Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DS2OS – Deliverable D
Running Environment Design

Giovanni Schmid, Alessandra Rossi

RT-ICAR-NA-2010-05 **Dicembre 2010**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

DS2OS – Deliverable D
Running Environment Design

Giovanni Schmid[1], Alessandra Rossi

Rapporto Tecnico N.:
RT-ICAR-NA-2010-05

Data:
Dicembre 2010

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, Via P. Castellino 111, 80131 Napoli

DS2OS Deliverable D
Running Environment Design

Giovanni Schmid, Alessandra Rossi

December 27, 2010

1 Introduction

As we saw in Deliverable A, Dynamic Delegation (DD) is a new access control model which involves users, services, and applications. DD introduces two new kinds of principal's profiles - the *guest* and the *sponsor* profiles - with the following special features:

1. A guest g for an host system H does not follow a standard registration process on H; that is, it is not subjected to the accounting procedure performed by a system administrator to create a suitable standard account on H. Conversely, g is granted by a user s of H with a special privilege, called sponsor, the use (of some of) the credentials and resources that s is entitled to get on H;

2. g's capabilities on H are defined by s as a subset of its ones through a special *attribute certificate* (AC). Depending on what is stated in the above AC, then:

   (a) g is allowed to operate on H with a set of *security attributes* $A_H(g)$ such that

   $$A_H(g) = A \sqcap A_H(s) \ ,$$

   where $A_H(s)$ is s's set of security attributes at runtime, A are the security attributes specified for g by s in the AC, and $\sqcap$ denotes the operation of *x-intersection* (see Deliverables B and C);

   (b) g is eventually entitled to log in H. In such case, g has its own home directory and its own shell, but operates in a sandboxed environment in which it can performs safely its actions under the constraints imposed by its sponsor;

   (c) s can eventually share some of its own directories and files with g;

3. g does not make use of the standard accounting coordinates (username and password) in order to perform its own authentication to H. Actually, g's identification and authentication phases on H are not managed in the usual way through the standard repositories `passwd` and `shadow` (or their remote counterparts). Indeed, its *identifiers* $I_H(g)$ on H are defined at runtime by:

   $$I_H(g) = \{UID(g) = \phi(g, UID(s)), GID(g) = GIDG\} \ ,$$

where $GIDG$ is a special GID value reserved to guest profiles, and ϕ is a suitable function which allows to uniquely determine g given s and $UID(g)$ (see Deliverables A and B)[1];

4. g's authentication on H is realized through a *public-key certificate* (PKC) issued by s for g. Depending on what is stated in such PKC, g could be allowed to use its profile on H only for specified periods of time and/or as a consequence of suitable conditions, as defined by its sponsor s according to the security policy of H;

5. a single user on H can act as sponsor s for different guests g_j, and each g_j must have its own separate authorization profile and its own separate running environment;

6. a single principal p can be the guest of different sponsors s_j on the same host H. In this case, p has different profiles on H, one for each s_j, and these profiles must be kept separate and not-joinable.

This deliverable discusses some of the implementation issues of dynamic delegation at the operating system layer. From the previous listing of DD features, it should be clear that the main difficultes arose in providing a suitable running environment for the guest profile. Indeed, the sponsor profile is just a standad user with one more special right, that of being a sponsor. Thus, w.r.t. our development platform OpenSolaris, implementing the sponsor profile basically consists in the specification of a new *rights profile* (see Deliverable B). Conversely, a guest profile requires:

R1 suitable identification, authentication and authorization workflows in order to accomplish items 2, 3 and 4;

R2 the provision of a filesystem configuration and an environment set-up supporting features 2.1, 5 and 6;

R3 the introduction of some more fine-grained *privileges* (see Deliverable B) that enable guest users to affect processes and resources of their sponsors.

Section 2 describes our implementation approaches to satisfy requirement **R2**. Section 3 is devoted instead to detail about the implementation of **R1**. As we told in Deliverable A, our prototipal implementation of dynamic delegation is limited to the entry service SSH. Since authentication and authorization for entry services in OpenSolaris are managed through the *pluggable authentication module* (PAM) framework (see Deliverable B), **R1** turns out in a design which provides a suitable SHH login workflow thanks to ad-hoc PAM modules and its configuration. Finally, in Section 4 we detail about the new rights profile required to realize the sponsor profile, and some new privileges as prescribed in **R3**.

