I. J. Computer Network and Information Security, 2012, 11, 11-20
Published Online October 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.11.02

Data Sharing for Context-Sensitive Access Control Policy Evaluation and Enforcement

Hassan Rasheed
Taif University, Taif, Saudi Arabia
Email: hsrasheed@acm.org

Abstract — Context-awareness has long been an important building block in designing systems that vary their operating behavior based on an analysis of rapidly changing operating conditions. There is the need however to define context more formally so that context data-sharing can take place between systems and more complex interactions between connected systems can be developed. The area of computer security is examined in particular as an area where the representation and sharing of context data can lead to more effective policy enforcement. A framework is proposed for sharing data between assessment sensors and enforcement mechanisms in order to facilitate more accurate policy enforcement. A detailed performance analysis of the proposed system is offered along with conclusions on the feasibility of such systems.

Index Terms — Context Awareness, Systems Integration,Data Sharing, Adaptive Access Control

I. Introduction

Context-awareness has long been an important building block in designing systems that vary their operating behavior based on an analysis of rapidly changing operating conditions. The term has, however, become overly used to refer to a wide variety of approaches. In many ways, context-awareness as it will be considered is already achieved implicitly by many systems. There is the need however to define context more formally so that context data-sharing can take place between systems and more complex interactions between connected systems can be developed. The area of computer security is examined in particular as an area where the representation and sharing of context data can lead to more effective systems.

We consider, for instance, the security paradigm in which the security of a system is due to the enforcement of a predetermined policy of allowed and disallowed actions. Although the policy may be written statically it may nonetheless include values and properties whose exact value is resolved dynamically when the policy is being evaluated. For example, the XACML [5] schema abstracts this into policy decision-making and policy enforcement and makes provisions for the evaluation of access control policies based on contextual information in the form of custom attribute evaluation modules which can return the value of a system property dynamically. Ordinarily, such contextual property-evaluation modules are developed on an as-needed basis and any infrastructure needed to return a value for that property must be done at that time.

So moving one step forward, we propose a general solution to the problem of gathering contextual information from various sensors and then making it available in a structured way to enforcement mechanisms which would then, in turn, use that information to enforce policy more effectively. One key strategy for achieving the type of flexibility and situation-aware enforcement demanded by modern security systems is to design frameworks that allow data sharing between otherwise autonomous security mechanisms. This strategy provides flexibility in the performance of the individual security tasks due to the modularity of each enforcement function. It also ensures the extensibility of the framework because its components are loosely coupled. Such a general solution has multiple challenges, however, which must be addressed. In order to facilitate the discussion of context sharing as a process, we will abstract it into three phases: data acquisition, data analysis and application.

The acquisition process consists of all tasks necessary to discover and retrieve context data based on certain criteria for relevancy. This is distinct from the process of analysis that derives secondary information from the data that is acquired during the first stage. The application stage, therefore, makes use of primary and secondary context data to fulfill some security assurance task.

This paper will present a detailed discussion of the design and implementation required to achieve context acquisition. A brief summary of the results obtained from testing policy enforcement will also be given to demonstrate the soundness of the approach. A complete discussion of the analysis algorithms and data application methods used for policy enforcement as well as a complete discussion of the policy enforcement testing results are detailed elsewhere in a forthcoming paper. In sum, this paper will address the design of frameworks for data sharing leading to context-aware policy evaluation and enforcement but will not detail the analysis algorithms and procedures used in the system or the strategies employed for data application at the point of enforcement for the sake of brevity.

The paper will begin with a detailed background on the system integration challenges which were addressed followed next by a description of the design goals which were upheld. Next we will present a description of a system implementation addressing the design challenges and a detailed performance analysis of that system.

II. Background

A. Defining Context Data

Schilit and Theimer [12] first mentioned the term context-aware with the explanation that such systems, "[adapt] according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes in such things over time." Other definitions such as the one offered by Dey in [3] have taken a human user-centric view of context, defining it as: "any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application." Broader definitions also exist such as, "all the knowledge that constrains problem solving at a given step without intervening in it explicitly" [1]. There are a few problems with definitions such as those stated above when trying to apply them to the domain of computer security. They are either are overly broad - lumping many types of information together as context - or they are overly specific, restricting context to types of information related to the physical surroundings of the system in question.

Many applications in the security domain, however, typically consider patterns of behavior over time to be a primary type of context. For an application such as intrusion detection, the physical context of a system is less important than the history of related events that have occurred in the system. In addition we would like to develop a way of describing context and context-ownership that facilitates context-sharing among entities in a given domain. This would be difficult if we consider a model where only objects possess context. If we were to develop a model based on the additional abstraction of an event (along with objects), then we could begin to describe the context of an event as being other events that are related to it based on some application-dependent criteria for relatedness. Then context-sharing becomes as easy as providing information on related events to all objects in the domain where the present event is now occurring.

Moving forward, the following definition for context will be used, "context is the set of interrelated conditions and secondary events surrounding and connected to a primary event under consideration which define and distinguish the environment in which it has occurred."

B. Integration Techniques for Distributed Systems

There are three main characteristics that distinguish an integrated system is mentioned by Hasselbring in [6]: heterogeneity, autonomy and distribution. From the perspective of systems integration, all of these issues are risks which must be mitigated (i.e. they are things standing in the way of a fully integrated system). But the mitigation often does not change the fundamental characteristics of the constituent systems and so the same characteristics are usually present before and after integration - integration merely provides a bridge so that these factors can be overcome. Heterogeneity can manifest itself in two main areas: technical and conceptual. Technical heterogeneity can come from differences in things such as: hardware platforms, operating systems, database management systems and programming languages. Conceptual heterogeneity can be produced by differing programming and data models or differences in modeling real-world concepts. Autonomy usually occurs in the areas of design or communication and execution.

