International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

NoSQL Injection Attacks and their Mitigation

Ankita Urade[1], Rutuja Hirve[2], Rachana Badekar[3], Ashwini Gaikwad[4]
[1,2,3,4]*AISSMS Institute of Information Technology*.

***Abstract*— NoSQL data storage systems have a wide acceptance due to their scalability and ease of use. Unfortunately, they lack the security measures and awareness that are required for data protection. The attackers get new opportunities for injecting their malicious code into the statements passed to the database because the new data models and query formats of NoSQL data stores make old attacks such as SQL injections tangential. A large amount of data is accumulated by organizations who wish to protect this data from all types of abuse. Thus, security is a prime concern with respect to multinationals hosting their websites. This paper addresses this issue while proposing ways to attenuate the problems.**

***Keywords*—NoSQL, SQL Injection,**

I. INTRODUCTION

Along with information security, database security has also become a crucial aspect when it comes to the term security. Lack of suitable security systems make it convenient for the attackers to get control over critical data by accessing the database. Since the systems are vulnerable they easily become the victims of these attack. One such attack is and SQL injection attack. It is an attack which inserts malicious code into the statement that is passed to the database by the application. This gives the attacker freedom to perform any unwanted operations like accessing unauthorized data, deleting, altering or inserting data. Although SQL injection exploitation has declined steadily over the years owing to secure frameworks and improved awareness, it remains a high-impact means to exploit system vulnerabilities. Web applications are prone to many attacks. Every month the web application receives more than four web attack campaigns and it is analyzed that SQL injections are the most popular attacks. It has also been observed that SQL injection vulnerabilities have an impact on 32 percent of all the web applications.

One of the trending and well known term in modern data stores is NoSQL (not only SQL) which basically refers to non-relational databases. Various storage mechanisms such as document store, key-value store, and graph are used by NoSQL databases. The requirements of modern large-scale applications have been encapsulated in these databases, some of them are Facebook, Amazon and Twitter.

They need to distribute data across ample of servers. This is one of the key benefits of modern relational databases. Traditional relational database lacks this factor as they do not meet these requirements. This task is time consuming in them because a single database node performs all the tasks of the same transaction.

Thus, this emerging NoSQL key-value stores is highly beneficial for the modern large-scale applications as it fulfils all its requirements. These data stores consist of different NoSQL databases like MongoDB and Cassandra along with different in-memory stores and caches like Redis and Memcached. There has been tremendous increase in the popularity of NoSQL databases due to it's key factors. Among the 10 most popular databases MongoDB is fourth ranking database. In this article, we provide analysis of various NoSQL threats and different mitigation mechanisms.

II. NOSQL SUSCEPTIBILITIES

Their primary advantage is that, unlike relational databases, they handle unstructured data such as documents, e-mail, multimedia and social media efficiently. The common features of NoSQL databases can be summarized as: high scalability and reliability, very simple data model, very simple (primitive) query language, lack of mechanism for handling and managing data consistency and integrity constraints maintenance (e.g., foreign keys), and almost no support for security at the database level. Like other emerging technologies NoSQL databases are not fully secured in all aspects. NoSQL databases afflicted by deficiency of encryption, appropriate authentication, role management. Supporters of NoSQL databases are web 2.0 companies which are Amazon and Google Only.

III. LITERATURE SURVEY

1 L.Okman et al. "Security Issues in NoSQL Databases," Proc.IEEE 10th Int'l Conf. Trust, Security and Privacy in Computing and Communications (TrustCom), 2011, pp. 541–547.

This paper describes two of the most popular NoSQL databases (Cassandra and MongoDB) and outlines their main security features and problems.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Some of the security features of Cassandra and MongoDB are as follows:

1. MongoDB Data Files Mongo data-files are unencrypted, and Mongo doesn't provide a method to automatically encrypt these files. This means that any attacker with access to the file system can directly extract the information from the files. In order to mitigate this, the application must explicitly encrypt any sensitive information before writing it to the database. In addition, operating -system level mechanisms (file system permissions, file system level encryption, etc.) should be used to prevent access to the files by unauthorized users.
2. Cassandra Data Files: The data in Cassandra is kept un - encrypted and Cassandra does not provide a mechanism to automatically encrypt the data in storage. This means that any attacker with access to the file-system can directly extract the information from the files. In order to mitigate this, the application must explicitly encrypt any confidential information before writing it to the database. Also, operating-system level mechanisms (filesystem permissions, file system level encryption, etc.) should be used to prevent access to the files by unauthorized users.

2.A.Lane,"NoSQL and NoSecurity," blog, 9 Aug.2011;www.securosis.com/blog/nosql-and-no-security.