2 Guest's environment set-up

Guests access to an operating system, which has its own security mechanisms and policies. In order to keep such system in a safe and reliable state, it is necessary to enforce for those non standard profiles the security controls and auditing already provided for standar users, plus some more controls and insulation guaranteeing that guests are constrained to the limits

[1]Note that g denotes a principal, not a username on H; indeed, g could not have a username on H. Actually, our prototipal implementation provides usernames to guests performing a login, and this process is managed through the `passwd` database as usual. This is because a full implementation of requirement (3) would require some more programming efforts which were outside the resources of our project.

imposed by their respective sponsors, and that the same principal cannot perfom a privilege escalation by joining two or more of its guest profiles on the system. On the other hand, however, guests are granted their profiles in order to interact and collaborate effectively and efficiently with their respective sponsors. Therefore, guests require both an isolated and integrated running environment, such that they can use system resources, services and applications as specified by their sponsor but with no losses in the overall system security.

At the OS layer, insulation is generally achieved by way of *virtualization*. As we saw in Deliverable B, OpenSolaris offers a set of native virtualization technologies that culminate in *zones* and *containers*. However, zones and containers are very useful if the purpose is to consolidate a number of operating environments on a single piece of hardware, but they are not a good choice in our case, since they are far more complex and resource consuming than required for setting up suitable running environments for guest profiles.

Rather, in our case it suffices to rely on insulation at the file system level, as provided by the *loopback filesystem* technique. A **Loopback file system (LOFS)** [7] creates a new virtual file system so that one can access files and directories provided by another (real) filesystem using an alternative pathname. This is realized through links to the needed files and directories, and access control to those resource can be enforced indipendently of their original permissions thanks to suitable permissions on such links.

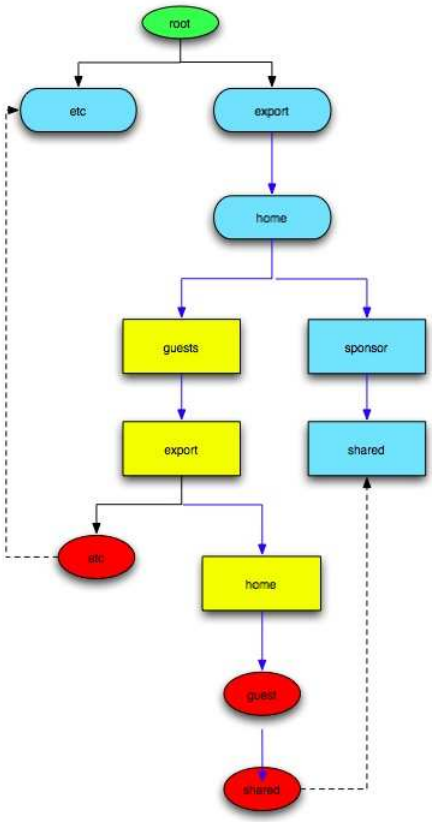

Figure 1: ZFS set-up for the guest profile home directory.

In the following, we will suppose that our system has *ZFS* (see Deliverable B) as root filesystem. This is the default for OpenSolaris from build 88. Relying on ZFS greatly simplifies our implementation, since the volume management features of ZFS allow for the creation of single-user filesystems easily and efficiently. Moreover, ZFS disposes of more advanced ACLs than the legacy UNIX filesystem (UFS) [4].

As seen in Section 1, the guest's environment must support features 2.1, 5 and 6. In order to accomplish this, we did the following choices in our prototipal implementation:

Requirement Set 1 (Filesystem Set-up Requirements) *:*

FS1 *A principal g acting as a guest on a system H, and allowed to login in H, has a username and a home directory which varies with its sponsor; that is, the existence of multiple sponsors in H for g results in multiple g's usernames and home directories in H. Home directories are rooted at* `/export/home/guests`[2] *through the LOFS technique, in such a way that all the guest profiles logged in H see* `/export/home/guests` *as the root of the global filesystem.*

FS2 *A home directory for g is a ZFS whose properties are specified by its sponsor s, according to the security policies on H. For example, if s is subjected to a* quota *mechanism, then the same must occur for g. The filename for such home directory equals the username for g on H, and is desumed from g's distinguished name and s's username. For example, if the distinguished name of g in its PKC is* `bob@gmail.com`*, and its sponsor username on H is* `alice`*, then the filename of the home directory is* `bob_alice` *(for a precise formulation of this rule see Deliverable C).*

FS3 *System directories containing common resources and configurations for users (e.g.,* `/bin`*,* `/etc`*,* `/dev`*) are accessible read-only to g as LOFSes with the same pathnames under* `/export/home/guests`*.*

FS4 *Filesystem resources that have to be shared between s and g, are exported as LOFSes for g, or issued as sigle files with suitable ACLs, under s's control.*

The above are somewhat arbitrary choices, and we could have chosen a quite different set-up. For example, conversely than in **FS1** and **FS2**, we could have provided a unique home directory for g on H, regardless of its sponsor. In that case, such directory could have a conventional pathname such as `/export/home/guest`n, where the number n is chosen by the system depending on the overall number of guest profiles in H, as the smallest positive integer available.
Both alternatives have some inconvenience, and - ultimately - it is a matter of taste.

