5-2018

Prevention of SQL Injection Attacks using AWS WAF

Mohammed Kareem
St. Cloud State University, makareem0618@gmail.com

Prevention of SQL Injection Attacks using AWS WAF

by

Mohammed Abdul Kareem

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in

Information Assurance

May 2018

Starred Paper Committee:
Susantha Herath, Chairperson
Dien D. Phan
Balasubramanian Kasi

Abstract

SQL injection is one of several different types of code injection techniques used to attack data driven applications. This is done by the attacker injecting an input in the query not intended by the programmer of the application gaining the access of the database which results in potential reading, modification or deletion of users' data. The vulnerabilities are due to the lack of input validation which is the most critical part of software security that is often not properly covered in the design phase of the software development lifecycle. This paper presents different techniques and some of the countermeasures for detection and prevention of SQL injection attacks. The proposed procedure in the paper is to use a database firewall between the client (user) side and the database server through AWS to avoid the malicious codes injected by the attackers.

Table of Contents

List of Tables 6

List of Figures 7

Chapter I: Introduction 8

Introduction 8

Problem Statement 9

Nature and Significance of the Problem 9

Web Application Environment 10

Objective of the Research 12

SQL Injection Attack Overview 12

Applications Vulnerable to SQL Injection 14

Summary 16

Chapter II. Background and Literature Review 17

Introduction 17

Literature Related to the Problem 18

Literature Related to the Methodology 20

Types of SQL Injection Attacks 21

Tautologies 21

Illegal or Logically Incorrect Queries 22

Union Query 24

Piggy-Backed Queries 25

Stored Procedures 26

Inference 28
Blind Injection 29
Timing Attacks 29
Alternate Encodings 31
Main Causes of SQL Injection 33
Detection and Prevention Techniques 35
Injection Detection at the Web Tier 39
Summary 40
Chapter III. Methodology 41
Introduction 41
Design of the Study 42
Data Analysis 44
Summary 45
Chapter IV: Analysis of Results 46
Introduction 46
Data Presentation 47
Compute 47
Storage 49
Security 51
Networking 52
Data Analysis 52
Overview for IPv4 57

Overview for IPv6 58
Create a VPC 64
Template 1 69
Create an Amazon S3 Bucket 76
Template 2 76
Summary 80
Chapter V: Conclusions and Future Work 81
Results 81
Conclusion 82
Future Work 82
References 83

List of Tables

4.1 Routing for IPv4 60

4.2 Routing for IPv6 61

4.3 Security for IPv4 62

4.4 Security for IPv6 64

List of Figures

1.1 Web Application Firewall 10

1.2 Web Tier Environment 11

1.3 SQL Injection Attack Overview 14

3.1 AWS WAF Architecture 43

4.1 Configure Instance Details 53

4.2 Configure Security Group 54

4.3 Launsh an Instance 55

4.4 Change Security Group 56

4.5 Overview for IPv4 58

4.6 Overview for IPv6 59

Chapter I: Introduction

Introduction

In today's world where almost every task is performed through web applications such as banking, online shopping, and bill payments we entrust our personal information to these web applications and their underlying databases because of the trust on the confidentiality and integrity of the security of their data. As the usage of these services is increasing day by day on a large scale we are also facing a devastating increase in the number of attacks which can potentially give an attacker complete access to an individual's database such as one containing credit card information underlying the secured database.

SQL injection attacks (SQLIAs) are the most effective and malicious system attacks which can be used to gain or manipulate the data in data-driven systems. The risk of SQLIAs is that when they are performed by the victim back-end system, they will be running with the same privileges that the system has in the database, that means if the system has been assigned a role as a power user or administrator which has the read and write permissions then the injection code could be executed with disastrous effect on the victim machine.

A SQL Injection attack (SQLIA) is one in which a malicious minded person injects their own crafted query as an input and replaces the default query. The backend server executes the injected query statement and sends the result to the attackers. Therefore, most of the attackers use SQL for accessing the database and for the detection and prevention of these attacks various tools have been developed. There are multiple types

of SQLIA's and each one of them has a different approach and effect for attacks on the target website. To counter these attacks, we will be extensively discussing some of the modern SQL Injection attacks and the ways to protect and defend against these types of attacks. The negligence at the initial stage of development can lead to monetary losses at later stages.

Problem Statement

Because of the large variation in the pattern of SQL injection attacks the use of a Web Application Firewall (WAF) is often unable to protect the databases from attack. Besides, it is very difficult for startups & small business firms to meet the high-end capital and time requirements for the installation and maintenance of a database firewall.

Nature and Significance of the Problem

One of the most commonly used approaches to identify SQL injection attacks is using WAF (Web Application Firewall). A WAF which operates in front of the Web servers monitors the traffic which goes in and out of the Web servers and attempts to identify patterns that constitute a threat. While this can be effective in detecting certain classes of attacks against Web applications, it has proven ineffective in detecting all but the simplest SQL injection attacks.

Considering the poor detection of the SQL injection attacks and because of the high-end capital and time-consuming prerequisites for maintaining a WAF will not be that useful in the Web security environment. On the other hand, WAFs provide reasonable protection from header injection, XSS (Cross-Site-Scripting) attacks and

many more simple attacks. Considering the additional benefits of a WAF it should always be considered as a part of Web security defense in depth strategy.

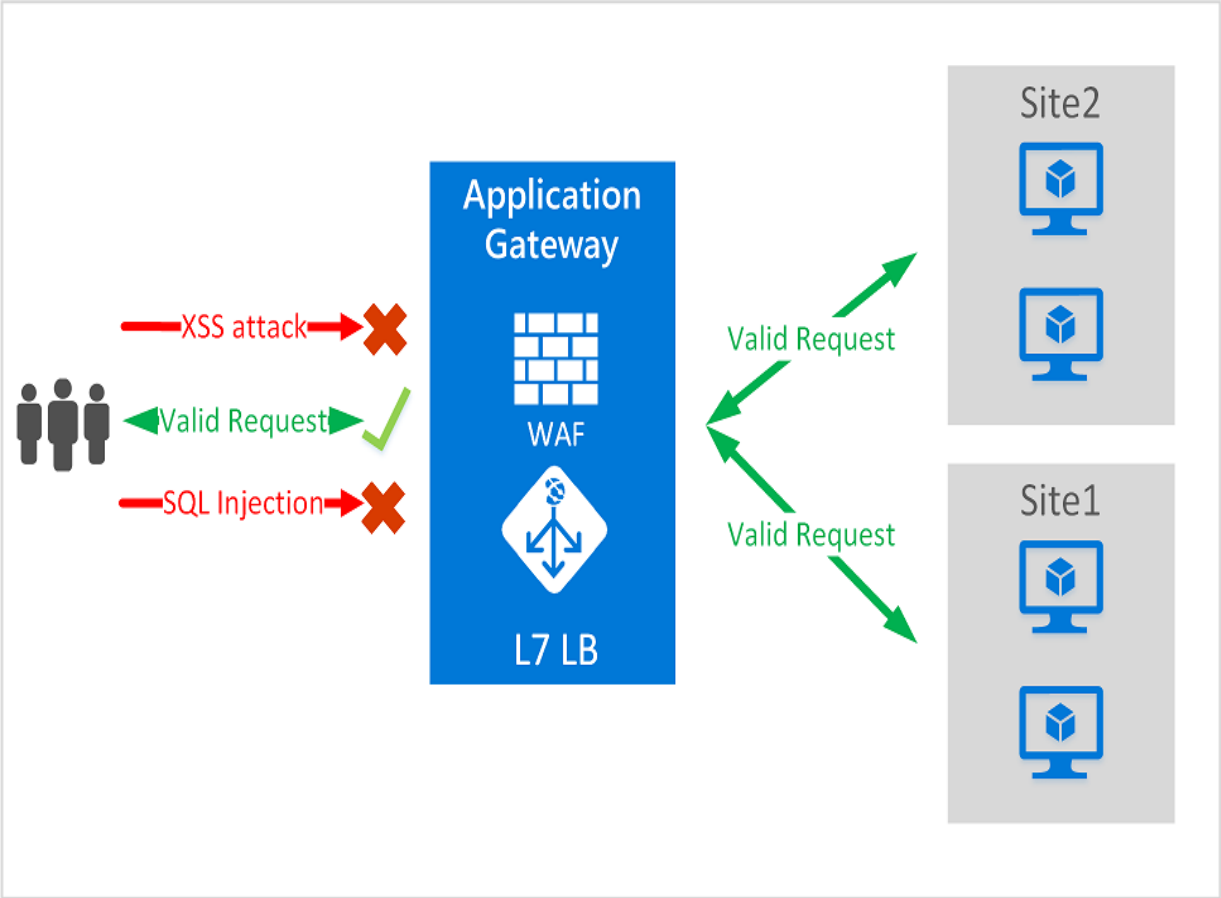

Figure 1.1 Web Application Firewall. A taxonomy of SQL injection detection and prevention techniques (p. 54), by Sadeghian, A., Zamani, M., & Manaf, A. A. (2013). 2013 International Conference on Informatics and Creative Multimedia.

Web Application Environment

Before we initiate any discussion on the approaches for detection and prevention of SQL injection attacks, let's first explore the Web application environment itself. In a Web Application environment, the web application information is presented to the Web server by the user's client, in the form of URLs, cookies and form inputs (POSTs and GETs). These inputs drive both the logic of the applications as well as the queries which

help the attacker to gain access to these applications for creating and sending a query to the database to extract relevant data.

Unfortunately, many applications do not frequently validate user input and so are more susceptible to SQL injection. Attackers capitalize on these flaws to attempt to hack the backend database to do something different than what the application or the search is intended for. This can include extracting sensitive information of employees, customers, destroying information or executing a DOS (Denial of Service) attack that limits the usage of the application.

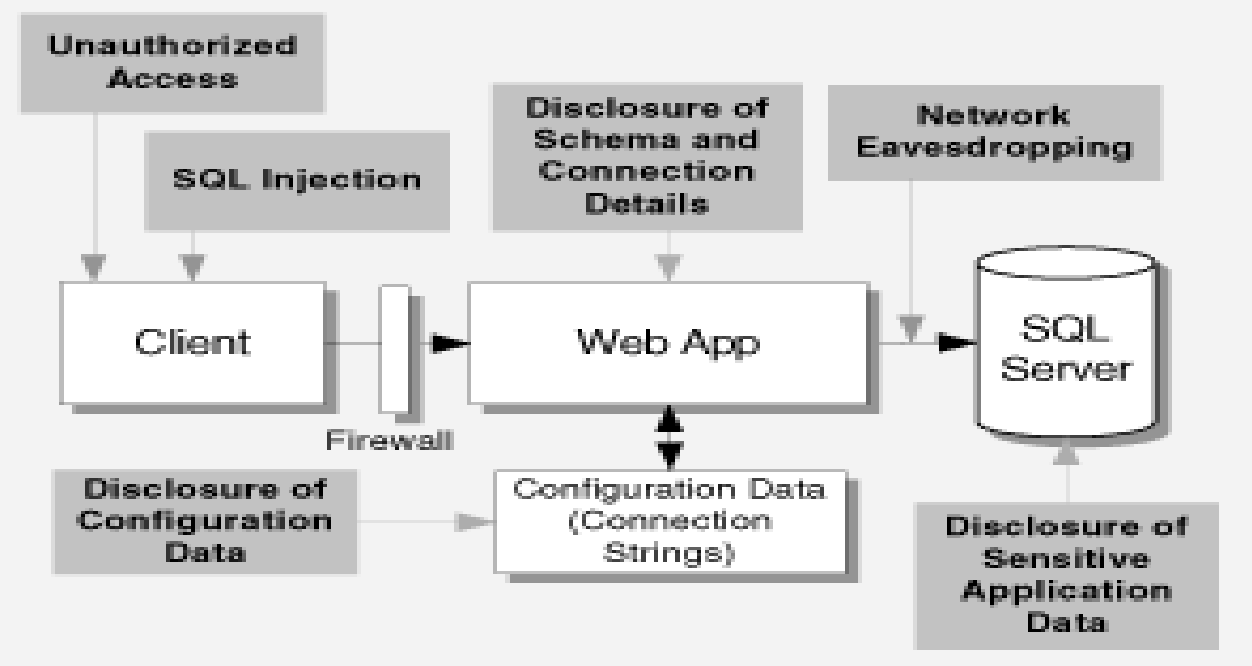

Figure 1.2 Web Tier Environment. SQL injection is still alive: A study on SQL injection signature evasion techniques (p.256), by Sadeghian, A., Zamani, M., & Ibrahim, S., 2013, International Conference on Informatics and Creative Multimedia.

Objective of the Research

- The main objective of this research is to provide multiple layers of security to protect databases from SQL Injection from a method which has highly durable storage and high-performance databases.
- To provide virtual clouds for organizations which are easy to access, have low maintenance and a capital prerequisite which can be taken care of even by small private companies and startup firms.

SQL Injection Attack Overview

SQL injection attacks are initiated by the manipulation of the data input on a Web form such that the traces of the SQL instructions are passed to the Web applications and these Web applications then combine with the rogue SQL fragments with the proper SQL dynamically generated by the application and create valid SQL requests. These new, unanticipated requests cause the database to perform the tasks intended by the attacker.

To have a clear understanding let us consider an example: If we have an application whose web page contains a simple form of the query with the input fields for username and password. With these credentials, the user can get a list of all the credit card accounts the various customers hold with a bank. Further, if the bank’s application was built without taking into consideration the potential of SQL injection attacks.

In this situation, it is reasonable to assume that the application merely takes an input the user types and places it directly into the SQL query constructed to retrieve that user's information. In PHP, the query string would be like this:

$query = “select accountName, accountNumber from creditCardAccounts where username='”.$_POST[“username”].”' and password='”.$_POST[“password”].”'”

