An Anomaly-driven Reverse Proxy for Web Applications

Fredrik Valeur
University of California
Santa Barbara
fredrik@cs.ucsb.edu

Giovanni Vigna
University of California
Santa Barbara
vigna@cs.ucsb.edu

Christopher Kruegel
Technical University
Vienna
chris@cs.ucsb.edu

Engin Kirda
Technical University
Vienna
ek@infosys.tuwien.ac.at

ABSTRACT

Careless development of web-based applications results in vulnerable code being deployed and made available to the whole Internet, creating easily-exploitable entry points for the compromise of entire networks. To ameliorate this situation, we propose an approach that composes a web-based anomaly detection system with a reverse HTTP proxy. The approach is based on the assumption that a web site's content can be split into security sensitive and non-sensitive parts, which are distributed to different servers. The anomaly score of a web request is then used to route suspicious requests to copies of the web site that do not hold sensitive content. By doing this, it is possible to serve anomalous but benign requests that do not require access to sensitive information, sensibly reducing the impact of false positives. We developed a prototype of our approach and evaluated its applicability with respect to several existing web-based applications, showing that our approach is both feasible and effective.

Keywords

Anomaly Detection, Web Proxy, Data Distribution

1. INTRODUCTION

Web-based applications represent a serious security exposure. These applications are directly accessible through firewalls by design, and, in addition, they are often developed in a hurry by programmers who focus more on functionality and appearance than security. As a result, web-based applications have recently become the primary target of attempts to compromise networks. This is confirmed by an analysis of the web-related vulnerabilities published in the CVE database, which showed that the percentage of web-related vulnerabilities increased from 16% in 1999 to 26% in 2004 [7]. Furthermore, recent web-based compromises, such as the Tower Records incident [13] and the Victoria Secret incident [18], cost a considerable amount of money in terms of settlements paid to users whose private information was disclosed as a result of the attacks.

The security of web-based application should be addressed by means of careful design and thorough security testing, but, unfortunately, in the real world it is often the case that web-accessible vulnerable systems store and manage sensitive information. For this reason, security-conscious development methodologies should be complemented by an intrusion detection infrastructure that is able to provide early warning about malicious activity.

Web-based attacks can be detected (and possibly blocked) by intrusion detection systems (IDSs) that use signature-based techniques. For example, Snort 2.3 [14] devotes 1006 of its 2564 signatures to detecting web-related attacks. However, many web-based attacks cannot be easily modeled by signatures because they are application-specific, and, in addition, they often do not contain any common characterizing feature (such as the NOP sledge in a buffer overflow attack).

To deal with attacks that are tailored to a specific web-application, anomaly detection systems have been proposed that characterize the "normal" use of a web-based application. These systems are able to block web requests that do not fit the established parameters of "normality" [5, 8]. Unfortunately, anomaly detection systems are prone to false positives due to over-simplified modeling techniques or insufficient training. Therefore, if the anomaly score of a web request is used as the basis to deny access to a web site, a false positive may cause the denial of a legitimate request. This situation is the result of an "all-or-nothing" approach, in which all requests that are identified as anomalous are automatically considered malicious. In the real world, however, anomalous requests may be benign.

The "all-or-nothing" approach is often applied to the design of the web site content as well. This means that sensitive content (e.g., private information about users) and non-sensitive content (e.g., a product catalog) are both stored and accessible on the same server.

To mitigate the problem of false positives generated by anomalous but benign requests, we propose a novel solution based on data compartmentalization and anomaly-based reverse proxying. The idea is to replicate a web site on two or more "sibling" servers with different levels of privilege (e.g., different levels of access to sensitive information). An anomaly detection system then scores incoming requests and uses the anomaly score to drive a reverse HTTP proxy. A reverse HTTP proxy is a proxy application that is deployed close to one or more web servers. The proxy intercepts HTTP requests for the web servers, performs some process-

SAC'06 April 23-27, 2006, Dijon, France

ing, and subsequently forwards the request to one of the web servers. Typically, reverse proxies are used to perform load balancing among multiple servers or to increase performance by caching static content. In our case, the purpose of the reverse proxy is to forward anomalous requests to sibling servers with only limited (or no) access to sensitive content. By doing this, it is possible to provide some form of service even in the case of anomalous requests.

