Stale-Safe Security Properties for Group-Based Secure Information Sharing

Ram Krishnan
George Mason University
Fairfax, VA, USA
rkrishna@gmu.edu

Jianwei Niu
Univ of Texas at San Antonio
San Antonio, TX, USA
niu@cs.utsa.edu

Ravi Sandhu
Univ of Texas at San Antonio
San Antonio, TX, USA
ravi.sandhu@utsa.edu

William H. Winsborough
Univ of Texas at San Antonio
San Antonio, TX, USA
wwinsborough@acm.org

ABSTRACT

Attribute staleness arises due to the physical distribution of authorization information, decision and enforcement points. This is a fundamental problem in virtually any secure distributed system in which the management and representation of authorization state are not globally synchronized. This problem is so intrinsic, it is inevitable that access decision will be based on attribute values that are stale. While it may not be practical to eliminate staleness, we can *limit* unsafe access decisions made based on stale subject and object attributes. In this paper, we propose and formally specify four stale-safe security properties of varying strength which limit such incorrect access decisions. We use Linear Temporal Logic (LTL) to formalize these properties making them suitable to be verified, for example, using model checking. We show how these properties can be applied in the specific context of group-based Secure Information Sharing (g-SIS) as defined in this paper. We specify the authorization decision/enforcement points of the g-SIS system as a Finite State Machine (FSM) and show how this FSM can be modified so as to satisfy one of the stale-safe properties.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection – Access controls; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection –Unauthorized access

General Terms

Security

Keywords

Stale Attributes, Security Properties, Information Sharing

1. INTRODUCTION

The concept of a stale-safe security property is based on the following intuition. In a distributed system authoritative information about subject and object attributes used for access control is maintained at one or more secure authorization information points. Access control decisions are made by collecting relevant subject and object attributes at one or more authorization decision points, and are enforced at one or more authorization enforcement points. Because of the physical distribution of authorization information, decision and enforcement points, and consequent inherent network latencies, it is inevitable that access control will be based on attributes values that are stale (i.e., not the latest and freshest values). In a highly connected high-speed network these latencies may be in milliseconds, so security issues arising out of use of stale attributes can be effectively ignored. In a practical real-world network however, these latencies will more typically be in the range of seconds, minutes and even days and weeks. For example, consider a virtual private overlay network on the internet which may have intermittently disconnected components that remain disconnected for sizable time periods. In such cases, use of stale attributes for access control decisions is a real possibility and has security implications.

We believe that, in general, it is not practical to eliminate the use of stale attributes for access control decisions.[1] In a theoretical sense, some staleness is inherent in the intrinsic limit of network latencies, of the order of milliseconds in modern networks. We are more interested in situations where staleness is at a humanly meaningful scale, say minutes, hours or days. In principle, with some degree of clock synchronization amongst the authorization information, decision and enforcement points, it should be possible to determine and bound the staleness of attribute values and access control decisions. For example, a SAML (Security Assertion Markup Language) assertion produced by an authorization

[1]Staleness of attributes as known to the authoritative information points due to delays in entry of real-world data is beyond the scope of this paper. For example, if an employee is dismissed there may be a lag between the time that action takes effect and when it is recorded in cyberspace. The lag we are concerned with arises when the authoritative information point knows that the employee has been dismissed but at some decision point the employee's status is still showing as active.

decision point includes a statement of timeliness, i.e., start time and duration for the validity of the assertion. It is upto the access enforcement point to decide whether or not to rely on this assertion or seek a more timely one. Likewise a signed attribute certificate will have an expiry time and an access decision point can decide whether or not to seek updated revocation status from an authorization information point.

Given that the use of stale attributes is inevitable, the question is how do we safely use stale attributes for access control decisions and enforcement? The central contribution of this paper is to formalize this notion of "safe use of a stale property" in the specific context of group-based secure information sharing (g-SIS) as defined in this paper. We also demonstrate specifications of systems that do and do not satisfy this requirement. We believe this formalism can be extended to more general contexts beyond the group-based secure information sharing considered in this paper, but this is beyond the current scope. We believe that the requirements for "safe use of a stale property" identified in this paper represent fundamental security properties the need for which arises in virtually any secure distributed systems in which the management and representation of authorization state is not centralized. In this sense, we suggest that we have identified and formalized a *basic security property* of distributed enforcement mechanisms, in a similar sense that non-interference [5] and safety [6] are basic security properties that are desirable in a wide range of secure systems.

Specifically, we present formal specifications of two *stale-safe* properties, one strictly stronger than the other. The most basic and fundamental requirement we consider deals with ensuring that while authorization data cannot be propagated instantaneously throughout the system, in many applications it is necessary that a request should be granted only if it can be verified that it was authorized at some point in the recent past. The second, stronger property says that to be granted, the requested action must have been authorized at a point in time after the request and before the action is performed. We believe that the first property, *weak stale-safety*, is a requirement for most actions (*e.g.*, read or write) in distributed access control systems. We also believe that the second property, *strong stale-safety*, is (further) required of some or all actions in many applications.

We further show how these two properties can be strengthened to bound the acceptable level of staleness in terms of time elapsed between the point at which the request was last known to have been authorized and the point at which the action is performed. Thus, including these two strengthened versions, we have a total of four stale-safe properties.

We formalize these four properties in Linear Temporal Logic (LTL), making them suitable to be verified by using model checking. We show how these properties can be applied in the specific application domain of group-based secure information sharing (g-SIS). We specify one component of a g-SIS system as a finite state machine (FSM). We present an FSM that does not satisfy the weakest of our state-safe properties and show how it can be modified to satisfy the property.

The remainder of the paper is organized as follows. In section 2, we discuss the group-based Secure Information Sharing problem which will be used throughout the paper to illustrate the stale-safe properties. In section 3, we formalize the stale-safe security properties using Linear Temporal Logic. We specify a weak and strong version of the properties each of which is further restricted with a notion of elapsed time between the time at the which the operation is authorized and performed. In section 4, we demonstrate the construction of two Finite State Machines (FSM)—one FSM that is stale-unsafe and the other that enforces a g-SIS policy in a stale-safe manner. In section 5, we discuss related work, future work and conclude.

2. GROUP-BASED SECURE INFORMATION SHARING

Secure Information Sharing (SIS) or sharing information *while* protecting it is one of the earliest problems to be recognized in computer security, and yet remains a challenging problem to solve. A detailed discussion of SIS problem motivation and solution approaches can be found in [9]. The central problem is that copies of digital information are easily made and controls on the original typically do not carry over to the copies. One approach tried in the past has been to tie access control to each copy also so that copies are as tightly controlled as the original. The most common form of this approach is so-called mandatory or lattice-based access control [15] where copies are also labeled to reflect security sensitivity of the original. More recently, an alternate approach has emerged wherein plaintext unprotected copies are prohibited, while encrypted protected copies can be freely made. This implies that access controls need to be enforced on the client machines where the content is decrypted and displayed, so as to ensure that only authorized users get to see the content and that they are unable to make plaintext unprotected copies. There has been considerable interest in this approach, initially driven by the forces of digital rights management for entertainment content seeking to protect revenue but more generally seeking to protect content for its sensitivity.