1) Architectures for Systems Integration

There are two main architectures for systems integration. The first integration architecture is termed a component coalition. The architecture integrates independent components by providing a custom solution that will link the interfaces of the two components. These coalitions maintain the independence of the individual components in the following ways: each component has its own interface and each component has independent control of its data and processing.

The second architecture for systems integration is the federation. The main concept underlying component federations is the creation of a platform which can support a myriad of components as long as they conform to a set of standards. The federation provides infrastructure for inter-component communication and data sharing. Therefore in contrast with component coalitions, component federations are more general-purpose and more flexible [6].

2) Data and Control Integration Mechanisms

Data integration mechanisms are of two types: those which achieve data persistence and those which provide common data semantics. Data persistence is either achieved by data conversion in which components maintain separate data stores and data is translated to a format consumable by other components or by a common data store which is a single source that accumulates data from all of the components. Data semantics, on the other hand, is either achieved by using a common schema or common data formats.

The main method for achieving control integration is message passing. This message passing solution is actually the product of a mechanism to enable communication and a protocol to define the communication pattern.

III. Related Work

The approach to integrated security used by Ryutov et al. [11] is based the notion of an advanced security policy that can specify allowed activities, detect abuse and respond to intrusions. Each of these tasks (access control, intrusion detection and intrusion response) is performed by a single, multi-phase policy evaluator. A global 'System Threat Level' is used to integrate information from outside intrusion detection systems.

Teo et al. [13] propose a system to manage network level system access that considers threat information. Each node and service in the system has an associated access threshold. This threshold is checked against the

threat level of a part requesting access to determine if access is granted. The threat level of a source is regulated (increased and decreased) when signatures are triggered that specify the type of action to match and the type of threat level adjustment that should be performed.

In [4] the authors use a framework to assess the risk associated with granting a given access request and a corresponding level of trust required by any subject seeking to execute the request. In parallel, the trust level of the actual requesting subject is calculated and compared with the established value for the request to form a decision for the request.

IV. General Approach to Context Sharing for Adaptive Policy Enforcement

The general approach employed involves a few key design decisions and a general architecture. The first key for the approach is termed *Dynamic Context Discovery*. Using the notion that the context of an event consists of other, related events we then establish a criteria for relatedness that is appropriate for each individual security mechanism and then frame the context of an event based on those two factors. This context acquisition strategy must allow security components to select and receive only that data that is relevant to the decision they are trying to make. Because security mechanisms deal with events, they should be able to select the other events that relate to the event under consideration without necessarily having to process and deal with every event that occurs in the domain. As noted in [8] this property is not so much a desired trait as a required one as the volume of events processed solely by intrusion detection systems can reach tens of thousands per day. This implies also that the strategy for context acquisition must be able to search for events based on characteristics of relevance. So the first required property of the context acquisition approach is that it must provide relevant data.

The second key of the approach is *Implementation Transparency*. Another goal of our approach with regards to acquisition of context data is to allow security mechanisms to acquire data from other security controls while remaining agnostic of their implementation details: that a security component can acquire context data merely by knowing the features of the data it would like to receive. In this case that will entail the features of the event that is being evaluated and the domains from which the data should be gathered. For example, an access control system could acquire assessment data rating the risk of a particular user without knowledge of whether the sources of the data are anomaly or misuse detection systems and whether they operate at the network or host level.

The next key of the general approach is *Provider and Consumer Decoupling*. Another necessary feature is that the provider and consumer should be decoupled in time and space. We would like to provide functionality where an event provider can register or publish event information and then consumers can access that data according to their own constraints around what constitutes relevant context data. This also implies that the accesses of the provider are to be asynchronous, while those of the consumers will be synchronous. Decoupling in space is also necessary to support distribution.

The last key of the general approach behind the proposed context sharing framework is *Allowing Policy Level Description of Relevant Context*. Before we can analyze context data, or even search for it, we must have a means to describe its features and characteristics. One primary way of achieving this is through policy-specifications that include the features of context data.

These design goals with serve as the criteria for comparing between the different integration methods needed to overcome the heterogeneity of the systems being examined. A diagram of the general architecture is shown in the following figure.

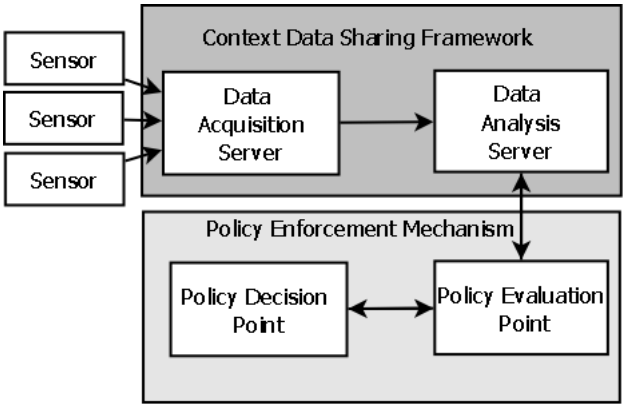

Figure 1: General architecture for context data sharing

Fig. 1 depicts the flow of context data within the general framework from sensors to the policy enforcement mechanism using the context sharing framework as a mediating service which provides context aggregation, analysis and then finally distributes the processed data to mechanisms which will apply the data in policy evaluation. The framework in this sense serves as a mediating service between loosely coupled data providers and consumers. The general approach has been applied to data sharing between intrusion detection sensors (as providers of context data) and access control systems (as consumers of such data) to demonstrate the effects of the approach and provide a platform for testing the performance of the framework. The proposed system which facilitates context data-sharing for adaptive policy enforcement has been termed the ABACUS framework. Subsequent sections will address the design and testing of this implementation.

