Analysis and Diminution of NoSQL Injection attacks

Pritesh Patil[1], Ashwini Gaikwad[2], Rachana Badekar[3], Ankita Urade[4], Rutuja Hirve[5]

[1]Prof Pritesh Patil AISSMS Institute of Information Technology
[2]Ashwini Gaikwad AISSMS Institute of Information Technology
[3]Rachana Badekar AISSMS Institute of Information Technology
[4]Ankita Urade AISSMS Institute of Information Technology
[5]Rutuja Hirve AISSMS Institute of Information Technology

Abstract

Scalability and ease of use has been one of the major reasons behind the popularity of NoSQL storage systems. Flexibility and scalability of NoSQL databases are major cause for adoption and popularity. Unfortunately, they lack the security measures and awareness that are required for data protection. The global exposure of these applications makes them prone to the attacks because of presence of vulnerabilities. These security vulnerabilities continue to infect the web applications through injection attacks thus enabling the attackers to attack on the database by injecting malicious code into the statements passed to the database because the new data models and query formats of NoSQL data which in turn makes the attack divergent.

Keywords- NoSQL , SQL Injection, Mitigation, Security

1. INTRODUCTION

Database security has been one of the most critical aspects of application security. Lack of suitable security systems make it convenient for the attackers to get control over critical data by accessing the database. Recently, the NoSQL databases have become more and more popular because NoSQL databases provide looser consistency restrictions than traditional SQL databases do. NoSQL databases often offer performance and scaling benefits by requiring fewer relational constraints and consistency checks. This enables att ackers to do almost anything with the data, including accessing unauthorized data and altering, deleting, and inserting data. Example of such attack is SQL injection is a code injection technique that is used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution. The malicious user can use SQL commands insert to the Web form submission or enter the domain name to achieve the purpose of tricking the server to execute malicious SQL commands. This gives the attacker freedom to perform any unwanted operations like accessing unauthorized data, deleting, altering or inserting data.

NoSQL (not only SQL) is a trending term in modern data stores; it refers to nonrelational databases that rely on diff erent storage mechanisms such as document store, key-value store, and graph. NoSQL is a wide class of database management systems that are not traditional relational database management systems. They do not use SQL language as the primary query language, nor do they typically require fixed table schemas. NoSQL database system allows a user to change data attributes at any time, and data can be added anywhere. Various storage mechanisms such as document store, key-value store, and graph are used by NoSQL databases. The wide adoption of these databases has been facilitated by the new requirements of modern large-scale applications, such as Facebook, Amazon, and Twitter, which need to distribute data across a huge number of servers. One of the biggest advantages is the ability to change attributes because of the weakening of structural, so modification process is very convenient.

Indeed, the popularity of NoSQL databases has grown consistently over the past several years due to it's key factors. Among the 10 most popular databases MongoDB is fourth ranking database. In this article, we provide analysis of NoSQL threats and their mitigation mechanisms.

2. NoSQL FRAGILITY

NoSQL is vulnerable the same way SQL databases are vulnerable. Some attacks which are relevant in SQL databases become obsolete in NoSQL databases. Like almost every new technology, NoSQL databases lacked security when they first emerged. NoSQL databases are not fully secured in all aspects. NoSQL databases afflicted by deficiency of encryption, appropriate authentication, role management. The poor programming/coding practice leads to vulnerabilities. For example, vulnerability like loopholes attract the attacker to customize attacks. The attacker can plan a particular attack according to the specific vulnerability present in the application. NoSQL databases are supported by web 2.0 companies which are Amazon and Google Only.

3. NoSQL INJECTION BLITZ

NoSQL injections enable an attacker to inject code into the query that would be executed by the database. These flaws are introduced when software developers create dynamic database queries that include user supplied input. These attacks are broadly categorized into five types:-

3.1 Union Query

An attacker injects an UNION SELECT to trick the application into returning data from a table different from the one that was intended. Here is a common form using a single quote for this attack:

normal SQL statement + "semi-colon" + UNION SELECT <rest of injected query>.

3.2 Tautologies

An attacker injects a query that always evaluates to true for entries in the database to bypass authentication, identify injectable parameters, or extract data. For example:

Username known
Input (username): jdoe' or '1'='1--
Sql: SELECT * FROM students WHERE username = 'jdoe' or '1'='1' -- AND password =
Result: All students are retrieved.
Both username and password not known
Input (username): ' or " = '
Input (password): ' or " = '
Sql: select * from students where username = " or " = " and password = " or " = "
Result: All students are retrieved.