Normally this would work properly as a user entered their credentials, say johnSmith and my Password, and forms the query:

$query = “select accountName, accountNumber from creditCardAccounts where username='johnSmith' and password='myPassword'

This query will come up with the total number of accounts Mr. John Smith is holding. Now consider someone with a fraudulent intent. If the person attempts viewing the account information of one or more of the bank’s customers, he enters the following credential into the form:

' or 1=1 -- and anyThingsAtAll

When this SQL fragment is inserted into the SQL query by the application it becomes:

$query = “select accountName, accountNumber from creditCardAccounts where username='' or 1=1 -- and password= anyThingsAtAll

The injection of the term, ' or 1=1 --, accomplishes two things. Firstly, it causes the first term to be true for all the rows of the query in the SQL statement; Secondly, it causes the rest of the statement to be treated as a comment and is ignored during runtime. Thus, as a result, the attacker has all the valuable information customers were seeking all the credit card information up to the limit the Web page will list.

It should be noted that this simple example is just one of an infinite number of variations that can be used to accomplish the same attack. Further, there are many other ways to exploit a vulnerable application.

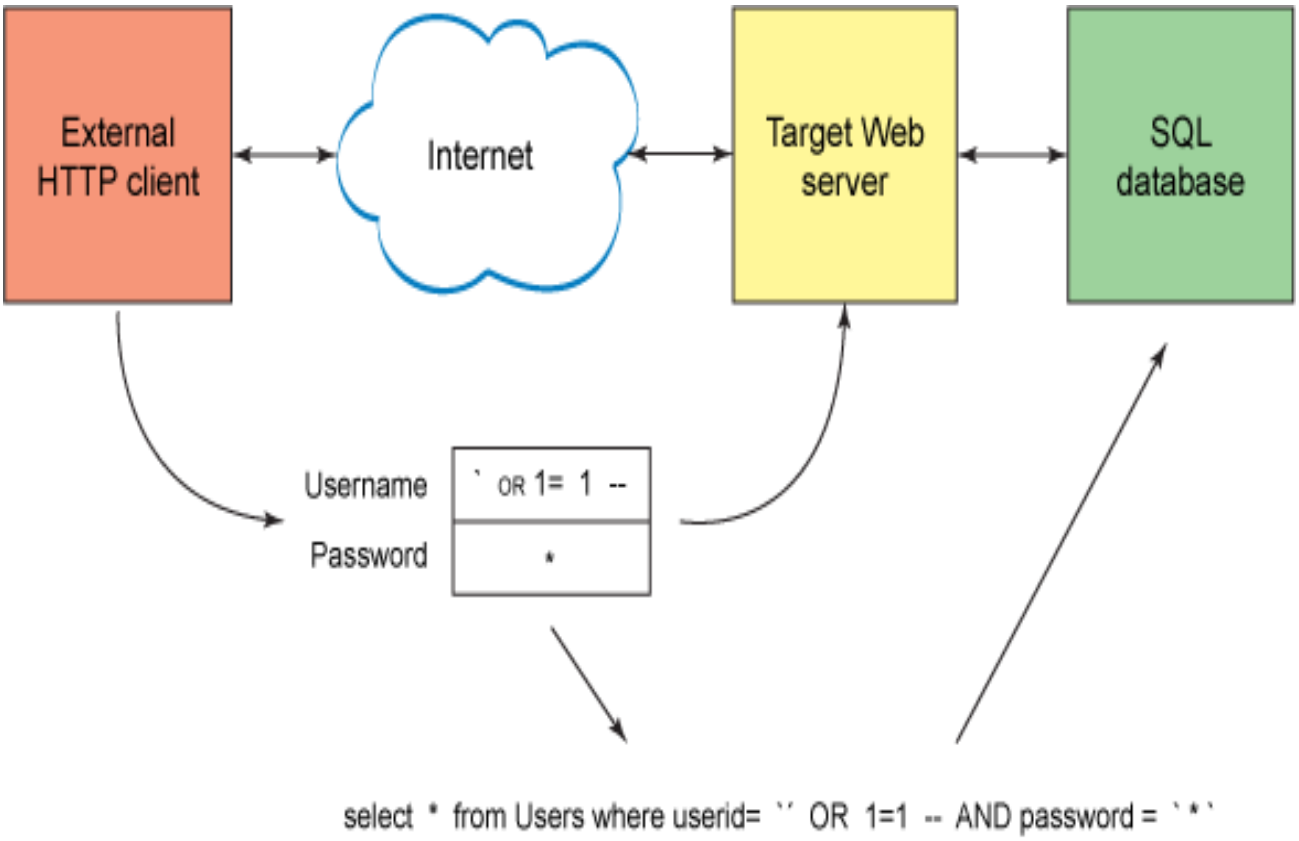

Figure 1.3 SQL Injection Attack Overview. Runtime monitors for tautology-based SQL injection attacks (p. 26), by Dharam, R.; Shiva, S.G., 2012, Cyber Security, International conference on Cyber Warfare and Digital Forensic (CyberSec).

Applications Vulnerable to SQL Injection

Due to several factors, writing these applications securely has become very rare. Many applications were written at the time when Web security was not a major threat. While due to the recent discussions on SQL injection at security conferences and other settings, an awareness was spread that the attack frequency of SQL injection only five or so years ago was so low that most developers were simply not aware.

In addition, the applications were exposed to the web with a lower security threshold and subsequently exposed to the web without even considering the security threats that it might have in the future because of SQL injections. Even applications

which are written and deployed today often inadequately addresses security concerns. IBM's X-Force project recently found that 47% of all vulnerabilities that result in unauthorized disclosures are Web application vulnerabilities by Kar, Panigrahi, Sundarajan (2016) For packaged applications from commercial software vendors Cross-Site Scripting & SQL injection vulnerabilities continue to dominate as the attack vector of choice. Vulnerabilities in custom applications were not reported. Since this software is generally not as carefully treated for security robustness, it is reasonable to assume that the problem is much greater because 97% of data breaches worldwide are still due to SQL injection somewhere along the line by Kar, Panigrahi, Sundarajan (2016).

Interestingly, modern environments and development approaches create a subtle vulnerability. By the advent of Web 2.0, there has been a massive shift in how developers treat user input. In these applications, the input transmits the information to the web server directly in a simpler form for processing. Most frequently the JavaScript portion of the application performs input validation so the feedback to the user is handled more smoothly. This often creates the sense that the application is protected because of this very specific input validation; resulting in the negligence of the server side on a large scale. Unfortunately, attackers will not inject their input into an application using another application rather they leverage intermediate applications to capture the client-side input and allow them to manipulate it.

Summary

The introduction gives a brief overview of the different types of SQL Injection attacks and how the web application firewalls are used to obstruct the unwanted queries in malfunctioning the codes of any web application. A brief overview of how a SQL injection attack is performed by an attacker is explained with the causes initiating the SQL injection attacks.

Chapter II. Background and Literature Review

Introduction

Over the past few years, SQL Injection attacks have been slipping seamlessly through the network firewalls over port 80 (HTTP) or 443 and are bypassing their web application firewalls (WAF) through obfuscation, thereby breaching many organizations. Moreover, the count of SQL injection attacks against organizations has increased over the years causing devastating effects on their databases and security. At that point, the attacker can exploit the soft internal network and vulnerable databases because SQL injection has become the most dangerous threat that is being tackled by many organizations.

Detection of SQL fragments injected into a Web application has proven extremely challenging. There are several preventions and security measures that enterprises can adopt. When implementing prevention and remediation efforts, the enterprise strives to develop secure code and/or encrypt confidential data stored in the database. However, these are not always available options. For example, in some cases, the application source code may have been developed by a third party and not be available for modification. Additionally, patching deployed code requires significant resources and time because of which rewriting an existing operational application would need to be prioritized ahead of projects driving new business. Similarly, efforts to encrypt the confidential data stored in the database can take even longer time and require more resources. Given today’s compressed development cycles, and a limited number of developers with security domain experience, even getting the code rewrite

project off the ground could prove difficult.

Literature Related to the Problem

A novel technique was proposed by Wei Ke, Muthuprasanna, and Kothari, (2006) to defend the SQL Injection attacks targeted at stored procedures. This technique was the combination of static application code analysis with runtime validation which can eliminate the occurrence of such attacks. The technique, in which a stored procedure parser was designed for any SQL statement which depends on user inputs to compare the original SQL statement structure to the user inputs was used.

An anomaly-based approach was described by Kiani, Clark & Mohay, (2008) which utilizes the character distribution of certain sections of HTTP requests to detect previously unseen SQL injection attacks. This approach does not require user interaction, and no modification of, or access to, either the backend database or the source code of the web application itself.

The hybrid approach based on the Adaptive Intelligent Intrusion 725 Detector Agent (AIIDA-SQL) proposed by Pinzon, Paz, Bajo & Herrero, (2010) was used for the detection of various SQL Injection attacks. “The AIIDA-SQL agent incorporates a Case-Based Reasoning (CBR) engine which is equipped with learning and adaptation capabilities for the classification of SQL queries and detection of malicious user requests” Pinzon et.al (2010). To carry out the tasks of attack classification and detection, the agent incorporates advanced algorithms in the reasoning cycle stages.

Basically, an innovative classification model based on a mixture of an Artificial Neuronal Network together with a Support Vector Machine is applied in the reuse stage

of the CBR cycle. This strategy enables to classify the received SQL queries in a reliable way. Finally, a projection neural technique is incorporated, which notably eases the revision stage carried out by human experts in the case of suspicious queries by Pinzon et.al (2010).

The Database driven web application is subsequently threatened by SQL Injection Attacks (SQLIAs) because this type of attack can compromise confidentiality and integrity of information in databases and to stop these type of attacks various approaches had been proposed but because of their respective limitations they are not enough to block these attacks Tajpour & Jor, (2010).

To test the tools in a realistic scenario, Vulnerability and Attack Injection is applied in a setup based on three web applications of different sizes and complexities designed by Elia, Fonseca & Vieira, (2010). Results show that the assessed tools have a very low efficiency and only perform well under specific circumstances, which highlight the limitations of current intrusion detection tools in detecting SQL Injection attacks.

Based on the class of injection flaw in which specially crafted input strings leads to illegal queries to databases, an effective solution TransSQL was developed by Zhang, Lin, Chen, Hwang, Huang & Hsu (2011). TransSQL automatically translates a SQL request to an LDAP-equivalent request. After queries are executed on a SQL database and an LDAP one, TransSQL checks the difference in responses between a SQL database and an LDAP one to detect and block SQL injection attacks.

A framework which can be used to handle tautology-based SQL Injection Attacks using a post-deployment monitoring technique was proposed by Dharam & Shiva,

(2012). Their framework uses two pre-deployment testing techniques i.e. basis path and data flow testing techniques to identify legal execution paths of the software. Runtime monitors are then developed and integrated to observe the behavior of the software for identified execution paths such that their violation will help to detect and prevent tautology-based SQL Injection Attacks.

Wu & Chan (2012), proposed a very effective method named k-centers (KC) to detect SQL injection attacks (SQLIAs). The number and the centers of the clusters in KC are adjusted according to unseen SQL statements in the practical environment, and in which the types of attacks are changed after a period to adapt to different kinds of attacks.

One of the most common solutions for defending against SQL Injection Attacks is the use of web application firewalls. Usually, these firewalls use signature-based techniques as the main core for the detection in which the firewall checks each packet against a long list of predefined SQL injection attacks known as signatures. “The problem with this technique is that an attacker with a good knowledge of SQL language can change the look of the SQL queries in a way that firewall cannot detect them but still they lead to the same malicious results” Sadeghian, Zamani & Abdullah (2013).

Literature Related to the Methodology

Amazon Web Services (AWS) provides a variety of infrastructure services, such as computing power, storage options, networking, and databases. These databases will be available in seconds and are delivered as a utility. “This allows enterprises, start-ups, small and medium-sized businesses, and customers in the public sector to access the

building blocks they need to respond quickly required to change the business requirements" Mathew, (2006). In 2006, Amazon Web Services (AWS) began offering IT infrastructure services to businesses in the form of web services—now commonly known as cloud computing Amazon Web Services, including the Web Application Firewall (AWS WAF) by Mathew (2006).

One of the key benefits of the AWS WAF is the opportunity to replace up-front capital infrastructure expenses with low variable costs that scale with the business. With the help of AWS WAF, businesses no longer need to plan for and procure servers and other IT infrastructure weeks or months in advance. Instead, they can instantly spin up hundreds or thousands of servers in minutes and deliver results faster.

Types of SQL Injection Attacks

There are different types of attacks depending upon the goal of an attacker which are performed together or sequentially.

Tautologies

The tautology-based attack is basically injecting the code into one or more conditional statements, so the statements always evaluate to true. The results of this attack depend on how the queries are used within the application proposed by Anley, (2002). The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE condition statement.

According to McDonald (2002), the database table gets targeted by the returned query by transforming the conditional query. For a tautology-based attack to work, an

attacker must not only consider injecting the vulnerable parameters, but also the coding which evaluates the query results. An attack is successful when the code either displays all the returned records or performs some action if at least one record is returned. Example: “In this example attack, an attacker submits “’ or 1=1 - - ” for the login input field (the input submitted for the other fields is irrelevant).

The resulting query is:

SELECT accounts FROM users WHERE login=’’ or 1=1 -- AND pass=’’ AND pin=

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology” by Halfond, Vieagas & Orso (2006).

The above condition is used as the base for evaluating each row and deciding which ones should return to the application. Because the above condition is a tautology, the query validates to be true for each row in the table and returns all the values related to the query. In the above example, the returned set evaluates to a nonnull value, which causes the application to conclude that the user authentication was successful by Howard, LeBlanc, (2003). Therefore, all the application would invoke method displayAccounts() and show all of the accounts in the set returned by the database.

Illegal or Logically Incorrect Queries

The attacker gathers important information about the type and structure of the organization’s back-end database of a Web application. This attack is considered a preliminary, information gathering step for other attacks. Because of the vulnerability caused by this attack, the default error page is returned by the application servers and often are very detailed. In fact, according to Anley, (2002), injectable parameters can be

generated by an attacker from the simple error messages that are displayed using any web application. While, additional error information, is used for debugging the applications by the programmers, will adversely help the attackers to gain information about the functioning queries of the back-end database.

As proposed by Litchfield (2002), while performing this attack, the statements that cause a syntax, type conversion, or logical error are manipulated by the attackers into the database. Injectable parameters can be identified by syntax errors. Type errors are used to deduce the data types of certain columns or to extract data. Logical errors can reveal the names of the tables and columns from the database that causes an error.

