www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 7 (July 2013), Page No. 2162-2166

An Intrusion Detection System for Multitier Web Applications Using Double Guard

Shapna Rani.E, G.Sathesh Kumar, Mythili.R , Karthick.R

Assistant Professor M.A.M College of Engineering & Technology, Trichurappalli.
Email ID: shapnaedwin@gmail.com

Assistant Professor M.A.M College of Engineering & Technology, Trichurappalli.

Email ID: gskme83@gmail.com

Assistant Professor Mahendra College of Engineering, Namakkal

Email ID: mythili.infotech@gmail.com

Assistant professor M.A.M College of Engineering and Technology,Trichy

Email ID: karthickkiwi@gmail.com

Abstract: Internet services and applications have become an inextricable part of daily life, enabling communication and the management of personal information from anywhere. To accommodate this increase in application and data complexity, web services have moved to a multi tiered design wherein the web server runs the application front-end logic and data are outsourced to a database or file server. Presenting Double Guard, an Intrusion Detection

System that models the network behavior of user sessions across both the front-end web server and the back-end database. By monitoring both web and subsequent database requests, it is possible to ferret out attacks that independent IDS would not be able to identify.

Keyword - Anomaly detection, virtualization, multitier web application.

1. Introduction

Web delivered services and applications have increased in both popularity and complexity over a past few years. Daily tasks, such as travel, and social networking, are all done via the web. Such services typically employ a 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. Due to their ubiquitous use for personal and/or corporate data, web services have always been the target of attacks. These attacks have recently become more diverse, as attention has shifted from attacking the front end to exploiting vulnerabilities of the web applications, In order to corrupt the back-end database system. A plethora of Intrusion Detection Systems (IDSs) currently examine network packets individually within both the web server and the database system. However, there is very little work being performed on multi tiered Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions. In such multi tiered 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 that use web requests as a means to exploit the back end.

To protect multi tiered web services, Intrusion

detection systems have been widely used to detect known attacks by matching misused traffic patterns or signatures. A class of IDS that leverages machine learning can also detect unknown attacks by identifying abnormal network traffic that deviates 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.

Using Double Guard approach, both the front end and back end transactions can be prevented from attacks.

2. Threat Model and System Architecture

Setting up our threat model to include our assumptions and the types of attacks we are aiming to protect against. We assume that both the web and the database servers are vulnerable. Attacks are network borne and come from the web clients; they can launch application-layer attacks to compromise the web servers they are connecting to. The attackers can bypass the web server to directly attack the database server. We assume that the attacks can neither be detected nor prevented by the current web server IDS, that attacker may take over the web server after the attack, and that afterward they can obtain full control of the web server to launch subsequent attacks. For example, the attackers could modify the application logic of the web applications, eavesdrop or hijack other users' web requests, or intercept and modify the database queries to steal sensitive data beyond their privileges.

On the other hand, at the database end, assuming that the database server will not be completely taken over by the attackers. Attackers may strike the database server through the web server or, more directly, by submitting SQL queries, they may obtain and pollute sensitive data within the database. These assumptions are reasonable since, in most cases, the database server is not exposed to the public and is therefore difficult for attackers to completely take over. We assume no prior knowledge of the source code or the application logic of web services deployed on the web server. In addition, analyze only network traffic that reaches the web server and database. Assuming that no attack would occur during the training phase and model building.

2.1 Architecture and Confinement

In our design, 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. A single physical web server runs many containers, each one an exact copy of the original web server. Our approach dynamically generates new containers and recycles used ones. As a result, a single physical server can run continuously and serve all web requests. However, from a logical perspective, each session is assigned to a dedicated web server and isolated from other sessions. Since we initialize each virtualized container using a read-only clean template, we can guarantee that each session will be served with a clean web server instance at initialization. Choosing to separate communications at the session level so that a single user always deals with the same web server. Sessions can represent different users to some extent, and we expect the communication of a single user to go to the same dedicated web server, thereby allowing us to identify suspect behavior by both session and user. Detecting abnormal behavior in a session, treat all traffic within this session as tainted. If an attacker compromises a vanilla web server, other sessions' communications can also be hijacked. In our system, an attacker can only stay within the web server containers that he/she is connected to, with no knowledge of the existence of other session communications. Ensuring that legitimate sessions will not be compromised directly by an attacker.

2.2 Building the Normality Model

This container-based and session-separated web server 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 web server requests and the subsequent DB queries, and to utilize such a mapping model to detect abnormal behaviors on a session/client level. In typical three-tiered web server architecture, the web server receives HTTP requests from user clients and then issues SQL queries to the database server to retrieve and update data. These SQL queries are causally dependent on the web request hitting the web

server. Prepare model such causal mapping relationships of all legitimate traffic so as to detect abnormal/attack traffic.

