International Journal of Science, Environment and Technology, Vol. 3, No 6, 2014, 1996 – 2009
ISSN 2278-3687 (O)

DATA PROTECTION IN WEB APPLICATIONS BY USING TRIPLE IDS

S. Magesh Kumar[1], E. Anbalagan[2] and Dr Siva Subramanian
[1]Research Scholar, Bharat University, Chennai
[2]Principal, Pallava Raja College of Engineering, Kolivakkam, Iyyengarkulam, Kanchipuram
[3]Dhanalakshmi College of Engineering, Chennai

Abstract: Web services and applications have increased over the past few years. Recently, the web applications are using multi tier design. The front end include web server which can used to process the request and gives that output to back end. An attention of hackers has been focus from attacking the front end and to taint the back-end database system. We present Triple guard system that models the network behaviour of user sessions across both the front-end web server and the back end database and also provides confidential service. By using Triple IDS, we able to ferret out attacks and free from traffic that occur in web application.
Keywords: Light weight virtualization, Anomaly Intrusion detection, Knowledge based Intrusion detection system.

1. INTRODUCTION

Intrusion detection system [1], [3] is a computer-based information system designed to collect information about malicious activities in a set of targeted IT resources, analyze the information, and respond according to a predefined security policy. The most common computer Triple IDS systems detect signatures of known attacks by searching for attack-specific keywords in network traffic. Triple IDS systems aim at detecting attacks against computer systems and networks or in general, against information systems. This strategy is mainly focus on to detect intrusion in multitier web applications [5],[13]. This strategy is useful to identify the intrusion at both front end and back end.

An Anomaly-Based Intrusion Detection System [14],[1] is a system for detecting computer intrusions, Knowledge Based Intrusion Detection System [1],[9] is used to detect the user as normal or anomalous by using signature and Network Based IDS [1] is used to monitor the traffic. Triple guard achieved by we employ a lightweight virtualization technique to assign each user's web session to a dedicated container [6], an isolated virtual computing environment. We use the container ID to accurately associate the web request with the subsequent DB queries. Thus, Triple guard system can build a causal mapping profile by

*Received Oct 15, 2014 * Published Dec 2, 2014 * www.ijset.net*

taking both the web server and DB traffic into account. An alternative is lightweight virtualization [16], generally based on some sort of container concept. With containers, a group of processes still appears to have its own dedicated system, but it is really running in a specially isolated environment. All containers run on top of the same kernel. With containers, the ability to run different operating systems is lost, as is the strong separation between virtual systems. Triple IDS might not want to give root access to processes running within a container environment. On the other hand, containers can have considerable performance advantages, enabling large numbers of them to run on the same physical host.

2. RELATED WORK

2.1 Intrusion Alerts

Intrusion Detection System (IDS) [1] can be classified into: network IDS, anomaly detection and misuse detection. Anomaly detection first requires the IDS to define and characterize the correct and acceptable static form and dynamic behaviour of the system, which can then be used to detect abnormal changes or anomalous behaviours [14]. The boundary between acceptable and anomalous forms of stored code and data is precisely definable. Behaviour models are built by performing a statistical analysis on historical data [11] or by using rule-based approaches to specify behaviour patterns. An anomaly detector then compares actual usage patterns against established models to identify abnormal events. Our detection approach belongs to anomaly detection, and we depend on a training phase to build the correct model. As some legitimate updates may cause model drift, there are a number of approaches that are trying to solve this problem. Our detection may run into the same problem; in such a case, our model should be retrained for each shift.

Intrusion alerts correlation [17] 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. Triple Guard differs from this type of approach that correlates alerts from independent IDS. Rather, Triple 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. An IDS uses temporal information to detect intrusions. Triple IDS, however, does not correlate events on a time basis, which runs the risk of mistakenly

considering independent but concurrent events as correlated events. Triple IDS does not have such a limitation as it uses the container ID for each session to causally map the related events, whether they are concurrent or not.

