AUTHENTICATION / SECURITY SERVICES IN THE ROOT FRAMEWORK

G. Ganis, CERN[#], Geneva, Switzerland
gerardo.ganis@cern.ch

Abstract

The security services available in the ROOT framework [1] are described, focusing in particular of client / server authentication.

INTRODUCTION

Security is a major concern of any modern software system. ROOT users are confronted to the problem when they need to access a server daemon running on a remote host. Currently there are three basic types of daemons accessed from a ROOT interactive session: file servers (rootd, xrootd, httpd, rfiod), the proofd daemon used to start the master and worker servers in the Parallel ROOT Facility [2], and interactive sessions setup as servers (by means of the TServerSocket class). Two aspects of security are relevant here: *authentication*, where the client proves her identity to the server; and *confidentiality*, either of generic data or of authentication-related data.

Recently the security services of ROOT have been restructured with the aim of making the system more complete and flexible to cover most of the needs ROOT users will have using the coming clusters and computing facilities.

This paper is organized as follows. In the next section we review the available authentication protocols and the features of the framework in which they are embedded. In the following section we will discuss the additional requirements from PROOF. The future plans are then presented, followed by a summary.

AUTHENTICATION SERVICES

Authentication Protocols

The needs of ROOT users in terms of client / server authentication vary significantly: from the case where read-only access is given to a file, requiring weak - or even no – authentication, to the case where a daemon has to be run with super-user privileges, therefore requiring more protection. It is therefore important to offer a wide spectra of protocols to cover as much as possible the potential situations. The protocols presently available in ROOT are:

- A fast identification protocol, based on the matching of the user and group IDs, intended for use in intrinsically secure situations.
- Password-based protocols, whose strength depends on the underlying implementation;
- Two of the mostly used network authentication protocols, Kerberos [3] and GSI [4].

Password-based protocols

The password-based protocols typically have the advantage of requiring none or very little setup, and therefore are very convenient in many circumstances. Users are given the possibility to define a ROOT-specific password, which may give additional protection by limiting the damage of password disclosure.

Server may also test the system password files (e.g. `/etc/passwd`) but this typically requires special privileges, since most of the systems use shadow passwords to protect the password section of the system files. Workarounds to allow regular privileged daemons to access the system password files requires delegation to a third daemon with the right privileges; if AFS is available ROOT servers can be configured to use the AFS API's to perform this task. Another possibility is to use the `sshd` daemon: the server generates some information uniquely identifying the session and the client, and requires the latter to prove her identity by `scp`-ing a file with related secret information in the area she wants to log on the remote machine; since the file to be `scp`-ied is session dependent, it cannot be re-used for reply attacks; `scp` is invoked on the command line, so the user can use her `SSH` key files.

Basic password-based protocols are susceptible of network-based dictionary attacks. Immunity to this kind of attacks (strong authentication) requires the use of asymmetric keys. Protocols based on these techniques are available on the market [5].

ROOT supports one of these protocols, the *Secure Remote Password* protocol [6], SRP, based on asymmetric key exchange technology; it requires the user to logon once on the remote to create the secrets related to the password; the advantage of this method is that the password never goes over the network, but it is only used to create credential information to negotiate with the server. It also provides a session key for subsequent encryption.

Additional features available for the password-based protocols include support for {host ,user} equivalence via the usual `/etc/hosts.equiv` and `$HOME/.rhosts` files, and for auto-login via the standard `$HOME/.netrc` and an extension of this file (`$HOME/.rootnetrc`) provided to serve also SRP.

Kerberos V

The Kerberos network authentication protocol was one of

[#]Funded by Bundesministerium für Bildung und Forschung, Berlin, Germany

the first attempts to solve the problem of user / service mutual authentication in consistent way, using strong cryptography and a centralized trusted third party (the Kerberos server) to deliver tickets. Though Kerberos has some weak points [7], it still provides a good solution to the authentication problem and it is still widely used [8]. ROOT applications are kerberized using the standard Kerberos 5 API's `krb5_sendauth` and `krb5_recvauth`. Clients need a principal, servers use the "`host`" service. ROOT exploits the possibility to forward the credentials for later use and to use the file `.k5login` to provide user-equivalence, allowing to disentangle the target username from the principal name.

