Context-Aware Identity Delegation

Naveed Ahmed and Christian D. Jensen

Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Lyngby
Denmark
nahm@kth.se,cdj@imm.dtu.dk

Abstract. In emerging ubiquitous computing, related nomadic users often perform similar tasks and share the same computing infrastructure. This means that security of the shared resources is of prime importance. Frequent delegation of tasks among users must be anticipated as most nomadic environments are hectic and very dynamic. A delegation mechanism with a slightly complicated user interface will not only reduce the productivity but also provide nomadic users with a strong motivation to circumvent the mechanism itself.
Delegation in access control domain is not practical for the most of nomadic users due to its complicated and complex structure. Identity delegation at authentication level provides improved usability, which reduces the risk of circumventing the delegation mechanism; at the same time, however, identity delegation violates the principle of least privileges. We use contextual information of a delegatee to mitigate this violation, which helps to achieve a higher level of practical security in nomadic environments.
Keywords: Context-aware, Identity Delegation, Nomadic User, Practical Security

1 Introduction

The pervasive use of computing technology in our life has caused a shift in how we interact with computers. Earlier, in the age of mainframes and desktops, a user had to physically move to a computer to access information or computing resources. When technology allowed manufacturing of lightweight devices along with a well connected wireless communication infrastructure, the paradigm of mobile computing emerged. This allows people to move freely in an environment along with their computing devices, so information and (modest) computing resources are ubiquitously available. More recently, we have started embedding computing devices into work environments, which makes the nomadic use of computing possible. Nomadic users move freely in their work environment and use shared devices that are embedded in the environment on which they can invoke their unique sessions. A typical example is a hospital, where computers and equipment are shared among doctors and nurses. These people access patients' health-care data from different terminals in wards using their unique identities.

In collaborative work environments, nomadic users often need to work in each other's place, for instance, in the nomadic environment of hospitals, a senior doctor may need to delegate his duties to another doctor when a patient in a critical condition arrives in the emergency unit. In this case, the senior doctor is primarily concerned with two things: to *whom* should he delegate and *which* of the patients and wards need to be taken care of. In the hectic environment of current hospitals, doctors generally prefer to simply share their passwords and sessions for the delegation [5], rather than doing so by a complicated procedure that may offer a higher level of security. For these nomadic users, it does not make sense to engage them beyond their simple concerns of Whom and Which for the purpose of delegation; otherwise, they would tend to circumvent the delegation mechanism due to its usability issues.

From a user's perspective, delegation is only concerned with the two simple questions: to *whom* one should delegate, the delegatee; and in *which* context the delegatee should be authorized to work on one's behalf. From a security perspective, however, it is also accompanied with fine level details of individual authorizations, in order to follow the principle of least privileges [3, 15]. A pure security perspective represents one extreme, where a delegation mechanism has fine granularity and thus each privilege is transferred individually, however, this fine control also makes it cumbersome and error prone. Considering the usability perspective, delegation becomes more and more coarse grained, where many privileges are transferred simultaneously, at the expense of the principle of least privileges, but its simplicity could make it more user-friendly and thus more likely to be used in practice.

Delegation is necessarily regulated by a security policy, which defines the privileges of each user and their authority to delegate them. We call this the *Designed Security Level*. The designed security level is not an actual assessment of the security of the system, but captures the intentions of the designers through their specification of policies and choice of mechanisms. However, the security that we achieve in practice is often less than or at most equal to the designed level, we call this the *Effective Security Level* because security mechanisms may contain vulnerabilities and users may decide to circumvent the mechanism, e.g., by sharing passwords. Most systems are designed to achieve a specific level of security, which is perhaps mandated by legislation, so the designed security level tends to remain constant over time. The effective security level, however, tends to degrade with time as vulnerabilities are discovered – but not necessarily patched, advances in technology makes new attacks possible, new techniques in cryptanalysis are discovered or simply that the vigilance of users erode and shared passwords proliferate in the system. This increasing *security gap* between the designed and effective security levels is illustrated in Figure 1.