In practice, building such mapping under a classic three-tier setup is quite difficult. Although the web server can distinguish sessions from different clients, the SQL queries are mixed and all from the same web server. It is impossible for a database server to determine which SQL queries are the results of which web requests, much less to find out the relationship between them. Even if we knew the application logic of the web server and were to build a correct model, it would be impossible to use such a model to detect attacks within huge amounts of concurrent real traffic unless we had a mechanism to identify the pair of the HTTP request and SQL queries that are causally generated by the HTTP request. However, within our container-based web servers, it is a straightforward matter to identify the causal pairs of web requests and resulting SQL queries in a given session. Moreover, as traffic can easily be separated by session, it becomes possible for us to compare and analyze the request and queries across different sessions. To that end, we put sensors at both sides of the servers. At the web server, our sensors are deployed on the host system and cannot be attacked directly since only the virtualized containers are exposed to attackers. Our sensors will not be attacked at the database server either, as we assume that the attacker cannot completely take control of the database server. In fact, we assume that our sensors cannot be attacked and can always capture correct traffic information at both ends.

The overall architecture of our system is presented in the figure fig 1.

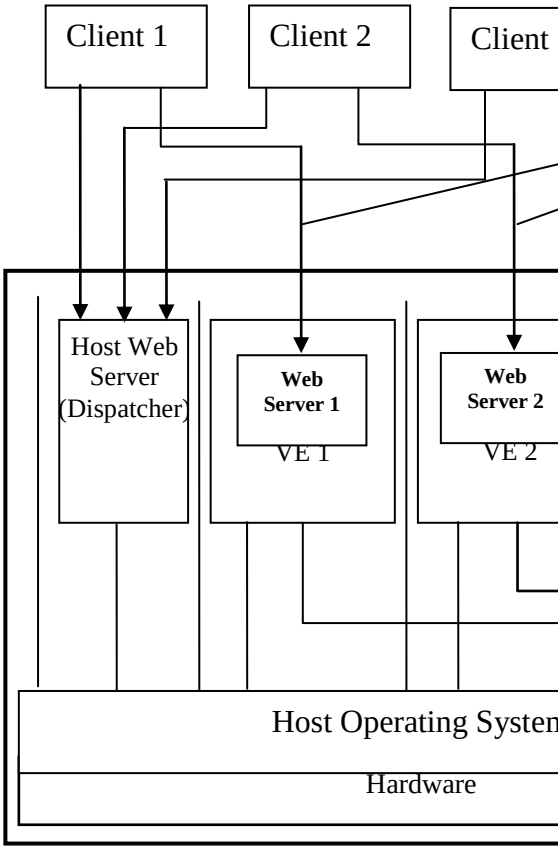

Figure 1: System Architecture

3. Modeling Deterministic Mapping Patterns

3.1 Deterministic Mapping

This is the most common and perfectly matched pattern. That is to say that web request r_m appears in all traffic with the SQL queries set Q_n. For any session in the testing phase with the request r_m, the absence of a query set Q_n matching the request indicates a possible intrusion. On the other hand, if Q_n is present in the session traffic without the corresponding r_m, this may also be the sign of an intrusion.

3.2 Empty Query Set

In special cases, the SQL query set may be the empty set. This implies that the web request neither causes nor generates any database queries. For example, when a web request for retrieving an image GIF file from the same webserver is made, a mapping relationship does not exist because only the web requests are observed. During the testing phase, we keep these web requests together in the set EQS.

3.3 No Matched Request

Unmatched queries in a set NMR are kept. During the testing phase, any query within set NMR is considered legitimate.

4. Static Model Building Algorithm

Require: Training Data set, 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

5. 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 IDSs, Double-Guard forms container-based IDS with multiple input streams to produce alerts. We have deployed using Apache Web server, a blog application, and a MySQL back end, Double-Guard was able to identify a wide range of attacks with minimal false positives.

6. References

1. Anley. (2002) "Advanced Sql Injection in Sql Server Applications," technical report, Next Generation Security Software, Ltd.,
2. Bai K Wang H and Liu P (2005). "Towards Database Firewalls," Proc. Ann. IFIP WG 11.3 Working Conf. Data and Applications Security (DBSec '05).
3. Barry B.I.A. and H.A. Chan (2009). "Syntax, and Semantics-Based Signature Database for Hybrid

Intrusion Detection Systems," Security and Comm. Networks, vol. 2, no. 6, pp. 457-475.

4. Christodorescu M and Jha S (2003) "Static Analysis of Executables to Detect Malicious Patterns," Proc. Conf. USENIX Security Symp.

5. Cova M, Balzarotti D, Felmetsger V, and Vigna G (2007) "Swaddler:An Approach for the Anomaly-Based Detection of State Violations in Web Applications," Proc. Int'l Symp. Recent Advances in Intrusion Detection (RAID '07).

6. Debar H. Dacier M. and Wespi A. (1999) "Towards a Taxonomy of Intrusion-Detection Systems," Computer Networks, vol. 31, no. 9, pp. 805-822.