ISSN:0975-9646

An Efficient Guarding by Detecting Intrusions in Multi-Tier Web Applications

A Yugandhara Rao[1], Meher Divya Tatavarthi[2], S P Ravi Teja Yeeramilli[2], Mohan Raj Simhadri[2], Bhadur Sayyad[2]

[1]Asstistant Professor, [2]B.Tech student
Computer Science & Engineering, Lendi Institute of Engineering and Technology (LIET)
Vizianagaram, A.P, India

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 multitier design wherein the web server runs the application front-end logic and data are outsourced to a database or file server. In this paper, we present Efficient Guarding, 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 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 Efficient Guarding using an Apache web server 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 Efficient Guarding, 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; multitier web application; virtualization*

I. INTRODUCTION

Flaws or Drawbacks with Existing System:

- In existing system 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.
- Attacker may take over the web server after the attack, and that afterwards 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.

Proposed System:

Some previous approaches have detected intrusions or vulnerabilities by statically analyzing the source code or executables. Others dynamically track the information flow to understand taint propagations and detect intrusions. In an Efficient Guarding, the new container-based web server architecture enables us to separate the different information flows by each session. This provides a means of tracking the information flow from the web server to the database server for each session. Our approach also does not require us to analyze the source code or know the application logic. For the static web page, our Efficient Guarding 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.

Advantages of Proposed Systems:

- The proposed system is well-correlated model that provides an effective mechanism to detect the different types of attacks.
- The proposed system will also create a causal mapping profile by taking both the web server and DB traffic into account.
- It provides a better characterization for anomaly detection with the correlation of input streams because the intrusion sensor has a more precise normality model that detects a wider range of threats.
- The proposed system will also isolate the flow of information from each web server session with a lightweight virtualization.

Problem Statement:

Lot of existing intrusion Detection Systems (IDSs) examines the 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. In order to protect multi tiered web services, an efficient system called as Intrusion detection systems is needed to detect known attacks by matching misused traffic patterns or signature.

