Evaluation of $SAAM_{BLP}$

Kyle Zeeuwen, Konstantin Beznosov
{kylez,beznosov}@ece.ubc.ca

Laboratory for Education and Research in Secure Systems Engineering
lersse.ece.ubc.ca
University of British Columbia
Vancouver, Canada

Technical report LERSSE-TR-2006-01*

Last Modification Date: 2006/07/21

Revision: #11

*This and other LERSSE publications can be found at http://lersse-dl.ece.ubc.ca

Abstract

Request response access control systems that use Policy Decision Points have their reliability and latency bounded by network communication. We propose the use of a secondary decision point that combines previously computed authorizations with knowledge of the security model to infer the result of authorization requests. We demonstrate that this approximate recycling approach increases the reliability of a system to a greater extent than existing precise authorization recycling solutions.

A simulation is described that compares system reliability while using both precise recycling and approximate recycling in a system that uses the Bell LaPadula model. Results show that an approximate recycling component is a much as 28% more likely to produce a valid response than a precise recycling component. It is also shown that increasing the number of subjects and objects managed by a system increases the hit rate improvement offered by approximate recycling, that the ratio between subjects and objects in the system affects the behavior of an approximate recycling component, and that the use of narrower Bell LaPadula security lattices result in greater hit rate gains than wider lattices under the same circumstances.

Contents

1 Introduction **4**

2 Methodology **5**

2.1 Analytical Model of the System 5

2.1.1 Bell LaPadula PDP 7

2.1.2 SDP 7

2.1.3 Reliability Calculations 7

2.1.4 Latency Calculations 9

2.1.5 Simplifying Assumptions 9

2.2 Design of Simulation Framework 10

2.2.1 Simulation Testbed 11

2.2.2 Data Input Files 11

2.2.3 Input Parameters 12

2.2.4 PDP 12

2.2.5 Precise Recycling Component 12

2.2.6 Approximate Recycling Component 12

3 Evaluation **13**

3.1 Impact of SAAM SDP on System Reliability 13

3.2 Impact of Subject and Object Population on SAAM SDP 14

3.3 Impact of BLP Lattice Shape on SAAM SDP 16

4 Future Work **18**

5 Summary and Conclusions **19**

A Input Generation Algorithms **19**

1 Introduction

Modern access control architectures are implemented using the request-response paradigm. Requests for access to resources are granted or denied based on the security policy of the system. The policy decision point (PDP) is separated from the policy enforcement point (PEP) in these systems. This design is illustrated in Figure 1. Examples of this type of system include Access Manager [8], GetAccess [6], SiteMinder [10], and ClearTrust [11]. The separation of decision and enforcement points allows multiple PEP's to use a single PDP and reduces the administrative overhead placed on IT personnel. As a consequence of this design, an upper bound is placed on the reliability of the system: if the PDP fails or the network becomes partitioned then the system will invariably fail. The separation of PDP from PEP also increases the latency observed by the users of the system [3].

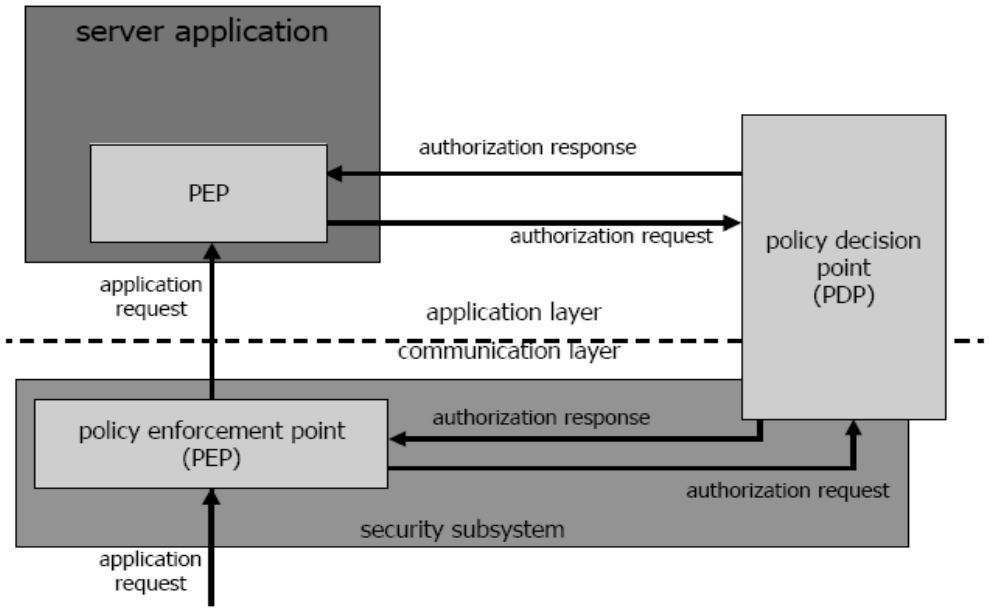

Figure 1: Request-response paradigm in authorization systems

One improvement to the design shown in Figure 1 is to reuse previously computed (i.e., secondary) authorizations. Existing approaches recycle an authorization only if it exactly matches the pending authorization request [8, 10]. This approach is referred to as precise authorization recycling. Precise recycling schemes introduce a cache-like component which serves as a secondary decision point (SDP). Although we refer to the precise recycling component as a secondary *decision* point, the component does not make decisions, it only reuses old ones. In this improved architecture both the PDP and SDP have to fail to produce a response in order for the system to be unable to service requests.

The Secondary and Approximate Authorizations Model (SAAM) [5], provides an improvement over the precise SDP. The SAAM SDP combines knowledge of the security

model with previously observed authorizations to produce approximate authorizations. In some systems a SAAM SDP will produce requests more frequently than a precise SDP. This increased percentage of resolved requests results in improved system reliability.

