Detecting and Preventing DDoS attacks, Taking Counter Measures for Successful Attack

Ch.Annapurna[1], Mr.G.Praveen Babu[2]
[1]M.Tech student of CS, [2]Assosiate Professor of CSE
[12]JNTUH,SIT

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 webserver runs the application front-end logic and data are outsourced to a database or file server.
As the increasing of usage of multi tier web services ,attacks on multi tier web services are also increases. The main attack we are considering here is Distributed Denial-of-Service attack. So to prevent and detect the possibility of DDoS attack here we are using a Double Guard Technology.
Double Guard differs from this type of approach that correlates alerts from independent IDSs. Rather, Double-Guard 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. 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.

Keywords- Double Guard, Intrusion Detection System

I. INTRODUCTION

Web-delivered services and applications have increased in both popularity and complexity over the past few years. 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.
Because of increasing the use of personal and/or corporate data, web services have always been the target of attacks.
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.However, IDS cannot detect cases wherein normal traffic is used to attack the web server and the database server. Another drawback of using of IDS is in current multi-threaded web server architecture, it is not feasible to detect causal mapping between web server traffic and DB server traffic since traffic cannot be clearly attributed to user sessions.
In this paper, we present DoubleGuard, a 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. To achieve this, we employ a lightweight virtualization technique to assign each user's web session to a dedicated container, an isolated virtual computing environment. We use the container ID to accurately associate the web request with the subsequent DB queries. Thus, DoubleGuard can build a causal mapping profile by taking both the web sever and DB traffic into account.
As we are using multi tierwebservices, it gets requests from many clients. So to not to correlate the information between two clients we are using container IDs. It means for each user we are assigning a different id. So for each user it will create a seperate session called containers.
For each user session we are assigning a virtualized server. It means for each user request we are creating a virtual server. So if any attack happens on server it will not effect on actual server. Because we are creating a virtual server.
Because of we are using containers and virtualization we are calling it as Double Guard Technology.
No prior knowledge of the source code or the application logic of web services deployed on the webserver. Virtualization is used to isolate objects and enhance security performance. Lightweight containers can have considerable performance advantages over full virtualization.

II. BACKGROUND 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 [1], [2].
Intrusion alerts correlation [3] 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. DoubleGuard differs from this type of approach that correlates alerts from independent IDSes. Rather, DoubleGuard operates on multiple feeds of network traffic using a 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 [4] also uses temporal information to detect intrusions. DoubleGuard, however, does not correlate events

on a time basis, which runs the risk of mistakenly considering independent but concurrent events as correlated events. DoubleGuard 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 [5], [6], [7] and database firewalls [8]. Thesesoftwares, such as Green SQL [9], 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 [10] 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.

In the existing system we use desktop systems [11] that use lightweight virtualization to isolate different application instances. Such virtualization techniques are commonly used for isolation and containment of attacks. However, in our DoubleGuard, 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.

III. IMPLEMENTATION

In an general case we have two types of web sites. Those are static and dynamic. 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 served data. This creates tremendous challenges for IDS system training because the HTTP requests can contain variables in the passed 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. In other cases, one request will invoke a number of database queries 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 we know that all communications from the clients to the database are separated by a session. We assign each session with a unique session ID. DoubleGuard normalizes the variable values in both HTTP requests and database queries, preserving the structures of the requests and queries. To achieve this, DoubleGuard substitutes the actual values of the variables with symbolic values.

Following this step, session i will have a set of requests, which is Ri , as well as a set of queries, which is Qi . 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 rm∈ REQ may also appear several times in different Ri where i can be 1, 2 ... N. The same holds true for qn∈ SQL.

If several SQL queries, such as qn, qp, are always found within one HTTP request of rm, then we can usually have an exact mapping of rm → {qn, qp}. However, this is not always the case. Some requests will result in different queries based on the request parameters and the state of the web server. For example, for web request rm, the invoked query set can sometimes be {qn,qp} or, at other times, {qp} or {qq,qn,qs}. The probabilities for these queries are usually not the same. For 100 requests of rm, the set is at {qn,qp} 75 times, at {qp} 20 times, and at {qq,qn,qs} only 5 times. In such a case, we can find the mapping of rm → qp is 100%, with a rm → qn possibility of 80% and a rm → qs occurrence at 5% of all cases. We define this first type of mapping as deterministic and the latter ones as non-deterministic.

We developed an algorithm that takes the input of training dataset and builds the mapping model for static websites. For each unique HTTP request and database query, the algorithm assigns a hash table entry, the key of the entry is the request or query itself, and the value of the hash entry is AR for the request or AQ for the query respectively.

