Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dorota Gajdošová

NoSQL - Netradičné injection útoky

Bezpečnosť informačných technológií

November 2022

Obsah

Kapitola 1

Úvod

Nerelačné databázy sú čoraz populárnejšou alternatívou k použitiu známych relačných databáz. Aj napriek tomu, že nerelačné databázy všeobecne nepodporujú Structured Query Language (SQL), stále sú zraniteľné na injection útoky a spolu s ich rastúcou popularitou rastie aj potreba chrániť dáta, ktoré sú v nich uložené.

Cieľom seminárnej práce je oboznámiť čitateľa so zraniteľnosťou na NoSQL injection, jej dopadom a riešením. V analytickej časti práce sa budeme zaoberať krátkym opisom nerelačných databáz, podrobnejšie sa pozrieme na databázu MongoDB a útok NoSQL injection. To zahŕňa jeho priebeh čo sa týka MongoDB databázy, spôsob zabezpečenia a dopad. Závažnosť NOSQL injection zraniteľnosti budeme demonštrovať prostredníctvom vykonania útokov na zraniteľné aplikácie OWASP Juice Shop a nosqlinjection od Stanislava Dashevskyia. Nakoniec navrhneme spôsoby ich zabezpečenia. V závere seminárnej práce zhodnotíme dosiahnuté výsledky.

Kapitola 2

Analýza

2.1 NoSQL

Pojem NoSQL bol prvýkrát použitý v roku 1998 pre relačné databázy, ktoré nepodporovali použitie jazyka SQL a následne bol opätovne pripomenutý na konferencii open-source nerelačných databáz v roku 2009, ktorá sa konala v San Franciscu [11]. NoSQL je skratkou pre "Not only SQL" alebo aj "No to SQL"[11]. Ide o databázy, ktoré ukladajú dáta v inom formáte ako relačné databázy čím čiastočne riešia ich nedostatky. Konkrétne najmä škálovateľnosť a s ňou súvisiacu výkonnosť [9]. Aj napriek tomu, že relačné databázy nie sú úplne neškálovateľné - sú ťažko škálovateľné, nerelačné databázy to značne uľahčujú [11].

Medzi hlavné výhody použitia NoSQL patrí rýchle zapisovanie a čítanie dát, podpora veľkokapacitného úložiska, ľahká rozšíriteľnosť a nízka cena [5]. Na rozdiel od relačných databáz väčšina NoSQL systémov nepodporuje ACID vlastnosti - atomicitu (atomicity), konzistentnosť (consistency), izolovanosť (isolation) a trvácnosť (durability) [9]. Namiesto toho garantuje BASE vlastnosti a je v súlade s CAP teorémom (systém je schopný spĺňať najviac dve z vlastností konzistentnosť,

dostupnosť a odolnosť voči prerušeniam) [9]. BASE vlastnosti hovoria o tom, že systém je dostupný v čase výpadku, že stav systému sa môže časom meniť a že systém sa stane konzistentným v momente kedy prestane prijímať vstupy.

Aj napriek tomu, že nerelačné databázy majú mnoho výhod, nie sú univerzálnym riešením. Na základe vlastností konkrétnej nerelačnej databázy a účelu, ktorý ma spĺňať vieme vybrať vhodný typ databázy. Všeobecne rozlišujeme štyri typy NoSQL:

- kľúč-hodnota databázy,
- stĺpcovo-orientované databázy,
- dokumentové databázy,
- grafové databázy [11].

Čo sa týka kľúč-hodnota databáz ide o databázy, ktoré ukladajú dáta podobne ako slovníky či hashovacie tabuľky. Dáta sú reprezentované párom obsahujúcim unikátny kľúč (alebo viaceré kľúče) a samotnú hodnotu. Hodnota môže byť jednoduchý dátový typ ale aj komplexný objekt [8]. Pri vyžiadaní hodnoty sa kľúč konvertuje na lokalitu, kde sa zapíšu, upravia alebo vymažú požadované dáta [1]. Výhodou tohto typu je rýchlosť, škálovateľnosť, jednoduchosť, horizontálna distribúcia a nevýhodou, že množstvo dát sa nedá formátovať ako pár kľúč:hodnota [1]. Patria sem databázy ako Redis, Tokyo Cabinet/Tyrant, MemcacheDB a iné [1].

Stĺpcovo-orientované databázy sú hybridom medzi relačnými a nerelačnými databázami. Tento typ databáz ukladá a spracováva dáta po stĺpcoch namiesto riadkov a neriadi sa tak striktnými pravidlami ako relačné databázy [1]. Stĺpce sú často rovnakého dátového typu a umožňujú efektívnu kompresiu, rýchle čítanie dát a vykonávanie výpočtov nad nimi [8]. Stĺpcovo-orientované databázy sú prispôsobené

najmä na analytické účely [1]. Ide o databázy ako napríklad Big Table, Cassandra a iné [9].

Dokumentové databázy ukladajú dáta v podobe dokumentov (napríklad PDF, JSON, BSON, XML), ktoré je možné zoskupovať do kolekcií [1]. Jedna kolekcia môže obsahovať ľubovolný počet dokumentov ľubovolného typu [1]. Každému dokumentu je priradený unikátny kľúč. Tieto databázy majú dobrú výkonnosť a ponúkajú možnosť horizontálnej škálovateľnosti [9]. Okrem týchto vlastností sú aj flexibilné, keďže umožňujú developérom meniť a tak prispôsobovať štruktúru dát aktuálnym požiadavkám aplikácií [8]. Dokumentové databázy sú využívané napríklad pri vývoji mobilných aplikácií alebo v ecommerce softvéroch a patria medzi ne databázy MongoDB, CouchDB a iné [9].

Posledné, grafové databázy, sú zamerané na zachytávanie vzťahov medzi dátovými elementami [8]. Dáta sú ukladané v podobe grafov. Graf pozostáva z uzlov a hrán. Uzly reprezentujú jednotlivé elementy a hrany vzťahy medzi nimi [9]. Každý uzol priamo ukazuje na susedný uzol [9]. Čo sa týka použitia v praxi, grafové databázy sú väčšinou používané spolu s inými tradičnejšími databázami napríklad pri odhaľovaní podvodov, v bioinformatike alebo sociálnych sieťach [8]. Databázy sú rýchle, v súlade s vlastnosťami ACID a sú inšpirované téoriou Eulerovského ťahu [9].

Tabuľka 2.1 zobrazuje porovnanie jednotlivých typov nerelačných databáz medzi sebou ako aj s relačnými databázami vzhľadom na ich charakteristické vlastnosti.

2.1.1 MongoDB

Vo vlastnom riešení budeme využívať derivát databázy MongoDB, ktorej sa v skratke venujeme v tejto sekcii. Ako bolo v podkapitole 2.1 spomínané, ide o

Typ databázy	Výkonnosť	Škálovateľnosť	Flexibilita	Zložitosť	Funkčnosť
Kľúč-hodnota	Vyoká	Vyoká	Vyoká	Žiadna	Variabilná (Žiadna)
Stĺpcovo-orientovaná	Vyoká	Vyoká	Mierna	Nízka	Minimálna
Dokumentová	Vyoká	Variabilná (Vyoká)	Vysoká	Nízka	Variabilná (Nízka)
Grafová	Variabilná	Variabilná	Vysoká	Vysoká	Grafová teória
Relačná	Variabilná	Variabilná	Nízka	Mierna	Relačná algebra

Tabuľka 2.1: Porovnanie typov databáz

dokumentovú databázu, číže pozostáva z kolekcií obsahujúcich dokumenty. MongoDB je databáza bez preddefinovanej schémy napísaná v jazyku C++ vyvíjaná ako open-source projekt pod vedením firmy 10gen Inc, ktorá bola uverejnená v roku 2009 [11].

