NoSQL Racket: A Testing Tool for Detecting NoSQL Injection Attacks in Web Applications

Ahmed M. Eassa
Information Systems Department
Faculty of Computer & Information Sciences
Mansoura University
Mansoura, EGYPT

Hazem M. El-Bakry
Information Systems Department
Faculty of Computer & Information Sciences
Mansoura University
Mansoura, EGYPT

Omar H. Al-Tarawneh
Information Technology Department
Al-Zahra College for Women,
Muscat, Oman

Ahmed S. Salama
Computer and Information Systems Department
Sadat Academy for Management Sciences
Cairo, EGYPT

***Abstract*—A NoSQL injection attack targets interactive Web applications that employ NoSQL database services. These applications accept user inputs and use them to form query statements at runtime. During NoSQL injection attack, an attacker might provide malicious query segments as user input which could result in a different database request. In this paper, a testing tool is presented to detect NoSQL injection attacks in web application which is called "NoSQL Racket". The basic idea of this tool depends on checking the intended structure of the NoSQL query by comparing NoSQL statement structure in code query statement (static code analysis) and runtime query statement (dynamic analysis). But we faced a big challenge, there is no a common query language to drive NoSQL databases like the same way in relational database using SQL as a standardized query language. The proposed tool is tested on four different vulnerable web applications and its effectiveness is compared against three different well known testers, none of them is able to detect any NoSQL Injection attacks. However, the implemented testing tool has the ability to detect the NoSQL injection attacks.**

Keywords*—NoSQL; injection attack; web application; web security; testing tool

I. Introduction

The recent advance in cloud computing and web applications has created the need to store large amount of data in multi-different databases that provide high availability and scalability. In last years, more and more of companies have adopted different types of non-relational databases, commonly referred to as NoSQL "Not only SQL" databases, and as the applications they serve emerge, they gain wide market interest. The NoSQL databases are not relational by definition and therefore they do not support full SQL functionality, instead of relational databases, they trade consistency and security for performance and scalability. As increasingly sensitive data is being stored in NoSQL databases, security issues become growing concerns [1]-[3].

In this paper we propose a web based tool named "NoSQL Racket". This tool has ability to detect and prevent NoSQL injection attacks in web applications.

II. Related Work

Many researchers have contributed in the area of NoSQL security. Bryan Sullivan [4] explained security issues related to NoSQL databases and differences with relational databases, and what extra set of issues need to be considered when designing and developing systems using these types of data stores. He discussed injection techniques against MongoDB and then moved on to compelling examples of server-side JavaScript injection using Node.js as an example. He discussed risky constructs to look for, during code review and ways to avoid some typical pitfalls.

Sooel S. et al. [5], describes the design and implementation of Diglossia, a tool detects code injection attacks on server-side Web applications generating SQL and NoSQL queries. To detect injected code in a generated query, Diglossia parses the query in tandem with its shadow and checks that the two parse trees are syntactically isomorphic, and all code in the shadow query is in shadow characters and, therefore, originated from the application itself, as opposed to user input.

Okman, L. et al. [1], discusses two of the most common NoSQL databases (Cassandra and MongoDb) and outlines their main security weaknesses and problems.

IBM eBook report [6], it provides a basic introduction to the topic of NoSQL and its rapid growth and adoption. In addition to, it's focus on two primary areas around data security and protection, and how "IBM InfoSphere Guardium" solutions can help with both of them.

Adrian Lane [7], it examining security for "big data" environments, reviewing built-in protections and weaknesses of these systems which are depending on the Hadoop framework and the other common NoSQL databases (Cassandra, MongoDB, Couch, Riak, etc.).

Amreen and Dadapeer [8], Present a reversible watermarking algorithm to provide the security for NoSQL by using a unique watermark to mark the data and by using

reversible watermarking technique which allows recovery of original data along with the embedded watermark information.

Aviv Ron and Alexandra Shulman-Peleg [9], Present a few techniques for attacking NoSQL databases such as injections and CSRF. Also, they present methodologies to mitigate these attacks.

III. Injection Attacks Types

"The OWASP Top 10" [10] and "The 2011 CWE/SANS Top 25" [11] lists injection attack as the most common security risk to web applications. Injection is an entire class of attacks that rely on injecting data into a web application in order to facilitate the execution or interpretation of malicious data in an unexpected manner. Examples of attacks within this class include Cross-Site Scripting (XSS), SQL/NoSQL queries, Header Injection, Log Injection and Full Path Disclosure.

OWASP 2010 defines injection as follows:

"Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code, often found in SQL queries, LDAP queries, XPath queries, OS commands, program arguments, etc."[12].

But this definition was modified several times by OWASP from 2013 to 2017 and ended up defining injection which includes NoSQL and became as follows:

"Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc." [10], [13].

Injection attacks have ruled in the top of web application vulnerability reports for much of the past decade. The OWASP Top 10 Project (2013, 2017), which tests and evaluates the most critical threat categories against web applications, places 'Unvalidated Input' in the top spot, followed by the related XSS Flaws and Injection Flaws in 3th and 8th place respectively. The CWE/SANS Top 25 Most Dangerous Software Errors list also places high risk on the same issues [11].

