International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 4, Jul - Aug. 2013 pp-2014-2018 ISSN: 2249-6645

Double guard: Detecting Interruptions in N- Tier Web Applications

P. Krishna Reddy[1], T. Manjula[2], D. Srujan Chandra Reddy[3], T. Dayakar Reddy[4]
[1]*M. Tech (CSE), VITS, Kavali*
[2, 3, 4]*Associate Professor, CSE, VITS, Kavali*

Abstract: *Internet services and applications contain develop into an inextricable part of daily life and make possible communication between the management of personal information from anywhere. To put up 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 is outsourced to a database or file server. Double Guard is an IDS 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 succeeding database requests, we are able to search out attacks that an independent IDS would not be able to identify. In addition, we compute the limitations of any multitier IDS in terms of training sessions and functionality coverage. Then we collected and processed real-world traffic over a system deployment in both dynamic and static web applications.*

Index Terms: *Web services, Multi-Tier web application, Virtualization, SQL Injection, Vulnerable*

I. INTRODUCTION

WEB-DELIVERED services and applications have enlarged in both popularity and complexity over the past few years. Everyday tasks, like banking, social networking, are all done through the web. Such services typically employ a web server front end that runs the application user interface logic as a back-end server that consists of a database or file server. The attacks have recently become more different, 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 (e.g., SQL injection attacks). A surplus of Intrusion Detection Systems (IDSs) currently examine network packets individually within both the web server and the database system In multitier architectures, the back-end database server is often protected at the back a firewall while the web servers are remotely accessible over the Internet. The IDSs cannot detect cases where in normal traffic is used to attack the web server and the database server. Unluckily, within the current multithreaded web server architecture, it is not possible to detect or profile such causal mapping between web server traffic and DB server traffic since traffic cannot be clearly attributed to user sessions.

In this paper, we present DoubleGuard, a system used to detect attacks in multitier web services. Our approach can create customariness models of isolated user sessions that include both the web front-end (HTTP) and back-end (File or SQL) network transactions. To achieve that, we use a lightweight virtualization technique to assign each user's web session to a dedicated container, an isolated virtual computing environment. We use the container ID to accurately associate the web request with the subsequent data base queries. Thus, DoubleGuard can build a fundamental mapping profile by taking both the web server and data base traffic into account. We have implemented our DoubleGuard container architecture using OpenVZ and performance testing shows that it has reasonable performance overhead and is practical for most web applications. When the request rate is moderate (e.g., under 110 requests per second), there is almost no overhead in comparison to an unprotected vanilla system. The container-based web architecture not only fosters the profiling of causal mapping, but it also provides an isolation that prevents future session-hijacking attacks. Within a lightweight virtualization environment, 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 rom 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. Our experimental evaluation, using real-world network traffic obtained from the web and database requests of a large center, showed that we were able to extract 100 percent of functionality mapping by using as few as 35 sessions in the training phase. it does not depend on content changes if those changes can be performed through a controlled environment and retrofitted into the training model. We refer to such sites as "static" because, though they do change over time, they do so in a controlled fashion that allows the changes to propagate to the sites' normality models. In addition to this static website case, there are web services that permit persistent back-end data modifications. These services, which we call dynamic, allow HTTP requests to include parameters that are variable and depend on user input. Sometimes, the same application's primitive functionality (i.e., accessing a table) can be triggered by many different web pages. To address this challenge while building a mapping model for dynamic web pages, we first generated an individual training model for the basic operations provided by the web services. We demonstrate that this approach works well in practice by using traffic from a live blog where we
progressively modeled nine operations. Our results show that we were able to identify all attacks, covering more than 99 percent of the normal traffic as the training model is refined.

II. THREAT MODEL & SYSTEM ARCHITECTURE

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

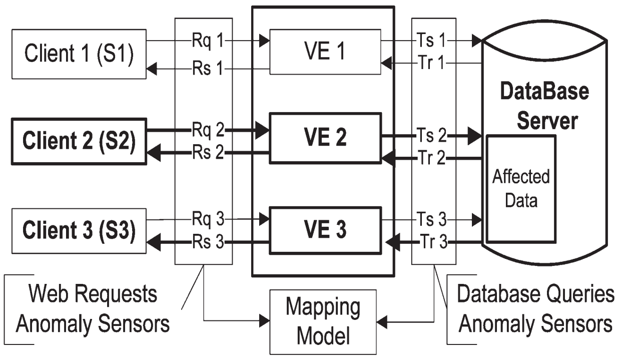

Fig 1: Classic three-tier model

3.1 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. Our sensors will not be attacked at the database server too, as we assume that the attacker cannot completely take control of the database server. 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 exist any request or query that violates the normality model within a session, then the session will be treated as a possible attack.

