J-INTECH (Journal of Information and Technology)
Terakreditasi Kemendikbud SK No. 204/E/KPT/2022
E-ISSN: 2580-720X || P-ISSN: 2303-1425

Optimasi Serangan *Blind* Nosql Injection Dengan Pendekatan Algoritma *Binary Search*

Roby Firnando Yusuf [1], Daniel Rudiaman Sijabat[2*]

[1,2] *STIKI Malang, Teknik Informatika, Jl. Tidar 100 Malang, Jawa Timur, Indonesia*

Informasi Artikel
Diterima: 05-10-2023
Direvisi: 30-11-2023
Diterbitkan: 22-12-2023

Kata Kunci
Informasi; Blind NoSQL Injection, NoSQL, Binary Search, Database Security

***Email Korespondensi:**
daniel223@stiki.ac.id

Abstrak
NoSQL Injection adalah salah satu jenis serangan pada Database Management System (DBMS) NoSQL. Serangan ini dapat dimaanfaatkan oleh *attacker* dengan cara mengirim *request arbitrary code* ke server database. Apabila server memberikan respon *error query* atau query tidak valid, *attacker* akan melakukan manipulasi pada *query*. Proses melakukan Blind NoSQL Injection itu cukup rumit. Akibatnya, Pentester seringkali membutuhkan waktu yang lama untuk dapat memperoleh informasi dan menembus server database. Berdasarkan permasalahan tersebut, penelitian ini akan memberikan solusi dengan mengembangkan tool untuk mengotomasi serangan Blind NoSQL Injection. Hasil penelitian ini menunjukkan bahwa pengembangan tool *exploit* dapat meningkatkan kinerja dan efisiensi. Algoritma binary search memiliki keunggulan waktu yang lebih singkat pada parameter *runtime* dibandingkan linear search, sehingga binary search lebih efektif digunakan. Selain itu, pendekatan mitigasi dengan sanitasi dan validasi input pada setiap key object terbukti efektif dalam mencegah serangan NoSQL Injection.

Abstract
NoSQL Injection is one type of attack on the NoSQL Database management system (DBMS). This attack exploits a vulnerability that allows the attacker to send arbitrary requests to the server. If the server responds to an error query or an invalid query, the attacker will manipulate the query. The process of doing Blind NoSQL Injection is complicated. As a result, Pentester often takes a long time to be able to obtain information and penetrate the database server. Based on these problems, this research will provide a solution by developing a tool to automate Blind NoSQL Injection attacks. The results of this research indicate that the development of an exploit tool can enhance performance and efficiency. The binary search algorithm demonstrates a shorter runtime compared to linear search, making it a more effective choice. Additionally, the mitigation approach involving sanitization and validation of input for each key object has proven to be effective in preventing NoSQL Injection attacks.

1. Pendahuluan

Semua aplikasi modern baik berbasis *web*, *desktop* maupun *mobile* menggunakan *database* untuk menyimpan informasi. *Database management system* (DBMS) yang umum digunakan di zaman modern ini adalah SQL (*structured query language*) dan NoSQL (*Not Only SQL*). Menurut data dari DB-Engines Ranking, penggunaan DBMS NoSQL mengalami peningkatan. Banyak aplikasi modern yang menggunakan *database* NoSQL karena menawarkan performa dan keunggulan dalam *scaling* yang baik. NoSQL menyimpan data secara tidak tabular dan data yang bersifat non-relasional serta memiliki sintaks yang berbeda dengan SQL.

Perkembangan teknologi ini tidak diimbangi dengan peningkatan keamanan. Berdasarkan data OWASP Top 10 2017, serangan terbanyak pada survei tahun 2017 adalah serangan berbasis injeksi seperti NoSQL, SQL, OS dan LDAP (OWASP Top Ten, n.d.). Pada tahun 1998, Carlo Strozzi menemukan *database* non-relasional sebagai pengembangan dari *database* SQL. Pemilihan sistem basis data NoSQL dipengaruhi oleh fleksibilitas data, kemampuan beroperasi dengan cepat pada data dan pengguna yang besar, serta peningkatan performa untuk memenuhi kebutuhan pengolahan data yang cepat. Meskipun NoSQL memiliki keunggulan, namun juga rentan terhadap serangan NoSQL *Injection*, seperti PHP *Tautologies injection*, Union *queries*, Javascript *Injection*, *Piggybacked queries*, dan *Origin violation*. (Shachi et al., 2021).

Blind NoSQL *Injection* (*Blind* NoSQLi) adalah jenis serangan injeksi yang rumit, membutuhkan waktu lama untuk menembus *server database*. Serangan ini bergantung pada pengiriman *query* SQL yang memaksa aplikasi memberikan nilai balikan yang berbeda berdasarkan kebenaran atau ketidakbenaran *query* (*TRUE* atau *FALSE*). Dengan penelitian ini, proses injeksi *syntax* dihasilkan secara otomatis. Tujuan dari penelitian ini adalah untuk merancang dan mengimplementasikan otomatisasi *Blind* NoSQL *injection* menggunakan algoritma pencarian *linear* dan *binary*, serta menganalisis *runtime* untuk masing-masing algoritma yang diimplementasikan.