Injection attacks can be classified according to OWASP into the following types:

- Blind SQL Injection.
- Blind XPath Injection.
- Buffer Overflow.
- Format String Attack.
- LDAP Injection.
- OS Commanding.
- SQL Injection.
- SSI Injection.
- XPath Injection.
- NoSQL Injection.

But for the purpose of this paper, we will be focusing on NoSQL injection attack and will be discussed in the following section.

IV. NoSQL Injection Attack

NoSQL injection refers to an injection attack through the placement of malicious code (like other web attack ways) in NoSQL statements through web page input controls. The attacker takes the advantage of poorly filtered or not correctly escaped characters within part of NoSQL statements and injects arbitrary data into a string that's eventually run by the NoSQL database engine (e.g. a login form) as shown in Fig. 1.

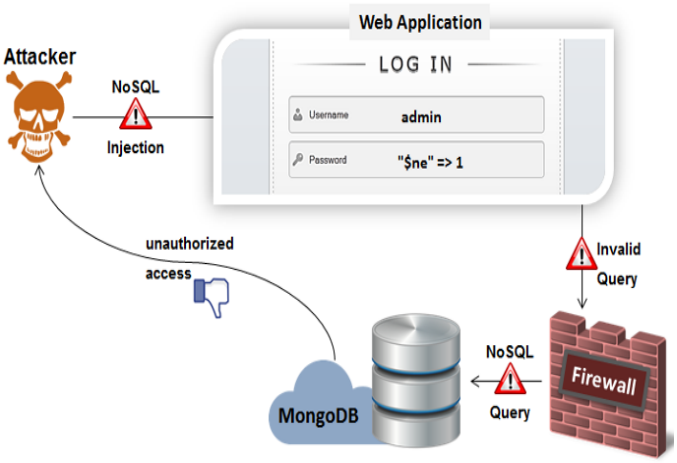

Fig. 1. NoSQL injection attack in web applications.

Through vulnerable Web applications, attacker can get unauthorized access to a NoSQL database and can modify or delete data. Currently almost all NoSQL databases such as MongoDB, Hadoop/HBase, Cassandra, CouchDB, and Riak are potentially vulnerable to NoSQL injection attacks. NoSQL injection attack can occur in web applications through some methods, such as Injection through web page input controls and cookie files.

Web based forms allow somewhat access to back-end NoSQL database to allow adding or modifying the stored data. Any web form, even a simple login form, signup form or search box (where user can input or modify data), might provide access to back-end NoSQL database. This means that there is a high probability for injecting malicious code and attacker can bypasses firewalls and endpoint defenses.

The common reason that a web application is vulnerable to NoSQL injection is incorrect filtering and poor validation for user input. Web forms are quite common to collect data from user. So, practically it is not suitable to lock all the entry points to bar NoSQL injection attackers. To prevent attacks, web developers must apply proper filtration/validation on all forms. For more clarifying, we will show in the following an example for NoSQL injection attack.

Let's suppose that some PHP web application requests through the screen a user name and a password to access a private area. The application will pick these values and it will

collect a query to send to the NoSQL database (e.g. MongoDB).

The MongoDB collection "regusers" contains two documents for authorized users as shown in Fig. 2.

The PHP webpage might look like Fig. 3.

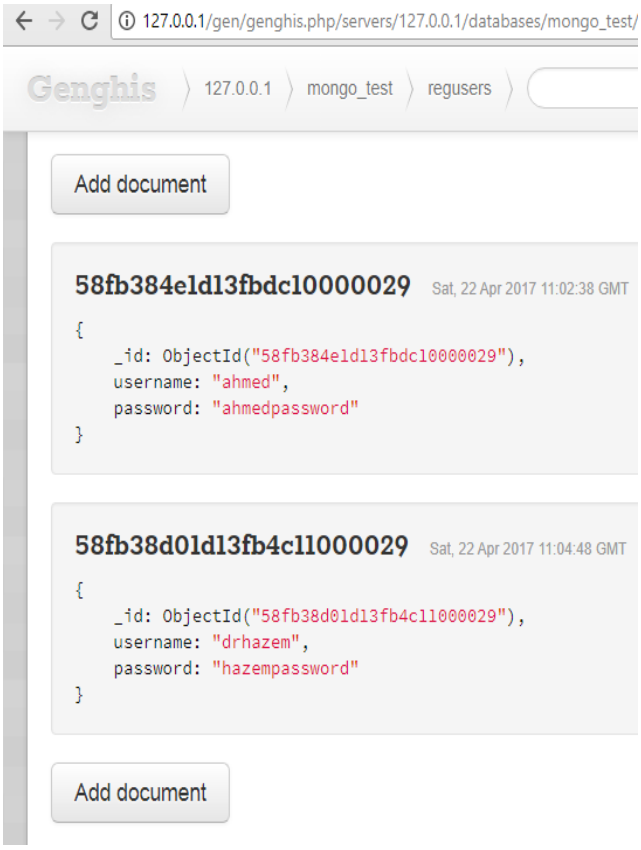

Fig. 2. MongoDB collection "regusers".