V. The ABACUS Framework for Context-Aware Policy Evaluation

We summarize the approach being implemented in the following way: evidences of vulnerability exploitation (collected from intrusion detection sensors) are collected and analyzed into a higher level risk assessment for the sources and targets of access control requests. This risk assessment is subsequently used as an additional

parameter or contextual property in access control policies so that permit and deny decisions for an incoming request are based on an assessment of the risk posed by the requesting source and/or the risk posed to the targeted resource. This approach has been termed the Adaptive Assessment-Based Access Control System (ABACUS). The underlying methodology for this approach is that adaptive policy mechanisms must essentially rely on three interrelated processes: context data acquisition, analysis and application.

The remainder of this section will elaborate on how the architecture fulfills the design goals related to acquisition of context data which were enumerated previously. The reader interested in more detail on the implementation of the portions of the framework responsible for data analysis and application is referred to a previous paper [10].

C. Architecture

The primary components of the framework architecture are an alert server, which receives and processes assessment information, an analysis server, which responds to requests for analysis data, and the actual access control mechanisms which performs policy evaluation and enforcement. The access control system integrated with this architecture is the Apache webserver. The webserver is extended to perform the three intrusion responses discussed previously as the means to attack resistance: forcing additional authentication, restricting user permissions and restricting access to a target. Based on the resource and the actions available on that resource, a threshold is determined for the source and target associated risk above which, requests are denied. The intrusion detection system listens on the link for incoming requests and reports alerts for any requests that seem intrusive (in this case specifically, those requests that appear to be an attempt to exploit a known software vulnerability). The raw alerts from the IDS are passed through the alert processing server that performs any required filtering and also updates the risk assessments for the appropriate entities. Finally, the data from the new events is stored in an event database.

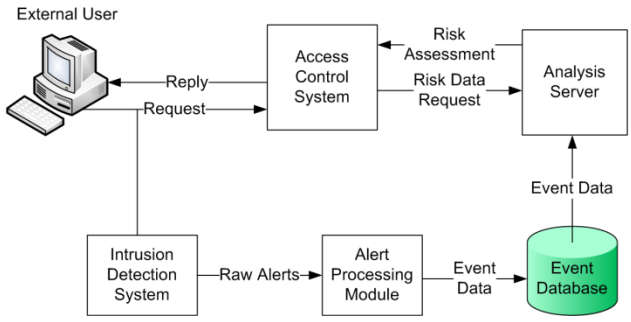

Figure 2: Proposed ABACUS System Architecture

The architecture is shown in Fig. 2.

D. Alert Processing Server

The alert processing module is responsible for extracting the information for each of the tables mentioned previously from the alerts it receives. In addition it can perform the functions of filtering out alerts that do not reference concrete vulnerabilities, or alerts for which the vulnerability does not match the current system configuration. Because of the nature of the analysis model, many of the most critical analysis functions are actually performed by the alert server. The present analysis model requires that the primary analysis function (updating risk values for entities) occurs as the events are processed (and consequently must be performed by the alert server and not by another entity).

E. Analysis Server

The analysis server receives client requests for assessment data, extracts the appropriate information from the event database and sends a response to the client (in this case the webserver).

F. Event Database

The event database is backed by a relational database implementation (in this case MySQL). Some of the structure of this database was derived from the IDMEF schema [2]. Some of the tables contained in the event database are the following:

• CVSS Vulnerabilities - this table stores information regarding current vulnerabilities from the National Vulnerability Database (NVD), which has adopted the CVSS scoring system. Each vulnerability is listed with its CVSS base score, exploit subscore, impact subscore, overall score and vector.

• Network Access Requests - Entries in this table are generated on the receipt of an IDS alert by the alert processing engine. The IP address and port of the source node are listed with the IP address and port of the target node. The time of the request, action being performed and target entity are also included in this table.

• Entity Tables - individual tables for the Nodes, Ports, Files and Users references in requests

• Intrusion Assessments - this table links individual requests to an intrusion assessment. Each assessment provides a classification for the event, its severity (which may be provided by the intrusion detection sensor) and whether or not the attack completed successfully.

• Vulnerability Descriptions - a vulnerability description provides information on a concrete software vulnerability. Each vulnerability description is provided by a vulnerability database (for the purposes of this study we only use CVE vulnerabilities because they have an objective scoring system). Each vulnerability description, therefore, only links to one element in the table of CVSS vulnerabilities and, consequently, only has one base score.

• Request Risk Cache - this table stores a calculated risk value for each request ID by querying for the CVSS score for all of the vulnerability descriptions that are linked to an intrusion assessment (and which provide a CVE ID). As mentioned in the section describing the model, the exponential average of all of the CVSS scores for the vulnerability descriptions used in a particular intrusion assessment are taken, and this value is stored in the request risk cache. When a particular risk handler queries the risk cache to produce a risk evaluation for a

particular entity, the risk estimate is multiplied by the decay factor to produce a dynamic risk estimate for that particular request.

