To Detect Intrusions in Multitier Web Applications by using Double Guard Approach.

K.Karthika, K.Sripriyadevi

***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 multitiered design wherein the webserver runs the application front-end logic and data are outsourced to a database or file server. In this paper, we present Double Guard, an IDS system that models the network behavior of user sessions across both the front-end webserver and the back-end database. By monitoring both web and subsequent database requests, we are able to ferret out attacks that independent IDS would not be able to identify. Furthermore, we quantify the limitations of any multitier IDS in terms of training sessions and functionality coverage. We implemented DoubleGuard using an Apache webserver with MySQL and lightweight virtualization. We then collected and processed real-world traffic over a 15-day period of system deployment in both dynamic and static web applications. Finally, using DoubleGuard, we were able to expose a wide range of attacks with 100 percent accuracy while maintaining 0 percent false positives for static web services and 0.6 percent false positives for dynamic web services.

***Keywords* -**Anomaly detection, virtualization, multitier web application..

1. Introduction

WEB-DELIVERED services and applications have increased in both popularity and complexity over the past few years. Daily tasks, such as banking, travel, and social networking, are all done via the web. Such services typically employ a webserver front end that runs the application user interface logic, as well as a back-end server that consists of a database or file server. The 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 (e.g., SQL injection attacks). A plethora of Intrusion Detection Systems (IDSs) currently examine network packets individually within both the webserver and the database system. In multitiered architectures, the back-end database server is often protected behind a firewall while the webservers are remotely accessible over the Internet. The IDSs cannot detect cases wherein normal traffic is used to attack the webserver and the database server. Unfortunately, within the current multithreaded webserver architecture, it is not feasible to detect or profile such causal mapping between webserver 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 multitiered web services. Our approach 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, we employ 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 DB queries. Thus, DoubleGuard can build a causal mapping profile by taking both the webserver and DB 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 webserver instances in different containers so that each one was isolated from the rest. Asephemeral 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 webserver 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 webpages. To address this challenge while building a mapping model
for dynamic webpages, 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.

2. Related work.

A network Intrusion Detection System can be classified into two types: anomaly detection and misuse detection. Anomaly detection first requires the IDS to define and characterize. However, we have found that DoubleGuard 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 websever). the correct and acceptable static form and dynamic behavior of the system, which can then be used to detect abnormal changes or anomalous behaviors. Intrusion alerts correlation provides a collection of components that transform intrusion detection sensor alerts into succinct intrusion reports in order to reduce the number of replicated alerts, false positives, and nonrelevant positives. DoubleGuard differs from this type of approach that
correlates alerts from independent IDSs. Rather, Double-Guard operates on multiple feeds of network traffic using a single IDS that looks across sessions to produce an alert without correlating or summarizing the alerts produced by other independent IDSs. DoubleGuard does not have a limitation as it uses the container ID for each session to causally map the related events, whether they be concurrent or not. The system proposed in composes both web IDS and database IDS to achieve more accurate detection, 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 DoubleGuard, the new container-based webserver architecture enables us to separate the different information flows by each session. For the static webpage, our DoubleGuard 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. DoubleGuard focuses on modeling the mapping patterns between HTTP requests and DB queries to detect malicious user sessions. Building the mapping model in DoubleGuard would require a large number of isolated web stack instances so that mapping patterns would appear across different session instances.

3. Threat model and system architecture

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

3.1 Architecture and Confinement

In our design, we make 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.

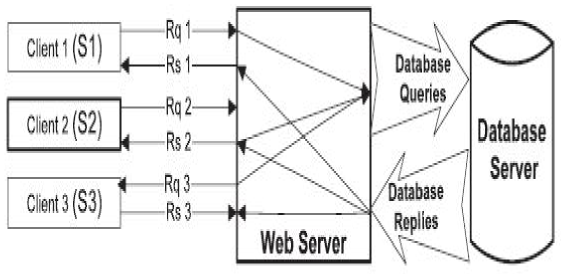


Fig 1. Classic three-tier model. The webserver acts as the front end, with the file and database servers as the content storage back end.

In the classic three-tier model database side, we are unable to tell which transaction corresponds to which client request. The communication between the webserver and the database server is not separated, and we can hardly understand the relationships
among them.

3.2 Building the Normality Model

This container-based and session-separated webserver 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 webserver requests and the subsequent DB queries, and to utilize such a mapping model to detect abnormal behaviors on a session/client level.