Static Model building algorithm:

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

In our prototype, 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..It uses the container ID to accurately associate the web request with the subsequent DB queries.
Thus, DoubleGuard 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.

IV. ARCHITECTURE

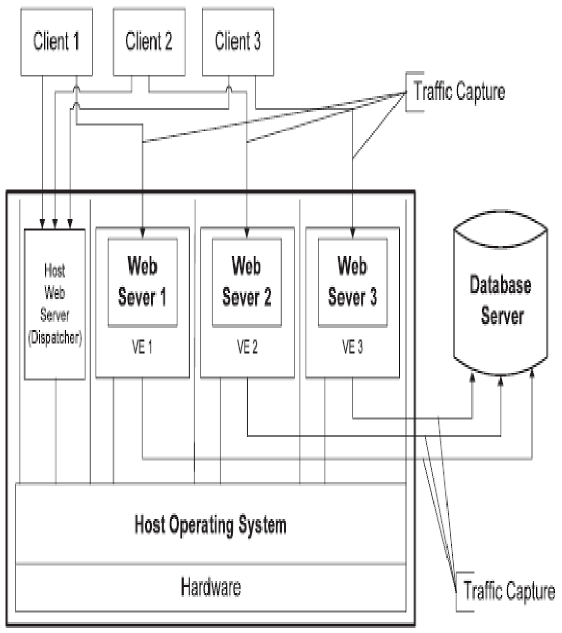


Fig.1: Using of DoubleGuard in multitier web servers

In anmulti tier architecture server gets request from many users. So for each request we are assigning container id so that one user cannot steal the information from another user. Each of these requests from clients should be assigned to server. But here we are using virtualization technique on server. It means every request from client is assigned to each virtualized server. So if any attack is happend on actual server, it will not effect on original server . So that attacks that are happend on network get reduced. Because of we are using containr ID and virtualization Techniques we are calling it as Double Guard technology.

V. ADVANTAGES OF DOUBLE GUARD TECHNOLOGY

A. Attack Detection

Once the model is built, it can be used to detect malicious sessions. We manually launch attacks against the testing website, and we mixed these attack sessions.

A.Injection Attack

Attacks such as SQL injection do not require compromisingthe webserver. Attackers can use existing vulnerabilities inthe webserver logic to inject the data or string content thatcontains the exploits and then use the webserver to relay theseexploits to attack the back-end database. Since our approachprovides two-tier detection, even if the exploits are acceptedby the webserver, the relayed contents to theDBserver wouldnot be able to take on the expected structure for the givenwebserver request. For instance, since the SQL injectionattack changes the structure of the SQL queries, even if theinjected data were to go through the webserver side, it wouldgenerate SQL queries in a different structure that could bedetected as a deviation from the SQL query structure thatwould normally follow such a web request

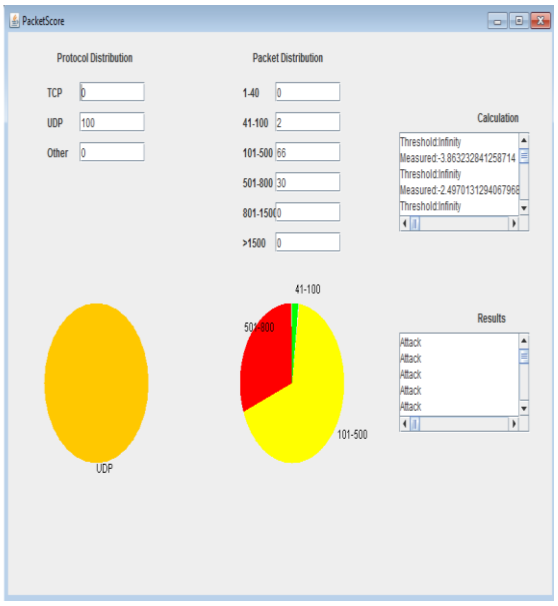
B.Direct DB Attack

It is possible for an attacker to bypass the webserver orfirewalls and connect directly to the database. An attackercould also have already taken over the webserver and besubmitting such queries from the webserver without sending web requests. Without matched web requests for suchqueries, a webserver IDS could detect neither. Furthermore,if these DB queries were within the set of allowed queries,then the database IDS it would not detect it either.However, thistype of attack can be caught with our approachsince we cannot match any web requests with these queries.

.