3 PAM extension design

The PAM framework [6, 8] was extensively studied in Deliverable B. Here it could be useful just to remember that PAM is based on a series of *services modules*, and that its configuration is specified through the `pam.conf` file. Services modules implement one or more of the following services: *authentication*, *account management*, *session management* and *password management*, whereas `pam.conf` is used to control the behavior of PAM-aware applications: the stacks listed therein indicate the right flow of operations, in term of services modules involved, for each application.

[2] Differently than in other Unix-like systems, home directories default path in OpenSolaris is `/export/home/` because of the *automounter* service.

In PAM jargon, the term "authentication" encompasses all the three phases of identification, authentication and authorization decribed in Deliverable B. Thus, our design results essentially in new PAM authentication (auth) modules, although - actually - we needed to modify all the phases of the SSH login process in order to accomplish the DD requirements. Notice that the DD-aware PAM modules which we developed for the SSH daemon can be effectively used for any DD-aware system entry service. We will refer in the sequel to a such login workflow as *DD-aware login*.

The DD-aware login flow for SSH is depicted on the left side of Figure 2. The process is realized through the PAM auth type modules `pam_X509_authE`, `pam_X509_authO`, `pam_X509_cred`, and the PAM session type module `pam_X509_session`. These modules are designed in such a way they can be applied to any initiator (user) U; that is, both in case of a principal with a guest profile and a principal with a standard user profile on the target host. The only assumption is that U uses a X.509-compliant distinguished name (DN) and a X.509-compliant public key as identifier and authentication token, respectively. The distinction among standard and guest profiles is based upon the DN given in input by U:

- if the host portion of SUS's DN equals the hostname of the target host, then U is considered a standard user who wishes to perform a X.509 public key based authentication;
- otherwise, U is considered a guest, and her successful login requires a standard user S acting as sponsor.

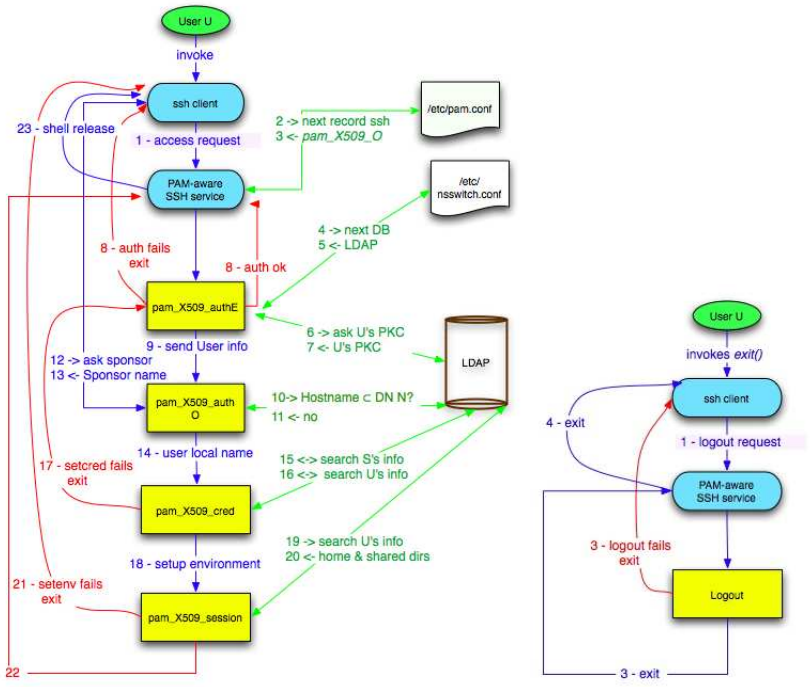

Figure 2: The DD-aware login and logout workflows for the SSH service.