Example: In this example, the attacker's goal is to cause a type conversion error that can reveal relevant data. To do this, the attacker injects the following text into input field pin: "convert (int,(select top 1 name from sysobjects where xtype='u'))".

The resulting query is:

SELECT accounts FROM users WHERE login='' AND pass='' AND pin= convert (int,(select top 1 name from sysobjects where xtype='u'))

In the above example, the select query injected into the attack string attempts to extract the first user table (xtype='u') from the database's metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert the specified table name into an integer and as this type of conversion is not legal in Microsoft SQL Server, the database throws an

error stating Microsoft OLE DB Provider (0x80040E07) Error converting varchar value ’CreditCards’ to a column of data type int. Halfond, Vieagas & Orso (2006).

There are two useful pieces of information which help an attacker according to Halfond, Vieagas & Orso (2006).

- “First the attacker can see that the database is a SQL Server database, as the error message explicitly states this fact.
- Second, the error message reveals the value of the string that caused the type of conversion to occur.” Halfond, Vieagas & Orso (2006).

In the above scenario, the table that is been attacked first is a user-defined table in the database called “CreditCards”. Each column in the database can be extracted by using the similar strategy. More threats can be created to the database by an attacker using the same information about the schema of the database, which targets specific pieces of information in the database.

Union Query

For a given Query the attacker exploits a vulnerable parameter and changes the dataset returned in this type of attack. According to Anley, (2002) the application can be tricked into returning data from a different table that was not intended by the developer to be returned for the respective query. The most commonly injected statement used by the attackers is of the form: UNION SELECT. The information of the table can be retrieved by the attackers as they have complete control over the second/injected query which aids in accessing the permission rights to the database. Because of this attack,

the final database will be a combination of the original query which was created by the developer and the modified second query injected by an attacker.

Example: Referring to the running example, an attacker could inject the text "' UNION SELECT cardNo from CreditCards where acctNo=10032 - -" into the login field, which produces the following query:

SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND pass='' AND pin=

The first query (Original) will return with a result of null set considering there is no login equal to "", whereas the second query (Injected) returns data from the "CreditCards" table. The column "cardNo" for account "10032" will be returned by the database. Because of these two queries, the database will return the union of them to the application. Because of the union of these two queries, the cardNo would show up with the account information in the application by Halfond, Vieagas & Orso (2006).

Piggy-Backed Queries

Original query is injected with the additional queries in this attack type. This is distinguished typed from others as here the attacker modifies the original query instead of a new one. This includes the "piggy-back" queries on the original query. Numerous SQL queries are returned from the database because of this query. First the intended query is executed then the subsequent queries that are entered are the injected ones, and they are in addition to the previous one. According to Anley, (2002) this type of attacks are very vulnerable and if it is successful, any SQL command can be injected by the attackers virtually. The original query is injected and executed along with the stored

procedure as an example into the additional queries. This type of attacks usually happens to a database where the configuration allows multiple statements to be contained in a single string, they are very vulnerable to the structured database. McDonald, (2002).

Example: If the attacker inputs “’; drop table users - -” into the pass field, the application generates the query:

SELECT accounts FROM users WHERE login=’doe’ AND pass=’’; drop table users -- ’ AND pin=123 by Halfond, Vieagas & Orso (2006).

After the completion of the first query, the database will inject the second query after recognizing the query parameter (“;”) and the injected second query will be executed. Valuable information will be destroyed from the database if the injected second query is executed and the tables are dropped. Other types of queries could insert new users into the database or execute stored procedures Howard & LeBlanc, (2003). Simply scanning for a query separator will not be a good idea to detect the injected queries as the databases do not require special characters to separate and identify distinct queries.

Stored Procedures

According to Halfond, Vieagas & Orso (2006), stored procedures are routines stored in the database and run by the database engine. These procedures can either be user-defined procedures or procedures provided by the database by default. SQLIAs of this type try to execute stored procedures present in the database. The database interaction with the operating system is limited now a day to an extent with the help of

stored procedures, as they set a standard functionality and most of the vendors provide that set by default while delivering the database. The SQLIAs can be used to execute the stored procedures in that database, once the attacker knows which type of database is used in the backend. Stored procedures also interact with the operating system.

Using the stored procedures while coding the Web applications renders them invulnerable to SQLIAs. The stored procedures are not much dependent from the developer side as these procedures are most vulnerable to the attacks on the applications Howard & LeBlanc, (2003). The attackers get the access to run the arbitrary codes on the server or to escalate the privileges as the stored procedures are often written in special scripting languages and additionally they can contain other types of vulnerabilities, such as buffer overflows. Labs, (2002).

CREATE PROCEDURE DBO.isAuthenticated @userName varchar2, @pass varchar2, @pin int AS EXEC("SELECT accounts FROM users WHERE login='" +@userName+ "' and pass='" +@password+ "' and pin=" +@pin); GO

Example: The SQLIA can be used to exploit the parameterized stored procedure in the above example. In the example, a stored procedure has been placed as an alternative for the constructed query string. To rightly authenticate the user credentials, the stored procedure returns a true/false statement. The attacker simply injects " ' ; SHUTDOWN; --" into either the userName or password fields to inject the SQLIA attack. Due to this the injection the following query is generated through the stored procedure:

SELECT accounts FROM users WHERE login='doe' AND pass=' '; SHUTDOWN; --

AND pin= Halfond, Vieagas & Orso (2006)

This attack is called a “piggy-bank” type attack. The injected or the malicious query is injected second into the database after the execution of the first normal query, due to which the database shuts down. In the above example, it illustrates that the stored procedures are as vulnerable to the same range of attacks as the traditional application code.

Inference

In this attack, the query is modified in such a way that any action executed will depend on the true or false answer values for the data which is altered in the database. In this type of injection, attackers generally attack a site that has enough security so that, whenever there is a successful injection, there should not be any usable feedback through database error messages. As the database error messages are unavailable or not sufficient for the attacker as no feedback is provided an alternate method should be used by the attackers for obtaining a response from the database by Anley, (2002).

According to Spett, (2003) by using an alternate method malicious commands will be injected by the attacker into the website and is studied for any functional changes on the website. After completely studying the effects caused by the injected commands like what changes the commands are making to the website interface and functioning the attacker can deduce the accurate commands to see what parameters are vulnerable to the change in the behavior of the. Most commonly there are two important attack techniques based on an inference which allows an attacker to extract data from a database and detect vulnerable parameters.

Blind Injection

According to Halfond, Vieagas & Orso (2006) the developers hide the error details during programming which ends up showing a generic page instead of an error message because of which the attacker gets the information of the tables related to the database structure by asking the true/false type of questions through SQL statements.

SELECT accounts FROM users WHERE login= 'doe' and 1 =0 -- AND pass = AND pin=O

SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=O

If there is no input validation the query will execute.

Timing Attacks

According to Halfond, Vieagas & Orso (2006), this attack particularly depends on the time lapses or delays. This time delays aid an attacker in gaining information of the database. The timing attack is pretty much like the blind injection except it uses a different inference method. For performing a timing attack, if/then statements are used as an injected query by the attacker which relates to the content of the database. The WAITFOR keyword which is used to delay the time response for a specified time uses the SQL Queries to construct the amount of time to execute each branch among all the other branches. A specific branch is picked by the attacker which either increases or decreases in response to the time of the database which gives the solution of the injected question to the attacker.

Example: A specific code is used in two different ways in which the attacks are explained by using the inference-based techniques. The parameters are identified using the blind injection technique in the first form while filling up two possible injections in the login field.

The first being “legalUser’ and 1=0 - -” and the second, “legalUser’ and 1=1 - -”. These injections result in the following two queries:

SELECT accounts FROM users WHERE login=’legalUser’ and 1=0 -- ’ AND pass=’’ AND pin=0

SELECT accounts FROM users WHERE login=’legalUser’ and 1=1 -- ’ AND pass=’’ AND pin=0

Considering two scenarios in which assuming the first scenario as a secure application which has a validated login input. As the SQL queries injected by the attacker will return with login error messages because of the incorrect login parameters making the query not vulnerable. In the second scenario, there will be two attempts by the attacker for the injection one with always a true statement and one with always false statement as we have an insecure application and the login parameter is vulnerable to injection. The first statement which will be false is injected by the attacker and as an expected result the application will return with a login error message.

There might be two reasons for an error message during login, one being the attack attempt validated correctly by the application and second, the injected attack itself caused the login error. Now the second statement which is always true is injected

by the attacker and there won't be any login error message which concludes to the attacker that the login parameter is vulnerable to the injection.

Data extraction can be carried out using the inference-based techniques by injecting a timing-based inference attack and extracting the table name from the database. In this attack, the following query is injected into the login parameter: ''legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 --''.

This produces the following query:

SELECT accounts FROM users WHERE login='legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 -- ' AND pass='' AND pin=0

In this attack, the attacker asks a series of questions about the first character of the first table's name (SUBSTRING) using a binary search strategy and if the value of X is greater-than or less-than-or-equal-to the value of ASCII value there is an additional 5 second delay in the response of the database, by which the attacker knows that the value injected is greater and then the value of the first character. Therefore, the value of X is adjusted by the attacker accordingly.

Alternate Encodings

This attack is used in combination with other attacks by injecting a modified query altered by defensive coding practices to avoid detection of the automated prevention techniques. In other words, as explained by Anley, (2002) alternate encodings are used as an aid by the attacker for evading the detection and prevention

techniques which might be exploitable and can carry vulnerabilities in the application. These evasion techniques are useful in scanning certain “bad characters,” such as single quotes and comment operators commonly used in the coding practices.

The common techniques are not enough capable of determining and scanning the specially encoded strings which use hexadecimal, ASCII, and Unicode characters which allows the SQL injection attacks go undetected. The alternate Encoding technique provides different layers in an application to evaluate all the specially encoded strings by scanning for certain escape characters that represent alternate encodings in its language domain and may even use different methods of encoding by Howard & LeBlanc, (2003).

A perfect code-based defense is practically very much difficult to build and implement in work environment as it requires the developers to consider all the possible scenarios which could affect a query string in different layers of an application through SQL injection. For example, “a database could use the expression char(120) to represent an alternately-encoded character “x”, but char(120) has no special meaning in the application language’s context Halfond, Vieagas & Orso (2006).” Therefore, the attackers are very much successful in injecting a coded query in the application code string.

Example: An alternately encoded attack is provided in the example in which the following text is injected into the login field: “legalUser’; exec(0x73687574646f776e) -- ”. The resulting query generated by the application is:

SELECT accounts FROM users WHERE login='legalUser';

exec(char(0x73687574646f776e)) -- AND pass='' AND pin=

In this example char() function is used with the ASCII hexadecimal encoding. The char() function returns the instance of that character and is considered as a parameter an integer or hexadecimal encoding of the particular character. The second line in the example is the ASCII hexadecimal encoding of the string "SHUTDOWN." Therefore, a SHUTDOWN command is executed whenever a code or string is interpreted by the database.

Main Causes of SQL Injection

In this section, various causes of SQL injection are presented:

Invalidated input. Any SQL query consists of some parameters such as INSERT, UPDATE, ALTER and some SQL control characters such as a semicolon and quotation mark. If there is no checking for these, web applications can potentially be abused in a SQL injection attack.

Generous privileges. Privileges are some rules for accessing some database for an object. SELECT, INSERT, and DELETE are actions of executing SQL queries that include typical privileges. Typically, a web application is used for accessing any specific information from the database.

Uncontrollable variable size: If any variable is used for the storage of a large amount of data there might be a chance of SQL injection of faked input values from the attacker.

Error message. An error message is generated when the wrong input values are inserted in web applications. Attackers may get the script structure or information about the database so that the attacker may create its own attack.

Client-side only controls. If input validation is implemented in client side-scripts only, then by using cross-site scripting security functions of a script at the client side it can be overridden, and an attacker can invalidate input for accessing the database.

Stored procedure. Stored Procedures are a small program with some functions which are called multiple times in execution. When these functions become calls so that stored procedures become calls in place of that function. These stored procedures become stored in the database. The problem with stored procedures is that an attacker can execute and damage the database.

Into out file support. A text file containing SQL query results may be gotten by manipulating a SQL query. This can be possible by using the condition of INTO OUTFILE clause that is beneficial for some relational databases.

Sub-select. When a SQL query is inserted in the WHERE clause of another SQL query this shows one of the weaknesses for a database. This weakness also makes the web application more vulnerable.

The challenge with detection. The goal of any security technology is to provide a robust threat detection for the database which is very easy to setup or which doesn't require any setup or configuration. Further, if that technology relies on learning or training to improve its ability to detect threats, those learning periods must be short and well-defined. The longer the time period, for learning the higher are the chances that

attacks may occur so there is a need to expedite the installation and minimize the risk of attacks.

Detection and Prevention Techniques

Researchers have proposed a range of techniques to assist developers and compensate for the shortcomings in the application of defensive coding.

Black box testing. A black-box technique called WAVES, was designed by Huang, Lin & Tsai (2003) for testing Web applications for SQL injection vulnerabilities. The technique uses a Web crawler to identify all points in a Web application that can be used to inject SQLIAs. It then builds attacks that target such points based on a specified list of patterns and attack techniques. The time response of the attacks over the application improves in the WAVES technique as it uses the machine learning approaches to guide the testing. This technique is safer compared to the other testing's but still cannot guarantee concerning complete security.

Static code checkers. JDBC-Checker technique is also known as Static code checker technique which is used to prevent the type of SQL injection attacks that occur due to the mismatch of the practically generated query string proposed by Gould, Su & Devanbu, (2004). This technique detects SQLIA code vulnerabilities, typo's in the code input. As this technique was not developed for detection and prevention of the SQL injection attacks is still used for the same purpose of finding the root vulnerabilities in the dynamically generated query string. Even after the combination of the static analysis with the automated reasoning, it was unable to detect different types of SQL injection attacks other than Tautologies.

