Adavance Double Guard System : Detecting & Preventing Intrusions In Multi-Tier Web Applications

[1]Ms. Shinde Jyoti R., [2] Asst. Prof. Dabhade Sheetal V., [3]Prof. Pathan S.K.

[1, 2, 3](, Department of Computer Engg, Smt Kashibai Nawale College of Engg , Vadgaon)

[1, 2, 3] (Pune, India.)

[1]shindejyoti247@gmail.com, [2]sheetal.dabhade@gmail.com, [3]spathan@rediffmail.com

Abstract—**In today's world there is huge amount of use of computer mainly for web application. Most of the people do their transaction through web application [1]. So there are chances of personal data gets hacked then need to be provide more security for both web server and database server. For that this Advance double guard system is used. In this Advance double guard system for detecting & preventing attacks, Intrusion detection system is used.**

This system Detects attacks and prevents user account from intruder from hacking his/her account. By using IDS, system can provide security for both web server and database server using mapping of request and query. The network behavior of user sessions across both the front-end web server and the back- end database that model using an IDS System. This system able to search for in a place (container) attacks that DoubleGuard would not be able to identify. System will try this by isolating the flow of information from each web server session. It quantify 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 different types of attacks. Moreover, system showed that this held true for dynamic requests where both retrieval 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 evolution 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, Mapping Patterns, Virtual Environment.

I. INTRODUCTION

In this paper, we present Advance Double Guard system, a system used to detect attacks in multi-tiered web services and prevents web servers from malicious attacks[2][3]. Our approach can create normality models of remote user sessions. This session include both the web front-end (HTTP) and back-end (File or SQL) network transactions. To accurately associate the web request with the subsequent DB queries the container ID is used. Thus, this 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 adopts the profiling of causal mapping, but it also provides an isolation that prevents future session-hijacking attacks. We ran many copies of the web server instances in different containers so that each one was isolated from the rest. As ephemeral containers can be easily instantiated and destroyed, 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.

Using our prototype, 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 case, there are web services that permit determined back-end data modifications. The services, which we call dynamic, allow HTTP requests to include parameters that are variable and depend on user input. Therefore, our approach to model the causal relationship between the front-end and back-end is not always deterministic and depends primarily upon the application logic.

II. RELATED WORK

An IDS such as [8] 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 [7], [6], [9] and database firewalls [5]. These software's, such as Green SQL [4], 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 [16] 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 [15], [19], [14].

Others [13], [20], [12] 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 [11], [21]. 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 [10], OpenVZ [25], or Parallels Virtuozzo [18]. 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 [17], [23] that use lightweight virtualization. Such virtualization techniques are commonly used for containment and isolation 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 [24] 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 DEFINATION

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 (HTTP) and back-end (File or SQL) network transactions. 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 DB queries. Thus, Advance Double Guard can build a causal mapping profile by taking both the web server and DB 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.

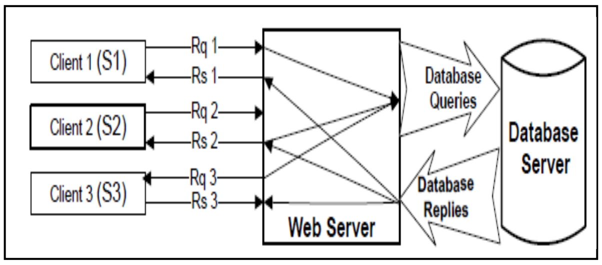

Figure 1. Basic Multitier Architecture.

The web server acts as the front-end, with the file and database servers acts 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.

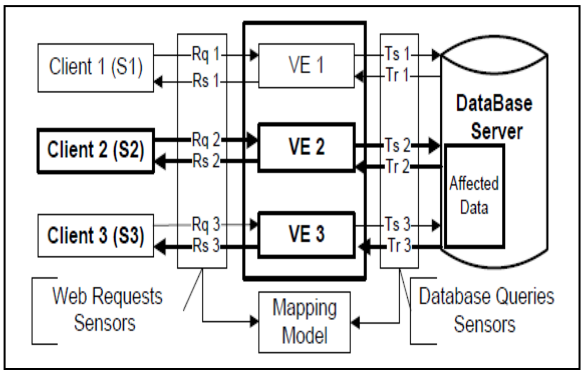

Figure 2. Web server occurrences running in containers.

Figure 2 shows how communications are categorized as sessions and how database transactions can be related to a corresponding session. In the figure 2, the client 2 will compromise only the VE2, and the corresponding database transaction T2 will be the affected section of the data within the database. Here, Rq is the Request made by the client to the "Web Server Virtual Machine", Rs is the Reply from the VE to the client, VE is the Virtual Environment, Ts is the Database Transaction set request, Tr is the Reply made by the database server with the corresponding set of queries.

