“Advance Double Guard System: Detecting & Preventing Intrusions in Multi-Tier Web Applications”

Shinde Jyoti R.¹, Prof. Dabhade Sheetal V.²

¹, ²Department of Computer Engg, Smt Kashibai Nawale College of Engg, Vadgaon, Pune, India.

Abstract— Today there is huge amount of use of computer especially for web application. Most of the people do their transaction through web application [6]. So there are risks of personal data gets hacked then need to be provide more security for both web server and database server. For that Advance double guard system is used. In Advance double guard system for detecting & preventing attacks Intrusion detection system is used.

This system prevents attacks and user account from intruder from hacking his/her account. By using IDS, system can deliver security for both web server and database server using mapping of request and query string. An IDS system that models the network performance of user sessions across both the front-end web server and the back-end database. System will try this by separating the flow of information from each web server session. It measure the detection accuracy when system attempt to model static and dynamic web requests with the back-end file system and database queries. For static websites, system built a well-correlated model, for effectively detecting various types of attacks. Moreover, system showed that this held true for dynamic requests where both recovery of information and updates to the back-end database occur using the web-server front end.

Network security has been a very important issue, since the rising development of the Internet. There has been an increasing need for security systems against the external attacks from the attackers in order to do this requirement we develop Advance Double Guard system in the network.

Keywords— Session ID, Query String, IDS, Deterministic Mapping, Container, Virtual Environment.

I. INTRODUCTION

In our prototype, we present Advance Double Guard; a system used to detect attacks in multi-tiered web services and prevents web servers from malicious attacks [13][14]. Our approach can create normality models of separate user sessions that include both the web front-end and back-end network transactions. In the web application For the front-end HTTP is used and for the back-end File or Structured Query language is used. We use the container ID to accurately associate the web request with the subsequent DB queries. Thus, Advance DoubleGuard can build a causal mapping profile by taking both the web server and DB traffic into account.

The container-based web architecture not only accepts the profiling of causal mapping, but it also provides a separation that prevents future session-hijacking attacks. Within a virtualization environment, we ran many copies of the web server instances in different containers so that each one was separated from the rest. As temporary containers can be easily instantiated and damaged, we assigned 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.

In this paper, we show that, for websites that do not permit content modification from users, there is a direct causal relationship between the requests received by the front-end web server and those generated for the database back-end. In addition to this static website, there are web services that permit determined back-end data modifications. These services, which we call dynamic, allow HTTP requests to include parameters that are flexible and depend on user input. Therefore, our ability to model the causal relationship between the front-end and back-end is not always deterministic.

II. RELATED WORK

An IDS such as [21] also uses sequential information to detect intrusions. Advance DoubleGuard, however, does not correlate events on a time basis. It runs the risk of mistakenly considering independent but concurrent events as correlated events. Advance Double Guard does not have such a limitation as it uses the container ID for each session to causally map the related events, whether they be concurrent or not.

Databases always contain more valuable information. So they should receive the highest level of protection. Therefore, significant research efforts have been made on database IDS [22], [20], [12] and database firewalls [17]. These software's, such as Green SQL [15], work as a reverse proxy for database connections. Web applications will first connect to a database firewall, instead of connecting to a database server. SQL queries are analyzed, if they’re safe, then they are forwarded to the back-end database server.
To achieve more accurate detection, the system proposed in [24] composes both 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. Some previous approaches have detected intrusions by statically analyzing the source code [3], [1], [9]. Others [23], [11], [4] dynamically track the information flow to understand taint propagations and detect intrusions. In Advance Double Guard, the new container-based web server architecture enables us to separate the different information flows by each session. For each session, this provides a means of tracking the information flow from the web server to the database server. Our approach also does not require us to know the application logic or analyze the source code. For the static web page, our Advance Double Guard approach does not require application logic for building a model. However, as we will discuss, although we do not require the full application logic for dynamic web services, we do need to know the basic user operations in order to model normal behavior.

Validating input is useful to detect or prevent SQL or XSS injection attacks [16], [10]. However, we have found that Advance 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).

Virtualization is used to increase security performance and isolate objects. An alternative is a lightweight virtualization, such as Linux-VServer [5], OpenVZ [19], or Parallels Virtuozzo [8]. In general, these are based on some sort of container concept. A group of processes still appears to have its own dedicated system, with its container, yet it is running in an isolated environment. To isolate different application instances, there are also some desktop systems [2], [25] that use lightweight virtualization. Such virtualization techniques are commonly used for restraint and separation of attacks. However, in our Advance Double Guard, we utilized the container ID to separate session traffic as a way of extracting and identifying causal relationships between web server requests and database query events.

CLAMP [18] is architecture for preventing data leaks even in the presence of attacks. By isolating data at the database layer by users and code at the web server layer, CLAMP guarantees that a user’s sensitive data can only be accessed by code running on behalf of different users. In contrast, Advance Double Guard focuses on modeling the mapping patterns between HTTP requests and DB queries to detect malicious user sessions.