Assuming the PAM stack for `sshd` as indicated in Table 1, the DD login flow is as follows:

- U requires to log in the target system through a suitable SSH client, which sends to `pam_X509_authE` U's X.509-style coordinates for authentication;
- `pam_X509_authE`:
  - checks if the authentication coordinates submitted by U are valid by looking at the available repositories through the name switch framework;
  - authenticates U through a challenge-response protocol;
  - returns to the SSH service the outcome (`'failure'` or `'success'`) of this authentication phase;
- if the previous outcome is `'failure'`, then the service checks for the legacy authentication module `pam_unix`, following the processing flow coded therein; otherwise, it loads `pam_X509_authO` with U's DN as input;
- `pam_X509_authO`:
  - checks if the host portion of the DN given in input equals the hostname;
  - if the outcome of the above test is `'yes'`, then the module outputs U's username as desumed by her DN; otherwise, it asks for the sponsor S username, returning both usernames as output;
- the SSH service loads `pam_X509_cred` with the ouput of `pam_X509_authO` as input;
- `pam_X509_cred`:
  - searches for the credential information related to the usernames got as input by looking at the available repositories through the name switch framework;
  - computes U's current set of credentials thanks to the *x-union* and *x-intersection* operations described in Deliverable C;
  - returns the above set to the SSH service;
- the control is passed again to the SSH service which loads `pam_X509_session`;
- Depending of the previous processing, `pam_X509_session` manages the opening, setting-up and closing of U's session as a standard user or a guest profile.

Service	Module Type	Control flag	Module Path
sshd	auth	required	pam_X509_authE.so
sshd	auth	required	pam_unix.so.1
sshd	auth	required	pam_X509_authO.so
sshd	auth	requisite	pam_X509_cred.so
sshd	session	required	pam_X509_session.so

Table 1: PAM stack for the DD-aware SSH service.

When the interpreter program (shell) loaded as consequence of the `pam_X509_session` set-up terminates its execution in response to U's request of logging out through the SSH client interface, then the SSH service perfoms some clean-up operations (e.g. the unmounting of U's home directory) and then returns control to the client. The logout workflow for SSH is depicted on the left side of Figure 2.

4 Expanding RBAC in OpenSolaris

Dynamic delegation (DD) requires some RBAC configurations, and some more OpenSolaris *security attributes* to be effectively implemented. More specifically, one ad-hoc *rights profile* is required for the sponsor profile, and some new *privileges* enabling guest users to affect processes and resources of their sponsors would be useful. These and other related issues will be discussed in the following two sections. The reader is referred to Deliverable B and [9] for an overview of the above and other RBAC-related concepts in OpenSolaris which are of relevance in these sections.

4.1 The sponsor profile

In DD, the sponsor profile - although not a primary administrator - must have the ability to create guest profiles, which in turn requires the ability of delegating sponsor's security attributes to other profiles, and - in case a guest is allowed to log in the system, the ability to create a standard user account with such attributes. Among its preconfigured *rights profiles*, OpenSolaris provides `Rights Delegation` and `User Management`.
`Rights Delegation` is a profile to delegate the ability to assign rights to users and roles; it is composed by the following authorizations:

- `solaris.role.delegate` gives the right to assign to others any owned roles;
- `solaris.profmgr.delegate` gives the right to assign to others any owned rights;
- `solaris.grant` gives the right to grant to others any owned solaris authorizations.

`User Management` is instead a profile to manage users, groups and home directories, which composes of the following authorizations:

- `solaris.profmgr.read` gives the ability to view rights;
- `solaris.admin.usermgr.write` gives the ability to add users;
- `solaris.admin.usermgr.read` gives the ability to view users;
- `solaris.admin.usermgr.manage` gives the ability to manage users.

and the following commands with security attributes:

- `/usr/sbin/groupadd:uid=0`
- `/usr/sbin/groupdel:uid=0`
- `/usr/sbin/groupmod:uid=0`
- `/usr/sbin/roleadd:uid=0`
- `/usr/sbin/roledel:uid=0`
- `/usr/sbin/rolemod:uid=0`
- `/usr/sbin/useradd:uid=0`
- `/usr/sbin/userdel:uid=0`
- `/usr/sbin/usermod:uid=0`

- `/usr/sbin/grpck:euid=0`
- `/usr/sbin/pwck:euid=0`
- `/usr/share/setup-tool-backends/scripts/users-conf:uid=0`

It is useful to stress here that not all the security attributes listed above are strictly necessary to implement the sponsor profile. For example, the set of authorizations `solaris.admin.usermgr.` × are typically required to operate through the *Solaris management Console*, a graphical front-end for administrative tasks.

