Int. J. Advanced Networking and Applications
Volume: 5 Issue: 6 Pages: 2123-2128 (2014) ISSN : 0975-0290

An Enhanced Intrusion Detection System for Multitier Dynamic Web Applications

S.Sasireka[1] . **N.Premalatha**[2]
[1] Assistant Professor, *Dr.MCET, Pollachi, Coimbatore – 641 035, INDIA,* sasirekasivasamy.65@gmail.com

[2] Assistant Professor, *Dr.MCET, Pollachi, Coimbatore – 641 035, INDIA,* prema.krnp@gmail.com

---ABSTRACT---
We present an efficient approach, a system used to detect attacks in multitiered web services and classify through Hierarchal clustering Algorithm. 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 with respect to Data volumes and Classify them. Implements a lightweight virtualization technique to assign each user's web session to a dedicated container, an isolated virtual computing environment. We use the cluster algorithm to accurately associate the web request with the subsequent DB queries. DoubleGuard can build a causal mapping profile by taking both the webserver and DB traffic into account.
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 DoubleGuard, 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 an 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.
--

Introduction

Web 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. 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 . A plethora of Intrusion Detection Systems (IDSs) currently examine network packets individually within both the webserver and the database system. However, there is very little work being performed on multitiered Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions. In such multitiered architectures, the back-end database server is often protected behind a firewall while the webservers 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.

Intrusion Detection Systems

To protect multitiered web services, Intrusion detection systems 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. However, we found that these IDSs cannot detect cases wherein normal traffic is used to attack the webserver and the database server. For example, if an attacker with nonadmin privileges can log in to a webserver using normal-user access credentials, he/she can find a way to issue a privileged database query by exploiting vulnerabilities in the webserver. 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 webserver is not associated with user-privileged access. 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.

Networks and their Security

Network security consists of the provisions and policies adopted by a network administrator to prevent and monitor unauthorized access, misuse, modification, or denial of a computer network and network-accessible resources. Network security involves the authorization of access to data in a network, which is controlled by the network administrator. Users choose or are assigned an ID and password or other authenticating information that allows them access to information and programs within their authority. Network security covers a variety of computer networks, both public and private, that are used in everyday jobs conducting transactions and communications among businesses, government agencies and individuals. Networks can be private, such as within a company, and others which might be open to public access. Network security is involved in organizations, enterprises, and other types of institutions. It does as its title explains: It secures the network, as well as protecting and overseeing operations being done. The most common and simple way of protecting a network resource is by assigning it a unique name and a corresponding password

Network security starts with authenticating the user, commonly with a username and a password. Since this requires just one detail authenticating the user name — i.e. the password, which is something the user 'knows'— this is sometimes termed one-factor authentication. With two-factor authentication, something the user 'has' is also used (e.g. a security token or 'dongle', an ATM card, or a mobile phone); and with three-factor authentication, something the user 'is' is also used (e.g. a fingerprint or retinal scan).
Once authenticated, a firewall enforces access policies such as what services are allowed to be accessed by the network users. Though effective to prevent unauthorized access, this component may fail to check potentially harmful content such as computer worms or Trojans being transmitted over the network. Anti-virus software or an intrusion prevention system (IPS) help detect and inhibit the action of such malware. An anomaly-based intrusion detection system may also monitor the network and traffic for unexpected (i.e. suspicious) content or behavior and other anomalies to protect resources, e.g. from denial of service attacks or an employee accessing files at strange times. Individual events occurring on the network may be logged for audit purposes and for later high-level analysis.
Communication between two hosts using a network may be encrypted to maintain privacy. Honeypots, essentially decoy network-accessible resources, may be deployed in a network as surveillance and early-warning tools, as the honeypots are not normally accessed for legitimate purposes. Techniques used by the attackers that attempt to compromise these decoy resources are studied during and after an attack to keep an eye on new exploitation techniques. Such analysis may be used to further tighten security of the actual network being protected by the honeypot.

Multitier Web Applications
In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client–server architecture in which presentation, application processing, and data management functions are logically separated. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the three-tier architecture. N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. Three-tier architectures typically comprise a *presentation* tier, a *business* or *data access* tier, and a *data* tier. While the concepts of layer and tier are often used interchangeably, one fairly common point of view is that there is indeed a difference. This view holds that a *layer* is a logical structuring mechanism for the elements that make up the software solution, while a *tier* is a physical structuring mechanism for the system infrastructure.
There have been growing concerns in recent years that many organizations are facing an excessive number of layers in their multi-layered architecture. These concerns stem from sprawling application architectures that are not well designed or managed, in which development teams create an ever-growing number of "wrapper" layers that compromise maintainability. The resulting architecture resembles a Rube Goldberg Machine that scares organizations from solving the root cause of the sprawling layers, resulting in the creation of more layers.
Three-tier is a client–server architecture in which the user interface, functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms. It was developed by John J. Donovan in Open Environment Corporation (OEC), a tools company he founded in Cambridge, Massachusetts. The three-tier model is a software architecture and a software design pattern. Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the *presentation tier* would only affect the user interface code.