Aj napriek tomu, že sa zaraďuje medzi nerelačné databázy, nesie niektoré prvky relačných databáz [11]. Poskytuje vlastný query jazyk MongoDB Query Language, sekundárne indexy a zaručuje silnú konzistenciu [3]. Medzi jej ďalšie výhody patrí automatický "sharding" a dobrá výkonnosť. "Sharding" je proces, ktorý nastáva ak aplikácii nepostačujú zdroje jedného databázového servera. Dáta sa v takom prípade prerozdelia medzi viacero databázových serverov (inštancií servera) tak aby bolo zachované poradie dát. Nevýhodou tejto databázy je, že môže byť nespolahlivá a indexovanie môže zaberať veľkú časť RAM.

Práca s MongoDB

MongoDB ukladá dokumenty vo formáte BSON, ktorý je binárnou reprezentáciou formátu JSON [8]. Príklad dokumentu na uloženie detailov objednávky by mohol vyzerať nasledovne:

```
{
   order_number: 123456,
   email: "jozko.m@eshop.com",
```

```
    phone_number: "0901234567",
    created: new Date("11/16/2022"),
    comment: "Please contact me before...",
    items: ["orange juice", "apple juice"],
    status: "ordered"
}
```

Databáza podporuje CRUD operácie nad dátami - vytvorenie (create), čítanie (read), aktualizácia (update) a mazanie (delete) [8].

Vytvorenie dokumentu:

```
db.<collection>.insertOne( { order_number: 123456, email: .... } );
```

V prípade, že by sme naraz chceli vložiť do kolekcie viacero dokumentov je vhodné použiť metódu `insertMany()`.

Čítanie dokumentu:

```
db.<collection>.find( { status: "ordered", .... } );
```

Metóda vráti dokumenty spĺňajúce dané kritérium.

Aktualizácia dokumentu:

```
db.<collection>.updateOne( { status: "ordered" }, { $set: {sta-
tus: "shipped"} } );
```

Ak by sme chceli aktualizovať všetky dokumenty spĺňajúce kritérium je vhodné použiť metódu `updateMany()`. Metóda `replaceOne()` nahradí prvý dokument v kolekcii vyhovujúci kritériu.

Mazanie dokumentu:

```
db.<collection>.deleteOne( { status: "ordered" } );
```

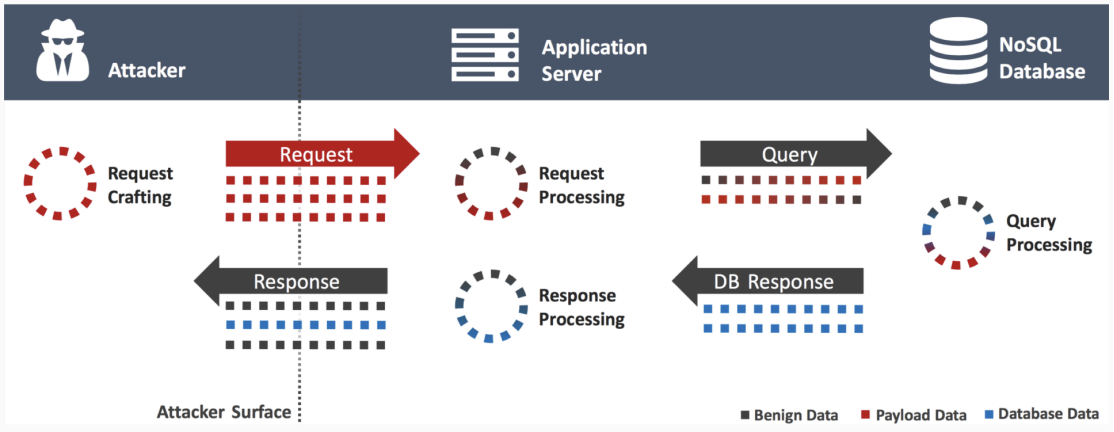
V prípade, že by sme naraz chceli zmazať všetky dokumenty spĺňajúce kritérium je vhodné použiť metódu `deleteMany()`.

2.2 NoSQL injection útok

Aj napriek tomu, že NoSQL databázy nevyužívajú tradičnú SQL syntax stále sú zraniteľné na injection útoky [4]. Tieto útoky môžu spôsobiť napríklad neoprávnené získanie prístupu, exfiltráciu databázy, úpravu záznamov databázy a viesť k jej celkovej kompromitácii. Útok sa realizuje vložením škodlivého kódu do vstupného poľa, ktoré nie je ošetrené alebo priamo do URL. V závislosti od použitého NoSQL API a dátového modelu sa môže škodlivý kód vykonávať na databázovej alebo aplikačnej vrstve [4]. Pred vykonaním útoku sa útočník musí oboznámiť so syntaxou (pre NoSQL databázy neexistuje univerzálny jazyk), dátovým modelom a programovacím jazykom a na základe týchto zistení napísať škodlivý kód [4].

Existuje päť typov hlavných mechanizmov SQL injection útokov relevantných aj pre NoSQL databázy:

- Tautológie - útočník zostaví query tak aby boli podmienené výrazy vyhodnotené vždy ako pravdivé,
- UNION query - útočník zneužije zraniteľný parameter query tak aby mu po jej vykonaní bol vrátený zmenený dataset,
- JavaScript injection - neošetrenie vstupu umožňuje vloženie ľubovolného JavaScript kódu,



Obr. 2.1: Priebeh NoSQL injection útoku [4]

- Piggybacked query - útočník ponechá pôvodnú query no pripojí za ňu jednu alebo viacero vlastných query,
- Origin violation - útočník zneužije na vykonanie nepovolenej akcie overeného používateľa [10].

V nasledujúcej sekcii sa budeme venovať zraniteľnosti databázy MongoDB na injection útoky, keďže práve MongoDB využívame vo vlastnom riešení.

2.2.1 Útok na databázu MongoDB

Ako už bolo spomínané vyššie, zraniteľnosť, ktorá útok na databázu umožňuje je nesprávne ošetrený alebo neošetrený vstup v aplikácii. To znamená, že vstupné údaje od používateľa sú uložené priamo do premennej bez kontroly. Táto premenná je následne použitá pri zostavovaní query a ak je namiesto zamýšľaného vstupu do poľa (alebo URL) vložený škodlivý kód, pri jej vykonávaní sa vykoná práve ten čím je narušená bezpečnosť databázy [6]. Tento proces je zobrazený na obrázku 2.1.

Aj napriek tomu, že MongoDB obsahuje nástroj na bezpečné zostavovanie BSON

query tak v niektorých alternatívnych parametroch povoľuje neserializované JSON a JavaScript výrazy [4]. Najčastejšie je takýto vstup podsunutý a spracovaný cez operátor $where, ktorý predstavuje jednoduchý filter [4]. Pokiaľ je tento spôsob ošetrený ďalšou možnosťou pre útočníkov je nahradiť samotný operátor $where PHP premennou nazvanou rovnako, ktorá vykoná požadovaný JavaScript kód [4].