2.1 Objectives

The group-based SIS (g-SIS) problem [9] is motivated by the need to share sensitive information amongst a group of authorized users. For simplicity we only consider the case of read access to the objects and addition of objects to the group. For purpose of this paper, we specify the following objectives for the g-SIS problem. For brevity, the terms subjects and objects refer to subjects and objects that belong to the group.

1. Objects are always protected (encrypted) and never exists in plain text except when viewed.

2. Objects are assumed to be available via super-distribution. This simply means that the objects are protected once and subjects may access them when authorized. They are not individually prepared for each subject.

3. Subjects can access objects off-line without involving the server using trusted access machines. The degree of trust required on the access machines may vary depending on the application and policy. In one case, the access machines may be implicitly trusted because of its physical location (e.g. access machines in an organization). In a completely distributed setting, a Trusted Reference Monitor (TRM) needs to be present on the access machines that can verify the integrity of the sys-

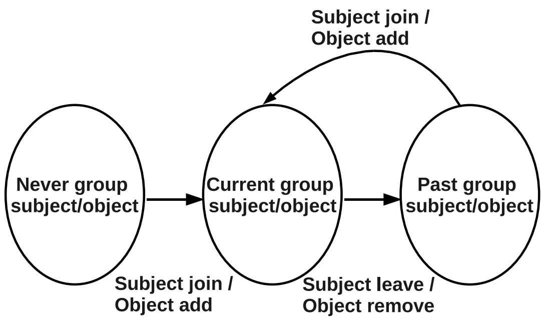

Subject join /
Object add
Never group subject/object
Current group subject/object
Past group subject/object
Subject join /
Object add
Subject leave /
Object remove

Figure 1: Subject and object membership states.

tem and enforce the authorization policies in a trustworthy manner. This can be achieved using integrity measurements, remote attestation and other features enabled by Trusted Computing Technology [1] or software analogs of this technology[2].

4. Each group has a Group Administrator (GA) who controls group membership and policies. The GA can add or remove subjects and objects from the group. Each group also has a Control Center (CC), a server that maintains authoritative subject and object attributes and provides group credentials to new members. Changes in subject and object attributes are updated by the GA at the CC and this change will eventually propagate to the subject's access machines (discussed later in detail).

5. When a subject joins or leaves the group, remaining members should not be affected. In other words, join and leave of a subject should be completely oblivious to other subjects. Remaining subjects cannot be forced to be online.

6. Secure multicast, a related problem, focuses on maintaining forward and backward secrecy of data [14]. Forward-secrecy requires that a leaving subject should not be able to read data that will subsequently be exchanged in the future. Backward-secrecy requires that a joining subject should not be able to read data exchanged amongst the remaining subjects in the past. However, *information* sharing may not be limited to forward and backward-secrecy. For g-SIS flexible membership policies are required.

2.2 Group Management and Policy Enforcement

Subjects and objects in a group go through various states as shown in figure 1. Different access policies are possible depending on the relative state of subjects and objects. For example, a joining subject could be allowed access only to new objects or also to objects that currently exist in the group. Similarly, a past subject may lose access to all objects or retain access to objects authorized during his membership period. When a subject rejoins the group, he may either gain access to objects authorized during his past membership or simply join the group as a new subject. Similarly, many different object policies are possible. Detailed discussions can be found in [9]. Each group may thus pick a specific set of group-level access policies for subjects and objects.

[2]It is generally accepted that software-only solutions will provide a lower degree of assurance than solutions with a hardware root of trust. The issues discussed in this paper are orthogonal to assurance so will apply to both software and hardware based solutions.

Figure 2 shows one possible enforcement model for the g-SIS problem and illustrates the interaction of various components in g-SIS. The Group Administrator (GA) controls group membership and policies. The Control Center (CC) is responsible for maintaining authoritative group credentials and attributes of group subjects and objects on behalf of the GA.

- *Subject Join*: Joining a group involves obtaining authorization from the GA followed by obtaining group credentials from the CC. In step 1.1, the subject contacts the GA using an access machine and requests authorization to join a group. The GA verifies that the subject is not already a member and authorizes the subject in step 1.2 (by setting AUTH to TRUE). The subject furnishes the authorization to join the group and the evidence that the access machine is in a good software state to the CC in step 1.3. The CC remotely verifies GA's authorization, if the subject's access machine is trustworthy (using the evidence) and has a known Trusted Reference Monitor (TRM) that is responsible for enforcing policies. In step 1.4, the CC provisions the attributes. sid is the Subject Id, Join_TS is the time-stamp of subject join (set to a non-NULL value), Leave_TS is the time at which a subject leaves the group (initially set to NULL), gKey is the group key using which group objects can be decrypted, Policy is the group's access policy, ORL is the Object Revocation List which lists the objects removed from the group.

- *Policy Enforcement*: From here on, the subject is considered a group member and may start accessing group objects (encrypted using the group key) as per the group policy and using the credentials obtained from the CC. This is locally mediated and enforced by the TRM. Note that the objects are available via super-distribution and because of the presence of a TRM on subject's access machines, objects may be accessed offline conforming to the policy. For example, the TRM on an access machine may allow the subject to access objects added after subject joined the group and disallow access to objects added before he/she joined the group. Such decisions can be made by using the join and leave time-stamps of subject, add and remove time-stamps of object and comparing their relative values. Objects may be added to the group by subjects by obtaining an add time-stamp (setting an Add_TS attribute for the object) from the CC. We assume object attributes are embedded in the object itself. Note that due to super-distribution, the remove time-stamps for objects cannot be embedded in the object (since there could be many copies of the same object). Instead, an Object Revocation List (with the remove time-stamps of object ids) is provisioned on the access machine.

- *Attribute Refresh*: Since subjects may access objects offline, the access machines need to connect to the CC and refresh subject attributes periodically. How this is

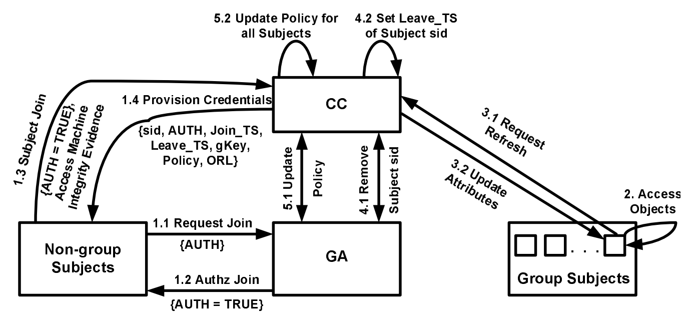


Figure 2: g-SIS System.

