ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 3, March 2014

Detection Block Model for SQL Injection Attacks

Diksha Gautam Kumar, Madhumita Chatterjee

Student, Dept. of C.E., PIIT, New Panvel, Navi Mumbai ,India

Professor, Dept. of C.E., PIIT, New Panvel, Navi Mumbai ,India

ABSTRACT: With the rapid development of Internet, more and more organizations connect their databases to the Internet for resource sharing. However, due to developers' lack of knowledge of all possible attacks, web applications become vulnerable to multiple attacks. Thus the network databases could face multiple threats. Web applications generally consist of a three tier architecture where database is in the third pole, which is the most valuable asset in any organization. SQL injection is an attack technique used to exploit code by altering back-end SQL statements through manipulating input. An attacker can directly compromise the database, that's why this is a most threatening attack.SQL injection attack occupies first position in top ten vulnerabilities as specified by Open Web Application Security Project[12]. It is probably the most common Website vulnerability today!

Current scenarios which provide solutions to SQL injection attack either have limited scope i.e. can't be implemented in all platforms or do not cover all types of SQL injection attacks. In this work we implement Detection Block model against SQL injection attacks. The model works both on client and server side. Client side implements a filter function and server side is based on information theory. MAC of static and dynamic query which is derived from entropy is compared to detect an attack.

Keywords: SQL injection, information theory, entropy, web attacks, system security.

I. Introduction

This SQL Injection Attacks are command-injection attacks where the attacker injects a malicious SQL query into back-end database through web application interface. The back-end database executes the injected SQL statement and sends the corresponding execution results back to the attacker. The attacker could submit malicious SQL commands directly to the back-end database to extract confidential information or even obtain the root privilege of database.

SQL Injection (SQLI) is a wide spread vulnerability commonly found in web-based programs. Exploitations of SQL injection vulnerabilities lead to harmful consequences such as authentication bypassing and leakage of sensitive personal information. It is probably the most common Website vulnerability found today. According to web Cohort report almost 92% of web applications are subjected to some type of attack, among them 60% are

SQLI attacks. Tools such as firewalls and Intrusion Detection Systems (IDSs) are ineffective against SQLIAs, because ports which are open in firewalls for regular web traffic in the application level are used to perform SQLIAs. SQL injection vulnerabilities have been described as one of the most serious threats for Web applications. Many techniques have been proposed to detect SQLI attacks. These include input character filtering or input validation [2],hybrid encryption [3], randomization of SQL keywords [4], translation and validation[5],statement sequence digest[6], semantic comparison[7] , removal of attributes and comparison[8] etc. However, all these approaches do not cover up all known SQL injection attacks and also cannot be implemented in all platforms.

Above mentioned approaches do not detect of SQLI attacks by measuring complexity of the query. As a result, most of the approaches work well for known malicious inputs and may not detect unknown attacks. Our proposed solution is based on the fact that query with malicious input will change the complexity of the query. Thus, measuring a query complexity statically and observing any deviation at runtime should provide us an indication of the occurrence of an SQLI attack.

This motivation leads us to a technique to detect SQLI based on complexity of query. Information theory is a widely used concept to measure the complexity of real world phenomenon and has been applied to tackle many network security related problems.

In this paper, we present an information-theoretic approach to detect SQLI attacks. Proposed system works both on client and server side. Client side implements a filter program that checks the length and data type of the submitted variables, and detect the injection-sensitive characters and keywords Client side plays preliminary examination and gives warning. Server side works in two phases training and detection. Entropy of each query which represents complexity of the query in the application is calculated statically in training phase and again dynamically when query is submitted. Message authentication algorithm (MAC) is applied on both static and dynamic entropy. A dynamic query with attack inputs alters its intended structure and hence the entropy level changes significantly which will change the corresponding MAC value. In contrast, a dynamic query with benign inputs does not result in any changes of the MAC values. Attack is detected by comparing MAC values generated statically and dynamically. Change in values signals SQL injection. Existing system works mainly on server side only by including client side we can save on network traffic and can avoid round trips to the server. Simple attacks or a typing mistake by user would be stopped then and there at client side. Proposed system provides additional security by adding MAC (Message authentication Code) in the system which provides integrity and authentication. It also secures the database which stores static entropy.

