Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering — CSEE 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5461-7

Container Architecture for Detection and Prevention of Intrusions using virtualization technique

Abstract- ***Network Intrusion Detection Systems (IDSs) which are based on sophisticated algorithms rather than current signature-base detections are in demand. Web services have moved to a multi-tiered design wherein the webserver runs the application front-end logic and data are outsourced to a database or file server in order to enable communication and the management of personal information from anywhere. The proposed system is Container based Intrusion Detection System, an IDS system that models the network behavior of user sessions across both the front-end webserver and the back-end database. This system used to detect attacks in multi-tiered web services. Our approach can create normality models of isolated user sessions that include both the web front-end (HTTP) and back-end (File or SQL) network transactions. For websites that do not permit content modification from users, there is a direct causal relationship between the requests received by the front-end webserver and those generated for the database back end. Virtualization is used to isolate objects and enhance security performance. Lightweight containers can have considerable performance advantages over full virtualization containers.***

Keywords—doubleguard, multi-tiered web services, virtualization.

S.Saravanan Department of Computer Science Engineering ,R.M.K.C.E.T

M.Ramakrishnan Department of Information Technology

Baskar.M Department of Information Technology R.M.K.C.E.T R.S.M Nagar,Puduvoyal-601 206(T.N). INDIA

Gnanasekaran.T Department of Information Technology ,R.M.K.C.E.T, R.S.M Nagar,Puduvoyal-601 206(T.N). INDIA

I. Introduction

Web applications are the most common way to make services and data available on the Internet. Unfortunately, with the increase in the number and complexity of these applications, there has also been an increase in the number and complexity of vulnerabilities. Current techniques to identify security problems in web applications have mostly focused on input validation flaws, such as cross site scripting and SQL injection, with much less attention devoted to application logic vulnerabilities. Application logic vulnerabilities are an important class of defects that are the result of faulty application logic [2]. These vulnerabilities are specific to the functionality of particular web applications, and, thus, they are extremely difficult to characterize and identify. Learning-based anomaly detection has proven to be an effective black-box technique for detecting unknown attacks. However, the effectiveness of this technique crucially depends upon both the quality and the completeness of the training data. Unfortunately, in most cases, the traffic to the system protected by an anomaly detector is not uniformly distributed. Therefore, some components (e.g., authentication, payments, or content publishing) might not be exercised enough to train anomaly detection system in a reasonable time frame. This is of particular importance in real-world settings, where anomaly detection systems are deployed with little or no manual configuration, and they are expected to automatically learn the normal behavior of a system to detect or block attacks. In this work, we first demonstrate that the features utilized to train a learning-based detector can be semantically grouped, and that features of the same group tend to induce similar models [6].

II. Motivation

Some previous approaches have detected intrusions or vulnerabilities by statically analyzing the source code or executables [1]. Other approaches dynamically track the information flow to understand propagations and detect intrusions [3]. In the DoubleGuard, the new container-based webserver architecture enables us to separate the different information flows by each session. This provides a means of tracking the information flow from the

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering — CSEE 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5461-7

webserver 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 webpage, our proposed approach does not require application logic for building a model.

Validating input is useful to detect or prevent SQL or Cross Site Scripting (XSS) injection attacks. This is orthogonal to the DoubleGuard approach, which can utilize input validation as an additional defense [7]. However, we have found that DoubleGuard can detect SQL injection attacks by taking the structures of web requests and database queries.

III. Our Approach

In the proposed model, Static Model building algorithm is used. It employs a virtualization technique to web request with the subsequent DB queries. Thus, Our proposed architecture in Fig.1 can build a causal mapping profile by taking both the webserver and DB traffic into account. In addition to this static website case, there are web services that permit persistent back-end data modifications. These services, which we call dynamic, allow HTTP requests to include parameters that are variable and depend on user input. Lightweight virtualization technique is to assign each user's web session to a dedicated container, an isolated virtual computing environment.

A. *User interface:*

This main module is responsible for accepting user queries and to generate http requests. This module is also responsible for displaying the query results to the user after the query has been executed by the web server.