In this paper we present a simulation which was used to study the behavior of a SAAM SDP component in a system using the Bell LaPadula (BLP) mandatory access control policy. More specifically, the percentage of responses produced by a SAAM SDP and a precise SDP component were observed under different conditions.

The purpose of this study was twofold: 1) to show that the model presented in [5] produces more reliable systems than those employing precise recycling, and 2) to better understand which factors influence reliability in a system using a SAAM SDP component and the BLP mandatory access control policy [1, 2].

Results show that a SAAM SDP was as much as 28% more likely to produce a response than a precise SDP. Increasing the number of subjects and objects managed by a system also increased the percentage of responses produced by the SAAM SDP. Maintaining a roughly equal ratio between the number of subjects and the number of objects in a system maximized the reliability gain realized by a SAAM SDP. It was also shown that narrower BLP security lattices resulted in a greater SAAM SDP response rate than wider lattices under the same circumstances.

The rest of the paper is organized as follows. Section 2 introduces the components of the simulation and provides formulas used to calculate latency and reliability. Section 3 presents and discusses the results that were gathered, Section 4 presents the future direction of the study, and Section 5 concludes.

2 Methodology

Our first approach to this evaluation was to develop a working simulation of a request-response access control system. The model for this simulation is presented in Section 2.1. Formulas to calculate reliability (Section 2.1.3) and latency (Section 2.1.4) are presented next. Several simplifying assumptions are presented in Section 2.1.5, and a revised model is presented in Section 2.2.

2.1 Analytical Model of the System

The analytical model developed for this simulation involved interaction between five components, as shown in Figure 2. The client component represented one or more machines running client software that issued requests to the application. The Policy Enforcement Point (PEP) intercepted these requests and generated authorization requests. These authorization requests were simultaneously sent to the Policy Decision Point (PDP) and the Secondary Decision Point (SDP). The PEP used the first authorization response it received to either allow or deny access to the requested application resource. The PEP also updated the SDP with each request-response pair received from the PDP.

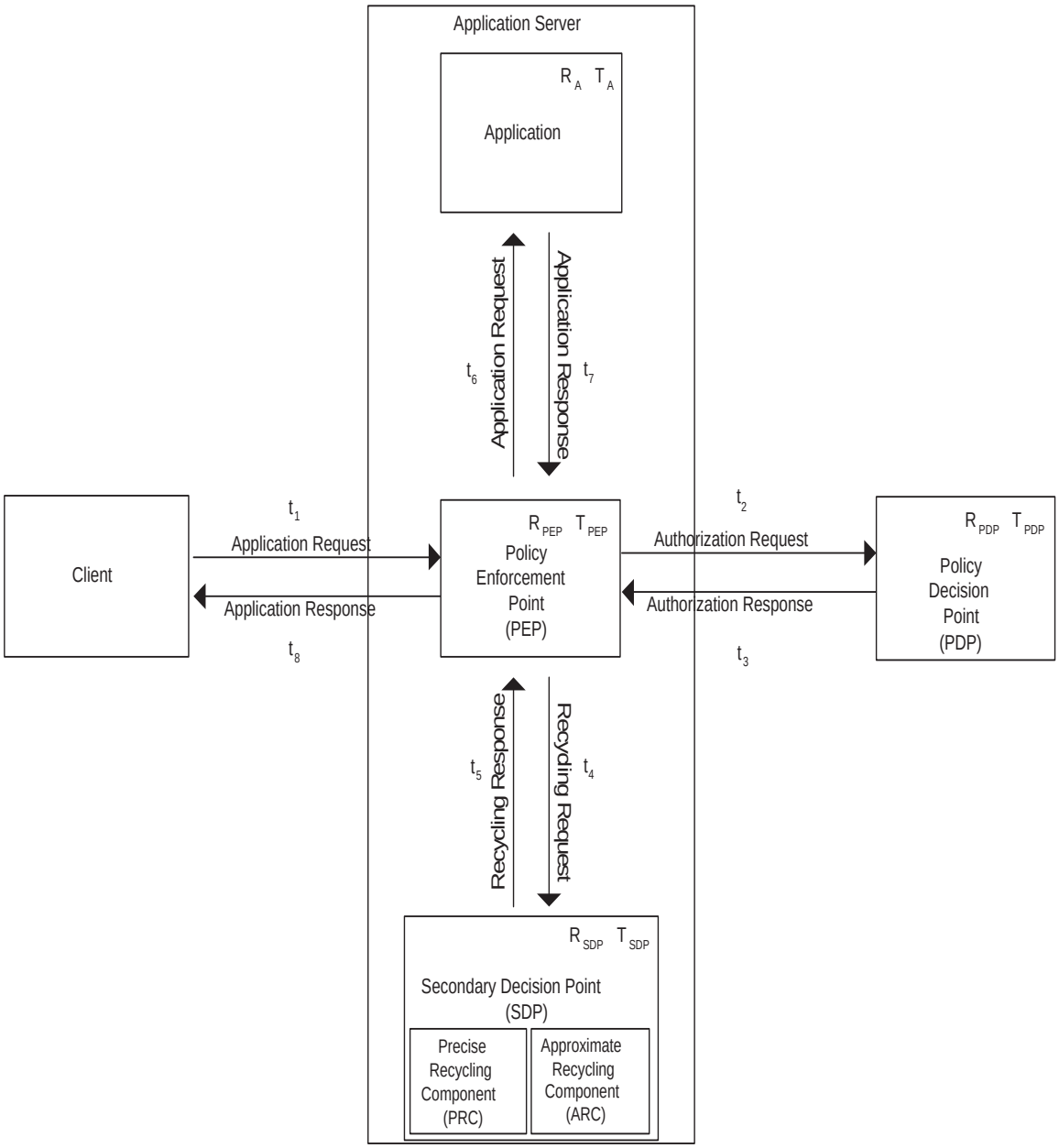

Figure 2: Request-response access control system with a secondary decision point

2.1.1 Bell LaPadula PDP

The PDP contained all the BLP specific security information for the subjects and objects of the system. Using the BLP access control policy, each subject and object was assigned a security level and zero or more categories. The security level (e.g., Unclassified, Confidential, Secret, Top Secret) represents the sensitivity of information when applied to an object or the clearance of a user when applied to a subject. Categories are used to classify objects and subjects into separate groups (e.g., Engineering, Administration). The combination of security level and categories assigned to an entity is referred to as the *security label* of that entity. To calculate the number of security labels in the system we used the following equation:

$$|security\_labels| = |security\_levels| * 2^{|categories|} \quad (1)$$

For this evaluation we assumed the BLP property of tranquility. In a tranquil BLP system, the security labels of subjects and objects do not change once they are assigned. When the PDP received a request, it determined if access should be allowed or denied based on the BLP model [1, 2]. The PDP communicated this decision to the PEP via an authorization response. In this evaluation only two access rights were considered: *read* and *append*.

2.1.2 SDP

The SDP received authorization requests from the PEP and searched for suitable responses. If a valid authorization response was found then the SDP returned the response to the PEP. Otherwise, the SDP indicated that no response was available. Figure 2 shows the SDP containing two subcomponents: the Precise Recycling Component (PRD), and the Approximate Recycling Component (ARD). The PRD was a hash table of key value pairs where previously observed authorization requests were used as keys and the corresponding authorization requests were used as values. The ARD was a more complicated component that used knowledge of the BLP model and a history of previously computed request-response pairs to determine the correct authorization response given an authorization request. Details of the ARP algorithms are presented in [5].

If a precise SDP was being simulated then only the PRD would be present in the SDP; if a SAAM SDP was being simulated then both the PRD and the ARD would be present. In the second case the authorization request would be simultaneously sent to the PRD and the ARD; the first response generated would be returned to the PEP.

2.1.3 Reliability Calculations

The reliability of the system was calculated from the reliability of the components of the system. A Reliability Block Diagram (RBD) for the system is shown in Figure 3. This diagram allowed us to visualize the failure paths of the system. The PEP represented a unique component with respect to reliability calculations because each request was handled

by the PEP several times during processing. As a result it was included three times in the RBD.

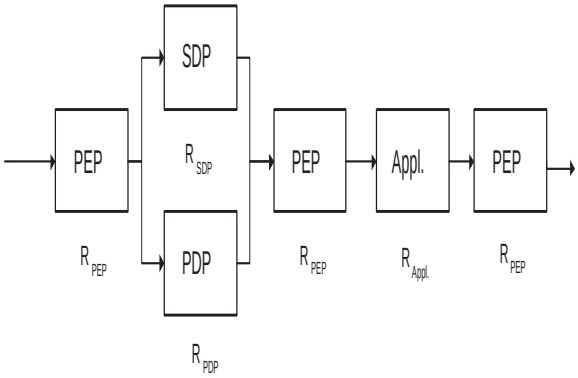

Figure 3: Reliability block diagram for simulation

In typical software systems, reliability is the probability that a system will function as intended at a given time. In our simulation we treated the SDP reliability in a unique manner. If the SDP component returns a response when queried then it functioned "as intended". We considered a lack of response from the component a failure, even though we did not expect the component to return responses in all cases. This allowed us to express the system reliability as a function of the SDP response rate, which we refer to as the SDP hit-rate, or simply the hit-rate.

The treatment of the SDP's hit-rate as its reliability introduced another problem: the hit-rate does not vary directly with time. Instead, the hit-rate was a function of how many previous request-response pairs have been observed by the SDP. We refer to this quantity as the SDP warmness. The SDP warmness was defined as:

$$sdp\_warmness = \frac{|observed\_pdp\_responses|}{|subjects \times objects \times access\_rights|} \tag{2}$$

For our evaluation, system reliability was treated as a function of time *and* SDP warmness. By applying the the combinatorial rules of reliability and modeling [9, pp.89] to the RBD shown in 3, we arrived at the following equation for system reliability:

$$\begin{aligned} R_{SYS}(t,x) = &\quad R_0(t) * (R_{PDP}(t) + R_{SDP}(x) - R_{PDP}(t)R_{SDP}(x)) \\ R_0(t) = &\quad R^3_{PEP}(t) * R_{Appl}(t) \end{aligned} \tag{3}$$

where t was time, x was SDP warmness, and R_A was the reliability of component A. This evaluation focused on determining the improvement in system reliability achieved by using a SAAM SDP versus a precise SDP. This was done by measuring the SDP hit-rate as the SDP warmness was increased; the progression of time was not simulated. Time was removed from equation 3 by assuming that each system reliability measurement was taken

at the same snapshot in time. As long as all reliability calculations are assumed to apply to that same snapshot in time then all the time dependant variables reduce to constants for that given instant in time. Using this assumption we reduced equation 3 to the following form:

$$R_{SYS}(x) = R_0 * (R_{PDP} + R_{SDP}(x) - R_{PDP}R_{SDP}(x)) \tag{4}$$
$$R_0 = R_{PEP}^3 * R_{Appl}$$

2.1.4 Latency Calculations

Latency was measured from the time the client issued a request to the time it received the response. Latency was computed as a sum of the communication delays and the processing times for each of the components involved in processing the request.