2. Metode Penelitian

2.1 Studi Kepustakaan

Penelitian ini menggunakan metode studi kepustakaan untuk mengumpulkan dan menganalisis informasi dari sumber-sumber yang relevan dan terpercaya. Penelitian dilakukan dengan menelaah dan/atau mengekplorasi beberapa jurnal serta sumber-sumber data dan atau informasi lainnya yang dianggap relevan dengan penelitian.

2.2 Data

Data yang digunakan pada penelitian ini adalah laboratorium localhost yang dibangun menggunakan NodeJS dan MongoDB. Data yang digunakan untuk melakukan pencarian adalah representasi hexadecimal dari karakter ascii pada rentang dari 0x20 hingga 0x7f.

2.3 Test Matrix

Untuk mengukur kinerja sistem yang sedang dibangun, hasil data dari setiap algoritma dibandingkan dalam bentuk waktu saat mencari setiap huruf yang benar. Dalam formula untuk menemukan waktu di bawah ini, T0 merupakan waktu awal ketika proses pencarian huruf dimulai, sedangkan T1 adalah waktu akhir ketika huruf yang dicari ditemukan. *Parameter* waktu dihitung menggunakan formula berikut:

$$\Delta T = T_1 - T_0 \qquad (1)$$

2.4 Desain Algoritma

*2.4.1 S*kema Serangan *Blind NoSQL Injection*

Dalam penelitian ini, jenis serangan yang digunakan adalah *Illegal / Logically Incorrect Queries*. Serangan ini memanfaatkan kesalahan sintaksis atau kesalahan logika sehingga aplikasi situs *web* akan mengembalikan halaman *error*. Injeksi dilakukan melalui *vulnerability* pada halaman situs *web* yang diambil dari *form login*. Pada *form* tersebut, jika dilakukan *view-source* di browser, akan terlihat elemen form yang dapat digunakan untuk otomatisasi. Terdapat tiga elemen dalam <form method = "post"> yang perlu digunakan, yaitu "*username*", "*password*", dan tombol dengan *class* tertentu.

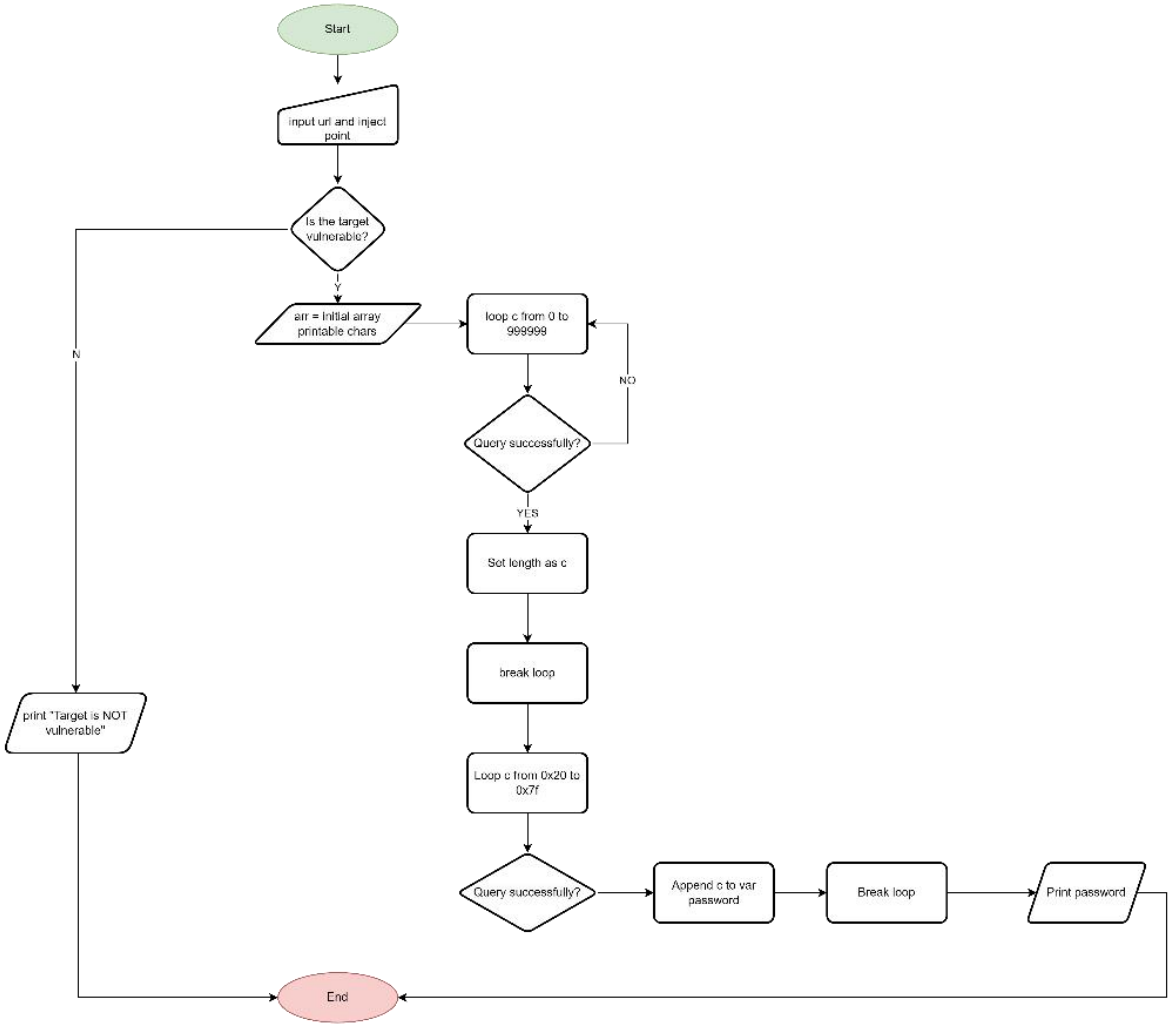
```
<div class="row justify-content-center">
    <div class="col-md-6">
        <form method="POST" action="/auth/login">
            <div class="form-group">
                <label for="email">Username</label>
                <input type="text" id="email" name="user" class="form-control" required>
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="pass" class="form-control" required>
            </div>
            <br>
            <button type="submit" class="btn btn-primary btn-block">Login</button>
        </form>
    </div>
</div>
```

