SECURE CLIENT TIER

FOR THE ACCELERATOR CONTROL SYSTEM

A. D. Petrov, D. J. Nicklaus
Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.

ABSTRACT

The central part of the Accelerator Control System at Fermilab is a cluster of Java Data Acquisition Engines (DAEs). In order to read or set data, an application needs to connect to one of the DAEs through the plain Remote Method Invocation (RMI) protocol. As the system grew over the past decade, new security concerns appeared. The existing client-server communication protocol failed to meet higher security requirements, because it employs fairly simple rules of authentication and does not support either encryption or data integrity checks. Besides that, the API providing access to all functions of the control system seemed to be too complex for inexperienced client application developers. Therefore, it was decided to introduce an intermediary level in the architecture between DAEs and client applications. This tier, named Secure Controls Framework (SCF), provides security for the client connections and offers new simplified API for Control System access. In the SCF, security features are implemented on the transport level by means of the Kerberos V5 protocol. They include strong user authentication and encryption (or message integrity codes) applied to the network traffic. Special attention was paid to automation of the authentication process and making it less annoying for the users. A generic Kerberos implementation in Java was extended to support various types of ticket caches, including memory caches on Windows and Macs, and implement an automated ticket discovery. The rewritten control's API is based on a new object-oriented data model. Legacy data structures, such as devices, arrays, properties, and scaled values were described as Java classes in a way that simplifies their usage in client applications.

INTRODUCTION

Rapid development of information technologies in recent years has caused new security issues to emerge. Continual on-line attacks, viruses, and malicious software forced the networking administrators to take measures, such as installing firewalls and establishing stricter security policies, in order to protect the systems they are responsible for. Up to recent times, however, the scientific community rarely paid enough attention to these issues: this was a quite controlled environment and the data were not very attractive for disruption. Thus, the developed custom software did not use adequate means to protect the information. Since scientific labs are now targets of on-line attack like anybody else with relatively little security built into the custom software, all the information security has relied on the network facilities. Such network traffic filtering often makes it impossible to get data from a central system to the client applications, unless they are running on a limited secure network. The standard solutions, such as VPN, introduce additional complexity levels and sometimes do not work because of incompatibility with the proprietary programs and protocols.

For a large data acquisition and control system, the networking tools alone can hardly provide the level of security that is adequate yet flexible enough for a wide user community. It is essential to implement some reliable authentication algorithms, as well as protection of communication channels, inside that system itself. These security features must be based on reliable and publicly approved protocols in order to satisfy the formal requirements that may arise.

In this paper, we discuss the design of the Secure Controls Framework, developed as a part of Fermilab's Accelerator Control System. The project aims to base the control system security functions on an independent infrastructure. Although the SCF has a very specific field of application, this technology is general enough to be employed in other systems where data protection is required.

ARCHITECTURE

The central part of the Java Control System [1] at Fermilab is a cluster of about 100 Data Acquisition Engines. They obtain and consolidate the front-ends' data, run central services, such as data loggers, and provide communication with the client programs. DAEs talk to the front-ends and between each other with the proprietary ACNET protocol. Client applications connect to the engines through Java RMI over plain TCP/IP. A common data acquisition (DAQ) API is used on the clients, as well as on DAEs, to read and set data.

In the existing design, all the engines reside on a secure network. The incoming traffic is controlled by a firewall. Neither ACNET nor the data acquisition RMI is allowed to reach that area from the outside world and some parts of the lab. The client programs have restricted connectivity to the DAEs because of the lack of security in the DAQ API, which has a fairly simple authentication system and communication channel vulnerabilities.

Many controls applications at Fermilab are designed to provide data for a broad community of scientists and engineers. They should be able to run their tools from many locations, both inside and outside the laboratory. Although the detailed regulations are to be defined by a security policy, there is a need to have technical means providing such kind of access. Changes in the existing DAQ API were not desired because it is very complex and is being used on the running system. In one solution, the data were provided by dedicated web-services. Although that allowed isolation of the engines, it did not guarantee much security and had many other disadvantages. It was proposed to design a new simplified API suitable for the tasks usually performed by the client applications, and develop an implementation that would satisfy the security demands.