II. RELATED WORK

The authors in [1] propose a system based on information theory. It measures query’s entropy statically using token probability distribution of a query. During execution compute the complexity to identify any changes in entropy measured earlier. Dynamic query with attack inputs alters its intended structure and hence the entropy level changes. Based on a summing up report authors in [2] checks for special characters in the submitted query. If a restricted character is found query is blocked else query is executed. Proposed method in [3] is an authentication scheme using hybrid encryption. Query generated by using encrypted user name and password is encrypted by applying RSA. In verification query is decrypted using server’s private key and username and password are verified. Finally decrypted user name and password are checked. Proposed scheme in [4] is based on randomization and is used to convert the input into a cipher text. Each input character is given one of four random values from a sample lookup table. Based on the next input character, one of these four values is substituted for a given character. Encrypted values are checked with database. The method proposed by the authors in [5] is based on translation and validation. It retrieves information from SQL database to produce a corresponding LDAP database. Authors in [6] implement a technique which is based on statement sequence digest (SSD).SSD is a profile of SQL statement which can be calculated using MD5, SHA etc. algorithms. SQL injection attack is detected by comparing SSD calculated statically and dynamically. Proposed scheme in [7] is based on semantic comparison. The semantic comparison is done by comparing the syntax tree structure of a query. If the syntax trees at training and run time are equivalent. Then the queries are inducing equivalent semantic actions and query is a safe query else attack is detected. Authors in [8] propose a simple method that removes attributes from SQL query. Attack is detected by user is executed both in LDAP and SQL server. If the result returned from both databases are inconsistent SQL injection attack is detected.

III. BACKGROUND OF SQL INJECTION ATTACK

Web-based programs store and retrieve sensitive information from databases by executing SQL queries, which include user supplied inputs that are not sanitized properly before being included in dynamically generated queries. As a result, the intended structures of dynamic queries get altered and result in SQL Injection (SQLI) attacks. The consequence of SQLI attacks could be devastating. Altered queries due to SQLI attacks might (i) add, delete or modify data (ii) run additional queries, (iii) insert, update, or delete new tables, and (vi) create or delete arbitrary tables.

A. Injection Mechanisms and Intention

An attacker can insert SQL command in to user input field in many different ways like injection through user input, Injection through cookies, Injection through server variables, Second-order injection

As classified by Halfond et al attacks can be characterized based on the goal, or intent of the attack [11]-Identifying injectable parameters, performing database finger-printing, determining database schema, extracting data, adding or modifying data, performing denial of service, evading detection, bypassing authentication, executing remote commands, performing privilege escalation

B. Types of SQL injection Attack

SQL injection attack can be categorized as tautologies, illegal/logically incorrect queries, union query, piggybacked, inference, alternate encodings and stored procedure.

Ⅳ. INFORMATION THEORY BASED FRAMEWORK FOR SQL INJECTION ATTACK DETECTION

We have implemented a Detection Block model for SQL injection attack detection. Our model conducts two checks both on the Client Side and Server Side.

A. Client

According to a summing-up report[3], the sensitive characters/keywords of the SQL injection attack include: "exec", "xp_", "sp_", "declare", "Union ","+","//","..."," ;",""","-- ","%"," 0x ", which are not to be bound to used in the general structure query statement. A filter function is set to filter these characters before the parameters are uploaded in the query. Client side implements a filter program that checks the length and data type of the submitted variables and detect the injection-sensitive characters and keywords. Figure 1 illustrates the client side framework. Client side plays preliminary examination and gives warning. Since all the people who have submitted the illegal characters could be SQL injection attackers. However, considering that the illegal characters may be submitted by user due to typing mistake, for which the check on the Client Side only gives a friendly error message and suspends submission. When user submits a request first it is checked for size if size is less than the specified limit.

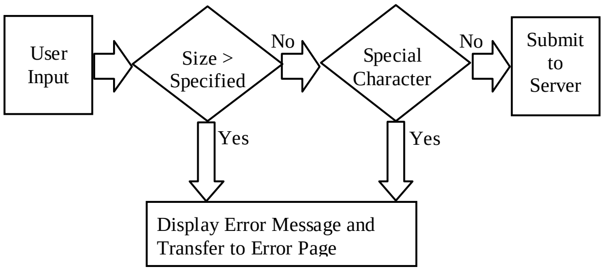

Fig. 1. Client side framework

Then it is checked for any forbidden special characters. If it passes both the tests request is submitted to server, else an error message is displayed and request is not submitted to server.