2.2.2 Zabezpečenie

Keďže zraniteľnosť vzniká práve z dôvodu nesprávne ošetreného vstupu jedným z hlavných riešení je práve jeho ošetrenie. To môžeme dosiahnuť napríklad prostredníctvom validácie vstupu, čo znamená, že overujeme vstup už priamo v mieste zadávania (limitácia znakov vstupu) [6]. Nevyhnutným krokom je taktiež kontrola vstupu v podobe premennej pred jeho vložením do query. To realizujeme či už prostredníctvom filtrovania určitých znakov alebo parametrizovaných výrazov [6]. Pri ošetrovaní je vhodnejšie použiť overené knižnice namiesto vlastných funkcii, kontrolovať nie len používateľské vstupy a taktiež sa predpokladá, že systém má viacero vrstiev, ktoré zaistia, že vstup používateľa nie je priamo vložený do query [10].

Okrem ošetrenia vstupu je odporúčaná aj izolácia privilegií, bezpečnostné skenovanie, testovanie aplikácie a ďalšie [10]. Pri detekcii sa odporúča implementácia WAF (Web application firewalls), IDS (Intrusion detection systems), monitorovanie aktivity a iné [10].

2.2.3 Dopad

Ako už bolo vyššie spomínané, úspešné zneužitie zraniteľnosti na NoSQL injection môže mať vážne následky ako neoprávnené získanie privilegovaného prístupu

alebo dát a zároveň môže predstavovať aj vstupnú bránu pre iné útoky ako napríklad DoS. Pozornosť, ktorá by mala byť venovaná zabezpečeniu rastie spolu s ich popularitou. V prvej desiatke sa aktuálne nachádzajú až tri nerelačné databázy - MongoDB (5. miesto), Redis (6. miesto) a Elasticsearch (7. miesto) a ich popularita stále rastie [2].

Jedným z prípadov, kedy došlo k NoSQL injection útoku bol prípad z roku 2016. Útočníkom sa podarilo získať a predať ukradnutú databázu a záznamy v celkovej hodnote 100 000$. Skupiny záznamov bolo možné kúpiť na čiernom trhu za 10 000$ a taktiež sa predávali aj informácie o zraniteľnostiach na ich webovej stránke. Išlo o 1.5 miliónov dát zákazníkov spoločnosti Verizon Enterprise Solutions, ktorá sa zaoberá poskytovaním telekomunikačných služieb. Útok bol realizovaný na databázu MongoDB.[7]

Kapitola 3

Riešenie

Vo vlastnom riešení sme na demonštráciu závažnosti zraniteľnosti využili dve existujúce zraniteľné aplikácie - OWASP Juice Shop a aplikáciu nosqlinjection z projektu damn-vulnerable-web-apps od Stanislava Dashevskyia. Pôvodne sme skúšali aj iné aplikácie ako napríklad nodejs-goof od Snyk Labs a aj vytvorenie vlastnej aplikácie. Tieto pokusy sa však do času finalizovania dokumentácie skončili neúspešne a tak sú dokumentované iba funkčné časti. Opakovane bol problém s pripojením k databáze a potrebnými balíčkami/modulmi/závislosťami (npm). Taktiež sme skúšali aj skenovanie aplikácií. Avšak nástroje nikto, OpenVAS, NoSQLMap ani Nessus zraniteľnosť NoSQL v OWASP Juice Shop ani v nosqlinjection aplikácii nenašli.

3.1 Príprava prostredia

Aplikácia OWASP Juice Shop bola spustená na virtuálnom stroji (Kali Linux) lokálne pomocou návodu v časti "From pre-packaged distribution" z repozitára https://github.com/juice-shop/juice-shop:

1. Inštalácia 64bit node.js
2. Stiahnutie najnovšej 64bit verzie aplikácie - juice-shop-14.3.1_node18_linux_x64.tgz
3. Rozbalenie a prejdenie priečinku
4. Spustenie `npm start`
5. Prejdenie na adresu http://localhost:3000

Aplikácia nosqlinjection z damn-vulnerable-web-apps bola spustená na MacOS lokálne pomocou návodu v časti " Installation" z repozitára https://github.com/standash/damn-vulnerable-web-apps:

1. Inštalácia Docker 4.13.1
2. Stiahnutie aplikácie z repozitára - `git clone`
3. Zostavenie aplikácie a Docker Image pomocou `./run.py nosqlinjection 8888 80`
4. Prejdenie na adresu http://localhost/login.html

3.2 OWASP Juice Shop

V prvom prípade sme začali skenom zraniteľnej aplikácie a nasledovalo zneužitie zraniteľnosti na NoSQL injection.

OWASP Juice Shop umožňuje realizáciu troch rôznych NoSQL útokov. V rámci riešenia sme realizovali jednoduchý DoS útok na aplikáciu a exfiltráciu databázy, ktorou sa nám podarilo získať všetky objednávky.

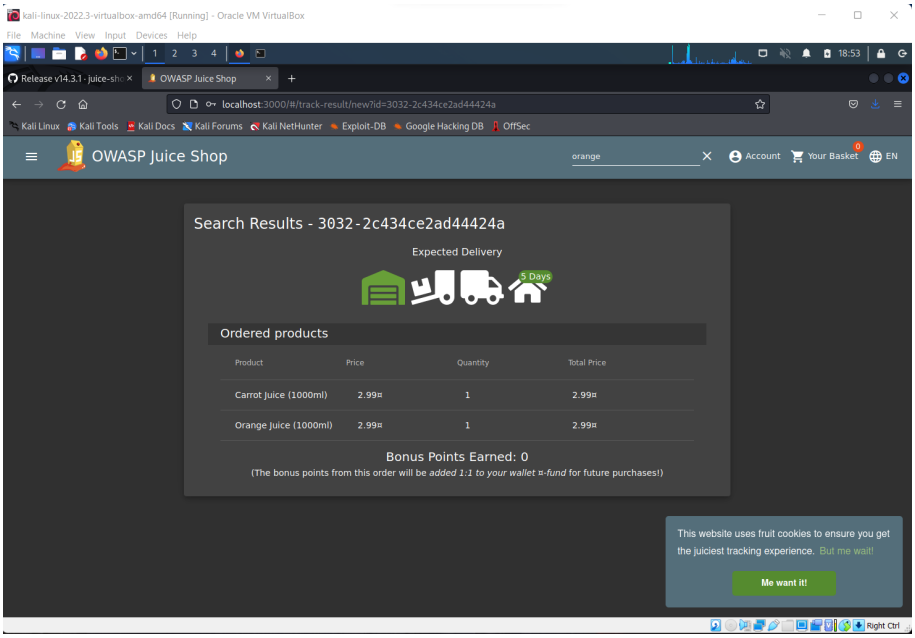
3.2.1 Exfiltrácia databázy

Zraniteľnosť NoSQL, ktorú sme pri tomto útoku zneužili sa nachádzala na podstránke na sledovanie objednávok. Aby sme sa k tejto podstránke dostali, bolo potrebné registrovať sa ako používateľ a vytvoriť objednávku. Registrovali sme sa pod emailom test@nosql.com a vytvorili objednávku s dvoma džúsmi.

