

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dorota Gajdošová

NoSQL - Netradičné injection útoky

Bezpečnosť informačných technológií

November 2022

Obsah

1	Úvod	1
2	Analýza	2
2.1	NoSQL	2
2.1.1	MongoDB	3
2.2	NoSQL injection útok	5
2.2.1	Útok na databázu MongoDB	5
2.2.2	Zabezpečenie	6
2.2.3	Dopad	7
3	Riešenie	8
3.1	Príprava prostredia	8
3.2	OWASP Juice Shop	9
3.2.1	Exfiltrácia databázy	10
3.2.2	Dočasné znepřístupnenie aplikácie	14
3.3	nosqlinjection	17
3.3.1	Prihlásenie	17
3.4	Zabezpečenie	20
4	Zhodnotenie	22

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 navrhne spôsoby ich zabezpečenia. V závere seminárnej práce zhodnotíme dosiahnuté výsledky.

Kapitola 2

Analýza

2.1 NoSQL

NoSQL je skratkou pre "Not only SQL" alebo aj "No to SQL"[10]. Ide o databázy, ktoré všeobecne nepodporujú použitie SQL jazyka a ukladajú dáta v inom formáte ako relačné databázy čím čiastočne riešia ich nedostatky. Konkrétne najmä škálovateľnosť a s ňou súvisiacu výkonnosť [8]. Aj napriek tomu, že relačné databázy nie sú úplne neškálovateľné - sú ťažko škálovateľné, nerelačné databázy to značne uľahčujú [10].

Medzi hlavné výhody použitia NoSQL patrí rýchle zapisovanie a čítanie dát, podpora veľkokapacitného úložiska, ľahká rozšíriteľnosť a nízka cena [4]. Na rozdiel od relačných databáz väčšina NoSQL systémov nepodporuje ACID vlastnosti - atomicitu (atomicity), konzistentnosť (consistency), izolovanosť (isolation) a trvácnosť (durability) [8]. 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) [8]. 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 No-SQL [10]:

- kľúč-hodnota databázy - dáta sú reprezentované párom obsahujúcim unikátny kľúč (alebo viaceré kľúče) a samotnú hodnotu,
- stĺpcovo-orientované databázy - dáta ukladajú a spracovávajú po stĺpcoch namiesto riadkov,
- dokumentové databázy - ukladajú dáta v podobe dokumentov (napríklad XML, JSON, BSON) s unikátnym kľúčom, ktoré je možné zoskupovať do kolekcií,
- grafové databázy - dáta sú ukladané v podobe grafov pozostávajúcich z uzlov a hrán, ktoré reprezentujú jednotlivé dátové elementy a vzťahy medzi nimi.

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 dokumentovú databázu, čiže 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 [10].

Aj napriek tomu, že sa zaraďuje medzi nerelačné databázy, nesie niektoré prvky relačných databáz [10]. Poskytuje vlastný query jazyk MongoDB Query Language, sekundárne indexy a zaručuje silnú konzistenciu [2]. 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 (inšancií servera) tak aby bolo zachované poradie dát. Nevýhodou tejto databázy je, že môže byť nespoľahlivá a indexovanie môže zaberať veľkú časť RAM.

Práca s MongoDB

MongoDB ukladá dokumenty vo formáte BSON, ktorý je binárnou reprezentáciou formátu JSON [7]. Príklad dokumentu na uloženie detailov objednávky by mohol vyzeráť nasledovne:

```
{  
  order_number: 123456,  
  email: "jozko.m@eshop.com",  
  created: new Date("11/16/2022"),  
  items: ["orange juice", "apple juice"],  
  status: "ordered"  
}
```

Databáza podporuje CRUD operácie nad dátami [7]:

- vytvorenie dokumentu - `db.<collection>.insertOne()/insertMany();`,
- čítanie dokumentu - `db.<collection>.find();`,
- aktualizácia dokumentu - `db.<collection>.updateOne()/updateMany()/replaceOne();`,
- mazanie dokumentu - `db.<collection>.deleteOne()/deleteMany();`.