The PEP uses the first authorization response it receives; therefore, a minimum function was required to determine which communication delays to use for each latency calculation. Equation 5 shows the latency calculations for a single request. The terms are taken from Figure 2. The terms grouped to form L_0 were treated as a single constant and not simulated. They were varied during the data interpretation phase to determine their effect.

$$L = L_0 + min((t_2 + T_{PDP} + t_3), (t_4 + T_{SDP} + t_5)) \tag{5}$$
$$L_0 = t_1 + t_8 + t_6 + t_7 + T_{PEP} + T_A$$

Equation 5 was used to derive an equation for average latency based on the SDP hit-rate. This is shown in Equation 6. This was done based on the observation that the latency is comprised of the PDP round trip time ($t_2 + T_{PDP} + t_3$) when the SDP does not produce a response. In the case where the SDP does produce a response, the minimum function from Equation 5 was invoked.

$$\overline{L(x)} = L_0 + R_{SDP}(x) * min((t_2 + \overline{T_{PDP}} + t_3), (t_4 + \overline{T_{SDP}(x)} + t_5)) + (1 - R_{SDP}(x)) * (t_2 + \overline{T_{PDP}} + t_3) \tag{6}$$

2.1.5 Simplifying Assumptions

Several components of the analytical model were safely ignored during our study. Simplifying the simulation served several purposes. First, the implementation was easier. Second, the correctness of the design was easier to verify. Lastly, the correlation between input and results became more pronounced as the number of variables were decreased.

The behavior of the application component was not important because the simulation was only concerned with measuring the behavior of the SDP component. The only

attributes of the application considered were its impact on the latency and reliability observed by the client.

It was not necessary to treat the client component as a unique entity. A list of requests was used in place of the client. It was still necessary to account for the network latency between the client and the PEP.

Of all the timing parameters shown in Figure 2, only l_2, l_3, T_{PDP}, l_4, l_5, and T_{SDP} affect the behavior of the simulation. The remaining values were combined into a constant that affected the observed latency at the client.

The reliability of the application and the PEP were combined into a constant that only affected the observed reliability of the client. To make this simplification we had to ignore the effects of PEP failures on the behavior of the PDP and SDP.

2.2 Design of Simulation Framework

The equations presented in 2.1 provided ways to calculate the reliability and latency observed at the client. To use these equations we needed our simulation framework to produce the following values: $\overline{T_{PDP}}, \overline{T_{SDP}(x)}$, and $R_{SDP}(x)$. The two timing averages, $\overline{T_{PDP}}, \overline{T_{SDP}(x)}$, were not captured in this simulation. This was left as future work. The framework used to generate $R_{SDP}(x)$ is shown in Figure 4. Each component of the simulation framework is discussed in the following subsections.

The analytical model shown in Figure 2 was altered in several ways to form the simulation framework. The behavior of the application and the PEP were not simulated because it was only necessary to simulate the behavior of the access control policy decision points (i.e., the PDP and SDP). The client component was replaced by sets of access requests. These requests were dispatched by the simulation testbed to the PDP and the SDP components. The role of the SDP in Figure 2 was to forward requests to its two subcomponents and combine their responses. This behavior was moved to the simulation testbed and the SDP was replaced by the PRC and ARC.

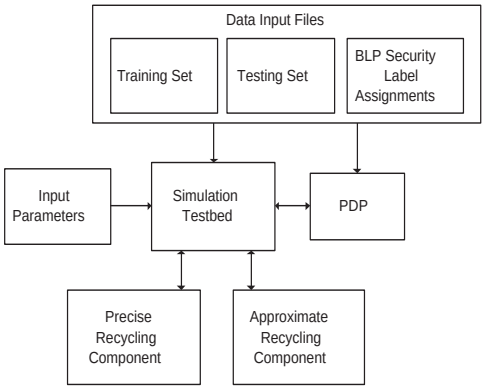

Figure 4: Simulation framework

2.2.1 Simulation Testbed

The simulation testbed was responsible for instantiating all simulation components, starting the simulation and recording the results. The simulation testbed operated in two different modes. The first mode was SDP warming. In this mode the engine submitted requests from the *training set* to the PDP. The testbed used the responses from the PDP to update the PRC and ARC. The testbed continued to update the PRC and ARC using responses from the PDP until a desired SDP warmness was achieved. At this point the testbed switched to data gathering mode. The PRC and ARC were not updated in this mode. The testbed submitted requests from the *testing set* to the PRC and ARC and recorded their responses. The responses gathered in this mode were used to calculate R_{SDP}. This process was repeated several times until the SDP had been completely warmed. The end result of the simulation was a set of R_{SDP} values for each SDP warmness level that was tested. If extended, the simulation testbed will also calculate $\overline{T_{SDP}}$ and $\overline{T_{PDP}}$,.

2.2.2 Data Input Files

The *training set* and *testing set* were simply lists of requests. Each request was made up of a subject, object, and access right. The *training set* was a randomized list of every possible request for a set of subjects, objects, and access rights. This means that its contents were deterministic and its order was random. The *testing set* was a random sampling of requests. It usually contained 10000 requests, unless the total number of possible requests was less than 10000.

The total size of the *testing set* was initially an integer multiple of the size of the training set. Iterating through the *testing set* for each SDP warmness measure being studied was accounting for almost 90% of the total time required to run a simulation. To reduce the total simulation time, we limited the number of tests in the test set. Based on the observed hit-rate from the requests in the test set, we generated a confidence interval to represent the hit-rate for the entire set of possible authorization requests. The confidence interval was calculated using an application of t-distributions. Calculations are shown below:

$$p = \hat{p} \pm T\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \tag{7}$$

where p was the hit-rate for the entire set of possible requests, $\hat{p}$ was the hit rate for the requests in the test set, n was the number of requests in the test set and T was the t-score for the desired confidence interval. For a confidence interval of 95% and n=10000, T=1.96. This yielded a maximum margin of error of $\pm$ 0.98% when $\hat{p} = 0.5$.

The *BLP security label assignments* file contained the security label for each subject and object in the system. This was required to use the BLP mandatory access control model. The file was generated randomly given the number of subjects, objects, classifications, and categories in the system.

Each of the three files discussed above were generated by tools that are included in the simulation package. Further details are provided in [12]. The algorithms for generating

these files are included in Appendix A.