Combined static and dynamic analysis. AMNESIA is a model-based technique designed by Halfond & Osro, (2005) that combines static analysis and runtime monitoring. There are two phases in this type of analysis static phase and dynamic phase. Static analysis is used to generate legal queries for an application at each point of access to the database by building models of different types of queries through a process called AMNESIA. Whereas Dynamic analysis validates all the unwanted queries before they are sent to the database for the statically built models through the same process. Queries which does not pass through the validation of AMNESIA are considered as SQLIAs which will be terminated from executing into the database. The primary limitation of this technique is the accuracy of the static analysis which is used for building the query models.

There are two more approaches related to the combined static and dynamic analysis. In the first approach runtime for the queries is verified to confirm the model for the expected queries should pass only the accepted queries. Whereas the SQLGuard model deduces the runtime by adding an additional user input known as SQLCheck -by the developer. Both the approaches share a secret key which is used to insert user input during parsing by the runtime checker. The developer must rewrite the use of special characters or markers in the code to develop a dynamically generated query so as the to avoid the attackers in finding out the secret key proposed by SQLGuard by Buehrer, Weide & Sivilotti, (2005) and SQLCheck by Wasserman & Su, (2004).

Taint-based approaches. The Taint Based approach uses a method called WebSSARI which is used to check the taint flows for sensitive functions which detect

the precondition points in which the filters and sanitization functions can automatically be added to satisfy the precondition parameters. It uses the predefined set of filters to sanitize the input. The primary drawback of this technique is that the sensitive functions in an injected code can be accurately expressed using the typing system through a certain type of filters which are not tainted stated by Huang, Yu, Hang, Lee & Kuo, (2004).

Livshits and Lam, (2005) proposed that using information flow techniques for detecting the tainted input using static analysis vulnerabilities in software can also be detected. A SQL query can be constructed with this technique to avoid the flagged as SQLIA vulnerabilities. Another approach made by Pietraszek and Berghe, (2005) used a context-sensitive analysis which used a PHP interpreter to track precise per-character taint information. The SQL injections would be validated depending on the false positive statements which intercept any untrusted query or code injected by an attacker. Only known patterns of SQLIAs can be detected by these two approaches which cause the common drawback for both the methods as they require modifications to the runtime environment, which affects portability.

Another technique is by using SecuriFly which validates the query strings generated by the tainted inputs, unlike the above two approaches which use a context-sensitive analysis and track the taint information depending on the per-string basis stated by Haldar, Chandra & Franz, (2005) and Martin, Livshits & Lam, (2005). But as there is no taint-based approach related to this method it does not give enough sanitization to regulate the injection in the numeric fields of the code. The main

drawback of this technique is identifying all the sources of tainted user input in web applications and accurately validating them.

New query development paradigms. A combination of two approaches, SQL DOM by McClure & Krugre, (2005) and Safe Query Objects proposed by Cook & Rai, (2005) offers an effective technique by changing the query building process using encapsulation of database queries in combination with the API string concatenation. This approach provides a safe and reliable way to access the databases and avoids the unwanted SQL injections. This technique needs a new development environment as it is a combination of the latest and the legacy approaches which creates a paradigm in which the SQL queries are developed. As it is a new environment the only drawback is the developers must learn a new programming language and there won't be any protection for the existing legacy systems.

Intrusion detection systems. IDS system builds models based on a machine learning technique which consists of typical queries and monitors the runtime of the application in real time that is being trained using a set of typical application queries. As the training set is required to monitor the application, a poor training set will generate many false positives and false negatives which is the only limitation of IDS stated by Valeur, Mutz and, Vigna, (2005).

Proxy filters. These filters have security gateways which provide the developer with a Security Policy Descriptor Language (SPDL), which has specified constraints and helps in filtering the unwanted injected codes coming from untrusted proxies to the web application. SPDL provides defensive programming which requires the developers to

know which data needs to be filtered and which proxies should be blocked and considered as untrusted and what patterns and filters should be applied to the existing database to suspend unwanted SQL injection attacks Scott & Sharp, (2002).

Instruction set randomization. SQLrand is based on the framework which helps the developers in creating the queries based on instruction-set randomization. Instruction-set randomization uses a proxy filter which intercepts the normal SQL keywords and pushes the randomized queries to the database. As the code injected by the attacker might not be constructed using the randomized instruction set the injected SQL query will fail in attacking the application. Like other techniques, SQLrand has a drawback that the code uses a secret key to modify the instructions which result in integration of a proxy with the tables present in the database of a system

Injection Detection at the Web Tier

There is a large variation in the pattern of SQL attacks, which makes it even more challenging for the detection of the initial point from where the attack is initiating in the Web server. Furthermore, the SQL requests sent to the database has special characters which may not be expected in a typical form sent by the attacker. There are URL’s, cookies, and form inputs (POSTs and GETs) to inspect and retrieve and inspecting each set of input values, makes it more difficult for a WAF. The SQL injection attacks are caused by coding the application using simple coding techniques and words such as “like” and “or” to catch every possible attack which practically is not possible.

Alternatively, as mentioned earlier, much more complex patterns that are clearly indicative of an attack can be used. Unfortunately, as discussed, the different types of

SQL injection attack the number and variation of possible attacks are so large that it is impossible to effectively cover all possible attack patterns. Creating the initial pattern set, being updated about the evolving attacks, and verifying that they are sufficiently unique so as not to show up in some fields is an almost impossible task. And now, considering that the applications are also changing and evolving over time, it requires more time so as more learning and hands-on skills for proper security of the databases without any breaches.

The Database Firewall is much more secure and effective than the previously used Web Application Firewall as it follows the structured analysis to build the SQL statements instead of the rudimentary input pattern validation used in WAF. It is more effective and secure because it monitors the networks between the application servers and databases with a much smaller set of SQL build statements. This database firewall is not that easy to build and maintain so we opt for different services such as Oracle but the latest most efficient and economical service to store and to secure the integrity of the data is provided through Amazon Web Services AWS.

Summary

The Background and Literature review helps in completely understanding about the SQL Injection attacks. Different types of SQL injection attacks are explained with the main causes and some of the detection and prevention techniques. The most efficient method of detecting and preventing the web applications from the SQL Injection attacks (Injection Detection at Web-Tier) is also explained.

Chapter III. Methodology

Introduction

Amazon Web Services Web Application Firewall (AWS WAF) helps to protect web applications from common web exploits like SQL injection attacks that could affect application availability, compromise security, or consume excessive resources. AWS WAF gives control over the traffic which allows or blocks the web applications by defining customizable web security rules. To create custom rules that block common attack patterns, such as SQL injection or cross-site scripting and to respond quickly for the change of patterns in the traffic, new rules can be deployed within minutes through AWS WAF. Also, AWS WAF includes a full-featured API that can be used to automate the creation, deployment, and maintenance of web security rules.

The strategy of configuring a web application firewall can be challenging and burdensome to large and small organizations alike, especially for those who do not have dedicated security teams. To simplify this process, AWS offers a solution that uses AWS Cloud Formation to automatically deploy a set of AWS WAF rules designed to filter common web-based SQL injection attacks. With AWS WAF we pay only for what we use. AWS WAF pricing is based on how many rules are being deployed and how many web requests the web application receives. These rules can be deployed by AWS WAF on either Amazon Cloud Front as part of the CDN solution or the Application Load Balancer (ALB) that fronts the web servers or origin servers running on EC2.

Design of the Study

Thus, far we have discussed different types of SQL injection attacks, the main causes of SQL injection and the method of detecting SQL injection attacks at the Web tier interface by a simple WAF system. A more effective and efficient method proposed in this paper to defend against SQL injection attacks is by using AWS WAF. This web application firewall allows us to monitor the HTTP and HTTPS requests which are forwarded to Amazon Cloud Front or an Application Load Balancer and allows us to control and access the content.

Based on conditions specified by the user, such as the IP addresses that the requests originate from or by the query string values, the Cloud Front or an Application Load Balancer responds to requests either with the requested content or with an HTTP 403 status code (Forbidden). The Cloud Front or an Application load balancer can also be configured in such a way that it returns with a custom error page when a request is blocked to analyze the actual SQL generated by the application as presented to the database by a firewall.

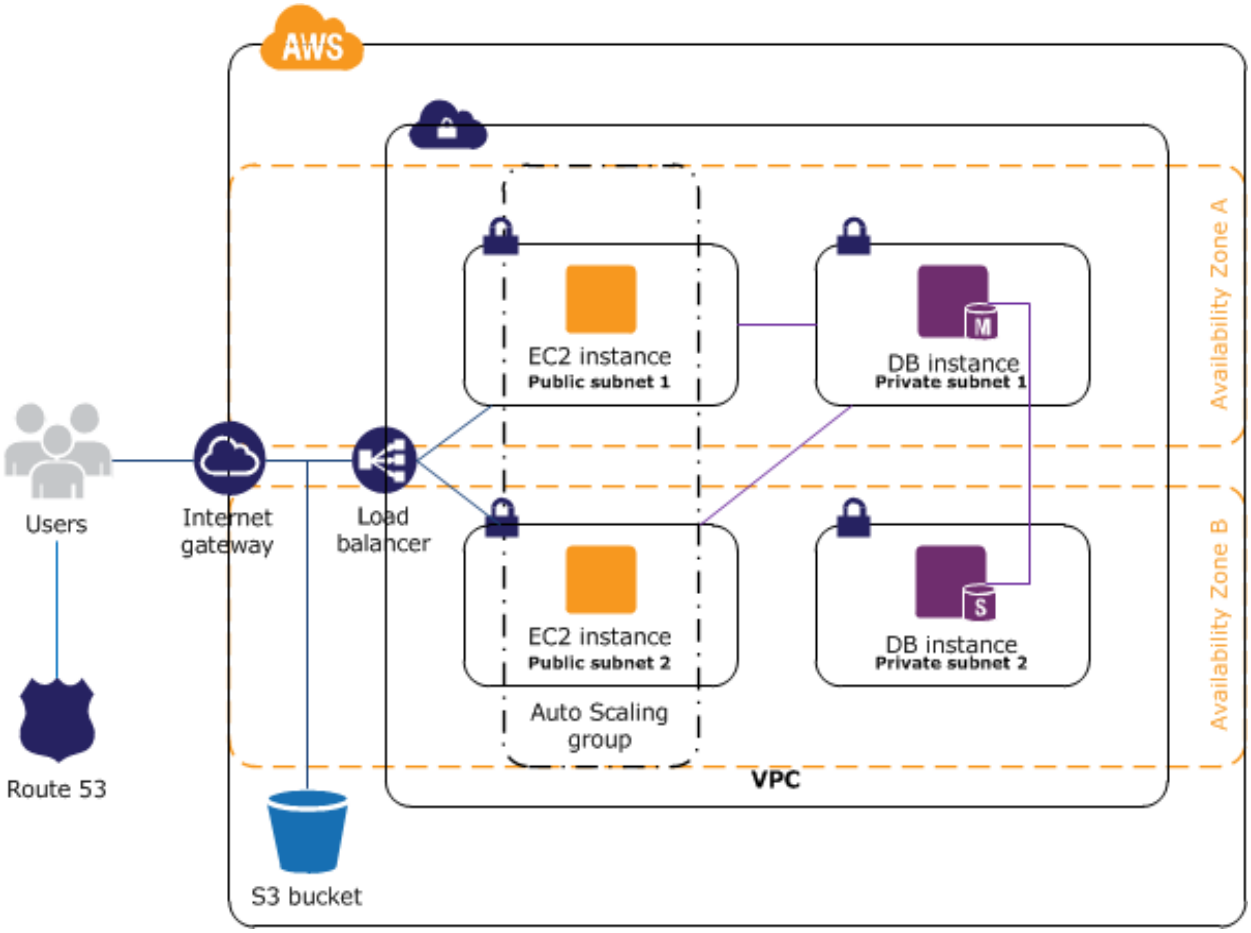

Figure 3.1 AWS WAF Architecture. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

The AWS WAF allows us to choose only the requests specified and block all the other unwanted requests such as SQL injections. It gives several other potential benefits such as providing rules which can be reused for multiple web applications, automated administration using AWS WAF API, real-time metrics and sampled web requests. A qualitative approach will be best suited for the proposed plan as it does not require any numerical data analysis. The following are the steps used for building an AWS WAF which will be discussed in detail in the next part of the paper.

- Step 1: Set Up for AWS WAF
- Step 2: Start the Wizard
- Step 3: Create an IP Match Condition
- Step 4: Create a String Match Condition
- Step 5: Create a SQL Injection Match Condition
- Step 6: Create Additional Conditions
- Step 7: Create a Rule and Add Conditions
- Step 8: Add the Rule to a Web ACL
- Step 9: Clean Up Your Resources

Data Analysis

Hardware and software requirements.

- Four virtual processors assigned to the VM.
- 12 GB of RAM assigned to the VM
- 80 GB of disk space for installation of VM image and system data
- General purpose instance family—m3 and m4 instance types
- Storage-optimized instance family—i2 and d2 instance types
- Compute-optimized instance family—c3 and c4 instance types
- Memory-optimized instance family—r3 instance types

Summary

An effective and efficient approach towards the protection of web applications is explained in the methodology. The technique AWS provides better security, no infrastructure, less capital, on-demand upgrade of processing speed, storage services and many computing clouds and subnets. The design and steps of building an Amazon Web Services Web Application Firewall are explained briefly.

Chapter IV: Analysis of Results

Introduction

Amazon Web Services was born out of the idea to provide multiple layers of security to avoid SQL injection attacks and to transfer the data from small scale to large scale. Amazon web services are available at any capacity on a moment's notice and without necessarily forecasting demand. Amazon meets this expectation in both of its key AWS products. Amazon's Elastic Cloud Computing (EC2) platform allows applications to run on an instantly scalable number of processors on demand, while Amazon's Simple Storage System (S3) allows access to a practically infinite allocation of disk space on demand. The Amazon EC2 platform allows applications to use as much processing power as they need at any given time, scaling up and down parallel to the demand. Similarly, S3 allows applications to scale storage needs exactly in parallel with demand.

Amazon began AWS by charging directly in proportion to usage (Amazon EC2 charges anywhere from $0.10 to $0.80 per processor hour while S3 charges up to $0.14 per GB per month of storage, with bandwidth costs of $0.10 to $0.15 per GB of bandwidth downloaded or uploaded. This inexpensive, pay-as-you-go price scheme eliminates the risk associated with investing in technologies never tested, encouraging system administrators and curious programmers to play with the service at extremely low costs.