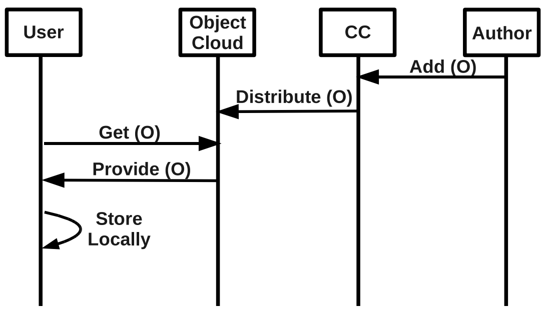


Figure 3: Create and Propagate.

done is a matter of policy and/or practicality. For example, a refresh could take effect in an access machine based on time or a usage count. Offline access to secure clock may be impractical in many circumstances. Usage count is a practical approach when using Trusted Computing Technology. The usage count limits the number of times the credentials may be used to access group objects (like consumable rights). Thus objects may be accessed until the usage count is exhausted and the access machine will be required to refresh attributes in step 3.1 and 3.2 before any further access can be granted. Attributes RT and N represent the refresh time-stamp and usage count of the subject respectively.

- *Administrative Actions*: The GA may have to remove a subject or object from the group or update group policy. In step 4.1, the GA instructs the CC to remove a subject. The CC in turn marks the subject for removal by setting the subject's Leave_TS attribute in step 4.2. This attribute update is communicated to the subject's access machine during the refresh step 3.1 and 3.2. In the case of object removal, the ORL is updated with the object's id and Remove_TS. Policy updates (or any other update for that matter) are handled in a similar manner as shown in step 5.1 and 5.2.

As one can see, there is a delay in attribute update in the access machine that is defined by the refresh window. Although a subject may be removed from the group at the CC, the access machines will let subjects access group objects until the subject attributes are refreshed at the next refresh step. This access violation is due to attribute staleness that is inherent to any distributed system however short the refresh window is. We discuss this topic in detail in the subsequent sections. This paper does not focus on building trusted systems to realize the architecture in figure 2 and is outside the scope of this paper.

3. STALE SAFE SECURITY PROPERTIES

As discussed earlier, in distributed systems access decisions are almost always based on stale-attributes which lead to critical access violations. In this section we propose stale-safe security properties that limit such access violations. Note that it is impossible to completely eliminate staleness in practice and thus our intension here is best effort. We first discuss a few scenarios where stale attributes lead to access violations using the g-SIS example and informally discuss the stale-safe properties. We formalize them next.

3.1 System Characterization

The g-SIS system consists of subjects and objects, trusted access machines (using which objects are accessed), a GA and a CC. Access machines maintain a local copy of subject attributes which they refresh periodically with the CC. Object attributes are part of the object itself. A removed object is listed in the Object Revocation List (ORL) which is provided to access machines as part of refresh. To easily illustrate the properties, we assume that each subject is tied to an access machine from which objects are accessed and there is a single GA and single CC per group. Also, we assume that the refresh is based on usage count. Suppose a policy that a subject is allowed to access an object as long as both the subject and object are current members of the group and the object was added after the subject joined the group. Thus the g-SIS system can be characterized as follows:

Subject attributes	{id, Join_TS, Leave_TS, ORL, gKey, RT, N}
Object attributes	{id, Add_TS}.
Refresh Time (RT)	TRM contacts CC to refresh subject attributes and ORL.
Refresh Window (RW)	Interval between two RT's (depends on how quick usage count is exhausted).
Access Policy	Authz$(S, O, OP) \rightarrow O \notin$ ORL $\wedge$Leave_TS(S) = NULL$\wedge$ Join_TS$(S) \leq$ Add_TS(O).

Figure 3 illustrates super-distribution. An Author (a group subject) creates an object, encrypts the object using the group key (mediated by TRM) and sends it to the CC for approval and distribution. The CC (or possibly a GA) approves the object, time-stamps object add and releases this protected object for distribution. Since the object is protected, it is not necessarily guarded by the CC. Instead it is made available to subjects by distribution through networks such as WWW, email, etc. This infospace is called the Object Cloud in figure 3. The User (another group subject) can obtain these encrypted objects and store them locally in his/her access machine. The sequence diagram in Figure 4 illustrates the staleness problem. The User and the TRM interacts with the GA and CC in steps 1 to 5 to join the group. The TRM refreshes attributes with the CC in steps 6 and 7. Briefly after the refresh, the GA removes this subject (step 8) by setting his/her Leave_TS attribute at the CC (a non-null value). Note that this step is not visible to the TRM until the next refresh steps 11 and 12. In the mean time, the User may request access to objects the were obtained via super-distribution (step 9). "Create and Propagate" refers to the scenario in figure 3. At this point, the TRM evaluates the policy based on the attributes that it maintains. This should be successful and the object is displayed to the user in step 10. Note the difference in Leave_TS values between the CC and TRM. Only after the following refresh (steps 11 and 12) does the TRM notice that the subject has been removed from the group and denies any further access (steps 13 and 14).

Figure 5 shows a timeline of events involving a single group. Subject S1 joins the group and the attributes are refreshed with the CC periodically. RT represents the time at which refreshes happen. The time period between any two RT's is a Refresh Window, denoted RW_i. After join, RW_0 is the first window, RW_1 is the next and so on. Suppose RW_4 is the current Refresh Window. Objects O1 and O2 were added to the group by *some* group subject (or the GA) during RW_2 and RW_4 respectively and they are available to S1 via super-distribution. In RW_4, S1 requests access to O1 and O2. An access decision will be made by the TRM in the access machine as per the attributes obtained at the latest RT.

Clearly, our access policy will allow access to both O1 and O2. However it is possible that S1 was removed by the GA right after the last RT and before Request(S1, O1, access) in RW_4 (see figure 4). Ideally, S1 should not be allowed to access both O1 and O2.

From a confidentiality perspective in information sharing, granting S1 access to O1 is relatively less of a problem than granting access to O2. This is because the CC or the GA can assume that S1 was always authorized access to O1 and hence information has already been released to S1. In the worst case, S1 continues to access the same information (O1) until the next RT. However, S1 never had an authorization to access O2 and letting S1 access O2 means that S1 has gained knowledge of new information. This is a critical violation and should not be allowed. Such scenarios are what our stale-safe security properties address. A subject cannot access an object if it was added to the group after the last refresh time even if the authorization policy allows access.

The property we discussed considers attributes to be stale if it is time-stamped later than the last refresh time-stamp of the access machine. A more strict property may require the access machine to refresh attributes before granting any access. That is, when S1 requests access to O1, the stricter version of the stale-safe property mandates that the access machine refreshes the subject attributes before making an authorization decision. Further, it is natural to consider elapsed time since the last refresh to be an important issue in limiting staleness of authorization data. We formalize these notions in the following subsection.