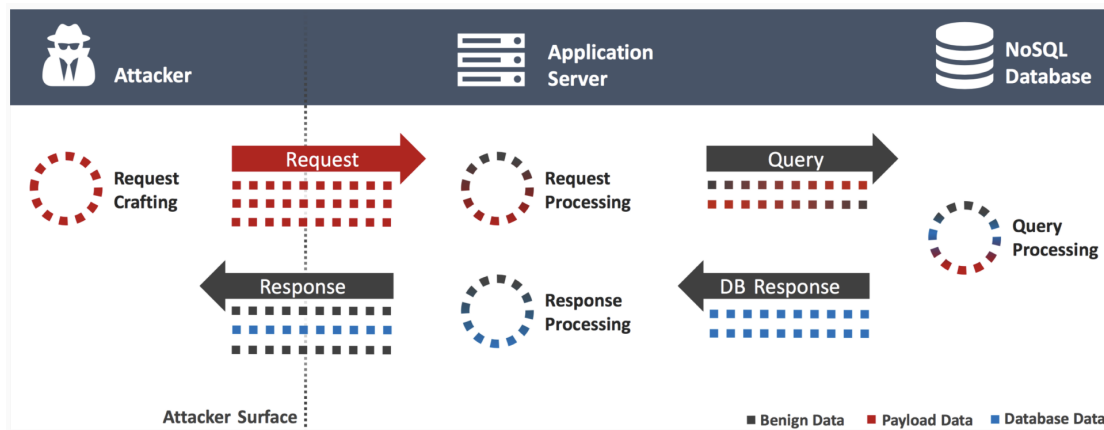
2.2 NoSQL injection útok

Aj napriek tomu, že NoSQL databázy nevyužívajú tradičnú SQL syntax stále sú zraniteľné na injection útoky [3]. 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 ošetrované alebo priamo do URL. V závislosti od použitého NoSQL API a dátového modelu sa môže škodlivý kód vykonávať na databázovej alebo aplikačnej vrstve [3]. 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 [3]. 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

Zraniteľnosť, ktorá útok na databázu umožňuje je nesprávne ošetrovaný alebo neošetrovaný 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 [5]. 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 [3]. Najčastejšie je takýto vstup podsunutý a spracovaný cez operátor \$where, ktorý predstavuje jednoduchý filter [3]. Pokiaľ je tento spôsob ošetrovaný ďalšou možnosťou pre útočníkov je nahradiť samotný operátor \$where PHP premennou nazvanou rovnako, ktorá vykoná požadovaný JavaScript kód[3].



Obr. 2.1: Priebeh NoSQL injection útoku [3]

2.2.2 Zabezpečenie

Keďže zraniteľnosť vzniká práve z dôvodu nesprávne ošetrovaného vstupu jedným z hlavných riešení je práve jeho ošetrovanie. To môžeme dosiahnuť napríklad prostredníctvom validácie vstupu, čo znamená, že overujeme vstup už priamo v mieste zadávania (limitácia znakov vstupu) [5]. 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 [5]. Pri ošetrovaní je vhodnejšie použiť overené knižnice namiesto vlastných funkcií, 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 [9].

Okrem ošetrovania vstupu je odporúčaná aj izolácia privilegií, bezpečnostné skenovanie, testovanie aplikácie a ďalšie [9]. Pri detekcii sa odporúča implementácia WAF (Web application firewalls), IDS (Intrusion detection systems), monitorovanie aktivity dát a iné [9].

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 [1].

Jedným z prípadov, kedy došlo k NoSQL injection útoku bol prípad z roku 2016. Útočníkom sa podarilo získať a predáť 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.[6]

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 prejdienie priečinku
4. Spustenie `npm start`
5. Prejdienie 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. Prejdienie 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

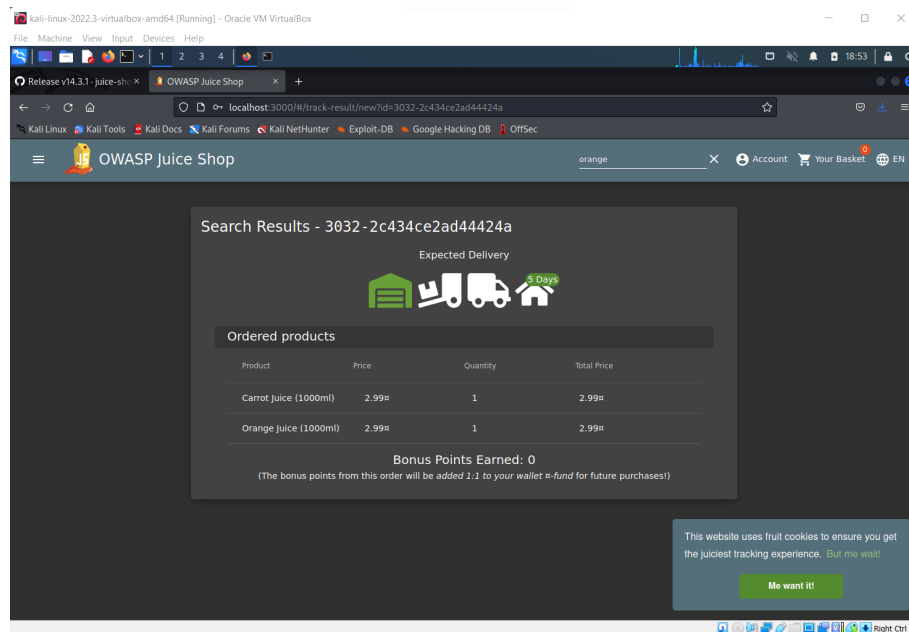
Zraniteľnosť NoSQL, ktorú sme pri tomto útoku zneužili sa nachádzala na podstránke na sledovanie objednávok. Aby sme sa k tejto podstránke dostali, bolo potrebné registrovať sa ako používateľ a vytvoriť objednávku. Registrovali sme sa pod emailom test@nosql.com a vytvorili objednávku s dvoma džúsmi.