Data Presentation

Amazon web services cloud platform. AWS consists of many cloud services and to access these services the AWS Management Console, and the AWS Command Line Interface is used.

AWS management console. Access and manage Amazon Web Services through the AWS Management Console, a simple and intuitive user interface.

AWS Command Line Interface

The AWS Command Line Interface (CLI) is a unified tool to manage the AWS services. With just one tool to download and configure, multiple AWS services can be controlled from the command line and automate them through scripts.

Compute

Amazon EC2. Amazon Elastic Compute Cloud (Amazon EC2) is a web service which provides secure, resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers and to reduce the time required to obtain and boot new server instances (called Amazon EC2 instances) to minutes, allowing to quickly scale capacity, both up and down, as the computing requirements change time to time.

Benefits.

***Elastic web-scale computing*.** Amazon EC2 enables to increase or decrease the capacity within minutes. Hundreds of thousands of server's instances can be controlled simultaneously. Because the instances are controlled by web service APIs, the application can automatically scale itself up and down depending on its needs.

***Completely controlled*.** There is a complete control of the Amazon EC2 instances having root access to each instance. While retaining the data on the boot partition, the Amazon EC2 instances can be stopped and then can be restarted subsequently using web service APIs. Instances can be rebooted remotely using web service APIs.

***Flexible cloud hosting services*.** There are multiple options for the instance types, operating systems, and software packages to choose from. Amazon EC2 allows the users to select the memory configuration, CPU, instance storage, and boot partition size.

Integrated

Amazon EC2 is integrated with most AWS services, such as Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), and Amazon Virtual Private Cloud (Amazon VPC) to provide a complete, secure solution for computing, query processing, and cloud storage across a wide range of applications.

***Reliable*.** Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned.

***Secure*.** Amazon EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality. The compute instances are in a VPC with an IP address range specified by the user which are exposed to the internet either to remain private or public. Security groups and network access control lists (ACLs) allows the user to control inbound and outbound network access to and from the instances. The

users can connect their existing IT infrastructure to resources in the VPC using industry-standard encrypted IPsec virtual private network (VPN) connections.

***Inexpensive*.** Amazon EC2 instances can be used at a very low rate for the compute capacity consumed by the users.

On-Demand Instances

With On-Demand instances, the users pay for computing capacity by the hour with no long-term commitments. The users can increase or decrease the compute capacity depending on the demands of the application and only pay the specified hourly rate for the instances used. The use of On-Demand instances frees the users from the costs and complexities of planning, purchasing, and maintaining hardware and transforms the large fixed costs into much smaller variable costs.

Storage

Amazon S3. Amazon Simple Storage Service (Amazon S3) is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It is designed to deliver 99.999999999% durability and scales past trillions of objects worldwide. It's simple to move large volumes of data into or out of Amazon S3 with Amazon's cloud data migration options. Once data is stored in Amazon S3, it can be automatically tiered into lower cost, longer-term cloud storage classes like Amazon S3 Standard - Infrequent Access and Amazon Glacier for archiving.

Amazon S3 features. Amazon S3 provides the most feature-rich object storage platform available in the cloud today, the following are a list of the Amazon S3 features:

Simple. Amazon S3 is simple to use with a web-based management console and mobile app. Amazon S3 also provides full REST APIs and SDKs for easy integration with third-party technologies.

Durable. Amazon S3 provides durable infrastructure to store important data and is designed for durability of 99.999999999% of objects. The data is stored in multiple facilities and multiple devices in each facility.

Scalable. With Amazon S3, the users can store as much data as they want and access it when needed. The future storage needs can be scaled up and down as required, dramatically increasing business agility.

Secure. Amazon S3 supports data transfer over SSL and automatic encryption of the data once it is uploaded. Bucket policies can also be configured to manage object permissions and to control access the data using Identity Access Management (IAM).

Low Cost. Amazon S3 allows the user to store large amounts of data at a very low cost. Using lifecycle policies, the users can set policies to automatically migrate the data to Standard - Infrequent Access and Amazon Glacier as it ages to further reduce costs.

Simple data transfer. Amazon provides multiple options for cloud data migration and makes it simple and cost-effective for the user to move large volumes of data into or out of Amazon S3. It can be selected from network-optimized, physical disk-based, or third-party connector methods for import to or export from Amazon S3.

Integrated. Amazon S3 is deeply integrated with other AWS services to make it easier to build solutions that use a range of AWS services. Integrations include Amazon

Cloud Front, Amazon Cloud Watch, Amazon Kinesis, Amazon RDS, Amazon Glacier, Amazon EBS, Amazon DynamoDB, Amazon Redshift, Amazon Route 53, Amazon EMR, Amazon VPC, Amazon Key Management Service (KMS), and AWS Lambda.

Security

AWS security. Cloud security at AWS is the highest priority because there are no physical servers or datacenters needed for processing and providing security to the database. All the migration, security and processing of the database is provided through software tools which cost far more less time, money and infrastructure for maintenance compared to the physical servers and storage devices.

An advantage of the AWS Cloud is that it allows the user to scale and innovate while maintaining a secure environment and paying only for the services they use. This means that they can have the security at a lower cost than in an on-premises environment.

Benefits of AWS security.

Keep data safe. The AWS infrastructure puts strong safeguards in place to help protect the user privacy. All data is stored in highly secure AWS data centers.
Meet Compliance Requirements: AWS manages dozens of compliance programs in its infrastructure. This means that segments of the compliance have already been completed.

Save money. Cut costs by using AWS data centers. Maintain the highest standard of security without having to manage your own facility.

Scale Quickly: Security scales with the AWS Cloud usage. No matter the size of the business, the AWS infrastructure is designed to keep the user's data safe.

Networking

Amazon VPC. Amazon Virtual Private Cloud (Amazon VPC) allows the user to create a logically isolated section of the AWS Cloud where they can launch AWS resources in a virtual network as defined. The user has complete control over the virtual networking environment, including the selection of their own IP address range, the creation of subnets, and configuration of routing tables and network gateways. Both IPv4 and IPv6 in the VPC can be used for secure and easy access to resources and applications.

The network configuration for the VPC can easily be customized. There are basically two subnets for the web servers to access the database. The private subnet comprises of all the sensitive database and backend system which does not have access to internet whereas the public subnets have the web servers which have complete access to the internet.

Data Analysis

Create and launch EC2 instance.

Step 1: To launch the EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Launch Instance.
3. Choose an Amazon Machine Image (AMI), find the Amazon Linux AMI at the top of the list and choose Select.

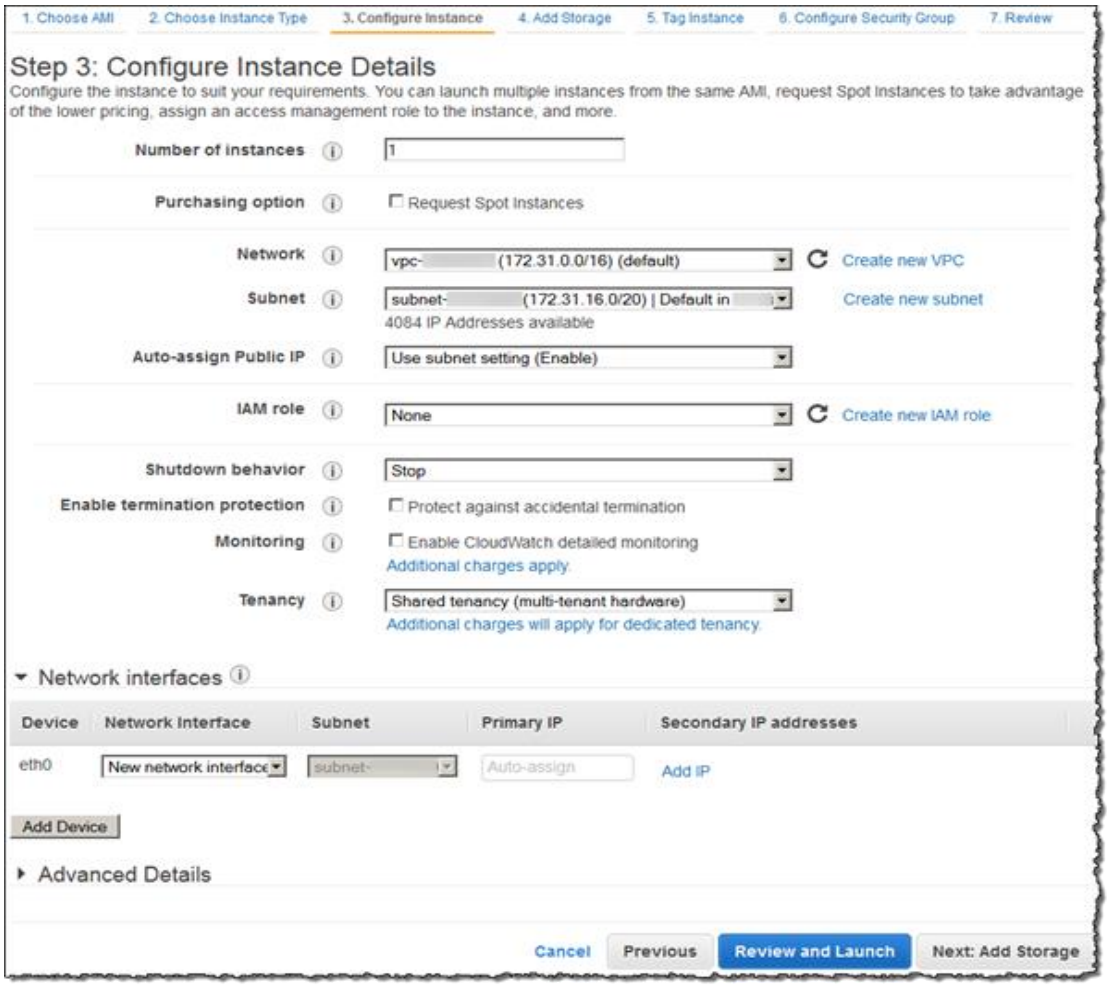

Figure 4.1 Configure Instance Details. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

- **Type:** SSH
- **Protocol:** TCP
- **Port Range:** 22
- **Source:** Anywhere 0.0.0.0/0

4. Choose an Instance Type, choose Next: Configure Instance Details.
    a. Configure Instance Details, choose Network, and then choose the entry for the default VPC. It will look something like vpc-xxxxxxx (172.31.0.0/16) (default).
    b. Choose Subnet, and then choose a subnet in any Availability Zone.
    c. Choose Next: Add Storage.
5. Choose Next: Tag Instance.
6. Name your instance and choose Next: Configure Security Group.

Configure Security Group, review the contents of this page, ensure that Assign a security group is set to Create a new security group, and verify that the inbound rule being created has the following default values.

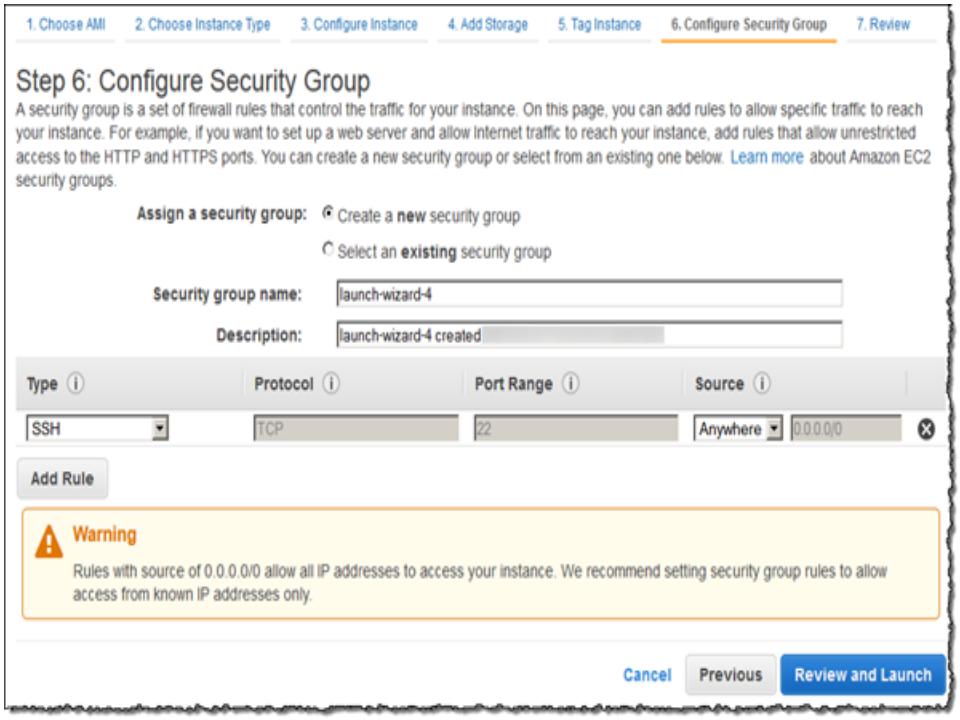

Figure 4.2 Configure Security Group. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

7. Choose Review and Launch.
8. Choose Launch.
9. Select the checkbox for the key pair that is created, and then choose Launch Instances.
10. Choose View Instances.
11. Choose the name of the instance just created from the list, and then choose Actions.
12. From the menu that opens, choose Networking and then choose Change Security Groups.