II. RELATED WORK

A network Intrusion Detection System (IDS) can be classified into two types: anomaly detection and misuse detection .Anomaly detection first requires the IDS to define and characterize the correct and acceptable static form and dynamic behavior of the system, which can then be used to detect abnormal changes or anomalous behaviors . The boundary between acceptable and anomalous forms of stored code and data is precisely definable. Behavior models are built by performing a statistical analysis on historical data or by using rule-based approaches to specify behavior patterns. An anomaly detector then compares actual usage patterns against established models to identify abnormal events. Our detection approach belongs to anomaly detection , and we depend on a training phase to build the correct model. As some legitimate updates may cause model drift, there area number of approaches that are trying to solve this problem. Our detection may run into the same problem; in such a case, our model should be retrained for each shift. 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 non-relevant positives. It also fuses the alerts from different levels describing a single attack, with the goal of producing a succinct overview of security-related activity on the network. It focuses primarily on abstracting the low-level sensor alerts and providing compound, logical, high-level alert events to the users. An Efficient Guarding differs from this type of approach that correlates alerts from independent IDSes. Rather, An Efficient Guarding operates on multiple feeds of network traffic using single IDS that looks across sessions to produce an alert without correlating or summarizing the alerts produced by other independent IDSs. An IDS such as also uses temporal information to detect intrusions. An Efficient Guarding, however, does not correlate events on a time basis, which runs the risk of mistakenly considering independent but concurrent events as correlated events. An Efficient Guarding 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. Since databases always contain more valuable information, they should receive the highest level of protection. Therefore, significant research efforts have been made on database IDS and database firewalls. These softwares , such as Green SQL, work as a reverse proxy for database connections. Instead of connecting to a database server, web applications will first connect to a database firewall. SQL queries are analyzed; if they're deemed safe, they are then forwarded to the back-end database server. 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 such cases, there would be no alerts to correlate. Some previous approaches have detected intrusions or vulnerabilities by statically analyzing the source code or executables dynamically track the information flow to understand taint propagations and detect intrusions. In An Efficient Guarding, the new container-based web server architecture enables us to separate the different information flows by each session. This provides a means of tracking the information flow from the web server to the database server for each session. Our approach also does not require us to analyze the source code or know the application logic. For the static web page, our An Efficient Guarding 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 . In addition, validating input is useful to detect or prevent SQL or XSS injection attacks. This is orthogonal to the An Efficient Guarding approach, which can utilize input validation as an additional defense. However, we have found that An Efficient Guarding 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 isolate objects and enhance security performance. Full virtualization and para-virtualization are not the only approaches being taken. An alternative is a lightweight virtualization, such as OpenVZ, Parallels Virtuozzo, or Linux-VServer . In general, these are based on some sort of container concept. With containers, a group of processes still appears to have its own dedicated system, yetit is running in an isolated environment. On the other hand, lightweight containers can have considerable performance advantages over full virtualization orpara-virtualization. Thousands of containers can run on a single physical host. There are also some desktop systems , that use light weight virtualization to isolate different application instances. Such virtualization techniques are commonly used for isolation and containment of attacks. However, in our An Efficient Guarding, 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 is an architecture for preventing data leaks even in the presence of attacks. By isolating code at the web server layer and data at the database layer by users, CLAMP guarantees that a user's sensitive data can only be accessed by code running on behalf of different users. In contrast, An Efficient Guarding focuses on modeling the mapping patterns between HTTP requests and DB queries to detect malicious user sessions. There are additional differences between these two in terms of requirements and focus. CLAMP requires modification to the existing application code, and the Query Restrictor works as a proxy to mediate all database access requests. Moreover, resource requirements and overhead differ in order of magnitude: An Efficient Guarding uses process isolation whereas CLAMP requires platform virtualization, and CLAMP provides more coarse-grained isolation than An Efficient Guarding . However, An Efficient Guarding would be ineffective at detecting attacks if it were to use the coarse-grained isolation as used in CLAMP. Building the mapping model in An Efficient Guarding would require a large number of isolated web stack instances so that mapping patterns would appear across different session instances.

III. ATTACKS

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 1 describes Privilege Escalation Attack.

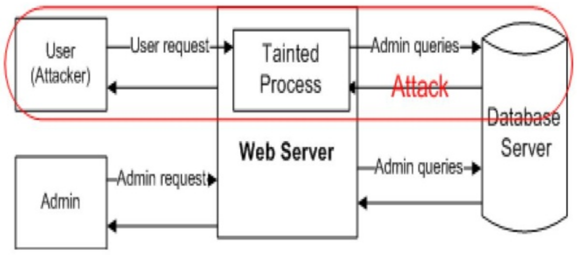


Fig 3.1 *Privilege Escalation Attack*

2. Hijack Future Session Attack:

This class 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 Ex-filtration 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. However, neither a conventional web server IDS nor a database IDS can detect such an attack by itself. Fortunately, the isolation property of our container-based web server architecture can also prevent this type of attack. As each user's web requests are isolated into a separate container, an attacker can never break into other users' sessions.

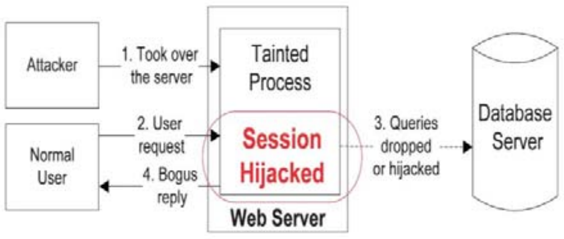


Fig 3.2 Hijack Future Session Attack

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. Since our approach provides a two-tier detection, even if the exploits are accepted by the web server, the relayed contents to the DB server would not be able to take on the expected structure for the given web server request. For instance, since the SQL injection attack changes the structure of the SQL queries, even if the injected data were to go through the web server side, it would generate SQL queries in a different structure that could be detected as a deviation from the SQL query structure that would normally follow such a web request.

