Comparative Analysis of Anomaly based Intrusion Detection Systems in Multi-tier Web Applications

Venkanna Thejavath¹, Vemula Manoj kumar², Madanna Bomme³

¹Assistant Professor, Department of Computer Science, Malla Reddy Institute of Technology and Science, Hyderabad. Andhra Pradesh, India.

²Assistant Professor, Department of Information Technology, Malla Reddy Institute of Technology and Science, Hyderabad. Andhra Pradesh, India.

³Assistant Professor, Department of Computer Science, Malla Reddy Institute of Technology and Science, Hyderabad. Andhra Pradesh, India.

Abstract: Now days, web applications and its services are useful in communication and the management of personal information from anywhere and 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 is outsourced to a database or file server. Our strategy is mainly focus on to detect intrusion at both front end and back end of web application and it is also able to detect intrusion in static and dynamic web application. IDS having maximum accuracy and is mainly responsible to identify intrusion. With the advent of anomaly-based intrusion detection systems, many approaches and techniques have been developed to track novel attacks on the systems. Though anomaly-based approaches are efficient, signature-based detection is preferred for mainstream implementation of intrusion detection systems. As a variety of anomaly detection techniques were suggested, it is difficult to compare the strengths, weaknesses of these methods.

This paper elaborates the foundations of the main anomaly based network intrusion detection technologies along with their operational architectures and also presents a classification based on the type of processing that is related to the “behavioral” model for the target system. The most significant open issues regarding Anomaly based Network Intrusion Detection systems are identified, among which assessment is given particular. By monitoring both web and subsequent database requests, we are able to ferret out attacks that independent IDS would not be able to identify.

Keywords: Intrusion Detection System, Container Architecture, Container ID, Pattern Mapping.

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 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 (e.g. SQL injection attacks). Embarrassments of Intrusion Detection Systems (IDS) 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 (IDS) 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. These IDS cannot detect cases wherein normal traffic is used to attack the web server and the database server. For example, if an attacker with non-admin privileges can log in to a web server using normal-user access credentials, he/she can find a way to issue a privileged database query by exploiting vulnerabilities in
the web server. Neither the web IDS nor the database IDS would detect this type of attack since the web IDS would merely see typical user login traffic and the database IDS would see only the normal traffic of a privileged user. This type of attack can be readily detected if the database IDS can identify that a privileged request from the web server is not associated with user-privileged access. Unfortunately, within the current multi-threaded web server architecture, it is not feasible 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, intricate the foundations of the main anomaly based network intrusion detection technologies along with their operational architectures and also present a classification based on the type of processing that is related to the “behavioral” model for the target system. It also describes the main features of several IDS systems/platforms that are currently available in a concise manner. The most significant open issues regarding Anomaly based Network Intrusion Detection systems are identified, among which assessment is given particular. 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.

Using this prototype, it shows that, for websites that do not permit content modification from users, there is a direct causal relationship between the requests received by the frontend web server and those generated for the database backend. Therefore, ability to model the causal relationship between the front-end and back-end is not always deterministic and depends primarily upon the application logic. For instance, we observed that the back-end queries can vary based on the value of the parameters passed in the HTTP requests and the previous application state. Sometimes, the same application’s primitive functionality (i.e., accessing a table) can be triggered by many different web pages. Therefore, the resulting mapping between web and database requests can range from one to many, depending on the value of the parameters passed in the web request.

2. RELATED WORK

Foremost Anomaly detection requires the IDS to define and describe the correct and acceptable static form and dynamic behavior of the system, which can then be used to detect abnormal changes or anomalous behaviors [1], [2]. 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 [3], [4], [5] or by using rule-based approaches to specify behavior patterns [6]. An anomaly detector then compares actual usage patterns against established models to identify abnormal events.

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 multitier Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions. In such multitier 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 multitier web services, an efficient system called as Intrusion detection systems is needed to detect known attacks by matching misused traffic patterns or signatures. IDSs are mostly used to perform security monitoring of the network infrastructure.

A network Intrusion Detection System (IDS) can be classified into two types: anomaly detection and misuse detection. An alert is generated when an attack is detected and this alert is used to describe the type of attack and the entities that are involved in it(e.g.-hosts, processes, users). IDS can perform focused analysis of the audit data and they are used to produce incorrect or wrong detections. The actions that are taken in a given environment are dynamically monitored by IDS and it also decides that whether these actions are permissible in the given environment.

This detection approach belongs to anomaly detection, and it is depending on a training phase to build the correct model. As some legitimate updates may cause model drift, there are a number of approaches [7] that are trying to solve this problem. Intrusion alerts correlation [8] 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.

Fig. 1.1 Simple Intrusion Detection Systems

Fig. 2.1 Three tier Architecture
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.

3. THREAT MODEL & SYSTEM ARCHITECTURE

In this model, include the assumptions and types of attacks aim to protect against. 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 and 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 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.

