implementiran na način da ne dozvoljava falsifikovanje i krađu korisničke sesije od strane bilo kakvih third-party napadača.

*Linkovi:* <https://www.playframework.com/documentation/1.3.x/secure>

<https://www.playframework.com/documentation/1.3.x/security#sessions>

*Link ka kodu:* <https://github.com/impedansa/Poslovna-banka/blob/master/app/controllers/Security.java>

***A3 – Cross-Site-Scripting (XSS)***

XSS je najčešći bezbednosni propust u web aplikacijama. XSS propusti nastaju kada aplikacija uključuje neproverene podatke u novoj web stranici bez njihove prethodne adekvatne validacije ili escape-ovanja, ili update-uje postojeću stranicu sa korisničkim podacima koristeći API web čitača koji može generisati JavaScript. Postoje dve vrste XSS propusta: uskladišteni i reflektovani, i svaki od njih se može pojaviti na serverskoj i/ili klijentskoj strani. Detekcija većine serverskih XSS propusta je prilično jednostavna putem testiranja ili analize koda, dok je klijentske XSS propuste veoma teško identifikovati.

*Tehnički uticaj napada*: umeren. Napadači mogu da izvrše skripte u žrtvinom web čitaču za preuzimanje korisničke sesije, uništavanje web sajta, umetanje neprijateljskog sadržaja, preusmeravanje korisnika na maliciozan sajt, preuzimanje korisnikovog čitača upotrebom malware-a itd.

*Sprečavanje propusta:* zahteva separaciju nepoverljivih podataka od aktivnog sadržaja pretraživača.

* Najpoželjnija opcija je pravilno escape-ovanje svih nepoverljivih podataka baziranih na HTML kontekstu (body, atributi, JavaScript, CSS ili URL) u koji će podaci biti ubačeni. Detalji traženih tehnika escape-ovanja podataka mogu se naći na OWASP XSS Prevention Cheat Sheat.
* Pozitivna ili „whitelist“ input validacija sa serverske strane se takođe preporučuje jer pomaže u zaštiti od XSS-a, ali nije kompletna zaštita jer mnoge aplikacije zahtevaju mogućnost unošenja specijalnih karaktera. Takva validacija bi trebalo da, koliko god je to moguće, validira dužinu, karaktere, format i pravila poslovanja nad podacima pre prihvatanja ulaza.
* Za bogat sadržaj, treba razmisliti o upotrebi „auto-sanitization“ biblioteka kao što je OWASP-ova AntiSamy ili Java HTML Sanitizer Project.
* Uzeti u obzir mogućnost upotrebe Content Security Policy (CSP) za odbranu od XSS napada na celom sajtu.

*Naša implementacija* zaštite od ovog napada podrazumeva upotrebu samog play framework-a čiji template engine automatski escape-uje stringove.

*Link:* <https://www.playframework.com/documentation/1.3.x/security#xss>

***A4 – Neadekvatna kontrola pristupa***

Za podatke, aplikacije i API-ji često koriste stvarno ime ili ključ objekta prilikom generisanja web stranica. Za funkcije, URL-ovi i nazivi funkcija često lako mogu da se pogode. Aplikacije i API-ji ne verifikuju uvek da li je korisnik ovlašćen da pristupi ciljanom resursu, što dovodi do mane u kontroli pristupa. Testeri mogu lako manipulisati parametrima da detektuju takve mane. Analiza koda brzo otkriva da li je autorizacija korektna.

*Tehnički uticaj napada:* umeren. Takvi propusti mogu kompromitovati sve funkcionalnosti ili dostupne podatke. Napadači mogu iskoristiti ove propuste za pristup neovlašćenim funkcionalnostima i/ili podacima, kao što je pristup nalozima korisnika, pregled osetljivih podataka, modifikovanje korisničkih naloga, izmena prava pristupa, krađa ili zloupotreba funkcionalnosti i podataka itd.

*Sprečavanje propusta:* zahteva odabir pristupa za zaštitu svake funkcije i svakog tipa podataka (npr. broj objekta, naziv fajla).

* Proveriti pristup. Svaka upotreba direktne reference od strane nepoverljivog izvora mora uključivati kontrolu pristupa kako bi se osiguralo da je korisnik ovlašćen da pristupi traženom resursu.
* Koristiti indirektne reference na objekte po korisniku ili po sesiji. Ovaj obrazac kodiranja sprečava napadače da direktno ciljaju neovlašćene resurse. OWASP-ov ESAPI uključuje i sekvencijalne i random acces mape referenci koje programeri mogu koristiti da eliminišu direktne reference na objekte.
* Automatska verifikacija. Snažna automatizacija za verifikaciju ispravnog razvoja autorizacije. Često je prilagođena pojedinačnim aplikacijama.