Gambar 1. Form Login

Setelah mendapatkan elemen-elemen ini, *tool exploit* yang dibangun akan mengirim *payload* pada elemen-elemen tersebut menggunakan *library* requests yang terdapat dalam *package* Python. HTTP Request yang digunakan adalah jenis post dan melakukan *pentesting* ke situs *web* melalui halaman *login*. *Query* yang dihasilkan oleh *tool exploit* bertujuan untuk akhirnya menemukan *password* di *database*.

2.4.2 Algoritma *Linear Search*

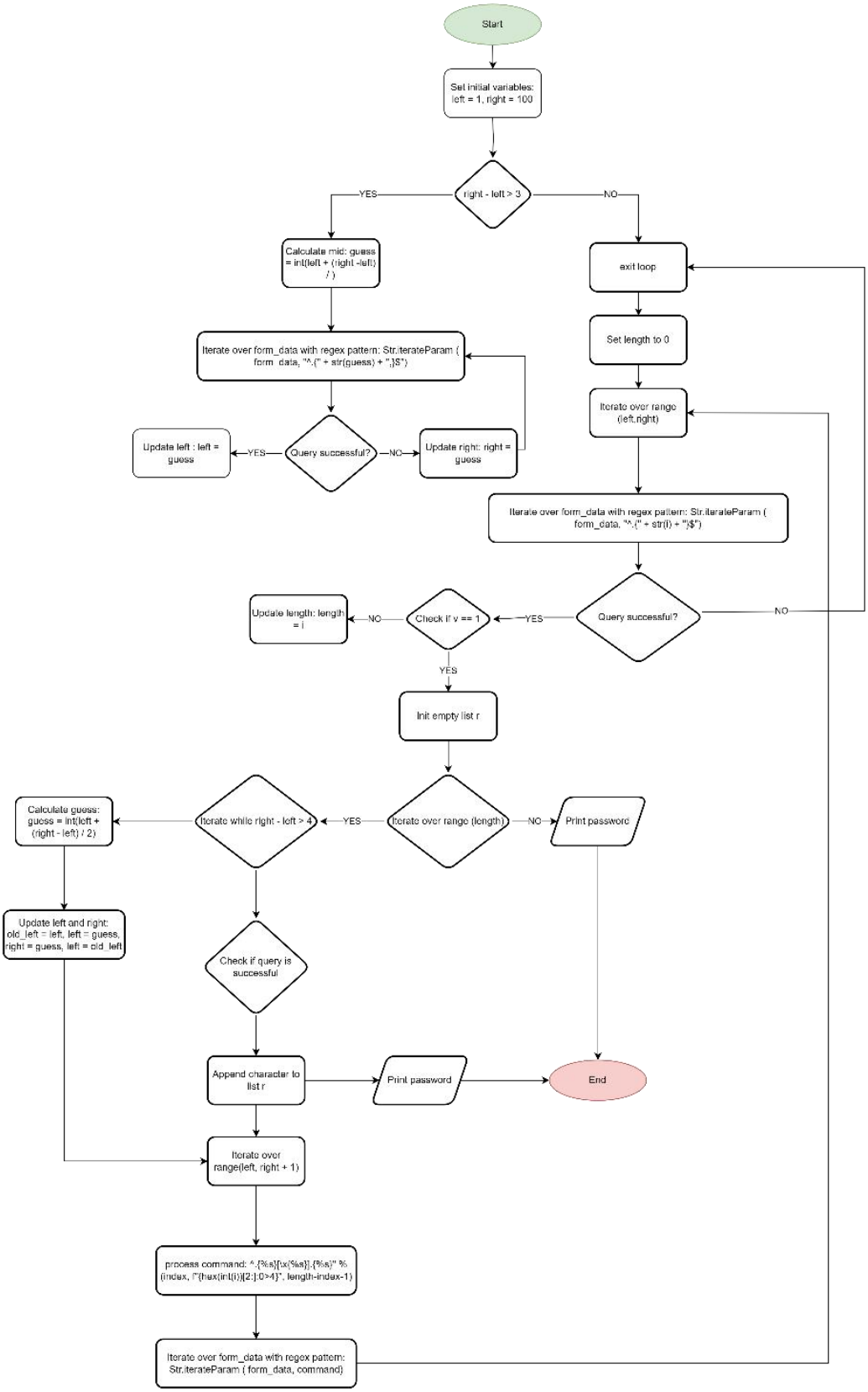
Skema otomatisasi serangan menggunakan algoritma pencarian linear dijelaskan melalui flowchart pada Gambar 2. Algoritma ini mencari setiap huruf yang benar dalam *database*, kemudian melakukan pencarian secara berulang sebanyak 100 kali. Pencarian ini menggunakan karakter *ASCII* yang direpresentasikan dalam heksadesimal secara berurutan mulai dari 0x20 hingga 0x7f.