Literature Review

[1] V. Jyothsna ,V. V. Rama Prasad ,K. Munivara Prasad proposed an Anomaly-Based Intrusion Detection System, is a system for detecting computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous. The classification is based on heuristics or rules, rather than patterns or signatures, and will detect any type of misuse that falls out of normal system operation. This is as opposed to signature based systems which can only

detect attacks for which a signature has previously been created This paper investigate this issue, the current state of the experiment practice in the field of anomaly based intrusion detection is reviewed and survey recent studies in this.

[2] Fredrik Valeur , Giovanni Vigna, Christopher Kruegel, Richard A. Kemmerer proposed This paper address the issue, researchers and vendors have proposed alert correlation, an analysis process that takes the alerts produced by intrusion detection systems and produces compact reports on the security status of the network under surveillance. Although a number of correlation approaches have been suggested, there is no consensus on what this process is or how it should be implemented and evaluated. In particular, existing correlation approaches operate on only a few aspects of the correlation process, such as the fusion of alerts that are generated by different intrusion detection systems in response to a single attack, or the identification of multistep attacks that represent a sequence of actions performed by the same attacker.

[3]. David Wagner, Drew Dean This paper focus on some security problems are directly attributable to faulty application logic, such as programs that fail to check authentication information before proceeding, and one limitation of our intrusion detection system is that it does not detect attacks that exploit logic errors. Application logic bugs, however, are dwarfed in practice by buffer overflow problems and other vulnerabilities that allow for execution of arbitrary machine code of the attacker's choice [8, 35], and it is the latter type of vulnerability.

[4] Ramesh Chandra, Taesoo Kim, Meelap Shah, Neha Narula, Nickolai Zeldovich proposed this paper Users or administrators must manually inspect the application for signs of an attack that exploited the vulnerability, and if an attack is found, they must track down the attacker's actions and repair the damage by hand. When an administrator learns of a security vulnerability in a web application, he or she can use WARP to check whether that vulnerability was recently exploited, and to recover from any resulting intrusions
An attacks can affect users' browsers, making it difficult to track down the extent of the intrusion purely on the server. For example attack, when Alice (or any other user) visits an infected Wiki page, the web server cannot tell if a subsequent page edit request from Alice's browser was caused by Alice or by the malicious JavaScript code. Yet an ideal system should revert all effects of the malicious code while preserving any edits that Alice made from the same page in her browser.

[5] Chris Anley research analysis discusses in detail the common "SQL injection" technique, as it applies to the popular Microsoft Internet Information Server/Active Server Pages/SQL Server platform. It discusses the various ways in which SQL can be "injected" into the application and addresses some of the data validation and database lockdown issues that are related to this class of attack. The paper is intended to be read by both developers of Web applications which communicate with databases and by security professionals whose role includes auditing these Web applications.

[6] K. Bai, H. Wang, and P. Liu, **"**Access control and integrity constraints are well known approaches to ensure data integrity in commercial database systems. However, due to operational mistakes, malicious intent of insiders or vulnerabilities exploited by outsiders, data stored in a database can still be compromised. When the database is under an attack, rolling back and re-executing the damaged transactions are the most used mechanisms during system recovery. This kind of mechanism either stops (or greatly restricts) the database service during repair, which causes unacceptable availability loss or denial-of- service for mission critical applications, or may cause serious damage spreading during on-the-fly recovery where many clean data items are accidentally corrupted by legitimate new transactions. To resolve this dilemma, we devise a novel mechanism, called database firewall in this paper. This firewall is designed to protect good data from being corrupted due to damage spreading. Pattern mining and Bayesian network techniques are adopted in the framework to mine frequent damage spreading patterns and to predict the data integrity in the face of attack. Our approach provides a probability based strategy to estimate the data integrity on the fly. With this feature, the database firewall is able to enforce a policy of transaction filtering to dynamically filter out the potential spreading transactions.

[7] B.I.A. Barry and H.A. Chan, proposed a Signature-based intrusion detection systems (IDSs) have the advantages of producing a lower false alarm rate and using less system resources compared to anomaly based systems. However, they are susceptible to obfuscation used by attackers to introduce new variants of the attacks stored in the database. Some of the disadvantages of signature-based IDSs can be attributed to the fact that they are mostly purely syntactic and ignore the semantics of the monitored systems. In this paper, we present the design and implementation of a signature database that assists a Specification-based IDS in a converged environment. Our design is novel in terms of considering the semantics of the monitored protocols alongside their syntax. Our protocol semantics awareness is based on the state transition analysis technique which models intrusions at a high level using state transition diagrams. The signature database is hierarchically designed to insure a balance between ease of use and fast retrieval in real time. The database prototype is tested against some implemented attacks and shows promising efficiency.

