PENETRATION TESTING

U cilju identifikovanja i evaluacije ranjivosti našeg sistema, iz perspektive napadača, sproveli smo penetraciono testiranje.

Za izradu penetracionih testova koristili smo sledeće alate:

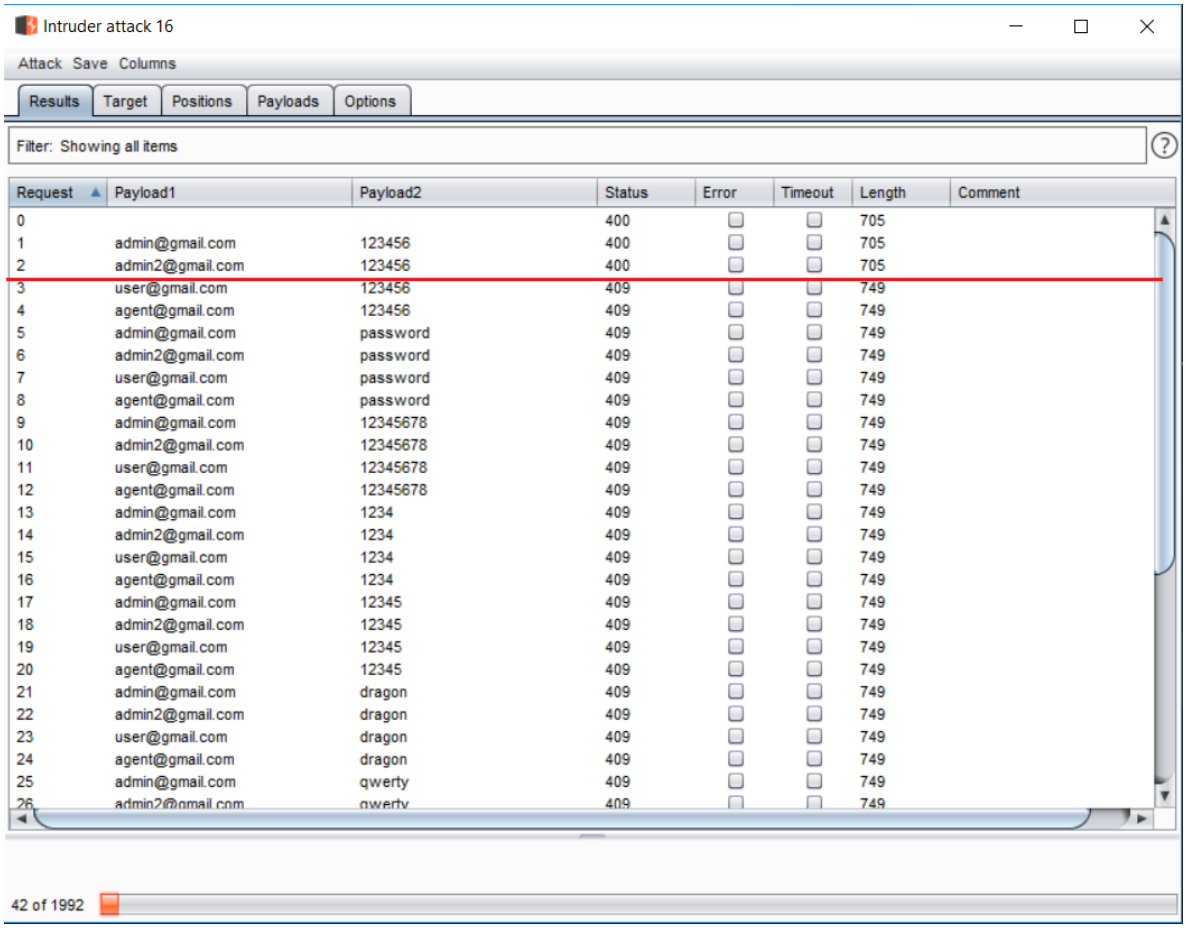
1. BURP SUITE za brute force napade kako bismo testirali broken authentication (2. na OWASP Top Ten listi)
2. SQLMAP kako bismo temeljno istestirali sql-injection napade (1. na OWASP Top Ten listi)
3. OWASP ZAP (u malim kolicinama) kako bismo testirali overall security aplikacije, mogućnost dobavljanja osetljivih podataka i razne druge napade koji spadaju u miscellaneous grupu napada.

