EAST: Exploitation of Attacks and System Threats in Network

Sachin Ahuja, Rahul Johari and Chetna Khokhar

Abstract In modern era, computer network is an emerging field. With the invention of powerful computer network concepts today we are able to share information with each other. We build complex systems so that user can use these systems with ease. But with this comes the question of security. There comes a question, is the data that we share safe? Complex systems built with the intent to use shared data for example such as social networking sites certainly have some security loopholes. The most important as well as the most difficult task of a developer is to ensure that whatever the situation be, the system is consistent. But, as a matter of fact, no developer can guarantee that. Systems do possess some vulnerabilities. In the work that follows, we have tried to explore some prevalent system vulnerabilities and network attacks. Using JAVA as a programming language we have shown the flaws/shinks in the Web Programming and successfully simulated the vulnerabilities and attacks and demonstrated encouraging results.

Keywords Dictionary attack · DoS attack · Brute force attack · Threat

1 Introduction

To begin with, System/Network security is a critical area which has become a hot buzz in the recent years owing to the enormous number of the web site/applications getting developed and hosted on the web server (both real and rogue). Some of these sites are poorly coded by the programmers as a result they become easy fish

S. Ahuja (✉) · R. Johari · C. Khokhar
USICT, GGSIP University, Dwarka, Delhi, India
e-mail: sachin.ahuja1992@gmail.com

R. Johari
e-mail: rahuljohari@gmail.com

C. Khokhar
e-mail: chetnakhokhar92@gmail.com

J.K. Mandal et al. (eds.), *Information Systems Design and Intelligent Applications*,
Advances in Intelligent Systems and Computing 339,
DOI 10.1007/978-81-322-2250-7_60

for millions of hackers and crackers looking for the vulnerabilities in the Web Sites, so that they can launch massive attacks and take over the control of these sites hosting millions of Apps. We have worked on various system vulnerabilities (password ageing, empty string password, empty catch block problem etc.) and on the network attacks (Denial of Services, Dictionary attack and Brute force attack). With the help of programming we have tried to explore these attacks and vulnerabilities. For better understanding we have carried out the mathematical modeling of the network. We have also discussed the problems that a system may face if it is not safe against these loopholes. For completeness and clarity the paper is organized as follows: Sect. 2 discusses about Objectives, Sect. 3 discusses about the related work, Sect. 4 discusses about the Simulation performed, Sect. 5 discusses about the methodology adopted, Sect. 6 discusses about the mathematical modeling of network attacks, Sect. 7 discusses about the result, Sect. 8 discuss about conclusion and future work. Section 9 contains all the figures followed by acknowledgement and references section.

2 Objective

The Open Web Application Security Project (OWASP) [1, 2] is a worldwide not-for-profit charitable organization focused on improving the security of software. The same has listed various vulnerabilities and network attacks. We through our work have tried to explore these vulnerabilities and attacks. We have tried to explore different security loopholes that a system may possess. A developer while developing any system should ensure that system is safe against these vulnerabilities. The system should also protect itself against an attack made by an attacker with the intent to gain control of the system.

3 Related Work

Huluka and Popov use Root Cause Analysis (RCA) in session management and broken authentication Vulnerabilities and identify the way to improve different aspects of security of web applications. Through RCA they found 9 root causes that lead to broken authentication vulnerability and 11 root causes that lead to session management vulnerability and they also provide deep detailed view of vulnerabilities, which results in effective solutions. These solutions are used to minimize the recurrence of attacks on web applications [3]. Fonseca et al. present a prototype tool and methodology for the evaluation of security mechanism of web applications. The idea behind their methodology is that they assess the existing mechanisms of security and tools in different scenarios by injecting realistic vulnerabilities in an application and attacking them. They also propose the Vulnerability and Attack Injector Tool (VAIT) which automates the entire process. They have shown