3.2 Formal Property Specification

In this section we use Linear Temporal Logic (LTL) [12] to specify four different formal stale-safety properties of varying strength. Temporal logic is a specification language for expressing properties related to a sequence of states in terms of temporal logic operators and logic connectives (e.g., $\wedge$ and $\vee$). Temporal logic operators are of two types: Past and Future. The past operators $\ominus$ and $\mathcal{S}$ (read previous and since respectively) have the following semantics. $\ominus\, p$ means that the formula p was true in the previous state. Note that $\ominus\, p$ is false in the very first state. $p\ \mathcal{S}\ q$ means that q has happened sometime in the past and p has held continuously following the last occurrence of q to the present. The future operators $\bigcirc$, $\diamond$, and $\square$ represent next state, some future state, and all future states respectively. For example, $\square p$ means that formula p is true in all future states. Also, the formula p *until* q (read p until q) means that q will occur sometime in the future and p will remain true at least until the first occurrence of q.

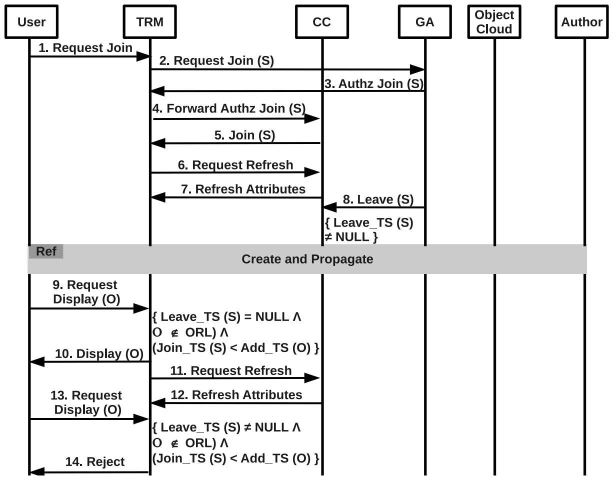


Figure 4: Staleness Illustration.

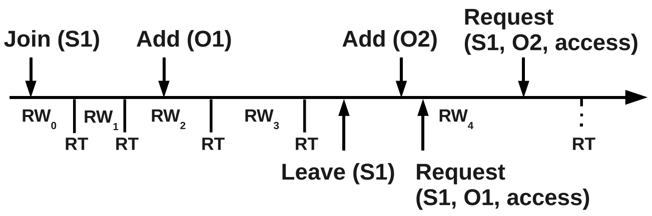


Figure 5: Events on a time line illustrating staleness leading to access violation.

Our formalization uses the following predicates:

request (S,O,OP)	S requests to perform an action OP on O.
Authz (S,O,OP)	S is authorized to perform an action OP on O.
Join (S) and Leave (S)	Join & Leave events of S.
Add (O) and Remove (O)	Add & Remove events of O.
perform (S,O,OP)	S performs OP on O.
RT (S)	The TRM contacts the CC to update subject attributes.

In the forthcoming formulae (φ_0, φ_1 and φ_2) and throughout this paper, we drop the corresponding parameters S, O and OP in these predicates for clarity. They should however be interpreted with the respective semantics described above. Further, we assume that the very first join event of a subject is equivalent to RT (since attributes are set at join time).

3.2.1 Access Policy Specification

We first formalize the access policy discussed in section 3.1 as an example. Note that, in distributed systems such as g-SIS, events such as Remove and Leave cannot be instantaneously observed by the TRM. Such information (that a subject or an object is no longer a group member) can only be obtained from CC at subsequent refresh times (RT's). Thus, we have a notion of ideal or desirable policy that assumes instant propagation of authorization information (like that of a centralized system). This is enforceable only at the CC. However, while designing the TRM (that is, in a distributed setting), one has to re-formulate this ideal policy using available authorization information so that it is enforceable locally by the TRM. We call the former Authz_{CC} and the latter $\text{Authz}_{\text{TRM}}$.

Authz_{CC} below is the same policy in section 3.1 represented using LTL. Figure 6 illustrates Authz_{CC}. Authz_{CC} says that S is allowed to perform an action OP on O if prior to the current state the object was added to the group and both the subject and object have not left the group since. Also, the subject joined the group prior to the time at which the object was added to the group and has not left the group ever since. Clearly, this can be enforced only by the CC.

$$\begin{aligned}
\text{Authz}_{\text{CC}} \equiv\ & ((\neg\text{Remove} \wedge \neg\text{Leave})\ \mathcal{S}\ (\text{Add}\wedge \\
& (\neg\text{Leave}\ \mathcal{S}\ \text{Join}))) \\
\text{Authz}_{\text{TRM}} \equiv\ & (\neg\text{RT}\ \mathcal{S}\ (\text{Add} \wedge (\neg\text{RT}\ \mathcal{S}\ (\text{RT}\wedge \\
& (\neg\text{Leave}\ \mathcal{S}\ \text{Join}))))) \vee \\
& (\neg\text{RT}\ \mathcal{S}\ (\text{RT} \wedge ((\neg\text{Remove} \wedge \neg\text{Leave}) \\
& \mathcal{S}\ \text{Add}) \wedge (\neg\text{Leave}\ \mathcal{S}\ \text{Join}))) \\
\varphi_0 \equiv\ & \circleddash\ ((\neg\text{perform} \wedge (\neg\text{RT} \vee (\text{RT} \wedge \text{Authz}_{\text{TRM}}))) \\
& \mathcal{S}\ (\text{request} \wedge \text{Authz}_{\text{TRM}}))
\end{aligned}$$

Let us now re-formulate Authz_{CC} so that it is enforceable locally at TRM. Recall from section 2 that once a subject joins the group, authorization information such as subject join time and object add time are available instantaneously to the TRM—subject join time is provisioned at the TRM on successful join and object add time is available via super-distribution. However, subject leave time and object remove time are only available at refresh times from the CC. Thus, in our re-formulation, we have a constraint that any mention of Leave and Remove occur only at RT time. However, Join and Add are free of this constraint and can be used at any point in time.

$\text{Authz}_{\text{TRM}}$ above shows the re-formulation. It is a disjunction of two cases. The first part takes care of the case in which the requested object is added after the most recent RT. This is illustrated in case (a) of figure 7 where we are only able to verify that the subject was still a member at RT. Since the object was not added prior to that point, we are unable to do a similar check for the object. The second part handles the situation where the object is added before the most recent RT. This is illustrated in case (b) of figure 7 where we are able to verify that at RT both the subject and object are current members. Note that in both cases (a) and (b), our evaluation of policy is based on authorization information available at RT.