In this proposed system one of the references was a blog named 'No SQL and No Security' in which Brian Sullivan gave a presentation on "Server-side JavaScript Injection: Attacking NoSQL and Node.js". Nowadays we are aware of the poor security of most NoSQL database installations especially their lack of support for authorization and authentication but we are not aware of their susceptibility to injection. Brian demonstrated NoSQL injection scripts that can both discover database contents and run arbitrary commands. Node and NoSQL are basically JavaScript based platforms with both server and client functionality which makes them susceptible to client and server side attacks. He further demonstrated the ability to inject changes to the node server, write an executable to the file system using Node.js calls and then running it.

3.M. Factor et al. "Secure Logical Isolation for Multi-tenancy in Cloud Storage," Proc. IEEE 29th Symp. Mass Storage Systems and Technologies (MSST), 2013, pp. 1–5.

A compromised web front end cannot access any customer's data directly since it is not privileged to use the data stores.

However, given that TLS termination occurs within this component, an attacker could mount a man in-the-middle attack against any tenant, accessing that tenant's data. Yet, the attack is limited in time. Additionally, the system may authenticate the requests (e.g., using signatures) and test for authenticity at the request processor; in this way, the attacker can be prevented from tampering with the requests. Request processor: In case a request processor is compromised, the attacker's process can only access the corresponding tenant-data, and is restricted to performing tenant-specific queries through the security gateway and proxy. This does not prevent the attacker from accessing data from another user of the same tenant, but cramped the attack within tenant.

IV. NoSQL Injection Attacks

- *Tautologies:*

These attacks allow bypassing authentication or access mechanisms by injecting code in conditional statements, generating expressions that are always true (tautologies). For example, in this article, we show how attackers can exploit the syntax of the $ne (not equal) operator, which lets them illegally log in to the system without appropriate credentials.

- *JavaScript injections:*

This type of new vulnerabilities introduced by NoSQL databases allows execution of JavaScript in the database context. JavaScript allows complicated transactions and queries on the database engine. Passing unsanitized user input to these queries might allow for injection of arbitrary JavaScript code, which results in illegitimate data extraction or alteration

V. Methods Used

- *JavaScript Object Notation Queries(JSON) and Data Formats:*

Queries and Data are represented in JSON format, which is better than SQL in terms of security because it is more "well defined", very simple to encode/decode and also has good native implementations in every programming language. Breaking the query structure as has been done in SQL injection is harder to do with a JSON structured query. A typical insert statement in MongoDB looks like:

```
db.books.insert({
title: 'As you like it'.
```

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Author: 'William Shakespeare'

})

This inserts a new document into the books collection with a title and author field. A typical query looks likeQueries can also include regular expression.

db.books.find ({title: As you like it})

- *PHP Tautology (array) injection:*

web application is implemented with a PHP backend, which encodes the requests to the JSON format used to query the data store. Let's use an example of MongoDB to show an array injection vulnerability – an attack similar to SQL injection in its technique and results.

array('title' => 'As you like it', 'author' => 'William Shakespeare');

would be encoded by PHP to the following json:

{"title": " As you like it", "author": "William Shakespeare" }

But PHP has a built in mechanism for associative arrays which allows an attacker to send the following malicious payload:

username[$ne]=1&password[$ne]=1

PHP translates this input into:

array("username" => array("$ne" => 1), "password" => array("$ne" => 1))

Which is encoded into the mongo query:

db.logins.find({ username: { $ne: 1 }, password: { $ne: 1 } })

SQL terminology this is equivalent to:

SELECT * FROM logins WHERE username <> 1 AND password <> 1

- *NoSQL Union Query Injection:*

One of the common reasons for a SQL injection vulnerability is building the query from string literals which include user input without using proper encoding. The JSON query structure makes it harder to achieve in modern data stores like MongoDB. Nevertheless it is still possible. Let us examine a login form which sends its username and password parameters via an HTTP POST to the backend which constructs the query by concatenating strings. For example the developer would do something like:

string query = "{ username: '" + post_username + "', password: '" + post_password + "' }"

This query will succeed as long as the username is correct. In SQL terminology this query is similar to:

SELECT * FROM logins WHERE username = 'shakespeare' AND (TRUE OR ('a'='a' AND password = '')) #successful MongoDB injection

That is, the password becomes a redundant part of the query since an empty query {} is always true and the comment in the end does not affect the query. How did this happen? Let's examine the constructed query again and color the user input in bold red and the rest in black:

{ username: 'shakespeare', $or: [{}, { 'a': 'a', password: '' }], $comment: 'successful MongoDB injection' }