On the other hand, at the database end, assume 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. Assume no prior knowledge of the source code or the application logic of web services deployed on the web server. In addition, we are analyzing only network traffic that reaches the web server and database. We assume that no attack would occur during the training phase and model building.

**Architecture & Confinement:** The web server acts as the front-end, with the file and database servers as the content storage back-end. 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 this design, the use of lightweight process containers, referred to as “containers,” as ephemeral, disposable servers for client sessions.

![Fig. 3.1 Classic 3-tier Model.](image)

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. In this 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; initialize each virtualized container using a read-only clean template and guarantee that each session will be served with a clean web server instance at initialization. Choose a 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 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 abnormal behavior in a session was detected and it treats all traffic within this session as tainted. If an attacker compromises a vanilla web server, other sessions’ communications can also be hijacked. In this, 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 and ensure that legitimate sessions will not be compromised directly by an attacker.

In practice, sometimes is not able to build such mapping under a classic 3-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.

To that end, keep the 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. Figure 3.2 shows the locations of our sensors.

Once the mapping model was build then 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.

**Attack scenarios**

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

- **Privilege Escalation Attack**: Let’s assume that the website serves both regular users and administrators. For a regular user, the web request $r_u$ will trigger the set of SQL queries $Q_u$; for an administrator, the request $r_a$ will trigger the set of admin level queries $Q_a$. 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 $r_u$ and $Q_u$ are legitimate requests and queries. However, this approach can detect this type of attack since the DB query $Q_a$ does not match the request $r_u$, according to the mapping model. Figure 3.3 shows how a normal user may use admin queries to obtain privileged information.

- **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 Exfiltration Attack, a Denial-of-Service/Packet Drop attack, or a Replay attack.

Figure 3.4 illustrates a scenario wherein a compromised web server can harm all the Hijack Future Sessions by not generating any DB queries for normal user requests.

According to the mapping model, the web request should invoke some database queries, 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.

- **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 this 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. Figure 3.5 shows the injection attacks in sql.

- **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. However, this type of attack can be caught with this approach since to illustrate the scenario wherein an attacker bypasses the web server to directly query the database. Figure 3.6 shows the direct DB attack.
Vulnerabilities Due to Improper Input Processing: Cross Site Scripting (XSS) is a typical attack method wherein attackers embed malicious client scripts via legitimate user inputs. In Double Guard, the entire user input values are normalized so as to build a mapping model based on the structures of HTTP requests and DB queries. Once the malicious user inputs are normalized, Double Guard cannot detect attacks hidden in the values. These attacks can occur even without the databases. It offers a complementary approach to those research approaches of detecting web attacks based on the characterization of input values.

Within the same session that the attacker connects to, it is allowed for the attacker to launch “mimicry” attacks. It is possible for an attacker to discover the mapping patterns by doing code analysis or reverse engineering, and issue “expected” web requests prior to performing malicious database queries. However, this significantly increases the efforts for the attackers to launch successful attacks. In addition, users with non-admin permissions can cause minimal (and sometimes zero) damage to the rest of the system and therefore they have limited incentives to launch such attacks.

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). For a static website, we can build an accurate model of the mapping relationships between web requests and database queries since the links are static and clicking on the same link always returns the same information. However, some websites (e.g., blogs, forums) allow regular users with non-administrative privileges to update the contents of the server data. This creates tremendous challenges for IDS system training because the HTTP requests can contain variables in the formal parameters.

For example, instead of one-to-one mapping, one web request to the web server usually invokes a number of SQL queries that can vary depending on type of the request and the state of the system. Some requests will only retrieve data from the web server instead of invoking database queries, meaning that no queries will be generated by these web requests. In other cases, one request will invoke a number of database queries. Finally, in some cases, the web server will have some periodic tasks that trigger database queries without any web requests driving them. The challenge is to take all of these cases into account and build the normality model in such a way that we can cover all of them.

As illustrated, all communications from the clients to the database are separated by a session and assign each session with a unique session ID. Double Guard normalizes the variable values in both HTTP requests and database queries, preserving the structures of the requests and queries. To achieve this, Double Guard substitutes the actual values of the variables with symbolic values.

A. Inferring Mapping Relations

The four possible mapping patterns are used. Since the request is at the origin of the data flow, we treat each request as the mapping source. The possible mapping patterns are as follows and shown in the figure of 4.1.

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_m. For any session in the testing phase with the request r_m, the absence of a query set Q_m matching the request indicates a possible intrusion. On the other hand, if Q_m is present in the session traffic without the corresponding r_m, this may also be the sign of an intrusion. In static websites, this type of mapping comprises the majority of cases since the same results should be returned for each time a user visits the same link. It is shown in the figure of 4.2.

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 web server is made, a mapping relationship does not exist because only the web requests are observed.
No Matched Request: In some cases, the web server may periodically submit queries to the database server in order to conduct some scheduled tasks, such as cron jobs for archiving or backup. This is not driven by any web request, similar to the reverse case of the Empty Query Set mapping pattern.