It is important to note, however, that the curve shown in the figure is for illustrative purposes and exact shape of the curve is not relevant; the actual shape depends on the type of the system, the configuration and operation of the system, and parameters of the computational environment in which the system is deployed. For example, new security mechanisms, for which the users are not

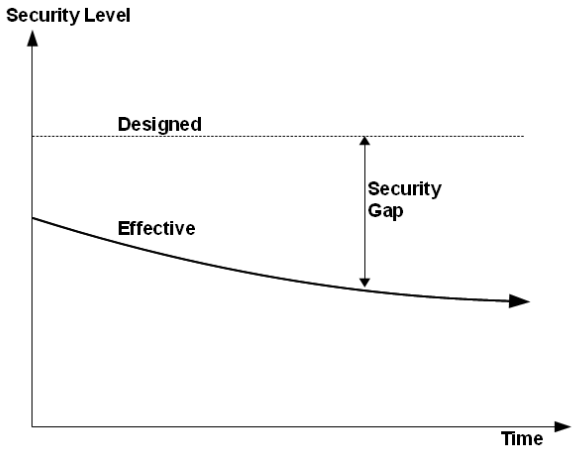

Fig. 1. Difference between Effective and Designed Security Levels

yet trained or aware of its purpose, the curve of effective security could have the positive slopes in the start. In any case, the poor usability in delegation provides a motivation to users for deceiving the access control mechanism by sharing their authentication tokens, which also contributes to the widening security gap.

The security gap could be reduced by improving the usability of delegation. The identity delegation at authentication level has a considerable usability advantage over classic delegation models, but on the expense of violating the principle of least privileges [2]. Still, the identity delegation could provide more effective security than classic delegation mechanisms for many nomadic environments where there are pre-established trust relationships among users, such as in a group of doctors who work together in a hospital. We extend the identity delegation [2] to include contextual information, which further increase the usability while at the same time provides more justifications to use the identity delegation despite of its obvious violation of the principle of least privileges. The use of context limits the unnecessary spread of authorizations. We have implemented the proposed mechanism in form of a prototype.

In the next section, we describe part of the state of the art in delegation. Section 3 presents our delegation mechanism in detail. In Section 4, we describe the details of the prototype. Section 5 provides evaluation of the mechanism. In the final section, we present some conclusions.

2 Related Work

The term *delegation* has many definitions, for instance Abadi et al. [1], Barka and Sandhu [6], Gasser and McDermott [9], Gladney [10], etc. In the following, we adhere to the definition of Zhang et al. [19], i.e., we consider authentication

as a process in which one active entity in a system transfers its authority to another active entity in order for that entity to carry out a job on behalf of the first entity. In this paper, we only focus on human-to-human delegation and consider a *delegator* who delegates his authorizations, while one who receives these authorizations is referred as a *delegatee*. [3]

Human-to-human delegation can be at access control level or at authentication level. At the access control level, Gasser and McDermott [9] propose a delegation technique with cryptographic assurances of the revocation if the system is subsequently compromised. Varadharajan et al. [16] consider delegation in distributed systems as the problem of verification for a delegatee using signature-based scheme with certain assumptions of trust relationships among the system entities. Zhang et al. [19] propose a rule-based specification language and a rule-based framework for the role-based multi-step delegation. The delegation at permission level, although with very limited options, can also be accomplished in the UNIX access control model. A formal framework for delegation under Role Based Access Control(RBAC) is proposed by Barka [6]. Bertino et al. [7] presents the notion of context in form of Temporal- RBAC that supports periodic role enabling and the notion of time in form of temporal dependencies among permissions. Covington et al. [8] extends the context of model beyond time by incorporating constraints of location and system status. Kumar et al. [4] presents a formal context-sensitive RBAC model, which enables complex security policies using the context information.

At user's authentication level, an identity delegation essentially assigns the system identifier of a delegator to a delegatee. A framework at this level is invented by Mercredi and Frey [14], whose primary objective is to enable a person to sign in on the behalf of another person. Ahmed and Jensen [2] propose a simpler architecture of identity delegation, which is derived from the usability factors of delegation in nomadic environments. Using *sudo* or *su* commands of UNIX are implementation dependent alternatives.

An identity delegation transfers all authorizations of a user most of which might not be required for the delegated job. Nevertheless, an identity delegation can be more user friendly as all the necessary privileges are delegated in one fell swoop along with the identity, while at access control level, it is usually hard for a user to figure out the exact privileges necessary for performing a particular job. Li and Wang [13] address this problem in access control domain, by bundling the authorizations of a job in form of a unit. On the other hand, revocation of delegation is equally important and might be non-trivial [17].