the effectiveness of proposed methodology by running the tool on set of experiments. The results of their research proved that the methodology proposed by them is an effective way not only for the evaluation of weakness of security mechanism but also helps in identifying the ways of its improvement [4]. Sadeghian et al. presents a comprehensive review of different types of SQL injection detection and prevention techniques. They made the detailed analysis of all the techniques and provided the strengths and weaknesses of each technique. The structural classification of the SQL injection detection and prevention techniques assists other researchers in the adoption of correct technique for their studies [5]. In [6] author(s) demonstrate the comparative performance analysis of MD5, DES and AES encryption algorithms [1] on the basis of execution time, LOC (Lines of Code) over a web application. In [7] author(s) discusses and analyzes the current developments in online authentication procedures including one-time-password systems, biometrics and Public Switched Telephone Network for cardholder authentication. The author(s) proposes a complete new framework for both onsite and online (Internet shopping) credit card transactions. In [8, 9] author(s) presents a detailed review on various types of vulnerabilities, Structured Query Language Injection attacks, Cross Site Scripting Attack, and prevention techniques. The Author(s), also proposes future expectations and possible developments of countermeasures against Structured Query Language Injection attacks. In [10] author(s) presents an integrated model to prevent reflected cross site scripting attack and SQL Injection attacks in applications which are made in PHP. These models work in two modes which are production and safe mode environment. They create sanitizer model for reflected cross site scripting attack and security query model for SQL Injection attack in safe mode. They validate user input text against sanitizer model and input entries which create SQL queries are validated against security query model in production mode. In [11] author(s) demonstrates the exploitation of web vulnerabilities in a credit card validation web application using brute force and dictionary attack. In [12] author(s) also proposes a similar technique to handle the security of the alphabets and numbers but without any detailed comparison. In [13] author(s) proposes a technique to encrypt and decrypt the Alphabets, Numbers and Alphanumeric data in minimum span of time with minimum lines of code, designed logic of which has been coded in JAVA. In [14] Scholte et al. represents IPAAS, a novel technique. This technique is based on automated detection of data type of input parameters which successfully prevents the exploitation of XSS and SQL injection vulnerabilities [15]. They implemented this technique for PHP applications and also analyzed the performance of this technique by running this technique on five real-world web applications. Their technique successfully prevented 65 % of XSS vulnerabilities and 83 % of SQL injection vulnerabilities. In [16] author(s) have designed a Java based tool to show the exploitation of Injection using SQL Injection attack [1] and Broken Authentication [2] using Brute Force Attack and Dictionary Attack and the prevention of all these attacks by storing the data in our database in encrypted form using AES algorithm [17].

4 Simulation Performed

We have used java programming paradigm as a tool for exploring the above discussed attacks and vulnerabilities. We have used JAVA DEVELOPMENT TOOLKIT as a tool for stimulating the same. The results along with the outputs are discussed above. We have used Microsoft Access to build our own database and used Microsoft Excel to draw the plots that we have shown in our work.

5 Our Methodology

We have divided our methodology into two parts:

5.1 Methodology for Exploring Vulnerabilities

5.1.1 Vulnerability

Although many definitions of vulnerabilities exist. But, simply speaking it is a 'flaw' or a Programming bug committed by a novice programmer. If a system has a flaw, the attacker can access that flaw and will take advantage of this flaw to reduce system assurance and reliability. The different vulnerabilities are explained below:

Password Ageing

Password ageing can result in the possibility of diminished password integrity. If a user does not change his/her password for a long period of time, his/her user account can be tracked by any person and make his account insecure. If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner. Therefore, support for password ageing mechanisms must be added in the design phase of development.

Empty String Password

Using an empty string as a password can make an account insecure. It is never a good idea to assign an empty string to a password variable. If the empty password is used to successfully authenticate against another system, then the corresponding account's security is likely to be compromised because it accepts an empty password. Thus, a constraint should be imposed that the user should not make empty string as password and if he does there should be an error 'empty string password…'.

Empty Catch Block Problem

It is usually a bad idea having an empty catch block. When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution. Instead, one should catch the exact exception types that he expects because these are the types the program code is prepared to handle. For example, concept of dangling pointers can be a consequence of empty catch block problem.

5.2 Methodology for Exploring Network Attacks

5.2.1 Network Attacks

In computers, a network attack is any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of a resource (say data). The attacker tries to invade into the system to destroy it. Any attempt made for such intent is called as a network attack. Various network attacks are explained below:

Denial of Service

DOS stands as an acronym for "Denial of Service". This is caused due to the massive traffic at the server end. The large amount of traffic causes the throughput of the server to reduce. The large amount of requests causes the server to go busy. The high amount of traffic causes the server to response abnormally. At the end the server will not be able to respond to the clients properly. As a result of which the following two scenarios can occur:

(i) The response time of server for client reduces drastically due to which system throughput and efficiency decreases.
(ii) The system hangs due to increased traffic as a result of which the system crashes.

Figures 1 and 2 show a client server architecture. Figure 1 represents server window which is ready to receive request from clients. Clients in turn try to request to server. As a client is connected to server, a "Hello Client" message is sent on client window by server as a confirmation. It has been found that in DOS, the Server is bombarded by millions of requests, some genuine and the rest bogus/fake, resulting in collapse of the Services provided by the Server.