2.2.3 Input Parameters

A full list of input parameters and usage examples are in [12].The input parameters controlled the following aspects of the simulation:

- input files
- logging
- $SDP_{SAAM-BLP}$ behavior (i.e., search depth)
- which SDP's were simulated (i.e.,($SDP_{Precise-BLP}|SDP_{SAAM-BLP}$)
- plug-ins

2.2.4 PDP

The PDP implemented the BLP model as specified in [1, 2]. Given a request it always issued a grant or deny response. We considered two approaches to simulate PDP failures. The first was to design the PDP component to fail to return a decision a fixed percentage of the time. The second approach was to design the PDP to always return a response and introduce PDP failures by setting the R_{PDP} in equation 4. We chose the second method because we required our simulation to produce repeatable results for a given set of data input files. As a consequence of this design our results depend upon on the correctness of our analytical model for system reliability (equation 4).

2.2.5 Precise Recycling Component

The Precise Recycling Component (PRC) was implemented as a hash table. The requests were used as keys that map to the corresponding responses. When the simulation testbed queried the PRC with a request a lookup was performed using the request as the key. If a corresponding response was found then it was returned to the simulation testbed, otherwise the component indicated that no response was available.

2.2.6 Approximate Recycling Component

The Approximate Recycling Component (ARC) leveraged knowledge of the BLP model to infer responses based on previously observed request-response pairs. The algorithms used in the ARC are discussed in in [5]. The simulation testbed sent authorization requests to the ARC; if the ARC was able to produce a valid response it returned the response, otherwise the component indicated that no response was available.

3 Evaluation

Simulation results were gathered on a commodity PC using an Intel Pentium 4 2.8 GHz hyper-threaded processor with 1 GB of RAM. The OS on this machine was Windows XP Professional SP2. The simulation was written in Java and run on the Sun's 1.4.2_09 JRE. High resolution timing using the Sun.Misc.Perf Java class was disabled due to its poor performance. The simulation code did not perform well with more than 1000 subjects and 1000 objects. As a result, the experiments performed did not include tests with large populations (i.e., 10^4 subjects and 10^6 objects[7]) where a SAAM SDP component was expected to be particularly effective.

We conducted experiments to validate the following hypothesis:

1. The use of a SAAM SDP results in a more reliable system than the use of a precise SDP.
2. An increase in the ratio of subjects and objects to security labels will improve the hit rate of a SAAM SDP.
3. A SAAM SDP will exhibit a higher SDP hit rate in systems with roughly equal ratios between the number of subjects and objects.
4. A SAAM SDP will exhibit a higher SDP hit rate in systems with fewer BLP categories.

In the following sections an approach to validate each hypothesis is presented, followed by results and discussion for each experiment performed. The discussion included in these sections assumes knowledge of the details of the $SAAM_{BLP}$ algorithms presented in [5].

3.1 Impact of SAAM SDP on System Reliability

To evaluate the impact of a SAAM SDP on system reliability we ran two simulations with identical parameters. We used 1000 subjects, 1000 objects, three security levels and three categories. The testing set contained 10000 requests; using equation 7 the margin of error for the resulting hit rates was no more than 0.98%. One simulation used a precise SDP and the second used a SAAM SDP. We captured the SDP hit rate for 10 different levels of SDP warmness. This is shown in Figure 5.

By combining the SDP hit rate values at a SDP warmness of 10% from Figure 5 with the system reliability equation derived in 2.1.3 (equation 4), we calculated the system reliability for different R_{PDP} values. This is shown in Figure 6.

The results in Figure 5 show that the SAAM SDP outperforms the precise SDP in terms of hit rate. The highest performance gain between the SAAM SDP and the precise SDP was 28%, which occurs at an SDP warmness of 10%. Averaging the 11 measurements shows an average hit rate improvement of 13% when using a SAAM SDP compared to a precise SDP.

The graph also shows that the SAAM SDP generates most of its improvement at low SDP warmness values. From 0% to 10% SDP warmness the SAAM SDP exhibits a 38%

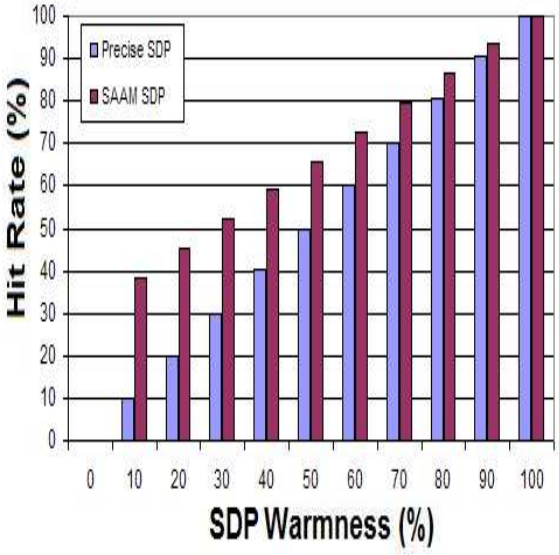

Figure 5: **Hit rate for precise and SAAM SDPs.** The hit rate for both components is shown for different SDP warmness values. These simulations were run with 1000 subjects, 1000 objects, 3 security levels, three categories, and 10000 requests in the testing set.

gain in hit rate. For every other 10% increment in SDP warmness the SAAM SDP only gains about 7% hit rate over the previous SAAM SDP hit rate measurement.

We chose to show the system reliability at a 10% SDP warmness because in many implementation environments the space limitation imposed on the SDP will limit the SDP warmness to low values. The results from Figure 6 show that even at a low SDP-warmness a SAAM SDP provides improvements over a precise SDP. As the R_{PDP} value increases the improvements provided by a SAAM SDP diminish. This is a natural consequence of Ahmdal's law; if the system reliability is already at least 99% due to R_{PDP} then the improvement offered by a SAAM SDP is at best 1%. We do not feel that the small improvements shown in Figure 6 for higher R_{PDP} values are negative results. In highly available systems, reliability is a top priority, thus even small improvements to system reliability represent substantial gains.