Client side does not provide solution to all the attacks, but provides basic security to prevent illegal attacks. It is also helpful in decreasing network traffic. Advantage of client side is that it reduces CPU cycles since it avoids a number of round trips to the server. Limitations of client side are firstly limiting the size of input and restricting the use of special characters cannot be imposed on user in all applications. Secondly the protection provided by client side scripts can be easily bypassed. Client side scripting can easily be bypassed so server side is required for complete security.

B. Server

On server side we implement entropy computational model it measures the complexity of a given query. Entropy is defined as the expected value of the information contained in a message. It is an indicator of the complexity of the query written by a programmer.

Server side works in two phases training and detection phase. In training phase we identify static SQL queries present in the program. Entropy of each query is calculated which is based on complexity of the query. Entropy is derived from probability distribution of token's present in the query [1]. Next we apply Message authentication Code (MAC) on entropy calculated from first step, application of MAC enhances the security by safeguarding the entropy value. Value of MAC calculated here is stored in a database.

Detection phase begins with a database query invocation. When a request is submitted a dynamic SQL query is invoked. The generated dynamic query is analysed to compute the entropy and MAC is applied on calculated entropy. The approach then relies on the assumption that dynamic queries with attack inputs result in increased or decreased level of entropy. In contrast, a dynamic query with benign inputs does not result in any change of entropy value. A dynamic query with attack inputs alters its intended structure and hence the entropy level changes significantly which will change the corresponding MAC. In contrast, a dynamic query with benign inputs does not result in any changes of the entropy values and thus MAC remains unchanged. Thus by comparing MAC calculated before program deployment and MAC calculated after query invocation will detect attack .Change in value of MAC signals that entropy has changed, while entropy will change only if tokens probability distribution will change. The approach can reveal several unknown vulnerabilities not reported before because it is not based on any particular attack input.

Figure 2 illustrates server side framework. Functionality of each module in server side framework (Refer figure 2) is explained below:

Training Phase

Program Source code and Server Script Analyzer

During training phase first program source code is analyzed to find all static the queries in the application.

Static Entropy Calculator

After all the queries are revealed entropy of each query is calculated which is based on probability distribution of tokens present in the query. Entropy actually represents query’s complexity. This entropy should remain intact and any alteration indicates the presence of malicious inputs. The entropy (denoted as H) [2] of all the queries present in the program can be computed as follows:

The entropy represents the average amount of information required to represent queries in the application.

Q= {q1 ,q2 ,q3……………qn}be set of queries in the application

Ώ={x1,x2,x3............................xl}set of all tokens present in a query.

P(x) probability of a token x in query q

Entropy of the query is represented by:

$$H(q) = H(x_1, x_2, x_{3,\ldots\ldots\ldots\ldots\ldots,} x_n) = \sum_{i=1}^{n} P(x_i) * \log P(x_i)$$

Entropy calculated here in training phase is represented as Static entropy.

MAC

A message authentication code (MAC) is a cryptographic checksum on data that uses a session key to detect modifications of the data. It is a small fixed-size block of data that is generated based on a message M of variable length using secret key K[9] as follows.

MAC = C (K, M)

Applying MAC on entropy provides us authentication and integrity. MAC is applied on entropy calculated from previous step. If entropy is stored in database there is a possibility of attack on entropy database, which if attacked can compromise the entire security. By applying MAC on entropy we are enhancing the security. If attacker attacks MAC database then also entropy can’t be reveled from it because key is not known to attacker.

Proposed model implements MAC as follows:

1. Retrieve static entropy (E) from entropy calculator.
2. Retrieve key (K) form key database.
3. Take hash of entropy and key, we get static MAC.

MAC (K, E) = H ((K // E)

Static MAC (represented as SMAC) calculated here is stored in database to be compared later.

Detection Phase

Query Invocation

The detection phase begins when a query is fired for the application. At runtime when query is invoked necessary elements are calculated as stated below.

