

AMETAS

Good Migrations.

AMETAS White Paper Series

by Klaus Herrmann
and Michael Zapf

On Privileges

July 2000

Johann Wolfgang Goethe-Universität Frankfurt/Main, Germany
Department of Computer Science

www.ametas.de

Authorized Personnel Only.

Agents wandering from node to node are a potential security problem. Especially when you allow agents coming from other people to be executed on your places you want, of course, make sure that they cannot tamper with your data, install Trojan Horses and the like. Whether an agent may do some action or be denied to do it is determined by a set of so-called privileges and permissions. We will give you a brief overview on how to work with these important components of security enforcement.

Your Identity, Please.

Place Users may be identified by means of the so-called *Place User ID*. This ID is unique, containing the creation location as well as a timestamp, and therefore suitable to easily address any running Place User. However, as each Place User has a different Place User ID, there is no easy way to determine who was the initiator.

As agents may be used by different people, the kind of agent cannot be used to determine if an action may be performed or not. You, for instance, want to have some more rights to do even critical actions on your own place, but other people should not. If the assignment of permissions and privileges were related to the kind of agent we could not re-use the code by different people. Therefore, the *actual user* must be taken into account when deciding to allow an operation.

AMETAS defines *identities* which may be uniquely assigned to people utilizing Place Users. The identity is immutable and contains a public and an encrypted private key. Using this identity it is possible to sign an agent or to check its signature. As long as the passphrase that protects the private key is not compromised, the author and the user of a Place User may reliably be determined. Whenever a Place User is started, the identity of the actual user is determined and associated to this agent instance for the rest of its lifetime. If there is no valid identity the Place User cannot be started; if a remote place does not recognize the identity it can be configured to reject the migrating agent. Identities can be uniquely referenced using the *identity ID*.

Complete identities contain both public and private keys while *public identities* only contain the public key. Public identities are required to check the validity of a signature or to encrypt data for the owner of the identity. The private key in the complete identity is encrypted with a passphrase only known to the owner. This key is used for signing and decrypting. As the identity file could be subject to attacks it is suggested to use the complete identity file only at the local place while the public identity may be deliberately distributed.

Authors

Place Users are stored in containers — Signed Place User (SPU) containers — that are signed twice:

- The code must be signed by at least one of its authors. This signature is immutable and always accompanies the SPU.
- The code container, the identity ID of the user and a time stamp are signed by the user as a whole.

The user signature is created by the place automatically. However, as signatures in general require the secret key of the signer, only Place Users started at the place the

user is connected to can be signed. If the user requires all places to sign his agent on his behalf he can instruct the places to send the passphrase along with the agent.

Places can be configured to only allow Place Users to be started if there is a valid author signature. The signature must come from an identity known to the place checking the signature. This allows to keep out untrusted code; furthermore, you may even allow unknown users to send agents to your place as long as you know who wrote the agents. The signature algorithm is MD5/RSA which guarantees that the code cannot be modified unnoticedly.

Users

As you can imagine, knowing who uses the agent is somehow more important to know than who wrote the agent. Some users may be more privileged than others. The agent may then be granted more privileges and perform restricted tasks (see below).

A physical user may have several identities. This can be interpreted as playing different roles in the system. One identity can be used for everyday usage while another one is used for administrative purposes. It should be noted that if you plan to use a temporary place you *need* an administrative identity to be able to shut down your own place. However, you should withstand the temptation to always use this identity when working with applications because if something goes wrong, your Place Users can do considerable damage to your system with administrative privileges!

Identities may represent any entity in the system but should be primarily used for human users. Each agent that is started by a user's Place User is associated with his identity. This identity is the base for the domain access policy to determine the set of valid privileges.

Certification Authorities

An identity can be easily created using the tools that come with the AMETAS distribution. But that also means that it is even more important to provide a way to find out if a given identity is a valid identity that actually belongs to that person it claims to belong to. AMETAS defines certification authorities (CAs) that are used to sign identities. Every identity requires the approval of at least one CA. Identities without a certificate are considered invalid.

A place administrator must decide whether to trust a CA or not. This must be decided very carefully: When you trust a CA, you will automatically accept every identity signed by this CA. Every CA has an identity of its own that must be imported at every place that is to accept this CA. These identities may in turn be signed by another CA, creating a *chain of certification*.

Levels of Security: Privileges and Permissions

AMETAS provides agents with an abstraction of the underlying system. For them, the world actually consists of other agents, places, messages, and events. Agents do not have access to files, to sockets, or to any other system resource.