The introduced Secure Controls Framework consists of the server and the client parts. These parts communicate through Java RMI with an encryption algorithm built inside the underlying TCP/IP ports. The clients are using a strong user authentication protocol, a simplified version of the data acquisition API, and a new object-oriented data model. The SCF servers handle remote sessions, and convert user credentials and incoming data requests in a conventional form understandable by DAEs. Physically, the servers are running along with the engines on the same hardware.

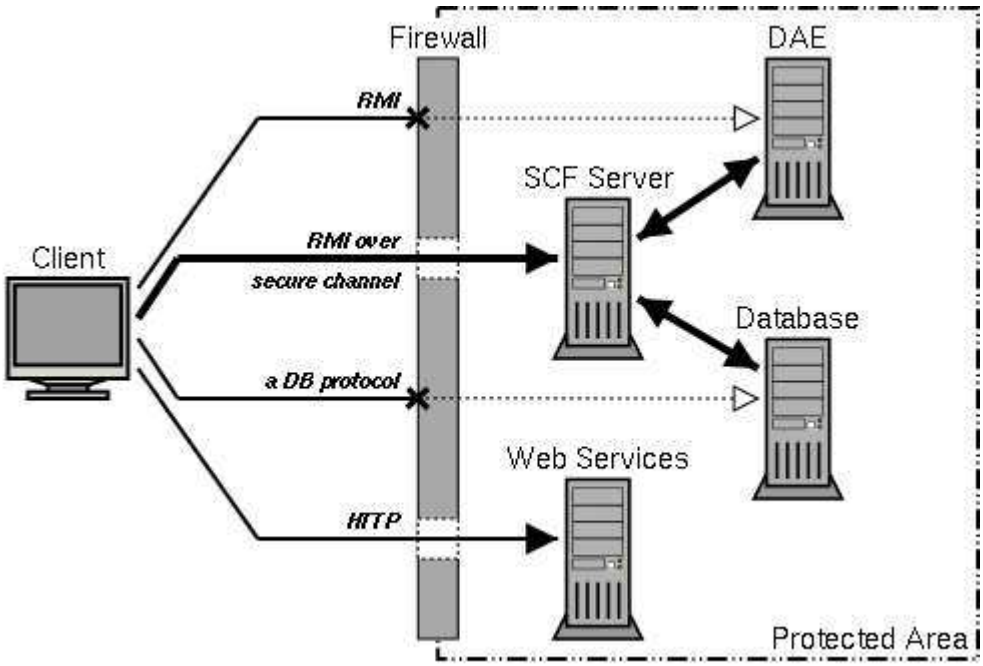

Fig 1. Secure Controls Framework Architecture.

As an addition, SCF provides access to some essential database resources through ad hoc requests. The databases are located within the restricted network and normally are not directly available from the outside.

THE CHOICE OF A SECURITY PROVIDER

A few years ago, Fermilab adopted the Kerberos V5 protocol [2] as a uniform security solution. This infrastructure, including client utilities, has been installed on a vast number of computers throughout the lab. When people started using it for common network services, such as *rlogin* and *ftp*, it becomes a routine to get a fresh ticket every morning. The Kerberos ticket (which essentially is a piece of binary data stored in a local file or in the memory) represents the user's identity for a certain period of time and can be used to request access to remote hosts. In that way, almost all the users have had a reliable platform-independent digital credential on their machines that could be used to secure the controls data.

Kerberos allows the authentication procedure to be inconspicuous for people and less annoying than asking passwords. As soon as the valid ticket is available, an application can use it to verify the user's identity and open a secure communication channel. This can be considered as an alternative to X.509 certificates providing similar capabilities. The Kerberos infrastructure, however, is quite different from the Private Key Infrastructure (PKI) engaged in the certificates issuance and distribution. Fermilab uses PKI for some applications, but the Control System has become oriented on Kerberos.

The Kerberos protocol provides strong authentication of principals with a high degree of robustness without requiring physical security of all the hosts on the network. To transfer credentials between two hosts, a technology named General Security Services (GSS) API [3] can be employed. As a result of the GSS handshake, two peers can establish a *common security context* and use per-message security services.

Another advantage of Kerberos is that the Control System no longer needs to keep and manage an extensive list of its users. All the principals are already stored in the central Key Distribution Center database, maintained by someone else with lab-wide responsibility. Even though the controls system might not have the link to it, a valid GSS context on the server always represents a legitimate user on the client. If needed, this user later can be tracked down, but for a while he or she has default privileges and can start working immediately. For the most users, these default privileges are sufficient to conduct their job.