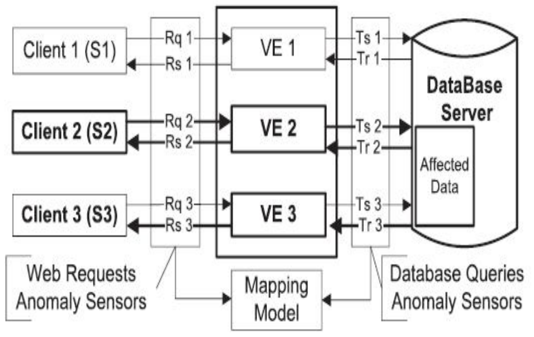


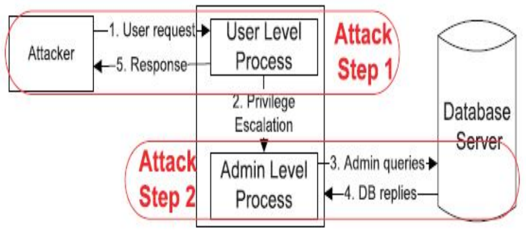
Fig.2. Webserver instances running in containers.

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

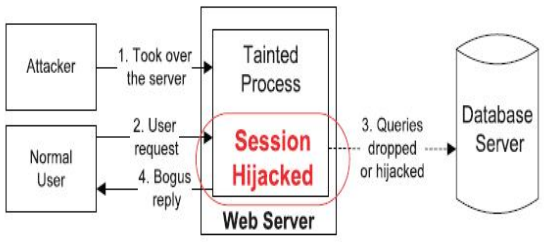
3.3 Attack Scenarios

Our system is effective at capturing the following types of attacks:

3.3.1 Privilege Escalation Attack



3.3.2 Hijack Future Session Attack



3.3.3 Injection Attack

Attacks such as SQL injection do not require compromising the webserver. Attackers can use existing vulnerabilities in the webserver logic to inject the data or string content that contains the exploits and then use the webserver to relay these exploits to attack the back-end database

.

3.3.4 Direct DB Attack

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

4.MODELING DETERMINISTIC MAPPING AND 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 passed parameters. DoubleGuard normalizes the variable values in both HTTP requests and database queries, preserving the structures of the requests and queries. To achieve this, DoubleGuard substitutes the actual values of the variables with symbolic values.

4.1 Inferring Mapping Relations

we classify the four possible mapping patterns. Since the request is at the origin of the data flow, we treat each request as the mapping source. In other words, the mappings in the model are always in the form of one request to a query set rm ! Qn.

4.1.1 Deterministic Mapping

This is the most common and perfectly matched pattern.That is to say that web request rm appears in all traffic with the SQL queries set Qn. The mapping pattern is then rm ! Qn . For any session in the testing phase with the request rm, the absence of a query set Qn matching the request indicates a possible intrusion.

4.1.2 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.

4.1.3 No Matched Request

In some cases, the webserver may periodically submit queries to the database server in order to conduct some scheduled tasks, such as cron jobs for archiving or backup.

4.1.4 Nondeterministic Mapping

The same web request may result in different SQL query sets based on input parameters or the status of the webpage at the time the web request is received. In fact, these different SQL query sets do not appear randomly, and there exists a candidate pool of query sets (e.g., fQn;Qp;Qq . . .g). Therefore, it is difficult to identify traffic that matches this pattern. This happens only within dynamic websites, such as blogs or forum sites.

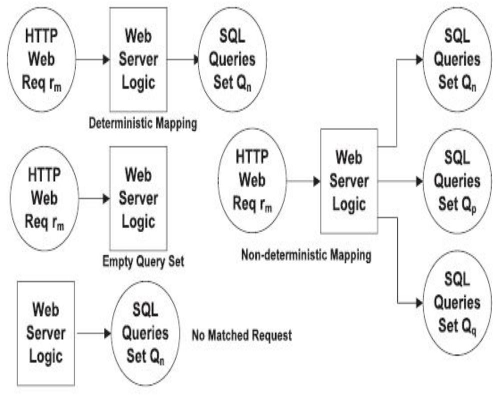


Fig. 3. Overall representation of mapping patterns.

4.2 Modeling for Static Websites

We can easily classify the traffic collected by sensors into three patterns in order to build the mapping model.

