International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 3 Issue 1, January-2014, pp: (107-113), Impact Factor: 1.252, Available online at: **www.erpublications.com**

# Is the Future of Cloud Computing More Insecure?

## Distraction and Security best practices for cloud computing measures using Apache Hive

Subodh Kant[1], Puneet Kannaujia[2], Leena Singh[3]
[1,2,3]Department of CSE, SRMSCET, Bareilly (U.P.), INDIA

**Abstract: This paper notes some significance trends related to internet and "Cloud Computing Security issues" that will handle the way cloud security is delivered and experienced. Today more information and latest communication technology is experiencing change that, regardless of some skepticism and bringing to the concept of " utility computing" it's our opinion , it prepare ground for constructive critiques regarding the security of such a computing paradigm and especially one of the key components: web services and security best practices for cloud computing using Apache Hive and SQL injection. Apache hive is an open source data warehouse system for Hadoop framework. This paper gives a brief overview of Hive and describe how the use of Thrift classes while executing a Hive query through a java application is vulnerable to SQL injection and how it can be mitigated, the mitigation process makes Hive application secure.**

**Keywords: Apache Hive, SQL injection, web services, cloud computing, security management system.**

## I. INTRODUCTION

Internet security problems remain a major challenge with many security concerns such as internet worms, span and phishing attack. Anyone can write a web application but not everyone can secure one by them service. Because web application is easy to develop, many are being rushed to market without end user focus on security.

Consider a few skeletons in a rich variety of attack available to hacker: JSON Hijacking, Cross side request forgery, Http response splitting, SQL injection, LDAP injection, X-Path injection, Shell command injection, server side include injection, cross side scripting and a directory traversal. In this paper we have summarized various security features implemented using Apache Hive.

Easy data summarization. Analysis of large database stored in Hadoop-DFS.Adhoc queries using a SQL like language, called Hive query language. Allows map/reduce programmers to plug in their custom mappers and reducers.

Section II describes literature review. Section III describes the architecture of Hive. Section IV describes the Hive data model. Section V describes executing Hive query. Section VI describes Hive web interface. Section VII describes demo of SQL injection on Hive. Section VIII describes demo of Hive prepared statement. Finally, section IX describes the conclusion and the future work.

## II. LITERATURE REVIEW

In this section we briefly review the major challenges brought forth by cloud computing and service orientation. In particular, we focus on security-related challenges, referring the interested reader to [1] for a more comprehensive vision of the new obstacles induced by cloud computing. We divide security-related challenges into two groups. On one hand, there are challenges that are already evident to web users (e.g., web service providers, developers), hence need practical solutions (). On the other hand, there are challenges that will significantly influence the security of the Web in the long run. As motivated in Section II.B, this second group of challenges is relevant, yet less obvious.
If you're thinking about moving some of your IT services to the cloud, or if you are a cloud services provider, you're probably thinking about security. According to a recent IBM survey of IT managers, business stakeholders in IT and CIOs,

respondents said that cloud computing was a risky to extremely risky proposition. Seventy-seven percent of respondents believe that adopting cloud computing makes protecting privacy more difficult.

### A. Challenges with immediate impact

The challenges described in this section are already visibly impacting users and providers. In particular, given the amount of data shared across these infrastructures, data confidentiality, trust relationships and shared reputation are concerning issues.

#### 1) Data confidentiality

The obvious and, in general, effective measure to protect data confidentiality is encryption. However, encryption is not always a feasible solution, especially for data-intensive applications that require high I/O throughput (note that, in [1], the relatively low speed of the Internet has been al- ready identified as a concerning obstacle). Although homomorphic encryption [10] can be exploited for limiting decryptions and re-encryptions when data needs to be transformed, in its current stage this solution requires significant efforts to be adopted in high-speed, real-world deployments.