Burp Suite smo, takođe, koristili u proxy-ju kako bismo presretali zahteve i unosili ih u sqlmap, kako bi on dalje testirao našu aplikaciju na tim mestima za SQL Injection (Koriscen je Mozzila kao browser i FoxyProxy koji nam je omogućio da nam Burp Suite zapravo bude presretač svih zahteva).

Burp Suite

Sa Burp Suite smo testirali Broken Authentication, tako što smo presretali svaki request koji bismo poslali iz Mozzila browsera preko FoxyProxy plugina. Te requestove bismo tada slali u Burp Suite Intrudera koji smo podešavali na Cluster Bomb tip napada koji nam je omogućavao da selektujemo tačno vrednosti koje želimo da menjamo u zavisnosti od liste inputa koje bismo mu kasnije prosledili. Tada smo odgovarajuće parametre menjali sa listama mogućnosti (konkretno smo primenjivali ovo na https://localhost:8443/auth/login jer nema smisla igde drugde da se primeni za broken authentication). Liste šifri smo našli na internetu kao top 500 worst passwords koje su do 7 karaktera, ne sadrže specijalne karaktere, što bi značilo - ili su samo brojevi ili samo slova. A username-ove , budući da su nam email-ovi, smo ručno ukucali 4,5 email-ova (inace bismo naravno stavljali vise radi boljeg testiranja).

Tada smo startovali brute force attack preko intrudera gde je pokusavao 4*500 razlicitih napada i kao sto se na slici (slika br1.) vidi nakon par 400 statusa (Bad Credentials) je presao na 409. To se desi jer smo Broken Authentication nakon 3 neuspesna logina sa iste IP-adrese zabranjivali i odmah u pocetku vracali HTTPStatus.CONFLICT (409)) i tada je ta IP adresa udaljena od logina na odredjeno vreme sto predstavlja dodatni sloj zastite za Broken Authentication.



