The Never-Ending Vulnerability Cross-Site Scripting

Submitted by:

Diya Regmi(230052)

BSc(Hons) Ethical hacking and cyber security

Batch: 34

CUID: 14187193

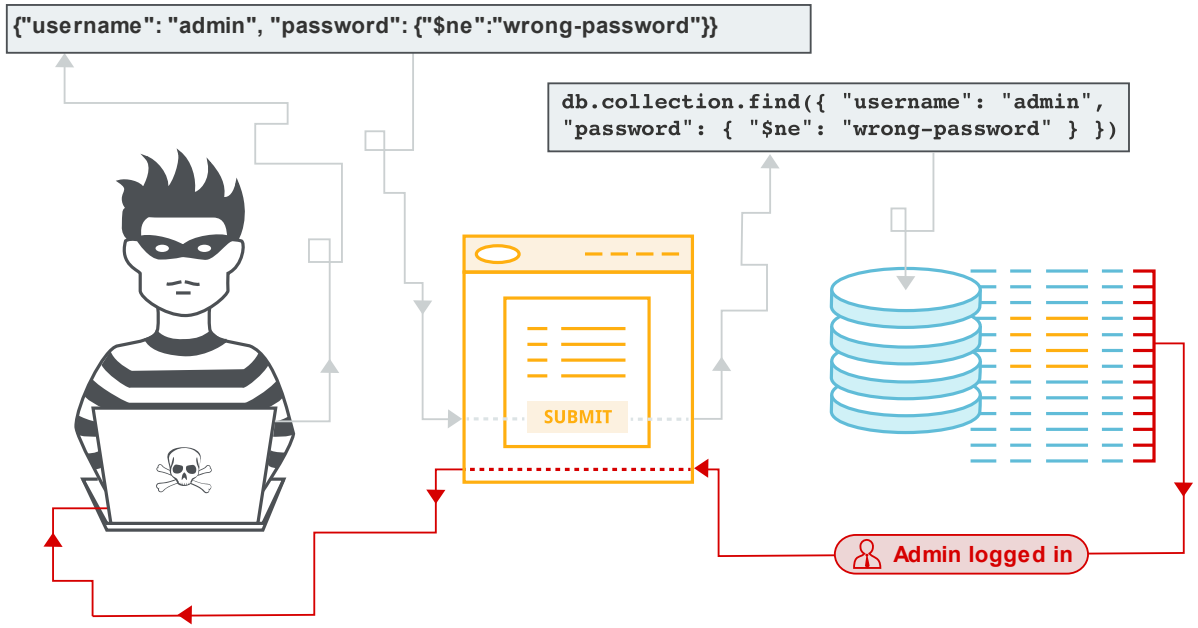
Submitted to:

Mr. Arya Pokharel

Softwarica College of IT and E-commerce

Introduction

A security bug known as "NoSQL (Not Only SQL) injection" targets NoSQL databases by taking advantage of holes in the way applications communicate with them. Attackers alter input data in a manner akin to SQL injection to deceive applications into carrying out unwanted database operations. NoSQL databases use diverse query languages and store data in different ways, similar to MongoDB. Through injection, an attacker can run code on the server, cause a denial of service, extract or alter data, and circumvent authentication. Using security mechanisms like parameterized queries, access limits, and input validation is part of prevention. Despite the speed advantages, NoSQL databases are still vulnerable to injection attacks, which could have a worse impact than with standard SQL injection because of the lack of consistency constraints. Protection needs to take into account the unique traits of NoSQL databases as well as the variety of formats they employ.



Types of NoSQl Injection

Syntax Injection: When an attacker modifies the query syntax in a NoSQL database, harmful payloads can be injected. This is known as syntax injection. Similar to SQL injection, however with different query languages and structures used by NoSQL databases, this is not the same.

Operator injection: This is the process of manipulating queries using NoSQL query operators.

NoSQL Injection in MongoDB Authentication: Security Risks

This is an elementary illustration that performs authentication by accessing a MongoDB database.

```
$username = $_POST['username'];
$password = $_POST['password'];
$connection = new MongoDB\Client('mongodb://localhost:27017');
if($connection) {
    $db = $connection->test;
    $users = $db->users;
    $query = array(
        "user" => $username,
        "password" => $password
    );
    $req = $users->findOne($query);
}
```

The query is directly filled in using the username and password parameters that are used for authentication, which are obtained from a POST request. This lets a malevolent user enter a NoSQL injection payload that grants unauthenticated access, just like other kinds of injection.