[8]. D. Bates, A. Barth, and C. Jackson, describes a

Cross-site scripting flaws have now surpassed buffer overflows as the world's most common publicly-reported security vulnerability. In recent years, browser vendors and researchers have tried to develop client-side filters to mitigate these attacks. We analyze the best existing filters and find them to be either unacceptably slow or easily circumvented. Worse, some of these filters could introduce vulnerabilities into sites that were previously bug-free. We propose a new filter design that achieves both high performance and high precision by blocking scripts after HTML parsing but before execution. Compared to previous approaches, our approach is faster, protects against more vulnerabilities, and is harder for attackers to abuse. We have contributed an implementation of our filter design to the WebKit open source rendering engine, and the filter is now enabled by default in the Google Chrome browser.

[9]. M. Christodorescu and S. Jha, Malicious code detection is a crucial component of any defense mechanism. In this paper, we present a unique viewpoint on malicious code detection. We regard malicious code detection as an obfuscation-deobfuscation game between malicious code writers and researchers working on malicious code detection. Malicious code writers attempt to obfuscate the malicious code to subvert the malicious code detectors, such as anti-virus software. We tested the resilience of three commercial virus scanners against code-obfuscation attacks. The results were surprising: the three commercial virus scanners could be subverted by very simple obfuscation transformations! We present an architecture for detecting malicious patterns in executables that is resilient to common obfuscation transformations. Experimental results demonstrate the efficacy of our prototype tool, SAFE (a static analyzer for executables).

[10]. M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna**,** In recent years, web applications have become tremendously popular, and nowadays they are routinely used in security-critical environments, such as medical, financial, and military systems. As the use of web applications for critical services has increased, the number and sophistication of attacks against these applications have grown as well. Most approaches to the detection of web-based attacks analyze the interaction of a web application with its clients and back-end servers. Even though these approaches can effectively detect and block a number of attacks, there are attacks that cannot be detected only by looking at the external behavior of a web application.
In this paper, we present Swaddler, a novel approach to the anomaly-based detection of attacks against web applications. Swaddler analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state. By doing this, Swaddler is able to identify attacks that attempt to bring an application in an inconsistent, anomalous state, such as violations of the intended workflow of a web application. We developed a prototype of our approach for the PHP language and we evaluated it with respect to several real-world applications.

Existing Research

Web 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. 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 . A plethora of Intrusion Detection Systems (IDSs) currently examine network packets individually within both the webserver and the database system. However, there is very little work being performed on multitiered Anomaly Detection (AD) systems that generate models of network behavior for both web and database network interactions. In such multitiered architectures, the back-end database server is often protected behind a firewall while the webservers 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.

Problem Definition

An IDS system only considers models the network behavior of user across both the front-end webserver and the back-end database without any clear partitioning. By monitoring both web and subsequent database requests, we are able to ferret out attacks that an independent IDS would not be able to identify .The limitations of any multitier IDS in terms of training sessions and functionality coverage Cannot be done for all the users in the network.

Proposed System

In this project we present a system used to detect attacks in multitiered web services and classify through clustering Algorithm. 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 with respect to Data volumes and Classify them. The project implements a lightweight virtualization technique to assign each user's web session to a dedicated container, an isolated virtual computing environment. We use the cluster algorithm to accurately associate the web request with the subsequent DB queries.

The system build a causal mapping profile by taking both the webserver and DB traffic into account. The system uses a multi-tier approach which makes web applications retain their simplicity for the user and complexity for the attacker.

Methodology

Building the Normality Model

We deployed a static testing website using the Content Management System. We chose to assign each user session into a different container. We can assign a new container per each new IP address of the client. The container will log all the SQL Queries executed by client in database. Deterministic Mapping and the Empty Query Set Mapping patterns are discovered from training sessions.

Finding Deterministic Mapping Queries

Deterministic Mapping is the most common and perfectly matched pattern. Web request rm appears in all traffic with the SQL queries set Qn. If Qn is present in the session traffic without the corresponding rm is classified as intrusion.