*Naša implementacija* kontrole pristupa rađena je po RBAC modelu upotrebom deadbolt mehanizma za autorizaciju, ugradnjom deadbolt modula u aplikaciju i njegovim korišćenjem.

*Link:* <https://www.playframework.com/modules/deadbolt-1.5.4/home>

*Link ka kodu*: <https://github.com/impedansa/Poslovna-banka/blob/master/app/controllers/MyDeadboltHandler.java>

***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. Programeri i administratori sistema treba da rade zajedno kako bi se osiguralo da je cela aplikacija adekvatno konfigurisana. 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.

*Tehnički uticaj napada:* umeren. Sistem bi mogao u potpunosti biti kompromitovan bez ikakvog znanja programera i administratora o tome. Svi podaci mogu biti ukradeni ili modifikovani tokom vremena. Troškovi za oporavak mogu biti veliki.

*Sprečavanje propusta:* primarne preporuke obuhvataju obezbeđivanje svega od navedenog:

* Ponavljani proces jačanja sistema („hardening“) koje omogućava brzo i lako deploy-ovanje novog okruženja koje je pravilno iskonfigurisano. Okruženja za razvoj, QA i produkciju treba da budu identično konfigurisana (sa različitim lozinkama za svako okruženje). Ovaj proces treba da bude automatizovan da bi se smanjio napor neophodan za podešavanje novog bezbednog okruženja.
* Proces koji ide u korak sa novim softverskim unapređenjima i deploy-uje ih blagovremeno u svakom okruženju. Ovo treba da uključuje i sve biblioteke.
* Snažna arhitektura aplikacije koja omogućava efikasnu i sigurnu separaciju između komponenti.
* Uzeti u obzir periodično pokretanje skeniranja i revizije kao pomoć za detekciju budućih propusta u konfigurisanju.

Sama konfiguracija prilikom postavljanja našeg sistema u produkciju trebalo bi da isprati sve parametre navedene na sledećim linkovima: <https://www.playframework.com/documentation/1.2.7/configuration>

<https://www.playframework.com/documentation/1.3.x/production>

***A6 – Okrivanje osetljivih podataka***

Najčešća greška podrazumeva da se osetljivi podaci ne enkriptuju. Kada se koristi enkripcija, generisanje slabih ključeva, neadekvatno upravljanje ključevima i upotreba slabog algoritma su česte pojave, posebno korišćenje slabih tehnika za heširanje lozinki. Slabosti web čitača su veoma česte i jednostavne za otkrivanje, ali teške za eksploataciju u velikim razmerama. Eksterni napadači imaju poteškoća sa otkrivanjem propusta na serverskoj strani zbog ograničenog pristupa, a takođe ih je obično teško iskoristiti. Mnoge web aplikacije i API-ji ne štite pravilno osetljive podatke. Napadači mogu ukrasti ili modifikovati takve slabo zaštićene podatke. Osetljivi podaci zahtevaju dodatnu zaštitu, kao što je enkriptovanje prilikom skladištenja i transporta, kao i posebne mere predostrožnosti tokom razmene takvih podataka sa web čitačem.

*Tehnički uticaj napada:* ozbiljan. Često dolazi do kompromitovanja svih podataka koji je trebalo da budu zaštićeni. Tipično, ovi podaci uključuju osetljive podatke kao što su zdravstveni podaci, kredencijali, lični podaci, brojevi kreditnih kartica itd.

*Sprečavanje propusta:* Sve opasnosti od nesigurne kriptografije, upotrebe nebezbedne SSL konfiguracije i neadekvatne zaštite podataka izlaze izvan okvira Top 10. Zato, za sve osetljive podatke, treba primeniti sledeći minimum:

* Uzimajući u obzir pretnje od kojih planirate da zaštitite podatke, treba enkriptovati sve osetljive podatke u skladištu i prilikom transporta na način koji ih štiti od tih napada.
* Ne treba nepotrebno čuvati osetljive podatke. Treba ih ukloniti što pre je to moguće. Podaci koje ne posedujete ne mogu biti ukradeni. ☺
* Obezbediti upotrebu jakih standardnih algoritama i jakih ključeva, kao i adekvatno upravljanje ključevima. Razmisliti o upotrebi FIPS 140 proverenih kriptografskih modula.
* Osigurati da se lozinke kriptuju algoritmima specijalno namenjenim za zaštitu lozinki, kao što su bcrypt, PBKDF2 ili scrypt.
* Onemogućiti automatsko dovršavanje izraza na formama koje služe za prikupljanje osetljivih podataka i onemogućiti keširanje stranica koje sadrže osetljive podatke.