CLAMP requires modification to the existing application code, and the Query Restrictor works as a proxy to mediate all database access requests. CLAMP requires platform virtualization whereas, Advance Double Guard uses process isolation, and CLAMP provides more coarse-grained isolation than Double Guard. However, Advance Double Guard would be ineffectual at detecting attacks if it were to use the coarse-grained isolation as used in CLAMP. Building the mapping model in Advance Double Guard would require a large number of isolated web stack instances so that mapping patterns would appear across different session.

### III. Problem Definition

We present Advance Double Guard system, a system used to detect and prevent attacks in multi-tiered web services. We can create normality models of isolated user sessions that include both the web front-end and back-end network transactions. In the front end HTTP is used and in the back end File or Structured Query Language is used. To achieve this, to assign each user’s web session to a dedicated container. We use the container ID to accurately associate the web request with the subsequent Data Base queries. Thus, Advance Double Guard can build a causal mapping profile by taking both the web server and Data Base traffic into account.

Before Advance Double guard was developed the system which is present prevents web server and database from Linearization only. Before double guard not much security provided to the web server and database. This system cannot handle all attack. We need to use two different technologies, one for web server and another for database to prevent from attacks.

#### A. Goal

Our primary goal is to build Advance Double Guard model to improve on detection rate and prevention rate and compare results with IDS model and Existing Double Guard System and improve the search efficiency & Performance of Advance Double Guard. Our aim to enable strong data detection and protection for web applications while at the same time we minimize the false positive rate.

### IV. System Architecture and Detention

We assume that the database server will not be totally taken over by the attackers. Attackers may attack the database server through the web server or, more directly, by submitting SQL queries, they may obtain and infect sensitive data within the database.
A. Normal Model of System:

We assume no prior information of the source code or the application logic of web services used on the web server. We are only analyzing network traffic that spreads the web server and database. We assume that no attack would occur during the training phase and model building.

Figure 1. Basic Multitier Architecture.

The web server acts as the front-end, with the file and database servers acting as the content storage back-end. Here Rq1 is the Request from Client 1(S1) to the Web server and Rs1 is the Response from web server to the client 1(S1) and so on.

In our design, we use containers as temporary, throwaway servers for client sessions. It is possible to initialize thousands of containers on a single physical machine, and these virtualized containers can be rejected, regressed, or quickly reinitialized to serve new sessions. Only one physical web server runs many containers, each one an exact copy of the original web server. Our approach dynamically generates new containers and reprocesses used ones. As a result, only one physical server can run continuously and serve all web requests.

Figure 2. Web server occurrences running in containers.

B. Scenarios of Attack

Our approach is effectively capturing the following type of attacks.

Session Fixation Vulnerabilities: If you log out your current account, you get login page. After that, immediately if you undo the page you get account which is logged out. In our Prototype, When user logout and after that click on back button instead of getting last visited page he redirect to login page.

Figure 3: Session Fixation Vulnerabilities

Brute Force Attack: In this attack attacker tries different techniques and many attempts to get user id and password. While trying different techniques by using many attempts he/she would access account of authorize user. In our system, User account will be lock after entering wrong password 3 times.
**International Journal of Emerging Technology and Advanced Engineering**

Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal)

**Figure 4. Brute Force Attack**

**SQL Injection:** User enter particular line (' OR 1 = 1; --) into user name and enter any password he/she login into account and access system as a authorize user. In our prototype for preventing system from sql injection use stored procedure. In that instead of passing direct value to database value should be pass in the form of paramaterised query.

**Figure 5. SQL Injection**

**V. Modeling Mapping Patterns**

Different web applications exhibit different characteristics due to the various functionality. Some Web sites allow regular users with the non-administrative rights to update the contents of the server data. This creates challenge for IDS system because the HTTP requests can contain variables in the past parameters. Our approach normalizes the variable values in both HTTP requests and database queries, protecting the structures of the requests and queries. Following this step, session i will have a set of requests (Ri), as well as a set of queries (Qi). If the total number of sessions of the training phase is N, then we have the set of total web requests (R) and the set of total SQL queries (S) across all sessions. Each single web request rm € R may also appear several times in different Ri where i = 1,2 . . . n. The same holds true for qs € S. We classify the four possible mapping patterns [7]. The mappings in the model are always in the form of one request to a query set rm Qn. The possible mapping patterns are Deterministic Mapping, No Matched Request, Empty Query Set and Nondeterministic Mapping.

**VI. Modeling For Static And Dynamic Websites**

In the case of a static website, the non-deterministic mapping does not exist as there are no available input variables or states for static content. We can easily classify the traffic collected by sensors into three patterns in order to build the mapping model. As the traffic is already separated by session, we begin by iterating all of the sessions from 1 to N. For each rm € R, we maintain a set ARm to record the IDs of sessions in which rm appears. The same holds for the database queries. We search for the AQs that equals the ARm. When ARm = AQs, this indicates that every time rm appears in a session, then qs will also appear in the same session, and vice versa. Some web requests that could appear separately are still present as a unit. In contrast to static webpages, dynamic webpages allow users to generate the same web query with different parameters. Additionally, dynamic pages often use POST rather than GET methods to commit user inputs. Based on the web servers application logic, different inputs would cause different database queries. By placing each rm , or the set of related requests Rm , in different sessions with many different possible inputs, we obtain as many candidate query sets { Qn , Qp , Qq . . .} as possible. This mapping model includes both deterministic and nondeterministic mappings, and the set EQS is still used to hold static file requests.