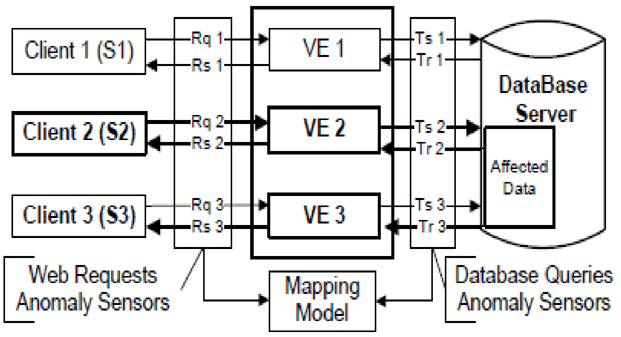

Fig. 2: Web server instances running in containers

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 is any request or query that violates the normality model within a session, then the session will be treated as a possible attack.

3.2 Attack Scenarios

Our approach is effective at capturing the following type of attacks.

3.2.1 Privilege Escalation Attack

Let's assume that the website serves both regular users and administrators. For a regular user, the web request ru will trigger the set of SQL queries Qu; for an administrator, the request ra will trigger the set of admin level queries Qa. Now suppose that an attacker logs into the web server as a normal user, 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 or the database IDS since both ru and Qa are legitimate requests and queries. Our approach, however, can detect this type of attack since the DB query Qa does not match the request ru, according to our mapping model.

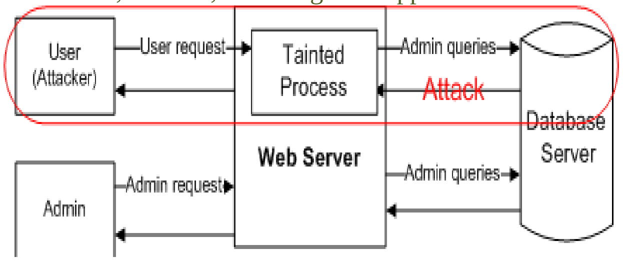

Fig. 3: Privilege Escalation Attack

3.2.2. Hijack Future Session Attack

This type of attacks is mainly aimed at the web server side. An attacker usually takes over the web server and therefore hijacks all subsequent legitimate user sessions to launch attacks. For instance, by hijacking other user sessions, the attacker can eavesdrop, send spoofed replies, and/or drop user requests. A session hijacking attack can be further categorized as a Spoofing/Man-in-the-Middle attack, an Exfiltration Attack, a Denial-of-Service/Packet Drop attack, or a Replay attack. According to the mapping model, the web request should invoke some database queries (e.g., a Deterministic Mapping), then the abnormal situation can be detected.

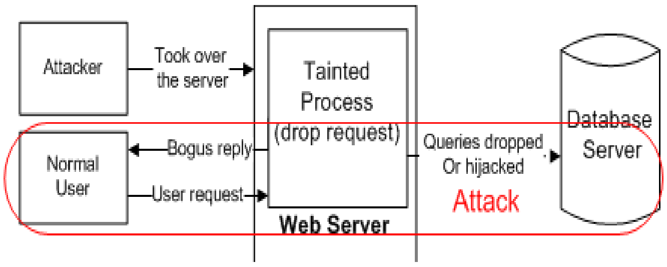

Fig. 4: Hijack Future Session Attack.

3.2.3. Injection Attack

Attacks such as SQL injection do 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.

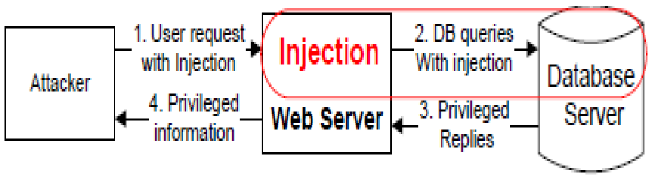
Attacker
1. User request with Injection
Injection
2. DB queries With injection
Database Server
4. Privileged information
Web Server
3. Privileged Replies

Fig. 5: Injection Attack

.

3.2.4. Direct DB Attack

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

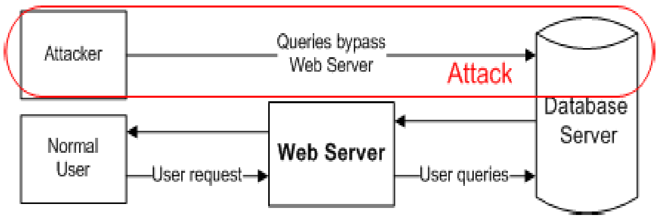

Fig.6: DB Query without causing Web requests.

III. MODELLING MAPPING PATTERNS

Due to their diverse functionality, different web applications exhibit different characteristics. Many websites serve only static content, which is updated and often managed by a Content Management System (CMS).This creates tremendous challenges for IDS system training because the HTTP requests can contain variables in the past parameters. The variable values in both HTTP requests and database queries, preserving the structures of the requests and queries.