Po dokončení objednávky sa nám zobrazila obrazovka s možnosťou jej sledovania, prešli sme teda na obrazovku sledovania objednávok. 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.

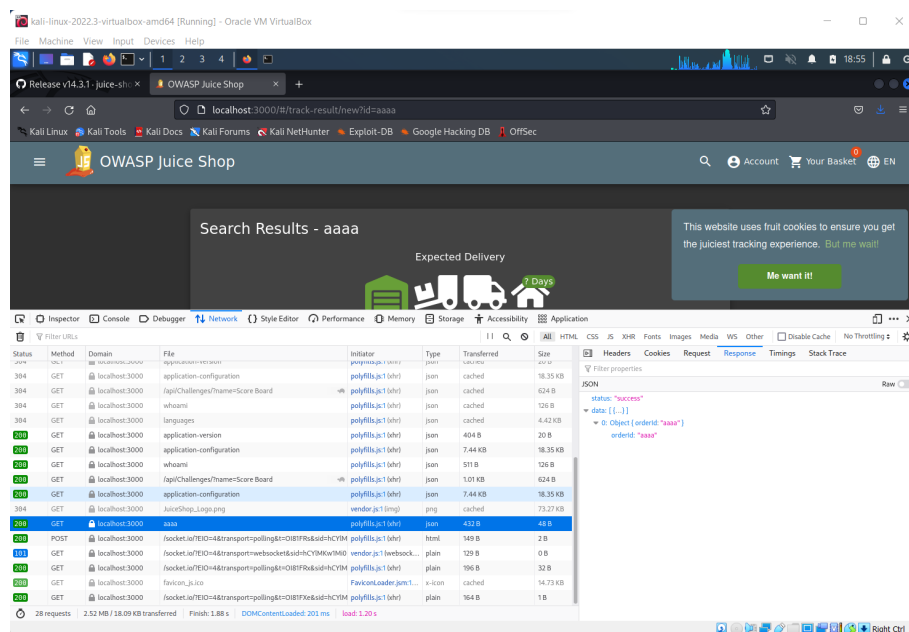
Keďže ide o podstránku, ktorá ma zobrazíť objednávku s id zadaným v podobe parametru, query pravdepodobne vyberá objednávky z kolekcie objednávok na základe tohto parametru. Úlohou bolo teda vytvoriť takú časť query a vložiť ju do parametru id aby vrátilo všetky objednávky z kolekcie.

Skúšali sme rôzne zápisy ako napríklad "{> ' ' }", "' || 1==1", "' 1==1 })//", "' 1==1// " a ďalšie. Nakoniec sa nám exfiltráciu podarilo dosiahnuť vstupom "' || 1==1 || ' ". Pri vytváraní tejto časti query sme sa inšpirovali verejne dostupnými známymi škodlivými vstupmi ako napríklad https://github.com/DevanshRaghav75/NoSQL_injection_stuff/blob/main/README.mdpayloads.

