DualGuard: Detecting Intrusions in Multitier Web Applications

Sachin Narayan Wandre¹, Manohar S. Chaudhari², Thaksen J. Parvat ³

Department of Computer Engineering, University of Pune
Sinhgad Institute of Technology, Lonavala, Pune, Maharashtra, India

Abstract- In our day to day life internet plays very important role, in terms communication and management of personal information from anywhere. Therefore for increase in application and complexity of data, webservices take interest in multitiered design like 1-tier, 2-tier and 3-tier design. By the use of these design webservers run the application front end logic and data are outsourced to a database or file server. For this we proposed a DualGuard, an IDS (Intrusion detection system) system. This system performs the network behavior of user sessions across the both the front end and the back end process. This system in front end handle the user session in webservice and in back end handle the user session for database. The basic performance of device is like a device or software application that monitors network and/or system activities for malicious activities or policy violations and produces reports to a Management Station. In this paper we qualify the limitation of multitier IDS system in terms of the training sessions and the functionality coverage. In the proposed work DualGuard is implemented using an Apache server with the MySQL database and lightweight virtualization.

Keywords— Multitier web application, Intrusion, Anomaly detection.

I. INTRODUCTION

Popularity and complexity of the web-delivered services and applications is raised in the recent decade. Banking, Business applications, travel, social networking all is handled via web. Such applications are handled web server front end that runs the application user interface logic, as well as a back-end server that consists of a database or file server. Web services are provided ubiquitously to handle personal or corporate operations, transactions etc. As web services handle such crucial operations it is always been the target of the attack. There are different types of attack on web services such as 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]). Current Intrusion detection Systems (IDS) examines network packets individually within both the web server and the database system. There is very little work being performed on multitier Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions.

Generally back end data server is protected using firewall while web server is remotely accessible over internet. Although back end data servers are protected against direct remote attacks there is possibility of attack by using web requests. Signatures or misused traffic patterns of known attacks are used to detect attack on the multitier architecture [7], [8], [9], [10]. A type of IDS that leverages machine learning can also recognize unknown attacks by identifying abnormal network traffic flow that is different from the so-called “normal” behavior previously profiled during the IDS training phase. Individually, the web IDS and the database IDS can detect abnormal network traffic sent to either of them. However, we found that these IDSs cannot detect attacks wherein normal traffic is utilized 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, attacker can find a way to issue a privileged database query by exploiting vulnerabilities in the web server. Web IDS and database IDS cannot 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 present multithreaded web server architecture, it is not feasible to detect or profile such causal mapping between web server traffic and database server traffic since traffic cannot be clearly attributed to user sessions. So there is need of IDS which can a system used to detect attacks in multitier web services.

This paper presents a novel Dual-guard IDS for multitier web services. Our approach combines the concept of web IDS and DB IDS, so it observes web front end (HTTP requests) and back end network transactions of each user of web service. To achieve this, we develop a lightweight virtualization technique to assign each user’s web session to a dedicated container, an isolated virtual computing environment. Container ID is used to map web request of the same user to respective database queries.
Within a lightweight virtualization environment, system ran many clones of the web server instances in different containers so that each one was separated from the rest. As momentary containers can be easily instantiated and destroyed, system assigns each client session a dedicated container so that, even when an attacker may be able to compromise a single session, the damage is confined to the compromised session; other user sessions remain unaffected by it.

Web services can of two types, one is static web services and another is dynamic web services. Static web services don’t allow content modification i.e. any changes in database from user dynamically. In static web services there is straight relationship between requests received to front end web server and request generated for database backend. Same database query is generated for same HTTP request every time. For static web services we can model relationship between requests of front end web server and generated database queries. In addition to the static websites there are websites which allows users for content modification. So HTTP requests from user may be different every time, generated database queries may be different each time. In the case of dynamic web services it is not easy to model relationship between HTTP requests and subsequent database queries.