**VII. Performance Evaluations**

The implementation of our prototype involves the web server and the back-end DB. We also used two testing websites, static and dynamic. We analyzed three classes of attacks and measured the false positive rate for each of the two websites. Finally we compared the user behavior for each of the session for a different set of users. The following represents the implementation of our prototype and the attack detection rates.

**A. Prototype Implementation**

In our design, we choose to assign every user session into a different container which was the security design decision. Each and every new client (IP address) is assigned to a new container and these containers are cast-off or recycled based on the session time out. The session time out is considered to be 30-minute. Thus, we are capable of running multiple instances in a single server.
Figure 6: The overall Architecture of the model

The above figure shows the architecture and session management of our prototype, where the host web server acts as the dispatcher. In the case of the static website, we served 14 unique web pages and collected real traffic to this website and obtained 300 user sessions. In the case of the dynamic websites, the site visitors are allowed to read post and comment on articles.

The Query Controller is a trusted module that controls Web Server to access sensitive database content. In our implementation, the Query Controller interrupts all databases traffic without requiring changes to the Web Server.

We evaluated the overhead of our container-based server against an IDS web server. The testing website was the dynamic blog website, and both IDS web server and the container-based web server (Advance Double Guard System) connected to the same SQL Server (Database server) on the host machine. For the container-based server, we maintained a pool of 120 web server instances on the machine. We used the rate of 5 (i.e., it emulated 5 concurrent users). We tested under the parameters of 100, 200, and 400 total fetches, as well as 5 and 15 seconds of fetches. For example, in the 100-fetches benchmark, it fetches the URLs as fast as it can 100 times.

Similarly, in the 15 seconds benchmark, it fetches the URLs as fast as it can during the last 15 seconds. We picked 15 major URLs of the website and tested them against both servers. Figure 7 shows our experiment results. The value of fetches per second is the most important indicator to reflect web server throughput performance. From the figure, we can observe that the overhead varied from 11.3% to 27.2%, under the full working load. When we put the parameters at 5 and 15 seconds, the overhead was about 22%.

B. Static Website Model in Training Phase

For the static website, Deterministic Mapping and the Empty Query Set Mapping patterns appear in the training sessions. We first collected 140 real user sessions for a training data set before making the website public so that, there was no attack during the training phase. We used part of the sessions to train the model and all the remaining sessions to test it. For each number on the x-axis of Fig. 6, we randomly picked the number of sessions from the overall training sessions to build the model using the algorithm, and we used the built model to test the remaining sessions. We repeated each number 15 times and obtained the average false Positive rate (since there was no attack in the training data set).
Fig. 8 shows the training process. As the number of sessions used to build the model increased, the false positive rate decreased (i.e., the model became more accurate). From the same figure, we can observe that after taking 33 sessions, the false positive rate decreased and stayed at 0. This implies that for our testing static website, 33 sessions for training would be sufficient to correctly build the entire model. Based on this training process accuracy graph, we can determine a proper time to stop the training.

C. Dynamic Model Detection Rates

We also conducted model building experiments for the dynamic blog website. We obtained 165 real user traffic sessions from the blog under daily workloads. During this phase, we made our website available only to internal users to ensure that no attacks would occur. We then generated 13 attack traffic sessions mixed with the normal legitimate user session, hence the mixed traffic is used for the attack detection.

The figure 9 shows the ROC curves for the testing result. We built our models with different number of operations, and each point on the curves indicates the threshold value. The threshold value is defined as the number of HTTP requests or SQL queries in a session that are not matched with the normality model. The nature of the false positives comes from the fact that our manually extracted basic operations are not sufficient to cover all legitimate user behaviors.

Figure 9: False positive rate for dynamic models

VIII. CONCLUSION

This paper consists of intrusion detection system to Detect & prevent web application from intruders. Paper developed IDS for prevents both website, static and dynamic from intruder. The attack which cannot be prevented by IDS that attacks also prevented by Double guard. This paper find intruder by using container id that contain session id and IP address of that user. IDS do the mapping of request and query and by mapping it identify user is authorize user or intruder.

We achieved this by isolating the flow of information from each web server session with a virtualization technique. Also, we quantified the detection accuracy of our Approach when we attempted to model static and dynamic web requests with the back-end file system and database queries. When we organized our prototype on a system that working on Internet Information Security (IIS) server, a blog application and a SQL Server back-end, Advance Double Guard was able to identify a wide range of attacks with minimal false positives. Finally, for dynamic web applications, we reduced the false positives to 0.63%.
REFERENCES