Algorithm. Static Model Building Algorithm

```
Require: Training Data set, Threshold t
Ensure: The Mapping Model for static website
 for each session separated traffic Ti do
 Get different HTTP requests r and DB queries q in
 this session
 for each different r do
 if r is a request to static file then
 Add r into set EQS
 else
 if r is not in set REQ then
 Add r into REQ
 Append session ID i to the set ARr with r as
  the key
 for each different q do
 if q is not in set SQL then
 Add q into SQL
 Append session ID i to the set AQq with q as the
 key
 for each distinct HTTP request r in REQ do
 for each distinct DB query q in SQL do
 Compare the set ARr with the set AQq
 if ARr = AQq and Cardinality(ARr) > t then
 Found a Deterministic mapping from r to q
 Add q into mapping model set MSr of r
 Mark q in set SQL
 else
 Need more training sessions
 return False
 for each DB query q in SQL do
 if q is not marked then
 Add q into set NMR
 for each HTTP request r in REQ do
 if r has no deterministic mapping model then
 Add r into set EQS
 return True
```

4.3 Testing for Static Websites

The testing phase algorithm is as follows:
1. If the rule for the request is Deterministic Mapping r -> Q , we test whether Q is a subset of a query set of the session. If so, this request is valid, and we mark the queries in Q. Otherwise, a violation is detected and considered to be abnormal, and the session will be marked as suspicious.
2. If the rule is Empty Query Set r then the request is not considered to be abnormal, and we do not mark any database queries. No intrusion will be reported.
3. For the remaining unmarked database queries, we check to see if they are in the set NMR. If so, we mark the query as such.
4. Any untested web request or unmarked database

query is considered to be abnormal. If either exists within a session, then that session will be marked as suspicious. In our implementation and experimenting of the static testing website, the mapping model contained the Deterministic Mappings and Empty Query Set patterns without the No Matched Request pattern. This is commonly the case for static websites.

4.4 Modeling of Dynamic Patterns.

The algorithm for extracting mapping patterns in static pages no longer worked for the dynamic pages, we created another training method to build the model. First, we tried to categorize all of the potential single (atomic) operations on the webpages. For instance, the common possible operations for users on a blog website may include reading an article, posting a new article, leaving a comment, visiting the next page, etc. All of the operations that appear within one session are permutations of these operations. If we could build a mapping model for each of these basic operations, then we could compare web requests to determine the basic operations of the session and obtain the most likely set of queries mapped from these operations. If these single operation models could not cover all of the requests and queries in a session, then this would indicate a possible intrusion. 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 fQn, Qp, Qq . . .g as possible. We then establish one operation mapping model Rm ! Qm (Qm =Qn [Qp [Qq [. . .), wherein Rm is the set of the web requests for that single operation and Qm includes the possible queries triggered by that operation.

4.5 Detection for Dynamic Websites

Once we build the separate single operation models, they can be used to detect abnormal sessions. We then take the entire corresponding query sets in these models to form the set CQS. For the testing session i, the set of DB queries Qi should be a subset of the CQS. Otherwise, we would find some unmatched queries. For the web requests in Ri, each should either match at least one request in the operation model or be in the set EQS. If any unmatched web request remains, this indicates that the session has violated the mapping model. The model of two single operations such as Reading an article and Writing an Article. The mapping models are READ RQ and WRITE WQ, and we use them to test a given session i. For all the requests in the session, we then find that each of them either belongs to request set READ or WRITE. (You can ignore set EQS here.) This means that there are only two basic operations in the session, though they may appear as any of their permutations. Therefore, the query set Qi should be a subset of RQ WQ, which is CQS.

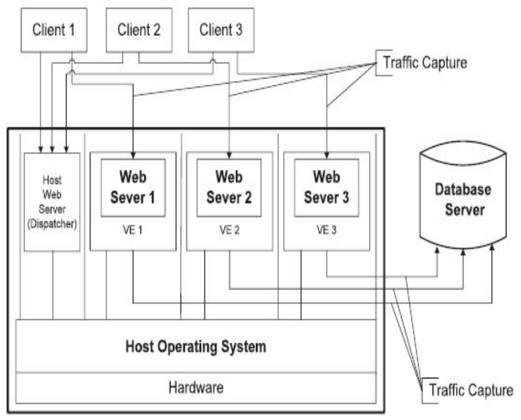


Fig. 4. The overall architecture of our prototype.

5. Performance Evaluation

We implemented a prototype of DoubleGuard using a webserver with a back-end DB. We also set up two testing websites, one static and the other dynamic. To evaluate the detection results for our system, we analyzed four classes of attacks.

5.1 Implementation

In our prototype, we chose to assign each user session into a different container; however, this was a design decision. we could maintain a large number of parallel-running Apache instances similar to the Apache threads that the server would maintain in the scenario without containers. If a session timed out, the Apache instance was terminated along with its container. In our prototype implementation, we used a 60-minute timeout due to resource constraints of our test server.