Finding 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. Ex when a web request for retrieving an image GIF file from the same webserver is made, a mapping relationship does not exist because only the web requests are observed. This type of mapping is called rm->O ;. During the testing phase, we keep these web requests together in the set EQS

Algorithms Used

Algorithm for Intrusion Analysis

Input : Log Dataset and Training data's(SQL)
Output : Intrusion data.

```
1. for each session separated traffic Ti do
2. Get different HTTP requests r and DB queries q in
   this session for each different r do
3: if r is a request to static file then
4: Add r into set EQS
5: else
6: if r is not in set REQ then
7: Add r into REQ
8: Append session ID i to the set ARr with r as the key
9: for each different q do
10: if q is not in set SQL then
11: Add q into SQL
12: Append session ID i to the set AQq with q as the
    key
13: for each HTTP request r in REQ do
14: if r has no deterministic mapping model then
15: Add r into set EQS
16: return True
```

Hijack Future Session Attack and Injection Attack

An attacker usually takes over the webserver and therefore hijacks all subsequent legitimate user sessions to launch attacks. We find hijacking other user sessions, the attacker can eavesdrop, send spoofed replies, and/or drop user requests.

Finding Injection Attack

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.
This SQL injection attack changes the structure of the SQL queries, even if the injected data were to go through the webserver 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.
These module contains the unique idea that compares SQL query strings and blocks suspicious sql-query and passes original sql-query. The classification of Suspecious query is done by analyzing the datasets of Original query and suspicious query. classifies learns the dataset and according to learning procedure ,it classifies the queries.

Example

Original query=select * from admin where uid="1";
Suspecious query=select * from admin where uid=" „ OR 1=1;--„
Here, original query is passed and suspicious query is blocked.
Word-list contains the tokens of sql-query strings.
„O"-Original query
„S"-Suspecious query
Ex: („O") select * from admin where uid = „1„;
(„S") select * from admin where uid = „ „ OR 1=1;--„
(„O") select * from admin where uid ="1" && pwd ="abc";
(„S") select * from admin where uid = „ „ OR 1=1;--„

Algorithm

Step 1. Select a reasonable amount as the training set.
Step 2. Input the SQL-Query string.
Step 3. Feed the training set into the process to generate a model.
Step 4. Now classify the model using training dataset.
Step 5. Labeled output with O and S
Step 6. Mark S as suspicious query

Experimental Results

The user sessions are tracked by the ipaddress, date and time of visit in container. The full details about the users can be viewed by the administrator. The tracking of the details except ipaddress is general. The ipaddress is tracked by using the following one line code: $ip=$_SERVER['REMOTE_ADDR']. This code is placed in the code file to track the users who are all visiting the web site.

The System is experimented with open source web programming language and Database for logging PHP & MYSQL with WAMP server in windows platform which is a container stored in the web server and is available in the admin end. The container file consists of the information about the query, ipaddress, date and time of visit. It consists of all records i.e., the information about all the clients who are all visited the web site with their database query. From 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.

To evaluate the detection results for the system, analyze classes of attacks like deterministic mapping, sql injection , empty query list and etc. When deployed a prototype on a system that employed Apache webserver, a cms application and a MySQL back end. The Proposed system was able to identify a wide range of attacks with minimal efforts

Conclusion

Intrusion detection system that builds models of normal behavior for multi- tiered 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, after empowering it forms a container-based IDS with multiple input streams to produce alerts. 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.

References

[1]. C. Anley, “Advanced Sql Injection in Sql Server Applications,” technical report, Next Generation Security Software, Ltd., 2002.

[2]. K. Bai, H. Wang, and P. Liu, “Towards Database Firewalls,” Proc. Ann. IFIP WG 11.3 Working Conf. Data and Applications Security (DBSec ’05), 2005.

[3]. B.I.A. Barry 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.

[4]. D. Bates, A. Barth, and C. Jackson, “Regular Expressions Considered Harmful in Client-SideXSS Filters,” Proc. 19th Int’l Conf. World Wide Web, 2010.

[5]. M. Christodorescu and S. Jha, “Static Analysis of Executables to Detect Malicious Patterns,” Proc. Conf. USENIX Security Symp., 2003.

[6]. M. Cova, D. Balzarotti, V. Felmetsger, and G.Vigna, “Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications, ” Proc. Int’l Symp. Recent Advances in Intrusion Detection (RAID ’07), 2007.

[7]. H. Debar, M. Dacier, and A. Wespi, “Towards a Taxonomy of Intrusion-Detection Systems,” Computer Networks, vol. 31, no. 9, pp. 805-822, 1999.

[8]. V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, “Toward Automated Detection of Logic Vulnerabilities in Web Applications,” Proc. USENIX Security Symp., 2010.

[9]. Y. Hu and B. Panda, “A Data Mining Approach for Database Intrusion Detection,” Proc. ACM Symp. Applied Computing (SAC), H. Haddad, A. micini, R.L. Wainwright, and L.M. Liebrock, eds., 2004.

[10]. Y. Huang, A. Stavrou, A.K. Ghosh, and S. Jajodia, “Efficiently Tracking Application Interactions Using Lightweight Virtualization,” Proc. First ACM Workshop Virtual Machine Security, 2008.