Dynamic Entropy Calculator

It works in the same manner as static entropy calculator. The entropy calculated here is represented as dynamic entropy.

MAC

It works in the same manner as MAC in training phase. MAC calculated over here is represented as dynamic MAC (DMAC).

Comparison

Ideally, the MAC of the dynamic query should match with the pre-recorded MAC in the database learned from the training phase i.e. SMAC. Static MAC and dynamic MAC are compared here. If SMAC is same as DMAC there is no injection and query is genuine. If DMAC is not equal to SMAC that means query is modified, SQL injection is detected.

Execute

If SMAC and DMAC are same submitted query is genuine and request is submitted to server. Query is allowed to execute and result is returned to the sender.

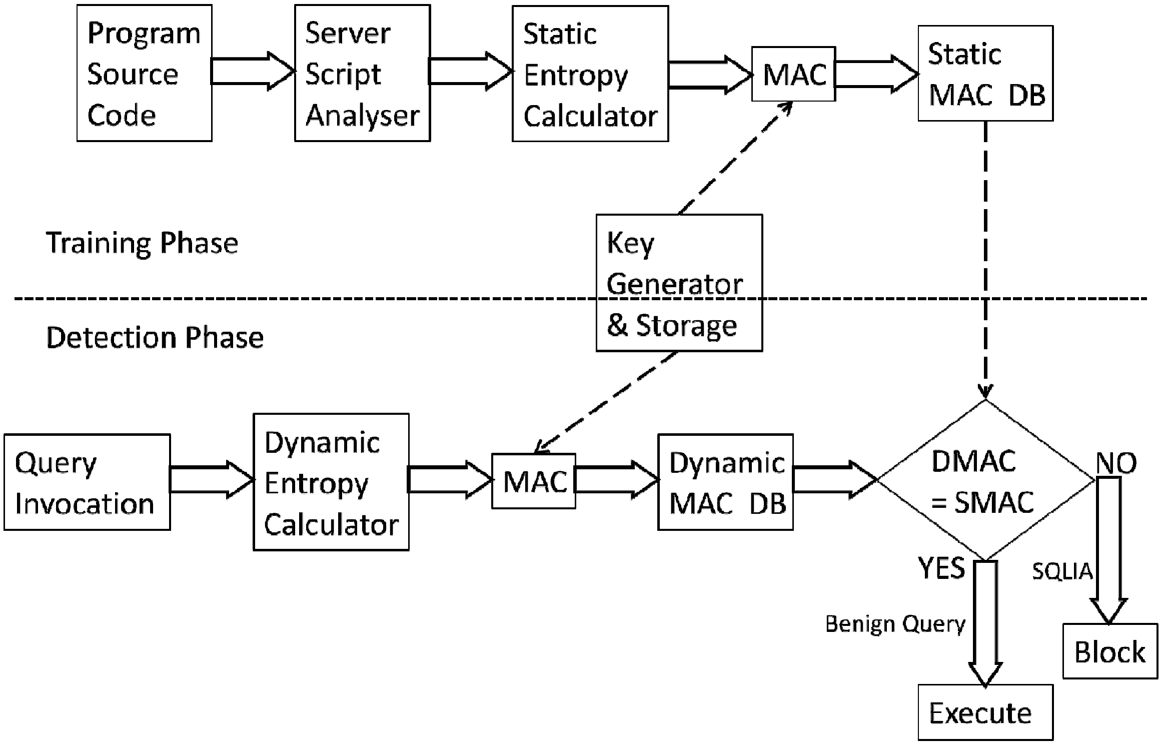

Fig. II Server Side Framework

Block

If DMAC and SMAC are not same, SQL injection is detected. The query is blocked i.e. not executed and an entry is made in blocked ip's table in database. For this danger signal, the server will record the IP address into a database for future reference, and will transfer the request to a error message page. . Blocking of ip address will not allow any input from that ip address in future.

Key generation and storage

This module will generate a random key every time. Generated random key is stored in database. Key value will be fetched from here for MAC calculation.

C. Algorithm

Client side:

- Input text
- Check for length of input submitted

- Check for injection sensitive characters and keywords as specified.
- If found sensitive character is found or size greater than specified return error message.
- Else submit query to server.