GSI

Finally, ROOT may also be configured to use Globus Security Infrastructure deployed on current grids. This requires the availability of the security bundles of the Globus Tool Kit [4]. As for Kerberos, the authentication handshake is delegated to the appropriate `gss_assist` API functions, which are used to acquire the credentials from a valid proxy or host certificate and to initiate or accept a security context over the open connection. Servers started with regular user privileges can use the user proxy certificate to authenticate clients. The location and list of user, host certificates and gridmap files are configurable. Globus delegation features are exploited by servers to export the client credentials to a shared memory segment for later use.

Server Access Control

A key feature of server security is the possibility to allow or deny access to a given request, on the base of host from where it comes from and/or the user account (and possibly the service) where it is directed. Server administrators can specify their preferred access rules in a special configuration file `system.rootdaemonrc`; the default version of this file is created under `$ROOTSYS/etc` while configuring the ROOT installation. The default list includes the available password- and network-based protocols, enabled by default for any hosts, user, service. The configuration file allows to specify rules that apply to subsets of machines either by accepting wild cards in *FQDN*s or by accepting sub-domain specification; it is possible to specify different rules for access to file servers, `proofd`'s or interactive servers; finally, it is possible to deny access to a given user, or to require stronger authentication protocols for a given user. Examples of typical directives are shown here:

```
# Allow use of ROOT-specific password;
# else require SSH protocol
# Access to 'baduser' is denied
default usrpwd:-baduser ssh:-baduser
# Fast ID from the local domain
pc*.local.domain uidgid
```

```
# PROOF cluster in the local Kerberos
# realm or grid
*:proofd krb5 globus
```

Client / Server Protocol Negotiation

ROOT supports a simple mechanism for client/server negotiation. The mechanism uses the list of protocols available to the client; the default list is created by configure and stored in a special file `$ROOTSYS/etc/system.rootauthrc`; the list also specifies the order of preference in which the protocols should be chosen. As an example, the default entry for a ROOT distribution compiled with Kerberos 5 support is:

```
# Created at configure
default list usrpwd ssh krb5 uidgid
```

The entry indicates that when a client contacts a server she will first attempt to use the regular password-based protocol, i.e. her special ROOT-password; if the server policy accepts this protocol it will continue the negotiation handshake; otherwise it will send back to the client the list of protocols (if any) that it may accept from this particular client (based on the related entry in the `rootdaemonrc` file previously discussed). There is no overhead if the first attempt is successful; the server takes control on the protocol choice from the second step in the iteration.
The client can modify the default information by specifying her own preferences in a private file `$HOME/.rootauthrc` , host-, user- and service- base. The file allows also to change many other defaults, like the location and names of the globus certificate directory and user certificate and key files.

ROOT *token*

ROOT can be configured in such a way that a unique and secret tag is assigned to a successful authentication attempt; this *token* can be used by the client to open an additional connection to ROOT servers running on the same remote host without going through the full handshaking of the authentication process: this may considerably speed-up the process of user identification, especially for SSH and GSI protocols. On the client side, a token correspond to an instance of the `TSecContext` class: its validity is configurable, but can never extend beyond the duration of the interactive session. On the server side, token information is kept in dedicated tab files.

Securing sensitive information

The transmission of sensitive information, like password or tokens, over the network is secured using PKI technology and symmetric ciphers.