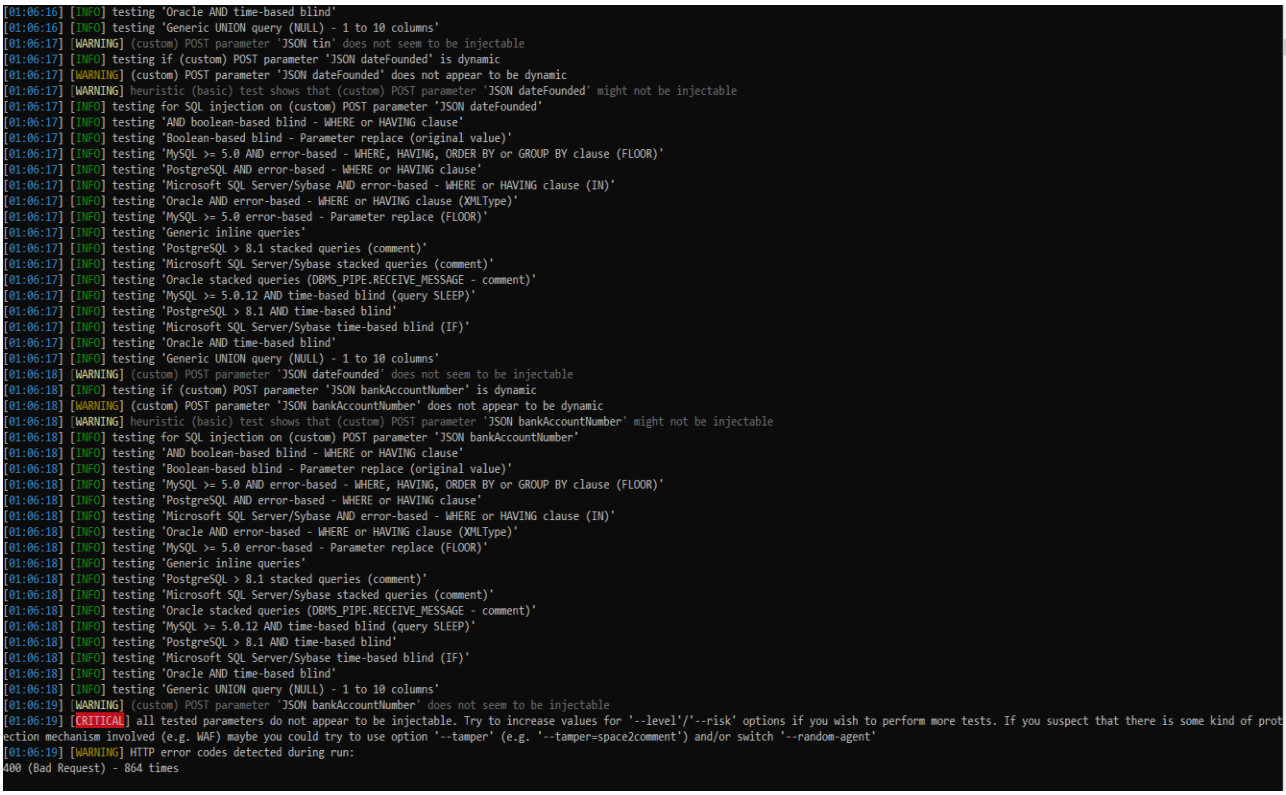
Slika 1. Testiranje brute-force napada

SQLMap

Koristeći SQLMap sledeće rute su testirane na SQLInjection napad:

- POST https://localhost:8443/auth/login
- POST https://localhost:8443/auth/create-simple-user
- POST https://localhost:8443/auth/create-agent
- GET https://localhost:8443/search/light?city=Novi Sad&from=13:20 2020-06-19&to=05:41 2020-06-24
- POST https://localhost:8443/ads
- POST https://localhost:8443/gearshift-types
- POST https://localhost:8443/car-brand
- POST https://localhost:8443/car-model
- POST https://localhost:8443/fuel-type
- POST https://localhost:8443/car-accessories
- POST https://localhost:8443/message

Ove rute su namerno izabrane, s obzirom da one sadrže najviše free texta koji korisnik može da unese. Takođe, podešavali smo da rizik bude mnogo veći od normalnog i level na mnogo veći od običnog (3 a default level i rizik su 1). Tada SQLMap pokusava svakakve vrste SQLINjection napada kako bi expose-ovao nasu aplikaciju. Međutim, čak I sa povećanim rizikom, naš sistem se pokazao bezbednim za ovu vrstu napada (slika 2)



