Using Container Architecture to Detect Intrusion for Multitier Web Application

Manoj E. Patil
Associate Professor
SSBT's COE,
Bambhori, Jalgaon

Rakesh D. More
Student
SSBT's COE,
Bambhori, Jalgaon

ABSTRACT

An intrusion detection system is a computer-based information system designed to collect information about malicious activities in a set of targeted IT resources, analyze the information, and respond according to a predefined security policy. The most common computer intrusion detection systems detect signatures of known attacks by searching for attack-specific keywords in network traffic. Intrusion-detection systems aim at detecting attacks against computer systems and networks or in general, against information systems. This strategy is mainly focus on to detect intrusion in multitier web applications. Multitier web application include two ends that is front end as well as back end of the applications. The front end include web server which can responsible to run the application and gives that output to back end i.e. file server. This strategy is useful to identify the intrusion at both front end and back end of web application.

General Terms

Intrusion Detection System, Multitier Architecture, Pattern Mapping.

Keywords

Container Architecture, Session ID.

1. INTRODUCTION

The attention of attackers has shifted from attacking the front end to exploiting vulnerabilities of the web applications [1], [2], [3] in order to corrupt the back end database system [4] (e.g., SQL injection attacks [5], [6]). However, there is very little work being performed on multi-tier Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions. In such multi-tier architectures, the back-end database server is often protected behind a firewall while the web servers are remotely accessible over the Internet. Unfortunately, though they are protected from direct remote attacks, the back-end systems are susceptible to attacks.

Intrusion detection systems have been widely used to protect multitier web services, such as to detect known attacks by matching misused traffic patterns or signatures [7-10]. Individually, the web IDS and the database IDS can detect abnormal network traffic sent to either of them. However, these IDSs cannot detect cases wherein normal traffic is used to attack the web server and the database server. For example, if an attacker with non admin privileges can log in to a web server using normal-user access credentials, he/she can find a way to issue a privileged database query by exploiting vulnerabilities in the web server. Neither the web IDS nor the database IDS would detect this type of attack since the web IDS would merely see typical user login traffic and the database IDS would see only the normal traffic of a privileged user. This type of attack can be readily detected if the database IDS can identify that a privileged request from the web server is not associated with user-privileged access. Unfortunately, within the current multithreaded web server architecture, it is not feasible to detect or profile such causal mapping between web server traffic and DB server traffic since traffic cannot be clearly attributed to user sessions.

In this approach [11], it presents container based approach as shown in Fig 1.1 which is used to detect attacks in multi-tier web services. This approach can create normality models of isolated user sessions that include both the web front-end (HTTP) and back-end (File or SQL) network transactions. There is use of the container ID to accurately associate the web request with the subsequent DB queries. Thus, this guarding can build a causal mapping profile by taking both the web server and DB traffic into account.

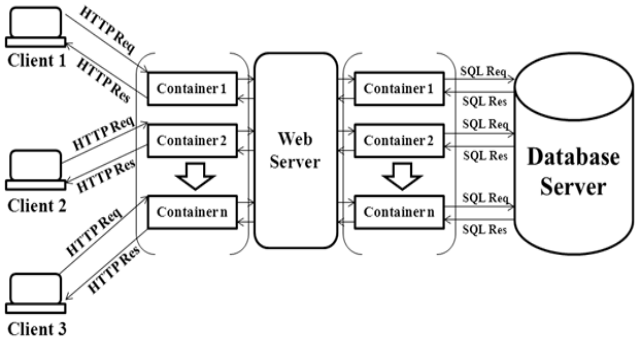

Fig 1.1: Container Architecture

In addition to this static website case, there are web services that permit persistent back-end data modifications. These services, which we call dynamic, allow HTTP requests to include parameters that are variable and depend on user input. Therefore, the ability to model the causal relationship between the front end and back end is not always deterministic and depends primarily upon the application logic. For instance, the backend queries can vary based on the value of the parameters passed in the HTTP requests and the previous application state. Sometimes, the same application's primitive functionality (i.e., accessing a table) can be triggered by many

different web pages. Therefore, the resulting mapping between web and database requests can range from one to many, depending on the value of the parameters passed in the web request.

To address this challenge while building a mapping model for dynamic web pages, I will first generate an individual training model for the basic operations provided by the web services.

2. PROPOSED WORK

The breakdown structure mainly focuses on following areas –

1. Module 1: Responsible for user control; restricts unauthorized users.
2. Module 2: Creates and monitors user session.
3. Module 3: Checks and filters users query.
4. Module 4: Maps HTTP queries with equivalent SQL queries.
5. Module 5: Generates a log showing log of attacks.