Figure 8 is a pictorial representation of formula φ_0. It illustrates how a reference monitor traditionally reacts to a request to access an object. When a request arrives from a subject, the TRM verifies if the policy ($\text{Authz}_{\text{TRM}}$) holds

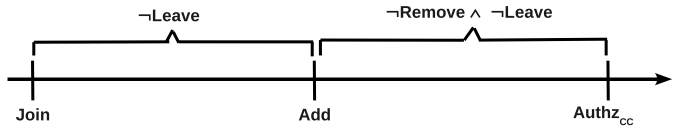


Figure 6: Ideal Access Policy (Authz_{CC}).

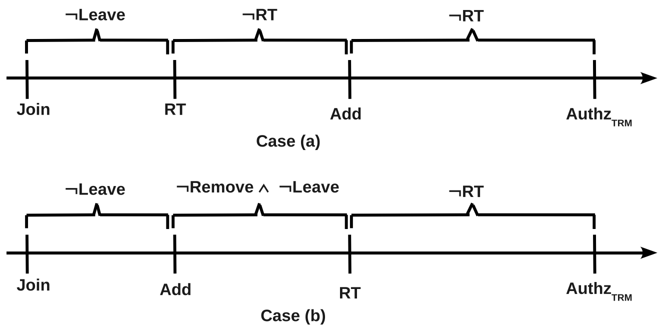


Figure 7: Approximate Access Policy ($\text{Authz}_{\text{TRM}}$).

at that point. If successful, the TRM allows the subject to perform the requested action subsequently. Note that if an RT occurs in the meantime, the TRM re-evaluates the policy with the updated attributes. Thus, φ_0 says that the operation was authorized at the time of request and prior to the current state, the operation has not been performed since it was requested. One can now see that the formula $\Box$ (perform $\rightarrow \varphi_0$) reflects this behavior of the TRM.

However, observe that verifying that $\text{Authz}_{\text{TRM}}$ holds at the time of request will allow the subject access to objects that were added during the time between RT and request $\wedge$ $\text{Authz}_{\text{TRM}}$ in figure 8. We illustrated this in case (a) of figure 7. As discussed earlier, it is unsafe to let subjects access these objects before a refresh can confirm the validity of their group membership.

We next specify stale-safe security properties of varying strength. The weakest of the properties we specify requires that a requested action be performed only if a refresh of authorization information has shown that the action was authorized at that time. This refresh is permitted to have taken place either before or after the request was made. The last refresh must have indicated that the action was authorized and all refreshes performed since the request, if any, must also have indicated the action was authorized. This is the *weak stale-safe security property.* By contrast, the *strong stale-safe security property* requires that the confirmation of authorization occur after the request and before the action is performed.

3.2.2 Weak Stale-safe Security Property

Let us introduce two formulas formalizing pieces of stale-safe security properties. Intuitively, φ_1 can be satisfied only if authorization was confirmed prior to the request being made. On the other hand, φ_2 can be satisfied only if authorization was confirmed after the request. Note that weak

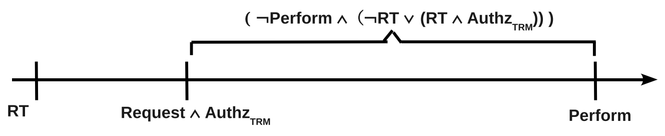


Figure 8: Formula φ_0.

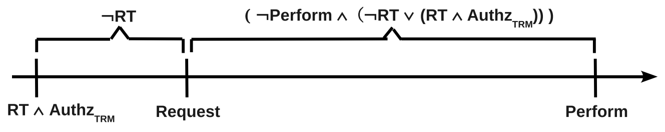


Figure 9: Formula φ_1.

stale safety is satisfied if either of these is satisfied prior to a requested action being performed.

$$\begin{aligned}\varphi_1 \equiv\ & \circleddash\ (\neg\text{perform} \wedge (\neg\text{RT} \vee (\text{RT} \wedge \text{Authz}_{\text{TRM}})))\\ & \mathcal{S}\ (\text{request} \wedge (\neg\text{RT}\ \mathcal{S}\ (\text{RT} \wedge \text{Authz}_{\text{TRM}})))\\ \varphi_2 \equiv\ & \circleddash\ (\neg\text{perform} \wedge \neg\text{RT})\ \mathcal{S}\ (\text{RT} \wedge \text{Authz}_{\text{TRM}} \wedge\\ & ((\neg\text{perform} \wedge (\neg\text{RT} \vee (\text{RT} \wedge \text{Authz}_{\text{TRM}})))\ \mathcal{S}\ \text{request}))\end{aligned}$$

Figure 9 illustrates formula φ_1. φ_1 says that prior to the current state, the operation has not been performed since it was requested. Also since it was requested, any refreshes that may have occurred indicated that the operation was authorized ($\neg$RT $\vee$ (RT $\wedge$ Authz$_{\text{TRM}}$)). Finally, a refresh must have occurred prior to the request and the last time a refresh was performed prior to the request, the operation was authorized.

Observe that formula φ_1 mainly differs from φ_0 on the point at which Authz$_{\text{TRM}}$ is evaluated. Referring to figure 9, evaluating Authz$_{\text{TRM}}$ at the latest RT guarantees that requests to access any object that may be added during the following refresh window will be denied.

Note that φ_1 is satisfied if there is no refresh between the request and the perform. It requires that any refresh that happens to occur during that interval indicate that the action remains authorized. In our g-SIS application, this could preclude an action being performed, for instance, if the subject leaves the group, a refresh occurs, indicating that the action is not authorized, the subject rejoins the group, and another refresh indicates that the action is again authorized. For some applications, this might be considered unnecessarily strict.

Figure 10 illustrates formula φ_2. φ_2 does not require that there was a refresh prior to the request. Instead it requires that a refresh occurred between the request and now. It further requires that the operation has not been performed since it was requested and that every time a refresh has occurred since the request, the operation was authorized.

Note that φ_2 can be satisfied without an authorizing refresh having occurred prior to the request, whereas φ_1 cannot. Thus, though φ_2 ensures fresher information is used to make access decisions, it does not logically entail φ_1 as it is satisfied by traces that do not satisfy φ_1.

We call perform $\rightarrow \varphi_1$ *backward-looking stale safety*, as it does not require that a confirmation of authorization occur after the request has been received. We call perform $\rightarrow \varphi_2$ *forward-looking stale safety*, as it requires that confirmation

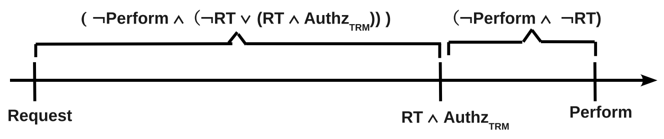


Figure 10: Formula φ_2.

of authorization is obtained after the request, before the action is performed.