B. *Query classifier:*

This module is deployed between web based applications and the back-end database server. The SQL queries sent by the applications are captured and sent to the IDS for analysis. The query classifier module parses each incoming SQL queries that are variable and depend on user input.

C. *Fuzzy anomaly detection module:*

The fuzzy decision manager first extracts from the requested URL, the path to the web application being invoked, along with the arguments passed to it.

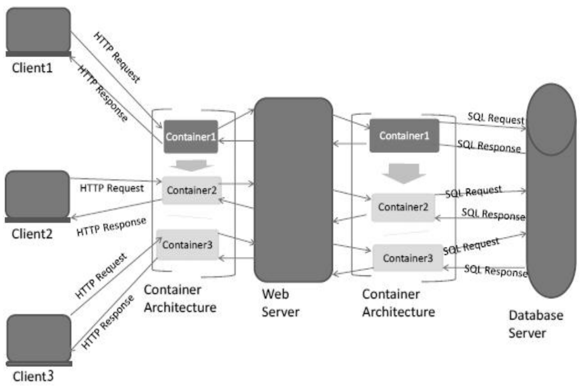

Fig.1 Container based IDS architecture

The fuzzy decision manager then looks up the profile associated with the web application. A profile is a collection of statistics associated with one specific web application. The fuzzy decision manager present in the profile is a set of keys and scores used to evaluate the features of a query and operate in one of two modes, learning or detection. In the learning phase we use C4.5 classification technique which models and builds a profile of the "normal" Characteristics of a given feature of a query (e.g., the normal lengths of values for attributes),[4] setting a dynamic detection threshold for the attributes. During the detection phase, models return an anomaly score for each observed example of attribute values using the fuzzy rules generated from the training phase. This is just a probability value in the range of 0-1 indicating the degree of anomaly of the observed value in relation to the existing profile for that query which is computed using a fuzzy logic decision manager.

D. *Access control manager:*

The architecture of this anomaly detection system necessitates the existence of an access control manager between the query classification component and web servers. This manager is utilized when a malicious web request that was let through by the query classifier to access the web server which can be checked for privilege level using anomaly score. In this system, access control can choose to update privilege levels of the web request to control malicious requests. This process involves characterizing the incoming anomaly using fuzzy rules and then generating updating messages and finally updating the access privilege levels to reflect the level of anomaly. In this three access levels namely privilege user lever, application programmer level and naïve user level are used. Queries with

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering — CSEE 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5461-7

privilege user and application programmer level are sent to the smart server whereas the queries with naïve user levels are sent to dump server.

IV. Testing for Websites

Once the model is built, it can be used to detect malicious sessions. For our static website testing, we used the production website, which has regular visits of around 50-100 sessions per day. We collected regular traffic for this production site we used the attack tools listed in to manually launch attacks against the testing website, and we mixed these attack sessions with the normal traffic obtained during the training phase.

The testing phase algorithm is as follows:

1. If the rule for the request is Deterministic Mapping r ! Q (Q ¼6 ;), we test whether Q is a subset of a query set of the session. If so, this request is valid, and we mark the queries in Q. Otherwise, a violation is detected and considered to be abnormal, and the session will be marked as suspicious.
2. If the rule is Empty Query Set r ! ;, then the request is not considered to be abnormal, and we do not mark any database queries. No intrusion will be reported.
3. For the remaining unmarked database queries, we check to see if they are in the set NMR. If so, we mark the query as such.
4. Any untested web request or unmarked database query is considered to be abnormal. If either exists within a session, then that session will be marked as suspicious.

In our implementation and experimenting of the static testing website, the mapping model contained the Deterministic Mappings and Empty Query Set patterns without the No Matched Request pattern. This is commonly the case for static websites.

V. Performance Evaluation

A prototype of Container Architecture is implemented using a webserver with a back-end DB. We also set up two testing websites, one static and the other dynamic. To evaluate the detection results for our system, we analyzed four classes of attacks, as discussed [10], and measured the false positive rate for each of the two websites.