For example, in the case of an e-commerce site, a sibling server might provide access to product-related information (e.g., a product catalog) and no (or fake) sensitive information about users. Another sibling server would instead be able to access all sensitive information. In this case, anomalous queries are directed by the reverse proxy to the first sibling, while normal queries are forwarded to the second (fully functional) server. If a user performs an anomalous but benign query that does not involve any sensitive data, the query will be served by the first sibling and the user will receive the correct information. Requests that are both anomalous and malicious, however, will not be able to access sensitive information.

This paper presents our approach to mitigate the impact of false positives, describes a prototype implementation of our system, and provides a discussion of its applicability to real-world applications. More precisely, in Section 2 we discuss different high-level designs to achieve processing of web requests at different levels of privilege. Then, in Section 3, we describe how web requests are first analyzed to determine their anomaly score and then routed to different sibling web servers. In Section 4, we provide a discussion of the applicability of our approach with respect to several existing web-based applications. Finally, Section 5 discusses related work and Section 6 briefly concludes.

2. CHARACTERIZING INFORMATION

Web-based attacks are aimed at either obtaining control of the host running the web server application (e.g., through a buffer overflow) or disclosing sensitive information (e.g., through an SQL injection attack that dumps the content of a database table).

The first type of attack is caused by vulnerabilities in the web server software or in a server-side web-based application that allow one to compromise the security of the underlying host. The second type of attack is usually made possible by the fact that a single back-end database is used to store all the persistent information of a web-based application. Therefore, by exploiting a vulnerability in code that in principle has access to a limited portion of the database contents, it is possible to extend one's access to the database and retrieve sensitive information. For example, Figure 1 (a) shows an e-commerce web site implemented with a single server that relies on a single back-end database and that accesses a credit card processing server. All the application functions (i.e., `f1`, `f2`, and `f3`) have the same level of access to the database, even though they use different tables (for example, `f2` uses table `x` only, while `f1` does not use any table of the database).

A web site could be made more resilient to attacks if it would be possible to design both the server and the database infrastructure so that different levels of access to the database and the hosts running the server processes could be clearly enforced.

For example, the e-commerce application could be structured so that: (i) non-sensitive, static information about the e-commerce company (e.g., company contacts and support information) is accessible through one server; (ii) the non-sensitive, dynamic information about product availability is accessible through a second server that accesses a product database; and, finally, (iii) the sensitive information about users is accessible through a third server that relies on a user database, which is separated from the product database. This last server has also access to the credit card processing server. This is the design shown in Figure 1 (b), where each function is implemented on a different server and accesses only the needed information.

In this example, the compromise of the server providing product information would not allow the attacker to access the database containing the sensitive information about the users or the credit card transaction server.

Unfortunately, this type of "compartmentalized" design cannot be easily applied to existing applications, where the different functions provided by a web site are closely intertwined.

Therefore, we propose an alternative design where the web site's contents are replicated across servers, instead of being partitioned. In addition, the database used by the web-based application is extended with user accounts at different levels of privilege. Each of these accounts is associated with one of the replicated servers. Then, an anomaly detection system is used to determine the likelihood that a request represents an attack. This information is used by a reverse web proxy to forward the request to the web server that is able to provide the best possible level of service, given the anomaly score of the request.

According to this design, the e-commerce application described above would be implemented as shown in Figure 1 (c). In this case, all the site's functionality (i.e., `f1`, `f2`, and `f3`) is replicated on three servers (`A`, `B`, and `C`). This step requires no modification of the original application. The web server proxy is configured so that queries that are highly anomalous are sent to server `A`, queries that are considered moderately anomalous are sent to server `B`, and normal queries are sent to server `C`. Server `C` is the only one that is able to access the credit card server.

The database is modified to create two different users `u1` and `u2`, where `u1` is allowed to access table `x` only and `u2` is able to access both table `x` and table `y`. User `u1` is associated with server `B` and user `u2` is associated with server `C`.

In this design, if a request that uses function `f2` is sent to server `A`, the request will fail because `f2` uses the database but no connections are allowed to the database. On the other hand, a highly anomalous request for function `f1` will be correctly executed. If identifying this request as highly anomalous represents a false positive, then the user will not be denied access. If the request actually represents an attack, then two cases are possible. In the first case, the attack aims at accessing the database, and, in this case, the attack will be foiled. In the second case, the attack aims at compromising the host. In this case, assuming that the attack is successful, the host being compromised has no access to the credit card server and the damage is contained. Note also that this host is likely to serve a small portion of the requests and therefore it could be "hardened" to be more resilient to certain types of attacks at the cost of some performance degradation (as it is done, for example, in [15]).