Gambar 2. Flowchart Linear Search

2.4.3 Algoritma *Binary Search*

Algoritma ini mencari setiap huruf dalam data dengan membagi data menjadi dua bagian. Selanjutnya, dilakukan perbandingan apakah hasilnya berada di atas atau di bawah, hingga diperoleh semua hasil dari setiap huruf yang disusun menjadi sebuah kata.



Gambar 3. Flowchart Binary Search

3. Hasil dan Pembahasan

3.1 Pentesting Blind NoSQL Injection

Pada tahap ini identifikasi celah keamanan dilakukan pada lab simulasi menggunakan beberapa *malicious syntax* berikut.

3.1.1 Not Equal Expression

Syntax	**:** [$ne]
Positive Result	**:** Tampil Dashboard Admin
Negative Result	**:** Kembali ke halaman login

Sintaksis ini digunakan untuk melakukan perbandingan tidak sama (*not equal*) dalam pengambilan data dari database. Jika sintaksis ini berhasil dieksploitasi, maka hasilnya akan menampilkan Dashboard Admin, yang menandakan keberhasilan dalam mengakses hak akses admin secara tidak sah. Tahap ini bertujuan untuk menguji kerentanan terhadap serangan Blind NoSQL Injection dengan menggunakan sintaksis "*Not Equal Expression*". Dengan sintaksis di atas kondisi berhasil dimanipulasi menjadi TRUE sehingga aplikasi lab simulasi menampilkan halaman dashboard dan proses injeksi dapat dilanjutkan ke tahap selanjutnya.

3.1.2 Greater Than Expression

Syntax	**:** [$gt]
Positive Result	**:** Tampil Dashboard Admin
Negative Result	**:** Kembali ke halaman login

Sintaksis ini digunakan untuk melakukan perbandingan lebih besar dari (*greater than*) dalam pengambilan data dari database. Jika sintaksis ini berhasil dieksploitasi, maka hasilnya akan menampilkan Dashboard Admin, yang menandakan keberhasilan dalam mengakses hak akses admin secara tidak sah. Namun, jika eksploitasi tidak berhasil, pengguna akan diarahkan kembali ke halaman login, menunjukkan bahwa akses tidak sah tidak berhasil dilakukan. Tahap ini bertujuan untuk menguji kerentanan terhadap serangan Blind NoSQL Injection dengan menggunakan sintaksis "*Greater Than Expression*". Dengan sintaksis di atas kondisi berhasil dimanipulasi menjadi TRUE sehingga aplikasi lab simulasi menampilkan halaman dashboard dan proses injeksi dapat dilanjutkan ke tahap selanjutnya.

3.1.3 Regex Expression

Syntax	**:**1.^.{x}[\x{**hexvalue**}-\x{ **hexvalue** }].{y} 2. ^.{x}$ 3.^(?!.*abcdefghijklm) ^%s([\x{**hexvalue**}-\x{**hexvalue**}])
Positive Result	**:** Tampil Dashboard Admin
Negative Result	**:** Kembali ke halaman login

Pada tahap ini, dilakukan identifikasi celah keamanan pada lab simulasi menggunakan sintaksis "*Regex Expression*". Terdapat tiga bentuk sintaksis yang digunakan, yaitu:

Sintaksis pertama ^.{x}[\x{hexvalue}-\x{hexvalue}].{y} digunakan untuk melakukan pencarian menggunakan regular expression dengan batasan karakter tertentu yang ditentukan oleh nilai x dan y, serta rentang karakter dalam bentuk heksadesimal.

Sintaksis kedua ^.{x}$ digunakan untuk melakukan pencarian menggunakan regular expression dengan batasan karakter tepat sejumlah x.

Sintaksis ketiga ^(?!.*abcdefghijklm) ^%s([\x{hexvalue}-\x{hexvalue}]) digunakan untuk melakukan pencarian menggunakan regular expression dengan pemisahan karakter tertentu yang tidak mengandung substring "abcdefghijklm" di dalamnya, serta rentang karakter dalam bentuk heksadesimal.