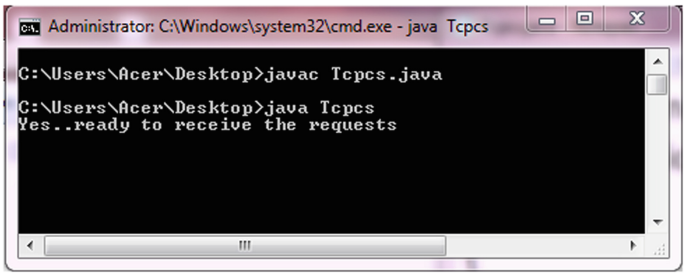
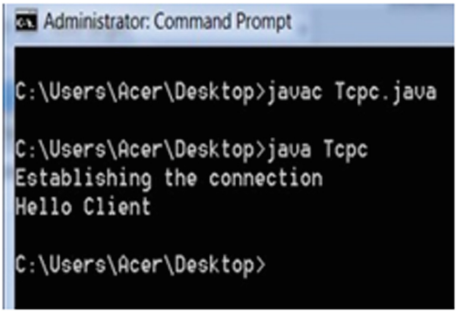


Fig. 1 Server side window for denial of service attack

Fig. 2 Client side window for denial of service attack



Dictionary Attack

The dictionary attack is a type of attack in which a hacker can spy on the system by using a pre designed list of probable passwords. The programmer can develop a database consisting of these probable passwords. These passwords can be formed by reviewing the profile of the person. A Database was created in MS Access and populated with hundreds of ID's bearing Username and Password. The above formed list of passwords can be compared against the password entered by the programmer. If the password entered by the user matches any of the password in the list then the system is at risk and it can easily be attacked by an experienced hacker. We have implemented dictionary attack using a Microsoft Access database which is connected to a Java program using jdbc-odbc drivers. Figure 3 shows the simulation of Dictionary Attack being successful. The username and password entered by the user in a java form is matched in the usernames and passwords present in the database. The program of "Dictionary Attack" has been executed and the "number of iterations" versus "length of the password" has been plotted as shown in Fig. 4. It can be seen from the plot that the number of iterations to find a given password does not depend on the length of the string entered. Rather, it depends on the number of

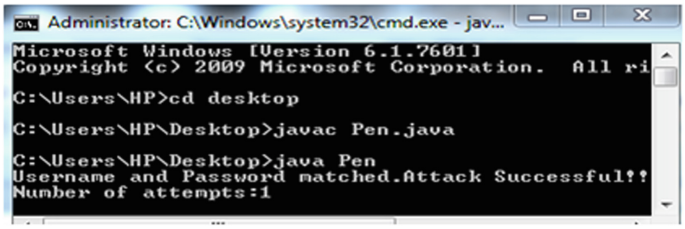


Fig. 3 Execution of dictionary attack showing the attack being successful

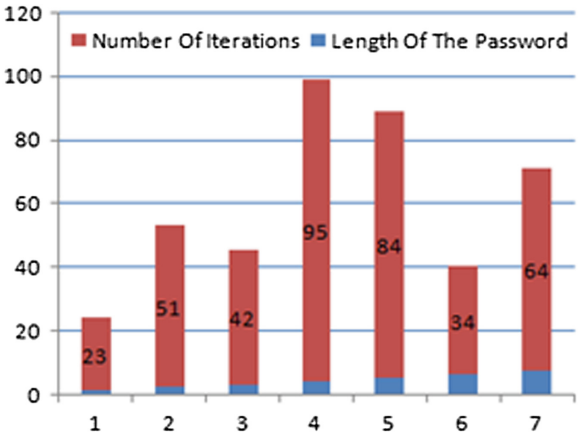


Fig. 4 Histogram for dictionary attack showing number of iterations versus length of the password

records that the database contains. We have executed this program by taking a small database of about 100 records where each record consists of characters (a-z) from UNICODE character set. If the entered password is not there in the database we have to iterate a maximum of iterations equal to the number of records in the database.