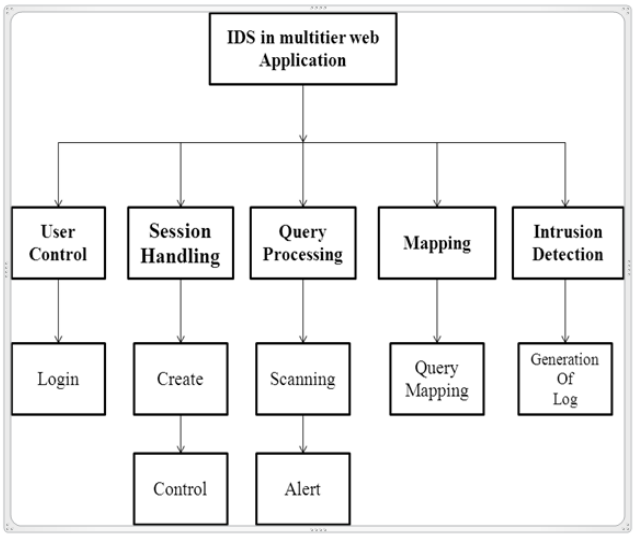

Fig 2.1 Work Breakdown Structure

2.1 Module 1: User Control

Input: Registration details with username and password as input.

Output: Successful or unsuccessful login.

Algorithm:

1. New user will fill a registration form.
2. Get user name and password.
3. Logs into the system.
4. Starts his new session.
5. After completion of session user logs out.

The above algorithm shows how exactly the login module will provide security to the entire system to prevent unauthorized access of system. If any new user is there, wants to enter into the system then he has to fill a new user registration form. In that registration form user has to fill his personal information along with his username and password. When user clicks on save button all his information get inserted into the database.

Now this user has its own username and password. By clicking "click here to login" link he will redirect to login page. Here user will login into the system with his personal username and password. If user enters correct username and password as filled in the registration form; a "Login Successful" message will displays else if he enters wrong username or password then the system will displays "Invalid username or password message". Thus this module gives security and provides user control to the system.

2.2 Module 2: Session Handling

Input: HTTP query r and SQL query q.

Output: Session id for r and q in the sets ARr and AQq respectively.

Algorithm:

1. For each session separated traffic Ti do
2. Get different HTTP requests 'r' and DB queries 'q' in this session for each different r do
3. If r is a request to static file then
4. Add r into set EQS (Empty Query Set)
5. Else
6. If r is not in set REQ then
7. Add r into REQ
8. Append session ID i to the set ARr with r as the key
9. For each different q do
10. If q is not in set SQL then
11. Add q into SQL
12. Append session ID i to the set AQq with q as the key

Session handling module is responsible for assigning correct and unique ID to the HTTP request and equivalent SQL request. If input HTTP query is for any static data/file; means if the requested content is available at web server itself then r is added into Empty Query Set. This type of query doesn't get any kind of ID. If r is not in the set of REQ means the input query is new of arrives first time into

the system then r is added into REQ i.e. request query set. By taking r as a key session ID i is appended to the set of ARr.

Similarly for each SQL query if q is not into the set of SQL query then it is added into the SQL set. Same as above by taking q as key session ID i is appended to the set of AQq.

2.3 Module 3: Query Processing

Input: HTTP query r and SQL query q.

Output: Insertion of queries into different Query Sets.

Algorithm:

1. For each session separated traffic Ti do
2. Get different HTTP requests 'r' and DB queries 'q' in this session for each different r do
3. If r is a request to static file then
4. Add r into set EQS (Empty Query Set)
5. Else
6. If r is not in set REQ then
7. Add r into REQ
8. For each different q do
9. If q is not in set SQL then
10. Add q into SQL

Query Processing is the module for assigning adding different requests into proper sets of query. If input HTTP query is for any static data/file; means if the requested content is available at web server itself then r is added into EQS (Empty Query Set). If r is not in the set of REQ means the input query is new of arrives first time into the system then r is added into REQ i.e. request query set. Similarly for each SQL query if q is not into the set of SQL query then it is added into the SQL set.

2.4 Module 4: Query Mapping

Input: Set of ARr, Set of AQq and Cardinality t.

Output: HTTP query gets mapped with equivalent SQL query.

Algorithm:

1. For each distinct HTTP request r in REQ do
2. For each distinct DB query q in SQL do
3. Compare the set ARr with the set AQq
4. If ARr =AQq and Cardinality(ARr) > t then
5. Found a Deterministic mapping from r to q
6. Add q into mapping model set MSr of r
7. Mark q in set SQL
8. Else
9. Need more training sessions
10. Return False
11. For each DB query q in SQL do
12. If q is not marked then
13. Add q into set NMR (No Matched Request)
14. For each HTTP request r in REQ do
15. If r has no deterministic mapping model then
16. Add r into set EQS (Empty Query Set)
17. Return True

The user request comes to the web server in the form of HTTP request and a equivalent SQL query is generated by web server. Query mapping module maps the HTTP query with the equivalent SQL query. As we have seen the working of session handling module and query processing module. Mapping module use the output generated by these modules. A HTTP query with its ID stored in ARr set and a SQL query with its ID stored in AQq set; both are matched with each other if both ID are equal and Cardinality of ARr is greater than 1 then there is a deterministic map is found. q is then added into the matched set query and it is also marked in the set of SQL queries. After performing all training data sets if any query from the set q is not marked then that q is moved to the NMR (No Matched Request) set. Similarly for every HTTP request r; if r has no deterministic mapping then that r is added into the EQS (Empty Query Set).