G. Access Control System

The access control system integrated with the framework was the Apache webserver. In order to make as few modifications as possible to its existing access control policy evaluation mechanism, the ability to make and specify custom access control handlers for certain resources was utilized. Rather than returning a value for a specific attribute and querying against the event database within the access control handlers, the querying and analysis functions were abstracted into an external analysis server that provides risk analysis as a service. Requesting access control systems (such as the Apache webserver implementation) submit requests to the analysis server specifying the type of desired risk analysis (source, target or system) and the attributes of the entity which the analysis should center on (in the case of the source and target analyses). Based on the risk assessment returned and the risk threshold that is assigned to that particular resource or action a permit or deny decision is returned by the webserver.

VI. Preliminary Access Control Enforcement Results

As mentioned previously, the focus of the current study is examining the infrastructure necessary for context data acquisition and evaluating the performance of such a system using real-time data. However, we will provide here a brief overview of some preliminary results from using context data in access control policy enforcement for the purpose of completeness. This set of testing results is designed to demonstrate results of testing the ABACUS framework integrated with an Apache webserver as the access control policy evaluation and decision point. This testing will take place with real time incoming requests. Three techniques were selected to respond to probable intrusive behavior: forcing (additional) authentication, restricting subject permissions and restricting object permissions. In order to effectively illustrate the effect of these techniques, a scenario was generated with a webserver traffic simulator and requests were sent to two different webservers: one using the three analysis modules described previously, and another only using the notion of the global system threat to trigger response techniques. Whereas validation of the risk model could be performed with a captured data set being replayed over the network, the use of the response strategies will require active connections to the access control system and hence demands live traffic.

The traffic simulator creates an array of requesting nodes S where s_i is a member of S, each with an intrusiveness rating i_r, an inter-request period p and a total request life *l*. The webserver is arranged as an array of target resources T (where t_i is a member of T). Each t_i has a set of valid actions $\{a_1, a_2, \ldots a_n\}$ and invalid or intrusive actions $\{i_1, i_2, \ldots i_k\}$. Every p seconds (or some randomized derivative of p seconds) request source s_i selects a member of T and then based on its intrusiveness rating, selects either a normal or intrusive action to perform on the resource. Sources with a higher i_r, have a greater probability of selecting an intrusive action for each request. In practice, these intrusiveness or maliciousness ratings range from 0% to 90%.

The risk analysis model was fixed for the simulation of the scenario detailed below. Vulnerability weightings were the following: high severity ($w(H) = 3$), medium severity ($w(M) = 2$) and low severity ($w(L) = 1$). The risk multiplier (γ) was set to 10, to provide a more noticeable difference between various assessments.

In this intrusion scenario, a single intruder executes intrusive requests on several system resources - a method indicative of probing for which vulnerabilities have been patched or which configuration holes have been closed. The rest of the sources generating system requests are normal users - executing little or no requests that could be categorized as intrusive. The requests were generated over the course of a three hour simulation. The request trace for the intruder demonstrates that requests for different actions are denied based on his overall risk profile and eventually the intruder is locked out from all system requests. Meanwhile, requests from the other users are still permitted. A summary of the results for a simulation of this scenario are presented in Table .

Table I: Simulation Results for Scenario

Property	Server 1 (Source Risk)	Server 2 (System Risk)
Total Requests	2472	2472
Total Intrusive Requests	230	230
Intrusive Requests Denied	229	179
Percentage Denied	99.5%	77.8%
Total Normal Requests	2242	2242
Normal Requests Denied	16	1751
Percentage Denied	.7%	78.1%

In this scenario all of the intrusive requests were from the single intruder. Server 1 began to deny requests from the intruder after their source risk passed the threshold of 45. The normal requests blocked by server 1 were also from the intruder. Once the system risk for server 2 passes the threshold, it begins to deny requests from all sources.

VII. Performance Testing Results

A. Overview

This section is designed to provide some insight into the issues faced when designing systems that use real-time context data by examining various performance results. Two closely-related strategies for implementing the phases of acquisition and analysis were used and tested.

The first approach (referred to as version 1 of the framework) performed on-demand analysis: abstracting acquisition and analysis into different server mechanisms and performing the analysis by aggregating requests and deriving a risk assessment as new requests came in. Essentially, under the first approach, the acquisition server collects a set of intrusion detection alerts regarding various users and resources in the system but the data is merely stored in a database until there is a request from the data consumer (the access control system) to provide a risk analysis for a request source or request target. At that point the analysis server queries the database and returns a result. The set of results discussed as the first version of the framework are pertaining to this implementation.

The second approach (referred to as version 2 of the framework) differs in that the analysis function is triggered by the arrival of new security events from the sensors and consequently, the analysis does not take place as a function of an incoming request from the consumer. Risk assessments are continually maintained for all of the entities in the system (all resources and known request sources) so data for previous risk assessments are cached. In addition, as new assessment data becomes available, those risk assessments are updated for the entities in that event. The set of results mentioned as version 2 of the framework pertain to this implementation. After offering performance results of each version separately, a summary and relative comparison of the two approaches is offered.