*Naša implementacija* podrazumeva šifrovanje podataka koje smo deklarisali kao osetljive, a oni obuhvataju podatke o klijentima i podatke o računima. Takođe, predviđeno je i šifrovanje podatka na koji račun se prebacuje novac prilikom zatvaranja računa, kao i podataka iz analitike izvoda poput toga ko šalje novac, kome se isti šalje, zajedno sa brojevima računa pošiljaoca i primaoca, ali će navedeni deo biti implementiran nakon što se podaci budu slali putem xml-a preko servisa jer su navedene tabele u aplikaciji read-only. Podaci su šifrovani AES algoritmom uz korišćenje najboljih praksi, a detaljnije objašnjenje o tome može se pročitati iz samih komentara u kodu.

Linkovi ka kodu: <https://github.com/impedansa/Poslovna-banka/blob/master/app/controllers/AES.java>

<https://github.com/impedansa/Poslovna-banka/blob/master/app/controllers/Klijenti.java> (enkriptovanje i dekriptovanje podataka o klijentima)

***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. Ovakvi napadi ukazuju na zlonamernog ili kompromitovanog korisnika koji pokušava da ispita i eksploatiše ranjivosti. Otkrivanje i blokiranje manuelnih i automatizovanih napada je jedan od najefektnijih načina za povećanje niva bezbednosti. Značajno pitanje je koliko brzo se može regulisati kritična slabost koja je upravo otkrivena. Zaštita od napada daleko prevazilazi osnovnu validaciju ulaznih podataka i uključuje automatsko detektovanje, logovanje, reagovanje, pa čak i blokiranje pokušaja eksploatacije resursa.

*Tehnički uticaj napada:* umeren. Najuspešniji napadi počinju sa ispitivanjem ranjivosti sistema. Dozvoljavanjem takvih proba da se nastave može podići verovatnoću uspešne eksploatacije do 100%. Spor deployment sigurnosnih patch-eva ide u korist napadača.

*Sprečavanje propusta:* Postoje tri osnovne stavke koje treba ispoštovati kako bi se obezbedila dovoljna zaštita od napada:

* Detektovanje napada: Da li se dogodilo nešto što je nemoguće da je legitiman korisnik izazvao (npr. input koji legitiman korisnik nije u stanju da generiše)? Da li se aplikacija koristi na način na koji je običan korisnik nikad ne bi koristio (npr. prebrz tempo, atipičan input, neobični šabloni upotrebe, ponavljani zahtevi)?
* Reagovanje na napade: Logovi i notifikacije su kritični za pravovremeno reagovanje. Treba odlučiti da li će se automatski blokirati zahtevi, IP adrese ili opseg IP adresa. Treba uzeti u obzir i onemogućavanje ili monitoring korisničkih naloga koji se neadekvatno ponašaju.
* Brzo „patch-ovanje“: ako proces ne može sam da iznese, treba deploy-ovati virtuelni patch koji analizira HTTP saobraćaj, protok podataka i/ili izvršenje koda i sprečava da slabost bude eksploatisana.

U našem sistemu su trenutno implementirani logovi koji beleže sa koje IP adrese stižu zahtevi, kao i svaku značajnu i kritičnu radnju koja se odvila u aplikaciji, uključujući i pokušaje neovlašćenog pristupa delovima aplikacije.

***A8 – Cross-Site Request Forgery***

CSRF napad 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. Takav napad omogućava napadaču da natera žrtvin web čitač da generiše zahteve koje će ranjiva aplikacija smatrati legitimnim zahtevima od žrtve. CSRF koristi prednost činjenice da većina web aplikacija dozvoljava napadačima da predvide sve detalje određene akcije. Zbog toga što web čitači automatski šalju kredencijale kao cookie-je sesije, napadači mogu da kreiraju maliciozne web stranice koje generišu falsifikovane zahteve koji se ne mogu razlikovati od onih legitimnih. Detektovanje CSRF propusta je poprilično jednostavno putem penetracionih testova ili analiziranja koda.