In order to get a sponsor profile as prescribed by DD requirements, it suffices to consider the `Rights Delegation` profile plus the commands with security attributes:

- `/usr/sbin/useradd:uid=0`
- `/usr/sbin/userdel:uid=0`
- `/usr/sbin/usermod:uid=0`

However, in order to grant the sponsor to assign to others any owned privileges, we need to introduce a new ad hoc authorization, denoted as `solaris.priv.delegate`. Adding to the OpenSolaris distribution authorizations or profiles to be used by a command or through the SMF framework [3]) are quite simple tasks, since they do not require kernel programming: [1] and [2] are useful on-line references on these subjects.

5 Guest-aware privileges

The fine-grain approach implemented in OpenSolaris at the kernel level through process privileges has evolved over time with the introduction of new privileges, and the *Fine Grained Access Policy* (FGAP) project [5] is committed to provide a mechanism to "sandbox" applications by first removing basic privileges and then granting them on a case-by-case basis. At the time of writing the *basic privileges set*, that is the set of privileges granted by default to all standard users in an OpenSolaris environment, composes of `proc_exec`, `proc_fork`, `net_access`, `file_link_any`, `proc_info and proc_session` privileges.
Expanding in some way this basic set is valuable for an effective implementation of DD, since a guets profile must have less rights than ordinary users.

The `proc_exec` and `proc_fork` privileges allow, respectively, a process to call `exec` and `fork`, (`fork1`, `vfork`).routines. These privileges are essential for all basic operations, and for any kind of user or application profile in a system, since they are required for the execution of programs.

The `net_access` privilege allows a process to open a TCP or UDP network endpoint. Thus, by omitting this privilege in the basic set for a user we can prevent such a user from creating sockets and network end points, isolating it from the network. This is not an option for a guest profile, since guests are - by construction - remote users.

Instead, for the remaining three privileges

- `file_link_any` : allow a process to create hardlinks to files owned by a UID different from the process's effective UID;
- `proc_info` : allow a process to examine the status of processes other than those to which it can send signals;

- `proc_session` : allow a process to send signals or trace processes outside its session;

it would be advisable to introduce guest-aware counterparts in such a way the operations granted by them are restricted only to file and processes owned by the related sponsor.

Likewise, it would be advisable to add to the existing `file_dac_read/write/execute/search` privileges, that allow a process to read/write/execute/search a file or directory whose permission bits or ACL would otherwise disallow the process' read/write/execute/search permission, their guest-aware versions allowing the above operations just on files owned by the sponsor.

Once new privileges are coded in the kernel, they have to be added to the system by:

- adding privileges definitions in `priv_name.h`;
- updating the `priv_name` privilege description database;
- adding policies that check for the presence of the new privileges. The files that must be updated in such respect are:
  - `common/sys/policy.h` for the functions declarations.
  - `common/os/policy.c` for the functions definitions.
- changing all the needed applications in order to modify their behaviour according to the new privileges;
- changing the filesystem configuration files in order to support the new privileges.

References

[1] Opensolaris ARC Best Practices: Adding Authorizations
//hub.opensolaris.org/bin/view/Community+Group+arc/rbac-auths

[2] Opensolaris ARC Best Practices: Building RBAC Rights Profiles
http://hub.opensolaris.org/bin/view/Community+Group+arc/rbac-profiles

[3] OpenSolaris Community Group SMF
http://hub.opensolaris.org/bin/view/Community+Group+smf/

[4] OpenSolaris Community Group ZFS
http://hub.opensolaris.org/bin/view/Community+Group+zfs/

[5] OpenSolaris Project: Fine Grained Access Policy
(FGAP) http://hub.opensolaris.org/bin/view/Project+fgap/

[6] Samar V., C. Lai: Making Login Services Independent of Authentication Technologies, *Proceedings of the SunSoft Developers Conference* (1996)

[7] Solaris 10 Reference Manual Collection Section 7: Device and Network Interfaces : LOFS
http://docs.sun.com/app/docs/doc/817-3947/6mjgnrlbg?l=en&a=view&q=LOFS

[8] Solaris 10 System Administrator Collection: Solaris Security Services Chapter 17 Using PAM
http://docs.sun.com/app/docs/doc/816-4557/pam-1?l=all&a=view

[9] Solaris 10 System Administrator Collection: Solaris Security Services: Part III Roles, Rights Profiles, and Privileges
http://docs.sun.com/app/docs/doc/816-4557/prbactm-1?l=en&a=view