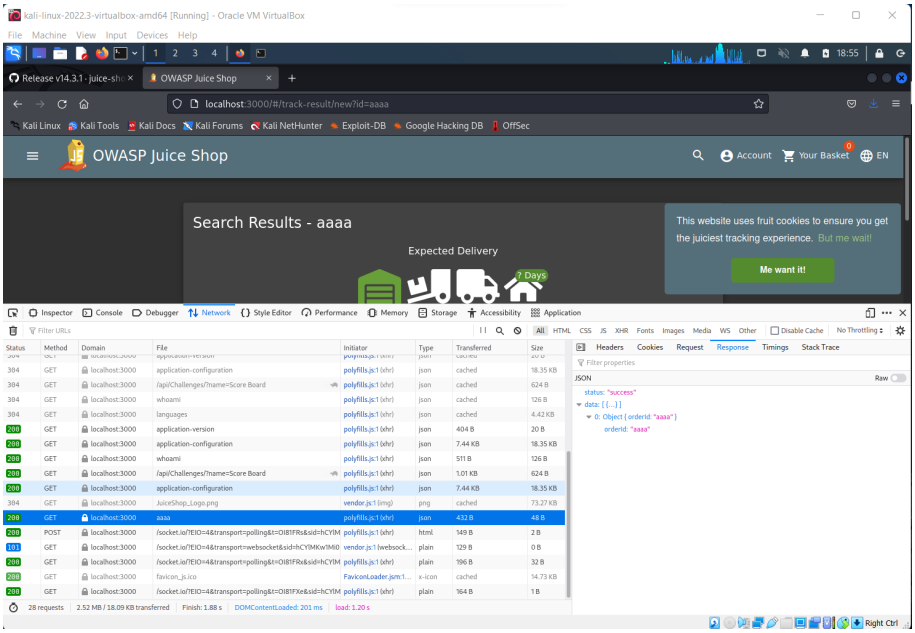
Po dokončení objednávky sa nám zobrazila obrazovka s možnosťou jej sledovania, prešli sme teda na obrazovku sledovania objednávok (obrázok 3.1). Pri prezeraní obrazovky sme si všimli v URL adrese parameter id, ktorý by mohol predstavovať zraniteľnosť. Skúsili sme preto do parametre vložiť náhodný reťazec a znaky " aaaa ", " ’ ", " ” ", " || " a v Dev Tools sme videli chybu a zároveň vidíme, že filtrácia vstupu tam nie je implementovaná (obrázky 3.2, 3.3, 3.4).

Keďže ide o podstránku, ktorá ma zobraziť objednávku s id zadaným v podobe parametru, query pravdepodobne vyberá objednávky z kolekcie objednávok na základe tohto parametru. Úlohou bolo teda vytvoriť takú časť query a vložiť ju do parametru id aby vrátilo všetky objednávky z kolekcie.

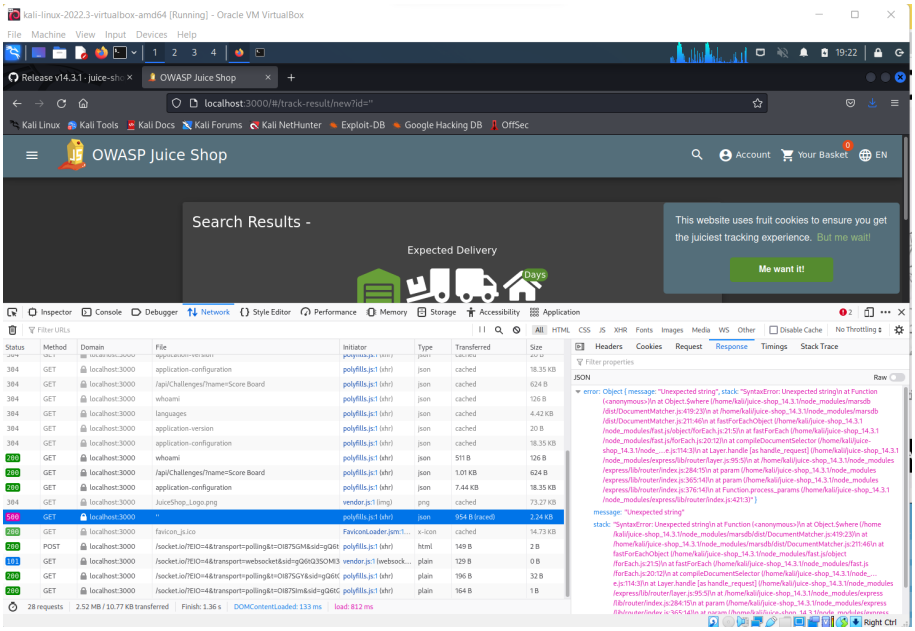
Skúšali sme rôzne zápisy ako napríklad "{$gt: ’ ’ }", "´ || 1==1", "´ 1==1 })//", "´ 1==1//" a ďalšie. Nakoniec sa nám exfiltráciu podarilo dosiahnuť vstupom "´ || 1==1 || ’ "(obrázky 3.5, 3.6). Pri vytváraní tejto časti query sme sa inšpirovali verejne dostupnými známymi škodlivými vstupmi ako napríklad https://github.com/DevanshRagha
oSQL_injection_stuff/blob/main/README.mdpayloads.