Kapitola 3. Riešenie

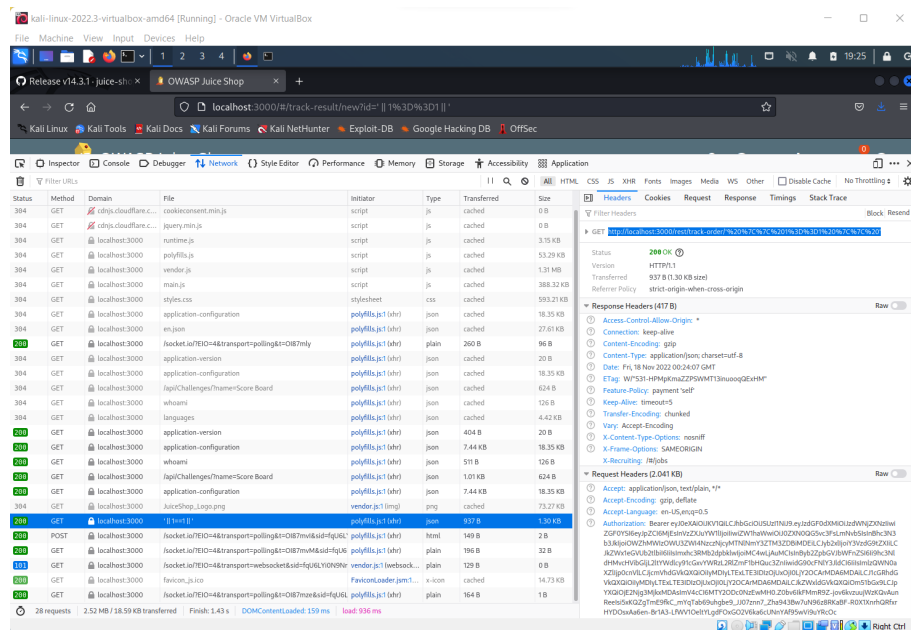


Obr. 3.1: Sledovanie objednávky

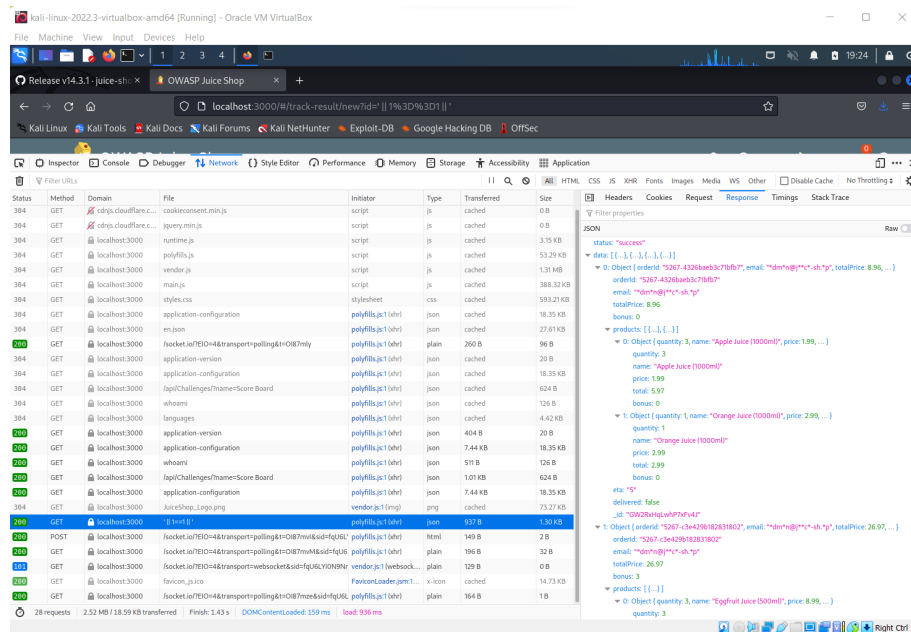


Obr. 3.2: Vstup "aaaa"