B. *Performance Testing Methodology*

In order to compare the performance of the final version of the ABACUS framework against the earlier version and also against a normal Apache webserver, each server was stress-tested. This part of the testing relied on a regression testing and benchmarking utility called Siege [7]. The basic aim of this testing was to examine the behavior of each server subject to increasing load. The following parameters were used in the testing process:

• Number of clients - with the use of a wrapper for Siege called Bombard, the user is able to specify an initial number of clients an increment of how many clients the load should be increased by for each iteration and a total number of iterations (which also limits the maximum number of clients)

• A set of URLs - the same URLs from the scenario testing were used (both normal and intrusive). They were placed in a configuration file and read into memory by the utility when it starts. The clients then randomly request one of the URLs in the file for each request.

• Delay between requests - before each request, the client waits a random number of seconds between 0 and d, where d is the maximum delay between requests specified by the user

By testing in this way, we hope to draw conclusions on the following: the degree of improvement provided by the second iteration of the ABACUS framework over the first, the point at which each of the server types become overwhelmed given the hardware constraints as well as the specific reasons that account for the performance differences.

C. *Performance of Initial ABACUS Framework (v1)*

These results summarize the overall performance of the framework using the analysis model discussed in [9] which aggregated previous events on-demand or when a new request came in that required the information.

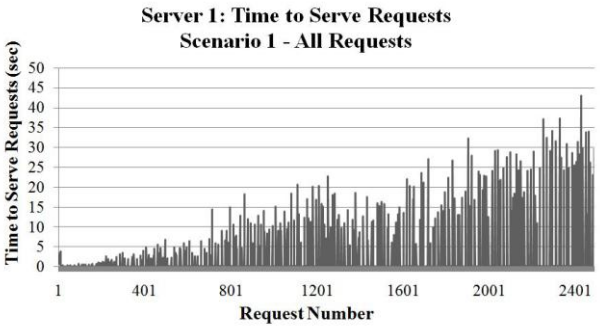

Figure 3: Time to Serve Requests - Scenario1 – All Requests

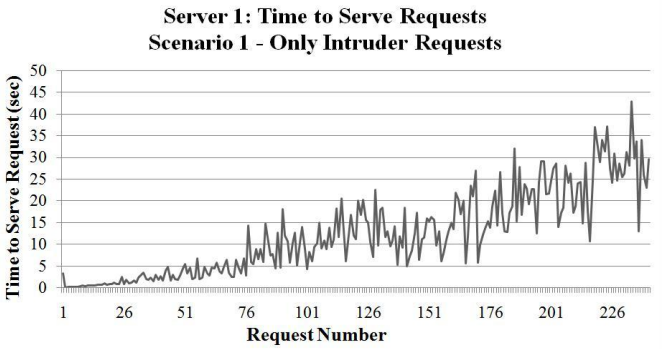

Figure 4: Time to Serve Requests - Scenario1 – Only Intrusive Requests

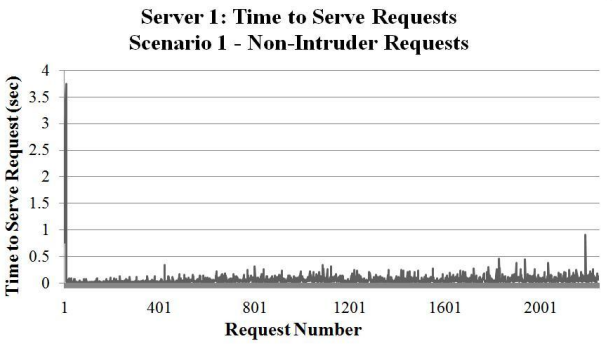

Figure 5: Time to Serve Requests - Scenario1 – Non-Intruder Requests

In Figure , Figure and Figure the time to serve requests on Server 1 during scenario 1 is shown as the number of requests increases. For the collection of this data, the simulator was set to generate 3 hours of traffic from 10 different nodes, only one of them executing intrusive requests (scenario one as described above). In Figure the time to serve is shown for all of the requests. In Figure , only the time to serve requests from the intruder is shown - this graph has the same linearly increasing pattern that is apparent when looking at the peaks of the graph in Figure . In Figure the time taken to

serve requests from the non-intruder nodes is graphed. The time to serve these requests remained relatively constant throughout the entire simulation, oscillating between zero and one seconds. The reason that the increase only occurred for the intruder node is that when the webserver requests risk data on that node, there is a constantly increasing amount of event data to analyze. For the other nodes, there is no such increase of data to analyze and, as a result, requests are served in the same amount of time for the duration of the simulation. This is undesirable, however, and could potentially create a scalability issue in scenarios where there are more nodes with intrusive behavior. In order to ameliorate these performance issues, a caching scheme was devised to facilitate faster generation of risk data.

D. Performance of ABACUS Framework with Recursive Analysis Model (v2)

In the second version of the framework, the risk analysis method was changed to a recursive one that would only require one calculation each time new data came in. This allowed risk calculations to be effectively cached and reused in subsequent requests which led to a significant performance increase. The performance analysis for the second version of the framework is shown in Fig. 6-9.