2.2 Knowledge Based IDS

Current state of the art intrusion detection and prevention systems (IDPS) are signature-based systems that detect threats and vulnerabilities by cross-referencing the threat or vulnerability signatures in their databases [9]. These systems are incapable of taking advantage of heterogeneous data sources for analysis of system activities for threat detection. This work presents a situation-aware Triple IDS model that integrates these heterogeneous data sources and builds a semantically rich knowledge-base to detect cyber threats/vulnerabilities.

2.3 Taint and Intrusion Detection

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 Triple IDS, 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 Triple IDS 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 behaviour.

In addition, validating input is useful to detect or prevent SQL or XSS injection attacks [12],[7]. This is orthogonal to the Triple IDS approach, which can utilize input validation as an additional defence. However, we have found that Triple - IDS can detect SQL injection attacks [2] by taking the structures of web requests and database queries without looking into the values of input parameters (i.e., no input validation at the web server).

2.4 Lightweight Virtualization

Virtualization [16] is used to isolate objects and enhance security performance. Full virtualization and Para-virtualization are not the only approaches being taken. An alternative is a lightweight virtualization, such as OpenVZ [4], Parallels Virtuozzo [6], or Linux-VServer. In general, these are based on some sort of container concept. With containers, a group of processes still appears to have its own dedicated system, yet it is running in an isolated environment. On the other hand, lightweight containers can have considerable

performance advantages over full virtualization or Para-virtualization. Thousands of containers can run on a single physical host. There are also some desktop systems [16] that use lightweight virtualization to isolate different application instances. Such virtualization techniques are commonly used for isolation and containment of attacks. However, in our Triple IDS, 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.

2.5 CLAMP Architecture

CLAMP [15] is architecture for preventing data leaks even in the presence of attacks. By isolating code at the web server layer and data at the database layer by users, CLAMP guarantees that a user's sensitive data can only be accessed by code running on behalf of different users. In contrast, Triple IDS focuses on modelling the mapping patterns between HTTP requests and DB queries to detect malicious user sessions. There are additional differences between these two in terms of requirements and focus. CLAMP requires modification to the existing application code, and the Query Restrictor works as a proxy to mediate all database access requests. Moreover, resource requirements and overhead differ in order of magnitude: Triple IDS uses process isolation whereas CLAMP requires platform virtualization, and CLAMP provides more coarse-grained isolation than Triple IDS. However, Triple IDS would be ineffective at detecting attacks if it were to use the coarse-grained isolation as used in CLAMP. Building the mapping model in Triple IDS would require a large number of isolated web stack instances so that mapping patterns would appear across different session instances.

3. SYSTEM ARCHITECTURE

This container-based [10] and session-separated web server architecture not only enhances the security performances but also provides us with the isolated information flows that are separated in each container session. It allows us to identify the mapping between the web server requests and the subsequent DB queries, and to utilize such a mapping model to detect abnormal behaviours on a session/client level. In typical three-tiered web server architecture, the web server receives HTTP requests from user clients and then issues SQL queries to the database server to retrieve and update data. These SQL queries are causally dependent on the web request hitting the web server. We want to model such causal mapping relationships.

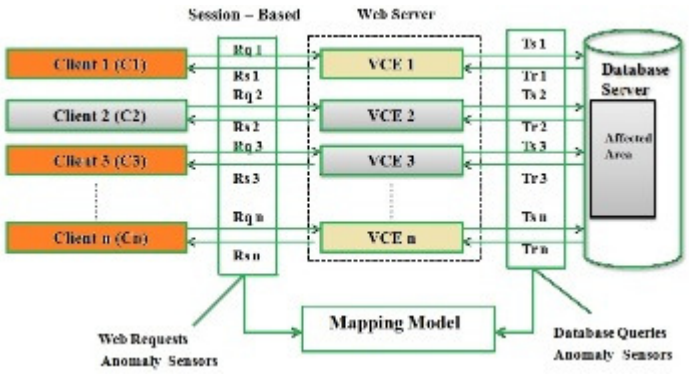

Figure 1. Triple Guard model

Figure 1 depicts how communications are categorized as sessions and how database transactions can be related to a corresponding session. Figure 1, Client 2 will only compromise the VCE 2, and the corresponding database transaction set T2 will be the only affected section of data within the database. 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. Section 5 further discusses how to build the mapping by profiling session traffics.