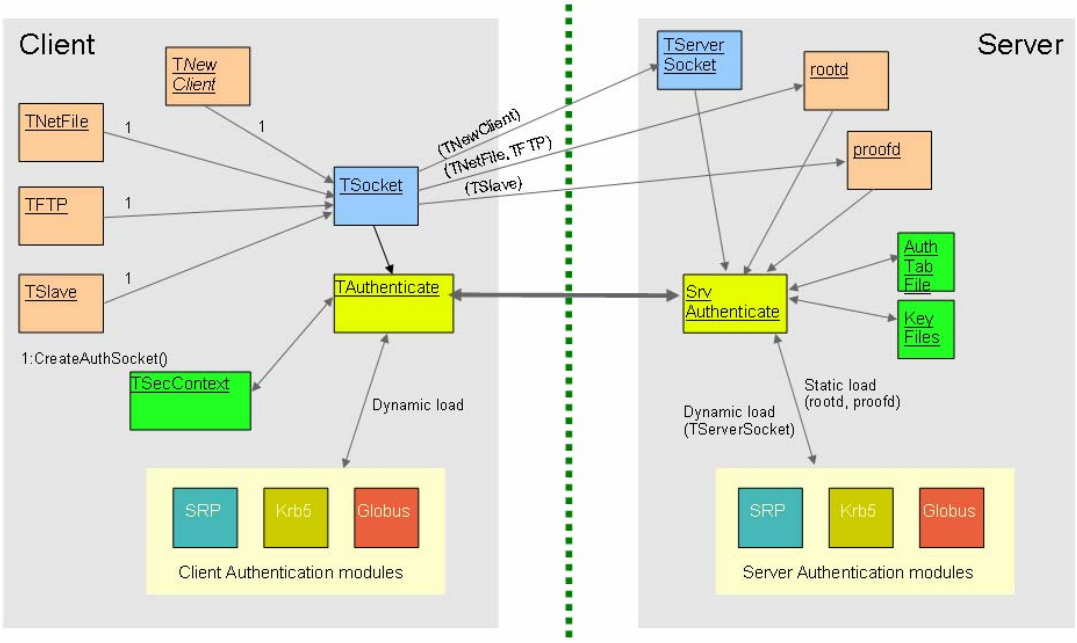

Figure 1 Structure of the Authentication code in ROOT

At startup, the servers generate an asymmetric key pair, which is kept in memory and whose duration validity is the server lifetime. When a client comes in, the server public key is used to protect the exchange of a symmetric session cipher to be used to encrypt the sensitive information during the authentication handshake. The cipher also expires at the end of the interactive session.

Code structure

Figure 1 shows the structure of the authentication code as presently implemented in ROOT. When a remote connection is needed, the client class instantiates a `TSocket` class which is responsible to contact the appropriate service on the remote side; if authentication is required, both sides delegate the handshaking to dedicated code, responsible, if required, to load the relevant plug-ins and to create and store the token.

PROOF

The Parallel ROOT facility, PROOF [2], is a virtual machine aimed at giving the physicists the possibility to analyze much larger sets of data on a shorter time scales, making use of the inherent parallelism in event data.

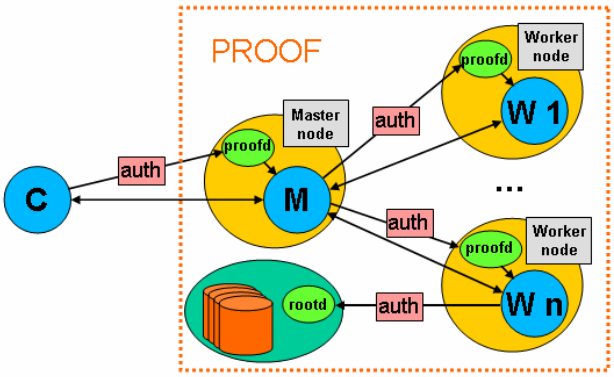

Figure 2 Startup of a PROOF virtual machine

PROOF consists of a 3-tier architecture (Figure 2): the ROOT client session, the PROOF master server and the PROOF worker servers. During startup the user connects from his ROOT session to a master server on a remote cluster and the master server in-turn creates worker servers on all the nodes in the cluster. At each step authentication may be required. While the first step (client-to-Master) is basically the same as for any client/server session, in the PROOF system there is the additional complication of the Master authentication *vis-à-vis* of the workers and, potentially, of the workers authentication *vis-à-vis* of data servers.