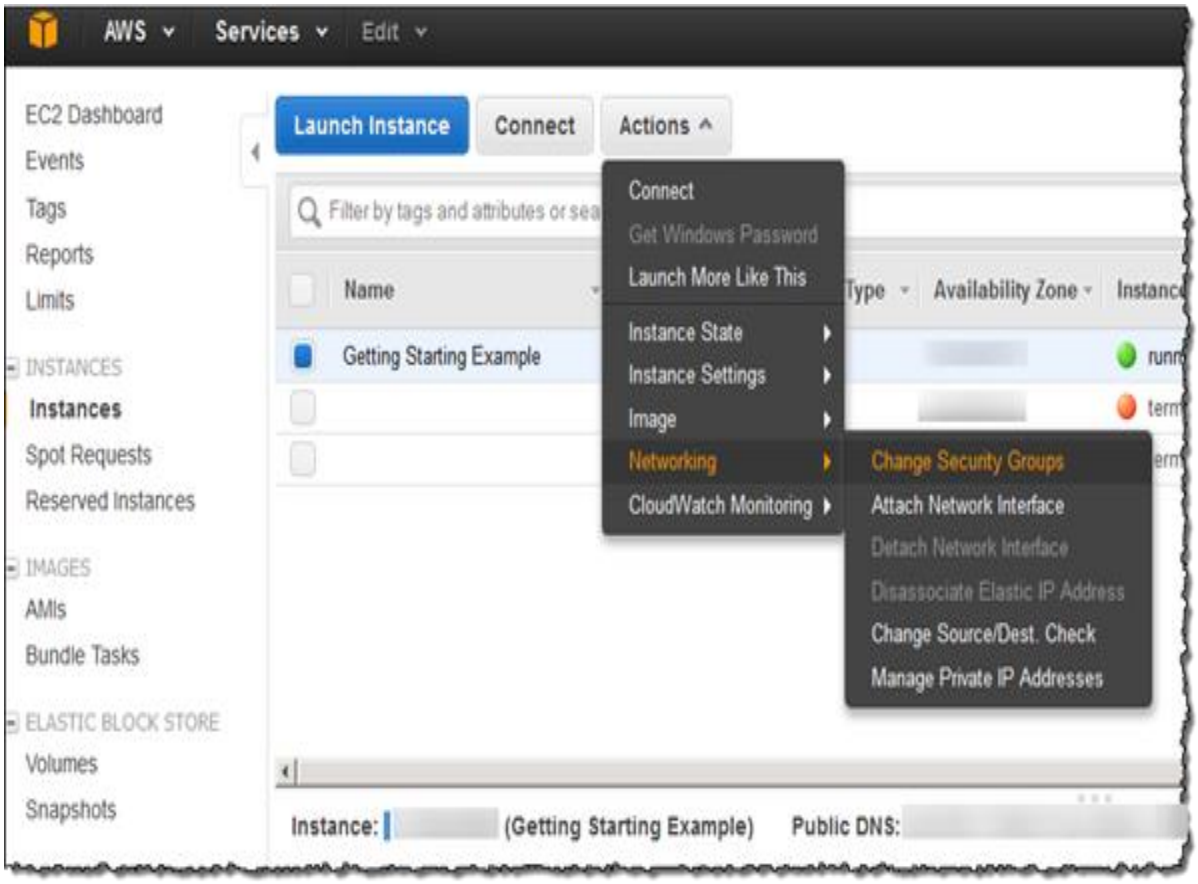

Figure 4.3 Launch an Instance. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

13. Select the checkbox next to the security group with the description default VPC security group.
14. Choose Assign Security Groups.

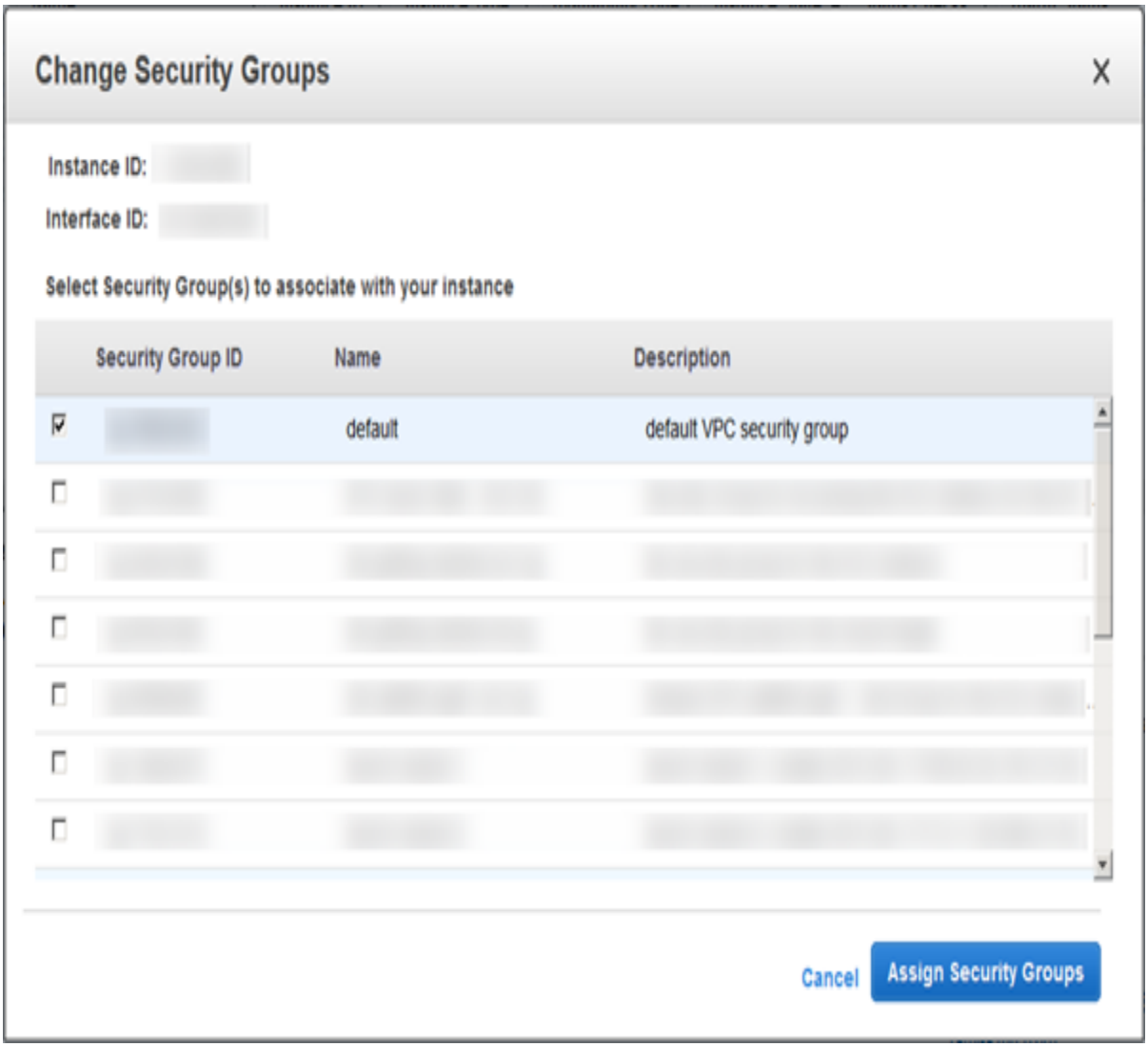

Figure 4.4 Change Security Groups. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

Overview for IPv4

The configuration for this scenario includes the following:

1. A virtual private cloud (VPC) with a size /16 IPv4 CIDR block (example: 10.0.0.0/16). This provides 65,536 private IPv4 addresses.
2. A subnet with a size /24 IPv4 CIDR block (example: 10.0.0.0/24). This provides 256 private IPv4 addresses.
3. An Internet gateway which connects the VPC to the Internet and to other AWS services.
4. An instance with a private IPv4 address in the subnet range (example: 10.0.0.6), which enables the instance to communicate with other instances in the VPC, and an Elastic IPv4 address (example: 198.51.100.2), which is a public IPv4 address that enables the instance to be reached from the Internet.
5. A custom route table associated with the subnet. The route table entries enable instances in the subnet to use IPv4 to communicate with other instances in the VPC and to communicate directly over the Internet. A subnet that's associated with a routing table that has a route to an Internet gateway is known as a public subnet.

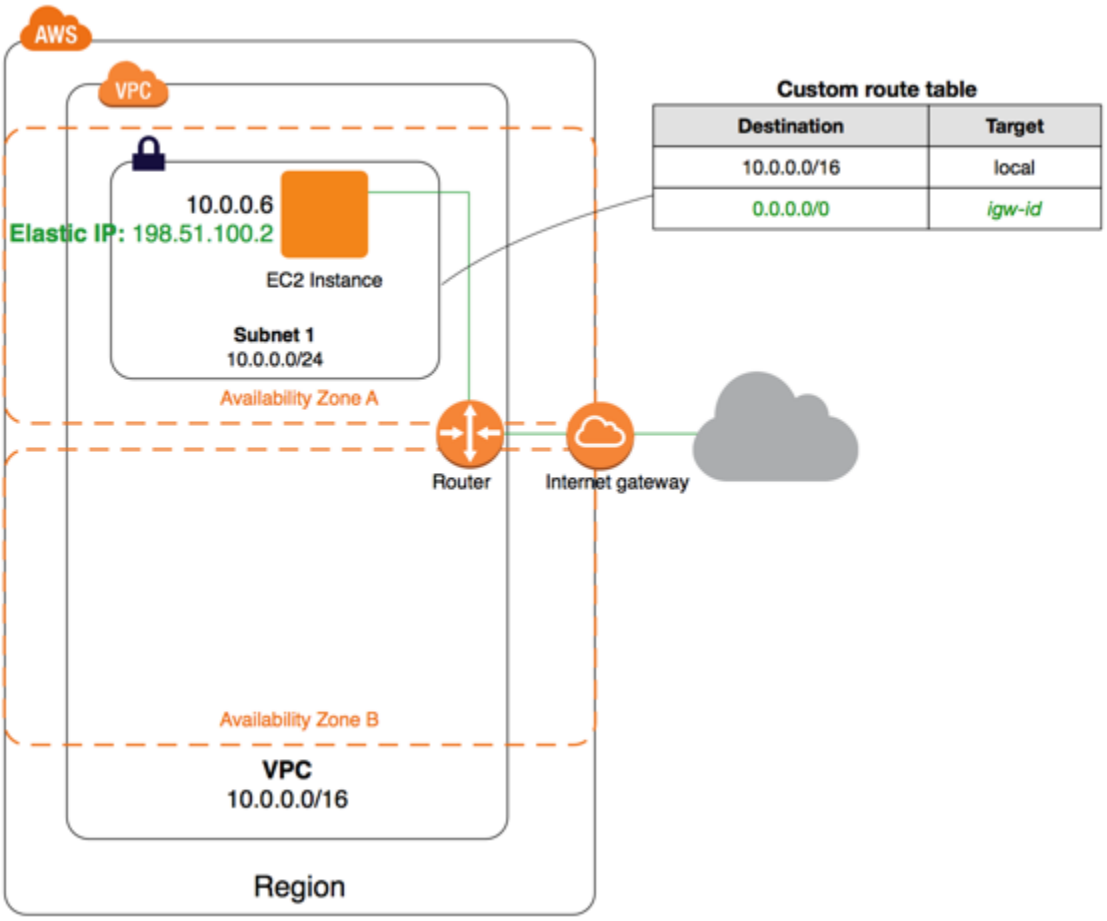

Figure 4.5 Overview for IPv4. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

Overview for IPv6

1. For the scenario, IPv6 can be enabled optionally. In addition to the components listed above, the configuration includes the following:
2. A size /56 IPv6 CIDR block associated with the VPC (example: 2001:db8:1234:1a00::/56). Amazon automatically assigns the CIDR

3. A size /64 IPv6 CIDR block associated with the public subnet (example: 2001:db8:1234:1a00::/64). You can choose the range for your subnet from the range allocated to the VPC. You cannot choose the size of the subnet IPv6 CIDR block.
4. An IPv6 address assigned to the instance from the subnet range (example: 2001:db8:1234:1a00::123).
5. Route table entries in the custom route table that enable instances in the VPC to use IPv6 to communicate with each other, and directly over the Internet.

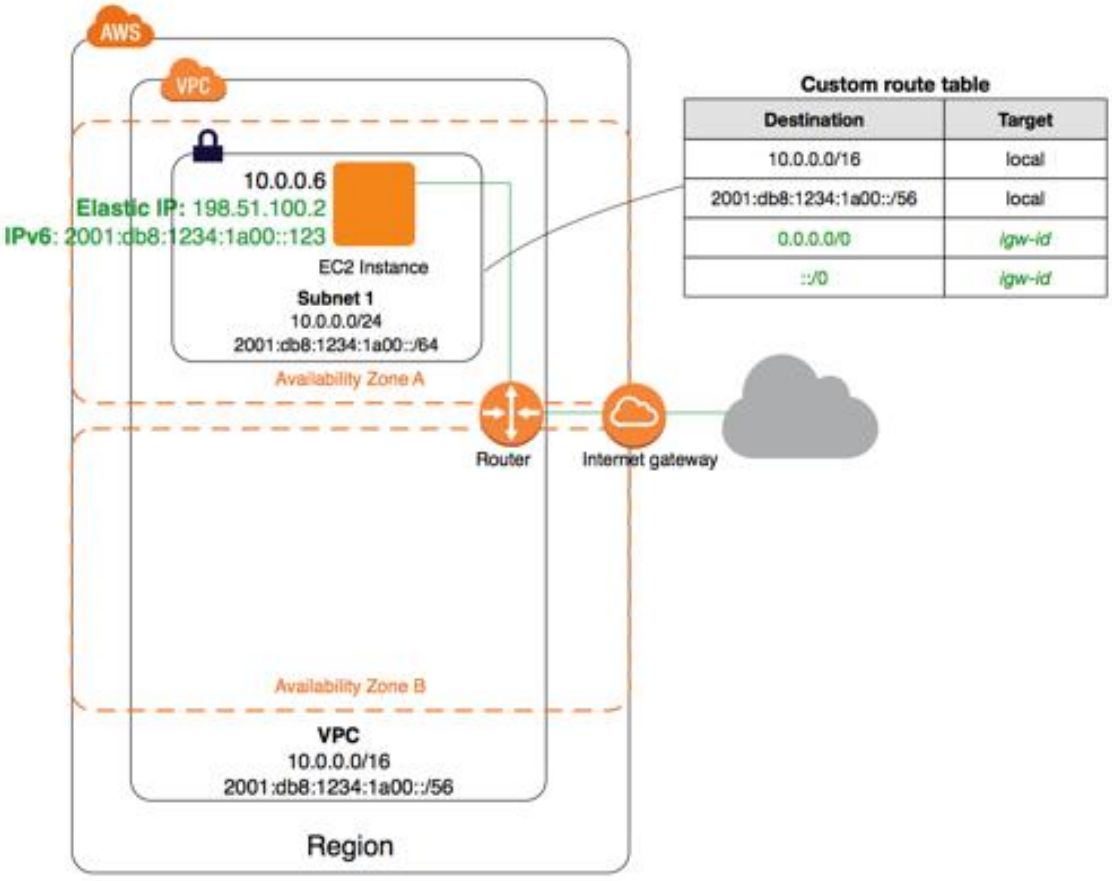

Figure 4.6 Overview for IPv6. Amazon Web Services Architecture, by Mathew, M., 2006, White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

Routing for IPv4. The VPC has an implied router (shown in the configuration diagram above, in Figure 10). In this scenario, the VPC wizard creates a custom route table that routes all traffic destined for an address outside the VPC to the Internet gateway and associates this route table with the subnet.