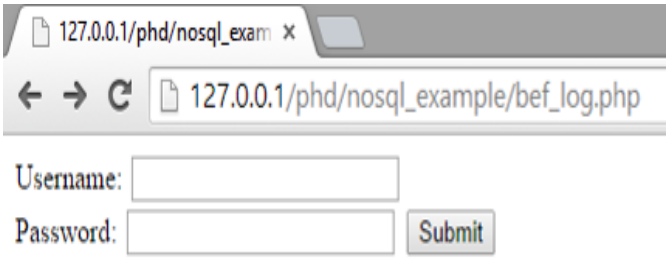

Fig. 3. PHP login webpage.

Supposing that is a PHP script selects a document from MongoDB. The following NoSQL query string verifies a username and password combination is valid or not:

$collection->find(array("username" => $_GET['username'],

"password" => $_GET['password']));

In this case, attacker user can write some texts that will be sent to the NoSQL database (MongoDB) without any verification. Given the case of a malicious user, he could write in the password field the string "$ne" =1 as shown in Fig. 4.

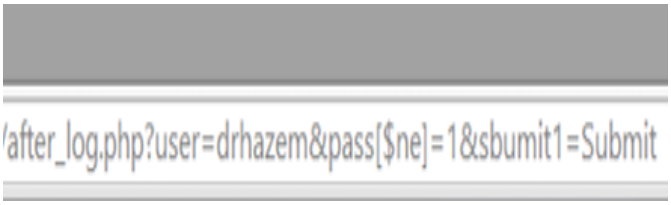

Fig. 4. NoSQL injection in password field.

In this case, the resulting query will be as follows:

$collection->find(array("username" => "drhazem",

"password" => array("$ne" => 1)));

"$ne" selects the documents where the value of the field is not equal to "1". So, this query will produce the same result as if the admin user had introduced their password correctly. According to this example, the web application will allow the access to administration area to a user who doesn't know the proper password.

V. Proposed Testing Tool "NoSQL Racket"

There are now over 225 NoSQL databases available for use with web applications. Each one offers different features and limitations. So, we faced a big challenge because there is not a common language between web applications and NoSQL databases [10].

For this reason, our proposed tool offers a general testing mechanism for detecting all NoSQL injection attacks without depending on specific syntax and data model. To overcome this challenge, we will create a simple database table named "Driverstbl". The table "Driverstbl" contains all query string forms and its types such as reserved keywords, logical operators and relational operators as shown in Table 1.

TABLE I. "Driverstbl"

NoSQL Database Type	String Type	Syntax
MongoDB	Reserved keywords (RK)	db
MongoDB	Reserved keywords (RK)	find
MongoDB	Reserved keywords (RK)	update
MongoDB	Operator(OP)	$and
MongoDB	Operator(OP)	$exists
MongoDB	Operator(OP)	$ne
and so on for other "String Types" in MongoDB.		
CouchDB	Reserved keywords (RK)	getDatabaseInfos
CouchDB	Reserved keywords (RK)	getDoc
CouchDB	Reserved keywords (RK)	storeDoc
CouchDB	Operator(OP)	&&
CouchDB	Operator(OP)	in
and so on for other "String Types" in CouchDB and so on for other NoSQL Database Types (Cassandra, Amazon DynamoDB,)		

Each code query statement and runtime query statement transformed into comparative patterns format depending on "Driverstbl" table as shown in Fig. 5.

The "NoSQL Racket" testing tool returns an array that contains the number of repetition for each string stored in "Driverstbl" table. Supposing the following PHP script in static code state is S1 and the same code statement in dynamic state is S2:

S1:$collection->find(array("username" => $_GET['username'],

"password" => $_GET['password']));

S2:$collection->find(array("username" => "drhazem",

"password" => array("$ne" => 1)));

According to "Driverstbl", The "NoSQL Racket" testing tool generates the following patterns for S1, S2:

S1 pattern: Array ([0] => Array ([0] => PK [1] => find [2] => 1) [1] => Array ([0] => PK [1] => array [2] => 1) [2] => Array ([0] => OP [1] => => [2] => 2)).

S2 pattern: Array ([0] => Array ([0] => PK [1] => find [2] => 1) [1] => Array ([0] => OP [1] => $ne [2] => 1) [2] => Array ([0] => PK [1] => array [2] => 2) [3] => Array ([0] => OP [1] => => [2] => 3)).

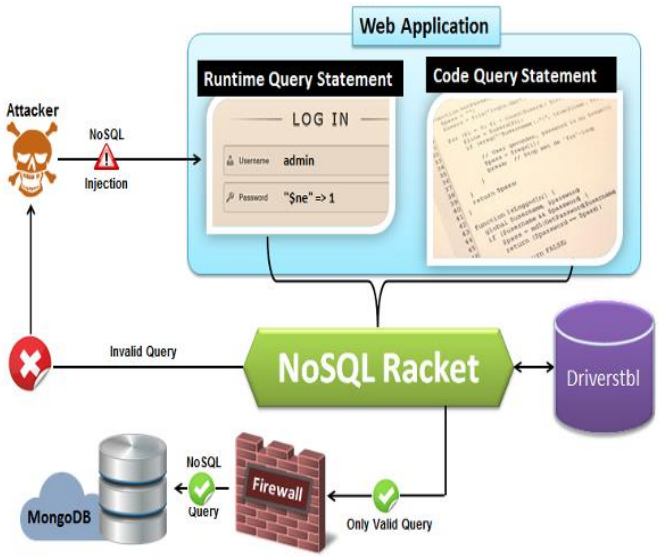

Fig. 5. NoSQL injection in password field.