#### 2) Sharing shared resources

The security issues typical of shared hosting environments are magnified in the case of highly distributed, in-the-cloud systems that host modern web services. The additional, un- perceived, complexity due to dynamic resource slicing, allocation, replication and optimization, gives indeed each user the illusion of being unique. In reality, each user (e.g., an actual system user or an application) operates in a shared environment with “porous" boundaries. Therefore, users may behave maliciously, or compromise virtualization software, affecting other users and their reputation.
In reality, a cloud instance is nothing more than an advanced and very well managed virtual machine hypervisor (and a web service is basically a sophisticated and well managed web application instance). Thus, the feasibility of compromising other slices of a cloud or other accounts of a service depends a good deal on the security of the service management platforms. As a representative example, vulnerabilities in VMware have grown 35 times between 1999 and 2007 [8]. This, unfortunately, is not comforting. In our opinion, the efforts to secure cloud instances should focus on two complementary directions. On one hand, hypervisor-based detection mechanisms such as the one proposed in [9] could be effectively adapted to cloud systems to recognize misbehaving slices. On the other hand, once identified, malicious slices could be dynamically re-allocated in a honeypot-like environment, not only to contain their activity but also to collect data and analyze their actions in order to design specific countermeasures.

### B. Challenges with delayed impact

Debugging and auditing in large-scale, distributed systems unavoidably affect the foundations of secure software development. Although their impact may be delayed, and no incidents can be attributed directly to them as of now, we believe that these obstacles will influence significantly the security of the software developed for, or deployed onto, modern computing infrastructures.

## III. HIVE ARCHITECTURE

Table 1: Hive Components

| Component | Description |
|---|---|
| Web UI/CLI | Interface for user to submit queries and other operation |
| Driver | Create a session |
| Compiler | Pass the query |
| Meta store | Store all the information about the table |
| Execution engine | Execution plan |

## IV. HIVE DATA MODEL

| Table | Partition | Bucket |
|---|---|---|

```
CREATE TABLE Cloud_page(view_Time INT,user_Id BIGINT, page_url STRING, ip STRINGCOMMENT ”ip address of the user”)
COMMENT ‘This is the page cloud table’
PARTITIONED BY (country STRING) CLUSTERED BY (user_id) SORTED BY (page_url) INTO 60 BUCKETS
ROW FORMAT DELIMETED
FIELDS TERMINATED BY “48”
STORED AS SEQUENCEFILE;
```

## V. EXECUTING HIVE QUERIES

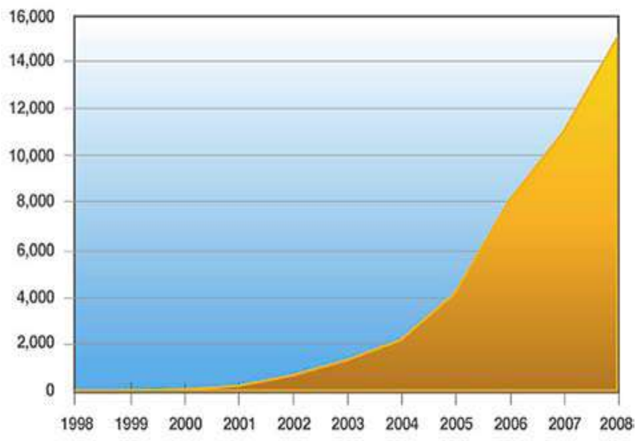
Hive queries language is a SQL like ad-hoc query language for provided by Hive. Hive Query Language (HQL) support the Data Definition Statement (DDS), Data Manipulation Statement (DMS), SQL operation and User Defined Function (UDF) and User Defined Aggregation Function (UDAF), Script. Following are the three standard ways to execute Hive queries:

Table 2: Executing Hive Queries

| | |
|---|---|
| Command Line Interface | To enter the Hive shell, execute the command $ Hive_Home/bin/Hive |
| Application Programming Interface (JAVA) | When we execute Hive queries through API like Java, **Thrift server** play a key role |
| Hive web interface (web GUI) | To work with the web interface, we need to start the web server listener. To access the web UI address<br><br>http://<host_name>:<Hiveserver_portnumber>/hwi |

**Web application cumulative vulnerability count 1998-2008**

As of the end of 2008, 54.9% of all disclosed vulnerabilities are Web application vulnerabilities. And out of all Web application vulnerabilities disclosed during 2008, 74 percent had no patch by the end of 2008. Reported Web application Vulnerabilities.