Brute Force Attack

The brute force attack is one of the best attacks if one wants to analyze the password sensitivity of a system. In this attack we have built a character set and the program tests all the possible combinations of characters present in the character set against the password entered by the user. If the password entered by the user matches with any of the combinations as discussed above the password is said to be cracked and the system is at risk. The simulation of same has been shown in Fig. 5. The program of "Brute Force Attack" has been executed and the "number of iterations*0.0001" versus "length of the password" has been plotted as shown in Fig. 6. It can be seen from the plot that the number of iterations to find a given

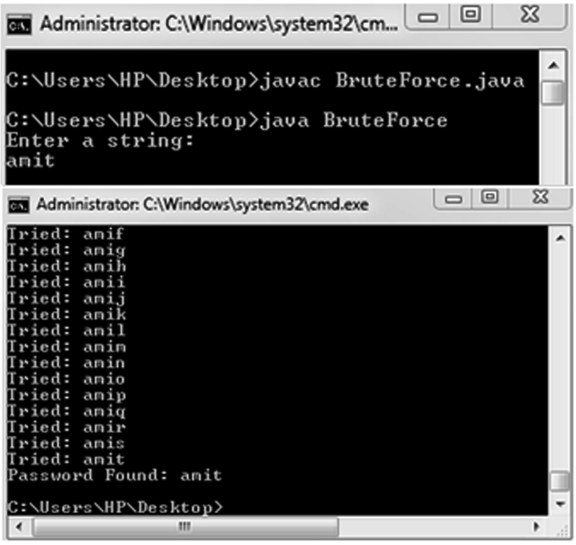


Fig. 5 Execution of brute force attack being successful

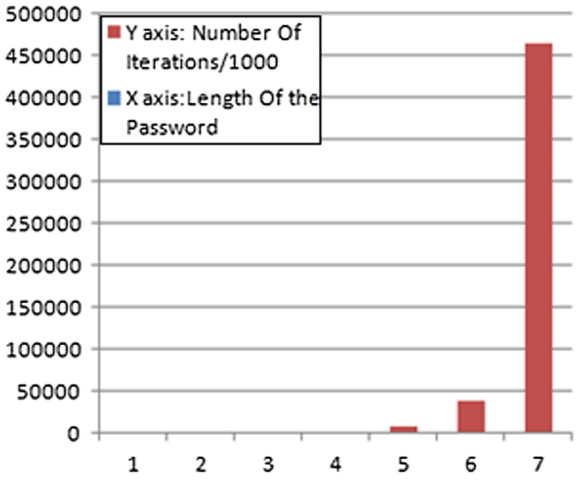


Fig. 6 Histogram showing number of iterations versus length of the password for brute force attack

password depends on the length of the string entered. The password entered by user is checked against all the possible combinations of characters (a-z) from UNICODE character set. Brute force is a stronger attack than Dictionary attack because it has a very high probability of cracking the password. It can be seen that as the length of the entered password increases, the number of iterations required to break the password also increases. The security of a system can be increased if we put some password restrictions and make the length of the password greater than or equal to 7 characters. This is so because it will hinder the attacker from cracking the password in that transient amount of time when the system is momentarily at rest.

6 Mathematical Modeling of Network Attacks

6.1 DoS

l ← Number of legitimate requests.; q ← Number of bogus requests

r ← Total number of requests generated; s ← Total number of requests being received at the server.; P ← Performance of the server at time (t).

DOS()

{if q ≫ l{P dips, affecting throughput due to the success of DOS attack} else if (l ≫ q){The attack was un successful, P and throughput increases} else{Server works at maximum speed and delivers at consistent throughput.} end if; end if}.

D ← Difference between number of legitimate requests and number of bogus requests. D = l − q. As the number of fake requests (q) increases the system throughput decreases. Simply put, as q increases the number of legitimate requests handled by the server decreases per unit time. Therefore the difference between the number of legitimate and bogus requests is the key parameter to represent the time complexity of the algorithm discussed above. P (Performance of server) = Θ(D) = Θ(l − q).

6.2 Brute Force Attack

```
Note: Our character set contains characters a-z.
P ← Plain text string to be searched.
S ← String which is compared with the pattern to be searched.
n ← Length of the string P
flag ← true, if attack is successful, otherwise false
Brute_Force_Attack()
{ for i =1 to n
    Compare all the combinations of S of length i with P
      if(S==P){ flag = true }  //Attack Successful
      else{flag = false}   //Attack not successful
end for }
```

Time Complexity: Suppose, if the length of the string P is 1 the number of maximum attempts will be 26 because the size of character set is 26. Now if size of P is 2 the number of possible combinations will be (26 + 26^2). Similarly, when the size of the P to be searched is "n" the number of combinations will be: (26 + 26^2 + 26^3 + ⋯ + 26^n). If we use the rules of asymptotic notations 26^n will be the dominant factor. Therefore in our case: Time Complexity = Θ (26^n). Now it is logical to say that because the size of character set was 26 in our case, there complexity of the algorithm comes out to be 26^n. However if the number of characters in the character set is m, then: **Time complexity**: Θ (m^n); **Space complexity**: num*n. Here: num are the number of bytes per symbol by the encoding scheme being employed according to the architecture.