In nomadic use of computing, where users frequently delegate to one another, complexity of delegation (and its revocation) results in poor usability and provides a strong incentive for users to bypass the secure way of delegation. As indicated by Bardram, et al. [5], users start sharing their authentication tokens for mutual collaboration. On the other hand, the existing user-friendly identity delegation techniques [14, 2] do not consider the context and thus cause a wider spread of authorities, which restricts their use to a limited number of nomadic environments.

3 The Mechanism for Context-Aware Identity Delegation

In the following discussion, the term *Validated identity* refers to the identity that an authentication mechanism concludes with help of one or more authentication techniques. Similarly, the so-called authenticated identity provided to an access control mechanism is referred as *Effective identity.*

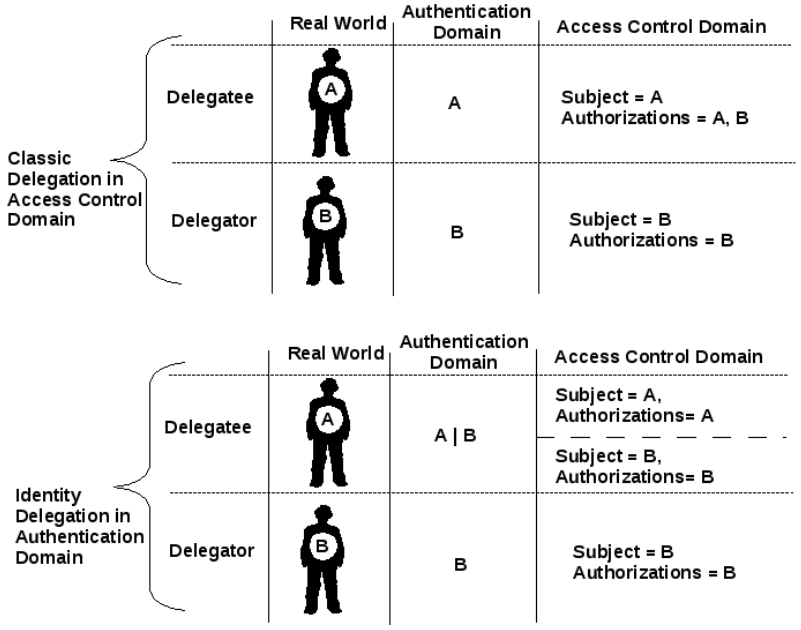

Fig. 2. Difference Between the Two Approaches for Delegation

Figure 2 exemplifies the two approaches for delegation: the delegation in access control domain; and the delegation in authentication domain. In the former case, a validated identity and the corresponding effective identity are assumed to be the same and therefore delegation is managed in the access control domain in terms of permissions and roles; as shown, a delegatee A is recognized as A in the access control domain and has the authorizations of A as well as of B. Delegation in authentication domain is achieved by distinguishing the notions of validated identity and effective identity. As shown in the lower part of the figure, a delegatee A, who is authenticated as A, may assume the effective identity of either A or B. This allows A to use the authorizations of either A or B, depending on the choice of effective identity.

In this paper, we extend the basic definition of identity delegation [2] to include context information. "A context-aware identity delegation at authentication level is a process in which an authentication mechanism provides an