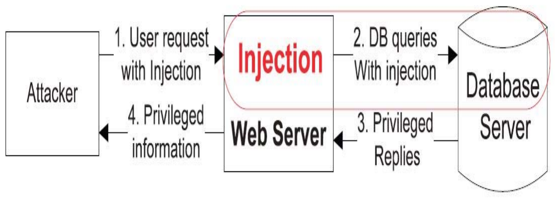

Attacker
1. User request with Injection
Injection
Web Server
2. DB queries With injection
Database Server
4. Privileged information
3. Privileged Replies

Fig 3.3 Injection Attack

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 neither. Furthermore, if these DB queries were within the set of allowed queries, then the database IDS itself would not detect it either. However, this type of attack can be caught with our approach since we cannot match any web requests with these queries.

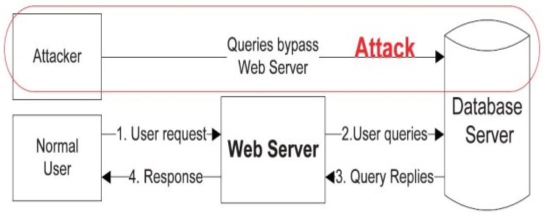


Fig 3.4. Direct DB Attack

IV. ARCHITECTURE

All network traffic, from both legitimate users and adversaries, is received intermixed at the same web server. If an attacker compromises the web server, he/she can potentially affect all future sessions (i.e., session hijacking). Assigning each session to a dedicated web server is not a realistic option, as it will deplete the web server resources. To achieve similar confinement while maintaining a low performance and resource overhead, we use lightweight virtualization. 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. 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.

We choose 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. If we detect abnormal behavior in a session, we will 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. We can thus ensure that legitimate sessions will not be compromised directly by an attacker.

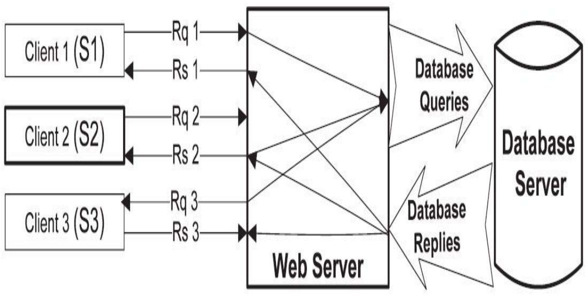


Fig 4.1 classic three tier architecture

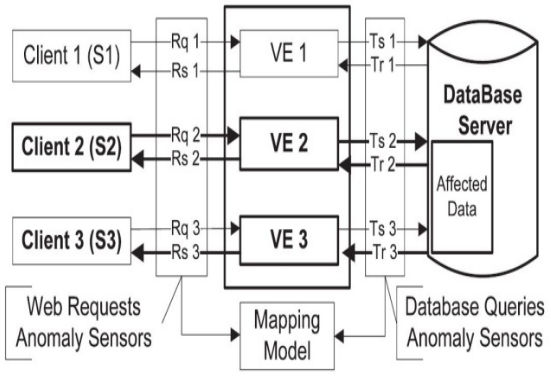


Fig 4.2 web server instance running in containers

In *Fig 4.1* at the database side, we are unable to tell which transaction corresponds to which client request. The communication between the web server and the database server is not separated, and we can hardly understand the relationships among them. Figure 4.2 depicts how communications are categorized as sessions and how database transactions can be related to a corresponding session. According to classic three tier architecture, if Client 2 is malicious and takes over the web server, all subsequent database transactions become suspect, as well as the response to the client. By contrast, according to the figure of web server instance running in containers, Client 2 will only compromise the VE 2, and the corresponding database transaction set T2 will be the only affected section of data within the database.

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

In practice, we are unable to build such mapping under a classic three-tier setup. 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. Section 4 further discusses how to build the mapping by profiling session traffics. 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 In figure of web server instance running in containers shows the locations of our sensors. 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.