A. *Implementation*

We chose to assign each user session into a different container; however, this was a design decision. For instance, we can assign a new container per each new IP address of the client. In our implementation, containers were recycled based on events or when sessions time out. We were able to use the same session tracking mechanisms as implemented by the Apache server (cookies, mod_usertrack, etc.) because lightweight virtualization containers do not impose high memory and storage overhead. Thus, we could maintain a large number of parallel-running Apache instances similar to the Apache threads that the server would maintain in the scenario without containers [9].

We evaluated the overhead of our container-based server against a vanilla webserver. In order to measure throughput and response time, we used two webserver benchmark tools: http_load and autobench. The testing website was the dynamic blog website, and both vanilla webserver and the container-based webserver connected to the same Mysql database server on the host machine. For the container-based server, we maintained a pool of 160 webserver instances on the machine.

B. *Attack Detection*

For the testing phase, we used the attack tools to manually launch attacks against the testing website, and we mixed these attack sessions with the normal traffic obtained during the training phase. We used the sqlmap which is an automatic tool that can generate SQL injection attacks. Nikto, a webserver scanner tool that performs comprehensive tests, and Metasploit were used to generate a number of webserver-aimed http attacks (i.e., a hijack future session attack). We performed the same attacks on both DoubleGuard and a classic three-tier architecture with a network IDS at the webserver side and a database IDS at the database side. As there is no popular anomaly-based open source network IDS available, we used Snort as the network IDS in front of the webserver, and we used GreenSQL as the database IDS.

Furthermore, we performed the same test for the dynamic blog website. In addition to the real traffic data that we captured for plotting the ROC curves, we also generated 1,000 artificial traffic sessions using Selenium and mixed the attack sessions together with all of them. As expected, the models for the dynamic website could also identify all of the same attack sessions as the static case.

Our proposed architecture is not designed to detect attacks that exploit vulnerabilities of the input validation of HTTP requests. The various possible attacks that can be detected and prevented in the multi-tiered environment are listed as below.

Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering — CSEE 2013
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-981-07-5461-7

1) *Privilege Escalation Attack*

In Privilege Escalation Attacks, the attacker visits the website as a normal user aiming to compromise the webserver 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. We 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, DoubleGuard 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, DoubleGuard can capture the unmatched cases.

2) *Hijack Future Session Attack (Webserver-Aimed Attack)*

Out of the four classes of attacks we discuss [11], session hijacking is the most common, as there are many examples that exploit the vulnerabilities of Apache, IIS, PHP, ASP, and cgi, to name a few. Most of these attacks manipulate the HTTP requests to take over the webserver. We first ran Nikto. As shown in our results, both Snort and Double-Guard detected the malicious attempts from Nikto. As a second tool, we used Metasploit loaded with various HTTP-based exploits. This time, Snort missed most of these attack attempts, which indicates that Snort rules do not have such signatures. However, DoubleGuard was able to detect these attack sessions. Here, we point out that most of these attacks are unsuccessful, and DoubleGuard captured these attacks mainly because of the abnormal HTTP requests.

Our proposed architecture can generate two classes of alerts. One class of alerts is generated by sessions whose traffic does not match the mapping model with abnormal database queries. The second class of alerts is triggered by sessions whose traffic violates the mapping model but only in regard to abnormal HTTP requests; there is no resulting database query. Most unsuccessful attacks, including 404 errors with no resulting database query, will trigger the second type of alerts

3) *Injection Attack*

We describe how our approach can detect the SQL injection attacks. To illustrate with an example, we wrote a simple PHP login page that was vulnerable to SQL injection attack. As we used a legitimate username and password to successfully log in, we could include the HTTP request.

The HTTP request is obtained from the SQL injection attacker. The parameter shown in the box is the injected content. After normalizing all of the values in this HTTP request, we had the same HTTP request. However, the database queries we received do not match the deterministic mapping we obtained during our training phase.

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.