AMETAS agents are implemented in Java. However, Java *does* know how to work with these resources. If we do not want agents to exploit these capabilities of Java, we need to restrict them using a *security manager*. AMETAS implements a security manager that defines restrictions for special kinds of Place Users and allows and denies access to components of one Place User.

The AMETAS Security Manager can only monitor accesses on system resources like files, network connections, threads, graphic displays and so on. This is achieved using callback method calls inside the library functions of the Java system which call the corresponding methods inside the security manager. However, guarding the access to resources is not the only base to grant or deny actions. For example, we want an agent to be useable only by a special group of people. Another scenario could be that an agent can only send messages to a service if it belongs to a special group. These restrictions cannot be covered by the Java security system.

For these aspects we introduce the concept of *privileges* and *permissions*. While permissions are required to grant access to low-level resources, privileges may be considered as groups that identities can be members of. Furthermore, privileges can contain a set of permissions that are required for the usage of some Place Users.

Permissions

Permissions are needed to gain access to low-level resources. Inside the place a special data structure is associated to each running Place User, containing the valid permissions. Whenever such a Place User performs an action that calls the security manager (e.g. opening a file), the data structure is checked for the existence of the corresponding permission. Permissions may also be checked for actions concerning the place functions. For instance, there are permissions that grant administrative access to the place or allow their owner to receive all kinds of events.

Having a permission does not necessarily entail the granting of the respective access. Agents, for example, must not open files at any time even if the system was configured to grant the file permission. If a file access is unavoidable, a service must be used. The security manager decides to deny or grant access depending on the kind of Place User.

Permissions are modeled as classes that can be instantiated. But permissions cannot simply be collected as needed. Any Place User may try to instantiate permission objects but it does not have a chance to put them into the aforementioned data structure. This effectively prevents Place Users to gain permissions they are not entitled to have. In order to get these permissions, Place Users and users must have a privilege that contains these permissions.

Privileges

Permissions are normally grouped in privilege objects. That means if a special permission is required for a Place User to perform some action, this permission must be contained in a privilege that the Place User may be granted. The assignment of a privilege depends on two requirements:

- The owner of the Place User is granted this privilege after log-in at the place. The place allows the owner to join this group when its domain access policy is defined accordingly.
- The Place User requests the privilege.

It is possible for Place Users to restrict the set of actually valid privileges. This, for instance, allows agents to be used by administrators without gaining administrative privileges they actually do not need. If anything weird happens, the user can prove that

his agent actually never had the necessary privilege or permission to be responsible for the action.

Setting Privileges and Permissions

Privileges and permissions are defined by the *domain access policy* of each place. This policy can only be defined and modified by the administrator of the respective place. Permissions are generally predefined; most of them can be parametrized like the *file permission*, taking e.g. the filename as an argument. There is also a *generic permission* that can be used when the other predefined permissions do not apply.

There are five predefined privilege definitions:

- *PUBLIC*: does not include any permission.
- *ADMIN*: includes the *Admin permission* required to do administrative tasks.
- *NETACC*: includes a generic *socket permission* to allow incoming and outgoing network connections.
- *RESFILEACC*: includes some *file permissions* to allow access to the file system.
- *ANYEVENT*: allows to receive all events at a place.

The *PUBLIC* privilege is sufficient for utilizing most Place Users. Agents having the *PUBLIC* privilege can migrate and send messages to other Place Users. User adapters may display a graphical interface without the need for special permissions. However, if special restricted tasks should be performed, some of the other privileges could be required.

There are two sets of privileges defined for each Place User: The *required privileges* are those that must have been granted to the caller before activating the Place User, and the *restricted privileges* are the maximum set of privileges that this Place User will ever receive. The author of the Place User is responsible for defining these two sets. The set of actually valid privileges (also implying the set of permissions) is created at run-time as an intersection of the caller's privileges with the restricted privileges.

Services play a slightly modified role. They are considered to be started by the place itself which has all privileges. Therefore, the actual privileges are always equal to the restricted privileges. As for the required privileges, the service manager only forwards those messages to the service objects if they originate from clients that were granted these privileges.

Defining New Privileges

Imagine you design an application where you want only special users to utilize your agents. Apart from that, your application does not need any further permissions. In this scenario you would simply define a privilege *MYPRIV*¹ without included permissions. This privilege can be seen as a group that identities may join or leave. In order to allow identities to join this group, you must define this privilege at all places that your agent may visit. Furthermore, make sure that the identity of the user is also assigned to this group by the *domain access policy* at all these places. Note that the identity may only join this group

¹You may of course choose any other label.

- if it is entitled to join the group according to the *domain access policy*
- and the Place User contains this group in its set of *restricted privileges*.

Figure 1 shows the relationship between the various components.