To that end, we put 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. 1 shows the locations of our sensors.

Once we build the mapping model, it can be used to detect abnormal behaviours. Both the web request and the database queries within each session should be in accordance with the model. If there exists any request or query that violates the normality model within a session, then the session will be treated as a possible attack.

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.

4. 1.Inferring Mapping Relations

Mapping relation explain about how the request and corresponding query are matched, causal relationship between rm to {qn,qp).Here qn, qp are mention the different database query.

The possible mapping patterns as follows,

- Deterministic mapping
- Empty Query set
- No matched Request
- Non deterministic mapping

4. 1.1 Deterministic mapping

This is the most common and perfectly matched pattern. That is to say that web request rm appears in all traffic with the SQL queries set Qn. (ie)Mmapping pattern is then rm to Qn. 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.

4. 1.2 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.

4. 1.3 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. These queries cannot match up with any web requests, and we keep these unmatched queries in a set NMR. During the testing phase, any query within set NMR is considered legitimate. The size of NMR depends on web server logic, but it is typically small.

4. 1.4 Non Deterministic Mapping

The same web request may result in different SQL query sets based on input parameters or the status of the webpage at the time the web request is received. In fact, these different SQL query sets do not appear randomly, and there exists a candidate pool of query sets (e.g.,

{Qn;Qp;Qq . . .}). Each time that the same type of web request arrives, it always matches up with one (and only one) of the query sets in the pool. Therefore, it is difficult to identify traffic that matches this pattern. This happens only within dynamic websites, such as blogs or forum sites.

5. ATTACK DETECTION

Once the model is built, it can be used to detect malicious Sessions. Triple IDS detect the following types of attack.

5.1 Privilege Escalation Attack

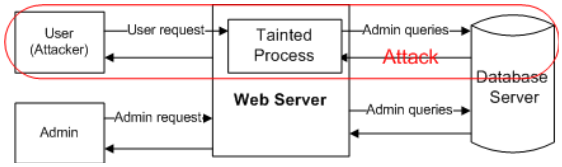

Figure2. Privilage Escalation Attack

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 [8] since the DB query Qa does not match the request ru, according to our mapping model.

5.2 Hijack Future Session Attack

This class of attack is mainly based on web server side. An attacker takes over web server by hijack the other user sessions by sending spoofed replies. In Triple IDS it is detected by causal mapping of a request without query. 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 user's sessions.

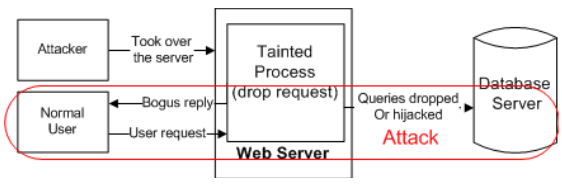

Figure 3. Hijack future session attack

5.3 Injection Attack

Attacks such as SQL injection [12] 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

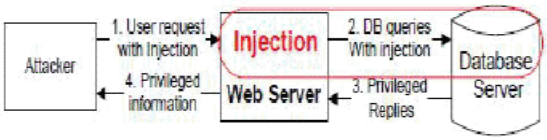

Figure 4. Injection attack

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.

5.4 Direct DB Attack

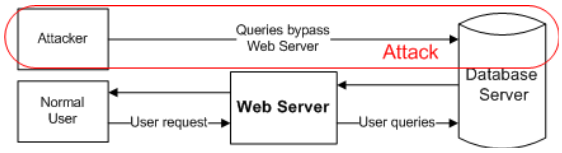

Figure 5. Direct DB attack

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

6. IMPLEMENTATION

6.1 State modeling algorithm

We developed an algorithm that takes the input of training data set and builds the mapping model. 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. The algorithm generates the mapping model**.**

Algorithm 1 Static Model 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

Testing for Static Websites

The testing phase algorithm is as follows:

1) If the rule for the request is Deterministic Mapping r→Q (Q ≠ ø), 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→ø, 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. As expected, this is also demonstrated in our experiments.