V. METHODOLOGY

Modeling Mapping Patterns

Due to the assorted functionality, different web applications exhibit different characteristics. Some websites allow regular users with the non- administrative privileges to update the contents of the server data. This creates challenge for IDS system because the HTTP requests can contain variables in the passed parameters .Our approach normalizes the variable values in both HTTP requests and database queries, preserving 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 (REQ) and the set of total SQL queries (SQL)across all sessions. Each single web request r_m REQ may also appear several times in different R_i where i = 1,2 . . . N. The same holds true for q_n SQL 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. The mappings in the model are always in the form of one request to a query set r_mQ_n. The possible mapping patterns are Deterministic Mapping, Empty Query Set, No Matched Request, and Nondeterministic Mapping.

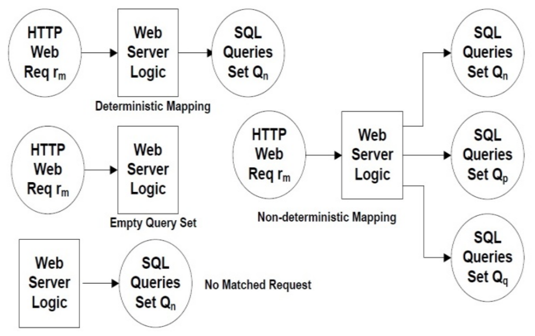


Fig 5.1 Overall Representation of mapping patterns

Modelling For Static And Dyanamic Websites :

In the case of a static website, the nondeterministic mapping does not exist as there are no available inputs 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 REQ, 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 web pages, dynamic web pages 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.

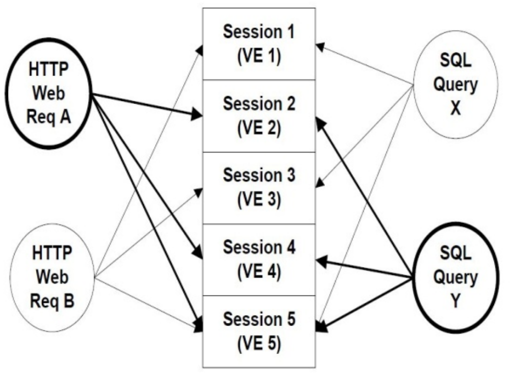


Fig. 5.2 Deterministic Mapping Using Session ID of the Container

Static Model Building Algorithm.

```
Require: Training Dataset, Threshold t
Ensure: The Mapping Model for static website
1: for each session separated traffic $T_i$ 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 $AR_r$ 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 $AQ_q$ 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 $AR_r$ with the set $AQ_q$
17: if $AR_r = AQ_q$ and Cardinality($AR_r$) > t then
18: Found a Deterministic mapping from r to q
19: Add q into mapping model set $MS_r$ 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
```

VI .PERFORMANCE EVALUATION

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 behaviour for each of the session for a different set of users. The following represents the implementation of our prototype and the attack detection rates.

Prototype Implementation

In our design, we choose to assign every user session into a different holder which was the security design decision. Each and every new client (IP address) is assigned to a new holder and these holders 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.

The below figure depicts 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 15 unique web pages and collected real traffic to this website and obtained 350 user sessions. In the case of the dynamic websites, the site visitors are allowed to read , post and comment on articles.

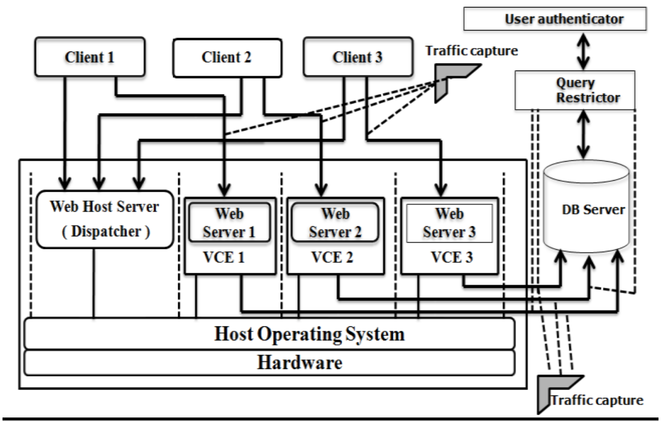


Fig. 6.1 the Overall Architecture of Our Prototype

Query Restrictor:

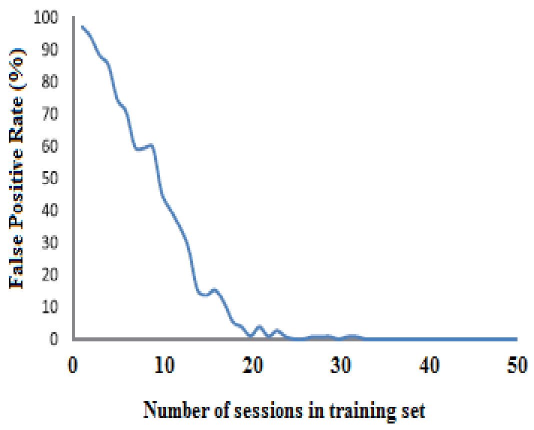
The Query Restrictor (QR) is a trusted module that restricts "Web Server Virtual Machine"
access to sensitive database content. In our implementation, the QR is a specialized SQL proxy that interposes on all databases traffic without requiring changes to the Web Server. When Web Server Virtual Machine acting on behalf of a user attempts to connect to the database, the QR instead connects the client to a separate restricted database tailored specifically to that user**.** Prevents one user from retrieving another users data from the database.

User Authenticator:

It binds the executing server code to the users identity. Because the Web Server
Virtual Machine is entrusted, our approach provides UA, to authenticate users. For instance, the UA may check
that the supplied password matches the users entry in the database, which the UA accesses via the QR.

Static Website Model in Training Phase

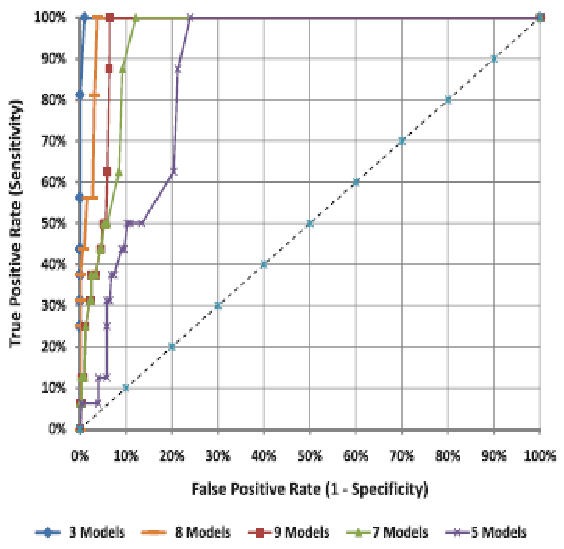
For the static website, Deterministic Mapping and the Empty Query Set Mapping patterns appear in the training
Sessions . We first collected 150 real user sessions for a training data set before making the website public so that there was no attack during the training phase. We used part of the sessions to train the model and all
the remaining sessions to test it. For each number on the x-axis of the following figure, 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 number10 times and obtained the average false positive rate (since there was no attack in the training data set).



The above diagram 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 35 sessions, the false positive rate decreased and stayed at 0. This implies that for our testing static website, 35 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.

Dynamic Model Detection Rates

We also conducted model building experiments for the dynamic blog website. We obtained 215 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 10 attack traffic sessions mixed with the normal legitimate user session, hence the mixed traffic is used for the attack detection.



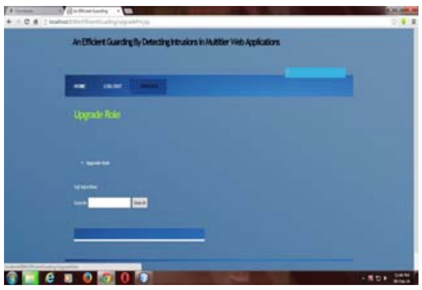
The above figure shows the ROC curves for the testing result. We built our models with different number of operations, and each point on the curves indicates the threshold value. The threshold value is defined as the number of HTTP requests or SQL queries in a session that are not matched with the normality 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.