Kapitola 3. Riešenie



Obr. 3.5: Vloženie škodlivého kódu



Obr. 3.6: Exfiltrácia databázy objednávok

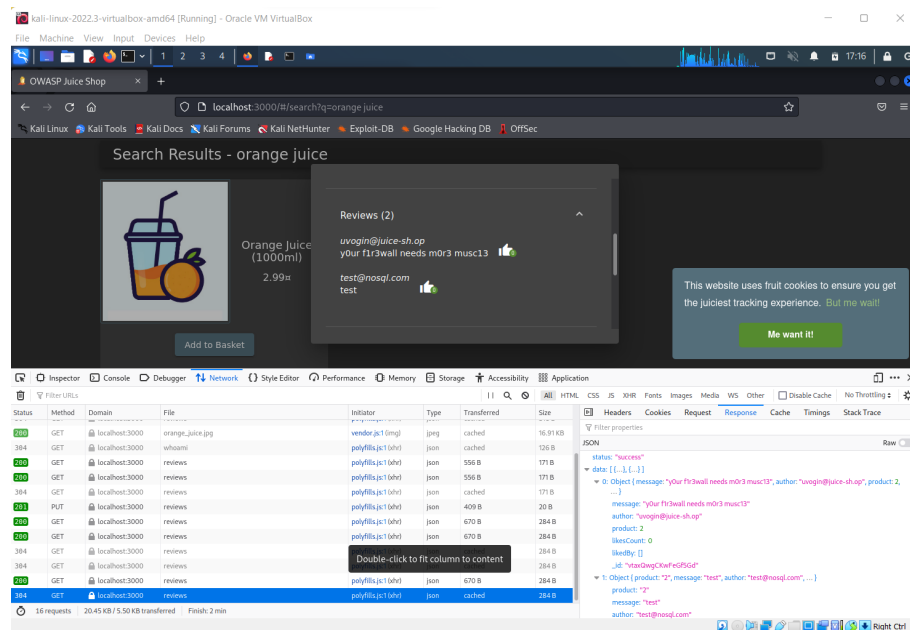
3.2.2 Dočasné zneprístupnenie aplikácie

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).

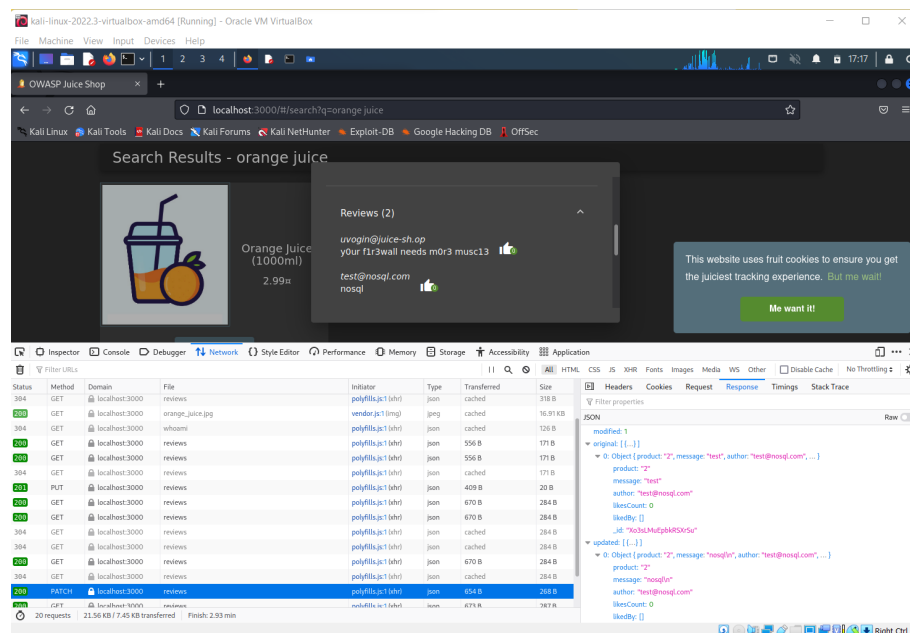
Následne sme sa pozreli do Dev Tools - Network a prezerali si hlavičky, obsah požiadaviek a odpovede na ne. Pri úprave recenzie z "text" na "nosql" sme pre požiadavku našli v hlavičke filename s cestou `/rest/products/2/reviews` (obrázok 3.13).

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 (3.15).