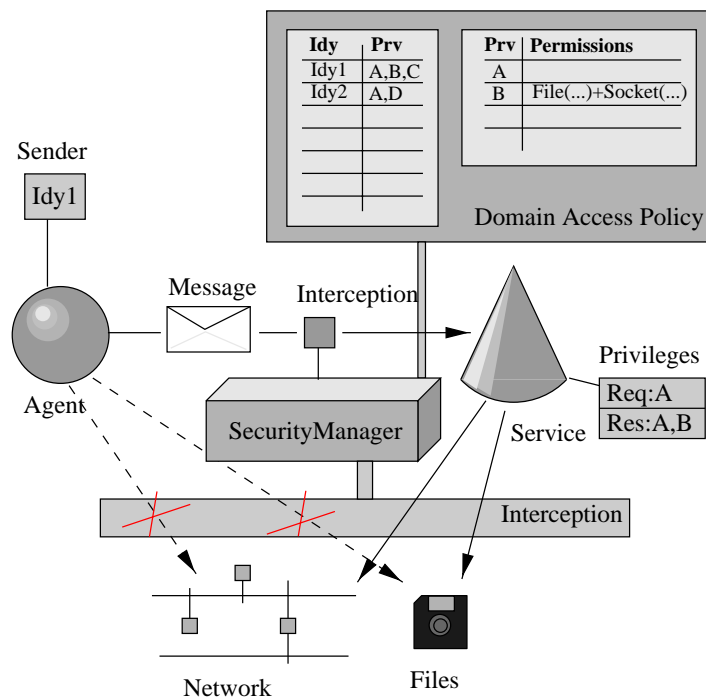


Figure 1: Controlled access to resources.

As you can see, agents must never access system resources like files or the network. Services may access them but usually require some specific permission for this. The privilege called *B* is defined to have the necessary privileges. As the user — identified by *Idy1* — is entitled to join group *A*, his agents will be granted access to services requiring the privilege *A*. Services may be executed with the full set of privileges defined by the set of restricted privileges which contains privilege *B*. This privilege entitles the service to do the intended actions.

If your application requires extended permissions you should at first consider carefully which object will actually perform the restricted access. As already mentioned, services have the advantage that they always get the maximum set of privileges listed as the restricted privileges, regardless of what privileges the caller has (as long as they suffice for the set of required privileges). So we would advise you to let your services do these “dangerous” jobs. If you want to use extended permissions for agents or user adapters you should be aware of the fact that certain permissions are *never* allowed for agents². Moreover, you have to propagate this assignment of privileges to all of your places.

²For example, agents must never use socket connections or open any files.

Assigning Privileges to Identities

The addition and removal of identities can only be controlled in two ways:

- utilizing the security configuration tool *SecAdmin* that is part of the AMETAS distribution;
- utilizing a Place User under an identity that is entitled to get the *Admin permission*.

Each place has got a special passphrase that must be provided at startup. This passphrase is also required when the user wants to configure the security system using *SecAdmin*. However, when several places should be configured in a similar way, it is recommended to employ a mobile agent carrying the ADMIN privilege. Of course, the identity under which you started the agent must be defined at every place to be managed and must be entitled to get the ADMIN privilege.

The AMETAS concept of managing identities, privileges, and permissions is quite complex compared to classical security systems, but security management is a very critical task in systems with mobile code. Novice users normally find it difficult to get used to the procedures how to assign privileges to identities. Therefore, we sum up the basic notions to provide a concise overview:

1. Every human user who wants to work with the agent system needs at least one identity. This identity contains a key pair that allows the user to authenticate himself and to create signatures. A user can have different identities that may be assigned different sets of privileges.
2. The identity itself does not imply any privileges or permissions.
3. Permissions are grouped in privileges. Permissions are granted by granting privileges that contain them.
4. You may only get a privilege if there is a corresponding entry for your identity in the domain access policy of the place.
5. Privilege assignments are only valid for one place. These assignments must be repeatedly done on every place. The assignments are persistent.
6. Your agent never gets more privileges than your identity is defined to get.
7. Your agent may dispense with some privileges you are entitled to get.
8. The identity which will be used for all Place Users started by you is set at the moment of connecting with an interface like AMAI or AMSI.
9. If you own a place, define two identities: One for the administration of your place, the other for everyday tasks.
10. If you are a place administrator, remember it is *you* and *only you* who determines if an agent gets a privilege or not.

As a simple user you must make sure that all places your agent visits either have your identity or allow guests, and if special privileges are required you also have to check with the respective system administrators whether the domain access policies are adequately defined. It is obvious that if you are an administrator you have a much higher responsibility regarding the system security than when you are a simple user.