Moderately anomalous requests are sent to server `B` in

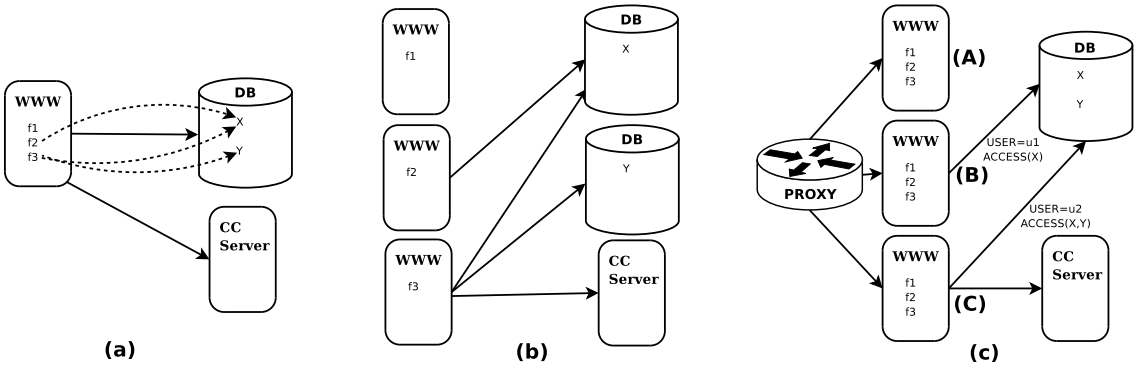

Figure 1: Web site designs.

this design. If a request of this type uses function `f2`, it is executed correctly, since server `B` has access to the database tables utilized by `f2`. Note that if the request would have failed if it had been sent to the more restrictive server `A`.

3. ROUTING WEB REQUESTS

We implemented the proxy-based approach described in the previous section by integrating an existing anomaly detection system into a reverse web proxy that we developed. The proxy is located at the public IP address of the web site being protected and receives all incoming HTTP requests. When a request is received, it is parsed and forwarded to the anomaly detector component, which calculates a score indicating how anomalous the request is. If the anomaly score is below a certain tunable threshold, i.e., the request is not suspicious, the request is forwarded to the real web server for processing. If a request is deemed anomalous, it is forwarded to a locked down system, where a malicious request would not be able to cause any harm.

The anomaly component of the proxy is based on the WebAnomaly system presented in [8]. The anomaly detector first extracts from the requested URL the path to the web application being invoked, along with the arguments passed to it. The anomaly detector then looks up the *profile* associated with the web application. A profile is a collection of statistical models which is associated with one specific web application.

The anomaly detection models contained in the profile are a set of procedures used to evaluate a certain feature of a query attribute, and operate in one of two modes, learning or detection. In the learning phase, models build a profile of the "normal" characteristics of a given feature of an attribute (e.g., the normal length of values for an attribute), setting a dynamic detection threshold for the attribute. During the detection phase, models return an anomaly score for each observed example of an attribute value. This is simply a probability on the interval $[0, 1]$ indicating how anomalous the observed value is in relation to the established profile for that attribute. Since there are generally multiple models associated with each attribute of a web application, a final anomaly score for an observed attribute value during the detection phase is calculated as the weighted sum of the individual model scores. The overall anomaly score is then used to decide which web server should process the request. For more information on the models themselves, as well as the anomaly detector as a whole and its evaluation on real-world data, please refer to [8].

Each of the back-end web servers has a different privilege level and processes requests with a different range of anomaly scores. For instance, the requests determined to be normal are sent to the web server with the highest privilege level, while anomalous requests are sent to a less privileged server. Depending on the privilege level of the server, the amount of sensitive information it can access varies. What information each web server can access is determined by what database it is connected to. The most privileged web server is connected to a database that contains all the information, while less privileged servers are connected to databases where part or all of the sensitive data is removed.

Instead of using multiple physical databases, our system implements the multiple database concept using multiple users within a single database system. Each web server is connected to the database as a different user. The access control system of the database is configured so that each user is only able to access the records associated with the privilege level of the corresponding web server. The advantage of using only one physical database is that it is much simpler to implement because no database replication is needed. The problem with this method is that the access control mechanisms of most databases are not very flexible. It is for instance easy to deny access to a table, but it is not easy to deny access to any row containing the string "sensitive".