4) *Direct DB Attack*

If any attacker launches this type of attack, it will easily be identified by our approach. First of all, according to our 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 our experiments, we generated queries and sent them to the databases without using the webserver containers. DoubleGuard readily captured these queries. Snort and GreenSQL did not report alerts for this attack.

Our proposed architecture offers the capability of normalizing the parameters so that the user can choose which values to normalize. For example, one can choose not to normalize the value “admin” in “user ¼ ‘admin’.” Likewise, one can choose to normalize it if the administrative queries are structurally different from the normal-user queries, which is common case. Addition-ally, if the database can authenticate admin and non admin users, then privilege escalation attacks by changing values are not feasible. 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. When we deployed our prototype on a system that employed Apache webserver, a blog application, and a MySQL back end, Container architecture was able to identify a wide range of attacks with minimal false positives.

VI. Conclusion

A container based 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 was proposed. Unlike previous approaches that correlated or summarized alerts generated by independent IDSs, DoubleGuard forms container-based IDS with multiple input streams to produce alerts. We achieved this by isolating the flow of information from each webserver session with a lightweight virtualization. Furthermore, we quantified the detection accuracy of our approach when we attempted to model static and dynamic web requests with the back-end file system and database queries. For static websites, we built a well-correlated model, for detecting different types of attacks.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which have greatly improved the quality of the paper.

References

[1] S. Kumar, "Classification and detection of computer intrusions", Ph.D. thesis, Purdue Univ., West Lafayette, IN, 1995.
[2] W. Lee and D. Xiang "Information-theoretic measures for anomaly detection", In Proc. of the 2001 IEEE Symp. on Security and Privacy, Oakland, CA, May, 2001, pp. 130-143.
[3] C. Anley, “Advanced Sql Injection in Sql Server Applica-tions,” technical report, Next Generation Security Software, Ltd., 2002.
[4] 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.
[5] 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.
[6] D. Bates, A. Barth, and C. Jackson, “Regular Expressions Considered Harmful in Client-Side XSS Filters,” Proc. 19th Int’l Conf. World Wide Web, 2010.
[7] M. Christodorescu and S. Jha, “Static Analysis of Executables to Detect Malicious Patterns,” Proc. Conf. USENIX Security Symp.,2003.
[8] 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.
[9] H. Debar, M. Dacier, and A. Wespi, “Towards Taxonomy of Intrusion-Detection Systems,” Computer Networks, vol. 31, no. 9, pp. 805-822, 1999.
[10] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, “Toward Automated Detection of Logic Vulnerabilities in Web Applica-tions,” Proc. USENIX Security Symp., 2010.
[11] Y. Hu and B. Panda, “A Data Mining Approach for Database Intrusion Detection,” Proc. ACM Symp. Applied Computing (SAC), H. Haddad, A. Omicini, R.L. Wainwright, and L.M. Liebrock, eds., 2004.
[12] 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.

About Authors:

S.Saravanan received M.E Computer Science and Engineering from Anna University and pursuing Ph.D in Anna University, Chennai. He is working as Assistant Professor in R.M.K College of Engineering and Technology. His current research interests include Mobile Adhoc network, Computer networks and network security.

M.Ramakrishnan received Ph.D from Anna University, Chennai. He is working as professor in Velammal Engineering College. He is a member of ISTE, CSI, IACSIT, IAEME and IEEE. His current research interests include computer netwoks and network security, wireless sensors networks and image processing.

Baskar.M received M.Tech Information Technology from Sathyabama University and pursuing Ph.D in Anna University, Chennai. He is working as Assistant Professor in R.M.K College of Engineering and Technology, Chennai. He is a member of ISTE. His current research interests include parallel and distributed systems, computer networks and network security.

Gnanasekaran.T received Ph.D from Anna University, Chennai. He is working as professor in R.M.K. Institutions. He is a member of IEI, ISTE, IAENG and ACEEE. His current research interests include Wireless sensor networks, WiMAX, Parallel and distributed Systems, Computer networks, Mobile Adhoc networks and Broadband wireless technology.