Server side:

- Analyze program source code to find all queries.
- For all queries in application calculate entropy which is called static entropy.
- Apply MAC (Message authentication code) on static query we get static MAC (SMAC).
- SMAC is stored in db.
- At Runtime when query is invoked. Dynamic entropy is calculated.
- Apply MAC (Message authentication code) on dynamic entropy we get dynamic MAC (DMAC).
- Compare DMAC and SMAC.
- If they are equal query is genuine.
- Else attack is detected, query is not executed. ip address is blocked and recorded

D. Advantages

Proposed scheme has various advantages as enlisted below.

- Client side reduces CPU cycles since it avoids a number of round trips to the server.
- Our technique can detect all known SQLI attacks.
- It can reveal several unknown vulnerabilities as it does not rely on the specific type of attack inputs.
- It does not require tainted data flow analysis or complex static analysis.
- It can be applied for a wide variety of scripting languages
- Application of MAC provide additional layer of security.

IV. Implementation and Evaluation

We implement a detection tool for testing SQLI. The tool accepts .net files and detects attack both on client and server side. Client side implements a java script file which filters all SQL injection sensitive characters based on a summing up report which specifying all SQL injection sensitive characters. Server side computes entropy for each queries present in a program. The entropy information is instrumented in program code and compared during actual program execution time. We use split function in .net for parsing and to count the tokens in a query.

We perform the evaluation in the following two steps.

Client side: Checks for all SQL injection sensitive characters.

Server side: First, we identify SQL queries in each web page. Then, we compute the entropy of the queries apply MAC on entropy and store the MAC in database. In the second stage, we run the programs by deploying them in a web server. Then, we visit the corresponding pages and supply malicious inputs in the input field of web application. We notice the instrumented code with entropy information successfully stops the malicious query execution and logs a warning.

V. Experimental Result

For testing our application we have considered all types of SQL injection attack. Response time for detection is very fast. Table I illustrates type of SQL injection attacks which are detected blocked and ip address of attacker is logged in database.

Table I
Result for All Type of Attacks

Attack Type	Detected	Blocked	Logged
Tautology	Yes	Yes	Yes
Piggybacked	Yes	Yes	Yes
Union	Yes	Yes	Yes
Alternate encoding	Yes	Yes	Yes
Illegal /logically incorrect	Yes	Yes	Yes
Blind	Yes	Yes	Yes
Timing	Yes	Yes	Yes

In our testing, we notice client side can detect various attacks. Attack input is not submitted to server and is stopped at client side only. Table II illustrates result attained from client side for different attack input. But we understand client side scripting can be easily bypassed. In our model if an attacker bypasses client side filter attack will be detected and blocked at server side. At server side all types of SQL injection attacks are detected, blocked and their ip addresses are stored in database for future references.

Table II
Results from client side

Attack	Input	Result	Detected
Piggy backed	Admin; select * from table --;	; is not valid	Detected
Tautology	‘ or 1=1--;	= is not valid.	Detected
Alternat encoding	exec(char(0x73687574 646f776e))--	exec is not valid	Detected
Union query	‘;union select usr from test--	select is not valid.	Detected
Illegal /incorrect	convert (int,(select usr from test where usr =’u’))	Convert is not valid	Detected
Blind	admin; or 1=1--; admin ; or 1=2--;	= is not valid.	Detected

Table III illustrates result from server side. When an attack takes place it is detected, that attack input is not executed and the ip address of attacker is stored in database and is blocked. Table three shows result from server side for different attack type. Blocking of ip address will not allow any input from that ip address in future. All the malicious query inputs have been blocked by the framework. Thus, the false negative rate in our evaluation is zero.