After generating patterns for each query statement code (S1) and query statement in running state (S2), The "NoSQL Racket" will check the matching between generated patterns as shown in the following algorithm:

Step 1: Get Code Query Statement (S1) and Runtime Query Statement (S2).

Step 2: Let DBT= NoSQL database type.

Step 3: Connect to nosqldbs and select all "String Type" and "Syntax" from "Driverstbl" table where NoSQL Database Type = DBT.

Step 4: Group and count words in S1 and S2 according to selected data in Step3.

Step 5: Generate patterns for each statement S1, S2.

Step 6: Set Inj =0

Step 7: For each item in S1 and S2 patterns, repeat until end of items.

Step 7.1: If S1[i] not equal to S2[i],then set Inj =1

Step 7.2: Go to Step 7.

Step 8: If Inj =1, then display error page and stop running, else continue running & execute query.

According to the results of matching patterns and input values, there are two decisions:

If the patterns are matched, the web application will continue running.

If the patterns are not matched, the web application will be terminated and the proposed algorithm displays an error page.

VI. Experiments and Results

To investigate the effectiveness of the proposed tool "NoSQL Racket", we will examine the detection ability through a comparative study with the most powerful testers for example, Netsparker, Vega and Skipfish. On the other hand, we will use four versions of web pages in our comparative study which covers all NoSQL databases types which are Document based, Column oriented and Key-valued. Each version connected to different NoSQL database which are (MongoDB, Cassandra, CouchDB and Amazon DynamoDB).

Also, we will examine the performance for our proposed tool "NoSQL Racket" through performance testing tool called "LoadComplete".

A. Detection Ability Comparative Study

Four PHP web scripts are used for examination purpose and all of them are vulnerable for NoSQL injection attack. These scripts execute after submitting the user login button. When user is submitting with correct username and password against each NoSQL database, output will be "Authorized User", but on the other wise if any one of the field or both are incorrect then the output will be "You are not authorized".

1) MongoDB Detection Ability Results: In MongoDB, it is possible to inject NoSQL keywords into submitted data from the login webpage. This could for example look like this

http://127.0.0.1/phd/MongoDB/after_log.php?user=ahmed &pass[$ne]=1&sbumit1=Submit

"$ne" selects the documents where the value of the field is not equal (i.e. !=) to "1". So, this query will produce the same result as if the admin user had introduced their password correctly. According to this example, the web application will allow the access to administration area to a user who doesn't know the proper password.

The login web page scanned by giving URL to each following scanner tester tool:

Netsparker testing results are figured out and shown in Fig. 6.

Skipfish testing results are figured out are shown in Fig. 7.

Vega testing results are figured out and shown in Fig. 8.

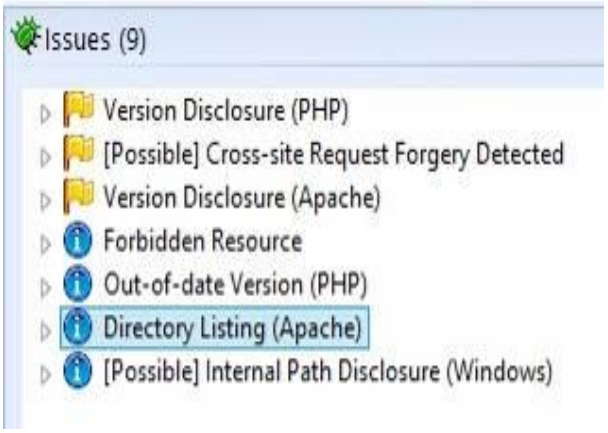

Fig. 6. Netsparker testing results for MongoDB.

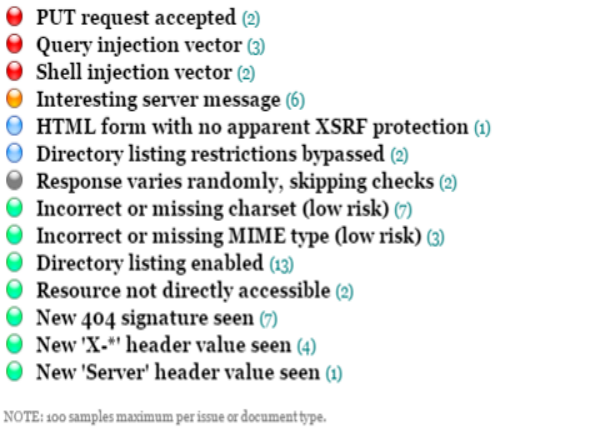

Fig. 7. Skipfish testing results for MongoDB.