Another possible way of implementing the split back-end database is using multiple physical database servers. The *master database* would contain an unaltered copy of the sensitive information. The non-sensitive data stored in the master database would be replicated to the less privileged servers. Depending on the privilege level of the database, more or less information would be censored. Replication of the databases would be done in a lazy manner when it is not critical that the censored databases contain completely up-to-date information. However, this approach is difficult to implement, as it might be hard to determine what information can be replicated in a lazy manner and what information must be immediately available to all web servers. For instance, if session information is stored in the database, lazy updates can not be used. Consider, for example, two requests performed in the same session, where one of the requests is flagged as anomalous while the other is determined to be normal. These two requests might be served by two different web servers, which would need to access consistent

state information.

In order to prevent potential attackers from simply bypassing the proxy by connecting directly to the backed web server, efficient firewalling has to be deployed in order to prevent this attack vector. We achieved this by assigning non-routable IP addresses to the web servers and deploying the servers on a network isolated from the Internet. A separate interface on the proxy server connects the proxy to the isolated network.

4. DISCUSSION

The purpose of this section is to demonstrate that our reverse proxy technique is capable of reducing the negative impact of false positives. To this end, we have to provide evidence that a considerable fraction of web requests can be handled by an application without relying on access to sensitive information stored in the database. Unfortunately, providing such evidence in a general fashion is difficult for the following reasons:

1. The fraction of requests that access sensitive information depends on the type of application. For example, a news portal that uses a read-only database will probably never require access to sensitive tables since all information should be publicly available. An application that is used by a company to keep track of the working hours of its employees, on the other hand, will most likely require a significant amount of access to sensitive data.

2. Even when only a single application is analyzed, its site-specific usage might dramatically influence the number of sensitive operations performed. For example, in a web shop application, the ratio between users who anonymously browse through the catalogs and users who actually login and purchase goods will determine how many of the requests need full access.

3. Finally, it is not always clear which database tables and operations should be classified as sensitive. While most people would agree that modifications to a table with user data should be treated as being privileged, it is not always that obvious. In a discussion board application, for example, should one consider the list of subscriptions to topics of interest as being sensitive because disclosure might violate user privacy? The answer may vary: When the topics are related to sport events, then the answer is likely no. When the discussions are on incurable diseases, the answer might be different.

To address the problem of the different types of web-based services, we selected three programs that represent a mix of typical web applications that require substantial program logic and a back-end database. An important requirement during the selection process was that the applications had to perform sensitive operations during normal usage. For our experiments, we chose `phPay 2.0` [12], `myBloggie 2.1.2` [9], and `punBB 1.2.5` [4]. `phPay` is a typical web-shop application that supports products in different categories, search functionality, user management, and on-line payment. `myBloggie` is a web-log script that allows users to lead an on-line diary and to comment on other people's public entries. `punBB` is a discussion board that supports different forums, rich text formatting, as well as user management and notification if messages are posted on subscribed topics. All three programs are written in `PHP` [11] and use the `MySQL` database [10] to store information. Table 1 provides more information on the selected applications such as the number of different `PHP` source files, their total lines of code, and the number of used database tables.

Name	Source Files	Lines of Code	Database Tables
`phPay 2.0`	43	3,023	29
`myBloggie 2.1.2`	41	5,277	4
`punBB 1.2.5`	56	15,993	17

Table 1: Applications used for the experiments.

4.1 Sensitive Path Coverage

Given our selection of web-based applications, the goal of our experiments is to assess the fraction of required sensitive data accesses. On one hand, we attempted to measure this fraction in a way that is independent of a certain site-specific work load. To this end, we performed static analysis on the application code as described below. On the other hand, and to complement the results of the static analysis, we also setup the three applications in a live test environment and run several attacks against them (refer to Section 4.3 for more details).

For our site-independent program analysis, we leveraged the concept of *path coverage*, a well-known metric in software testing that measures the fraction of execution paths through the program that are covered by test cases. The key idea is to adapt this metric and determine the fraction of all possible execution paths through the application that perform sensitive operations on the database. We call this fraction the *sensitive path coverage*. Note that the sensitive path coverage should not be mistaken for a precise estimate of the number of sensitive operations that can be expected to be performed. We assume here that all paths through an application occur with equal probability. Therefore, the result is more useful as a measure of how much code that accesses non-sensitive information is interspersed with code that performs sensitive database accesses. When sensitive operations are rare and cleanly separated from the rest of the application code (i.e., the sensitive path coverage is small), we expect the reverse proxy approach to be more successful.