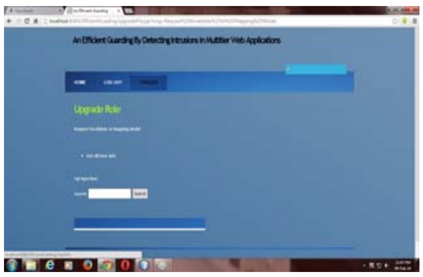
Conclusion

We presented an intrusion detection system that builds models of normal behavior for multi tired web applications from both the front-end web (HTTP) requests and back-end database (SQL) queries. Correlation of the input streams provides a better characterization of the system for anomaly detection because the intrusion sensor has a precise normality model that detects a wide range of attacks. We achieved this by isolating the information flow from each web server session with a virtualization technique. For static websites, we built a model which proved to be effective at detecting different types of attacks. Hence, we are able to identify a wide range of attacks with minimal false positives. The False positive rates for the static and dynamic websites are 0 and 0.7 respectively.

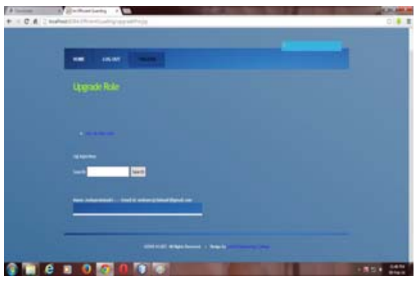
Results

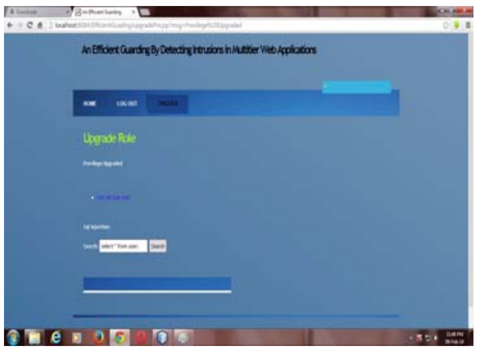
1.Privilege Escalation Attack

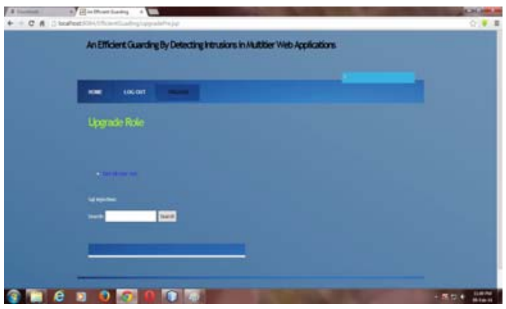

An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Upgrade Role


An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Upgrade Role

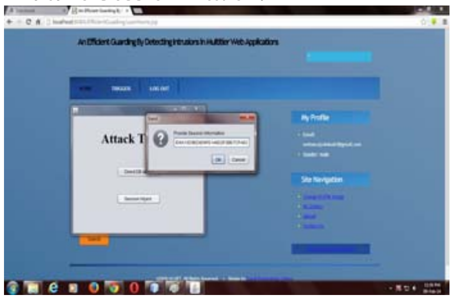
2. Injection Attack:

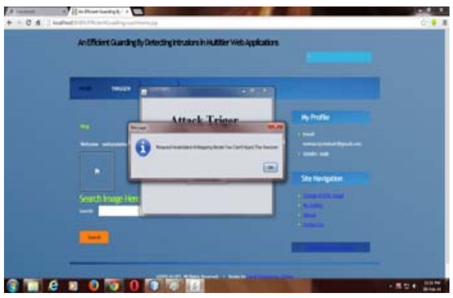



An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Upgrade Role


An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Upgrade Role

3. Hijack Future Session Attack:


An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Attack T
My Profile
Site Navigation


An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Attack Triger
My Profile
Site Navigation
Search Image Her

4. Direct DB Attack:

An Efficient Guarding By Detecting Intrusions In Multitier Web Applications
Attack Triger
My Profile
Site Navigation

REFERENCES

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

[2] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach." In Proceedings of the 1997 IEEE Symposium on Security and Privacy, pages 175to187, May 1997.