In fact, we assume our sensors cannot be attacked and we always intern correct traffic information at both the ends. Moreover, as traffic can be easily being divided by session, it is possible and easy for us to compare and analyze the request and queries across different sessions. Once we build the mapping model, it can be thus used to detect abnormal behaviors. Both the web request and the database queries within each session should be in unity with the model. If there rises any request or query violating the regular model within the session, then the identical session is well thought-out to be an attack.

B. *Scenarios of Attack*

Our approach is effectively capturing the following type of attacks.

a) *Session Hijaking*

When user copies the address from the address bar and paste it to another active address bar system get open the page corresponding that address. But our prototype provides to assign each user session into a different container; however this was a design decision. For instance, we can allocate a new container per each new IP address of the client. In our system, containers were reused based on events or when sessions time out.

b) *Session replay*

If your system remains idle for few minutes, after that you try some action on website, rather than perform that action system will go login page. How this is possible in or prototype because in our prototype implementation, we used a 60-minute timeout due to resource constraints of our test server. However, this was not a limitation and could be removed for a production environment where long-running processes are required.

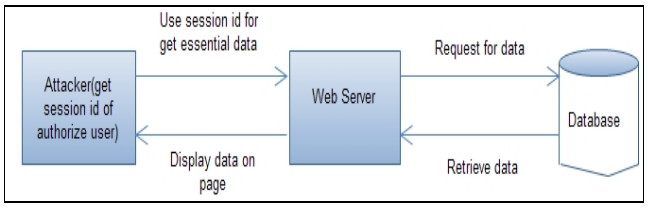
Use session id for get essential data
Attacker(get session id of authorize user)
Web Server
Request for data
Database
Display data on page
Retrieve data

Figure 3: Session replay

c) *Directory Browsing Attack*

On web servers, Hackers cannot directly get list of files. Directories on the web server are typically locked down to prevent remote browsing when the directory contains executable, text files, documentation, or configuration materials. In such cases either the entire directory is configured to block access, or access is granted on a per file basis, requiring a exact request to access objects in the directory. Directory listing can be prevented in server configuration files, but may also arise from susceptibility in a particular application.

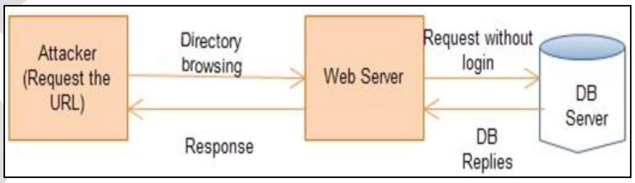

Figure 4. Directory Browsing Attack

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 (R_i), as well as a set of queries (Q_i). 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 r_m € R may also appear several times in different R_i where i = 1,2 . . .n.

The same holds true for q_n € S. We classify the four possible mapping patterns [22]. The mappings in the model are always in the form of one request to a query set r_m Q_n. 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 $r_m \in R$, we maintain a set AR_m to record the IDs of sessions in which r_m appears. The same holds for the database queries. We search for the AQ_s that equals the AR_m. When $AR_m = AQ_s$, this indicates that every time r_m appears in a session, then q_s 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 r_m, or the set of related requests R_m, in different sessions with many different possible inputs, we obtain as many candidate query sets $\{ Q_n , Q_p , Q_q . . .\}$ 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

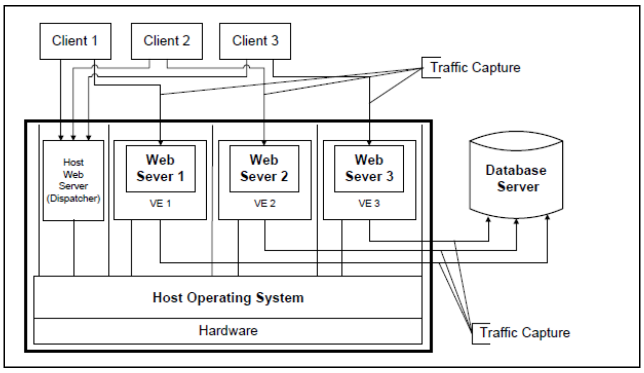

Figure .5 : The overall Architecture of the model

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 12 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.

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 150 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

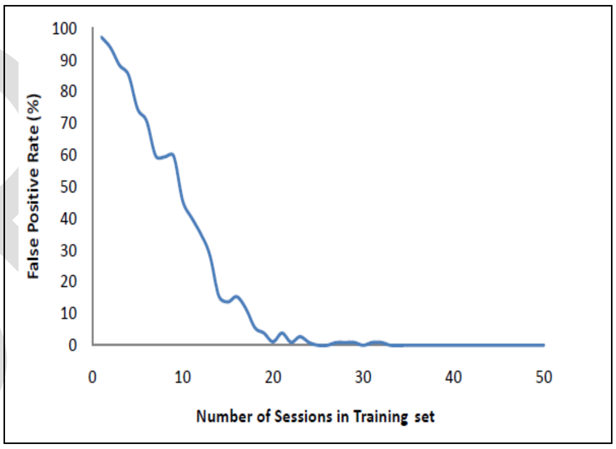

Figure.6 : False positives verses training time for static websites

positive rate (since there was no attack in the training data set).