VI. RESULTS

In a network we can get packets from either from TCP or UDP or any other protocol, Based on the packet type we are seperating the packets. Here for packet creation we are using jpcap package in java. If a client is sending requests for more number of times or packets flow is getting high over some peroid of time to a network we are considering that as an Attack. For every network there should be a Threshold value.It means maximum capable of accepting packets from another network or client. The thresould value range is 1 to 5000. The Defalut value is 1500.
Here we are performing an attack exernally to a web page. So we are calculate the incoming packets and requests. So those exceeds the range that we are chosen we are considering that as an attack. These attacks information can seen only to admin.


PacketScore

Protocol Distribution

TCP	0
UDP	100
Other	0

Packet Distribution

1-40	0
41-100	2
101-500	66
501-800	30
801-1500	0
>1500	0

Calculation

```
Threshold:Infinity
Measured:-3.863232841258714
Threshold:Infinity
Measured:-2.4970131294067968
Threshold:Infinity
```

Results

```
Attack
Attack
Attack
Attack
Attack
```

Fig.2: Final result

VII. CONCLUSION

Our assumption is that an attacker can obtain “full control”of the webserver thread that he/she connects to. That is, theattacker can only take over the webserver instance runningin its isolated container. Our architecture ensures that everyclient be defined by the IP address and port container pair,which is unique for each session. Therefore, hijacking anexisting container is not possible because traffic for othersessions is never directed to an occupied container. If thiswere not the case, our architecture would have been similarto the conventional one where a single webserver runsmany different processes. Moreover, if the databaseauthenticates the sessions from the webserver, then eachcontainer connects to the database using either admin useraccount or nonadmin user account and the connection isauthenticated by the database.

In such case, an attacker willauthenticate using a nonadmin account and will not beallowed to issue admin level queries. In other words, theHTTP traffic defines the privileges of the session which canbe extended to the back-end database, and a nonadmin usersession cannot appear to be an admin session when it comesto back-end traffic.Within the same session that the attacker connects to, it isallowed for the attacker to launch “mimicry” attacks. It ispossible for an attacker to discover the mapping patterns bydoing code analysis or reverse engineering, and issue“expected” web requests prior to performing maliciousdatabase queries.

However, this significantly increases theefforts for the attackers to launch successful attacks. Inaddition, users with nonadmin permissions can causeminimal (and sometimes zero) damage to the rest of thesystem and therefore they have limited incentives to launchsuch attacks.By default, DoubleGuard normalizes all the parameters.Of course, the choice of the normalization parameters needsto be performed carefully. DoubleGuard offers the capabilityof normalizing the parameters so that the user ofDoubleGuard can choose which values to normalize.

Forexample, we can choose not to normalize the value “admin” in “user= ‘admin’.” Likewise, one can choose to normalizeit if the administrative queries are structurally different fromthe normal-user queries, which is common case. Additionally,if the database can authenticate admin and nonadminusers, then privilege escalation attacks by changing valuesare not feasible (i.e., there is no session hijacking).

VIII. REFERENCES

[1]. H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusiondetection systems. Computer Networks, 31(8), 1999.
[2]. T. Verwoerd and R. Hunt. Intrusion detection techniques and approaches. Computer Communications, 25(15), 2002.
[3]. F. Valeur, G. Vigna, C. Krugel, and R. A. Kemmerer. A comprehensive ¨ approach to intrusion detection alert correlation. IEEE Trans. Dependable Sec. Comput, 1(3), 2004.
[4]. A. Seleznyov and S. Puuronen. Anomaly intrusion detection systems: Handling temporal relations between events. In RAID 1999.
[5]. Lee, Low, and Wong. Learning fingerprints for a database intrusion detection system. In ESORICS: European Symposium on Research in Computer Security. LNCS, Springer-Verlag, 2002.
[6]. Y. Hu and B. Panda. A data mining approach for database intrusion detection. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, SAC. ACM, 2004.
[7]. A. Srivastava, S. Sural, and A. K. Majumdar. Database intrusion detection using weighted sequence mining. JCP, 1(4), 2006.
[8]. K. Bai, H. Wang, and P. Liu. Towards database firewalls. In DBSec 2005.
[9]. G. Vigna, F. Valeur, D. Balzarotti, W. K. Robertson, C. Kruegel, and E. Kirda. Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries. Journal of Computer Security, 17(3):305–329, 2009.
[10]. S. Potter and J. Nieh. Apiary: Easy-to-use desktop application fault containment on commodity operating systems. In USENIX 2010 Annual Technical Conference on Annual Technical Conference.
[11]. Y. Huang, A. Stavrou, A. K. Ghosh, and S. Jajodia. Efficiently tracking application interactions using lightweight virtualization. In Proceedings of the 1st ACM workshop on Virtual machine security, 2008