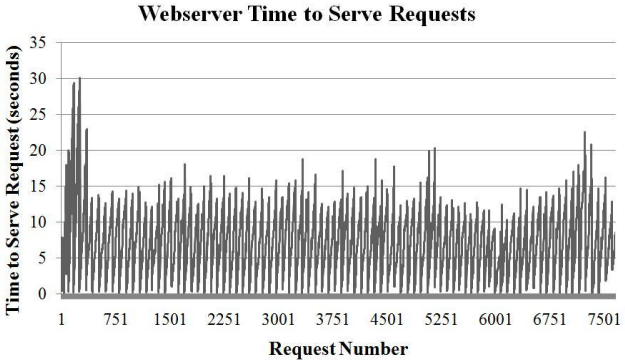

Figure 6: Statistics for ABACUS Framework Version 2 - Time to Serve Requests for Webserver

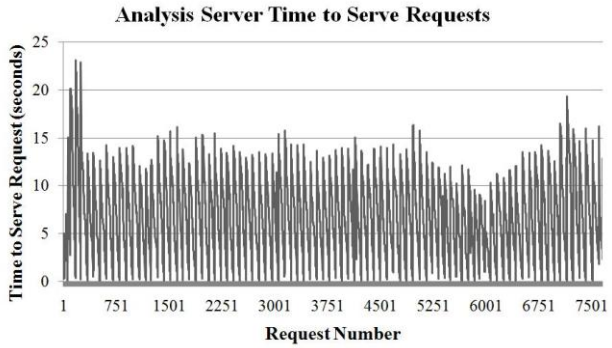

Figure 7: Statistics for ABACUS Framework Version 2 - Time to Serve Requests for Analysis Server

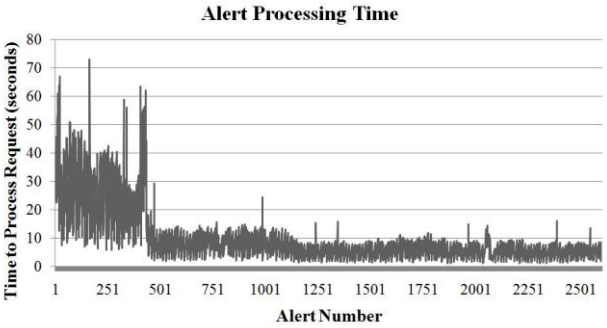

Figure 8: Statistics for ABACUS Framework Version 2 - Time for Alert Server to Process Alerts

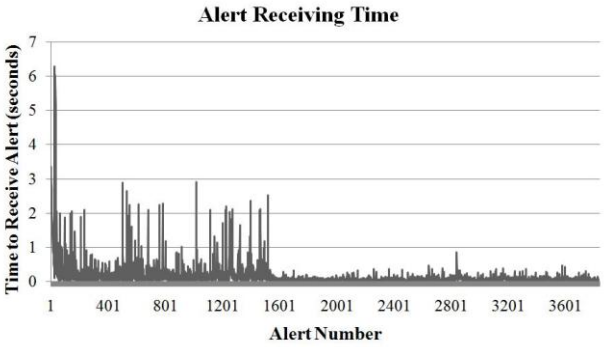

Figure 9: Statistics for ABACUS Framework Version 2 - Time for Alert Server to Receive Alerts

For the graphs in Figures 6-9, the following properties were used for traffic generation:

- 100 Active Clients
- 0-1 second randomized delay between requests
- 10 Minute Duration
- File size of 9.7KB
- Traffic Rate: ~45,000 Requests Per Hour

It is immediately noticeable from the graphs on version 2 of the framework that changing when the analysis takes place (on consumer demand versus upon being provided) eliminated the increasing request service time that was seen in the graphs for the first version of the framework. The performance was more stable in the face of an increased number of requesting nodes. The server was able to handle the load better and achieve higher throughput.

E. D. Performance Comparison Between ABACUS Framework v2 and Ordinary Apache Webserver

Fig. 10-12 summarize the server response, concurrency and transaction rate. time as seen from the client for three different server types: 1) a normal Apache webserver, with no integration of risk information, 2) an Apache webserver integrated with the first version of the analysis framework as discussed above (ABACUS Server v1) and 3) an Apache server integrated with the final version of the analysis framework (ABACUS Server v2). Fig. 13-15 present the same server comparisons, only with a larger randomized delay between subsequent requests. Server response time is the time between when a client initiates a request and when the server begins to respond with data. Concurrency is the number of clients the server

can handle simultaneously. Transaction rate is the number of requests which the server can successfully fulfill within a given amount of time. Both sets of graphs demonstrate that the server response, concurrency and transaction rate of the plain Apache webserver was higher than the ABACUS v2 server and that it, in turn performed better than the ABACUS v1 server.

For the first set of tests, the maximum number of clients which the ordinary Apache server could handle was approximately 310: at this point the response time spiked significantly and both the concurrency and transaction rate dropped off indicating that the server stopped responding. For the Abacus v2 server the maximum number of simultaneous clients was approximately 100 – the server indicated the same failure behavior, only the failure occurred much earlier. The point of failure for the Abacus v1 server was around 30 clients.

When the request delay was increased the estimated load of the normal Apache server increased its processing load to approximately 360 clients of the hardware being used for testing. The Abacus v2 server also increased its performance to approximately 220 clients and the Abacus v1 server only increased its performance marginally to approximately 35 clients.