3.2 Impact of Subject and Object Population on SAAM SDP

To study the effects of subject and object population as well as the effects of the number of security labels and categories on the SAAM SDP hit rate we must first define the notion of density. Density represents the ratio between the number of entities (i.e., subjects and objects) in the system to the number of security labels. It is defined below:

$$density = (|subjects| + |objects|)/(|security\_labels|) \tag{8}$$

In the experiments to study the effects of the subject and object population size, we

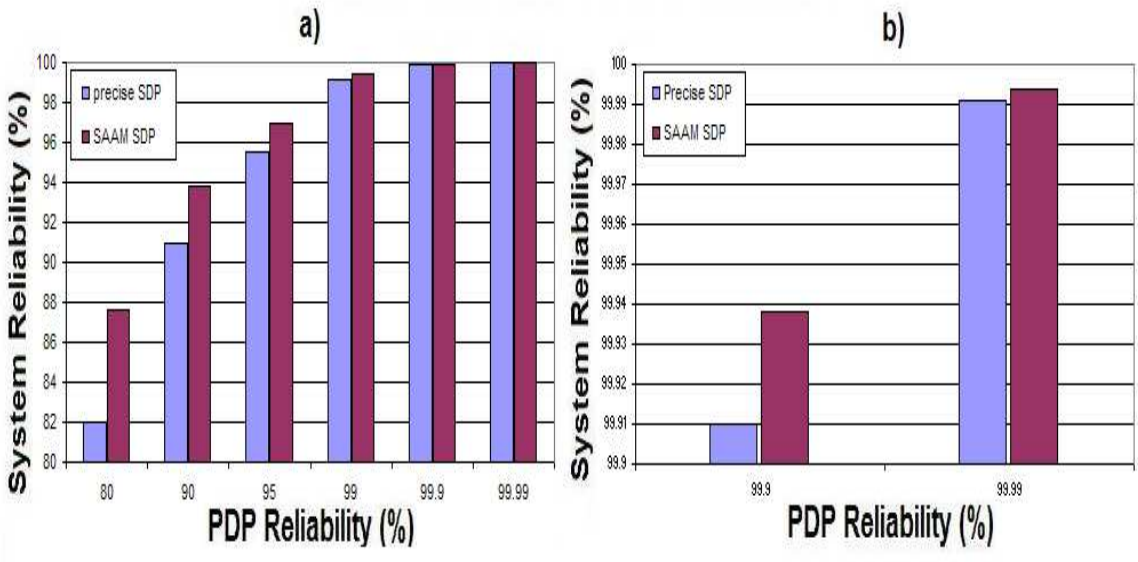

Figure 6: **System reliability for precise and SAAM SDPs.** The system reliability was calculated using R_{SDP} values from Figure 5 at an SDP warmness of 10% and equation 4. Graph b) shows the same information as a) but focuses on the improvements at high R_{SDP} values.

fixed the number of security labels and varied the number of subjects and objects. In our first experiment we fixed the density value and varied the ratio of subjects to objects. The number of security levels used was 3, the number of categories 3, and the number of entities (i.e., subjects and objects) was 1300. 10000 requests were used in the test set; therefore the margin of error for the resulting hit rates was no more than 0.98%. Figure 7 shows the SDP hit rate at a 10% SDP warmness for different ratios of subjects to objects.

Our second experiment measured the effect of increasing density by adding entities to the system. We simulated with an equal number of subjects and objects in each simulation to remove the effect of the subject to object ratio. We used three security levels and three categories. The testing set contained 10000 requests; therefore the margin of error for the resulting hit rates was no more than 0.98%. Figure 8 shows the hit rate for different density values.

Figure 7 shows that the SAAM SDP hit rate is highest when the ratio of subjects to objects is roughly equal. Also shown is that there is a range of ratios near 1 where the hit rate remains near its max before decreasing substantially. Future tests should determine if the region on the graph where hit rate is at its max, which stretches from a $|S|/|O|$ ratio of approximately 0.44 to 2.25 in our evaluation, will grow larger or smaller as the density is increased.

We can attempt to explain the data shown in Figure 7 in the following manner. Previously observed request-response pairs provide information to the SAAM SDP. For each data point in Figure 7 the number of total entities is fixed at 1300; the total number of

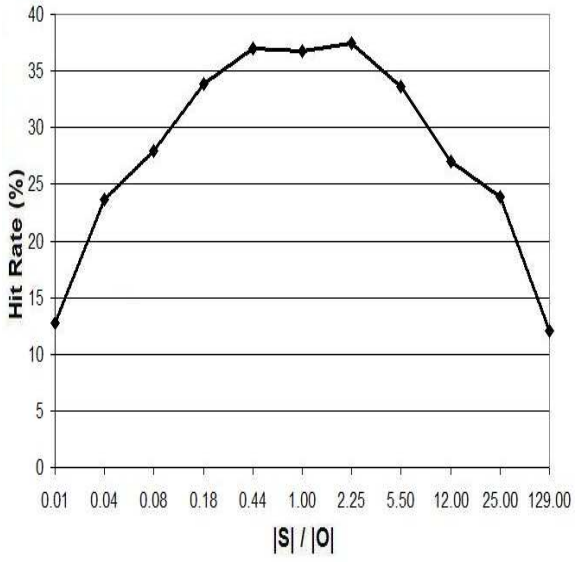

Figure 7: **Hit rate for SAAM SDP with different subject to object ratios.** These simulations were run with 3 security levels, three categories, and 10000 requests in the testing set. Hit rates at 10% SDP warmness are shown. The x-axis in this figure shows the ratio $|S|/|O|$. The leftmost value is for a simulation with 10 subjects and 1290 objects, the rightmost value is a simulation with 1290 subjects and 10 objects.

possible requests for each data point varies. At a subject to object ratio of 1 there are 845000 (i.e., $650 \times 650 \times 2$) possible requests, each one yielding information about the subjects and objects in the system. For a subject to object ratio of 0.01 or 129 (the two end-points) there are only 25800 requests; therefore there is less information available to the SAAM SDP.