6.2 Dynamic modelling

The algorithm for extracting mapping patterns in static pages no longer worked for the dynamic pages, we created another training method to build the model. First, we tried to categorize all of the potential single (atomic) operations on the web pages. For instance, the common possible operations for users on a blog website may include reading an article, posting a new article, leaving a comment, visiting the next page, etc. All of the operations that appear within one session are permutations of these operations. If we could build a mapping model for each of these basic operations, then we could compare web requests to determine the basic operations of the session and obtain the most likely set of queries mapped from these operations. If these single operation models could not cover all of the requests and queries in a session, then this would indicate a possible intrusion. Interestingly, our blog website built for testing purposes shows that, by only modeling nine basic operations, it can cover most of the operations that appeared in the real captured traffic. For each operation (e.g., reading an article), we build the model as follows. In one session, we perform only a single read operation, and then we obtain the set of triggered database queries. Since we

cannot ensure that each user perform only a single operation within each session in real traffic, we use a tool called Selenium to separately generate training traffic for each operation. In each session, the tool performs only one basic operation. When we repeat the operation multiple times using the tool, we can easily substitute the different parameter values that we want to test (in this case, reading different articles).Finally, we obtain many sets of queries from one session and assemble them to obtain the set of all possible queries resulting from this single operation.

By placing each rm, or the set of related requests Rm, in different sessions with many different possible inputs, we obtain as many candidate query sets {Qn, Qp, Qq... } as possible. We then establish one operation mapping model Rm → Qm (Qm = Qn U Qp U Qq U ...), wherein Rm is the set of the web requests for that single operation and Qm includes the possible queries triggered by that operation. Notice that this mapping model includes both deterministic and non-deterministic mappings, and the set EQS is still used to hold static file requests. As we are unable to enumerate all the possible inputs of a single operation (particularly write type operations), the model may incur false positives.

6.3 Detection algorithm

There are number of attacks performed by the attacker to retrieve the data from the web server or directly from the database. The attacks will be detected and controlled by using detection algorithm. In this algorithm the structure of the query, session id, session time and the user id will be compared with the information stored in the database and the web server the query will be processed only when the every condition will be satisfied otherwise the query will be neglected.

7. RESULT AND DISCUSSION

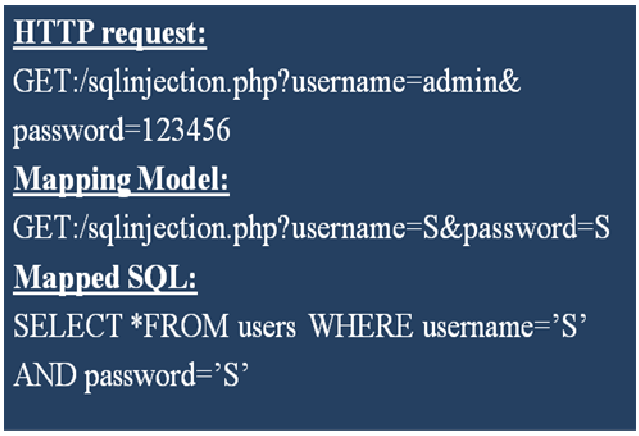

Figure: 6 Mapping Model of rm to qn

According to our experimental results, we normalized the value of ‘admin’ and ‘123456’, and repeated the legitimate login process a few times during the training phase. The mapping model that was generated is shown in Figure 6 (S stands for a string value), where the generalized HTTP request structure maps to the SQL queries. After the training phase, we launched an SQL injection attack that is shown in Figure 4. Note that the attacker was not required to know the user name and password because he/she could use an arbitrary username the password 1’ or ’1=1, which would be evaluated as true. After normalizing all of the values in this HTTP request, we had the same HTTP request as the one in Figure 6. However, the database queries we received do not match the deterministic mapping we obtained during our training phase.