Kapitola 3. Riešenie

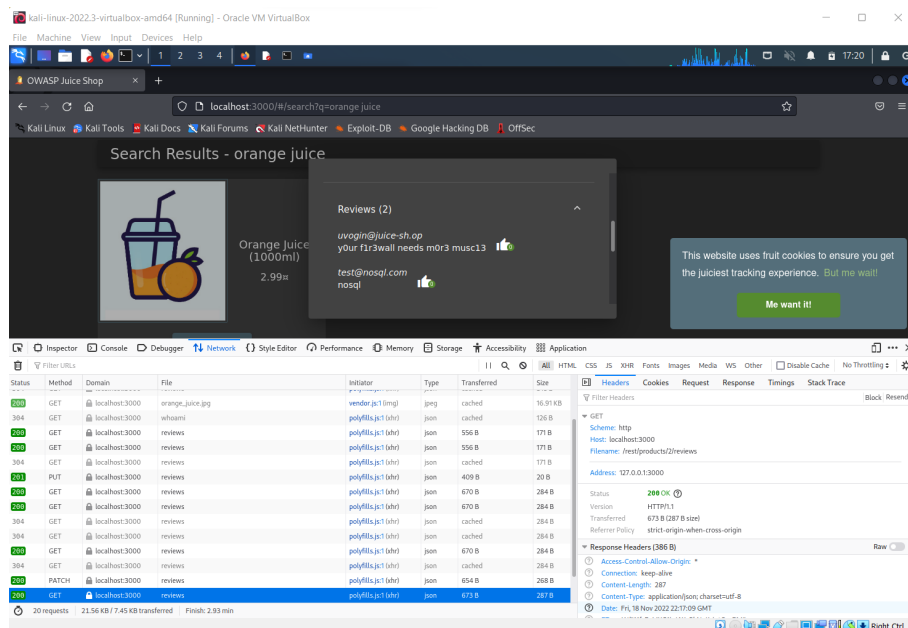


Obr. 3.7: Pridanie prvej recenzie

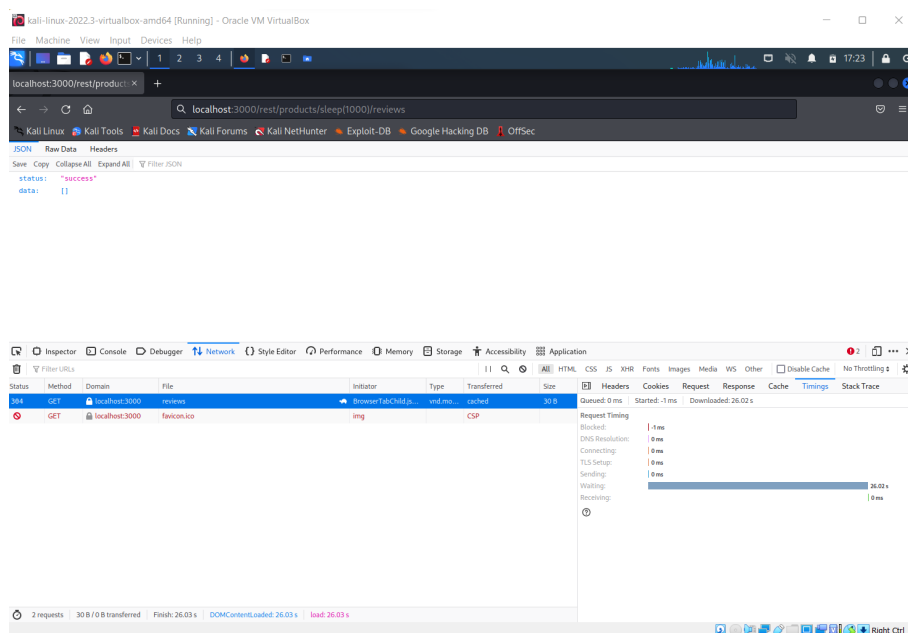


Obr. 3.8: Úprava recenzie

Kapitola 3. Riešenie



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.

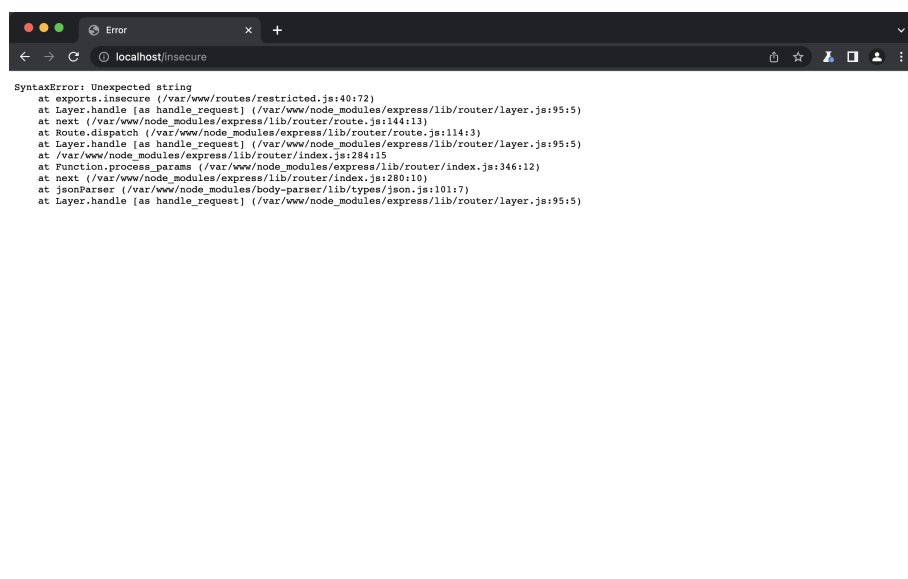
3.3.1 Prihlásenie

Najprv sme sa tak ako v predošlej aplikácii presvedčili o tom, že pole na používateľské meno je zraniteľné na NoSQL. Vytvorili sme chybu zadáním vstupu " ' ". 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.