Table III
Results from server side

Attack Type	Input 1	Input2	Detected	Blocked	Logged
Piggy backed	Admin; select * from table --;	Drop database Diksha --;	Yes	Yes	Yes
Tautology	‘ or 1=1--;	‘ or a=a--;	Yes	Yes	Yes
Alternate encoding	exec(char(0x73687574646f776e))--	exec(char(0x579757578889341e))--	Yes	Yes	Yes
Union query	‘;union select usr from test--	‘; union select entropy from blocked_entropy	Yes	Yes	Yes
Illegal /incorrect	convert (int,(select usr from test where usr =’u’))	convert(int,(select usr from test))	Yes	Yes	Yes
Blind	admin; or 1=1--; admin ; or 1=2--;	‘ or 2=2--; ‘ or 2=3--;	Yes	Yes	Yes
Timing	‘ and ASCII(SUBSTRING((select top 1 name from test),1,1)) > X WAITFOR 5 --	‘ and ASCII(SUBSTRING((select top 1 name from bl_entropy),1,1)) > X WAITFOR 10 --	Yes	Yes	Yes

VI. CONCLUSION

SQL injection is defined as one of the most serious and common web security threat that needs attention to provide secure web applications. Exploitations of SQLI vulnerabilities result in compromise of database, which is a valuable asset of an organization. Thus, SQLI mitigation needs to be considered seriously. Our model applies concept of information theory for attack detection. Entropy is defined as information content of a query written by a programmer which should remain intact. When a malicious input alters the static nature of the query, the complexity value changes. We apply MAC on entropy; we compare the statically computed MAC with that of dynamically computed MAC. The deviation indicates the presence of SQLI in a query. The prevention and block model of SQL injection attack mentioned in this paper checks the legality based on the information submitted, conducts two checks both on the Client Side and Server Side, and as long as any of the two checks does not pass, the information submitted will not be executed at the server.

Future Scope

Currently, our approach does not address the SQLI in stored procedures as it requires our approach to be extended at the database script level. Our future work includes extending the model to stored procedures. We also plan to apply our developed model for detecting other web-based attacks such as cross-site scripting. Our thanks to the experts who have contributed towards study of the SQL injection. We sincerely thank our colleagues and friends. This paper would have been uncertain without the help and guidance of my guide.

REFERENCES

1. Hossain Shahriar, Mohammed Zulkernine, “Information Theoretic Detection of SQL Injection Attacks” Proceedings of 14th International Symposium on High Assurance System Engineering, 2012.
2. Qian XUE, Peng HE, “On Defense and Detection of SQL SERVER Injection Attack”. Proceedings of International Conference on Security Systems, 978-1-4244-6252-0/11/ IEEE, 2011, pg 324-330.
3. Indrani Balasundaram, E.Ramaraj “An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption (PSQLIAHBE” (ISSN 1450-216X Vol.53 No.3 (2011), pp.359-368)

4. Srinivas Avireddy, Varalakshmi Perumal, Narayan Gowraj, Ram Srivatsa Kannan, Prashanth" Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection" Proceedings of 11th International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2012, p1327-1335

5. Kai-Xiang Zhang, Chia-Jun Lin, Shih-Jen Chen, Yanling Hwang" TransSQL: A Translation and Validation-based Solution for SQL-Injection Attacks" Proceedings of First International Conference on Robot, Vision and Signal Processing, IEEE, 2011, p248-252.

6. Baohua Huang, Tongyi Xie, Yan Ma "Anti SQL Injection with Statements Sequence Digest" National Science Foundation of China, Scientific Research and Development Plan of Nanning City (No. 10876012), IEEE 2012

7. Sruthy Mamadhan, Manesh T, Varghese Paul" SQLStor: Blockage of Stored Procedure SQL Injection Attack Using Dynamic Query Structure Validation" (No. 978-1-4673-5119-5/12/$31.00c) IEEE, 2012, p240-246

8. Jeom-Goo Kim"Injection Attack Detection using the Removal of SQL Query Attribute Values" 978-1-4244-9224-4/11/$26.00 ©2011 IEEE

9. Jueneman, R. R., Matyas, S. M., and Meyer, C. H., "Message Authentication", IEEE Communication, Vol 23, No. 9, 1985, pp 29-40.

10. Rahul Johari, Pankaj Sharma" A Survey On Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection" Proceedings of International Conference on Communication Systems and Network Technologies, IEEE, 2012, p453-459

11. W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," Proceedings of the International Symposium on Secure Software Engineering (ISSSE 2006), Mar. 2006

12. The Open Web Application Security Project (OWASP), Available: https://www.owasp.org/index.php/Top_10_2013-Top_10