The following malicious input data could be sufficient to carry out a successful MongoDB injection if it is sent in a POST request:

```
username[$eq]=admin&password[$ne]=foo
```

Attackers can alter queries by using MongoDB query operators like [$ne](not equal), [$lt], [$gt], and [$regex], which increases the risk of unauthorized access and possible user enumeration. This presents a security risk in situations like illegal login attempts, where appropriate input validation is essential to preventing exploitation.

Relationship

Nature	Type	CAPEC-ID	Name
Child of	M (Meta Attack Pattern)	248	Command Line Injection
Can follow	D (Detailed Attack Pattern)	460	HTTP Parameter Pollution

Threat Vulnerabilities and Risk of NoSQL Injection

Threat

Applications using NoSQL databases are vulnerable to a serious security risk due to NoSQL injection. Vulnerabilities are used by attackers to change queries and jeopardize data integrity.

Vulnerabilities

- **Lack of Input Validation**: When applications do not adequately validate user inputs, attackers might inject harmful queries.
- **Inadequate Access constraints**: Unauthorized users may be able to take advantage of injection vulnerabilities due to weak access constraints.
- **Operator Manipulation:** NoSQL query operators like [$ne], [$lt], [$gt], and [$regex] can be used by attackers to modify queries.

Risks

- **Unauthorized Access:** Sensitive data may be accessed without authorization through NoSQL injection.

- **Data Manipulation:** Data integrity may be compromised by attackers who alter or remove data from a NoSQL database.
- **User Enumeration:** Information about a user can be enumerated by taking advantage of injection vulnerabilities.

0 to <3	LOW
0 to <6	MEDIUM
6 to 9	HIGH

TVR Matrix of NoSQL Injection

Threat Agent Factors					Vulnerability Factors			
Skill Level	Motive	Opportunity	Size		Ease of Discovery	Ease of Exploit	Awareness	Intrusion Detection
Overall Likelihood=								

Technical Impact					Business Impact			
Loss of Confidentiality	Loss of Integrity	Loss of Availability	Loss of Accountability		Financial Damage	Reputational Damage	Non-compliance	Privacy Violation
Overall Likelihood=								

Red Teaming on NoSQL Injection

To assess the security of systems, red teaming entails imitating actual cyberattacks. Finding vulnerabilities, flaws, and possible exploits in a NoSQL database architecture is the aim of a red team exercise on NoSQL injection. The red teaming procedure for NoSQL injection can be summarized as follows:

Attack Surface Identification

- Examine the architecture of the application and how it communicates with the NoSQL database.
- Determine the places where NoSQL queries are influenced by user input.

Modeling Threats

- Make a threat model that is unique to NoSQL injection and rank possible attack scenarios.

Techniques of Exploitation

- Make use of different NoSQL injection methods, including operator manipulation, payload injection, and syntax injection.

List of Database Schemas

- Examine how to use injection payloads to list the database structure.
- Determine methods for obtaining data regarding the NoSQL database's structure.

Extracting and manipulating data

- Try to use injection payloads to retrieve and modify data from the NoSQL database.
- Evaluate how the application will be affected if data extraction or manipulation is effective.

Abuse of NoSQL Query Operators

- Use NoSQL query operators to change queries and get unexpected answers, such as **[$ne], [$lt], [$gt],** and **[$regex].**

- Determine how well these operators can evade security measures.

Let’s take into consideration the following code, which verifies a user's identity using their username and password:

```
var user = db.users.find({"username": userInputUsername, "password": userInputPassword});
```

If they supply {"$gt": ""} as both the username and password, attackers can essentially change the query to:

```
var user = db.users.find({"username": "administrator", "password": {"$gt": ""}});
```

In MongoDB, the "$gt" operator denotes "greater than." The attacker gets around authentication by altering the value to an empty string since the query returns the first user whose username and password are both greater than an empty string.

Persistence and Stealth

- Test the viability of long-term attacks by inserting payloads that endure.
- Analyze how stealthy the attacks are to evade security system detection.

Record-keeping and Reporting

- Record the vulnerabilities that have been found, the scenarios of successful exploitation, and the possible consequences.
- Give specific suggestions for reducing the risk of NoSQL injection and enhancing security posture in general.