Definition 3.1 (Weak stale safety). *An FSM has the* weak stale-safe security property *if it satisfies the following LTL formula:*

$$\Box\ (\text{perform} \rightarrow (\varphi_1 \vee \varphi_2))$$

3.2.3 Strong Stale-safe Security Property

Forward-looking stale safety is strictly stronger than weak stale safety. For this reason, and because, unlike backward-looking stale safety, it is a reasonable requirement for controlling many operations, we give it a second name.

Definition 3.2 (Strong stale safety). *An FSM has the* strong stale-safe security property *if it satisfies the following LTL formula:*

$$\Box\ (\text{perform} \rightarrow \varphi_2)$$

Note that our formulas (φ_0, φ_1 and φ_2) were concerned about the temporal placement of refresh time (RT) with respect to the occurrence of the time at which the request came from the subject, the time at which the requested action is performed and the evaluation of Authz$_{\text{TRM}}$. This separation of request and perform is important to differentiate weak-stale safety from strong-stale safety property. In weak-stale safety, although Authz$_{\text{TRM}}$ was evaluated at RT prior to request, it is possible for an RT to occur between request and perform. If fresh attributes are available, it is important to re-evaluate Authz$_{\text{TRM}}$. Formula φ_1 requires that Authz$_{\text{TRM}}$ continues to hold at such occurrences. On the other hand, strong-stale safety mandates that after request, the action cannot be performed until Authz$_{\text{TRM}}$ is evaluated with up-to-date attributes.

3.2.4 Quantifying "Freshness" of Authorization

Let us now consider how to express requirements that bound acceptable elapsed time between the point at which attribute refresh occurs and the point at which a requested action is performed. We refer to this elapsed time as the degree of *freshness*. For this we introduce a sequence of propositions $\{P_i\}_{0\leq i\leq n}$ that model n time intervals. These propositions partition each trace into contiguous state subsequences that lie within a single time interval, with each proposition becoming true immediately when its predecessor becomes false. They can be axiomatized as follows: P_1 Until ($\Box\neg P_1 \wedge (P_2$ Until ($\Box\neg P_2 \wedge (P_3$ Until (... Until ($\Box\neg P_{n-1} \wedge \Box P_n$)...))))). This partially[3] defines correct behavior of a clock, given by a component of the FSM. The current time can be interrogated by the other FSM components with which it is composed. It can also be referred

[3]Note that is it not possible to express in LTL that the clock transits from P_i to P_{i+1} at regular intervals of elapsed time.

to in the variant stale-safe properties presented in the following paragraphs. If the clock is accurate with respect to transiting from P_i to P_{i+1} at regular intervals, the enforcement machine obeying these variant properties will enforce freshness requirements correctly.

We now formulate variants of φ_1 and φ_2 that take a parameter k indexing the current time interval. These formulas use two constants, ℓ_1 and ℓ_2 which represent the number of time intervals since the authorization and the request, respectively, that is considered acceptable to elapse prior to performing the requested action. The formulas prohibit performing the action if either the authorization or the request occurred further in the past than permitted by these constants.

$$\begin{aligned}\varphi_1(k) \equiv \circleddash\, & ((\neg\text{perform} \wedge (\neg\text{RT} \vee (\text{RT} \wedge \text{Authz}_{\text{TRM}}))) \;\mathcal{S} \\ & (\text{request} \wedge \bigvee_{max(0,k-\ell_2)\leq i\leq k} P_i \wedge \\ & (\neg\text{RT} \;\mathcal{S}\; (\text{RT} \wedge \text{Authz}_{\text{TRM}} \wedge \bigvee_{max(0,k-\ell_1)\leq i\leq k} P_i))))\end{aligned}$$

$$\begin{aligned}\varphi_2(k) \equiv \circleddash\, & (\neg\text{perform} \wedge \neg\text{RT}) \;\mathcal{S} \\ & (\text{RT} \wedge \text{Authz}_{\text{TRM}} \wedge \bigvee_{max(0,k-\ell_1)\leq i\leq k} P_i \wedge \\ & ((\neg\text{perform} \wedge (\neg\text{RT} \vee (\text{RT} \wedge \text{Authz}_{\text{TRM}}))) \;\mathcal{S} \\ & (\text{request} \wedge \bigvee_{max(0,k-\ell_2)\leq i\leq k} P_i))\end{aligned}$$

With these formulas, we are now able to state variants of weak and strong stale safety that require timeliness, as defined by the parameters ℓ_1 and ℓ_2.

DEFINITION 3.3 (TIMELY, WEAK STALE SAFETY). *An FSM has the* timely, weak stale-safe security property *if it satisfies the following LTL formula:*

$$\Box\, (\bigwedge_{0\leq k\leq n} (\text{perform} \wedge P_k) \rightarrow (\varphi_1(k) \vee \varphi_2(k)))$$

DEFINITION 3.4 (TIMELY, STRONG STALE SAFETY). *An FSM has the* timely, strong stale-safe security property *if it satisfies the following LTL formula:*

$$\Box\, (\bigwedge_{0\leq k\leq n} (\text{perform} \wedge P_k) \rightarrow \varphi_2(k))$$

3.3 Stale-safe Systems

We discuss the significance of the weak and strong stale-safe properties in the context of stale-safe systems designed for confidentiality or integrity. Confidentiality is concerned about information release while integrity is concerned about information modification. Both weak and strong properties are applicable to confidentiality –the main trade-off between weak and strong here is usability. Weak allows subjects to read objects when they are off-line while strong forces subjects to refresh attributes with the server before access can be granted. Depending on the security and functional requirements of the system under consideration, the designer has the flexibility to choose between weak and strong to achieve stale-safety. In the case of integrity, the weak property can be risky in many circumstances –the strong property is more desirable. This is because objects modified by unauthorized subjects may be used/consumed by other subjects before the modification can be undone by the server. For instance, in g-SIS, a malicious unauthorized subject (i.e. a malicious subject who has been revoked group membership but is still allowed to modify objects for a time period due to stale attributes) may inject bad code and share it with the group. Other unsuspecting subjects who may have the privilege to execute this code may do so and cause significant damage. In another scenario, a malicious subject may inject incorrect information into the group and other subjects may perform certain critical actions based on faulty information. Thus, although both weak and strong properties may be applicable to confidentiality and integrity, the weak property should be used with a caveat in the case of integrity.