Figure 8 shows that a correlation between hit rate and density exists; however this correlation does not appear to be linear. The hit rate increases substantially between density values of 1 to 40. From 40 to 100 there is no apparent gain. We can interpret the results in two different ways. The first interpretation is that the hit rate reaches a maximum at a density of approximately 40. The second interpretation is that the hit rate increases after reaching a density of 40, but not in a linear rate. Neither interpretations can be argued given the limits of the current simulation framework; we cannot investigate the hit rate for density values greater than 100.

3.3 Impact of BLP Lattice Shape on SAAM SDP

This experiment studied the effect of changing the "shape" of the BLP lattice. If we view a typical BLP security lattice we can see that the number of security labels affects the depth of the lattice whereas the number of categories affects the width of the lattice. We ran simulations with a fixed density and varied the number of security levels and categories

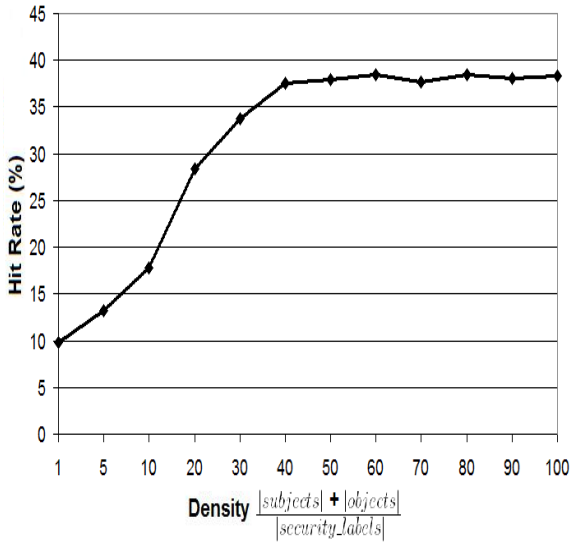

Figure 8: **Hit rate for SAAM SDP with varied density.** These simulations were run with 3 security levels, three categories, and 10000 requests in the testing set. Hit rates at 10% SDP warmness are shown. To achieve the density values in the figure the number of subjects and objects was varied from 13 to 1300; in eash simulation the $|S| = |O|$.

such that the total number of security labels remained the same. For all simulations 500 subjects and 500 objects were used. 10000 requests were used in the test set; therefore the margin of error for the resulting hit rates was no more than 0.98%. Figure 9 shows the results of the experiment.

Figure 9 shows that adding categories while keeping the density constant will reduce the SAAM SDP hit rate. To understand this relationship we must further explore the details of the BLP model [1, 2] and the BLP SDP [5]. Within the ARC of the SDP, a *dominance graph* is built to represent the dominance relations between the entities of the system. Subject A *dominates* Object B *iff* A has a higher security classification that B *and* the category set of B is a subset of the category set of A [4]. The SDP uses these dominance relations to produce approximate authorizations, therefore the more relationships observed and represented in this dominance graph, the greater the SDP's ability to produce approximate responses.

In a lattice with no categories every entity will either dominate or be dominated by every other entity. There will be no entities A and B such that $\lambda(A) \not> \lambda(B)$ *and* $\lambda(B) \not> \lambda(A)$. Once two or more categories are used it becomes possible for entities to have disjoint category sets. When entities have disjoint category sets there is no dominance relationship between the two entities. This means that they cannot be represented in the dominance graph and cannot be used to infer approximate responses. As the lattice becomes wider the rate of occurrence of disjoint entities increases and the SAAM SDP is unable to produce responses more often.

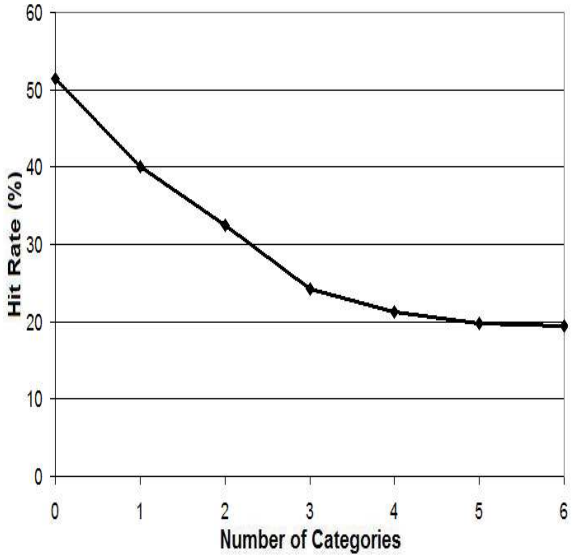

Figure 9: **Hit rate with fixed density and different number of BLP categories.** These simulations were run with 500 subject, 500 objects, and 10000 requests. Hit rates at 10% SDP warmness are shown. The number of security levels and categories was varied such that the total number of security labels was constant for each simulation. The left-most data point represents a BLP model with 64 security levels and 0 categories; the right-most represents 1 security level and 6 categories.

4 Future Work

There are several tracks of future work being pursued to continue this project. The current code base experiences performance problems when the subject and object sets exceed 1000 entities each. Simulations with larger sets take longer than a day to process and appear to make little or no progress. These simulations also consume greater than 1GB of memory, which limits the range of environments where simulations can be performed. There are several potential courses of action that could be taken to solve this problem. They are discussed in [12].

The $SAAM_{BLP}$ SDP currently does not use pre-computed deny responses; it can only infer responses based on collections of allow responses. There is ongoing work to leverage deny responses. A newer version of the SAAM SDP that includes deny response algorithms is currently under evaluation.

Timing was disabled due to the time overhead introduced by using the Sun.misc.performance class. This could be fixed by reducing the number of requests that are timed or by replacing Sun.misc.performance with another, more light-weight, timer. One such alternative is System.nanoTime(), which is available in j2se 1.5.0.