Injection Type	Vulnerable Input	Injected Query	Outcome
Syntax Injection	`username=admin' || '1'=='1' && password=admin`	`{ "username": "admin' || '1'=='1' && password=admin", ... }`	Bypass authentication by injecting true condition
Operator Manipulation	`{ "username": "admin", "password": { "$ne": null } }`	`{ "username": "admin", "password": { "$ne": null } }`	Authenticate without a valid password using `$ne`
Payload Injection	`{ "username": "admin", "password": { "$gt": "" } }`	`{ "username": "admin", "password": { "$gt": "" } }`	Retrieve data where password is greater than "

Blue Teaming on NoSQL Injection

The blue team concentrates on putting security procedures and instruments into place in the context of NoSQL injection to stop, identify, and react to such assaults. To reduce the danger of NoSQL injection, we can consider the following blue teaming techniques:

Input Validation and Sanitization

- Verify and eliminate any content including JavaScript, forbidden characters, and NoSQL keywords from user input.
- Check for expected types, apply character filters, and employ sanitization modules (such as "mongo-sanitize" for MongoDB).

Personalized Keyword Filtering for NoSQL

- Filter terms that are specific to NoSQL (such as $ne, $eq) to prevent injection attempts.

Parameterized Queries

- To keep user input distinct from the query structure and stop injection attacks, utilize prepared statements or parameterized queries.

Secure Request APIs

- To improve security, use secure APIs (such as **Flask-PyMongo** and **PyMongo**) rather than creating queries from strings.

Security of Query Operators (For PHP with MongoDB)

- Make sure all special query operators (beginning with $) in PHP when utilizing MongoDB utilize single quotes to guard against operator replacement attacks.

Principle of Least Privilege

- Restrict database user access to prevent overly privileged accounts for application connections and to reduce possible harm.

JavaScript Execution on the Server Side (For MongoDB)

- Disable server-side JavaScript execution while using MongoDB to reduce risks and improve security.

Monitoring and Auditing of Security

- Use comprehensive logs, live monitoring, and WAFs to identify and stop NoSQL injection patterns.

Frequent Security Assessments

- To identify and address vulnerabilities as soon as possible, do routine penetration tests and security evaluations.

Security Awareness and Training

- Development teams should receive security awareness training that promotes early reporting and timely resolution of security issues.

Personalized Error Pages

- To give general error information without revealing specifics about the database or application, create custom error pages.

Patch Management

- Regularly apply the most recent security updates for web servers, database systems, and application frameworks.

Incident Response strategy

- Create and manage an incident response plan specifically for NoSQL injection scenarios.

Red Team Collaboration

- Work together with red team efforts to develop practical NoSQL injection situations and improve security controls based on learned lessons.
-

Importance of Decision-Making

Scenario: E-commerce Database Injection using NoSQL

Background: A NoSQL database is used by a top e-commerce platform to handle inventories, transactions, and consumer profiles. The firm chooses to perform a Red Team exercise to evaluate the security of its application against potential NoSQL injection vulnerabilities since it is aware of the constantly changing threat landscape.

Red Team Objectives

The goal of the Red Team, which is made up of knowledgeable ethical hackers, is to locate and take advantage of NoSQL injection weaknesses in the e-commerce platform. Their goals encompass tampering with inquiries, introducing malevolent payloads, and maybe obtaining unapproved entry to confidential client information.

Blue Team Accountability

The cybersecurity defenders within the company make up the Blue Team, which is in charge of keeping an eye on and fending off the mock NoSQL injection attempts. They are responsible for spotting suspicious activity in real-time, putting safety precautions in place, and coming up with a plan of action to address any vulnerabilities found.

Red Team Decision-Making

i. **Payload injection in the Login page:** The Red Team makes the calculated decision to concentrate on inserting payloads into customer login page user input fields. By focusing on this crucial point of entry, they want to mimic situations in which hackers might alter user input to obtain unauthorized access

ii. **Modifying the Search Parameters**: Realizing how important product catalog queries are, the Red Team chooses to use NoSQL query operators to change search parameters. This method seeks to evaluate the system's resistance to manipulations that can affect search functions and possibly reveal private data.

iii. **Gradual Payload Complexity**: The Red Team chooses to gradually raise the complexity of inserted payloads to resemble a covert and sophisticated attack. By making this choice, they can test the system's defenses against increasingly complex and developing exploitation attempts, giving them a thorough assessment of the platform's security posture.