4. MODELING TRM

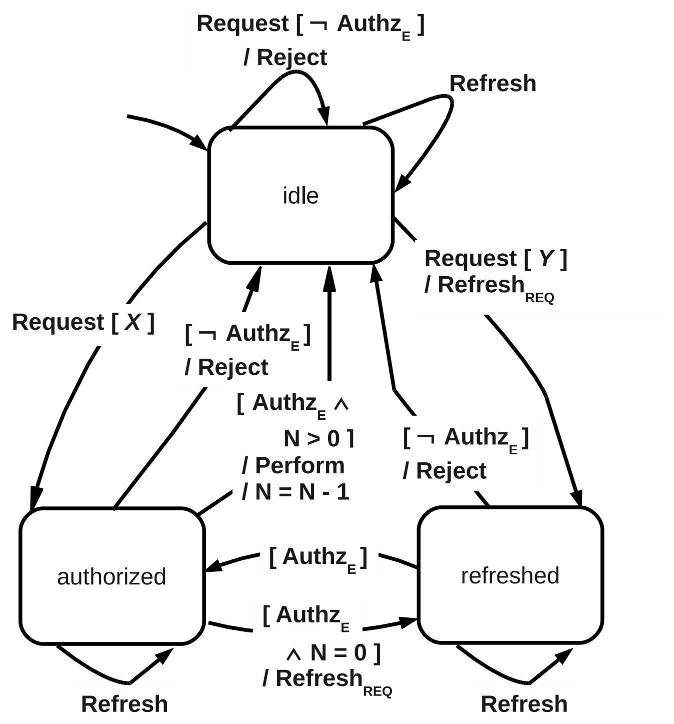
The TRM we consider models refresh based on usage count. The CC/GA determines a usage count for each subject which specifies the number of times the group credentials (e.g. group key) may be used by the TRM to access objects off-line before a refresh is required. Suppose N is the usage count. Every time a subject accesses an object, N is decremented by the TRM. Once N reaches zero for that subject, the TRM denies access to any object until the attributes are refreshed with the CC. As part of this refresh, N is reset to the initial value.

We discuss the construction of a TRM machine, FSM_{TRM}, that satisfies Weak Stale Safety property (Definition 3.1). Figure 11 shows one possible design of FSM_{TRM} to enforce an authorization policy given by Authz_{E}. Two versions of FSM_{TRM} are shown in the figure, one stale-unsafe and one stale-unsafe. Each version is obtained by interpreting X and Y, as indicated. Observe that the authorization policy (the LTL formula $\text{Authz}_{\text{TRM}}$ in section 3) has been re-written using attributes, denoted Authz_{E}. Since we are at the enforcement layer, we specify the policy using attributes instead of events. As earlier, Authz_{E} indicates that the object O has not been removed from the group, subject S has not left the group, and the subject S joined the group before the object O was added. We label state transitions using the format $e[C]/A$, in which e is the event, C is the condition that has to be satisfied to enable the transition, and A represents actions that need to be performed when the transition is taken.

The FSM_{TRM} is responsible for mediating request from the subject to access the object to which it corresponds. It remains in the idle state until a request to access the object arrives from the subject. At this point, FSM_{TRM} checks the authorization policy (Authz_{E}) to decide whether the subject can access the requested object. At this point, there are three possible paths the FSM can take, depending on which condition is satisfied. Let us first consider stale-unsafe FSM_{TRM}:

Request[$\neg\text{Authz}_{\text{E}}$]: If Authz_{E} fails, FSM_{TRM} rejects the request and remains in the idle state.

Request[$\text{Authz}_{\text{E}} \wedge N = 0$]: If Authz_{E} succeeds, but the usage count is exhausted, the machine is required to refresh attributes before any access can be granted. A refresh request action ($\text{Refresh}_{\text{REQ}}$) is initiated in this case. This creates a synchronizing event (transitions labeled Refresh in the idle, authorized and refreshed states) for all the FSM_{TRM} instances in the local TRM, which has the effect of updating all subject attributes, as well as the ORL, with the values that are current at the CC. The synchronous event simply ensures that the update is atomic with respect to transitions


Request [¬ Authz_E] / Reject
Refresh
idle
Request [Y] / Refresh_REQ
Request [X]
[¬ Authz_E] / Reject
[Authz_E ∧ N > 0] / Perform / N = N - 1
[¬ Authz_E] / Reject
authorized
[Authz_E]
refreshed
[Authz_E ∧ N = 0] / Refresh_REQ
Refresh
Refresh

$\mathrm{Authz}_E(S,O,OP) \rightarrow \neg\mathrm{Remove\_TS}(O) \wedge \neg\mathrm{Leave\_TS}(S) \wedge \mathrm{Join\_TS}(S) \leq \mathrm{Add\_TS}(O)$

$\mathrm{Stale} \equiv \mathrm{R\_TS} \leq \mathrm{Add\_TS}(O)$

Stale-Unsafe FSM_{TRM}:	Stale-Safe FSM_{TRM}:
$X \equiv \mathrm{Authz}_E \wedge N > 0$	$X \equiv \mathrm{Authz}_E \wedge N > 0 \wedge \neg\,\mathrm{Stale}$
$Y \equiv \mathrm{Authz}_E \wedge N = 0$	$Y \equiv \mathrm{Authz}_E \wedge (\mathrm{Stale} \vee N = 0)$

Figure 11: Two versions of $\mathrm{FSM_{TRM}}$: One Stale-unsafe and one Stale-safe. Each version is obtained by interpreting X and Y, as indicated.

at every $\mathrm{FSM_{TRM}}$ instance. After the refresh, the $\mathrm{FSM_{TRM}}$ enters the refreshed state. It again checks the authorization policy to see whether the subject is now allowed the requested access in light of the updated attribute values. If $\mathrm{Authz_E}$ holds, $\mathrm{FSM_{TRM}}$ directly enters the authorized state. If $\mathrm{Authz_E}$ does not hold, the $\mathrm{FSM_{TRM}}$ denies access and immediately returns to idle.

$Request[\mathrm{Authz_E} \wedge N > 0]$: If $\mathrm{Authz_E}$ succeeds and the usage count N is not exhausted, the machine enters the authorized state and waits for the subject to access the object. The requested action is performed only after re-checking the policy $\mathrm{Authz_E}$ and decrementing the usage count. If $\mathrm{Authz_E}$ does not hold, the action is rejected. This re-checking is critical because $\mathrm{Authz_E}$ checked earlier may no longer hold due to updated attributes received from the Refresh transition which could possibly be triggered by another instance of $\mathrm{FSM_{TRM}}$. We discuss this in more detail in the following paragraph. $\mathrm{FSM_{TRM}}$ thereafter returns to the idle state.

Consider the self-transitions labeled Refresh in each of the states in $\mathrm{FSM_{TRM}}$. It is needed to allow refreshes initiated by other $\mathrm{FSM_{TRM}}$'s to occur atomically with respect to other transitions. When a subject requests access to multiple objects, a separate instance of $\mathrm{FSM_{TRM}}$ runs for each such object. It is possible that updated attributes are available when a $\mathrm{FSM_{TRM}}$ instance is at authorized state. Consider the following example. In a typical system, multiple subjects request access to multiple objects at the same time. When an $\mathrm{FSM_{TRM}}$ instance enters and waits at authorized state for a specific request, it is possible that a refresh request was triggered by other $\mathrm{FSM_{TRM}}$ instances and updated attributes are available. In light of this update, the $\mathrm{FSM_{TRM}}$ waiting at authorized state has a chance to re-evaluate the policy before the action can be performed by the subject. Thus the Refresh transitions simply ensure that in the event of refresh triggerred by any instance of $\mathrm{FSM_{TRM}}$, all other running instances obtain the fresh authorization information and re-evaluate the policy.