Unfortunately, pure path coverage is impractical in practice. The reason is the problem of exponential path explosion that is caused by the fact that a code fragment with a succession of k decisions (e.g., due to if-statements) contains up to 2^k different execution paths. Also, many of these paths may be infeasible, i.e., there is no input to the program that can cause a particular path to be executed. Therefore, one usually attempts to aggregate paths to clusters or to operate at a higher level than individual statements. We also utilize this common strategy and calculate the possible *paths at the function level*. That is, we do not consider any intra-procedural control flow. To this end, we first generate a call graph in which each node of the graph represents an individual function. An edge is introduced from node v to node w when the function corresponding to node v calls the function corresponding to w. Based on this graph, we then determine the fraction of paths that perform sensitive operations.

In addition to the exclusive focus on inter-procedural control flow, we also need to perform path aggregation. Again,

the problem is that a node with k successor nodes in the call graph would account for 2^k paths, leading to path explosion and an excessive impact of functions that call many other functions. To reduce the influence of nodes with many successors, we use Equation 1 to determine the number of (aggregated) paths $p(N)$ through a node N in the call graph.

$$p(N) = 1 + k * \sum_{i=1}^{k}(p(S_i)) \quad (1)$$

In this equation, k is the total number of successor nodes of N, and S_i represents the number of paths through the i^{th} successor node. It can be seen that in this equation, the factor 2^k, which would be necessary to precisely count the number of possible paths, has been replaced with k. The number $p(R)$ of the root node R of the call graph denotes the total number of possible paths (or aggregated path clusters, to be more precise) through the application. Note that the algorithm would not terminate when the call graph contains loops as a result of direct or indirect recursive function calls. Therefore, such loops are removed from the call graph prior to processing, by using a simple depth first search.

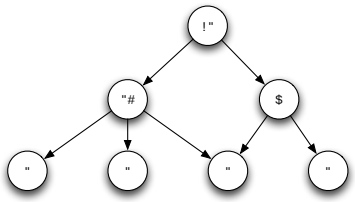

Figure 2: Call graph and path calculation.

Consider the example in Figure 2, which demonstrates the calculation of the aggregated paths for all nodes of a small call graph. Given the recursive definition of $p(.)$, which calculates the number of paths at a node based on the number of paths through all its successor nodes, the algorithm evaluates the nodes of the call graph in bottom-up order. The value of a leaf node is always 1. The total number of aggregated paths through the application in this example is 31.

In the next step, we have to determine the number of (aggregated) paths that pass through nodes that correspond to functions that perform sensitive operations. We call such paths *sensitive paths*. Given both the number of sensitive paths and the total number of paths, we can easily calculate the sensitive path coverage.

To find all sensitive paths, we have to first identify sensitive operations. In our current analysis, sensitive operations are found by checking for string constants in the `PHP` source code that performs database operations on privileged tables. Once all sensitive operations are found, each call graph node corresponding to a function that contains these operations is appropriately marked.

As mentioned previously, the exact types of operations and database tables that are considered sensitive depend on the application and the environment where it is deployed. We will discuss examples of different sensitive operations in more detail when describing the experiments with our three web applications in Section 4.2.

Hereinafter, we assume that all `SQL` operations can be seen as strings written into function bodies. This implies that the program neither utilizes global variables to store `SQL` statements, nor that they are generated in a completely dynamic fashion during run-time. Of course, it is possible (and common) that the parameters for the `where` clause of a `select` statement or the values for an `insert` operation depend on variables. However, we assume that the type of the `SQL` operation and the tables that the program operates on are statically known. Fortunately, the direct inlining of `SQL` statements is a very common programming practice in `PHP`, and, indeed, in all the programs we considered for our experiments all database access statements were directly embedded as strings that specify both the type of operation and the accessed tables.

The technique to calculate the number of sensitive paths that run through the marked nodes is very similar to the previous technique for deriving the total number of paths. When a node is marked, all paths that run through it are automatically considered sensitive. If a node contains no sensitive operations itself, but sensitive paths run through (some of) its successor nodes, the sensitive paths $ps(N)$ of the current node N are derived using Equation 2.

$$ps(N) = k * \sum_{i=1}^{k}(ps(S_i)) \quad (2)$$

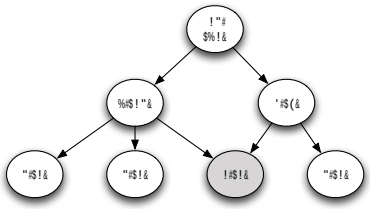

Figure 3: Call graph and sensitive path calculation.