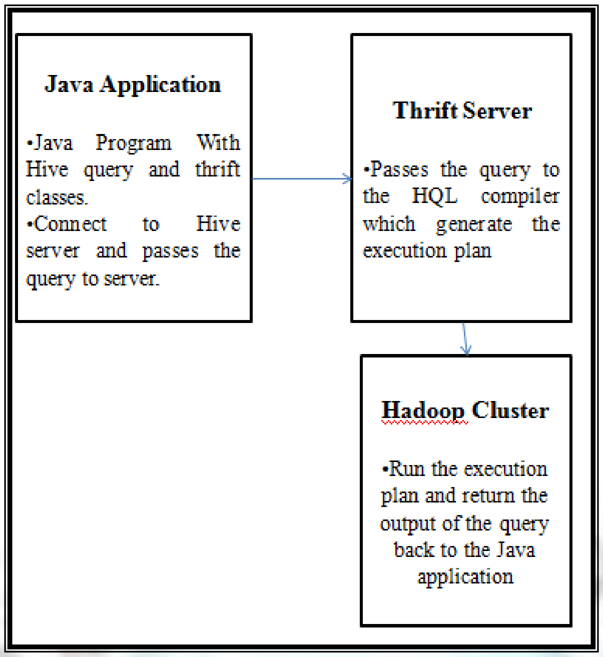

Java Application
•Java Program With Hive query and thrift classes.
•Connect to Hive server and passes the query to server.
Thrift Server
•Passes the query to the HQL compiler which generate the execution plan
Hadoop Cluster
•Run the execution plan and return the output of the query back to the Java application

Figure 1: Executing Hive Queries through Java application

## VI. DEMO OF SQL INJECTION ON HIVE

Let us consider a scenario:

There is a table Customer in the Hive database. There are 5 columns in the table: CustAcctId, Name, Age, Address and Zip code.

**Case 1:** Normal input (without any malicious input)

**Demoquery.java**

```
}
}
Public static void normal select query ()
{
TSocket transport=new TSocket(“10.68.203.176”,3333,9999);
Try
{
TBinaryProtocol protocol=new TBinaryProtocol (“transport);
{ThriftHive.Clientclient=new ThriftHive.Client(protocol);
String name=”rahul”;
String City=”UttarPradesh,INDIA”;
String query=”SELECT COUNT(*)FROM customer WHERE name=’””+name+” ‘ AND Address=’ “+city+ “ ‘ “;
Transport.open();
Client.execute(query);
List<String> rows=client.fetchAll();
System.out.prinln(“outfut for normal select query:”);
For (String str:rows)
{
System.out.println(str);
}
```

Figure 3: Vulnerable code with Normal Input

**Case 2:** Malicious Input (Results to SQL Injection)

**Demoquery.java**

```
}
}
Public static void DemoSqlInjection()
{

TSocket transport=new TSocket(“10.68.203.176”,3333,9999);
Try
{
TBinaryProtocol protocol=new TBinaryProtocol (“transport);
{ThriftHive.Clientclient=new ThriftHive.Client(protocol);
String name=”rahul”;
String City=”UttarPradesh,INDIA” OR ‘1’=’1’;
String query=”SELECT COUNT(*)FROM customer WHERE name=’””+name+” ‘ AND Address=’ “+city+ “ ‘ “;
Transport.open();
Client.execute(query);
List<String> rows=client.fetchAll();
System.out.prinln(“outfut for normal select query:”);
For (String str:rows)
{
System.out.println(str);
}
Transport.close();
```

## VII. HOW TO SQL INJECTION CAN BE MITIGATED

There is some possible solution for SQL Injection can be mitigated.

- Proper input validation/Whitelisting or blacklisting of characters
- Avoid using dynamic queries(use of precompiled queries like PreparedStatement Class)

Another way to mitigate the injection attack is not to use dynamic queries for execution.To implement the same ,the “Hive PreparedStatement” class should be used.The class is defined inside the package “org.apache.hadoop.hive.jdbc”,provided in Hive release.For our refrence ,we have used “Hive-0.7.1”release.