The following Table 1 shows the route table for the example in Figure 10 above. The first entry is the default entry for local IPv4 routing in the VPC; this entry enables the instances in this VPC to communicate with each other. The second entry routes all other IPv4 subnet traffic to the Internet gateway (for example, igw-1a2b3c4d).

Table 4.1 Routing for IPv4

Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Routing for IPv6. If an IPv6 CIDR block is associated with the VPC and subnet, the route table must include separate routes for IPv6 traffic. The following table shows the custom route table for this scenario if IPv6 communication is enabled in the VPC. The second entry is the default route that's automatically added for local routing in the VPC over IPv6. The fourth entry routes all other IPv6 subnet traffic to the Internet gateway.

Table 4.2 Routing for IPv6

Destination	Target
10.0.0.0/16	local
2001:db8:1234:1a00::/56	local
0.0.0.0/0	*igw-id*
::/0	*igw-id*

Security for IPv4. AWS provides two features that can be used to increase the security in the VPC: security groups *and* network ACLs. Security groups control inbound and outbound traffic for the instances while network ACLs control inbound and outbound traffic for the subnets. In most cases, security groups can meet the needs to avoid SQL injection attacks. However, network ACLs can also be used as an additional layer of security for the VPCs.

For this scenario, a security group is used but not a network ACL. VPC comes with a default security group. An instance that's launched into the VPC is automatically associated with the default security group if a different security group is not specified during the launch. Rules can be added to the default security group, but the rules may not be suitable for other instances that may be launched into the VPC. Instead, creating a custom security group for the web server is recommended.

For this scenario, create a security group named WebServerSG. When a security group is created, it has a single outbound rule that allows all traffic to leave the instances. Rules must be modified to enable inbound traffic and restrict the outbound

traffic as needed. This security group is specified when instances are launched into the VPC. The following are the inbound and outbound rules for IPv4 traffic for the WebServerSG security group.

Table 4.3 Security for IPv4

Inbound			
Source	**Protocol**	**Port Range**	**Comments**
0.0.0.0/0	TCP	80	Allow inbound HTTP access to the web servers from any IPv4 address.
0.0.0.0/0	TCP	443	Allow inbound HTTPS access to the web servers from any IPv4 address
Public IPv4 address range of your network	TCP	22	(Linux instances) Allow inbound SSH access from your network over IPv4. You can get the public IPv4 address of your local computer using a service such as http://checkip.amazonaws.com. If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.
Public IPv4 address range of your network	TCP	3389	(Windows instances) Allow inbound RDP access from your network over IPv4.
The security group ID (sg-xxxxxxxx)	All	All	(Optional) Allow inbound traffic from other instances associated with this security group. This rule is automatically added to the default security group for the VPC; for any custom security group you create, you must manually add the rule to allow this type of communication.

Outbound (Optional)			
Destination	**Protocol**	**Port Range**	**Comments**
0.0.0.0/0	All	All	Default rule to allow all outbound access to any IPv4 address. If you want your web server to initiate outbound traffic, for example, to get software updates, you can leave the default outbound rule. Otherwise, you can remove this rule.

Security for IPv6. If an IPv6 CIDR block is associated with the VPC and subnet, separate rules must be added to the security group to control inbound and outbound IPv6 traffic for the web server instance. In this scenario, the web server will be able to receive all Internet traffic over IPv6, and SSH or RDP traffic from the local network over IPv6. The following are the IPv6-specific rules for the WebServerSG security group (which are in addition to the rules listed above).

Table 4.4 Security for IPv6

Inbound			
Source	**Protocol**	**Port Range**	**Comments**
::/0	TCP	80	Allow inbound HTTP access to the web servers from any IPv6 address.
::/0	TCP	443	Allow inbound HTTPS access to the web servers from any IPv6 address.
IPv6 address range of your network	TCP	22	(Linux instances) Allow inbound SSH access over IPv6 from your network.
IPv6 address range of your network	TCP	3389	(Windows instances) Allow inbound RDP access over IPv6 from your network
Outbound (Optional)			
Destination	**Protocol**	**Port Range**	**Comments**
::/0	All	All	Default rule to allow all outbound access to any IPv6 address. If you want your web server to initiate outbound traffic, for example, to get software updates, you can leave the default outbound rule. Otherwise, you can remove this rule.

Create a VPC

Step 2: To create an AWS VPC

1. Open the Amazon VPC console at **https://console.aws.amazon.com/vpc/**.
2. In the dashboard, choose **Start VPC Wizard**.

3. Select the first option, **VPC with a Single Public Subnet**, and then choose **Select**.
4. For **VPC name** and **Subnet name**, you can name your VPC and subnet to help you to identify them later in the console. You can specify your own IPv4 CIDR block range for the VPC and subnet, or you can leave the default values (10.0.0.0/16 and 10.0.0.0/24 respectively).
5. (Optional, IPv6-only) For **IPv6 CIDR block**, choose **Amazon-provided IPv6 CIDR block**. For **Public subnet's IPv6 CIDR**, choose to **Specify a custom IPv6 CIDR** and specify the hexadecimal pair value for your subnet, or leave the default value (00).
6. You can leave the rest of the default settings, and choose to **Create VPC**.

2.1 To create a VPC and subnets using the AWS CLI

1. Create a VPC with a 10.0.0.0/16 CIDR block and associate an IPv6 CIDR block with the VPC.
2. aws ec2 create-vpc --cidr-block *10.0.0.0/16* --amazon-provided-ipv6-cidr-block
3. In the output that's returned, take note of the VPC ID.
4. Describe your VPC to get the IPv6 CIDR block that's associated with the VPC.
5. aws ec2 describe-vpcs --vpc-id *vpc-2f09a348* v6-cidr-block

6. Create a subnet with a 10.0.0.0/24 IPv4 CIDR block and a 2001:db8:1234:1a00::/64 IPv6 CIDR block (from the ranges that were returned in the previous step).
7. Create a second subnet in your VPC with a 10.0.1.0/24 IPv4 CIDR block and a 2001:db8:1234:1a01::/64 IPv6 CIDR block.]\

2.2 Configure a Public Subnet

1. Create an Internet gateway.
2. In the output that's returned, take note of the Internet gateway ID.
3. Using the ID from the previous step, attach the Internet gateway to your VPC.
4. Create a custom route table for your VPC.
5. In the output that's returned, take note of the route table ID.
6. Create a route in the route table that points all IPv6 traffic (::/0) to the Internet gateway.
7. To confirm that your route has been created and is active, you can describe the route table and view the results.
8. The route table is not currently associated with any subnet. Associate it with a subnet in your VPC so that traffic from that subnet is routed to the Internet gateway. First, describe your subnets to get their IDs. You can use the --filter option to return the subnets for your new VPC only, and the --query option to return only the subnet IDs and their IPv4 and IPv6 CIDR blocks.

9. You can choose which subnet to associate with the custom route table, for example, subnet-b46032ec. This subnet will be your public subnet.

2.3 To launch and connect to an instance in your public subnet

1. Create a key pair and use the --query option and the --output text option to pipe your private key directly into a file with the.pem extension.
2. In this example, launch an Amazon Linux instance. If you use an SSH client on a Linux or OS X operating system to connect to your instance, use the following command to set the permissions of your private key file so that only you can read it.
3. Create a security group for your VPC, and add a rule that allows SSH access from any IPv6 address.
4. Launch an instance into your public subnet, using the security group and key pair that you've created. In the output, take note of the instance ID for your instance.
5. Your instance must be in the running state to connect to the database. Describe your instance and confirm its state, and take note of its IPv6 address.
6. When your instance is in the running state, you can connect to it using an SSH client on a Linux or OS X computer by using the following command. Your local computer must have an IPv6 address configured.

2.4 Launch an Instance into Your Private Subnet

1. Create a security group in your VPC, and add a rule that allows inbound SSH access from the IPv6 address of the instance in your public subnet, and a rule that allows all ICMPv6 traffic:
2. Launch an instance into your private subnet, using the security group you've created and the same key pair you used to launch the instance in the public subnet.
3. Configure SSH agent forwarding on your local machine, and then connect to your instance in the public subnet. For Linux, use the following commands:
4. From your instance in the public subnet (the bastion instance), connect to your instance in the private subnet by using its IPv6 address:
5. From your private instance, a test that you can connect to the Internet by running the ping6 command for a website that has ICMP enabled, for example:
6. To test that hosts on the Internet cannot reach your instance in the private subnet, use the ping6 command from a computer that's enabled for IPv6. You should get a timeout response. If you get a valid response, then your instance is accessible from the Internet—check the route table that's associated with your private subnet and verify that it does not have a route for IPv6 traffic to an Internet gateway.

2.5 Clean Up

1. Delete your security groups
2. Delete your subnets
3. Delete your custom route tables
4. Detach your Internet gateway from your VPC
5. Delete your Internet gateway
6. Delete your egress-only Internet gateway
7. Delete your VPC.

Template 1


```
{
   "InternetGateway": {
      ...
      "InternetGatewayId": "igw-1ff7a07b",
      ...
   }
}
{
   "RouteTable": {
      ...
      "RouteTableId": "rtb-c1c8faa6",
      ...
   }
```

```
}
{
    "RouteTables": [
        {
            "Associations": [],
            "RouteTableId": "rtb-c1c8faa6",
            "VpcId": "vpc-2f09a348",
            "PropagatingVgws": [],
            "Tags": [],
            "Routes": [
                {
                    "GatewayId": "local",
                    "DestinationCidrBlock": "10.0.0.0/16",
                    "State": "active",
                    "Origin": "CreateRouteTable"
                },
                {
                    "GatewayId": "local",
                    "Origin": "CreateRouteTable",
                    "State": "active",
                    "DestinationIpv6CidrBlock": "::/0"
                }
```

```
        ]
    }
  ]
}
[
  {
    "IPv6CIDR": [
      "2001:db8:1234:1a00::/64"
    ],
    "ID": "subnet-b46032ec",
    "IPv4CIDR": "10.0.0.0/24"
  },
  {
    "IPv6CIDR": [
      "2001:db8:1234:1a01::/64"
    ],
    "ID": "subnet-a46032fc",
    "IPv4CIDR": "10.0.1.0/24"
  }
]
{
  "EgressOnlyInternetGateway": {
```

```
        "EgressOnlyInternetGatewayId": "eigw-015e0e244e24dfe8a",
        "Attachments": [
            {
                "State": "attached",
                "VpcId": "vpc-2f09a348"
            }
        ]
    }
}
{
    "GroupId": "sg-e1fb8c9a"
}
{
    "Reservations": [
        {
            ...
            "Instances": [
                {
                    ...
                    "State": {
                        "Code": 16,
                        "Name": "running"
```

```
            },
            ...
            "NetworkInterfaces": {

               "Ipv6Addresses": {
                  "Ipv6Address": "2001:db8:1234:1a00::123"
               }
            ...
         }
      ]
   }
]
}
{
   "GroupId": "sg-aabb1122"
}
```


2.6 To create the WebServerSG security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation panel, choose **Security Groups**.
3. Choose **Create Security Group**.
4. Provide a name and description for the security group. In this topic, the name WebServerSG is used as an example. Select the ID of the VPC from the **VPC** menu, and then choose **Yes, Create**.

5. Select the WebServerSG security group that has just been created. The details panel include a tab for information about the security group, plus tabs for working with its inbound rules and outbound rules.
6. On the **Inbound Rules** tab, choose **Edit**, and then do the following:
   a) Select **HTTP** from the **Type** list, and enter 0.0.0.0/0 in the **Source** field.
   b) Choose **Add another rule**, then select **HTTPS** from the **Type** list and enter 0.0.0.0/0 in the **Source** field.
   c) Choose **Add another rule**, then select **SSH** (for Linux) or **RDP** (for Windows) from the **Type** list. Enter the network's public IP address range in the **Source** field.
   d) (Optional) Choose **Add another rule**, then select **ALL traffic** from the **Type** list. In the **Source** field, enter the ID of the WebServerSG security group.
   e) (Optional, IPv6-only) Choose **Add another rule**, select **HTTP** from the **Type** list, and enter ::/0 in the **Source** field.
   f) (Optional, IPv6-only) Choose **Add another rule**, select **HTTPS** from the **Type** list, and enter ::/0 in the **Source** field.
   g) (Optional, IPv6-only) Choose **Add another rule**, select **SSH** (for Linux) or **RDP** (for Windows) from the **Type**list. Enter the network's IPv6 address range in the **Source** field.
7. Choose **Save**.
8. (Optional) On the **Outbound Rules** tab, choose **Edit**. Locate the default rule that enables all outbound traffic, choose **Remove**, and then choose **Save**.

9. **To launch an instance into the VPC**
10. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
11. From the dashboard, choose **Launch Instance**.
12. Follow the directions in the wizard. Choose an AMI, choose an instance type, and then choose **Next: Configure Instance Details**.
13. On the **Configure Instance Details** page, select the VPC that was created in step 1 from the **Network** list, and then specify a subnet.
14. (Optional) By default, instances launched into a nondefault VPC are not assigned as public IPv4 address. To be able to connect to the instance, assign a public IPv4 address, or allocate an Elastic IP address and assign it to the instance after it's launched. To assign a public IPv4 address, ensure that **Enable** should be selected from the **Auto-assign Public IP** list.
15. (Optional, IPv6-only) Auto-assign an IPv6 address to the instance from the subnet range. For **Auto-assign IPv6 IP**, choose **Enable**.
16. On the next two pages of the wizard, configuration for the storage of the instance, and addition of tags can be done. On the **Configure Security Group** page, select the **Select an existing security group** option, and select the **WebServerSG** security group which was created in step 2. Choose **Review and Launch**.
17. Review the settings and then choose **Launch** to choose a key pair and launch the instance.