II. RELATED WORK

Anomaly detection first requires the IDS to define and characterize the correct and acceptable static form and dynamic behavior of the system, which can then be used to detect abnormal changes or anomalous behaviors [11], [12]. An anomaly detector then compares actual usage patterns against established models to identify abnormal events. DualGuard comes under anomaly detection type in which and we depend on a training phase to build the correct model. An IDS such as in [13] also uses momentary information to detect intrusions. DualGuard, however, does not correlate events on a time basis, which runs the risk of mistakenly considering independent but concurrent events as correlated events. DualGuard does not have such a limitation as it uses the container ID for each session to causally map the related events, whether they are concurrent or not.

We know that database high security because it contains more valuable data. Therefore lots of efforts have been taken for the database IDS [14], [15], [16] and the database firewalls. The software such ad Green SQL, works as a reverse proxy for database connections. It will first connect to a database after that it connects to a database server and web application.

After connecting to the database SQL query are analyzed and if the problem is occurred then the process is forwarded to the back-end database server. For achieving more accurate detection the system proposed [17] to both the web IDS and database IDS and it also uses a reverse HTTP proxy to maintain a reduced level of service in the presence of false positives. However, we found that certain types of attack utilize normal traffics and cannot be detected by either the web IDS or the database IDS. In such cases, there would be no alerts to correlate.

Some previous approaches by analyzing the source code or executable detected intrusions or vulnerabilities. In our system the DualGuard design the new container-based webserver architecture possible to separate the different information flows by each session. It means that it is possible to track the information flow from the webserver to the database server for each session. In our design there is no need to analyze the source code or know the application logic. For the static webpage, by using DualGuard there is no need of application logic for building a model. For Dynamic web services, there is no need to know the basic user operation in order to model normal behavior.

For preventing or detecting SQL or Cross Site scripting injection attacks [18], [19] input validating is useful. This is orthogonal to the DualGuard approach, which can utilize input validation as an additional defense. DualGuard can detect SQL injection attack by taking the structure of web requests and database queries without looking into the values of input parameter. For building the mapping model in DualGuard required a large number of isolated web task instance so that mapping pattern would appear across different session instances.

III. PROPOSED WORK

Proposed Work

We developed the container-based and session-separated webserver. The architecture of container-based and session-separated not only increases the security performance but also provide isolated information flows that are separated in each container session. It gives us the permission for mapping between the webserver requests and the subsequent database queries, to utilize such a mapping model to detect abnormal behaviors on a session/client level. We know that in the three-tier webserver architecture, server receives HTTP request from user clients and then issues SQL queries to the database server to retrieve and update data. These SQL queries are casually dependent on the web request hitting the webserver.
For all these reasons we want to model mapping relationship. This mapping relationship includes all legitimate traffic for detecting abnormal/attack traffic. On the designing phase we put sensors at both sides of the server. This sensor is deployed on the host system and cannot be attacked directly, only the virtualized containers are exposed to attackers. This server will not attack on database server that is the attacker cannot take control to the database server.

**Block Diagram**

Fig. 1 shows the how the webserver running in the container. There are two sensors on the both end of the server as we discussed later. One sensor at the web request anamoly sensor and second is database queries anamoly server as shown in the following diagram. Mapping model relationship is present at the middle of the two web server and the database. As shown in fig.1 separate session/instance of web server is created for each client by virtual environment. Instance of web server for client is called as Container. Container keeps all web requests from client and its respective generated DB queries. One normality model is designed which contains all normal DB queries for respective HTTP requests. So to detect anomalous behavior entries in container and normality model is compared. If we get different entries then anomaly is detected.

The system is effective to capturing the following types of attack:

1. **Privilege escalation Attack**

   Let’s assume that the website serves both regular users and administrators. Both the user and admin triggers the SQL queries. Suppose an attacker log in as a regular user to the web server upgrades his/her privileges, and triggers admin queries so as to obtain an administrator’s data. This attack can never be detected by either the web server IDS. Our approach, however, can detect this type of attack.

2. **Hijack Future Session Attack**

   This type of mainly found at web server side. An attacker usually takes over the web server and therefore hijacks all subsequent legitimate user sessions to launch attacks. Fortunately, the isolation property of our container based web server architecture can also prevent this type of attack. As each user’s web requests are isolated into a separate container, an attacker can never break into other users’ sessions.