Non-deterministic Mapping: The same web request may result in different SQL query sets based on input parameters or the status of the web page at the time the web request is received.

B. Modeling for Static Websites
In the case of a static website, the non-deterministic mapping does not exist as there are no available input variables or states for static content. It is 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 and it is begin by iterating all of the sessions from 1 to N.

C. Testing for Static Websites
Once the normality model is generated, it can be employed for training and detection of abnormal behavior. During the testing phase, each session is compared to the normality model. Begin with each distinct web request in the session and, since each request will have only one mapping rule in the model and simply compare the request with that rule. 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.

D. Modeling of Dynamic Patterns
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 server’s application logic, different inputs would cause different database queries. For example, to post a comment to a blog article, the web server would first query the database to see the existing comments. If the user’s comment differs from previous comments, then the web server would automatically generate a set of new queries to insert the new post into the back-end database.

Otherwise, the web server would reject the input in order to prevent duplicated comments from being posted (i.e., no corresponding SQL query would be issued.) In such cases, even assigning the same parameter values would cause different set of queries, depending on the previous state of the website. Likewise, this non-deterministic mapping case (i.e., one-to-many mapping) happens even after we normalize all parameter values to extract the structures of the web requests and queries. Since the mapping can appear differently in different cases, it becomes difficult to identify the entire one to- many mapping patterns for each web request. Moreover, when different operations occasionally overlap at their possible query set, it becomes even harder for us to extract the one to- many mapping for each operation by comparing matched requests and queries across the sessions.

E. Detection for Dynamic Websites
Once construct the separate single operation models, they can be used to detect abnormal sessions.

In the testing phase, traffic captured in each session is compared with the model. Iterate the each distinct web request in the session and for each request, determine all of the operation models that this request belongs to, since one request may now appear in several models.

For example, consider the model of two single operations such as Reading an article and Writing an Article. If there are web requests in the session that belong to none of the operation models, then it either means that our models haven’t covered this type of operation or that this is an abnormal web request. As per the algorithm identify such sessions as suspicious so that it has false positives in the detections.

Furthermore, we performed the same test for the dynamic blog website. As expected, the models for the dynamic website could also identify all of the same attack sessions as the static case.

1) Privilege Escalation Attack: For Privilege Escalation Attacks, according to previous discussion, the attacker visits the website as a normal user aiming to compromise the web server process or exploit vulnerabilities to bypass authentication. At that point, the attacker issues a set of
 privileged (e.g., admin level) DB queries to retrieve sensitive information. User log and process both legitimate web requests and database queries in the session traffic, but there are no mappings among them. IDSs working at either end can hardly detect this attack since the traffic they capture appears to be legitimate. However, Double Guard separates the traffic by sessions. If it is a user session, then the requests and queries should all belong to normal users and match structurally. Using the mapping model that we created during the training phase, Double Guard can capture the unmatched cases.

2) Hijack Future Session Attack (Web Server aimed attack): Out of the four classes of attacks we discuss, session hijacking is the most common, as there are many examples that exploit the vulnerabilities of Apache, IIS, PHP, ASP. Most of these attacks manipulate the HTTP requests to take over the web server. Double Guard is not designed to detect attacks that exploit vulnerabilities of the input validation of HTTP requests.

3) Injection Attack: It describes how this approach can detect the SQL injection attacks. To illustrate with an example, write a simple PHP login page that was vulnerable to SQL injection attack and use a legitimate username and password to successfully log in. SQL injection attacks can be mitigated by input validation. However, SQL injection can still be successful because attackers usually exploit the vulnerability of incorrect input validation implementation, often caused by inexperienced or careless programmers or imprecise input model definitions. Establish the mappings between HTTP requests and database queries, clearly defining which requests should trigger which queries. For an SQL injection attack to be successful, it must change the structure (or the semantics) of the query.

4) Direct DB attack: If any attacker launches this type of attack, it will easily be identified by this approach. First of all, according to the mapping model, DB queries will not have any matching web requests during this type of attack. On the other hand, as this traffic will not go through any containers, it will be captured as it appears to differ from the legitimate traffic that goes through the containers. In different experiments, the generated queries and sent them to the databases without using the web server containers.

5. CONCLUSION

Web applications and internet services are useful in communication and the management of personal information from anywhere and 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 is outsourced to a database or file server. Our strategy is mainly focus on to detect intrusion at both front end and back end of web application. It is used to detect intrusion in static and dynamic web application. IDS having maximum accuracy and is mainly responsible to identify intrusion. With the advent of anomaly-based intrusion detection systems, many approaches and techniques have been developed to track novel attacks on the systems. Though anomaly-based approaches are efficient, signature-based detection is preferred for mainstream implementation of intrusion detection systems. As a variety of anomaly detection techniques were suggested, it is difficult to compare the strengths, weaknesses of these methods.

REFERENCES