18. If a public IPv4 address is not assigned to the instance in step 5, you will not be able to connect to it over IPv4. Assign an Elastic IP address to the instance:
19. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
20. In the navigation pane, choose **Elastic IPs**.
21. Choose **Allocate new address**.
22. Choose **Allocate**.

- Select the Elastic IP address from the list, choose **Actions**, and then choose an **Associate address**.
- Select the instance to associate the address with, and then choose **Associate**.

Create an Amazon S3 Bucket

Step 3: To create an Amazon S3 Bucket

To create an Amazon S3 bucket use the Amazon S3 console. But a simpler way to create resources is often to use an AWS Cloud Formation template. The following template creates an Amazon S3 bucket for this example and sets up instance profile with an IAM role that grants unrestricted access to the bucket.

Template 2

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Resources" : {
    "AppServerRootRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
```

```
      "AssumeRolePolicyDocument": {
        "Statement": [ {
          "Effect": "Allow",
          "Principal": {
            "Service": [ "ec2.amazonaws.com" ]
          },
          "Action": [ "sts:AssumeRole" ]
        } ]
      },
      "Path": "/"
    }
  },
  "AppServerRolePolicies": {
    "Type": "AWS::IAM::Policy",
    "Properties": {
      "PolicyName": "AppServerS3Perms",
      "PolicyDocument": {
        "Statement": [ {
          "Effect": "Allow",
          "Action": "s3:*",
          "Resource": { "Fn::Join" : ["", [ "arn:aws:s3:::", { "Ref" : "AppBucket" } , "/*" ]
          ] }
```

```
            } ]
          },
          "Roles": [ { "Ref": "AppServerRootRole" } ]
        }
      },
      "AppServerInstanceProfile": {
        "Type": "AWS::IAM::InstanceProfile",
        "Properties": {
          "Path": "/",
          "Roles": [ { "Ref": "AppServerRootRole" } ]
        }
      },
     "AppBucket" : {
      "Type" : "AWS::S3::Bucket"
      }
  },
  "Outputs" : {
      "BucketName" : {
          "Value" : { "Ref" : "AppBucket" }
      },
      "InstanceProfileName" : {
          "Value" : { "Ref" : "AppServerInstanceProfile" }
```

```
    }
  }
}
```

To create the Amazon S3 bucket:

1. Copy the example template to a text file on the system.

This example assumes that the file is named appserver.template.

2. Open the AWS Cloud Formation console and click **Create Stack**.

3.In the **Stack Name** box, enter the stack name.

This example assumes that the name is **AppServer**.

4. Click **Upload template file**, click **Browse**, select the Appserver.template file that was created in Step 1, and click **Next Step**.

5. On the **Specify Parameters** page, select **I acknowledge that this template may create IAM resources**, then click **Next Step** on each page of the wizard until you reach the end. Click **Create**.

6. After the **AppServer** stack reaches **CREATE_COMPLETE** status, select it and click its **Outputs** tab.

7. On the **Outputs** tab, record the **BucketName** and **InstanceProfileName** values for later use.

Summary

AWS consists of some major key components like EC2 instances, Storage Services, VPN, public and private subnets which are explained in analyzing the results. Steps in creating the Ec2 instances, coding the IPv4 and IPv6 instances and creating the public and private subnets with steps in creating the amazon s3 buckets both in IPv4 and IPv6 have been explained. Steps for creating the WebserverSG security group is also been discussed.

Chapter V: Conclusions and Future Work

Results

Amazon Web Services Web Application Firewall (AWS WAF) helps to protect web applications from common web exploits like SQL injection attacks that could affect application availability, compromise security, or consume excessive resources. Amazon's Elastic Cloud Computing (EC2) platform allows applications to run on an instantly scalable number of processors on demand, while Amazon's Simple Storage System (S3) allows access to a practically infinite allocation of disk space on demand with multiple layers of security.

1) Does the AWS firewall provide better security than the WAF?

A) Yes, it does provide multiple layers of security at every stage of process to avoid SQL injection attacks as it provides complete control to the user over the virtual networking environment, including selection of the own IP address range, creation of subnets, and configuration of route tables and network gateways as well as creating public and private subnets.

2) Does AWS provide high performance databases?

A) Amazon S3 provides the most durable, cost effective and highly secured databases. S3 buckets can be configured to control the access of the data through IAM.

3) Is AWS feasible for any organization?

A) Yes because of the cost-effective system and the users have to pay for what they use it is very convenient for any scale of organization.

Conclusion

As the WAF was not completely capable of defending the SQL injection attacks, AWS is being used because of the multiple layers of security and access, it provides for the databases either by providing a private VPN with gateway authorities or by creating multiple subnets or by providing access to ports for the databases.

AWS provides required number of processors on demand as well as a scalable number of databases on demand with multiple layers of security in the form of VPN's, gateways, portals, public and private subnets avoiding the hardware requirements for the organizations.

As the subnets can be created public and private the data and the permissions can be secured and authenticated at different role levels which protect the integrity and security of the data as the critical information will not be available to all the users.
The storage services particularly Amazon S3 is provided with the gateway services which blocks the unwanted IP addresses and allows access to the databases only for the IP addresses registered on S3.

Future Work

In-depth study of the AWS management console should be carried out like studying about the glacier, snowball which is an advanced level of data storage services in AWS. Similarly, creating multiple subnets in IPv4, IPv6 and route 53 privacy.

References

Anley, C., “Advanced SQL Injection in SQL Server Applications,” White paper, Next Generation Security Software Ltd, 2002.

Boyd, S. W., Keromytis, A. D., “SQLrand: Preventing SQL Injection Attacks,” In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, v. 3089, pp. 292–302, June 2004.

Buehrer, G., Weide, B. W., & Sivilotti, P. A. G. “Using parse tree validation to prevent SQL injection attacks.” Proceedings of the 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, pp. 106-113. doi:10.1145/1108473.1108496, 2005.

Cook, W. R., & Rai, S. “Safe query objects: Statically typed objects as remotely executable queries.” Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA, pp. 97-106. doi:10.1145/1062455.1062488, 2005.

Dharam, R.; Shiva, S.G., "Runtime monitors for tautology-based SQL injection attacks," Cyber Security, International conference on Cyber Warfare and Digital Forensic (CyberSec), pp. 26-28, June 2012.

Elia, I. A., Fonseca, J., & Vieira, M., “Comparing SQL injection detection tools using attack injection” An experimental study Software Reliability Engineering (ISSRE), IEEE 21st International Symposium on Software Reliability Engineering, pp. 289-298. doi:10.1109/ISSRE.2010.32, 2010.

Gould, C., Su, Z., & Devanbu, P., “JDBC checker: A static analysis tool for SQL/JDBC applications,” In Proceedings of the 26th International Conference on Software Engineering, pp. 697-698. doi:10.1109/ICSE.2004.1317494, 2004.

Gould, C., Su, Z., & Devanbu, P., “Static Checking of Dynamically Generated Queries in Database Applications,” In Proceedings of the 26th International Conference on Software Engineering, pp. 645–654, doi:10.1109/ICSE.2004.1317494, 2004.

Haldar, V., Chandra, D., & Franz, M., “Dynamic taint propagation for java,” In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05), pp. 309-311. doi:10.1109/CSAC.2005.21, Dec 2005.

Halfond, W. G., Viegas, J., & Orso, A. “AMNESIA: Analysis and monitoring for Neutralizing SQL-injection attacks,” Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, pp. 174-183, doi:10.1145/1101908.1101935, Nov 2005.

Halfond, W. G., Viegas, J., & Orso, A. “Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks,” In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), St. Louis, MO, USA, pp. 22–28, doi:10.1145/1101908.1101935, May 2005.

Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15). IEEE.

Howard, M., & Leblanc, D. E., “Writing secure code,” (2nd ed.). Redmond, WA, USA: Microsoft Press, 2003.

Huang, Y., Huang, S., Lin, T., & Tsai, C., “Web application security assessment by fault injection and behavior monitoring,” Proceedings of the 12th International Conference on World Wide Web, Budapest, Hungary, pp. 148-159. doi:10.1145/775152.775174, 2003.

Huang, Y., Yu, F., Hang, C., Tsai, C., Lee, D., & Kuo, S., “Securing web application code by static analysis and runtime protection,” Proceedings of the 13th International Conference on World Wide Web, New York, NY, USA, pp. 40-52. doi:10.1145/988672.988679, 2004.

Kar, D., Panigrahi, S., & Sundararajan, S. (2016). SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM. *Computers & Security*, *60*, 206-225.

Kiani, M., Clark, A., & Mohay, G., “Evaluation of anomaly-based character distribution models in the detection of SQL injection attacks.” Third International Conference on Availability, Reliability and Security, pp. 47-55. doi:10.1109/ARES.2008.123, 2008.

Litchfield, D., “Web Application Disassembly with ODBC Error Messages,” Technical document, @Stake, Inc., 2002.

Livshits, V. B., & Lam, M. S., “Finding security vulnerabilities in java applications with static analysis.” Proceedings of the 14th Conference on USENIX Security Symposium, Baltimore, MD. Volume 14, pp. 271–286, Aug. 2005.

Labs, S., “SQL Injection,” White paper, SPI Dynamics, Inc., 2002. http://www.spidynamics.com/assets/documents/ WhitepaperSQLInjection.pdf.

Mathew, M., “Amazon Web Services Architecture,” White Paper, Amazon Web Services, Inc., 2006. https://aws.amazon.com/security/documents/WhitepaperAWSWAF.pdf.

Martin, M., Livshits, B., & Lam, M. S., “Finding application errors and security flaws using PQL: A program query language.” In Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications (OOPSLA 2005), v. 40(10), pp. 365-383. doi:10.1145/1103845.1094840, 2005.

McClure, R. A., & Kruger, I. H., “SQL DOM: Compile time checking of dynamic SQL statements.” Proceedings of 27th International Conference on Software Engineering. ICSE 2005. pp. 88-96, doi:10.1109/ICSE.2005.1553551, 2005.

McDonald, S., “SQL Injection: Modes of attack, defense, and why it matters,” White paper, GovernmentSecurity.org, April 2002.

Pietraszek, T., & Berghe, C. V., “Defending against injection attacks through context-sensitive string evaluation.” Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection, Seattle, WA. pp. 124-145. doi:10.1007/11663812_7, 2006.

Pinzón, C., Paz, J. F. D., Bajo, J., Herrero, Á., & Corchado, E., “AIIDA-SQL: An adaptive intelligent intrusion detector agent for detecting SQL injection attacks.” 2010 10th International Conference on Hybrid Intelligent Systems, pp. 73-78. doi:10.1109/HIS.2010.5600026, Aug 2010.

Sadeghian, A., Zamani, M., & Ibrahim, S., “SQL injection is still alive: A study on SQL injection signature evasion techniques.” 2013 International Conference on Informatics and Creative Multimedia, pp. 265-268. doi:10.1109/ICICM.2013.52, 2013.

Sadeghian, A., Zamani, M., & Manaf, A. A., “A taxonomy of SQL injection detection and prevention techniques.” 2013 International Conference on Informatics and Creative Multimedia, pp. 53-56. doi:10.1109/ICICM.2013.18, Sept 2013.

Sadeghian, A., Zamani, M., & Manaf, A. A., “A taxonomy of SQL injection detection and prevention techniques.” 2013 International Conference on Informatics and Creative Multimedia, pp. 53-56. doi:10.1109/ICICM.2013.18 Sept 2013.

Spett, K., “Blind SQL injection,” White paper, SPI Dynamics, Inc., 2003. http://www.spidynamics.com/whitepapers/ Blind SQLInjection.pdf.

Scott, D., & Sharp, R., “Abstracting application-level web security.” Proceedings of the 11th International Conference on World Wide Web, Honolulu, Hawaii, USA, pp. 396-407. doi:10.1145/511446.511498, 2002.

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) SIGPLAN Not., 41(1), pp. 372-382. doi:10.1145/1111320.1111070, Jan 2006.

Tajpour, A., & Shooshtari, M. J. z., “Evaluation of SQL injection detection and prevention techniques.” 2010 2nd International Conference on Computational

Intelligence, Communication Systems and Networks, pp. 216-221. doi:10.1109/CICSyN.2010.55, July 2010.

Valeur, F., Mutz, D., & Vigna, G., “A learning-based approach to the detection of SQL attacks.” Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Vienna, Austria. pp. 123-140. doi:10.1007/11506881_8, July 2005.

Wan, M., & Liu, K., “An improved eliminating SQL injection attacks based regular expressions matching.” 2012 International Conference on Control Engineering and Communication Technology, pp. 210-212. doi:10.1109/ICCECT.2012.235, Dec 2012.

Wassermann, G., & Su, Z., “An Analysis Framework for Security in Web Applications,” In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2004), pp. 70–78, 2004.

Wei, K., Muthuprasanna, M., & Kothari, S., “Preventing SQL injection attacks in stored procedures.” Australian Software Engineering Conference (ASWEC'06), v.8, pp. 18-21. doi:10.1109/ASWEC.2006.40, April 2006.

Wu, X., & Chan, P. P. K., “SQL injection attacks detection in adversarial environments by k-centers.” 2012 International Conference on Machine Learning and Cybernetics, pp. 406-410. doi:10.1109/ICMLC.2012.6358948, July 2012.

Zhang, K. X., Lin, C. J., Chen, S. J., Hwang, Y., Huang, H. L., & Hsu, F. H., “TransSQL: A translation and validation-based solution for SQL-injection attacks.” 2011 First

International Conference on Robot, Vision and Signal Processing, pp. 248-251. doi:10.1109/RVSP.2011.59, Nov 2011.