2.5 Module 5: Intrusion Detection

Input: HTTP query r and SQL query q.

Output: Log showing malicious query/attacks.

Algorithm:

1. If the rule for the request is Deterministic Mapping r -> Q ($Q \neq \Phi$), we test whether Q is a subset of a query set of the session. If so, this request is valid, and we mark the queries in Q. Otherwise, a violation is detected and considered to be abnormal, and the session will be marked as suspicious.

2. If the rule is Empty Query Set r -> Φ, then the request is not considered to be abnormal, and we do not mark any database queries. No intrusion will be reported.

3. For the remaining unmarked database queries, we check to see if they are in the set NMR. If so, we mark the query as such.

4. Any untested web request or unmarked database query is considered to be abnormal. If either exists within a session, then that session will be marked as suspicious.

The intrusion detection module checks every r and q with the mapping model and then decides that whether it is from a general user or attacker. If there is mapping found between r and q then it is a considered as valid session, otherwise it have to checks other query sets. If query r is found in Empty Query

Set then it not considered as abnormal and no intrusion will be reported. For remaining unmark queries we check to see if they are in the set NMR. If so, we mark the query as such. Any query that comes directly to the database without any mapping then that session is considered as abnormal.

3. FUTURE SCOPE

It is possible to make some future modifications into the system; which can be make existing system more efficient. The Intrusion detection systems can be installing on wide range of machines having different operating system and platforms. The query processing mechanism can be made simpler by applying natural language processing (NLP); so as to convert simple English sentences into SQL queries.

Since the this system works on the basis of signature; each activity of intrusions is to be memorized by the system previously. New attacks are often unrecognizable by popular IDS. So there is continuous race going in between new attacks and detection systems have been a challenge. Nowadays Intrusion detection systems also work on the wireless networks. The latest wireless devices come with its own set of protocols for communication that break the traditional OSI layer model. So IDS must learn new communication patterns of the latest wireless technology.

4. CONCLUSION

This system is an intrusion detection system that builds normality model for multitier web applications. Unlike previous approaches this approach forms container-based IDS with multiple input streams to produce alerts. There will be lightweight virtualization technique to assign session ID to a dedicated container which is nothing but isolated virtual computing environment. Furthermore, there will specific detection of attacks such as Privilege Escalation Attack, Hijack Future Session Attack, SQL Injection Attack and Direct DB Attack. Log at IDS will show the details of these attacks. Also the requests which violate the normality model that will be treat as an intruder. This approach will be attempted to static and dynamic web requests with the back end file system and database queries.

5. REFERENCES

[1] "Five Common Web Application Vulnerabilities," http://www. symantec.com/connect/articles/five-common-web-applicationvulnerabilities, 2011.

[2] "Common Vulnerabilities and Exposures," http://www.cve. mitre. org/, 2011.

[3] SANS, "The Top Cyber Security Risks," http://www.sans.org/ top-cyber-security-risks/, 2011.

[4]A.Schulman,"Top10DatabaseAttacks,"http://www.bcs.org /server.php?show=ConWebDoc.8852, 2011.

[5] C. Anley, "Advanced Sql Injection in Sql Server Applications," technical report, Next Generation Security Software, Ltd., 2002.

[6] Y. Shin, L. Williams, and T. Xie, "SQLUnitgen: Test Case Generation for SQL Injection Detection," technical report, Dept. of Computer Science, North Carolina State Univ., 2006.

[7] J. Newsome, B. Karp, and D.X. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms," Proc. IEEE Symp. Security and Privacy, 2005.

[8] H.-A. Kim and B. Karp, "Autograph: Toward Automated Distributed Worm Signature Detection," Proc. USENIX Security Symp., 2004.

[9] Liang and Sekar, "Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers," SIGSAC: Proc. 12th ACM Conf. Computer and Comm. Security, 2005.

[10] B.I.A. Barry and H.A. Chan, "Syntax, and Semantics-Based Signature Database for Hybrid Intrusion Detection Systems", Security and Comm. Networks, vol. 2, no. 6, pp. 457-475, 2009.

[11] Meixing Le, Angelos Stavrou, Brent ByungHoon Kang, "DoubleGuard: Detecting Intrusions in Multitier Web Applications", IEEE transactions on dependable and secure computing, vol. 9, no. 4, july/august 2012.