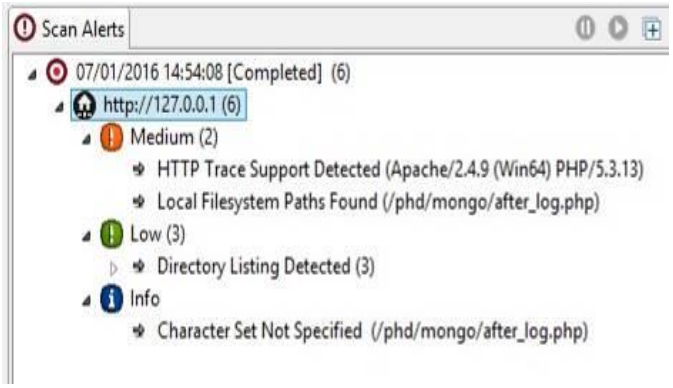

Fig. 8. Vega testing results for MongoDB.

"NoSQL Racket" testing results are figured out and shown in Fig. 9.

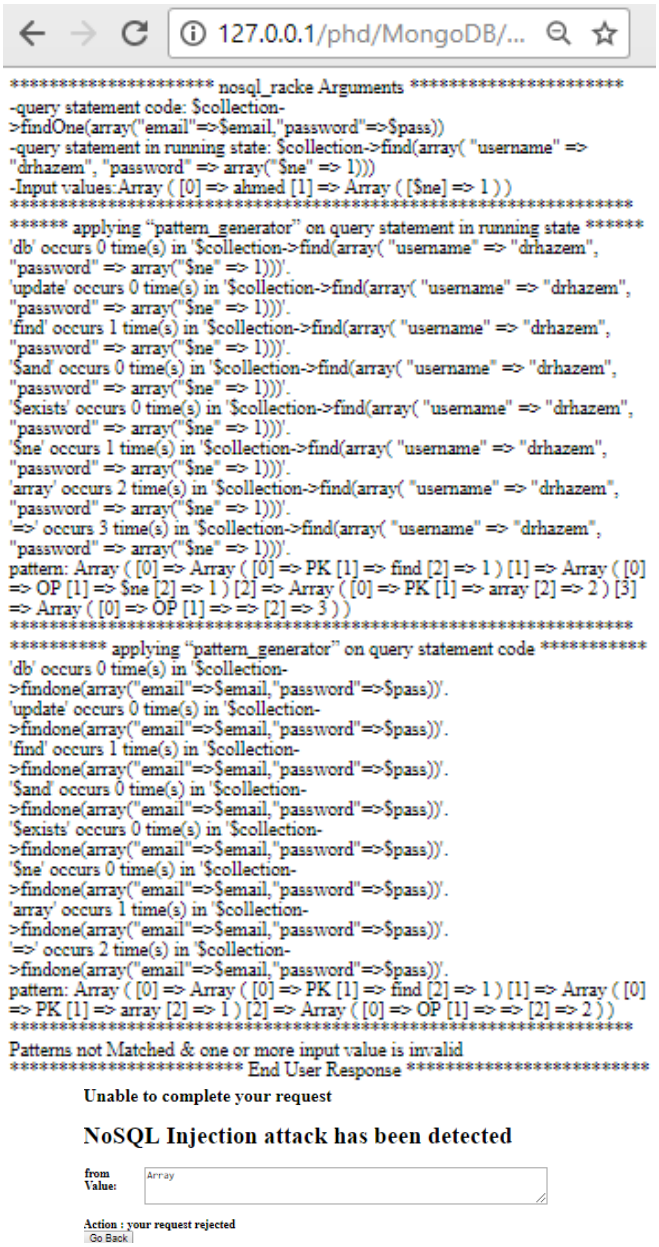

Fig. 9. "NoSQL Racket" testing results for MongoDB.

The login web page scanned by giving URL to Netsparker, Vega and Skipfish and none of them detect any issues related to NoSQL Injection. But when "NoSQL Racket" used, the NoSQL injection attack detected and testing results are figured out and shown in Fig. 9.

2) Cassandra Detection Ability Results: In Cassandra, The attacker may enter any user name and a password of:

ali'; DROP COLUMNFAMILY 'users

This results in a CQL query of:

('select * from reg_users where username = ali and password = ali'; drop columnfamily 'users', ['usern' => ,'passw' => ali'; drop columnfamily 'users])

The login web page scanned by giving URL to Netsparker, Vega and Skipfish and none of them detect any issues related to NoSQL Injection. But when "NoSQL Racket" used, the NoSQL injection attack detected and testing results are figured out and shown in Fig. 10.

Fig. 10. "NoSQL Racket" testing results for Cassandra.

3) CouchDB Detection Ability Results: In CouchDB, The attacker may enter any user name and a password of:

"or 1=1

This results in URL query of:

http://127.0.0.1/phd/CouchDB/after_log.php?user=test&pass=%27%27or+1%3D1&sbumit1=Submit

The login web page scanned by giving URL to Netsparker, Vega and Skipfish and none of them detect any issues related to NoSQL Injection. But when "NoSQL Racket" used, the NoSQL injection attack detected and testing result is figured out are shown in Fig. 11.