Jika salah satu dari sintaksis tersebut berhasil dieksploitasi, maka hasilnya akan menampilkan Dashboard Admin, yang menandakan keberhasilan dalam mengakses hak akses admin secara tidak sah. Namun, jika eksploitasi tidak berhasil, pengguna akan diarahkan kembali ke halaman login, menunjukkan bahwa akses tidak sah tidak berhasil dilakukan. Tahap ini bertujuan untuk menguji kerentanan terhadap serangan Blind NoSQL Injection dengan menggunakan sintaksis "*Regex Expression*". Dengan sintaksis di atas kondisi berhasil dimanipulasi menjadi TRUE sehingga aplikasi lab simulasi menampilkan halaman dashboard dan proses injeksi dapat dilanjutkan ke tahap selanjutnya.

3.2 Hasil Pengujian

Pengujian dilakukan dengan dibuatnya lima model skenario yang akan dijalankan dalam simulasi. Skenario ini didasarkan pada jumlah dan model string yang diproses dalam proses eksfiltrasi *field password*. Tabel 1 berikut menunjukkan beberapa contoh skenario yang dimaksud.

Tabel 1. Skenario Uji Coba

Skenario	Panjang karakter	Jenis karakter	*Output*	*Keyword*
1.	11 karakter	Alfabet *lowercase*	*Runtime* dan jumlah Iterasi	supersecret
2.	11 karakter	Alfabet *lowercase* & *uppercase*	*Runtime* dan jumlah Iterasi	SuPeRsEcR3T
3.	11 karakter	Alfabet *lowercase* dan numerik	*Runtime* dan jumlah Iterasi	ganesha1273
4.	11 karakter	Alfabet *lowercase, uppercase,* numerik dan simbolik	*Runtime* dan jumlah Iterasi	L1g7tM3Up!_
5.	11 karakter	Alfabet *uppercase*	*Runtime* dan jumlah Iterasi	UCANTBEATME

Percobaan pada masing-masing simulasi dilakukan sebanyak 10 kali dengan tujuan untuk mengetahui kekuatan masing-masing algoritma. Pengujian injeksi dilakukan pada komputer lokal dengan mematikan aplikasi-aplikasi lain, untuk mengurangi pengaruh proses-proses yang lain terhadap waktu yang dibutuhkan oleh sistem untuk melakukan proses pengujiannya.

Tabel 2. Hasil Simulasi Skenario 1

Percobaan	Keyword	Algoritma Linear		Algoritma Binary	
		Iterasi	Runtime	Iterasi	Runtime
1	supersecret	864	4.493202447891235	82	0.669348955154419
2		864	3.985246419906616	82	0.623244047164917
3		864	3.6847598552703857	82	0.6220629215240479
4		864	3.5550718307495117	82	0.663893461227417
5		864	3.4561209678649902	82	0.6046864986419678
6		864	3.392568826675415	82	0.5948975086212158
7		864	3.2472586631774902	82	0.5879745483398438
8		864	3.268374443054199	82	0.6000518798828125
9		864	3.0180246829986572	82	0.5916099548339844
10		864	3.0130066871643066	82	0.5808343887329102
Rata-Rata:		**864**	**3.511363482475281**	**82**	**0.6138604164123536**

Tabel 3. Hasil Simulasi Skenario 2

Percobaan	Keyword	Algoritma Linear		Algoritma Binary Search	
		Iterasi	*Runtime*	Iterasi	*Runtime*
1	SuPeRsEcR3T	622	2.1051366329193115	83	0.5561084747314453
2		622	2.068701982498169	83	0.5481853485107422
3		622	2.066567897796631	83	0.5388109683990479
4		622	2.0481116771698	83	0.5348448753356934
5		622	2.0660486221313477	83	0.5423941612243652
6		622	2.1343770027160645	83	0.6027331352233887
7		622	2.046624183654785	83	0.5882749557495117
8		622	2.0676398277282715	83	0.5750625133514404
9		622	2.0709469318389893	83	0.7515749931335449
10		622	2.0555660724639893	83	0.5791418552398682
Rata-Rata:		**622**	**2.0729720830917358**	**83**	**0.5817131280899048**

Tabel 4. Hasil Simulasi Skenario 3

Percobaan	Keyword	Algoritma Linear		Algoritma Binary Search	
		Iterasi	*Runtime*	Iterasi	*Runtime*
1	ganesha1273	591	1.911041498184204	85	0.5726726055145264
2		591	1.87856125831604	85	0.5661842823028564
3		591	1.8882038593292236	85	0.5650200843811035
4		591	1.8799762725830078	85	0.5452709197998047
5		591	1.885519027709961	85	0.5320241451263428
6		591	1.9054090976715088	85	0.5479273796081543
7		591	1.8862719535827637	85	0.5444211959838867
8		591	1.8908512592315674	85	0.5318973064422607
9		591	1.8874263763427734	85	0.5320703983306885
10		591	1.8802406787872314	85	0.5240228176116943
Rata-Rata:		**591**	**1.889350128173828**	**85**	**0.5461511135101318**