Fig. 6 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 30 sessions, the false positive rate decreased and stayed at 0. This implies that for our testing static website, 30 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 185 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 15 attack traffic sessions mixed with the normal legitimate user session, hence the mixed traffic is used for the attack detection.

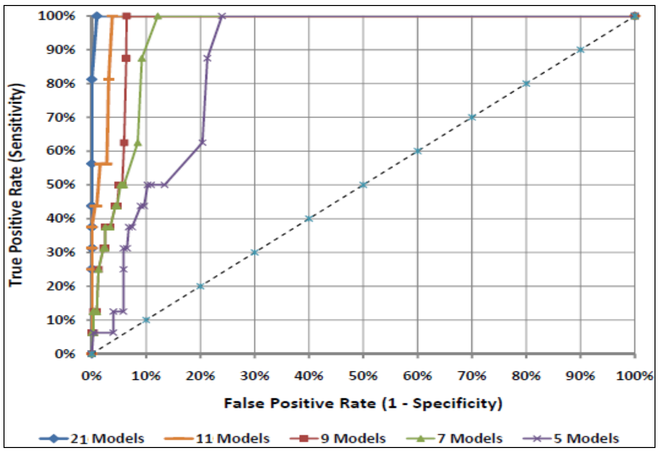

Figure 4: False positive rate for dynamic models

The above figure 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.

CONCLUSION

This paper consist 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 Doubleguard. 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 0.65%.

REFERENCES

[1]. SANS, “The Top Cyber Security Risks,” http://www.sans.org/top cyber-security-risks/,2011
[2]. D.E.Denning, “An Intrusion Detection Model”. IEEE Transactions on Software Engineering, 13(2):222to232, February 1987.
[3]. T. Lane and C.E. Brodley. “Temporal sequence learning and data reduction for anomaly detection”. In Proceedings of the 5th ACM conference on Computer and communications security, pages 150 to158. ACM Press, 1998.
[4]. greensql. http://www.greensql.net/.
[5]. K. Bai, H. Wang, and P. Liu. Towards database firewalls. In DBSec 2005.
[6]. Y. Hu and B. Panda. A data mining approach for database intrusion detection. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, SAC. ACM, 2004.
[7]. Lee, Low, and Wong. Learning fingerprints for a database intrusion detection system. In ESORICS: European Symposium on Research in Computer Security. LNCS, Springer-Verlag, 2002.
[8]. A. Seleznyov and S. Puuronen. Anomaly intrusion detection systems: Handling temporal relations between events. In RAID 1999.
[9]. A. Srivastava, S. Sural, and A. K. Majumdar. Database intrusion detection using weighted sequence mining. JCP, 1(4), 2006.
[10]. Linux-vserver. http://linux-vserver.org/.
[11]. D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in client-side xss filters. In Proceedings of the 19th international conference on World wide web, 2010.
[12]. P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kr¨ugel, and G. Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In NDSS 2007.
[13]. R. Sekar. An efficient black-box technique for defeating web application attacks. In NDSS. The Internet Society, 2009.
[14]. V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna. Toward Automated Detection of Logic Vulnerabilities in Web Applications. In Proceedings of the USENIX Security Symposium, 2010.
[15]. D. Wagner and D. Dean. Intrusion detection via static analysis. In Symposium on Security and Privacy (SSP ’01), May 2001.
[16]. G. Vigna, F. Valeur, D. Balzarotti, W. K. Robertson, C. Kruegel, and E. Kirda. Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries. Journal of Computer Security, 17(3):305–329, 2009.
[17]. S. Potter and J. Nieh. Apiary: Easy-to-use desktop application fault containment on commodity operating systems. In USENIX 2010 Annual Technical Conference on Annual Technical Conference.
[18]. Virtuozzo containers .http://www.parallels.com/products/pvc45/
[19]. M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns.
[20]. G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. ACM SIGPLAN Notices, 39(11), Nov. 2004.
[21]. T. Pietraszek and C. V. Berghe. Defending against injection attacks through context-sensitive string evaluation. In RAID 2005.
[22]. 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.
[23]. Y. Huang, A. Stavrou, A. K. Ghosh, and S. Jajodia. Efficiently tracking application interactions using lightweight virtualization. In Proceedings of the 1st ACM workshop on Virtual machine security, 2008.
[24]. B. Parno, J. M. McCune, D. Wendlandt, D. G. Andersen, and A. Perrig. CLAMP: Practical prevention of large-scale data leaks. In IEEE Symposium on Security and Privacy. IEEE Computer Society, 2009.
[25]. Openvz. http://wiki.openvz.org.