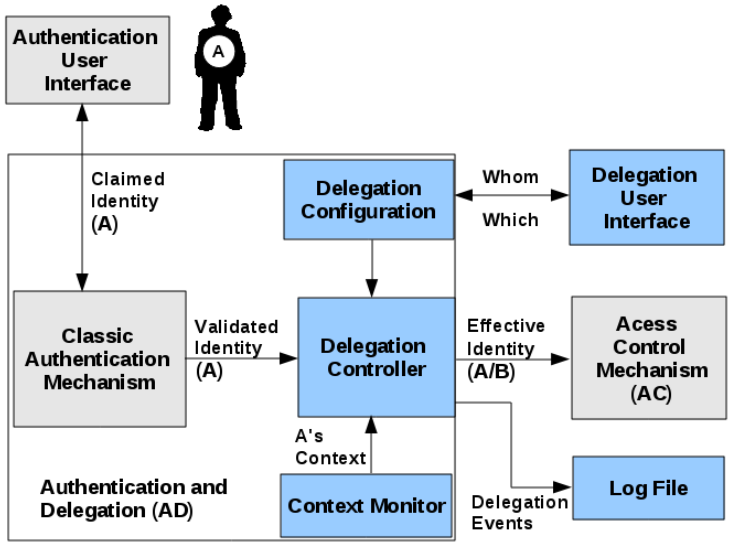

Fig. 3. Context-Aware Identity Delegation at Authentication Level

effective identity that is different from the validated identity of a user provided the following conditions are true."
1. *Whom*: The owner of the effective identity (delegator) has previously delegated his identity to owner of the validated identity (delegatee).
2. *Which*: The current context of authentication for the delegatee is same as previously specified by the delegator.

From the delegator's perspective these two conditions are two simple decisions: *whom* to delegate; under *which* context to delegate. In this paper we only consider time of day and the network address of a computer as the context information, but the definition of context is not limited to these two factors.

The architecture of proposed mechanism is shown in Figure 3. In our architecture, Authentication module is augmented by the three other modules and is shown as Authentication and Delegation(AD) part in Figure 3. These three modules are Delegation Configuration, Delegation Controller and Context Monitor. The Delegation Configuration module contains the delegation policies that are currently active in the system. In the prototype, Delegation Configuration consists of a database, which contains *Whom* and *Which* specifications of the all delegations in a system. The Context Monitor captures and distributes the context information of a delegatee; in our case, it supplies the current time and the network address of the computer on which the delegatee is validated by an Authentication module. The Delegation Controller performs the translation from a validated identity to an effective identity depending on the delegation

policy and input from the Context Monitor; it also records all delegation events in a local log file.

When a delegatee approaches a system, the claimed identity of the delegatee is validated by a classic authentication mechanism, in the usual way. After this, the module Delegation Controller maps the validated identity to an effective user identity based on the input from Delegation Configuration. Now, this effective identity is supplied to the access control mechanism. As defined earlier, this effective identity could either be of the delegatee or of the delegator, depending on the inputs from Delegation Configuration and Context Monitor. The figure shows that a user A can be recognized as a user B in the access control mechanism if B has previously delegated his identity to A and the current context of the system for A is same as previously specified by B.

The configuration in Delegation Configuration can be considered as the delegation policy of a system and is in the form of mappings between validated identity and effective identity. Each mapping relation is associated with some context constraints under which the relation holds. Additionally there is also a list of preferences for each system user that specifies the currently active identity among multiple available identities. Thus, depending on the mapping, context and the preferences a particular identity is provided to the access control mechanism.

We provide a simple user interface in the user's session to specify the delegation policy for new delegations and for changing existing policies. We refer to it as Delegation User Interface in the figure. This interface should only require from a user to decide to *whom* and under *which* context to delegate, so that the user does not worry about specific permissions, roles, security policy or security administrator.

The log file provides a level of accountability in the system. Since our mechanism is at authentication level, we cannot restrict unnecessary delegated authorizations as they are part of the access control domain. This drawback is inherited from the very nature of identity delegation and is justified by the log file and the assumption of mutual trust among co-workers and colleagues [2]. In our mechanism we restrict the propagation of unnecessary authorizations by limiting the delegation in particular context specified by the delegator. In this way we increase the security of a system by limiting the violation of the principle of least privileges.

4 The Prototype

The prototype consists of a personal computer running Debian Linux and is the extended version of our original prototype for identity delegation [2]. We have augmented the classic login based authentication mechanism with RFID based authentication. In the idle state when no user is present on the system, multiple sessions of users are suspended and the computer display is locked. When a user approaches, the relevant session is invoked instantly if RFID badge is validated. This invoked session could be the session of the user or it can be a delegated

session. This selection depends on the preference in the delegation configuration, which can easily be changed by a simple command. The interaction of the