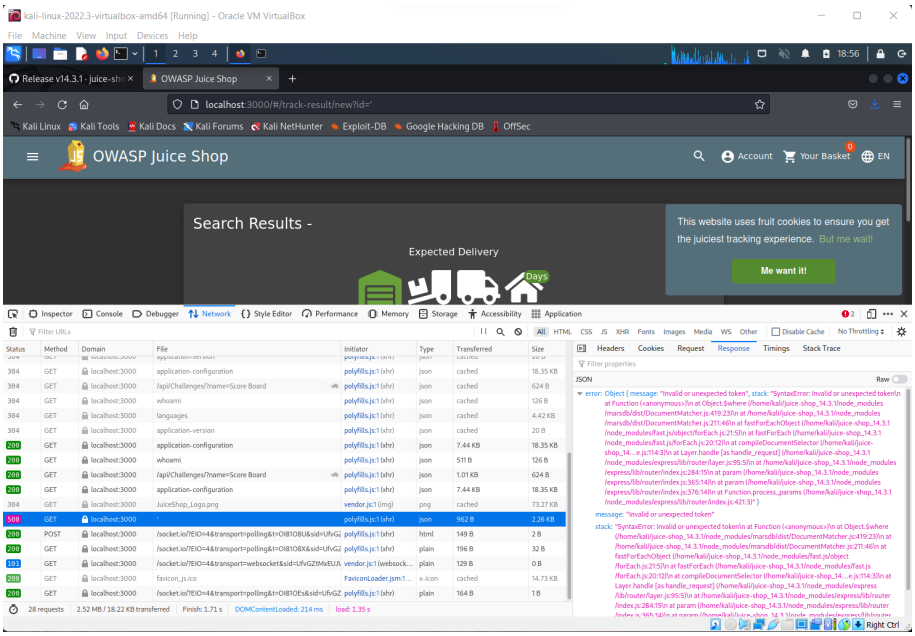
Obr. 3.1: Sledovanie objednávky



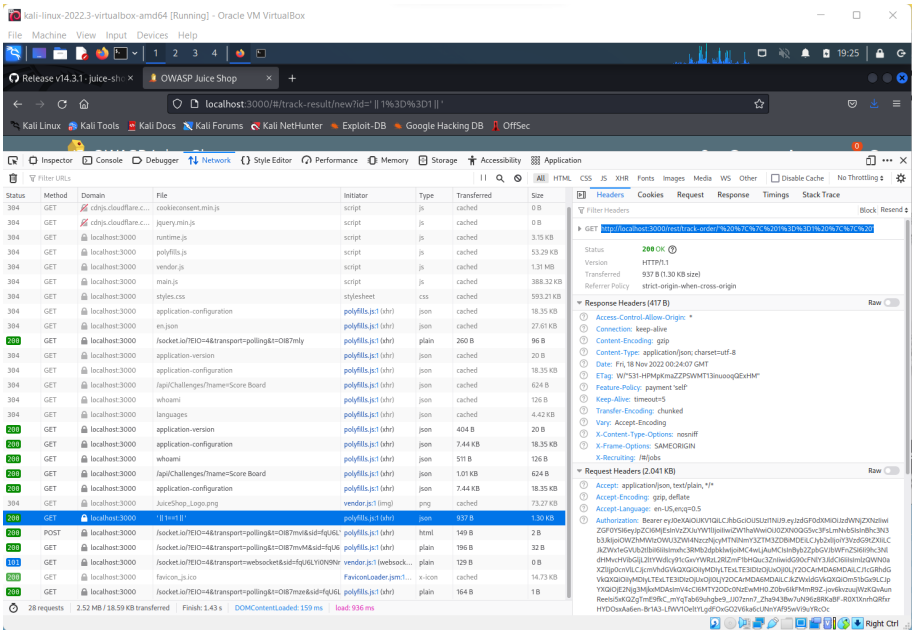
Obr. 3.2: Vstup " aaaa "



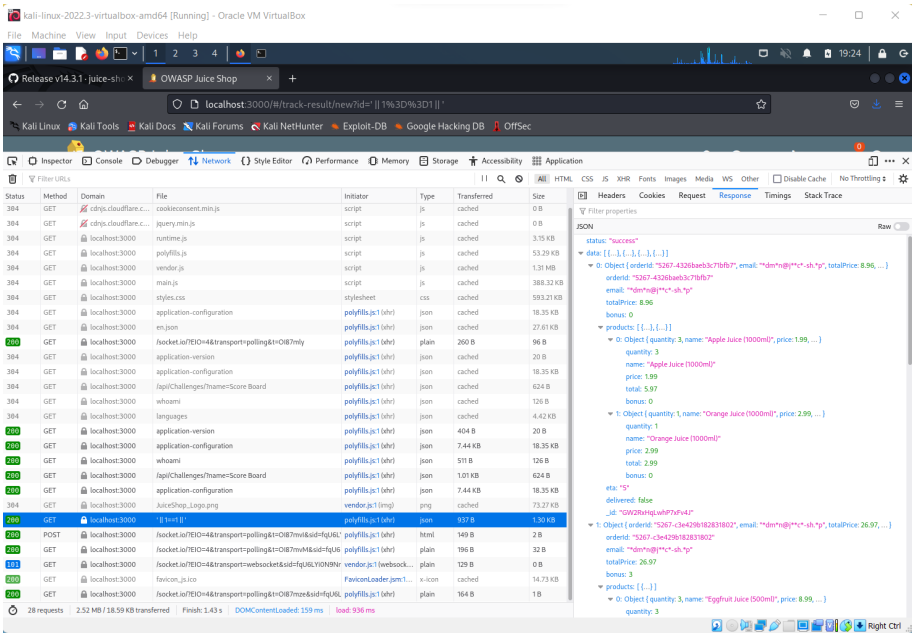
Obr. 3.3: Vyvolanie chyby vstupom " ” "



Obr. 3.4: Vyvolanie chyby vstupom " ’ "



Obr. 3.5: Vloženie škodlivého kódu



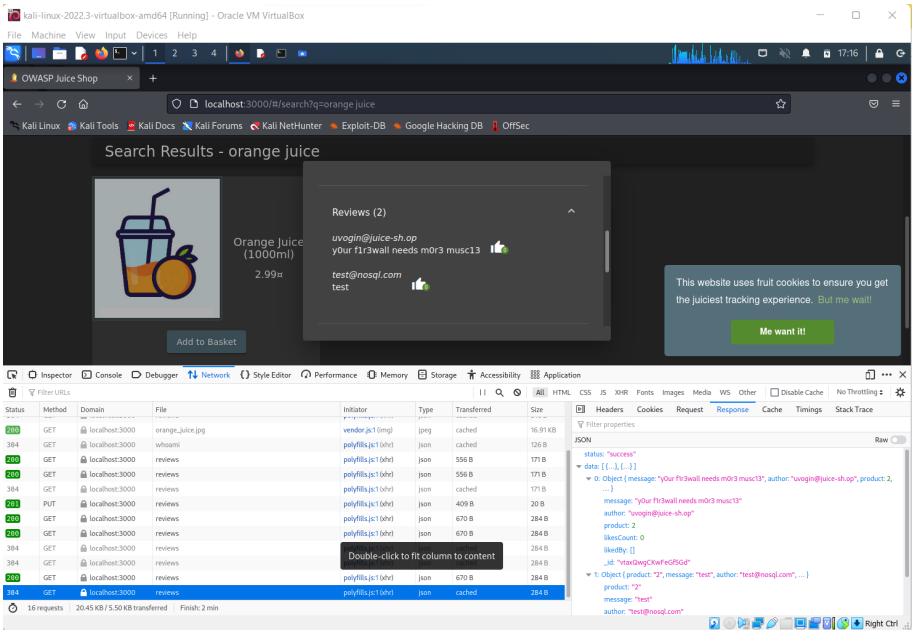
Obr. 3.6: Exfiltrácia databázy objednávok

3.2.2 Dočasné zneprístupnenie aplikácie

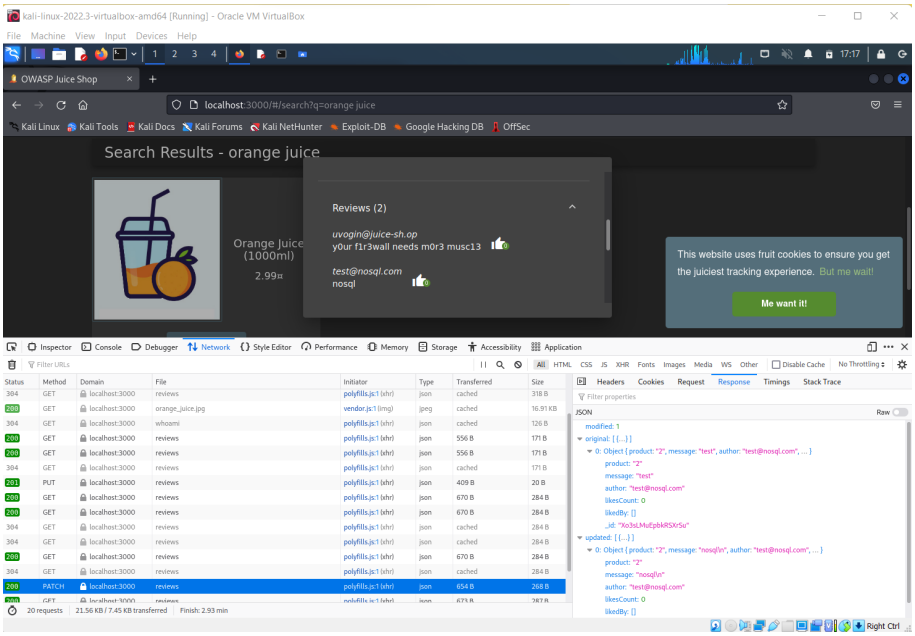
Zraniteľnosť NoSQL, ktorú sme pri tomto útoku zneužili sa nachádzala na podstránke na zadávanie recenzií. Na to aby sme mohli pridávať recenzie sme sa museli registrovať. Registrovali sme sa ako používateľ test@nosql.com a prešli sme na obrazovku s produktmi. Ako produkt na zadávanie recenzií sme zvolili " Orange juice", otvorili sme teda detail produktu a pridali recenziu s textom "test" (obrázok 3.7).