Another result is as follows for known privilege escalation vulnerability. There was a vulnerable check “if (strpos($ SERVER[‘PHP SELF’], ‘wp-admin/’)!== false) $this->is admin = true;” that used the PHP strpos() function to check whether the $ SERVER[‘PHP SELF’] global variable contained the string “wp-admin/”. If the strpos() function found the “wp-admin/” string within the $ SERVER[‘PHP SELF’] variable, it would return TRUE, which would result in the setting of the “is admin” value to true. This ultimately granted the user administrative rights to certain portions of the web application. The vulnerable code was corrected to “if (is admin()) $this->is admin = true;” in a later version, which added a function to determine whether the user has administrative privilege. With the vulnerable code, an unauthorized user could input a forged URL like “http://www.myblog.com/index.php/wp-admin/” so as to set the value of variable $this->is admin to TRUE. This would allow the unauthorized user to access future, draft, or pending posts that are administrator-level information.

8. CONCLUSION

In this way we surveyed for Triple IDS system that builds normality model for multitier web applications. Some of the technique use double IDS to detect and prevent web server from malicious request while some approach use combined approach to detect intrusions at both web and database level. Unlike previous approaches this approach forms container based IDS with multiple input streams to produce alerts. There will be lightweight virtualization technique to assign session ID to a dedicated container and having some additional detection capability to detect attack where normal traffic is used as means to launch database attack. This approach provides a better characterization of the system for Anomaly detection and

Knowledge based detection because the intrusion sensor has a more Precise normality model that detects a wider range of threats. This approach is more advantageous in sense that monitoring both web and subsequent database requests, we are able to detect attacks easily and provide data security.

REFERENCES

[1] Types of Intrusion Detection System.

[2] Five common web application vulnerabilities.
http://www.symantec.com/connect/articles/five-common-web-application-vulnerabilities.

[3] http://www.sans.org/top-cyber-security-risks/.

[4] Openvz. http://wiki.openvz.org.

[5] Multi-tierWebApplication. http://en.wikipedia.org/wiki/Multitier_architecture.

[6] Virtuozzo containers. http://www.parallels.com/products/pvc45/.

[7] Greensql. http://www.greensql.net/.

[8] A. Schulman. Top 10 database attacks.
http://www.bcs.org/server.php?show=ConWebDoc.8852.

[9] A Knowledge-Based Approach To Intrusion Detection Modeling Sumit More, Mary Matthews, Anupam Joshi, Tim Finin Computer Science and Electrical Engineering University of Maryland, Baltimore County Baltimore, MD, USA fsumitm1, math1, joshi, fining@umbc.edu

[10] Double Guard: Detecting Intrusions In Multi-tier Web Applications Meixing Le Angelos Stavrou Brent Byung Hoon Kang, George Mason University.

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

[12] Intrusion Detection and Prevention System: SQLInjection Attacks Varian Luong *San Jose State University*

[13] Intrusion Detection In Multitier Web Applications Shenbagalakshmi Gunasekaran, K. Muneeswaran, *Department of Computer Science and Engineering, Mepco Schlenk Engineering College, TamilNadu, India.*

[14] Anomaly Detection of Web-Based Attacks**.** *C. Kruegel and G.Vigna Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03), Oct. 2003*

[15] B. Parno, J.M. McCune, D. Wendlandt, D.G. Andersen, and A. Perrig. CLAMP: Practical prevention of large-scale data leaks. In IEEE Symposium on Security and Privacy. IEEE Computer Society, 2009.

[16] 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.

[17] F. Valeur, G. Vigna, C. Kr¨ugel, and R.A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. IEEE Trans. Dependable Sec. Comput, 1(3), 2004.