Static Website Model in Training Phase

For the static website, we used to build the mapping model, and we found that only the Deterministic Mapping and the Empty Query Set Mapping patterns appear in the training sessions. We expected that the No Matched Request pattern would appear if the web application had a cron job that contacts back-end database server; however, our testing website did not have such a cron job. We first collected 338 real user sessions for a training data set before making the website public so that there was no attack during the training phase. Based on this training process accuracy graph, we can determine a proper time to stop the training.

Dynamic Modeling Detection Rates

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.

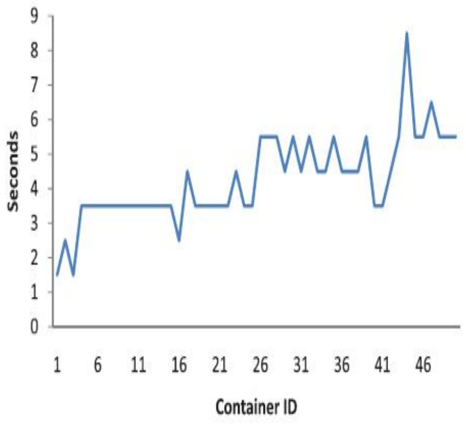


Fig. 5. Time for starting a new container.

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.

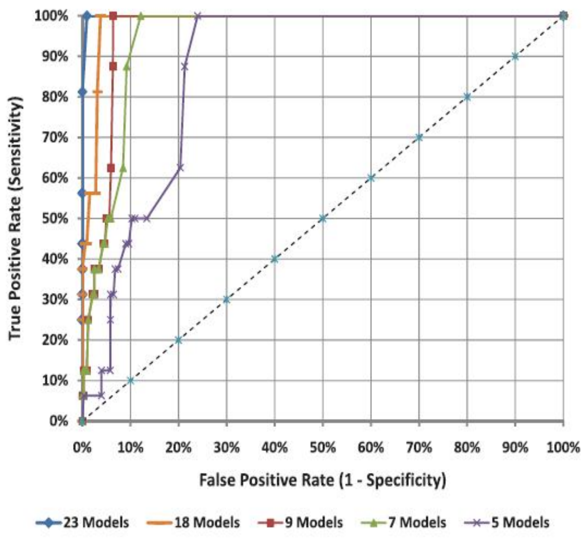


Fig.6. ROC curves for dynamic models.

6 CONCLUSION

We presented an intrusion detection system that builds models of normal behavior for multitiered 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, DoubleGuard forms a 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.
Weachieved this by isolating the flow of information from each webserver 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. For static websites, we built a well-correlated model, which our experiments proved to be effective at detecting different types of attacks. 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. When we deployed our prototype on a system that employed Apache webserver, a blog application, and a MySQL back end, DoubleGuard was able to identify a wide range of attacks with minimal false positives. As expected, the number of false positives depended on the size and coverage of the training sessions we used. Finally, for dynamic web applications, we reduced the false positives to 0.6 percent.

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., Nov. 1993.

[6] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-

[7] Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module*. 2550 Garcia Ave., Mountain View, CA 94043, December 1991.

[8]"Five Common Web Application Vulnerabilities,"http://www.symantec.com/connect/articles /five-common-web-application vulnerabilities,2011.

[9] Snort—The Open Source Network Intrusion Detection System, http://www.snort.org, 2004.

[10]S.J.Templeton and K. Levitt, "A Requires/Provides Model for Computer Attacks," Proc. New Security Paradigms Workshop, pp. 31-38, Sept. 2000.

[11] SANS, "The Top Cyber Security Risks," http://www.sans.org/top-cyber-security-risks/,2011.

[12] National Vulnerability Database, "Vulnerability SummaryforCVE-2010-4332," http://web.nvd.nist.gov/view/vuln/detail?vulnId= CVE-2010-4332, 2011.

[13] Lap Chung Lam and Tzi cker Chiueh. A general dynamic information flow tracking framework for security applications. In Annual Computer Security Applications Conference (ACSAC), 2006.

[14] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(707), 1966.

Based Approach," Proc. 1997 IEEE Symp. Security and Privacy, pp. 175-187, May 1997.

Author 1

K.Karthika, Received B.E (CSE) degree in 2010 from Anna University Chennai India .Received M.E(CSE) degree in Karpagam University, India. Currently Working as Assistant Professor in EASA college of Enggineering and Technology.coimbatore.

Author 2

K.Sripriyadevi, received B.E degree in 2010 from Anna University Chennai India. Currently pursuing M.E degree in Computer Science and Engineering in velalar college of Engineering and Technology.Erode.