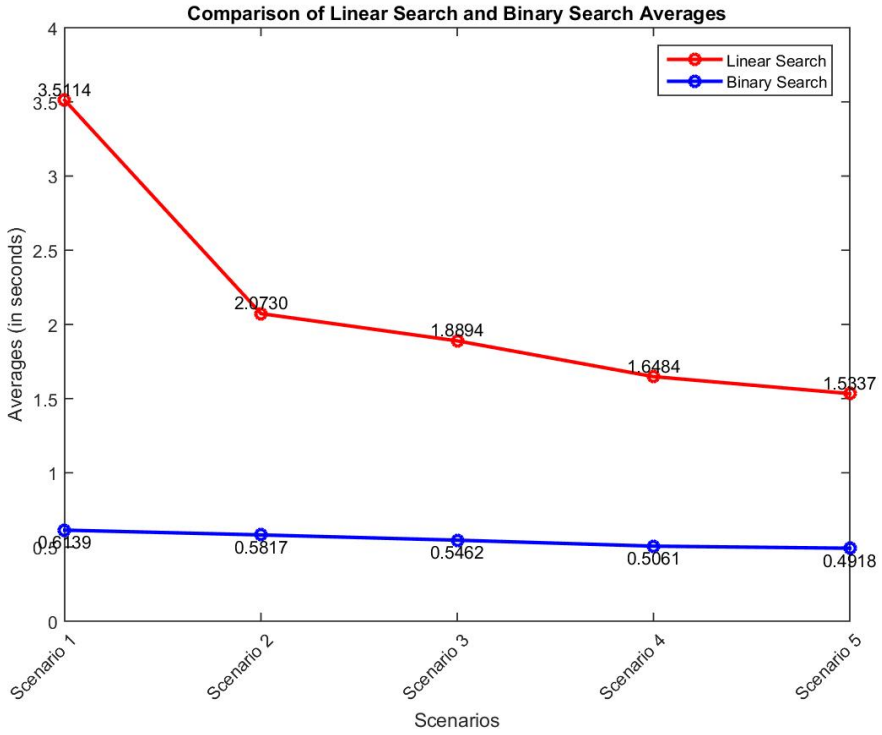
Tabel 5. Hasil Simulasi Skenario 4

Percobaan	Keyword	Algoritma Linear		Algoritma Binary Search	
		Iterasi	*Runtime*	Iterasi	*Runtime*
1	L1g7tM3Up!_	511	1.6447274684906006	81	0.5170412063598633
2		511	1.6487598419189453	81	0.5148077011108398
3		511	1.652939796447754	81	0.5009381771087646
4		511	1.636958122253418	81	0.5091135501861572
5		511	1.6545913219451904	81	0.506242036819458
6		511	1.6323118209838867	81	0.5057854652404785
7		511	1.65171480178833	81	0.49666523933410645
8		511	1.648987054824829	81	0.4959414005279541
9		511	1.6578466892242432	81	0.5101847648620605
10		511	1.6547000408172607	81	0.5037951469421387
Rata-Rata:		**511**	**1.6483536958694458**	81	**0.5060514688491822**

Tabel 6. Hasil Simulasi Skenario 5

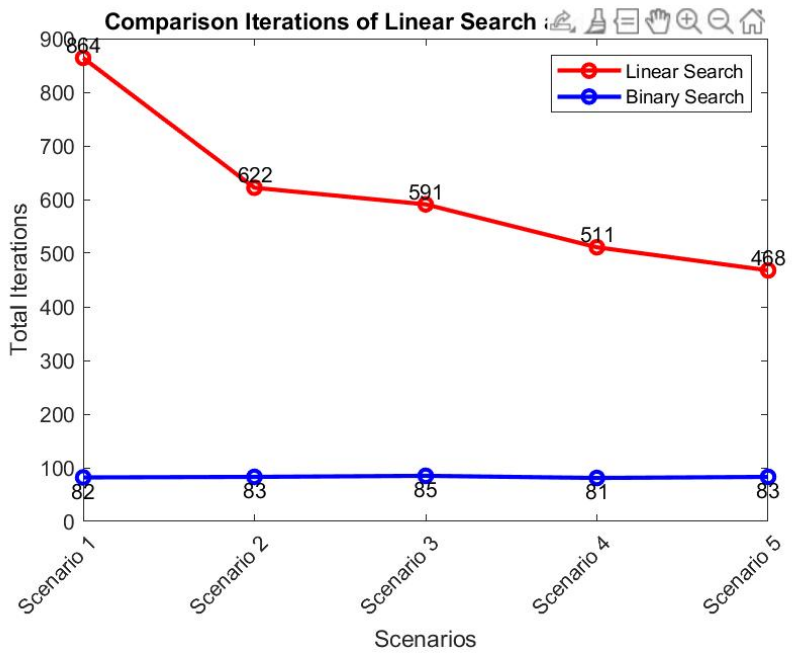
Percobaan	Keyword	Algoritma Linear		Algoritma Binary Search	
		Iterasi	*Runtime*	Iterasi	*Runtime*
1	UCANTBEATME	468	1.5077829360961914	83	0.4994790554046631
2		468	1.5229575634002686	83	0.488433837890625
3		468	1.482231855392456	83	0.5021708011627197
4		468	1.561063528060913	83	0.48605990409851074
5		468	1.6310975551605225	83	0.4955272674560547
6		468	1.5399210453033447	83	0.48767852783203125
7		468	1.5607295036315918	83	0.4870789051055908
8		468	1.5024855136871338	83	0.49344491958618164
9		468	1.503544807434082	83	0.4859957695007324
10		468	1.52557373046875	83	0.4925265312194824
Rata-rata:		**468**	**1.5337388038635253**	83	**0.4918395519256592**

