Intrusion Detection and Prevention System in Multitier Web Application Using Double guard

Jadhav Priya B.[1], Hadawale Shwetali N.[2], Pinjarkar N. R.[3]
UG students, Department of Computer SVCET Rajuri Pune-412411[1, 2]
Asst. Professor, Department of Computer SVCET Rajuri Pune-412411[3]
adhav.neha87@gmail.com, smitahadawale448@gmail.com[2] , nilesh.pinjarkar81@gmail.com[3]

***Abstract*- Web application is an application that is accessed over a network such as the Internet. They are increasingly used for critical services, In order to adopt with increase in demand and data complexity web application are moved to multi-tier Design. As web servers must be publicly available around the clock the server is an easy target for outside intruders. Thus web applications are become a popular and valuable target for security attacks. These attacks have recently become more diverse and attention of an attacker have been shifted from attacking the front-end and exploiting vulnerabilities of the web applications in order to corrupt the back-end database system. In order to penetrate their targets, attackers may exploit well known service vulnerabilities. To protect multi-tier web applications several intrusion detection systems has been proposed. An intrusion detection system (IDS) is used to detect potential violations in database security. In every database, some of the attributes are considered more sensitive to malicious modifications compared to others.**

***Index Terms*-Visualization, Multi-tier, IDS, Anomaly Detection, Container**

I. INTRODUCTION

To protect multi-tiered web services, Intrusion detection systems (IDS) have been widely used to detect known attacks by matching misused traffic patterns or signatures [1]. In the existing system we require different IDS one for web server and another for database server. Two IDSes required so we need to create two IDSes with different prevention measure first IDS that contains prevention measure related to web server so attack should not happen on web server but some time attack happen on database server by passing web server so for that reason need to create another IDS with prevention measure

related to database server attack. We want to avoid creating two IDS so we are creating one DoubleGuard system that act as IDS and prevent both side of attack. Attack may be on web server or database server. DoubleGuard actually invented by Le, Stavrou and Kung[1]. Most of the IDS examine the attack individually on web server and database server. In order to protect multi-tiered web services an efficient system call Intrusion Detection System is needed to detect attacks by mapping web request and SQL query, there is direct causal relationship between request received from the front end web server and those generated for the database backend.
Le, Stavrou and kung showed that this causality mode l can be generated accurately and without prior knowledge of web applications functionality. We implemented the DoubleGuard using sessions. It will allocate the isolated session for each user it is practical for most of the web applications. Static web sites has the controlled environment whereas dynamic website not. Dynamic web site allow persistent back end data modification through the HTTP requests to include the parameters that are variable and depend on the user input. Because of which the mapping between the web and the database rang from one to many as shown in the mapping model.

In the proposed system we are implementing Double Guard that handle both sides of attack.
Attack may be from static web site or dynamic web site. No need to create two different IDS for two different web site. Double Guard can handle both types of attack

Following tasks should be accomplished by DoubleGuard:

- It should prevent the damage that detected intrusion could cause
- it should mitigate the damage that detected intrusion could cause
- to discover the new attacks patterns
- Completeness: It should not fail to detect an intrusion. It is practically impossible because it is
- Fault tolerance : it should be resistant to attacks and its consequences
- Timeliness: DoubleGuard should perform the analysis as quickly as possible.

Apart from the functional requirements, DoubleGuard should satisfy the number of economical requirements, in particular case, cost.

- **c**ost of the product
- cost of additional computer resources needed
- cost of administration
- An importance of all this is oblivious. DoubleGuard should available not only to large enterprises, but also small enterprises, as well as private person.

II RELATED WORK

A network intrusion detection system can be classified into two types: signature detection and anomaly detection. Anomaly detection first requires the IDS to define the characterized the correct and acceptable static from dynamic behavior of the system. It is used to detect the abnormal behavior of the system. We first define the normal behavior of the system and create profile of the user. In early IDS system that use the independent IDS used. DoubleGuard use dependent IDS used. DoubleGuard use the container ID using this ID each user session is assigned to each id.

Our approach does not require input validation, source code validation and know the application logic. DoubleGuard uses the light weight virtualization to create and destroy the container by using the tool open VZ. We identify the causal relationship between web server request and database request. Our approach dynamically generate new containers and recycle the used ones.