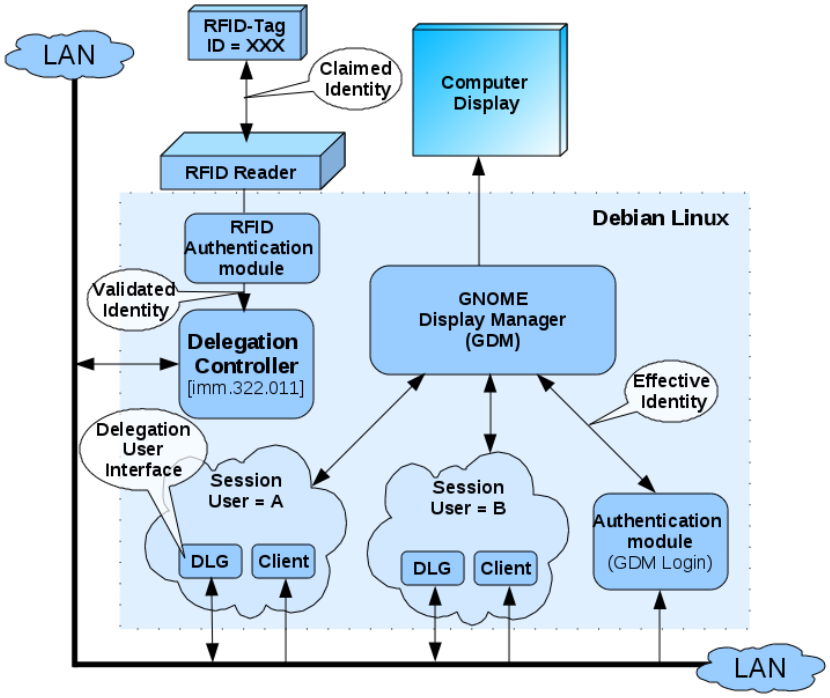

Fig. 4. Architecture of the Prototype

different modules in the prototype is shown in Figure 4. We have not shown Delegation Configuration and Log file. Context Monitor, which in our case consists of the system timer and the network address, is also not shown. RFID Authentication module grants or denies an authentication request using the identification which it receives from the RFID reader. Delegation Controller is launched with the privileges of "root" at start-up, just before GNOME display manager(GDM) is started. In the GNOME based desktop, a single GDM manages the sessions (X-Servers) of all users. The Delegation Controller in the prototype interacts with GDM, by sending commands in a customized format through Clients that run in each of the user's sessions. GDM provides an abstraction for multiple sessions and also invokes the password based login program (GDMlogin). However, GDMlogin program is only a front end for interacting with a user, the actual authentication is achieved by a pluggable authentication module associated with GDM. We use network sockets for the interprocess communication, in order to simplify its porting on distributed nomadic networks.

In typical use, a user carrying a valid RFID tag enters into the interrogation field of an RFID reader. The RFID reader reports the identification to the RFID Authentication module, which checks this identification in the local database to find a match for a system user. If a match is found, the Delegation Controller maps the matched identification (validated identity) to a new identification (effective identity) based on the status of the delegation configuration and the current context. As shown in the figure, the context in our case is limited to a pseudo location *imm.322.11* and the time of day. After deciding the effective identity, Delegation Controller activates the corresponding session.

For the delegation purpose, we have developed a console program in user's space, which represents the user interface for delegation. For example, if a user Bob wants to delegate his session to a user Alice, for the office hours on a computer in Room 11 of Building 322, he issues the following simple command.

```
>dlg set Alice @imm.322.011 [0800-1600]
```

Now, if Alice is in the right context, i.e. Room 11 of Building 322 between 8 am and 4 pm, then she can invoke B's session in one of the following way. Firstly, if she does not have her local account on the machine then the B's session can be invoked automatically when she walks up to the terminal. Secondly, if her local account exists on the machine and she has previously specified her preference for B's session on the machine then again the session can be invoked automatically. Lastly, if her local account exists but she has not previously used B'session then she need to issue following simple command from the terminal.

```
>dlg switch Bob
```

And finally when Bob want to revoke the delegation, he uses the following command.

```
>dlg reset Alice
```

Besides this basic example, there are many more user-friendly commands for different aspects of delegation. The use of location and time in the example is only one possible form of context and in principle, other parameters can be included, though we have not implemented them in the prototype. All the commands of delegation in the prototype share a common motive, which is the simplicity of user interface as we aim to make the delegation process to be more user friendly so that nomadic users actually use it rather than to circumvent it. The syntax of the delegation program is as follow.

```
                Syntax::= dlg <action> <parameters>
              <action>::= set | reset | switch | reset-rec |
                          get | reset-all | NULL
       set:<parameters>::= <user_login_name> <context>
             <context>::= @<location> [<time_period>]
      reset:<parameter>::= <user_login_name>
reset-rec:<parameters>::= NULL
```

```
      get:<parameters> ::= NULL
reset-all:<parameters> ::= NULL
```

Invoking *dlg* without any argument shows help for the command. Otherwise, *set* is for setting an outbound delegation, *reset* is for revoking an existing outbound delegation, *switch* is to switch among delegated sessions, *reset-rec* is to revoke all inbound delegated sessions, *get* is for getting information from existing delegation policy, *reset-all* is for resetting all outbound delegations in one go.