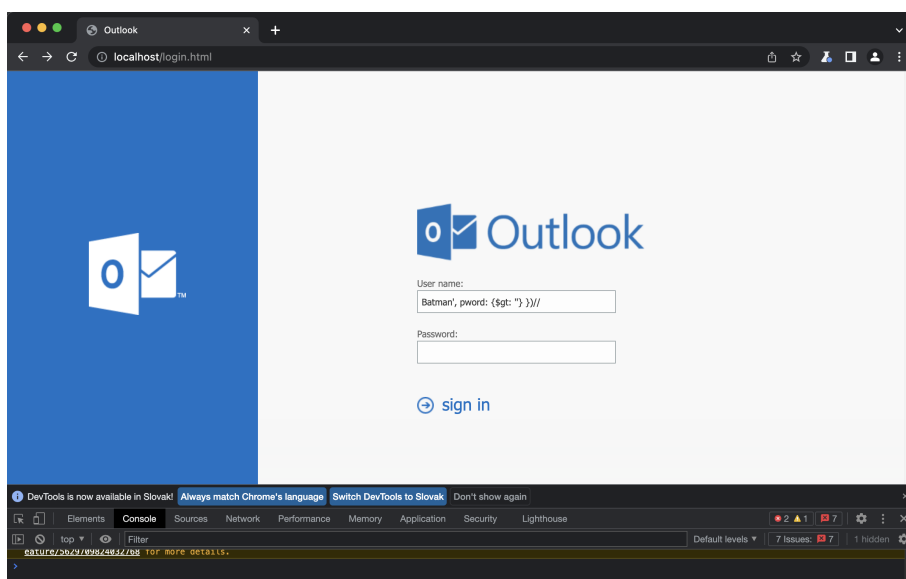
Skúšali sme rôzne vstupy ako "\${gt: '}', 'Batman', password: \${gt: '}' } //", 'Batman', password: \${gt: '}' } //". Keďže sa nám dlhší čas nepodarilo prísť na spôsob obídenia prihlásenia tak sme sa pozreli do kódu a zistili, že názov pre heslo nie je "password " ale "pword ". Po opätovnom vyskúšaní predošlých škodlivých výrazov nakoniec zafungoval výraz "Batman', pword: \${gt: '}' }) //". 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.