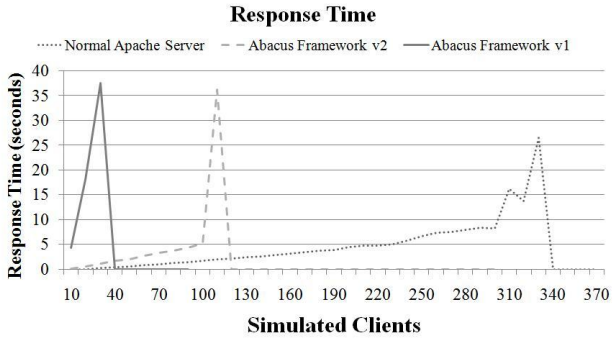

Figure 10: Server Comparison - 0 to 1 Second Delay Between Requests - Response Time

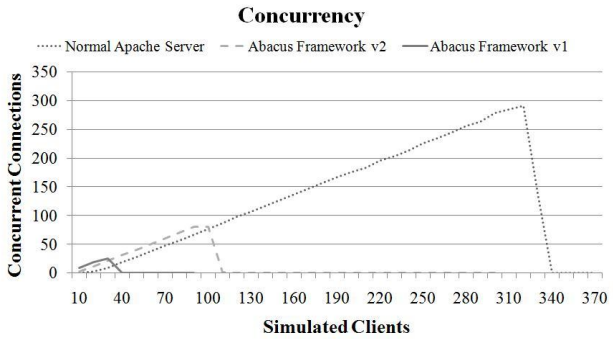

Figure 11: Server Comparison - 0 to 1 Second Delay Between Requests - Concurrency

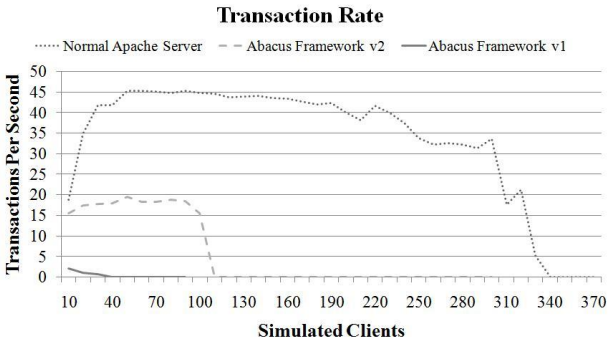

Figure 12: Server Comparison - 0 to 1 Second Delay Between Requests - Transaction Rate

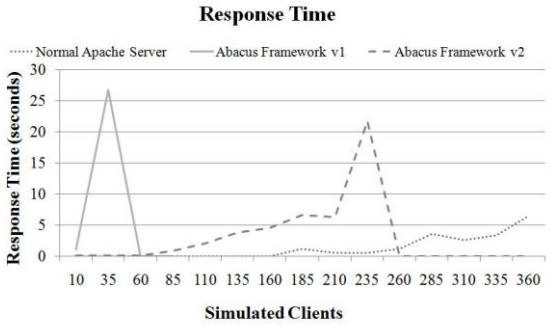

Figure 13: Server Comparison - 0 to 10 Second Delay Between Requests - Response Time

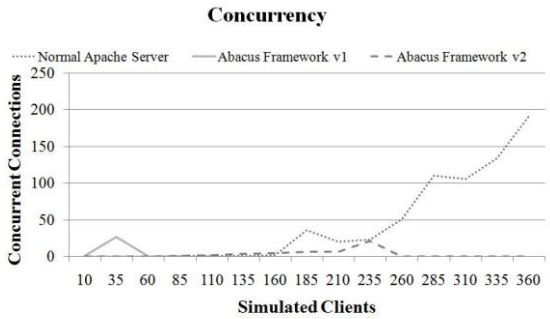

Figure 14: Server Comparison - 0 to 10 Second Delay Between Requests - Concurrency

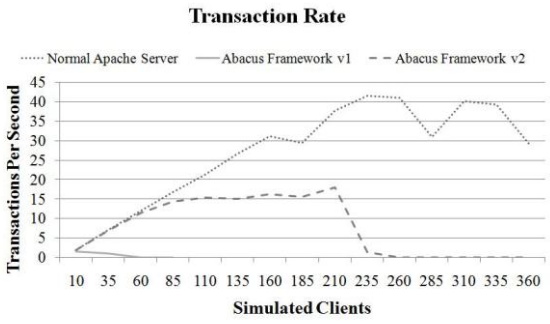

Figure 15: Server Comparison - 0 to 10 Second Delay Between Requests - Transaction Rate

F. Results Analysis and Discussion

It would be difficult to say that the first version of the framework (before changing the analysis approach) could realistically support any number of users for an extended period of time. As shown in Figure the time to serve

requests for the first version of the server was increasing even when the number of simulated clients was held constant. The performance determinant for the first version of the framework was the number of requests: increasing the number of simulated clients just caused the number of requests to increase more rapidly. The time to service each request was linear in the number of requests that the server had received up to that point. The caching approach allowed for a constant time to serve each request, but at the expense of data accuracy, where the algorithmic complexity was still the same.

The ABACUS Framework v2 was able to serve 100 simulated clients with a response time close to that of the Apache (5.16 seconds and 1.73 seconds), respectively. A critical factor, however, was that at this load the ABACUS Framework was maintaining the request frequency without a noticeable increase in processing time during the duration of the test, as demonstrated by Figure . Therefore, it is likely that the resources of the server machine were exhausted when the test moved to a higher number of simulated clients (110). The Apache webserver limits the number of forked client processes to 256 by default (this limit is compiled into the software). It appears based on the data that the server resources were exhausted by the increasing number of forked client processes being created by the Apache server. During testing, this led to incidences where the server machine locked up and required restarting. The Apache server access logs during these tests demonstrated that some of the requests for analysis data from the webserver access control modules were being denied due to increasing load; at the same time, however, the Apache web server was still accepting and queuing new client connections. In summary, the testing failure of the ABACUS Framework was due to the difficulty in controlling server resources: in particular of effectively limiting the incoming client connections in the face of increased concurrency and therefore increased response time per request. In order to remedy this, a rate limiting mechanism was built into the ABACUS Framework v2 whereby once a certain number of requests are queued, the server begins to deny incoming requests for risk data until more worker processes become available (to avoid forking too many processes to serve requests). A normal webserver performs the same function only it is able to determine the point at which to limit requests based solely on server resources whereas the mechanism built into the ABACUS framework did so based on the number of requests waiting for risk data. It was expected that there would be some performance penalty due to the additional processing required to achieve context-sensitive policy evaluation: it is an important finding to be able to quantify that penalty.