3.3 PiggyBacked Queries

An attacker injects additional queries into the original query to extract data, add or modify data, perform denial of service, or execute remote commands. In this scenario, the attacker does not intend to modify the original intended query but to include new queries that piggy-backon the original query. As a result, the DBMS receives multiple SQL queries. The first is the normal query which is executed normally, while the subsequent ones are executed to satisfy the attack. Here is a common form using a query delimiter (;) for this attack:

normal SQL statement + ";" + INSERT (or UPDATE, DELETE, DROP) <rest of injected query>

3.4 Stored Procedures:

When a normal SQL statement (i.e., SELECT) is created as a stored procedure, an attacker can inject another stored procedure as a replacement for a normal stored procedure to performing privilege escalation, create denial of service, or execute remote commands. Here is a common form using a query delimiter (;) and the "SHUTDOWN" store procedure for this attack:

normal SQL statement + "; SHUTDOWN; " <rest of injected query>

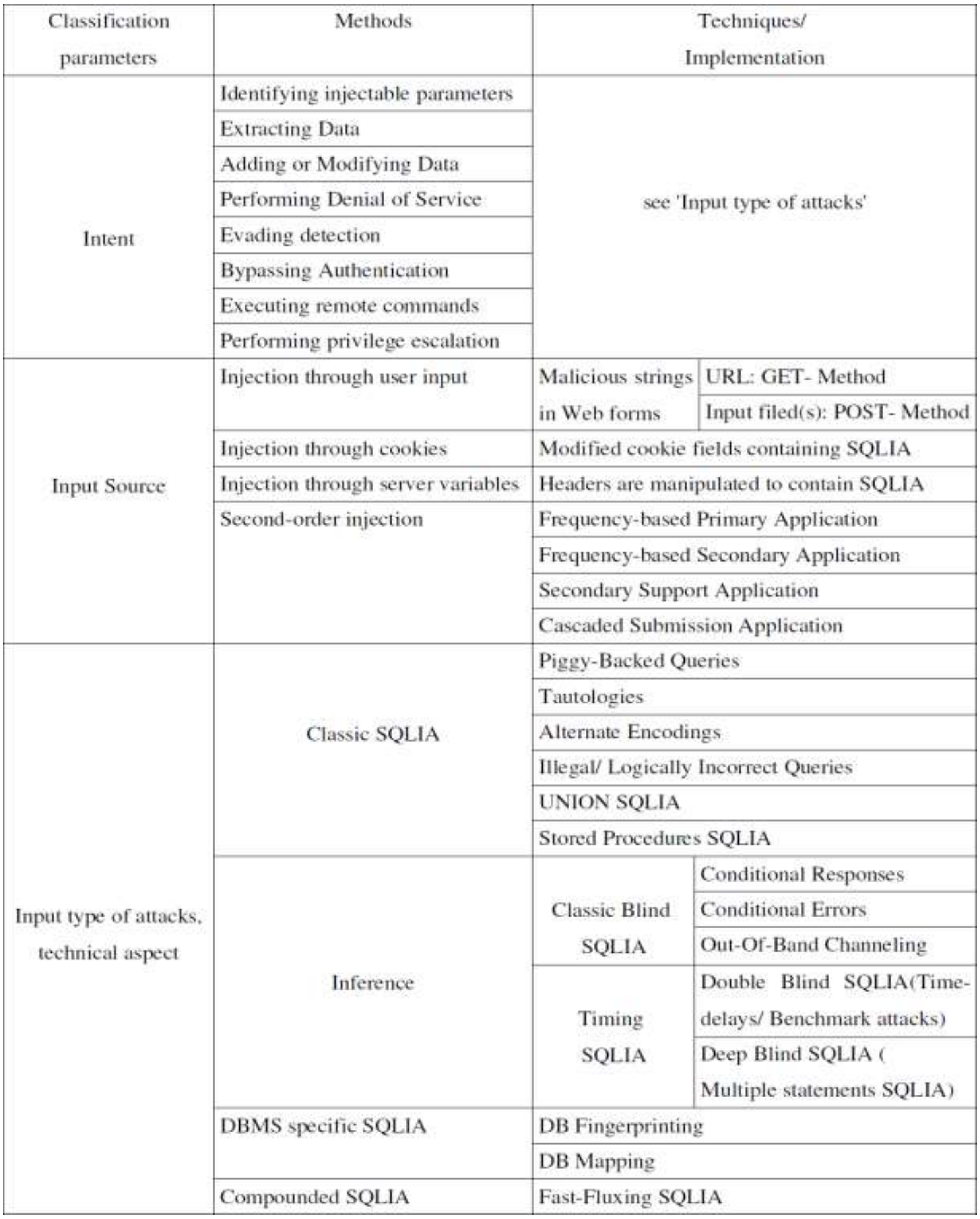
3.5 Javascript injection

Javascript injection allows you to change websites behavior without refreshing or leaving it. It provides on spot interaction with the source code of website from browser window. Javascript script might come really handy when you are hacking basic websites. Javascript injection allows you to alter the form values before sending it to server.

In Javascript injection, javascript codes are injected from address bar of the browser window. In this tutorial we’ll go through the basics of javascript injection, if you are javascript expert then it might be below your knowledge. However freshers might find it interesting and informative. To command any javascript code to your browser you must inform it that its javascript. It can be done by adding “Javascript:”(without quotes) just before your code.