CLAMP (Confidentiality to LAMP)[1] is an sensitive data leakage prevention technique even in the presence of attack. Clamp guarantees that user sensitive data can only be accessed by code running on the behalf of different user. Whereas DoubleGuard focuses on modeling the mapping

- Accuracy: It must not identify the legitimate action in system environment as anomaly or misuse like IDS
- Performance : DoubleGuard performance must be high enough to carry out the real time intrusion detection

impossible to have a global knowledge about past, present and future attacks. patterns between the HTTP request and the database queries to detect the malicious user session. CLAMP requires modification to the existing application code, and the Query Restrictor works as a proxy to mediate all database access requests. Moreover, resource requirements and overhead differ in order of magnitude: DoubleGuard uses process isolation whereas CLAMP requires platform virtualization, and CLAMP provides more coarse-grained isolation than DoubleGuard. However, DoubleGuard would be ineffective at detecting attacks if it were to use the coarse-grained isolation as used in CLAMP. Building the mapping model in DoubleGuard would require a large number of isolated web stack instances so that mapping patterns would appear across different session instances.

In addition, validating input is useful to detect or prevent SQL or XSS injection attacks [3], [6]. This is orthogonal to the DoubleGuard approach, which can utilize input validate on as an additional defense. However, we have found that Double-Guard can detect SQL injection attacks by taking the structures of web requests and database queries without looking into the values of input parameters (i.e., no input validation at the web server).

III. STATIC MODEL BUILDING ALGORITHM:

In this algorithm we are getting set of web request and gene rate SQL query according to web request. If user perform web request and for that web request SQL query is not generated then that web request mark as EQS (Empty Query Set) else generated SQL query and get result. If we got same result as expected up to threshold value then mapping is correct otherwise need more training sessions. In NMR (No match request) SQL query genera ted without web request from user but according to SQL query action will be performed. We have used the following static model building algorithm to create the static webpage.

Algorithm 1: Static Model Building Algorithm.
Require: Training Dataset, Threshold t
Ensure: The Mapping Model for static website

1: for each session separated traffic Ti do

2: Get different HTTP requests r and DB queries q in this session
3: for each different r do
4: if r is a request to static file then
5: Add r into set EQS
6: else
7: if r is not in set REQ then
8: Add r into REQ
9: Append session ID i to the set ARr with r as the key
10: for each different q do
11: if q is not in set SQL then
12: Add q into SQL
13: Append session ID i to the set AQq with q as the key
14: for each distinct HTTP request r in REQ do
15: for each distinct DB query q in SQL do
16: Compare the set ARr with the set AQq
17: if ARr = AQq and Cardinality(ARr) > t then
18: Found a Deterministic mapping from r to q
19: Add q into mapping model set MSr of r
20: Mark q in set SQL
21: else
22: Need more training sessions
23: return False
24: for each DB query q in SQL do
25 : if q is not marked then
26: Add q into set NMR
27: for each HTTP request r in REQ do
28: if r has no deterministic mapping model then
29: Add r into set EQS
30:return True .....

IV. THREAT MODEL AND SYSTEM ARCHITECTURE

We initially set up our threat model to include our assumptions and the types of attacks we are aiming to protect against. The attackers can bypass the webserver to directly attack the database server. We assume that the attacks can neither be detected nor prevented by the current webserver IDS, that attackers may take over the webserver after the attack, and that afterward they can obtain full control of the webserver to launch subsequent attacks. In addition, we are analyzing only network traffic that reaches the webserver and database. We assume that no attack would occur during the training phase and model building.

I.1 Architecture and Confinement

In our design, we make use of lightweight process containers, referred to as "containers," as ephemeral, disposable servers for client sessions. It is possible to initialize thousands of containers on a single physical machine, and these virtualized containers can be discarded, reverted, or quickly reinitialized to serve new sessions.

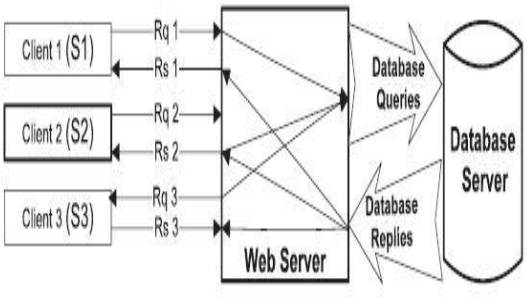

Fig. 1 Classic three-tier model.

The webserver acts as the front end, with the file and database servers as the content storage back end.

In the classic three-tier model database side, we are unable to tell which transaction corresponds to which client request. The communication between the webserver and the database server is not separated, and we can hardly understand the relationships among them.

I.2 Building the Normality Model

This container-based and session-separated webserver architecture not only enhances the security performances but also provides us with the isolated information flows that are separated in each container session. It allows us to identify the mapping between the webserver requests and the subsequent DB queries, and to utilize such a mapping model to detect abnormal behaviors on a session/client level.