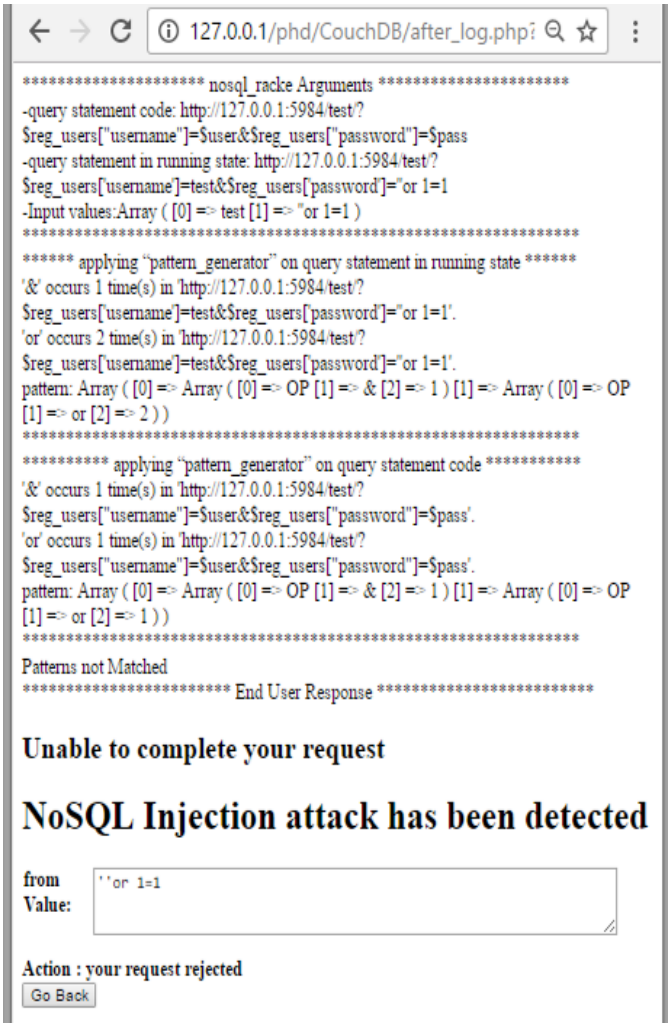

Fig. 11. "NoSQL Racket" testing results for CouchDB.

4) Amazon DynamoDB Detection Ability Results: In Amazon DynamoDB, it is possible to inject NoSQL keywords into submitted data from the login webpage. This could for example look like this:

http://127.0.0.1/phd/AmazonDynamoDB/after_log.php?user=ahmed&pass[$gt]=1&sbumit1=Submit

"$gt" selects those documents or keys where the value of the field is greater than (i.e. >) the specified value. Thus above statement compares password in database with empty string for greatness, which returns true. According to this example, the web application will allow the access to administration area to a user who doesn't know the proper password.

The login web page scanned by giving URL to Netsparker, Vega and Skipfish and none of them detect any issues related to NoSQL Injection. But when "NoSQL Racket" used, the NoSQL injection attack detected and testing result is figured out are shown in Fig. 12.

Fig. 12. "NoSQL Racket" testing results for Amazon DynamoDB.

The following Table 2 shows the comparison of detection ability for all testing tools with the proposed tool "NoSQL Racket" over the four NoSQL databases which are used in testing process.

TABLE II. COMPARISON OF DETECTION ABILITY FOR ALL TESTING TOOLS

Testing Tools \ NoSQL Databases	MongoDB	Cassandra	CouchDB	Amazon DynamoDB
Netsparker	×	×	×	×
Vega	×	×	×	×
Skipfish	×	×	×	×
The proposed tool "NoSQL Racket"	✓	✓	✓	✓

According to scanning results, the most common application injection scanners such as Netsparker, Vega and Skipfish not are able to detect any issues related to NoSQL Injection. However, the proposed implemented approach was able to detect the NoSQL Injection attack.

B. *Performance Testing for "NoSQL Racket"*

Performance testing is performed on the "NoSQL Racket" using LoadComplete testing tool. The LoadComplete testing tool is the desktop tool for load, stress, testing of website and web application.

The testing process is applied by increasing the number of concurrent users every one second. In this work test is firstly conducted for single user. Then number of concurrent users is increased by 50 concurrent users every one second. The testing environment consists of a machine running Windows 8.1- x64 with Intel core i7 processor and 8 GB RAM. The testing results can be shown in the following graphs:

Load Graph: The graph shown in Fig. 13 shows the relation between the number of concurrent users and the test execution time. As showed in the graph, it is observed that the proposed tool "NoSQL Racket" can load 50 simulated concurrent users every one second.

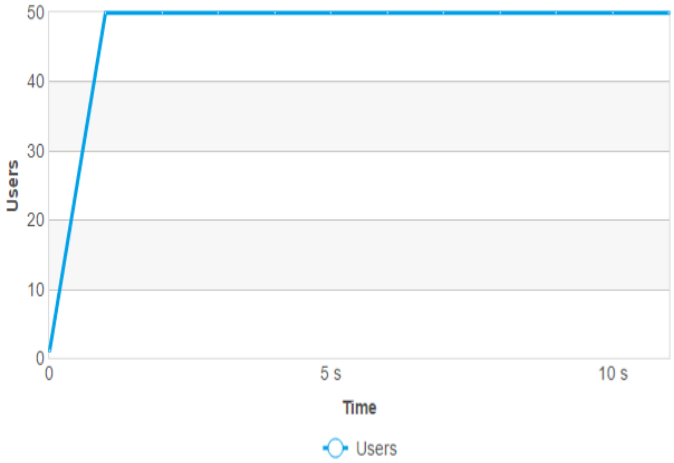

Fig. 13. Load graph.