The previous call graph example has been extended in Figure 3 by marking a node as containing sensitive operations (the node that is drawn shaded in the figure). In addition to the numbers of the total paths at each node, which are given in parenthesis, the number of sensitive paths for each node is shown.

4.2 Experimental Results

To be able to calculate the sensitive path coverage for a program, source code analysis is necessary. In particular, one has to generate the call graph and identify the `SQL` statements that represent sensitive database operations. To this end, we have developed a static analysis tool that can process `PHP` code, implementing the complete `PHP 4` standard. The choice of `PHP` was motivated by the fact that many web-based applications are developed in `PHP`. We used our tool to determine the sensitive path coverage for the three sample applications under different assumptions of what constitutes a sensitive database access. In the following paragraphs, the results are presented for each program. A summary of the results is shown in Table 2.

`phPay`: In a first step, we considered an access to the database as critical when it can modify the stored data. That is, we calculated the sensitive path coverage under the assumption that all `SQL` `insert`, `update`, and `delete` operations are sensitive. The result was a path coverage of 8.92%, mostly caused by modifications to the shopping cart table when items are purchased or quantities updated. This shows that

Program	Sensitive Operations	Sensitive Path Coverage (in %)
phPay	write	8.92
	write + read (user)	100.00
	write + modified read (user)	57.48
	write + mod. read (user,order)	57.48
	write + mod. read (user,order,cart)	78.84
myBloggie	write	100.00
	write w/o comments	2.37
	mod. write + read (user)	100.00
	mod. write + read (user.pass)	29.11
punBB	write	1.17
	write + read (user)	100.00
	write + modified read (user)	1.17
	write + mod. read (user, subscription)	1.17

Table 2: Sensitive path coverage results.

large parts of a web shop can be used without any write access to the database.

In the next step, we extended the definition of sensitive accesses to include `select` queries to the user table. The reason is that this table stores the user passwords, and thus can provide an attacker with valuable information. Unfortunately, the initial sensitive path coverage yielded 100%. Closer analysis revealed that a `select` query to the user table was included in the program's main method to be able to display the user names of people currently logged in. However, it is easy to perform a simple modification to the program (refactoring) and either remove this non-critical functionality or mirror the user name in the session information. In this case, the sensitive path coverage drops to 57.48%, indicating that a significant amount of requests can be successfully served without requiring access to the user table at all.

Finally, we included read access to other privacy-related tables into the set of sensitive operations. In particular, the table that stores the pending orders and the table that stores the cart data of currently shopping users were considered. By including the first table, the sensitive path coverage remained at 57.48%; when the second table was included, the sensitive path coverage increased to 78.84%.

`myBloggie`: Similar to the previous application, we checked the fraction of sensitive paths that perform (potential) write operations to the database. Initially, we received a sensitive path coverage of 100%. This was not surprising, as the program allows anyone to add comments to blog entries, a functionality that is included into the body of the main function. However, when checking for write access to all tables except the one for comments, the coverage is reduced to only 2.37%.

In the next step, we extended the set of sensitive operations with read access to the user table (which stores the user passwords in an encrypted field). Again, a path coverage of 100% was determined. This time, the reason was an `SQL` statement in the code that adds the user names to all log entries and comments that are displayed. When the application is slightly modified to directly store the user names with the log entries and the comments, then the sensitive path coverage is reduced to 29.11%.

`punBB`: Analyzing the path coverage under the assumption that only write operations are sensitive, we obtained sensitive path coverage of 1.17% for the message and discussion board applications. This unexpectedly low value is due to the fact that the code to insert new message items was confined to a single function that appeared close to the leafs of the call graph (and thus, is probably too small compared with what would be actually observed during normal program usage). However, the metric provides a good indication that the code is well-structured and shows that users who are only reading the presented information will be successfully served when using the reverse proxy, even when submitting anomalous queries (e.g., anomalous search queries).