5 Summary and Conclusions

This paper has presented a simulation framework that allowed us to study the impact of a SAAM SDP in a request-response system using the BLP access control model. The studies performed show that applying SAAM recycling to a system that uses the BLP access control model can yield an improvement in system reliability when compared to precise recycling. The reliability improvements offered by a SAAM SDP are dependant on its hit rate; therefore, effort should be focused on maximizing this value. A future area of research is to determine in which environments a SAAM SDP can produce hit-rates high enough to significantly improve the reliability of a system.

This study has also illustrated several factors which affect the hit rate of a SAAM SDP. We showed that increasing the density of a system will increase the effectiveness of a SAAM SDP. We also observed that maintaining the ratio of subjects to objects in the system near one produces better SDP hit rates. Finally, we showed that BLP configurations using fewer categories produce higher SAAM SDP hit rates than configurations with more categories.

A Input Generation Algorithms

This appendix provides pseudo-code to demonstrate the algorithms used to produce the input data sets for the simulations performed in this evaluation. Each algorithm is presented and then discussed. The algorithm to generate the testing set (Figure 10) and the training set (Figure 11) are very similar; the only difference being the number of requests that are included in the output.

```
request[] generate_testing_file(number_testing_samples: int, subjects: set,
          objects: set, access_rights: set)
  request[] ordered_request_list =
          generate_ordered_request_list(subjects: set, objects: set, access_rights: set);
  int number_requests = minimum( number_testing_samples, ordered_request_list.size() );
  request[] chosen_requests = new request[number_requests];
  for ( i ← 0 to number_requests ) do
    request_index = get_random_index( ordered_request_list.size() );
    while ( ordered_request_list[request_index].dirty and more_requests_left ) do
      request_index = ( request_index + 1 ) % ordered_request_list.size();
    end while
    if( more_requests_left )
      chosen_requests[i] = ordered_request_list[request_index];
      ordered_request_list[request_index].dirty = true;
  end for
  return chosen_requests
```

Figure 10: testing set generation algorithm

```
request[] generate_training_file(subjects: set, objects: set, access_rights: set)
  request[] ordered_request_list =
            generate_ordered_request_list(subjects: set, objects: set, access_rights: set);
  int number_requests = ordered_request_list.size();
  request[] chosen_requests = new request[number_requests];
  for ( i ← 0 to number_requests ) do
    request_index = get_random_index( ordered_request_list.size() );
    while ( ordered_request_list[request_index].dirty and more_requests_left ) do
      request_index = ( request_index + 1 ) % ordered_request_list.size();
    end while
    if( more_requests_left )
      chosen_requests[i] = ordered_request_list[request_index];
      ordered_request_list[request_index].dirty = true;
  end for
  return chosen_requests
```

Figure 11: training set generation algorithm

Both algorithms start by generating an ordered list of all the possible requests. In the actual implementation a single list is used for both file generations and only the dirty list is reset. Once an ordered list of requests has been generated both algorithms proceed to generate a random index and use the index to choose a request. If that request has already been used then the index is incremented (or wrapped in the case of the end of the list), and a new request is tested. If that request has also been used then the index is incremented again and so on until an unused request is found or all entries are tested.

The sampling process is repeated until enough samples have been collected, typically 10000 for a testing set and all of the requests for the training set.

The BLP security level assignments are generated in a similar fashion. First a list containing string representations of every possible security label is generated. This includes a low label which is dominated by all other labels and a high label that dominates all other labels. Once this list is created we simply randomly assign a label to each subject and each object in the system. This algorithm is shown in 12.

```
string[] generate_blp_security_labels(num_levels: int, num_categories: int,
        num_subjects: int, num_objects: int)   string[] category_combinations = gener-
ate_power_set_of_categories(num_categories: int);
  string[] security_labels = new string[ 2 + num_levels * pow(2,num_categories) ];
  security_labels_index = 0;
  \\ add a high label in addition to the body of the lattice
  security_labels[security_labels_index++] =
        toString(0) + category_combinations[pow(2, num_categories)];
  \\ add a low label in addition to the body of the lattice
  security_labels[security_labels_index++] =
        toString(num_subjects + 1) + category_combinations[0];
  \\ add the rest of the security labels
  for ( i ← 1 to num_levels ) do
    for ( j ← 0 to category_combinations.size() ) do
      security_labels[security_labels_index++] = toString(i) + category_combinations[j];
    end for    end for    return security_labels
```

Figure 12: BLP security label generation algorithm

References

[1] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-74-244, MITRE, March 1973.

[2] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, MITRE, March 1975.

[3] Konstantin Beznosov. Flooding and recycling authorizations. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 67–72, Lake Arrowhead, CA, USA, 20-23 September 2005.

[4] Matt Bishop. *Introduction to Computer Security.* Addison Wesley, Boston, MA, 2005.

[5] Jason Crampton, Wing Leung, and Konstantin Beznosov. Secondary and approximate authorizations model and its application to Bell-LaPadula policies. In *Proceedings of the Symposium on Access Control Models and Technologies (SACMAT)*, pages 111–120, Lake Tahoe, California, USA, June 7–9 2006. ACM, ACM Press.

[6] Entrust. getaccess design and administration guide. Technical report, Entrust, September 20 1999.

[7] H.M. Gladney. Access control for large collections. *ACM Transactions on Information Systems*, 15(2):154–194, 1997.

[8] G. Karjoth. Access control with ibm tivoli access manager. *ACM Transactions on Information and Systems Security*, 6(2):232–57, 2003.

[9] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software Reliability : Measurement, Prediction, Application.* McGraw-Hill series in software engineering and technology. McGraw-Hill, 1987.

[10] Netegrity. Siteminder concepts guide. Technical report, Netegrity, 2000.

[11] Securant. Unified access management: A model for integrated web security. Technical report, Securant Technologies, June 25 1999.

[12] Kyle Zeeuwen. Saam study user guide. Technical report, LERSSE, Dept. of Elec. and Comp. Engineering, University of British Columbia, 2006.