Kapitola 3. Riešenie

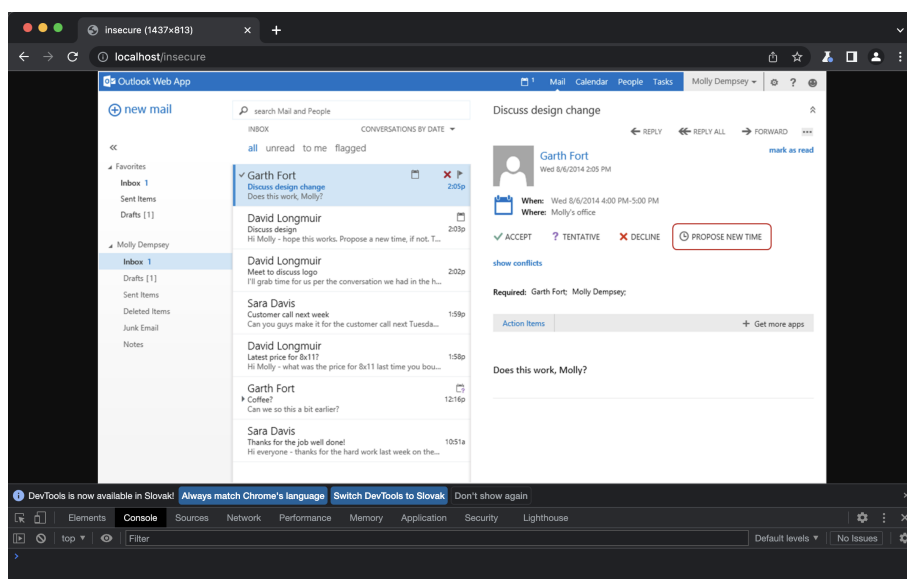


Obr. 3.11: Vyvolanie chyby

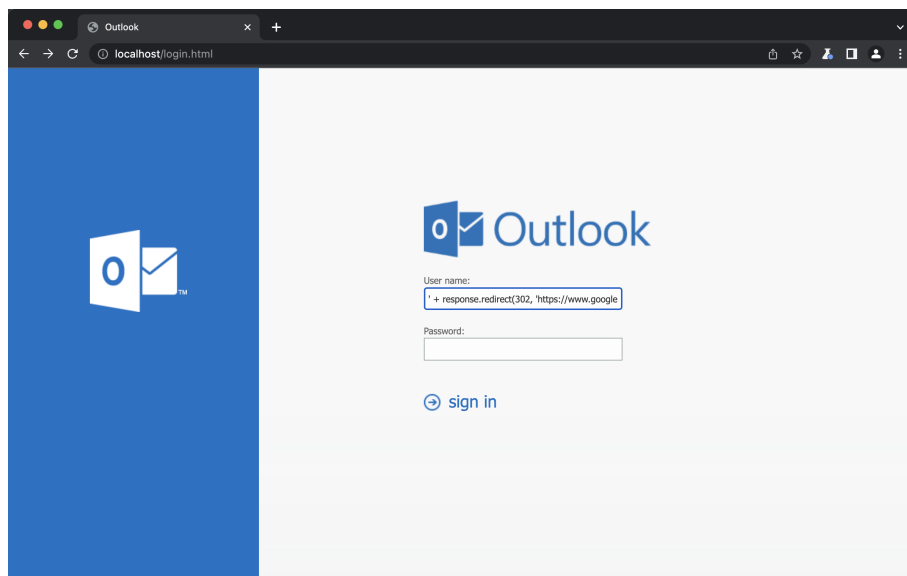


Obr. 3.12: Vloženie škodlivého kódu na prihlásenie

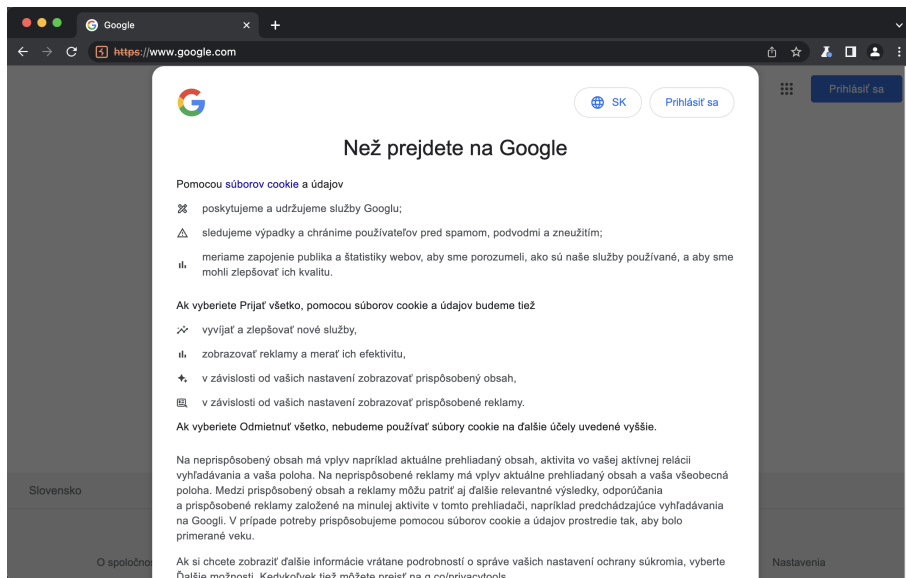
Kapitola 3. Riešenie



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 informácií môžeme predpokladať, že zraniteľný parameter má požadovaný tvar x-x, kde časti x sa skladajú z niekoľkých písmen a čísel. Regex by preto mohol vyzeráť 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 vyzeráť: `/([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 "/", "{ }", ":" na príklad 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] DB-Engines. *DB-Engines Ranking*. URL: <https://db-engines.com/en/ranking> (cit. 18.11.2022).
- [2] Miguel Diogo, Bruno Cabral a Jorge Bernardino. “Consistency models of NoSQL databases”. In: *Future Internet* 11.2 (2019), s. 43.
- [3] 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).
- [4] 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.
- [5] 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.
- [6] 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).
- [7] MongoDB. *MongoDB*. URL: <https://www.mongodb.com/> (cit. 15.11.2022).

- [8] 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.
- [9] Aviv Ron, Alexandra Shulman-Peleg a Anton Puzanov. “Analysis and mitigation of NoSQL injections”. In: *IEEE Security & Privacy* 14.2 (2016), s. 30–39.
- [10] Christof Strauch, Ultra-Large Scale Sites a Walter Kriha. “NoSQL databases”. In: *Lecture Notes, Stuttgart Media University* 20.24 (2011), s. 79.