*Tehnički uticaj napada:* umeren. 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.

*Sprečavanje propusta:* uglavnom zahteva uključivanje nepredvidivog tokena u svaki HTTP zahtev. Takvi tokeni bi minimalno trebalo da budu jedinstveni za svaku korisničku sesiju.

* Najpreporučljivija opcija je uključivanje tokena u skriveno polje. To uzrokuje da se vrednost šalje u telu HTTP zahteva, izbegavajući njegovo uključivanje u URL.
* Jedinstveni token takođe može biti uključen u sam URL, ili u parametar URL-a. Međutim, takvo postavljanje poseduje veći rizik da će URL biti izložen napadaču, čime će se kompromitovati tajni token. OWASP-ov CSRF Guard može automatski uključiti takve tokene u Java EE, .NET ili PHP aplikacije. OWASP-ov ESAPI uključuje metode koje programeri mogu da koriste da bi sprečili CSRF ranjivosti.
* Zahtevanje od korisnika da se ponovo autentifikuje ili dokaže da je korisnik (npr. putem CAPTCHA) takođe može zaštititi od CSRF-a.

*Naša implementacija* zaštite od ovog napada podrazumeva upotrebu autentifikacionih tokena ugrađenih u play framework.

Link: <https://www.playframework.com/documentation/1.3.x/security#csrf>

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

Komponente kao što su biblioteke, framework i ostali softverski moduli pokreću se nad istim privilegijama kao i sama aplikacija. Ako se ranjiva komponenta iskoristi, takav napad može olakšati ozbiljno gubljenje podataka ili preuzimanje servera. Aplikacije i API-ji koji koriste komponente sa poznatim ranjivostima mogu ugroziti odbranu aplikacije i omogućiti različite napade. Praktično svaka aplikacija ima ove probleme jer se većina razvojnih timova ne fokusira na to da njihove komponente/biblioteke budu ažurirane. U većini slučajeva, programeri ni ne znaju koje sve komponente koriste, bez obzira na njihove verzije. Zavisnosti komponenti čine stvari još lošijim.

*Tehnički uticaj napada:* umeren. Moguće je postojanje čitavog spektra slabosti, uključujući injection, neadekvatnu kontrolu pristupa, XSS i slično. Uticaj može varirati od minimalnog, pa sve do potpunog preuzimanja servera i kompromitovanja podataka.

*Sprečavanje propusta:* Jedna od opcija je da ne koristimo komponente koje nismo sami napisali. Ali to i nije baš realistično. ☺ Za većinu komponenti se ne prave “patch-evi” ranjivosti za stare verzije. Umesto toga, većina najčešće reši problem u sledećoj verziji. Dakle, upgrade-ovanje do tih verzija je od kritičnog značaja. Softverski projekti treba da imaju proces koji će:

* Identifikovati sve komponente i sve korišćene verzije, uključujići i sve zavisnosti.
* Pratiti bezbednost ovih komponenti u javnim bazama podataka, mejling listama projekata i bezbednosnim mejling listama i održavati ih ažuriranim.
* Uspostaviti sigurnosne polise koje regulišu upotrebu komponenti, kao što je zahtevanje određenih praksi razvoja softvera, sprovođenje i prolaženje sigurnosnih testova i prihvatljive licence.
* 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. Nažalost, dinamički i ponekad čak i statički alati ne funkcionišu dobro nad API-jima i teško ih je analizirati manuelno, pa ove ranjivosti često ostanu neotkrivene.

*Tehnički uticaj napada:* umeren. Moguć je čitav spektar negativnih ishoda, uključujući krađu podataka, njihovu izmenu i uništavanje, neovlaščeni pristup čitavoj aplikaciji i kompletno preuzimanje servera.

*Sprečavanje propusta:* ključ kod zaštite API-ja je obezbediti da se u potpunosti razume model pretnji i koje odbrane postoje u sistemu:

* Osigurati da postoji sigurna komunikacija između klijenta i API-ja.
* Obezbediti postojanje jake autentifikacione šeme za API, kao i to da su svi kredencijali, ključevi i tokeni osigurani.
* Osigurati da je konfiguracija parsera očvršćena protiv napada koji god format podataka da zahtevi koriste.
* Implementirati šemu za kontrolu pristupa koja štiti API od neispravnih poziva, uključujući i neovlašćene funkcije i reference podataka.
* Zaštititi se od injection-a na svim formama jer su takvi napadi jednako održivi preko API-ja kao što su i kod normalnih aplikacija.