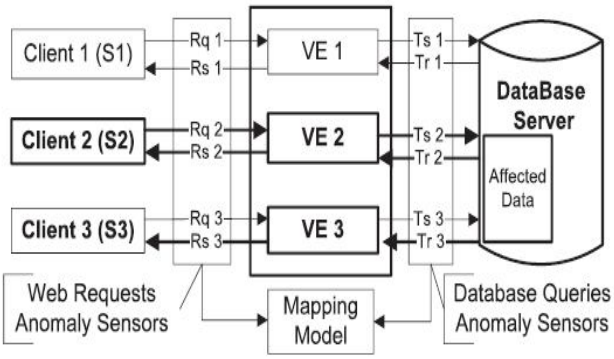

Fig.2. Webserver instances running in containers.

Once we build the mapping model, it can be used to detect abnormal behaviors. Both the web request and the database queries within each session should be in accordance with the model. If there exists any request or query that violates the normality model within a session, then the session will be treated as a possible attack.

I.3 Attack Scenarios

Our system is effective at capturing the following types of attacks:

I.3.1 Privilege Escalation Attack

This attack shows how the attacker can access the Admin's credentials and act as admin in the system. The attacker gets the admin's user id and password and gives command to web server to get the private database of user. Suppose, the attacker login into the web server as a normal user and trigger admin queries to obtain the administration data then this kind of attack can never be detected by the IDS or normal intrusion detection technique. But as our system is allocating the different sessions for different user we can easily detect this kind of attack.

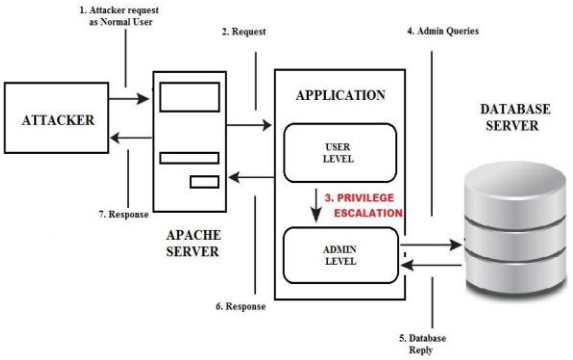

Fig.3. Privilege Escalation Attack

I.1.1 Hijack Future Session Attack

Figure shows how this attack attempted by the middle person/attacker. The third person accesses the username and password of normal user and misuse them. In banking, travelling, personal accounts these kinds of attacks are happened to get the personal information of normal user. But in my DoubleGuard technique this type of attack into possible. As every user is getting his/her personal session which no can access. So using this technique we can prevent this kind of attack.

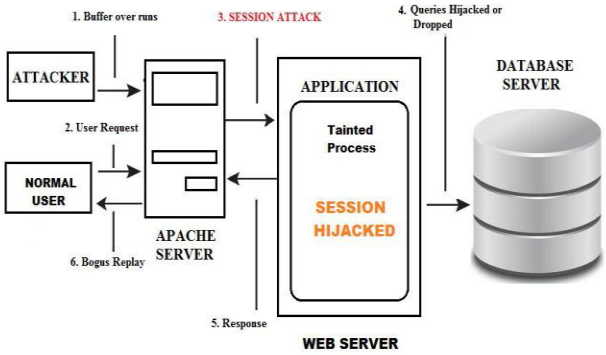

Fig.4.Hijack Future Session Attack

I.1.2 Injection Attack

Attacks such as SQL injection do not require compromising the webserver. Attackers can use existing vulnerabilities in the webserver logic to inject the data or string content that contains the exploits and then use the webserver to relay these exploits to attack the back-end database.

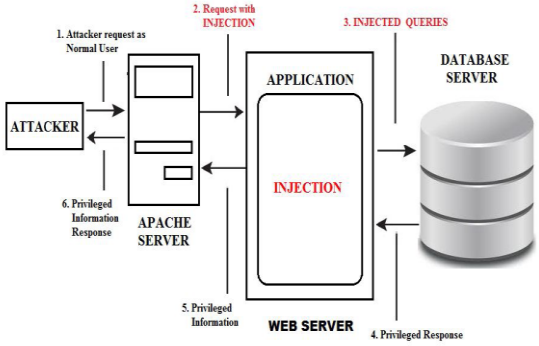

Fig.5.Injection Attack

I.1.3 Direct DB Attack

It is possible for an attacker to bypass the webserver or firewalls and connect directly to the database. An attacker could also have already taken over the webserver and be submitting such queries from the webserver without sending web requests. Without matched web requests for such queries, a webserver IDS could detect.