Následne sme sa pozreli do Dev Tools - Network a prezerali si hlavičky, obsah požiadaviek a odpovede na ne. Pri úprave recenzie z "text"na "nosql"sme pre požiadavku našli v hlavičke filename s cestou /rest/products/2/reviews (obrázok 3.9).

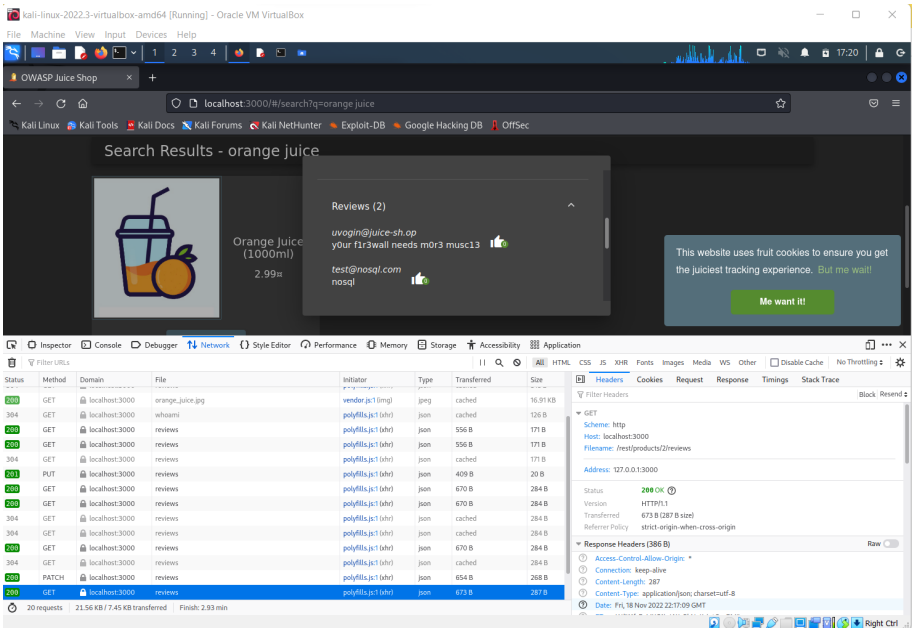
Túto cestu sme vložili za http://localhost/ a skúsili vyvolať chybu cestou nahradením 2 za string/znak. Vyvolanie chyby nám napovedalo zraniteľné miesto a použitie Express.js a tak ako posledný krok nám postačovalo nahradiť 2 v ceste funkciou `sleep()`. Pri testovaní sme použili hodnotu 1000 a teda do URL sme vložili funkciu `sleep(1000)`, ktorá vyvolala zneprístupnenie na 1 sekundu (3.10).



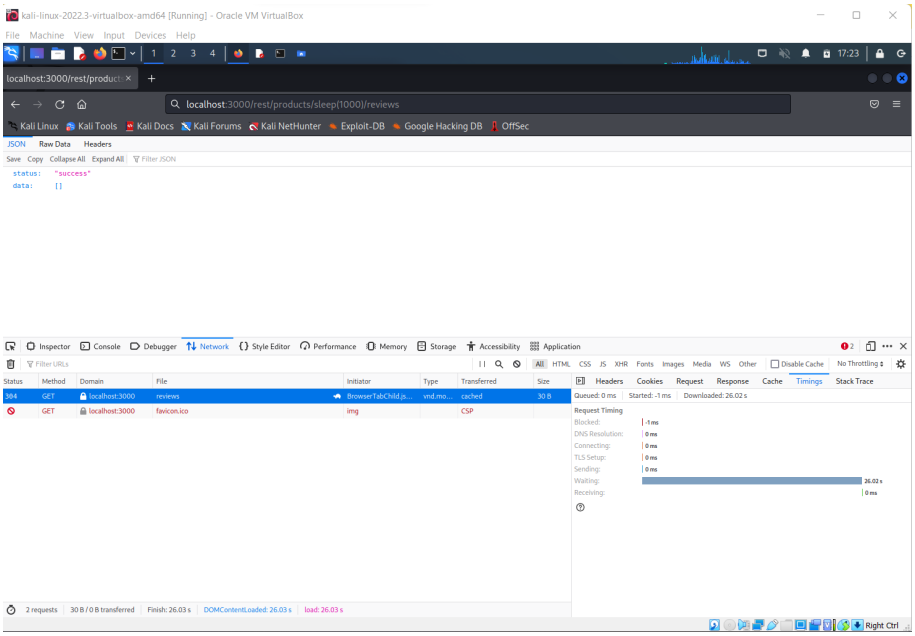
Obr. 3.7: Pridanie prvej recenzie



Obr. 3.8: Úprava recenzie



Obr. 3.9: Cesta



Obr. 3.10: Zneprístupnenie stránky

3.3 nosqlinjection

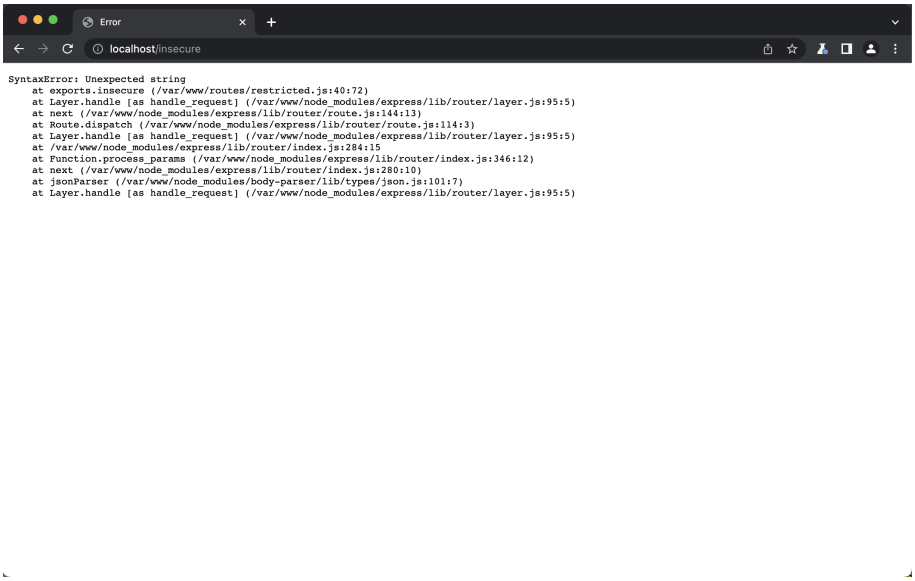
Pri aplikácii nosqlinjection sme začali rovno zneužitím zraniteľnosti. Keďže ide o stránku prihlásenia, môžeme predpovedať, že zraniteľnosť sa bude nachádzať práve v týchto poliach a parametroch meno používateľa a heslo. Z informácii o aplikácii vieme, že v nej je registrovaný používateľ pod menom "Batman". Našou úlohou bolo prihlásiť sa práve pod týmto používateľom bez poznania hesla.