3.3 *Test Results Analysis*



Gambar 4. Hasil Perbandingan Runtime

Berdasarkan grafik di atas ditunjukkan hasil perbandingan *runtime* antara algoritma *Linear* dan *Binary Search*. Pada grafik tersebut ditunjukkan bahwa algoritma *Binary Search* membutuhkan waktu yang paling sedikit dibanding algoritma *Linear Search*.



Gambar 5. Hasil Perbandingan Iterasi

Pada kedua grafik di atas dapat dilihat perbedaan waktu dan iterasi yang dibutuhkan untuk melakukan serangan Blind NoSQL Injection pada MongoDB cukup signifikan. Pada *parameter runtime* dan iterasi apabila angka yang didapat lebih kecil maka semakin baik algoritma tersebut, sehingga dapat disimpulkan bahwa Algoritma *Binary Search* lebih baik dalam *parameter runtime* dan jumlah iterasi dibandingkan dengan Algoritma *Linear Search*. Dalam eksperimen yang dilakukan sepuluh kali agar diperoleh nilai rata-rata dari proses pengujian berdasarkan *parameter runtime*. Hal ini disebabkan oleh waktu respons proses *localhost* yang berbeda yang mempengaruhi hasil saat mengambil data. Untuk mengurangi efek ini, proses pengumpulan data dilakukan ketika sistem operasi dinyalakan tanpa ada proses lain selain menjalankan algoritma eksperimental dan aplikasi lab sebagai target.

3.4 *Mitigasi Terhadap MongoDB Injection*

Kesuksesan eksploitasi vulnerability ini karena pada kasus ini tidak adanya sanitasi pada parameter *request*, sehingga memungkinkan *attacker* melakukan injeksi arbitrary JS code pada aplikasi lab simulasi.

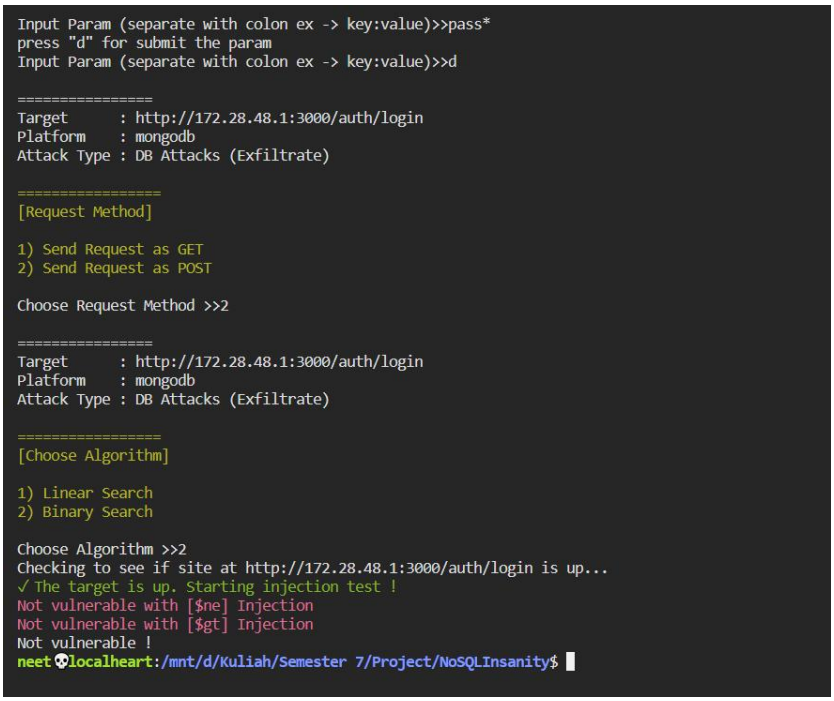
```
query = { username: req.body.user, password: req.body.pass }
const user = await User.findOne(query);
```

Dalam hal ini, solusi yang dapat diimplementasikan adalah dengan melakukan sanitasi sebelum *request dipassing* ke dalam *Object Data Model* (ODM) atau *Object-Relational Mapping* (ORM). Berikut *logic* sintaksis yang dapat diimplementasi sebagai solusi.

```
function sanitize(v) {
   if (typeof v === 'string') {
      return v.replace(/'/g, '');
   } else if (v instanceof Object) {
      for (var key in v) {
         if (/^\$/.test(key)) {
            delete v[key];
         } else {
            v[key] = sanitize(v[key]);
```

```
            sanitize(v[key]);
        }
    }
  }
  return v;
}
```

Hasil pengujian setelah dilakukan mitigasi terhadap serangan NoSQL Injection. Terlihat bahwa setelah penerapan sanitasi, serangan NoSQL Injection berhasil dicegah dan website tidak lagi vulnerable dapat dilihat pada gambar 6 berikut.