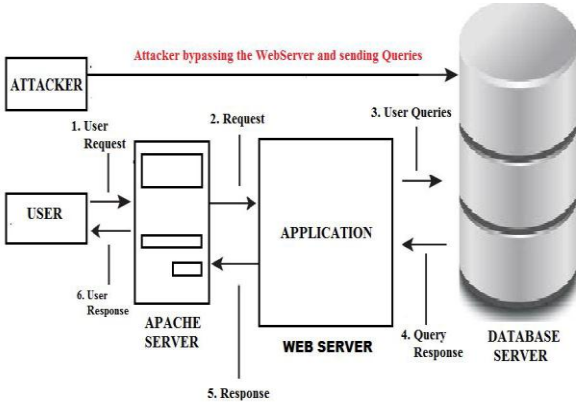

Fig.6.Direct DB Attack

V. IMPLEMENTATION

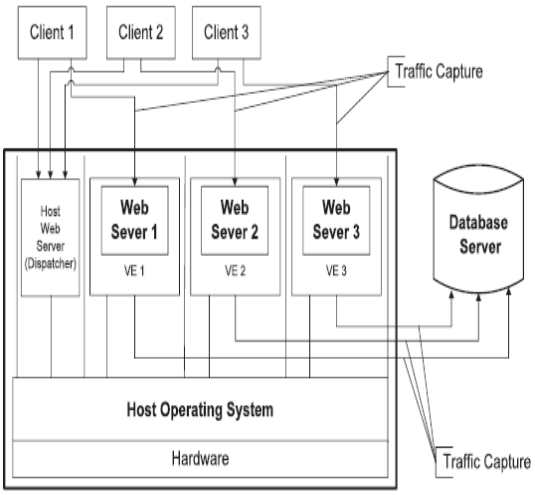

Fig.7. Implementation of the system

In our prototype, we chose to assign each user session into a different container; however, this was a design decision. For instance, we can assign a new container per each new IP address of the client. In our implementation, containers were recycled based on events or when sessions time out [3].

We were able to use the same session tracking mechanisms as implemented by the Apache server (cookies, user track, etc.) because lightweight virtualization containers do not impose high memory and storage overhead. Thus, we could maintain a large number of parallel-running Apache instances similar to the Apache threads that the server would maintain in the scenario without containers. If a session timed out, the Apache instance was terminated along with its container.

To test our system in a dynamic website scenario, we setup a dynamic Blog using the Word press blogging software. In our deployment, site visitors were allowed to read, post, and comment on articles. All models for the received front-end and back-end traffic were generated using these data. It's performance overhead, which is common for both static and dynamic models, in the following section. In our analysis, we did not take into consideration the potential for caching expensive requests to further reduce the end-to-end latency; this we left for future study.

VI. CONTAINER OVERHEAD

One of the primary concerns for a security system is its performance overhead in terms of latency. In our case, even though the containers can start within seconds, generating a container on the fly to serve a new session will increase the response time heavily. To alleviate this, we created pool of webserver containers for the forthcoming sessions to what Apache does with its threads. As sessions continued to grow, our system dynamically instantiated new containers upon completion of a session, we recycled these Containers by reverting them to their initial clean states [3].

The overhead of the server with container architecture was measured using a machine with the following specifications:
Four cores 2.8 GHz CPU, 8 GB memory, 100 MB/s .

VII EXPERIMENTAL RESULT

Fig.8.Main Page

Fig.9. Query Registration

Fig.10.User Login

VIII. CONCLUSION

We presented an intrusion detection system that builds models of normal behavior for multi- tiered web applications from both front-end web (HTTP) requests and back-end database (SQL) queries. Unlike previous approaches that correlated or summarized alerts generated by independent IDS, ADG forms session-based IDS with multiple input streams to produce alerts. Such correlation of different data streams provides a better characterization of the system for anomaly detection because the intrusion sensor has a more precise normality model that detect s a wider range of threats. Rather it also can prevent the web applications from intrusions.

REFERENCES:

1. Meixing Le, Angelos Stavrou, Brent ByungHoon Kang." DoubleGuard: Detecting Intrusions In Multi- tier Web Applications" IEEE transaction on dependable and secure computing vol.9 no.4 year 2012

2. Niraj Gaikwad, Swapnil Kandage, Dhanashri Gholap, "DoubleGuard: Detecting & Preventing Intrusions in Multitier Web Applications", International Journal of Networks and Systems, of Networks and Systems, 2(2), February – March 2013, 09 – 14, ISSN 2319 – 5975.

3. Rahul Dandwate, Lomesh Ahire, Dipali Kumbhar, Pratik Kamble, Aniket Shirude, Shweta Bhandakkar, "DOUBLEGUARD: DETECTING INTRUSIONS IN MULTITIER WEB ARCHITECTURE", Proceedings of IRF International Conference, 13th April-2014, Pune, India, ISBN: 978-93-84209-04-9

4. K.Karthika, K.Sripriyadevi ," To Detect Intrusions in Multitier Web Applications by using Double Guard Approach" , International Journal of Scientific & Engineering Research Volume 4, Issue 1, January-2013 1 ISSN 2229-5518.

5. S.Athirayan, A.Venkatesan ," Double Guard detection in multitier architecture".