With that said, the peak transaction rate for the ABACUS Framework v2 was still 15.53 transactions per second at a response rate of 5.73 seconds. This roughly equates to 931.8 transactions a minute, 55,908 transactions an hour and 1,341,792 transactions per day.

Figure IV: Estimated Peak Performance of ABACUS Framework With Current Hardware

Transactions/Sec	Response Rate (sec)	Transact/Hr	Transact/Day
15.53	5.73	55,908	1,341,792

Based on this data, we can conclude that the proposed approach could be implemented in a large, high traffic website - particularly with dedicated server hardware providing increased performance.

The data also demonstrates that failure of a similar nature occurs for the Apache web server in isolation. Because there was a slower growth in response time per request, the Apache server in isolation was able to handle a greater number of client connections before failure, but when the failure happened, it manifested with much the same behavior as was displayed when testing the final version of the ABACUS Framework.

VIII. Conclusion and Future Work

Extensive testing and multiple iterations of the ABACUS framework led to the conclusion that although the process of context-aware or adaptive decision making may be abstracted into three processes (data acquisition, analysis and application), the instantiation of those processes into actual software modules is highly application dependent. The best performance was achieved with an implementation that virtually joined the acquisition and analysis phases, such that all of the analysis tasks were performed as new data was acquired. The initial strategy of generating the analysis data when it was requested by the client proved to be prohibitively slow given the amount of data being generated in the system and the frequency of requests.

Another key challenge was how to design an attack response that was tempered and still effective. We chose to use a strategy of restricting access permissions as the response to likely intrusive behavior by attaching risk thresholds to permissions on the controlled resources. A risk assessment was synthesized from the provided data on vulnerability exploitation attempts in order to provide a quantifiable measurement of the changing state of system entities in relation to their prospect of being attacked. Because the risk assessments were calculated for individual system entities, the assessment data also allowed for more granular responses.

The actual results of the attack simulations showed a marked improvement for the ratio of intrusive requests that were denied using the risk assessments. In the scenario that simulated an attacker performing vulnerability probing against the webserver, 99% of the intrusive requests were denied, while only .7% of the normal requests were denied. In the case of multiple intruders for one target attack, the framework denied 93.5% of the intrusive requests while only denying 9.2% of the non-intrusive requests. Even in the scenario of multiple intruders on multiple resources, where authentication was employed as a response, more intrusive requests were authenticated than non-intrusive ones (93.5% to 87.9%,

respectively), leading to a more efficient use of resources over the approach of authenticating all requests in situations of elevated risk.

This approach also proved feasible from a performance perspective. The testing results showed that the framework, given less than optimal server resources, was able to receive and process requests at a rate equivalent to over 1.3 million per day, exceeding the processing requirements for many high traffic domains and web sites.

REFERENCES

[1] Patrick Brezillon, Ghita Kouadri Mostefaoui, and Jacques Pasquier-Rocha, "Context-aware computing: A guide for the pervasive computing community," Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on, 2004.

[2] Herve Debar, David A. Curry, and Benjamin S. Feinstein, "The intrusion detection message exchange format (IDMEF)," 2007. Request For Comments (Experimental).

[3] Anind K Dey, "Understanding and using context," Personal Ubiquitous Comput., 5:4–7, 2001.

[4] Nathan Dimmock, András Belokosztolszki, David Eyers, Jean Bacon, and Ken Moody, "Using trust and risk in role-based access control policies," SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies, pages 156–162, 2004.

[5] Simon Godik and Tim Moses eds., "Extensible access control markup language (XACML) version 2.0," OASIS Standard, February 2005.

[6] Wilhelm Hasselbring, "Information System Integration," Communications of the ACM, 43:32–38, 2000.

[7] Joe Dog Software, "Siege," http://www.joedog.org/index/siege-home, Accessed November 2008.

[8] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz, "A data mining analysis of rtid alarms," Computer Networks, 34(4):571–577, 10 2000.

[9] Hassan Rasheed and Randy Y. C. Chow, "Automated risk assessment for sources and targets of vulnerability exploitation," In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 01*, CSIE '09, pages 150–154, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Hassan Rasheed and Randy Y.C. Chow, "Adaptive risk-aware application-level access control" In *The 2009 Conference on Security and Management (SAM'09)*, pages 10–16, Las Vegas, NV, July 2009.

[11] Tanya Ryutov, Clifford Neuman, Dongho Kim, and Li Zhou, "Integrated access control and intrusion detection for web servers" Parallel and Distributed Systems, IEEE Transactions on, 14:841–850, 2003.

[12] Bill Schilit, Norman Adams, and Roy Want, "Context-aware computing applications," IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[13] Lawrence Teo, Gail-Joon Ahn, and Yuliang Zheng, "Dynamic and risk-aware network access management," SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies, pages 217–230, 2003.

Hassan Rasheed received his Ph.D. and M.S. degrees in Computer Engineering from the University of Florida. He is currently an Assistant Professor in the Deanship of Information Technology at Taif University. His research interests include network security, enterprise security, IT governance and distributed systems.