3.3.1 Prihlásenie

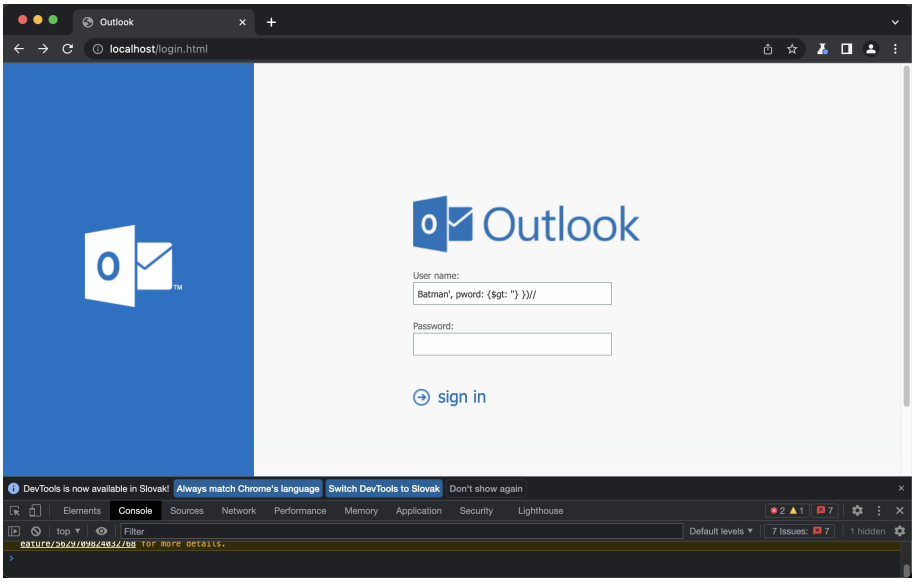
Najprv sme sa tak ako v predošlej aplikácii presvedčili o tom, že pole na používateľské meno je zraniteľné na NoSQL. Vyvolali sme chybu zadaním vstupu "´ "(obrázok 3.11). Keď už sme sa presvedčili o zraniteľnosti na NoSQL, mohli sme prejsť na vytvorenie škodlivej časti query. Predpokladáme, že vstup je spracovávaný tak, že záznamy z dokumentu z kolekcie porovnáva s hodnotami parametrov.

Skúšali sme rôzne vstupy ako "{$gt: ”}’", "Batman’, password: {$gt: ”} //", "Batman’, password: {$gt: ”} } //". Keďže sa nám dlhší čas nepodarilo prísť na spôsob obídenia prihlásenia tak sme sa pozreli do kódu a zistili, že názov pre heslo nie je "password " ale "pword ". Po opätovnom vyskúšaní predošlých škodlivých výrazov nakoniec zafungoval výraz "Batman’, pword: {$gt: ”} }) //"(obrázky 3.12, 3.13). Neskôr sme zistili, že by to šlo aj iba pomocou "Batman’ }) //"čo by zakomentovalo celú časť porovnávania hesla.

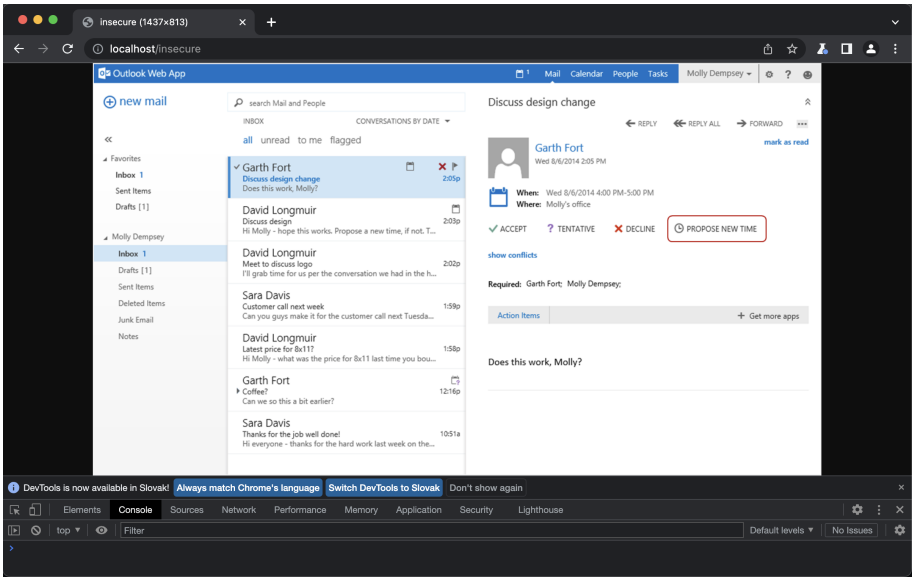
Nakoniec sme skúšali ešte aj vloženie nejakej funkcie a podarilo sa nám napríklad presmerovanie, čo teda predstavuje miesto zraniteľné aj na funkcie s Express (obrázky 3.14, 3.15).



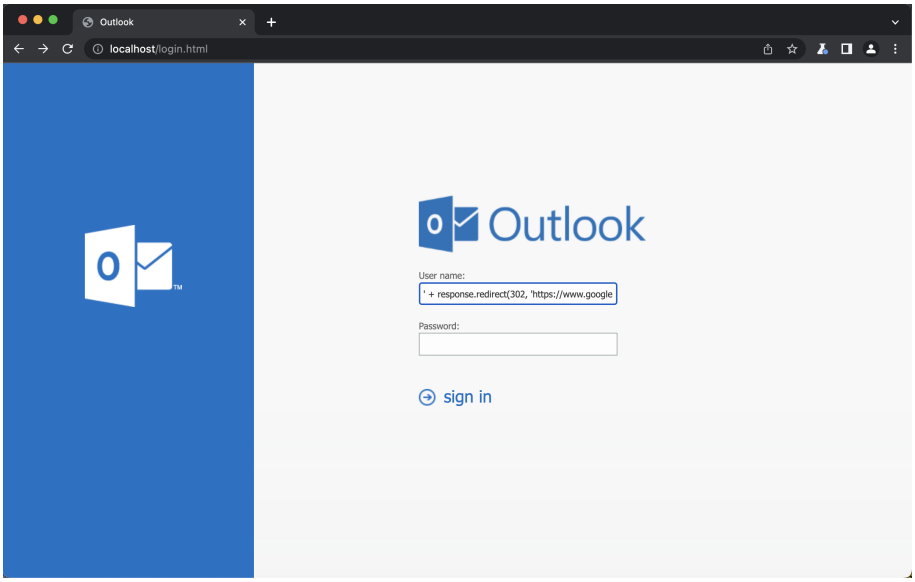
Obr. 3.11: Vyvolanie chyby



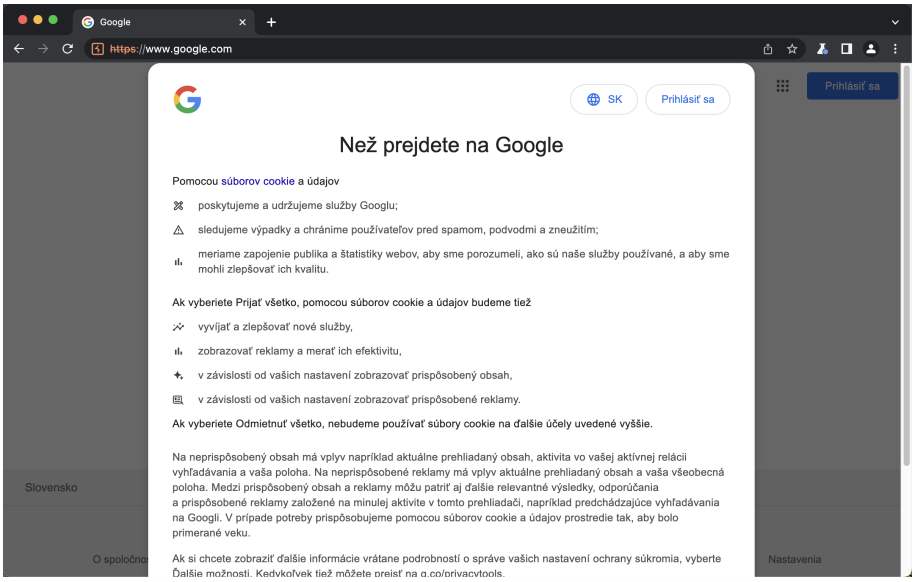
Obr. 3.12: Vloženie škodlivého kódu na prihlásenie