5 Comparison and a Formal Model

Similar to any delegation mechanism developed in the authentication domain, our mechanism violates the principle of least privileges [3], but we believe that this is justified in nomadic environments, where there are pre-established trust relationships among users [2]. The use of context limits the proliferation of unnecessary authorizations, which makes our mechanism more secure than a simple identity delegation at authentication level.

If the delegation is fine grained, then there is always a risk of under-delegation, which effectively results in a denial of service; this is very undesirable in most cases. Similarly, revocation becomes non-trivial in some fine grained delegation mechanisms designed at access control level. In our mechanism, delegation and revocation are just a matter of issuing a simple command.

The classic access control models of UNIX and Unix-like systems may achieve delegation by changing file permissions and group memberships, but this also implies that one should be either the owner of the resource or must have super user privileges; A user is not necessarily able to delegate all privileges which he is authorized to use. In our mechanism this problem is removed and delegation is a user level decision.

In order to compare our mechanism with classic delegation techniques we use a formal logic [18, 1, 12]. We start by introducing a few notions from this logic. When we write **A says S**, it implies that the entity A utters the statement S and believes in its truthfulness. We write $\mathbf{A} \Rightarrow \mathbf{B}$ to represent a kind of trust relationship in a way that whenever A makes a statement, B makes it too. Since the statement, made by a particular entity of the system, implies that the entity believes in it, thus the statement $\mathbf{A} \Rightarrow \mathbf{B}$ represents the trust of B on A. We write **A as R** to represent the entity A in a particular role R and consequently **A as R** has a subset of all authorizations that A normally possesses. For instance, $\mathbf{A'token}$ represents the role of A that possesses the correct authentication token of A. Similarly, $\mathbf{A'context}$ represents a particular context of A. We write (**A** | **B**) **says S** to represent that A is quoting a statement, S, of B. We write (**A for B**) **says S** to represent a quoted statement S and in addition, A is authorized to make such quoted statements on the behalf of B.

Each entity has a set of authorizations, which it can transfer or delegate to other entities in the system using following axioms.

If **A says** (**B** ⇒ **A**) then (**B** ⇒ **A**) for the system ... (1)
(This axiom represents the transfer of authorities from A to B.)

If **A says** ((**B** | **A**) ⇒ (**B for A**))
then ((**B** | **A**) ⇒ (**B for A**)) for the system ... (2)
(This axiom is the classic identity delegation in access control domain as the identity of A is retained)

Now, let us consider Figure 3 and divide it in three system level entities: the user, A; the authentication and the delegation primitives enclosed in the rectangle of the figure, AD; and the access control mechanism, AC. We also assume that the only role of the authentication module AD is to provide the identity of a user. The default trust relationships between these entities are represented by the following statements.

AD ⇒ **AC** ... (3)
(The access control mechanism trusts the authentication mechanism and thus the access control mechanism believes in the identity provided by the authentication mechanism.)

A as A′token ⇒ **AD** ... (4a)
and **B as B′token** ⇒ **AD** ... (4b)
(The authentication mechanism trusts a user with the role of possessing the correct authentication credentials in the form of a valid token that corresponds to the user in the authentication database.)

The statements (3) and (4) are enough to complete the trust chain from a user to the access control mechanism in order to invoke a user's session. For example a user A with A'token can invoke A'session. In our context aware identity delegation mechanism, a user A may delegate his identity to another user B by sending a request to the authentication mechanism in the following form.

(**A as A′token**) **says**
(((**B as B′token**)**and**(**B as B′Context**))|(**A as A′token**))
⇒ **A as A′token**) ... (5)
(An authenticated user A tells the authentication mechanism that if the authenticated user B, in a 'Context', quotes a statement of the authenticated user A then it must be considered as the actual statement from the authenticated user A. The user A specifies Context, in the form of environmental constraints, such as time, location, etc.)