The Package “org.apache.hadoop.hive.jdbc” contain the following classes:

| | | |
|---|---|---|
| HiveBaseResultSet | HiveCallable statement | HiveStatement |
| HiveConnection | HiveDatabaseMetaDtaa | jdbcColumn |
| HiveDataSource | HiveMetaDataResultSet<M> | Jdbc Table |
| Hive Driver | HivePreparedStatement | TestJdbcDriver |
| HiveQueryResultSet | HiveResultSetMetaData | |

```
Public static void DemoSqlInjection()
{

TSocket transport=new TSocket(“10.68.203.176”,3333,9999);
Try
{
TBinaryProtocol protocol=new TBinaryProtocol (“transport);
{ThriftHive.Clientclient=new ThriftHive.Client(protocol);
String name=”rahul”;
Public static void HivePreparedStatement_demo() throws SQLException,TTransportException
```

```
{
TSocket  transport =new TSocket("10.68.203.176",33333,99999);
Transport.open();
TbinaryProtocol protocol=new TBinaryProtocol(transport);
HiveClient client=new HiveClient(protocol);
```

## VIII. DEMO OF HIVE PREPAREDSTATEMENT

Steps to execute a hive query through a java API using HIVEClent class

- Create and initialize a Thrift Socket object to connect to the hive server and Thrift Binary Protocol Object
- Execute open () method of TSocket object
- Create an object of HiveClient class
- Create an Object of HivePrepaerdStatement class with query string
- pass the input values to the hivePreparedStatement
- Execute the Hive Prepared Statement.

**Case 1**: Normal Input (without any malicious input)

```
String name ="jakes";
String city="Kerla,India ;
String query="SELECT COUNT (*) FROM customer WHERE name=? AND Address=?";
HivePreparedStatement pre_stmt=new HivePreparedStatement(client,query);
Pre_stmt.setString(1,name);
Pre_stmt.setString(2,city);
ResultSet rs=pre_stmt.executequery();
```

**Case 2**: Malicious Input

```
String name ="jakes";
String city="Kerla,India or 1=1";
String query="SELECT COUNT (*) FROM customer WHERE name=? AND Address=?";
HivePreparedStatement pre_stmt=new HivePreparedStatement(client,query);
Pre_stmt.setString(1,name);
Pre_stmt.setString(2,city);
ResultSet rs=pre_stmt.executequery();
```

## IX. CONCLUSION AND FUTURE WORK

In this paper, we have discussed some key points that, in our opinion, motivate a constructive reconsideration of the current security measures. Even if exploring new business models opened up by cloud computing falls entirely outside of the scope of this paper, it is undeniable that the fast-growing underground economy has already embraced the cloud model (in fact, botnets are an embryonic distributed malicious infrastructure.

The use of **ThriftHive.Client** class to execute Hive queries is vulnerable to SQL injection.
The mitigate SQL injection, use of Hive prepared statement should be a part of SECURED JAVA CODING PRACTICE FOR HIVE.
The use HivePreparedStatement ensures that the SQL injection is mitigated.

## X. REFERENCES

[1]. S.Gordeychik Web application security statistics http://projects.webappsec.org/Web-Application_Security- Statistics, January 2010.
[2]. IBM Press Release, IBM Introduces New Software and Cloud Services to help Companies Improve Processes, Automate Decisions, October 12,2010.
[3]. http://cwiki.apache.org/Hive.
[4]. C.Balding.Cloud security .http://cloudsecurity.org/,2009.
[5]. http://hive.apache.org/
[6]. http://www.karmasphere.com
[7]. O'Reilly Hadoop: The Definitive Guide 2nd Edition
[8]. K. Lamb. Virtualization and security. http://blogs.iss.net/archive/virtblog.html, September 2007.
[9]. T. Gar_nkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In Proc. Network and Distributed Systems Security Symposium, February 2003.
[10]. C. Gentry. Fully homomorphic encryption using ideal lattices. In STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing, pages 169{178, New York, NY, USA, 2009. ACM.