Slika 2. Testranje SQLInjection napada

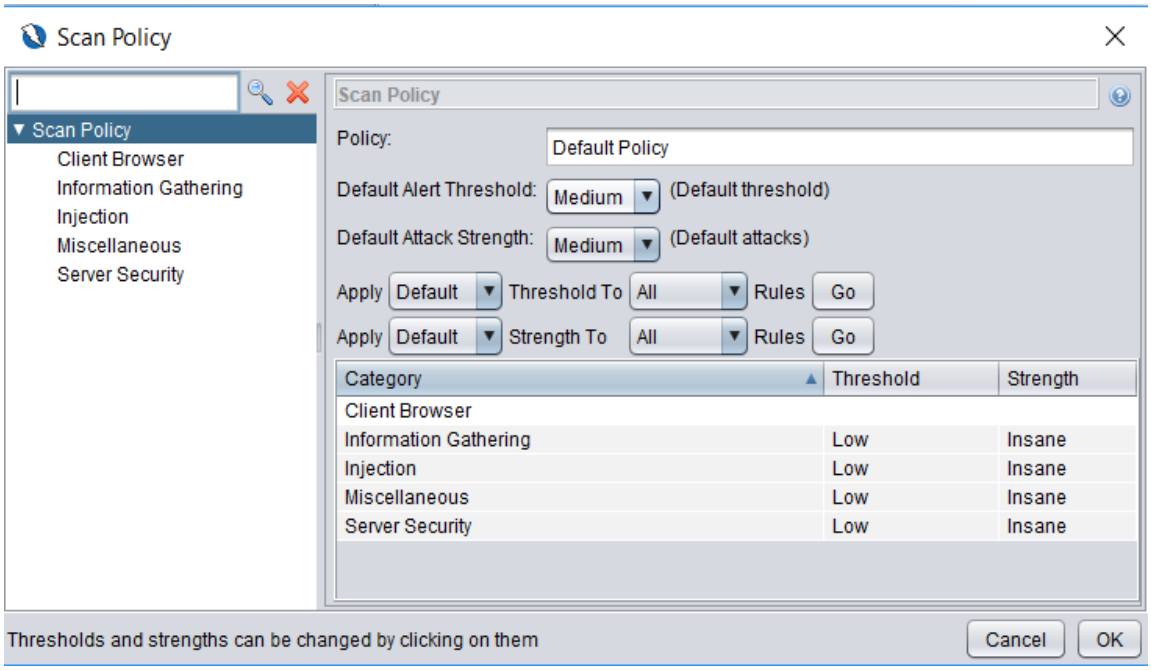
OWASP Zap

Za kraj, korišćen je OWASP Zap alat, koji je obavljao active scan nad određenim scope-om aplikacije. Active scan podrazumeva da, u zavisnosti od podešavanja, pustimo ZAP da pokuša sve forme napada na našu aplikaciju .Naravno, to nije preporučljivo ukoliko je aplikacija već u produkciji, što srećom za nas nije slučaj. Pri active scan-u na celoj aplikaciji testira se overall security naše aplikacije, SQL Injection, Miscellaneous napadi (svih vrsta) i napadi u cilju sakupljanja osetljivih podataka.

U našem slučaju, stavili smo threshold svakog od scana na low i strength na insane kako bi nam rezultati bili robustniji i realniji (slika 3). Sve ovo je pokušano na par GET metoda sa sledećim putanjama:

- https://localhost:8443/auth/ - vraća hello

- https://localhost:8443/message?receiver=b38a64e2-299b-4a05-bc30-5a45dd2ebdc0&sender=9220c03b-b0b5-46af-a821-249e2a97dcaa - vraća poruke između usera sa tim ide-evima tj. pokušava ako ništa (autorizacija)
- https://localhost:8443/search/light?city=Novi Sad&from=13:20 2020-06-19&to=05:41 2020-06-24 - vraća oglase koji su slobodni u tom periodu - nema autorizacije na ovoj metodi
- https://localhost:8443/request?status=RESERVED - vraća sve rezervisane requestove (imamo autorizaciju)



Slika 3. Konfiguracija owasp zap-a

Nad ova 4 GET requesta pronađen je jedan rizik niskog prioriteta i jedan srednjeg.

1. Rizik niskog prioriteta se desio jer nisu postavljeni headeri Pragma sa no-cache vrednosti, i Cache-Control: no-cache, no-store, must-revalidate jer je moguć napad koji ce sacuvati maliciozne podatke u cache-u nase aplikacije jer ne clear-ujemo cache i store-ujemo ga. Problem je resen dodavanjem ovih headera u response svakog zahteva koji dopre do aplikacije.

2. Ranjivost srednjeg nivoa tj. potencijalni format string napad. Do format string napada dolazi kada nepoznati korisnikov unos , kroz url, stavljamo “zdravo na gotovo” u neku funkciju za ispis. Tada maliciozni napadač može da zaobiđe proces formatiranja stringa tako što sam inicira sa %s da je sledeći argument na steku string i tada se izbacuju random sistemske vrednosti koje mogu biti od nekog značaja. Ekvivalent je sout(userInput), gde userInput može biti bilo šta, pa i %s koji hint-uje da je u pitanju string, a pored njega nije ništa prosleđeno. Tada sout izbaci neku sistemsku vrednost koju maliciozni napadač može da iskoristi (memorijski blok, vrednost u

memoriji). Ovo može postati problem, jer tada maliciozni napadač može da unese bilo koji string tj. funkciju koju može da odradi na našoj aplikaciji. Ukoliko maliciozni napadač unese %n tada može doći do potencijalnog DOS (Denial Of Service) napada, jer ce sout protumačiti to kao da treba da neki niz karaktera upiše na drugim pozicijama na steku, sto je time consuming (zahteva vremena za izvršiti) i tada može doći do zagušenja. Način da se to reši je da se eksplicitno formatira unos korisnika kao string, te tako neće doci do automatskog formatiranja i potencijalnog napada.