DoubleGuard: Detecting Intrusions In Multi-tier Web Applications

Abstract

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

We present DoubleGuard, 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. Using DoubleGuard, we were able to expose a wide range of attacks with 100% accuracy while maintaining 0% false positives for static web services and 0.6% false positives for dynamic web services.

Existing System

Disadvantages:

We initially set up our threat model to include our assumptions and the types of attacks we are aiming to protect against. We assume that both the web and the database servers are vulnerable. Attacks are network-borne and come from the web clients; they can launch application-layer attacks to compromise the web servers they are connecting to. The attackers can bypass the web server to directly attack the database server. We assume that the attacks can neither be detected nor prevented by the current web server IDS, that attacker may take over the web server after the attack, and that 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 DoubleGuard, 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 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.

**IMPLEMENTATION**

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and its constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

**Modules:**

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.

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 Exfiltration Attack, a Denial-of-Service/Packet Drop attack, or a Replay attack. According to the mapping model, the web request should invoke some database queries (e.g., a Deterministic Mapping), then the abnormal situation can be detected. 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.

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.
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.

System Configuration:-

H/W System Configuration:-

Processor - Pentium –III
Speed - 1.1 Ghz
RAM - 256 MB(min)
Hard Disk - 20 GB
Floppy Drive - 1.44 MB
Key Board - Standard Windows Keyboard
Mouse - Two or Three Button Mouse
Monitor - SVGA
S/W System Configuration:-

- Front End : java, jdk1.6
- Database : My sqlserver 2005
- Database Connectivity : JDBC.