Passed Requests Graph: The graph shown in Fig. 14 shows the relation between the number of concurrent users, the number of successfully passed requests and test execution time. As showed in the graph, it is observed that the proposed tool "NoSQL Racket" can pass 47 requests successfully from 50 requests every one second.

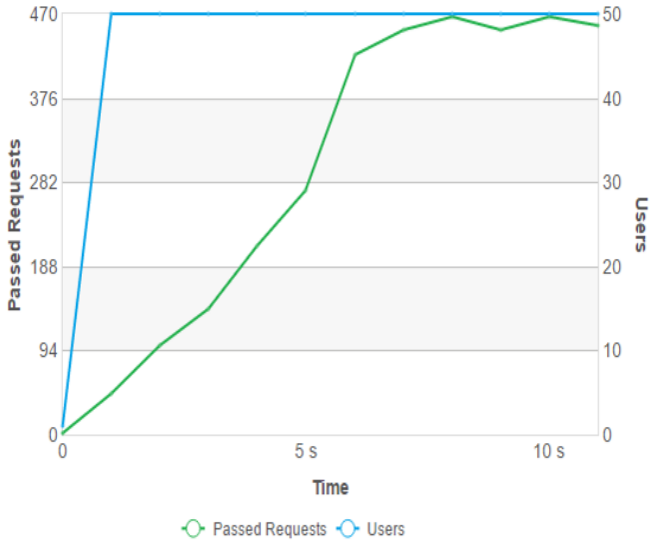

Fig. 14. Passed requests graph.

Warnings and Errors Graph: The graph shown in Fig. 15 shows the relation between the number of concurrent users, the number of web pages simulated with warnings and errors and the test run time. As showed in the graph, no any warnings or errors are detected.

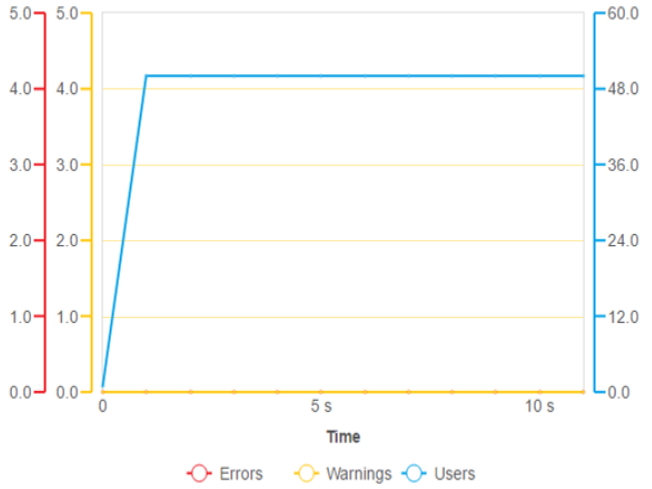

Fig. 15. Warnings and errors graph.

Page Load Time Graph: Page load time is the time period to download the web page content, including all the HTML tags, images, scripts, CSS files, and so on. The graph shown in Fig. 16 shows the relation between the page load time and the number of concurrent users. As showed in the graph, the maximum page load time is 850 ms and the average page load time is 75 ms.

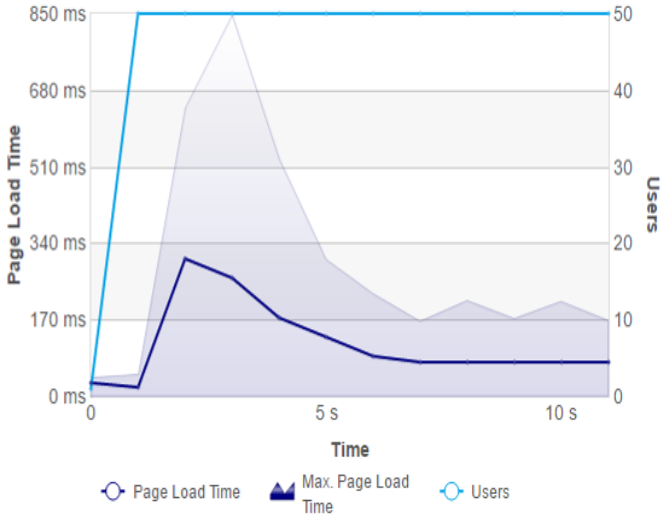

Fig. 16. Page load time graph.

Request Transfer Speed Graph: The request transfer speed refers to the speed of data transfer when the request was sent to the server.

The graph shown in Fig. 17 shows the relation between the number of concurrent users, the Request transfer speed metric and test execution time. As showed in the graph, the slowest speed for the requests transfer is 200 kB/s.