Gambar 6. Tampilan setelah dilakukan sanitasi terhadap vulnerability

Dengan menerapkan langkah ini, data yang diterima oleh sistem akan melalui proses sanitasi yang efektif, sehingga mengurangi risiko serangan NoSQL *Injection* yang dapat mengakibatkan kerugian yang serius. Selain itu, penting untuk menghindari melakukan passing langsung (*directly passing*) *request objects* pada ODM atau ORM.

```
const user = await collection.findOne(req.body); // Bad Practice
```

Langkah ini bertujuan untuk meminimalkan celah keamanan yang dapat dimanfaatkan oleh serangan NoSQL Injection. Sebagai alternatifnya, dianjurkan untuk melakukan validasi dan pemanggilan data secara eksplisit sebelum mengirimnya ke ODM atau ORM, sehingga memperkuat lapisan keamanan dan melindungi aplikasi dari serangan NoSQL *Injection*. Berikut kode yang dianjurkan dibandingkan melakukan *passing object request* secara langsung.

```
const user = await collection.findOne(sanitize({ username: req.body.username, password: req.body.password }));
```

Langkah-langkah pencegahan ini dirancang untuk memitigasi risiko dan menjaga keamanan database MongoDB dari ancaman serangan yang dapat membahayakan integritas dan kerahasiaan data. Implementasi langkah-langkah ini diharapkan dapat memberikan perlindungan yang kuat terhadap serangan NoSQL *Injection* dan menjaga keamanan data yang tersimpan di dalamnya.

4. Kesimpulan

Berdasarkan hasil penelitian dan analisa data, dapat disimpulkan bahwa terdapat pengaruh positif dan signifikan terhadap penelitian yang berjudul "Optimasi Serangan Blind NoSQL Injection dengan Pendekatan Algoritma Binary Search". Analisis data dan optimasi menggunakan pendekatan algoritma *Binary Search* menghasilkan kesimpulan sebagai berikut. Penggunaan Algoritma Binary Search dalam pemrosesan serangan Blind NoSQL Injection pada database MongoDB melalui tool NoSQLInsanity dengan lab simulasi yang telah disiapkan khusus sebagai objek observasi dapat meningkatkan kinerja dan efisiensi. Terdapat perbedaan yang signifikan antara Algoritma *Linear* dan *Binary Search* dalam *parameter runtime* dan iterasi. Algoritma *Binary Search* lebih efisien dan memiliki waktu dan jumlah iterasi yang lebih singkat dibandingkan dengan Algoritma *Linear Search*. Rekomendasi mitigasi adalah dilakukan sanitasi dan validasi terlebih dahulu untuk mencegah serangan NoSQL Injection pada *sisi code*.

5. Referensi

(n.d.). Retrieved from V8: https://v8.dev/

(n.d.). Retrieved from NodeJS: https://nodejs.org/en/about/

Fitri, M. O. (2013). TREND PENGGUNAAN NOSQL UNTUK BASIS DATA.

Guptaa, S., Singha, N. K., & Tomara, D. S. (2018). Analysis Of NoSQL Database Vulnerabilities. 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT).

Hou, B., Shi, Y., Qian, K., & Tao, L. (2017). Towards Analyzing MongoDB NoSQL Security and Designing Injection Defense Solution. IEEE 3rd International Conference on Big Data Security on Cloud.

IBM. (n.d.). Retrieved from IBM - United States: https://www.ibm.com/cloud/learn/nosql-databases

Nugroho, A. B., & Mandala, S. (2020, Desember 02). Study the Best PenTest Algorithm for Blind SQL Injection Attacks. INTL. JOURNAL ON ICT, 7-10.

Nugroho, A., & Winarko, E. (2013). Studi Perbandingan Perbedaan Konseptual Antara Sistem Basis Data Relasional Dengan Sistem Penyimpanan Data Bertipe Non-Relasional (No-Sql) : Eksplorasi Pada Server Data Cassandra.

Ombagi, J. (2017). Time-Based Blind SQL Injection via HTTP Headers: Fuzzing and Exploitation.

Priyadharshini, S., & Rajmohan, R. (2017). Analysis on Database Security Model Against NOSQL Injection.

Purnomosidi, B. (2013). Buku Cloud Node.js. Retrieved from https://github.com/bpdp/buku-cloud-nodejs

Simanjuntak, H. T., Simanjunta, L., Situmorang, G., & Saragih, A. (2015). Query Response Time Comparison Nosqldb Mongodb With Sqldb Oracle.

Singh, S. (2019). Security Analysis of MongoDB.

Trudeau, M., & Kolodny, J. (2017). An Analysis and Overview of MongoDB Security.

What is Python? Executive Summary. (n.d.). Retrieved from Python: https://www.python.org/doc/essays/blurb/

Shachi, M., Siddiqui Shourav, N., Syeed, A., Ahmed, S., Brishty, A. A., & Sakib, N. (2021). A Survey on Detection and Prevention of SQL and NoSQL Injection Attack on Server-side Applications. In International Journal of Computer Applications (Vol. 183, Issue 10