Due to the formal logic axioms, the effect of the statement (5) is the following new trust relationship, which is in fact the context-aware identity delegation.

$$(((\mathbf{B\ as\ B'token}) \mathbf{and} (\mathbf{B\ as\ B'Context})) | (\mathbf{A\ as\ A'token}))$$
$$\Rightarrow \mathbf{A\ as\ A'token} \quad \ldots (6)$$

The statement (6) represents that if a user B, in a specific context, requests for the session of user A, then this request is considered to be equivalent to the request directly made by the user A, which was in form of the statement (4a). Note that the identity delegation represented by (6) is different from the transfer of authorities in (1) and the classic delegation of (2). In (1) the identity of the delegator A is completely lost and in (2) the identity of delegator is present in all requests made by the delegatee. In our mechanism of (6) the identity of the delegator A is only present in the authentication and thus the access control mechanism does not distinguish between a delegator and a delegatee.

6 Conclusions

In nomadic environments, the usability of delegation is a decisive parameter in determining the effective level of system security. People in such environments mostly delegate to their co-workers and colleagues in whom they trust. This fact along with use of context and the logging of delegation events can be used to justify the security of our mechanism in nomadic environments. Since we provide a user friendly interface for delegation, thus many more people will use this secure way of delegation rather than attempt to deceive the access control mechanism for delegating their authorities, which helps to improve the effective level of security.

Despite the obvious security improvement, however, we have not conducted an evaluation to determine exact quantitative values and therefore our claims are only justified by intuitive arguments of the paper; nevertheless, finding good metrics for the accurate measure of security is an open field of research [11].

References

1. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(4):706–734, September 1993.
2. Naveed Ahmed and Christian D. Jensen. A mechanism for identity delegation at authentication level. In *The 14th Nordic Conference in Secure IT Systems, NordSec-2009*, Oslo, Norway, October 2009.
3. Mehran Ahsant. *On-demand Restricted Delegation: A Framework for Dynamic, Context-Aware, Least-Privilege Delegation in Grids*. PhD thesis, Kungliga Tekniska Högskolan, 2009.
4. Girish Chafle Arun Kumar, Neeran Karnik. Context sensitivity in role-based access contro. *ACM SIGOPS Operating Systems Review*, 36(3):53–66, July 2002.
5. Jakob Bardram, Thomas K., and Christina Nielsen. Mobility in health care - reporting on our initial observations and pilot study. Technical Report CfPC 2003-PB-52, Center for Pervasive Computing, 2003.

6. Ezedin S. Barka. *Framework for Role-Based Delegation Models*. PhD thesis, George Mason University, 2002.
7. Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233, August 2001.
8. Michael J. Covington, Srividhya Srinivasan Wende Long, Anind K. Dev., Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20, Chantilly, Virginia, United States, 2001.
9. M. Gasser and E. McDermott. An architecture for practical delegation a distributed system. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, U.S.A., 1990.
10. H.M. Gladney. Access control for large collections. *ACM Transactions on Information Systems*, 15(2):154–194, April 1997.
11. Dieter Gollmann. *Computer Security 2e.* John Wiley and Sons, 2005.
12. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265 – 310, November 1992.
13. Min Li and Hua Wang. ABDM: An extended flexible delegation model in RBAC. In *Proceedings of the 8th IEEE International Conference on Computer and Information Technology*, pages 390–395, Sydney, Australia, July 2008.
14. Dwayne Mercredi and Rod Frey. User login delegation. United States Patent Application Publication, US 2004/0015702 A1, January 2004.
15. J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of IEEE*, 63(9):1278–1308, September 1975.
16. Vijay Varadharajan, Philip Allen, and Stewart Black. An analysis of the proxy problem in distributed systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, U.S.A., 1991.
17. Hua Wang and Jinli Cao. Delegating revocations and authorizations. *Springer Berlin / Heidelberg Lecture Notes in Computer Science(LNCS)*, 4928:294–305, February 2008.
18. Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the taos operating system. *ACM Transactions on Computer Systems (TOCS)*, 12(1):3–32, February 1994.
19. Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):404–441, August 2003.