When including the user table into the set of critical tables, the path coverage reaches 100%. For this application, the culprit is an `SQL` statement in the main function that generates the statistics of all people that are currently logged in. When this statement is removed, the sensitive path coverage is reduced to 1.17%. Alternatively, the statistics could be generated from a dedicated table that stored only the names of the currently logged-in users, separately from the passwords. The sensitive path coverage remains the same when the access to the subscription table is included into the set of sensitive operations. This observation strengthens the belief that the code is well-arranged because manipulation to user state (either directly to the user table, or indirectly to the subscription table) is well-encapsulated and located together. We decided to include the subscription table because reading this table could lead to privacy problems in case people do not want to reveal their interest in certain topics.

For all three applications, our analysis indicates that the sensitive path coverage is quite small when classifying only database write operations as sensitive. This means that, using our reverse proxy approach, most anomalous requests could be successfully served when sensitive tables are accessible in read-only mode. With regards to read operations of sensitive tables (e.g., the user table with passwords), all programs initially showed 100% sensitive path coverage. However, with only trivial modifications to non-critical parts of functionality, this value could be significantly reduced.

4.3 Live System Evaluation

The focus of our approach is on preventing novel web-based attacks from accessing sensitive information while, at the same time, reducing as much as possible the impact of false positives.

The effectiveness of the approach depends on a number of application-specific conditions, as discussed in the previous sections. However, we performed a number of tests in a live setting to provide some useful insight on the general

Program	Training Set	Test Set	Anomalous Requests	Attacks	False Positives	Failed Benign Requests
phPay	2,530	371	21	10	11	1
myBloggie	3,247	488	28	20	8	0
punBB	1,292	422	16	10	6	1

Table 3: Live experiment results.

applicability of our technique.

Therefore, we installed the phPay, myBloggie, and punBB applications on two web servers, called smart and dumb. The two web servers used the same back-end MySQL database, but the smart server was able to access all of the application-specific database content, while the dumb server was able to access non-sensitive information only.

The two servers were protected by our anomaly-driven reverse HTTP proxy. The anomaly detection component was trained semi-automatically using 7,069 requests to the three test applications. For this experiment, we set the thresholds used by the anomaly detection system in a conservative fashion to produce a larger-than-usual number of false positives.

We then switched the IDS component to detection mode and executed 1,281 application requests. This traffic contained 40 attacks generated using 4 exploits that we collected from public security forums or that we developed ourselves. These exploits included several SQL injections (Bugtraq ID 14195/a, Bugtraq ID 13507, and one *ad hoc* attack against phPay) and an information tampering attack (Bugtraq ID 14195/b).

The system was able to classify all of the attacks as anomalous and forwarded the corresponding HTTP requests to the dumb server. As a consequence, the attacks were not able to access sensitive information or to modify sensitive content. Because of the conservative way in which the threshold were set, 25 benign requests were also marked as anomalous and forwarded to the dumb server. These requests represent a (somewhat artificially) large number of false positives. However, only two of these requests required access to sensitive information, and, therefore, it could not be completed correctly. The remaining 23 benign requests were served by the dumb server with no effect on the overall performance of the system.

The results of this experiment are shown in Table 3. Even though no general claim can be derived from these results, the experiment demonstrated that the system is actually able to sensibly reduce the impact of false positives for a representative class of web-based applications.

5. RELATED WORK

The detection of web-based attacks has recently received considerable attention because of the increasingly critical role that web-based services are playing. For example, in [2] the authors present a system that analyzes web logs looking for patterns of known attacks. A different type of analysis is performed in [3] where the detection process is integrated with the web server application itself. In [19], a misuse-based system that operates on multiple event streams (i.e., network traffic, system call logs, and web server logs) is proposed. The system demonstrates that it is possible to achieve better detection results when taking advantage of the specificity of a particular application domain.

The identification of web attacks is a critical component of our architecture, and we use an anomaly-based intrusion detection system that has been previously presented in [8] to perform this task. Anomaly detection systems can detect novel attacks, but they are also prone to make mistakes and incorrectly classify legitimate requests as malicious. Thus, it is important to develop strategies that can deal with false positives.

In [16] an anomaly detection system is used to filter out normal events so that signature-based detection is applied to anomalous requests only. This technique reduces the false positives generated by a misuse detection system and, in a way, is complementary to our approach.

Our approach is closely related to the use of hardened systems presented in [15]. The system presented there is composed of an anomaly detection system that uses abstract payload execution [17] and payload sifting [1] techniques to identify web requests that might contain attacks that exploit memory violations (e.g., buffer overflows and heap overflows). The requests that are identified as anomalous are then marked appropriately and processed by a hardened, "shadow" version of the web server. To implement this approach the Apache web server was modified to allow for the detection of memory violation attacks and the roll-back of modifications performed by malicious requests.