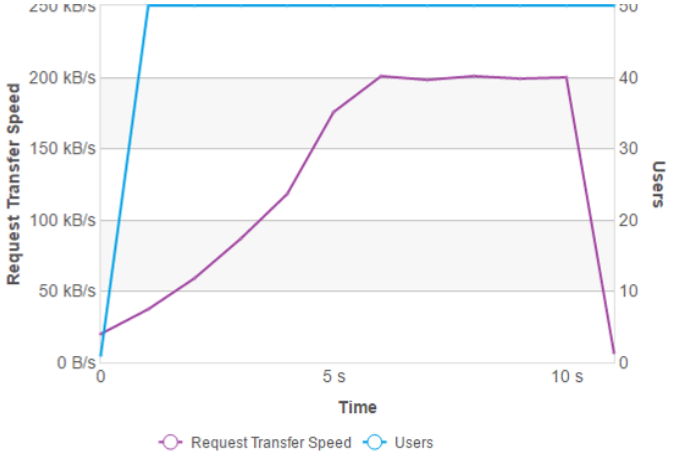

Fig. 17. Request transfer speed graph.

Response Transfer Speed Graph: The Response transfer speed refers to the speed of data transfer when the server sent back the response.

The graph shown in Fig. 18 shows the relation between the number of concurrent users, the Response transfer speed metric and test execution time. As showed in the graph, the slowest speed for the responses transfer is 1.52 MB/s.

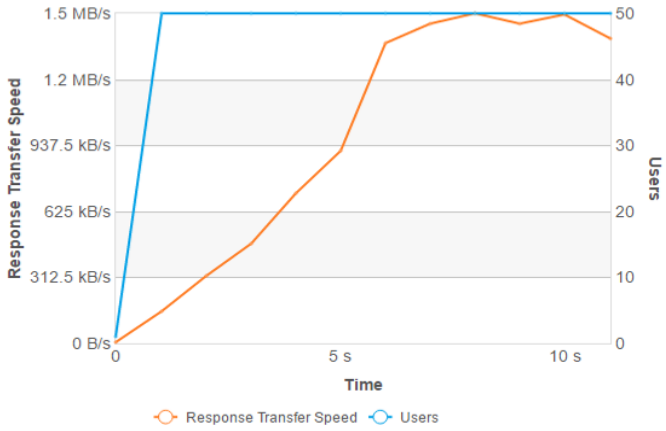

Fig. 18. Response transfer speed graph.

VII. Conclusion

This paper has presented a testing tool for detecting NoSQL Injection attacks which is called “NoSQL Racket”, this tool implemented as a PHP function. If no any NoSQL injection attacks detected, it will continue running for the nosql query; if it fails and one or more NoSQL injection attacks detected, it will display error page and stop running for the nosql query.

The proposed tool “NoSQL Racket” has been applied on four different NoSQL Databases which are MongoDB, Cassandra, CouchDB and Amazon DynamoDB. Also, its ability for detection and prevention has been compared with the most powerful web application testing tools which are Netsparker, Vega and Skipfish. According to the scanning results, none of mentioned tools has been able to detect NoSQL

Injection attack. However, the proposed implemented approach has the ability to detect the NoSQL Injection attack.

REFERENCES

[1] Okman L., Gal-Oz N., Gonen Y., Gudes, E., Abramov J, Security Issues in NoSQL Databases. Trust, Security and Privacy in Computing and Communications (TrustCom), pp.541:547, 2011.

[2] Nance Cory, Losser Travis, Iype Reenu, and Harmon Gary, NOSQL VS RDBMS - WHY THERE IS ROOM FOR BOTH. SAIS 2013 Proceedings 27, 2013.

[3] Aaron Schram, Kenneth M. Anderson, MySQL to NoSQL: data modelling challenges in supporting scalability. SPLASH '12 Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, 2012, pp.191:202.

[4] B. Sullivan, Server-side JavaScript injection, 2011. http://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf. Accessed 7 March 2017

[5] Sooel Son and Kathryn S. McKinley, Diglossia: Detecting Code Injection Attacks with Precision and Efficiency. 20th ACM Conference on Computer and Communications Security (CCS), pp.1181:1192, 2013.

[6] IBM Corporation, NoSQL does not have to mean no security: Data security and compliance best practices for NoSQL data systems, 2016. http://whitepapers.theregister.co.uk/paper/view/4306/nosql-does-not-have-to-mean-no-security.pdf. Accessed 7 March 2017

[7] Adrian Lane, Securing Big Data: Security Recommendations for Hadoop and NoSQL Environments, pp.4:6, 2012.

[8] Amreen and Dadapeer, A Survey on Robust security mechanism for NoSQL databases. International Journal of Innovative Research in Computer and Communication Engineering, pp.7662:7666, 2016.

[9] Aviv Ron, Alexandra Shulman-Peleg, Emanuel Bronshtein, No SQL, No Injection? Examining NoSQL Security", 36th IEEE Symposium on Security and Privacy 1, 2015.

[10] OWASP Top 10, web application security risks report for 2013. https://www.owasp.org/index.php/Top_10_2013-A1-Injection. Accessed 7 March 2017

[11] CWE/SANS Top 25 Most Dangerous Software Errors. http://cwe.mitre.org/top25/. Accessed 7 March 2017

[12] OWASP Top 10, web application security risks report for 2010. https://www.owasp.org/index.php/Top_10_2010-Injection. Accessed 7 March 2017

[13] OWASP Top 10, web application security risks report for 2017. https://www.owasp.org/index.php/Top_10_2017-A1-Injection. Accessed 1 November 2017.