3. **Injection Attack**

   This type of attack does not require compromising the web server. Attackers can use existing vulnerabilities in the web server logic to inject the data or string content that contains the exploits and then use the web server to relay these exploits to attack the back-end database. Since our approach provides a two-tier detection, even if the exploits are accepted by the web server, the relayed contents to the DB server would not be able to take on the expected structure for the given web server request.

4. **Direct DB Attack**

   To directly connect to the database by directly passing to the web server and the firewalls it is possible for the attacker. Without sending the web request attacker enters to the web server. The database IDS could not detect it. However, this type of attack can be caught with our approach since we cannot match any web requests with these queries.
Here we developed an algorithm which take the input as training data set and result as building the mapping model for static website. Algorithm assigns a hash table entry for each unique HTTP request and database query. In hash table HR for request and HQ for the query.

\textbf{Algorithm 1:} algorithm for Building a Static Model

\textbf{Input:} Training data set, Threshold \( l \)

\textbf{Output:} Mapping Model for Static Website

1: for each session traffic \( T \) do
2: \hspace{1em} Get different HTTP requests \( r \) and DB queries \( q \) in this session
3: \hspace{1em} for each different \( r \) do
4: \hspace{2em} if \( r \) is a request to static file then
5: \hspace{3em} Add \( r \) into set EQS
6: \hspace{2em} else
7: \hspace{3em} if \( r \) is not in set REQ then
8: \hspace{4em} Add \( r \) into REQ
9: \hspace{2em} Append session ID \( I \) to the set HR with \( r \) as the key
10: \hspace{1em} for each different \( q \) do
11: \hspace{2em} if \( q \) is not in set SQL then
12: \hspace{3em} Add \( q \) into SQL
13: \hspace{2em} Append session ID \( I \) to the set HQ with \( q \) as the key
14: \hspace{1em} for each distinct HTTP request \( r \) in REQ do
15: \hspace{2em} for each distinct DB query \( q \) in SQL do
16: \hspace{3em} compare the set HR with the set HQ
17: \hspace{2em} if HR = HQ and Cardinality (HR) > \( l \) then
18: \hspace{3em} Found a Deterministic mapping from \( r \) to \( q \)
19: \hspace{3em} Add \( q \) into mapping model set MSr of \( r \)
20: \hspace{3em} Mark \( q \) in set SQL
21: \hspace{2em} else
22: \hspace{3em} Need more training sessions
23: \hspace{2em} return False
24: \hspace{1em} for each DB query \( q \) in SQL do
25: \hspace{2em} if \( q \) is not marked then
26: \hspace{3em} Add \( q \) into set NMR (No match request)
27: \hspace{1em} for each HTTP request \( r \) in REQ do
28: \hspace{2em} if \( r \) has no deterministic mapping model then
29: \hspace{3em} Add \( r \) into set EQS
30: \hspace{1em} return true

\textit{D. Mathematical Model:}

\textbf{Input:} HTTP request (REQ\( i \)) by client (CL\( i \))

\textbf{Process:}

Container (Ci) created for CL\( i \)

Respective Database query (Qi) is generated.

Result (R) is generated for Q from DB

R is given to CL\( i \)

Ci records REQi and Qi

Ci is compared with N (normality model)

IF Ci abnormal intrusion is detected ID of Ci recorded

\textbf{Output:} ID of anomalous Ci

\textbf{IV. CONCLUSION AND FUTURE WORK}

For the security of database we proposed an intrusion detection system, which has a property of building models of normal behavior for multilitered web application from the front-end web (HTTP) requests and back-end database queries. In comparison to the previous approach the system correlated or summarized alerts generated by independent IDS, DualGuard forms a container based IDS with multiple input streams to produce alert. we have implemented that the correlation of input streams provides a better characterization of the system for anomaly detection because the intrusion sensor has a more precise normality model that detects a wider range of threats.

For achieving this we isolate the flow of information from each server session with a lightweight virtualization. By implementing static and dynamic web requests with the back-end file system and database queries we detect the accuracy of our approach. We implement the system on both static and dynamic website. In static website, we proved that our system to be effective at detecting different types of attack by building a well correlated model. Moreover, we showed that this held true for dynamic requests where both retrieval of information and updates to the back-end database occur using the webserver front end.
REFERENCE