Obr. 3.13: Získanie prístupu



Obr. 3.14: Vloženie škodlivého kódu na vykonanie funkcie



Obr. 3.15: Presmerovanie

3.4 Zabezpečenie

Pri vybraných aplikáciách sme navrhli zabezpečenie v podobe kontroly a filtrácie vstupu, čo znamená, že parameter zo vstupu nebude priamo vložený do query ale pred tým odkontrolovaný pomocou regexu.

Pri exfiltrácii databázy v OWASP Juice Shop zo získaných informácii môžeme predpokladať, že zraniteľný parameter má požadovaný tvar x-x, kde časti x sa skladajú z niekoľkých písmen a čísel. Regex by preto mohol vyzerať nasledovne: /([A-Za-z0-9]+-[A-Za-z0-9]+)/g. Pri DoS útoku na OWASP Juice Shop sme pracovali pravdepodobne tiež so zraniteľným parametrom id. Tento mal však pri prezeraní vždy iba číselnú hodnotu, takže regex by mohol vyzerať: /([0-9]+)/g.

Čo sa týka nosqlinjection aplikácie z damn-vulnerable-web-apps, používateľské meno by malo obsahovať iba písmená a čísla, to by mohol riešiť regex /([A-Za-z0-9]+)/g .

Na základe zachytenia škodlivého vstupu pomocou regexu môžeme následne buď vyvolať chybu alebo nahradiť nevyhovujúce a špeciálne znaky ako "/", "{ }", ":"napríklad medzerami.

Kód, ktorý zapríčinil zraniteľnosť NoSQL je zobrazený na obrázkoch 3.16, 3.17 a 3.18. Kód je možné nájsť buď manuálnym prechádzaním alebo pomocou nástrojov na skenovanie zdrojového kódu.

```
db.orders.find({ $where: `this.orderId === '${id}'` }).then((order: any) => {
  const result = utils.queryResultToJson(order)
  challengeUtils.solveIf(challenges.noSqlOrdersChallenge, () => { return result.data.length > 1 })
  if (result.data[0] === undefined) {
    result.data[0] = { orderId: id }
  }
  res.json(result)
}, () => {
  res.status(400).json({ error: 'Wrong Param' })
})
```

Obr. 3.16: Zraniteľný kód v aplikácii OWASP Juice Shop - podstránka sledovania objednávok

```
db.reviews.find({ $where: 'this.product == ' + id }).then((reviews: Review[]) => {
  const t1 = new Date().getTime()
  challengeUtils.solveIf(challenges.noSqlCommandChallenge, () => { return (t1 - t0) > 2000 })
  const user = security.authenticatedUsers.from(req)
  for (let i = 0; i < reviews.length; i++) {
    if (user === undefined || reviews[i].likedBy.includes(user.data.email)) {
      reviews[i].liked = true
    }
  }
  res.json(utils.queryResultToJson(reviews))
}, () => {
  res.status(400).json({ error: 'Wrong Params' })
})
```

Obr. 3.17: Zraniteľný kód v aplikácii OWASP Juice Shop - podstránka recenzií

```
server.dbprovider.findOne("users", loginParam, function(error, item) {
    try {
        if (error != null) {
            response.send("MongoDB ERROR: " + error);
            return;
        }
        if (item != null) {
            response.sendfile(__dirname + "/resources/logged-in.png");
        }
        else {
            response.sendfile(__dirname + "/resources/access-denied.png");
        }
    }
    catch (e){
        response.sendfile(__dirname + "/resources/access-denied.png");
    }
});
```

Obr. 3.18: Zraniteľný kód v aplikácii nosqlinjection

Kapitola 4

Zhodnotenie

V seminárnej práci sme v skratke analyzovali NoSQL databázy a databázu MongoDB. Taktiež sme sa pozreli na priebeh NoSQL injection útoku. Bližšie sme si rozobrali útok na MongoDB databázu, spôsob zabezpečenia a možný dopad. Po oboznámení sa s danou problematikou sme prešli k vlastnému riešeniu. Riešenie zahŕňalo vykonanie útoku na dve zraniteľné aplikácie OWASP Juice Shop a nosqlinjection aplikáciu od Stanislava Dashevskyia. Útokmi sa nám podarilo získať databázu objednávok, vykonať jednoduchý DoS útok a získať prístup do aplikácie, čím sme sa presvedčili o dôležitosti zabezpečenia aj pre nerelačné databázy.

Literatúra

[1] Deka Ganesh Chandra. "BASE analysis of NoSQL database". In: *Future Generation Computer Systems* 52 (2015), s. 13–21.

[2] DB-Engines. *DB-Engines Ranking*. URL: https://db-engines.com/en/ranking (cit. 18. 11. 2022).

[3] Miguel Diogo, Bruno Cabral a Jorge Bernardino. "Consistency models of NoSQL databases". In: *Future Internet* 11.2 (2019), s. 43.

[4] OWASP Web Security Testing Guide. *Testing for NoSQL Injection*. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.6-Testing_for_NoSQL_Injection (cit. 17. 11. 2022).

[5] Jing Han et al. "Survey on NoSQL database". In: *2011 6th International Conference on Pervasive Computing and Applications*. 2011, s. 363–366. DOI: 10.1109/ICPCA.2011.6106531.

[6] Boyu Hou et al. "MongoDB NoSQL injection analysis and detection". In: *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2016, s. 75–78.

[7] New Net Technologies Ltd. *1.5 MILLION VERIZON ENTERPRISE SOLUTIONS CUSTOMERS' INFORMATION BREACHED*. URL: https://

www.newnettechnologies.com/1-5-million-verizon-enterprise-solutions-customers-information-breached.html (cit. 15. 11. 2022).

[8] MongoDB. *MongoDB*. URL: https://www.mongodb.com/ (cit. 15. 11. 2022).

[9] Ameya Nayak, Anil Poriya a Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases". In: *International Journal of Applied Information Systems* 5.4 (2013), s. 16–19.

[10] Aviv Ron, Alexandra Shulman-Peleg a Anton Puzanov. "Analysis and mitigation of NoSQL injections". In: *IEEE Security & Privacy* 14.2 (2016), s. 30–39.

[11] Christof Strauch, Ultra-Large Scale Sites a Walter Kriha. "NoSQL databases". In: *Lecture Notes, Stuttgart Media University* 20.24 (2011), s. 79.