The $\mathrm{FSM_{TRM}}$ in figure 11 is not stale-safe because it allows access to objects that were added after the last refresh time. This problem is fixed in stale-safe $\mathrm{FSM_{TRM}}$ which we now discuss.

Consider the stale-safe version of $\mathrm{FSM_{TRM}}$ in Figure 11. As indicated, it differs from stale-unsafe $\mathrm{FSM_{TRM}}$ in the extra check that is carried out for stale safety. As per the weak-stale safety property, objects added after the most recent RT cannot be accessed until after refresh. The staleness check ($\mathrm{RT\_TS} \leq \mathrm{Add\_TS}(O)$) makes sure that the property holds. The transition from idle to authorized is enabled if the authorization policy succeeds, the usage count is still available *and* the attributes are not stale (i.e., $\mathrm{Add\_TS}(O) < \mathrm{RT\_TS}$). The transition from idle to refreshed is enabled if the authorization policy is successful but either the off-line usage limit is reached *or* the attributes are stale. From refreshed, $\mathrm{FSM_{TRM}}$ enters authorized state if $\mathrm{Authz_E}$ still holds with the refreshed attributes else it returns to idle. Thus this machine satisfies formula φ_1 (backward-looking stale-safety) when the attributes are not stale, and it satisfies formula φ_2 (forward-looking stale-safety) when the attributes are stale. Note that it is also possible to construct machines that will satisfy the other properties we discussed earlier.

5. RELATED WORK AND CONCLUSION

Attribute staleness is inherent to any distributed system and can result in serious access violations. In this paper, we proposed stale-safe security properties using the group-based Secure Information Sharing problem as an example. We formalized four stale-safe properties of varying strengths using Linear Temporal Logic. This formalization not only enabled us to precisely state the properties but also allows systems to be formally verified, for example, using techniques such as model checking. We discussed an FSM construction of the Trusted Reference Monitor resident in access machines that satisfies the weak stale-safe property. We believe that these properties can be generalized to any distributed applications that uses attribute-based access control with minor extensions/modifications if any.

To the best of our knowledge, this is the first effort towards formalizing the notion of stale-safety in attribute-based access control. The work of Lee et al [10, 11] is the closest to ours that we have seen in the literature, but focuses exclusively on the use of attribute certificates, called credentials, for assertion of attribute values. Lee et al focus on the need to obtain fresh information about the revocation status of credentials to avoid staleness. Our formalism is based on the notion of a "refresh time," that is the time when an attribute value was known to be accurate. We believe the notion of refresh time is central to formulation of stale-safe properties. Because Lee et al admit only attribute certificates as carriers of attribute information there is no notion of refresh time in their framework.

A technical report on a more exhaustive version of this work along with results from model checking the TRM can be found in [8]. There has been substantial work on the application of model checking to verify security properties. Formal specification and verification techniques and tools, such as model checking, have been increasingly leveraged to verify security properties of access control systems [3, 7, 17, 2, 4, 13, 16, 18].

Our future work is along three exciting paths. First, model checking the complete g-SIS system is a major future work. This is a complex problem which is composed of multiple FSMs for TRM, CC and GA. All these machines need to handle various operations such as membership management of subjects and objects, provisioning group credentials, multiple group memberships, etc. Second, the stale-safe properties need to be extended to accommodate multiple CCs, multiple groups and multiple access machines. We believe studying and formalizing these extensions is valuable to build stale-safe large-scale distributed systems. Third, we believe the notion of stale-safety as described in this paper can be extended to Attribute-based Access Control (ABAC) in general. This way, any system that uses ABAC to specify policy can be made stale-safe.

6. ACKNOWLEDGMENTS

Ravi Sandhu is partially supported by NSF grant IIS-0814027, AFOSR grant FA9550-06-01-0045, a research superiority grant from the State of Texas Emerging Technology Fund and a grant from Intel Corporation. Jianwei Niu is partially supported by NSF grant IIS-0814027 and THECB ARP grant 010115-0037-2007. William Winsborough is partially supported by NSF grants CCR–0325951, CCF-0524010 and CNS-0716750 and THECB ARP grant 010115-0037-2007. Ram Krishnan is partially supported by a grant from Intel Corporation.

7. REFERENCES

[1] TCG specification architecture overview. *http://www.trustedcomputinggroup.org.*

[2] A. K. Babdara, E. C. Lupu, and A. Russo. Using event calculus to formalize policy specification and analysis. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, pages 26–39, 2003.

[3] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tshchantz. Verification and change-impact analysis of access-control policies. In *ICSE*, pages 196–205. ACM Press, 2005.

[4] D. Gilliam, J. Powell, and M. Bishop. Application of lightweight formal methods to software security. In *Proceedings of the 14th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2005.

[5] J. Goguen and J. Meseguer. Security policies and security models. *IEEE Symposium on Security and Privacy*, 12, 1982.

[6] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Comm. of the ACM*, pages 461–471, August 1976.

[7] S. Jha and T. Reps. Model checking SPKI/SDSI. volume 12, pages 317–353, 2004.

[8] R. Krishnan, J. Niu, R. Sandhu, and W. Winsborough. Stale-safe security properties for group-based secure information sharing. *Technical report CS-TR-2008-012, Department of Computer Science, University of Texas, San Antonio*, 2008.

[9] R. Krishnan, R. Sandhu, and K. Ranganathan. PEI models towards scalable, usable and high-assurance information sharing. *Proc. of the 12th ACM Symposium on Access Control Models and Technologies*, pages 145–150, 2007.

[10] A. Lee, K. Minami, and M. Winslett. Lightweight consistency enforcement schemes for distributed proofs with hidden subtrees. *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 101–110, 2007.

[11] A. Lee and M. Winslett. Safety and consistency in policy-based authorization systems. *Proceedings of the 13th ACM conference on computer and communications security*, pages 124–133, 2006.

[12] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, Heidelberg, Germany, 1992.

[13] M. J. May, C. A. Gunter, and I. Lee. Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, 2006.

[14] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, pages 309–329, September 2003.

[15] R. Sandhu. Lattice-based access control models. *IEEE Computer*, pages 9–19, November 1993.

[16] A. Schaad, V. Lotz, and K. Sohr. A model checking approach to analysing organizational controls in a loan origination process. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT06)*, pages 139–149, 2006.

[17] A. P. Sistla and M. Zhou. Analysis of dynamic policies. In *Proceedings of Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, pages 233–262, 2006.

[18] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *Proceedings of the 8th Information Security Conference*, volume 3650 of *LNCS*, pages 446–460. Springer-Verlag, 2005.