6.3 Dictionary Attack

```
P ← Plaintext to be searched ; N ← Number of records in dictionary
i ← Number of iterations.
Assumptions : Threshold value for number of attempts to search N.
Dictionary_Attack()
{   for  i=1 to N
     R ← i^th record in the dictionary
        if ( R is equal toP){Record matched. Search successful}
       else  if  (i>n){Record not matched. Search unsuccessful}
            end if ; end if;  end for }
```

Time Complexity: O(N). The time taken to search for the plaintext will be a linear search (according to our algorithm) in the dictionary and will be of order N.

7 Result

Various attacks and vulnerabilities given by OWASP such as Denial of service (DOS) Attack, Brute Force Attack, and Dictionary Attack have been executed using Java programming platform. The results of various simulations have been shown wherever necessary and different graphs have been plotted.

8 Conclusion and Future Work

Various vulnerabilities and attacks have been explored. We suggest that for developing a secure and a reliable system the developer while writing code should take care of these vulnerabilities during the development phase of SDLC (software development life cycle). Otherwise system and data integrity may be at risk and the system will have some loopholes which an attacker can use to invade into the system. These are not the only vulnerabilities or attacks. There are many. In our future work, we will be exploring other vulnerabilities and attacks and will also be giving the solutions for the vulnerabilities and attacks discussed in this paper.

9 Figures

See Figs. 1, 2, 3, 4, 5 and 6.

References

1. https://www.owasp.org/index.php/Top_10_2013-A1-Injection
2. https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management
3. Huluka, D., Popov, O.: Root cause analysis of session management and broken authentication vulnerabilities. In: IEEE, pp. 82–86 (2012)
4. Fonseca, J., Vieira, M., Madeira, H.: Evaluation of web security mechanisms using vulnerability and attack injection. In: IEEE (2013)
5. Sadeghian, A., Zamani, M., Manaf, A.A.: A taxanomy of SQL injection detection and prevention techniques. In: IEEE, pp. 53–56 (2013)
6. Johari, R., Jain, I., Ujjwal, R.L.: Performance analysis of MD5, DES and AES encryption algorithms for credit card application. In: International Conference on Modeling and Computing, ICMC (2014)
7. Gupta, S., Johari, R.: A new framework for credit card transactions involving mutual authentication between cardholder and merchant. In: 2011 International Conference on Communication Systems and Network Technologies (CSNT), pp. 22–26. IEEE (2011)
8. Johari, R., Sharma, P.: A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. In: 2012 International Conference on Communication Systems and Network Technologies (CSNT), pp. 453–458. IEEE (2012)
9. Johari, R., Gupta, N.: Secure query processing in delay tolerant network using java cryptography architecture. In: International Conference on Computational Intelligence and Communication Networks (CICN), pp. 653–657. IEEE (2011)
10. Sharma, P., Johari, R., Sarma, S.S.: Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. In: International Journal of System Assurance Engineering and Management, pp. 343–351. Springer, Belin (2012)
11. Jain, I., Johari, R., Ujjwal, R.L.: Web vulnerability exploitation using brute force attack and dictionary attack. In: Proceedings of 9th National Conference on Smarter Approaches in Computing Technologies and Applications (2014)
12. Ruby, L., Johari, R.: Designing a secure encryption technique for web based application. Int. J. Adv. Res. Sci. Eng. (IJARSE) **3**(7), 159–163 (2014)
13. Ruby, L., Johari, R.: SANE : secure encryption technique for alpahamuneric data over web based applications. Int. J. Eng. Res. Technol. (IJERT) **3**(8), (2014)
14. Scholte, T., Robertson, W., Balzarotti, D., Kirda, E.: Preventing input validation vulnerabilities in web applications through automated type analysis. In: IEEE 36th International Conference on Computer Software and Applications, pp. 233–243 (2012)
15. https://www.owasp.org/index.php/SQL_Injection
16. Jain, I., Johari, R., Ujjwal, R.L.: CAVEAT: credit card vulnerability exhibition and authentication tool. In: Second International Symposium on Security in Computing and Communications (SSCC'14), pp. 391–399. Springer, Berlin (2014)
17. http://en.wikipedia.org/wiki/Advanced_Encryption_Standard