If the PROOF cluster is closed to the outside world, with only the Master allowing incoming connections, strong authentication is required only between Client and Master: a weak protocol (fast ID, host equivalence) or even no authentication will be sufficient for the remaining steps.

However, for clusters spread over different sites, strong authentication may be required at all stages, making a mechanism for delegation or credential forwarding unavoidable. Such mechanism are naturally provided by network authentication protocols like Kerberos and GSI. Servers can extract from the client ticket or proxy the appropriate information to act as *clients* vis-à-vis of the next-stage servers, on behalf of the mandating Client. As mentioned before, this feature is exploited in ROOT for both Kerberos and GSI.

For password-based protocols, credential forwarding means *sending the password to the server*. This is implemented in PROOF: the password may be sent, encrypted, at the end of the Master or Worker startup. However, for SRP, since it formally spoils one of the basic features of the protocol ("*passwords never leave the client machine*"), this feature is left as an option to the user (note, however, that the packets containing an encrypted SRP password cannot be used for a reply attack).

Encrypted-password forwarding requires that the password is the same on all nodes requiring authentication: if this is not the case, a manual setup is needed, using the auto-login facilities mentioned above.

FUTURE PLANS

Future plans focus mainly on the consolidation of the authentication code structure and the support for secure connections.

Consolidation of the underlying structure. The present implementation suffers from being the evolution of a much less ambitious design whose purpose aimed to provide basic authentication services to two standalone light daemons, `rootd` and `proofd`; this brought in the code an asymmetry between client and server sides, which has been reduced but not eliminated; a cleaner design, simplifying and consolidating the underlying structure, is envisaged. For this purpose, the integration of `XrdSec`, the authentication framework of the `xrootd` daemon [9], recently been included in ROOT, is under evaluation. `XrdSec` is a general purpose authentication framework, where protocols can be plugged as shared libraries; it provides server access control, authorization services, and a built-in simple protocol-negotiation mechanism. At present, only available are plug-ins for Kerberos 4 and 5. Several plug-ins are under development, including modules for generic password-based authentication and GSI.

Secure connections. It is foreseen to support data confidentiality in two ways. The first uses the cipher exchanged during authentication to encrypt any subsequent exchange; this requires optimized use of buffers for encryption and transmission, to minimize overload. The second provides a `TSocket` class, `TSecureSocket`, initiating a standard SSL connection with any server supporting SSL; this would allow `TWebFile` to open files on web servers running the `https` protocol. Prototypes for both implementations are under test.

SUMMARY

The security services of the ROOT system have been recently reviewed and augmented. The improved security modules provide a flexible authentication framework supporting a complete set of protocols, and mechanisms for credential forwarding in the PROOF system. Future plans include the consolidation of the underlying structure and support for secure connections.

The resulting system should cover most of the needs of the ROOT community.

ACKNOWLEDGEMENTS

The author would like to thank F. Rademakers, M. Ballantjin and P. Canal for useful discussions, suggestions and comments.

REFERENCES

[1] ROOT: http://root.cern.ch/
[2] PROOF: http://pierre.mit.edu/proof/
[3] MIT Kerberos: http://web.mit.edu/kerberos/www/
[4] Globus: http://www.globus.org/
[5] For a more detailed discussion, see R. Sandhu, M. Bellare, R. Ganesan, "Password-Enabled PKI: Virtual Smartcards vs. Virtual Soft Tokens", proceedings of 1st Annual PKI research Workshop, 2002, p. 89-96, and references therein.
[6] SRP: http://srp.stanford.edu
[7] See, for example, S. Garfinkel, G. Spafford, A. Schwartz, "Practical Unix & Internet Security", 3rd Ed., 2003, O'Reilly & Associates Inc.
[8] See M. Crawford, this conference.
[9] Xrootd: http://xrootd.slac.stanford.edu/; see also A. Hanushevsky, this conference.