Blue Team Decision-Making

i. **Improved Input Verification:** When the Blue Team finds anomalous query patterns, they add more input validation rules in response. To lower the likelihood of successful injection attacks, they seek to prevent unauthorized query manipulation and guarantee that user input follows expected forms by strengthening input validation systems.

ii. **Real-time Traffic Monitoring**: Rather than depending only on static security measures, the Blue Team also uses real-time traffic monitoring to identify patterns related to NoSQL injection attempts. This choice enables them to quickly recognize and react to unusual activity, which improves their capacity to thwart possible attacks as they emerge.

iii. **Working together with development teams**: Understanding the value of prompt action, the Blue Team works closely with development teams. The goal of this cooperative effort is to patch discovered vulnerabilities as soon as possible. The Blue

Team guarantees that security solutions are swiftly put into the program, minimizing the window of exposure to potential attacks, by closely collaborating with developers.

Communication, Teamwork, and Leadership

Red Team Communication

The NoSQL Injection Red Team Exercise depends heavily on effective communication. Members of the team make sure that everyone is aware of the goals, parameters, and particulars of the engagement. They discuss and determine the best attack vectors, attack techniques, and tools through open communication. Throughout the engagement, real-time communication allows the team to adjust strategy in reaction to changing circumstances, providing flexibility in the face of unforeseen challenges. Effective collaboration among team members and coordination of strategies are essential for a successful evaluation.

Read Team Teamwork

Collaborative Execution: By collaborating across functional boundaries to combine a variety of talents, views, and abilities, the Red Team's actions are more successful.

Creative Problem-Solving: Team members promote creative problem-solving by recognizing shortcomings from many perspectives, which cultivates a dynamic and adaptable approach.

Quality assurance: By constructively criticizing and improving each other's work, the team builds its collective intelligence and produces comprehensive evaluations and well-structured reports.

Effective Work Delegation: Adaptive work delegation makes sure that responsibilities are allocated in an efficient manner, which improves engagement execution and raises security assurance.

Red Team Leadership

The Red Team's operations are guided and facilitated by strong leadership. Setting clear goals that define the parameters and objectives of the engagement and guarantee consistency with the organization's security standards is the leader's responsibility. The leader wisely allocates staff,

equipment, and time among other resources. To ensure that the Red Team's activities are coordinated and improve the organization's overall security posture, the leader gives strategic direction.

Read Team Outcome

NoSQL injection vulnerabilities are successfully found thanks to the Red Team's cooperative efforts in leadership, teamwork, and communication. Effective communication guarantees that each member of the team is aware of their responsibilities, and collaboration enables a thorough evaluation from many angles. Strategic direction is given by leadership, which helps the team accomplish particular objectives and enhances organizational security. These components work together to improve the platform's resistance to NoSQL injection threats.

Blue Team Communication

The Blue Team creates unambiguous lines of communication in the event of a possible NoSQL injection attack. System administrators, security analysts, and other team members are kept informed about the type and extent of the threat through frequent updates and meetings. A shared understanding is cultivated by this open communication, which makes it possible for the team to act quickly and efficiently.

Blue Team Teamwork

- *Diverse Skill Sets*: The Blue Team makes use of a variety of talents, such as database managers, security analysts, and IT specialists.
- *Collaborative Workflow:* Team members combat NoSQL injection issues by collaborating effectively and sharing their specialized knowledge.
- *Comprehensive Vulnerability Analysis:* By working together, it is possible to conduct a full study that covers every facet of the NoSQL injection threat.
- *Effective Issue Mitigation:* By pooling their particular strengths, the team can identify, assess, and resolve NoSQL injection vulnerabilities more successfully, improving the organization's overall security posture.

Blue Team Leadership

The Blue Team needs strong leadership to give strategic direction and guarantee a coordinated reaction. Tasks are prioritized by leaders according to the seriousness of the NoSQL injection vulnerabilities. They also effectively distribute resources and assist the team in formulating preventative measures. Strong leadership enables the Blue Team to take long-term steps to fortify the system's defenses against potential attacks in addition to addressing urgent issues.

Blue Team Outcome

A solid defense against NoSQL injection vulnerabilities is the result of the Blue Team's coordinated efforts, which are marked by efficient communication, seamless coordination, and strong leadership. Increased capacity to identify and address new security threats is gained by the organization. The Blue Team works together to guarantee the continued security of the company's systems and data by using flexible tactics and constant communication.