VI. System Architecture

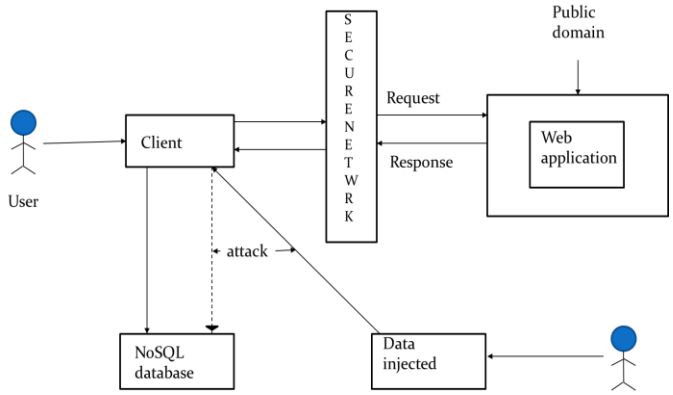

Fig. System architecture.

VII. Mitigation

Web sites that interface with databases are particularly vulnerable to SQL injection because they often rely on dynamic SQL, so Databases are the integral part of web application.One must has to secure it in order to protect user's personal, credential data. You must secure your database connections and limit access privileges where you can.One should also be vigilant about escaping and validating all user input.Mitigating security risks in NoSQL deployments is major part of different attacks we present in this paper. Unfortunately,code analysis of the application layer alone is not adequate to ensure that all risks are mitigated. These are commonly developed by open source communities and, in most cases, don't undergo comprehensive security testing.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Speed of modern code development with DevOps methodologies is one of the challenge, which aim to shorten the time between development and production. Finally, most application security testing tools can't keep up with the fast pace with which new programming languages are adopted. There are numerous ways a malicious user might penetrate your system using SQL injection and various defenses, but the simplest approach is to avoid dynamic SQL. Instead, use stored procedures everywhere.

There are various techniques proposed by us in this paper in order to mitigate NoSQL injection attack such as, prepared statement, validation techniques, Textbox condition check and intrusion detection. Some testing approaches will also be helpful such as Dynamic application security testing (DAST).

The mitigation proposed by us will include two phases:

1. Development and Testing:

In this, we consider the threats involved in the software development lifecycle of our online shopping website. The various attacked modules will be mitigated by using the following techniques

i. Using best practices of code like strong JSON structure, proper validation, prepared statement etc.
ii. Looking closely through the design aspects such as what need to be protected and how will this occur.
iii. Spreading awareness among the developers so that they are less likely to portray weaknesses in their code
iv. Running dynamic and static security testing so as to detect the vulnerabilities in code for injection attacks. We will run various test cases to check the performance of the tester.

2. Monitoring and Attack Detection

A look at the importance of adopting intrusion detection systems will be shown.

VIII. CONCLUSION

We will review different attacks which are vulnerable to the database. Main methodology behind the attacks are discussed. In order to protect the database from these attacks some mitigation techniques are proposed which were discussed earlier which we will carry out. Along with that, some test cases will be written for Dynamic application security testing (DAST) to grade the security level of the code. This paper will act as a guide to all the developers developing a web application to attain the level of security they wish for.

REFERENCES

[1] A. Lane, "No SQL and No Security," blog, 9 Aug. 2011; www.securosis.com/blog/nosql-and-no-security. 4.

[2] L. Okman et al. "Security Issues in NoSQL Databases," Proc. IEEE 10th Int'l Conf. Trust, Security and Privacy in Computing and Communications (TrustCom), 2011, pp. 541–547. 5.

[3] E. Sahafizadeh and M.A. Nematbakhsh. "A Survey on Security Issues in Big Data and NoSQL," Int'l J. Advances in Computer Science, vol. 4, no. 4, 2015, pp. 2322–5157.

[4] M. Factor et al. "Secure Logical Isolation for Multi- tenancy in Cloud Storage," Proc. IEEE 29th Symp. Mass Storage Systems and Technologies (MSST), 2013, pp. 1–5.

[5] "Security," MongoDB 3.2 Manual, 2016; http://docs .mongodb.org/manual/core/security-introduction.

[6] I. Novikov, "The New Page of Injections Book: Memcached Injections," Proc. Black Hat USA, 2014; www.blackhat.com /docs/us-14/materials/us-14-Novikov-The-New-Page -Of-Injections-Book-Memcached-Injections-WP.pdf.

[7] J. Williams, "7 Advantages of Interactive Application Security Testing (IAST) over Static (SAST) and Dynamic (DAST) Testing," blog, 30 June 2015; https://www .contrastsecurity.com/security-influencers/9-reasons -why-interactive-tools-are-better-than-static-or-dynamic -tools-regarding-application-security.

[8] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," in Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, Oct. 2007.

[9] S.M. Kerner, "Glass Box: The Next Phase of Web Application Security Testing?," blog, 3 Feb. 2012; www .esecurityplanet.com/network-security/glass-box-the -next-phase-of-web-application-security-testing.html.