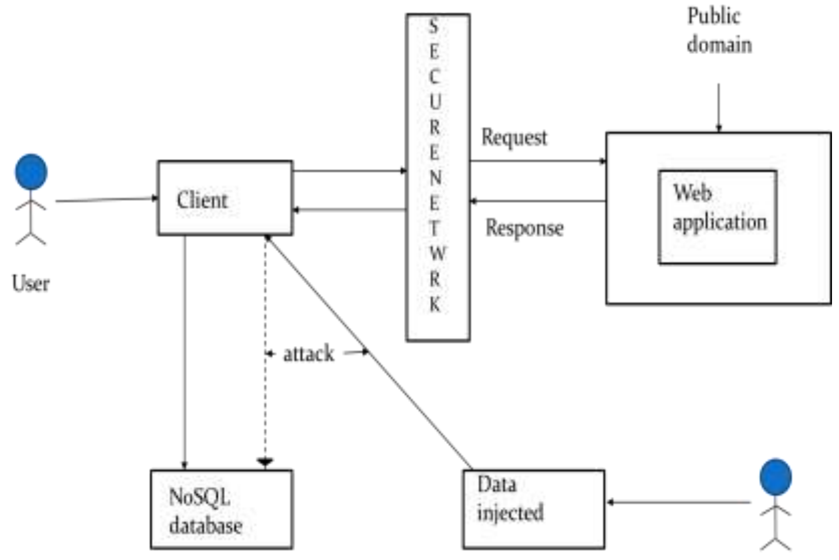
Below is the sample code to input in your browser.

Javascript: alert("Welcome to yahoo.com");



Classification parameters	Methods	Techniques/ Implementation	
Intent	Identifying injectable parameters	see 'Input type of attacks'	
	Extracting Data		
	Adding or Modifying Data		
	Performing Denial of Service		
	Evading detection		
	Bypassing Authentication		
	Executing remote commands		
	Performing privilege escalation		
Input Source	Injection through user input	Malicious strings in Web forms	URL: GET- Method
			Input filed(s): POST- Method
	Injection through cookies	Modified cookie fields containing SQLIA	
	Injection through server variables	Headers are manipulated to contain SQLIA	
	Second-order injection	Frequency-based Primary Application	
		Frequency-based Secondary Application	
		Secondary Support Application	
		Cascaded Submission Application	
Input type of attacks, technical aspect	Classic SQLIA	Piggy-Backed Queries	
		Tautologies	
		Alternate Encodings	
		Illegal/ Logically Incorrect Queries	
		UNION SQLIA	
		Stored Procedures SQLIA	
	Inference	Classic Blind SQLIA	Conditional Responses
			Conditional Errors
			Out-Of-Band Channeling
		Timing SQLIA	Double Blind SQLIA(Time-delays/ Benchmark attacks)
			Deep Blind SQLIA (Multiple statements SQLIA)
	DBMS specific SQLIA	DB Fingerprinting	
		DB Mapping	
	Compounded SQLIA	Fast-Fluxing SQLIA	

4. SYSTEM ARCHITECTURE



5. MITIGATION

An SQL injection is a well known attack and easily prevented by simple measures. 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. 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.

Let us scrutinize a few recommendations for each of the threats:

- Prepared statements with parameterized queries: Prepared statements should be used instead of building dynamic queries using string concatenation.
- Strong JSON structure queries
- Input Validation: Validate inputs to detect malicious values. For NoSQL databases, also validate input types against expected types
- Principle of least privilege: To minimize the potential damage of a successful injection attack, do not assign DBA or admin type access rights to your application accounts. Similarly minimize the privileges of the operating system account that the database process runs under.

The two phases of mitigation presented by us are:

5.1 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

- Looking closely through the design aspects such as what need to be protected and how will this occur.
- Using best practices of code like strong JSON structure, proper validation, prepared statement etc.
- Spreading awareness among the developers so that they are less likely to portray weaknesses in their code
- 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.

5.2 Monitoring And Attack Detection

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

6. CONCLUSION

We examined various threats that are vulnerable to the database. Different ways to tackle those attacks are discussed. In order to protect the database from these attacks some mitigation techniques are proposed which are Dynamic Application Security Testing (DAST). We will be using prepared statement and validation as a solution for mitigation of NoSQL injection.

7. REFERENCES

[1] Aviv Ron, Alexandra Shulman-Peleg, and Anton Puzanov "Analysis and Mitigation of NoSQL Injections"

[2] Halfond, W. G., Viegas, J., and Orso, A. A Classification of SQL-Injection Attacks and Countermeasures. In SSSE (2006).

[3] D. Litchfield. Web Application Disassembly with ODBC Error Messages. Technical document, @Stake, Inc., 2002.

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

[5] Damon Poeter. 'Close-Knit' Russian Hacker Gang Hoards 1.2 Billion ID Creds, *PC Magazine*, August 5, 2014

[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] Least Privilege mitigation to SQL injection https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Least_Privilege

[10] MongoDB web site www.mongodb.org

[11] http://cassandra.apache.org

[12] Chandershekhar Sharma, Dr. S.C.Jain "Analysis and Classification of SQL Injection Vulnerabilities and Attacks on Web Applications"

[13] 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.

[14] 9 Advantages of Interactive Application Security Testing (IAST) over Static (SAST) and Dynamic (DAST) Testing http://www1.contrastsecurity.com/blog/9-reasons-why-interactive-toolsare-better-than-static-or-dynamic-tools-regarding-application-security