[3] "Common Vulnerabilities and Exposures", http://www.cve.mitre.org/,2011

[4] B.I.ABarry and H.A. Chan,"Syntax,and semantics-Based Signature Database for Hybrid Intrusion Detection Systems,"Security and Comm. Networks, vol. 2, no. 6, pp. 457- 475, 2009.

[5] J. Newsome, B. Karp, and D.X. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms," Proc. IEEE Symp.Security and Privacy, 2005.

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

[7] A.K. Ghosh, J. Wanken, and F. Charron. , "Detecting Anomalous and Unknown Intrusions Against Programs". In Proceedings of the Annual Computer Security Applications Conference (ACSAC'98), pages 259to267, Scottsdale, AZ, December 1998.

[8] D.E.Denning.,"An Intrusion Detection Model". IEEE Transactions on Software Engineering, 13(2):222to232,February 1987.

[9] T. Lane and C.E. Brodley. "Temporal sequence learning and data reduction for anomaly detection". InProceedings of the 5th ACM conference on Computer and communications security, pages 150 to158. ACM Press, 1998.

[10] A. Schulman, "Top 10 Database Attacks," http://www.bcs.org/server.php?show=ConWebDoc.8852, 2011.

[11] C. Kruegel and G. Vigna, "Anomaly Detection of Web-Based Attacks," Proc. 10th ACM Conf. Computer and Comm. Security (CCS "03), Oct. 2003.

[12] H.-A. Kim and B. Karp, "Autograph: Toward Automated Distributed Worm Signature Detection," Proc. USENIX Security Symp., 2004.

[13] T. Pietraszek and C.V. Berghe, "Defending against Injection Attacks through Context-Sensitive String Evaluation," Proc. Int"lSymp. Recent Advances in Intrusion Detection (RAID "05), 2005.

[14] G.E. Suh, J.W. Lee, D. Zhang, and S. Devadas, "Secure Program Execution via Dynamic Information Flow Tracking," ACM SIGPLAN Notices, vol. 39, no. 11, pp. 85-96, Nov. 2004.

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

[16] G. Vigna, W.K. Robertson, V. Kher, and R.A. Kemmerer, "A Stateful Intrusion Detection System for World-Wide Web Servers,"Proc. Ann. Computer Security Applications Conf. (ACSAC "03),2003.

[17] A. Seleznyov and S. Puuronen, "Anomaly Intrusion Detection Systems: Handling Temporal Relations between Events," Proc Int"l Symp. Recent Advances in Intrusion Detection (RAID "99), 1999.

[18] M. Roesch,"Snort, Intrusion Detection system," www.snort.org/2011.

[19] W. Lee, S.J. Stolfo, "Data Mining Approaches for Intrusion Detection", Proceedings of the USENIX Security Symposium, pp. 79-94 (1998).

[20] Liang and Sekar, "Fast and Automated Generation of Attack Signatures: A Basis for Building self-Protecting Servers," SIGSAC:Proc. 12th ACM Conf. Computer and Comm. Security, 2005.

[21] Meixing Le, AngelosStavrou, BrentByungHoon Kang," DoubleGuard: Detecting Intrusions in Multitier Web Applications", IEEE transactions on dependable and secure computing, vol. 9, no. 4,July/august 2012.

[22] D. Bates, A. Barth, and C. Jackson, "Regular Expressions Considered Harmful in Client-Side XSS Filters," Proc. 19th Int"lConf. World Wide Web, 2010.