4.1 Deterministic Mapping
4.2 Empty Query Set
4.3 No Matched Request
4.4 Nondeterministic Mapping

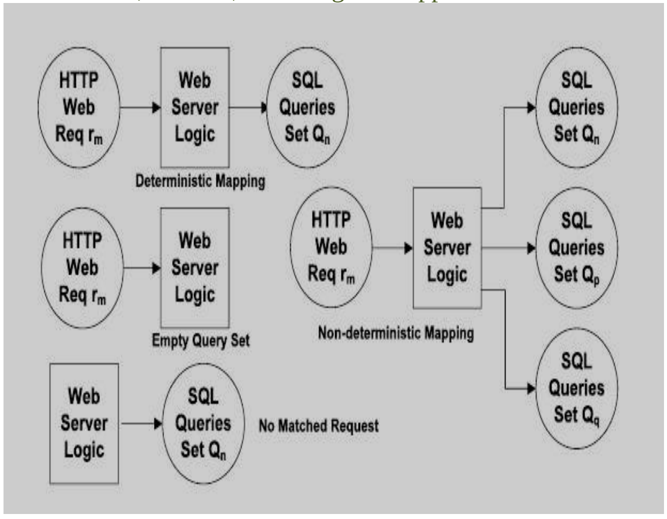

Fig.6: Overall representation of mapping patterns.

IV. MODELING FOR STATIC AND DYNAMIC

In the case of a static website, the nondeterministic 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 €REQ We maintain a set AR_m to record the IDs of sessions in which r_m appears. The same holds for the database queries.

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 webserver' application logic, different inputs would cause different database queries. By placing each rm, or the set 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 hold static file requests.

V. PERFORMANCE EVALUTION

We obtained 329 real user traffic sessions from the blog under daily workloads. During this seven-day phase, we made our website available only to internal users to ensure that no attacks would occur then generated 20 attack traffic sessions mixed with these legitimate sessions, and the mixed traffic was used for detection.

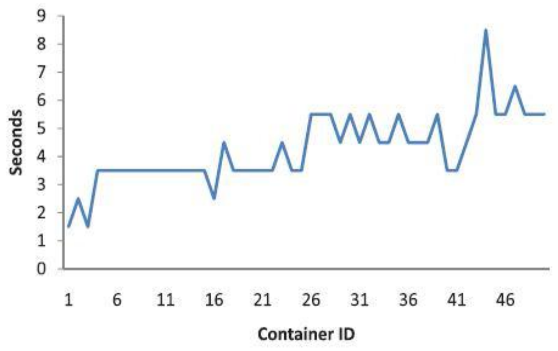

Fig. 7: Time for starting a new container.

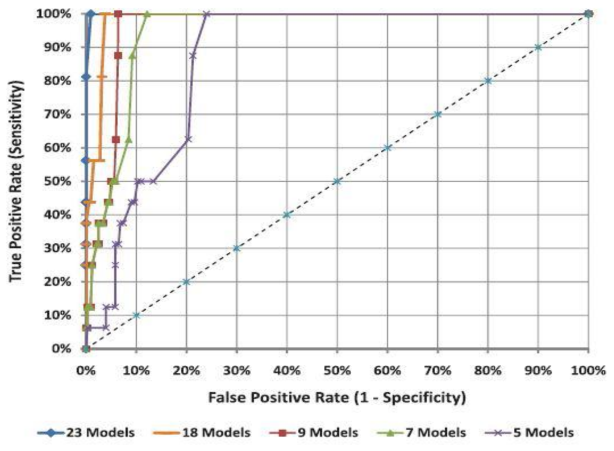

Fig.8. ROC curves for dynamic models.

We model nine basic operations; we can reach 100 percent Sensitivity with six percent False Positive rate. In the case of 23 basic operations, we achieve the False Positive rate of 0.6 percent. By Using Double guard approach we also avoid the following attacks. Privilege Escalation Attack, Hijack Future Session Attack (Webserver-Aimed Attack), Injection Attack.

VI. CONCLUSION

We presented an Intrusion Detection System that builds models of normal behavior for multitier web applications from both front-end web (HTTP) requests and back-end database (SQL) queries. It forms container based IDS with multiple input streams to produce alerts. We have shown that such 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. We achieved this by isolating the flow of information from each web server session with a lightweight virtualization. Furthermore, 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.

REFERENCES

[1] U. Shankar and V. Paxson, “Active Mapping: Resisting NIDS Evasion Without Altering Traffic,” Proc. IEEE Symp. Security and Privacy, 2003.

[2] T.H. Ptacek and T.N. Newsham. Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection Technical report, Secure Networks, January 1998.

[3] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA ’99 Conference*, November 1999.

[4] K. Kendall, “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems,” master’s thesis, MIT, June 1999.

[5] G.H. Kim and E.H. Spafford, “The Design and Implementation of Tripwire: A File System Integrity Checker,” technical report, Purdue Univ.,