The approach proposed in [15] is similar to ours because it attempts to ameliorate the problem of false positives in the detection of web-based attacks. However, the proposed solution is different from our approach in a number of ways. First of all, the implementation of the system is limited to dealing with buffer overflow/underflow attacks, while our system is able to mitigate also non-control-data attacks [6] that might eventually lead to the corruption of the database. Second, in [15] the attacks that can be rolled-back are limited to attacks against the web server itself, while our focus includes also all server-side components such as CGI programs, server-side scripts, and back-end databases. Third, our approach clearly differentiates between sensitive and non-sensitive information, while the approach presented in [15] does not make any distinction in the type of information managed by the protected web-based system. Fourth, the creation of a "shadow" server requires the modification of the source code of the web server, and, if extended to server-side applications, to the application code as well. Our approach is much less invasive and can be applied to servers that are closed-source. In addition, it requires minimal modification to server-side applications.

The two approaches have the potential of being integrated in one comprehensive solution, where the server that receives requests deemed anomalous by our anomaly detection system is modified to make it more resilient to certain types of attacks. This is certainly an interesting research direction and will be the focus of our future work.

6. CONCLUSIONS

This paper presents an approach to provide differentiated access to a web site based on the anomaly score associated with web requests. This design allows one to route anomalous requests to servers that have limited access to sensitive information. By doing this, it is possible to contain the damage in case of an attack, and, at the same time, it is possible to provide some level of service to benign (but anomalous) queries.

We implemented a prototype that composes an existing web-based anomaly detection system and a reverse HTTP proxy. The prototype is able to analyze in real-time the requests sent to a web site and determine the corresponding anomaly score. The score is then used to drive the reverse HTTP proxy.

To analyze the feasibility of our approach and the impact that false positives would have on the user's access to the system, we build a PHP analyzer and we developed a metric that characterizes the amount of execution paths involving critical database access. We then analyzed several existing web-based applications. The results show that our approach would be applicable and beneficial to web-based applications that manage and store sensitive information.

Acknowledgments

This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853.

7. REFERENCES

[1] P. Akritidis, K. Anagnostakis, and E. Markatos. Efficient Content-Based Detection of Zero-Day Worms. In *Proceedings of the International Conference on Communications (ICC)*, Seoul, Korea, May 2005.

[2] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, San Diego, CA, February 2000.

[3] M. Almgren and U. Lindqvist. Application-Integrated Data Collection for Security Monitoring. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, LNCS, pages 22–36, Davis, CA, October 2001. Springer.

[4] R. Andersson. punBB - fast and lightweight PHP-powered discussion board. `http://www.punbb.org/`, 2005.

[5] Breach Security. Breachgate. http://www.breach.com/, June 2005.

[6] S. Chen, J. Xu, and E. Sezer. Non-Control-Data Attacks Are Realistic Threats. In *Proceeding of the USENIX Security Symposium*, Baltimore, MD, August 2005.

[7] Common Vulnerabilities and Exposures. `http://www.cve.mitre.org/`, 2003.

[8] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '03)*, pages 251–261, Washington, DC, October 2003. ACM Press.

[9] myBloggie - PHP and mySQL Blog / Weblog script. `http://mybloggie.mywebland.com/`, 2005.

[10] MySQL - The world's most popular open-source database. `http://www.mysql.com/`, 2005.

[11] PHP: Hypertext Preprocessor. `http://www.php.net/`, 2005.

[12] phPay - webshop or catalog based on SQL and PHP. `http://phpay.sourceforge.net/`, 2005.

[13] K. Poulsen. Tower records settles charges over hack attacks. `http://www.securityfocus.com/news/8508`, April 2004.

[14] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA '99 Conference*, Seattle, WA, November 1999.

[15] K. A. S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceeding of the USENIX Security Symposium*, Baltimore, MD, August 2005.

[16] E. Tombini, H. Debar, L. Mé, and M. Ducassé. A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic. In *Proceedings of the Annual Computer Security Aapplications Cconference (ACSAC)*, Tucson, AZ, December 2004.

[17] T. Toth and C. Kruegel. Accurate Buffer Overflow Detection Via Abstract Payload Execution. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Zurich, Switzerland, October 2002.

[18] Victoria's Secret Reveals Too Much. http://www.cbsnews.com/, October 2003.

[19] G. Vigna, W. Robertson, V. Kher, and R. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.