Java SDK includes an extensive support of Kerberos V5 and the related GSS-API. Yet, there were a few serious drawbacks that required a workaround. The first problem was in the way Java searched for a Kerberos ticket—it was too simple and did not work reliably on all configurations we have had. The second problem was inability to read tickets from the operating system memory. Besides that, Java 1.4 did not provide an implementation for Kerberos cipher suites for the Transport Level Security (TLS) protocol.

AUTHENTICATION IN KERBEROS

The essential issue is an efficient acquisition of the Kerberos ticket granting ticket (TGT) that represents the user's credential.

It was decided not to request TGT in Java, but always read it from an external cache created by other utilities, such as *kinit* and *Leash32*. One of the reasons is not to deal with the Kerberos password, which is highly sensitive information: It is quite difficult to figure out whether a program is running on X terminal where password exposure is unwanted. Another reason is to keep using the tools that have already being used and have become habitual. Unfortunately, the generic implementation in Java (Krb5LoginModule) can not handle all the variety of configurations in the laboratory. It does not support memory caches, either.

In order to address those problems, a completely new Kerberos authentication module has been developed [4]. It is based on the regular Java concept of login modules. But unlike the generic implementation, the order in which the tickets are looked for is strictly defined for every operating system in the configuration file and can easily be adjusted.

The second important feature of the custom module is the support of memory caches. On Windows 2000/XP and Mac OS X the tickets can be stored in the operating system memory. To read them, one needs to use a native API. The module includes two native libraries that are called through JNI.

As will be described below, the server-side services also have to be authenticated. In most cases, this is done through keytab files that store secret keys (equivalents of passwords). The module provides the service authentication as well.

The Kerberos authentication module can be reused in other applications and applets. It is now available as an open source product.

SPECIAL USERS

In practice, we found a few places in the system where Kerberos authentication should be bypassed. In the Accelerator Control System, there are several control rooms with restricted physical access. The staff working there does not use Kerberos, but must be able to run the same software as other people. In SCF, that small group of users is authenticated by their host addresses. Each operator's console has a record in the database that indicates its corresponding default user name. All such nodes must be located in the area served by trusted network routers, to prevent IP spoofing.

TRANSPORT LEVEL SECURITY

In order to secure the communication, two authenticated nodes must establish a common security context, defined in Java in the GSSContext class. In a general case, on one node there is a user, and on another one there is a service. To create a GSSContext, the two hosts exchange few tokens via the insecure channel. The initiative always comes from the client. Upon completion of the handshake, the GSSContext instances on both ends get initialized with the opposite party's information and a secret key needed to proceed with the security services.

Security services provided by GSSContext include: encryption, message integrity codes (MIC), sequence detection, and replay detection. While a real encryption is hardly needed for the Control System, the MICs work as digital signatures and ensure that messages are coming from the legitimate party. The GSSContext class can “wrap” and “unwrap” streams to apply these features.

The standard Transport Level Security (TLS) protocol can support Kerberos [5]. However, Java 1.4 does not provide such an implementation. The SCF uses our own extensions of SSLSocket and SSLServerSocket that perform the GSS handshake and wrap/unwrap operations over the streams. The goal here is to comply with the standard specification from the RFC.

The new secure sockets are used in RMI through the corresponding RMISocketFactory.

DATA ACCESS

The Accelerator Control System is using a comprehensive Data Acquisition API. Its main idea, however, is quite simple—to read or set data, a program needs to create the data acquisition job with four parameters: source, disposition, item, and event. Over the time, most of the entities of the real system, such as the accelerator, data loggers, devices, and clock events, have been described in those terms. As this implementation includes a large amount of Java classes with many methods, it becomes less understandable for the occasional developers. This is one of the categories of users the Java Controls was intended for. The DAQ implementation complexity leads to a significant size of libraries (jar files) required for the data acquisition. Now it reaches at least 5.5 megabytes for a typical application. This makes it harder to create “thin” clients.

The SCF offers a simplified data acquisition API and new data structures. The idea of a four-parameter jobs stays unchanged, but the actual number of classes used, and the set of their methods have been significantly reduced. This reduction has been made at the expense of objects that are specific to the server-side operation and are rarely used in the remote applications.

For the data acquisition job items, completely new object-oriented data structures have been designed. These items are basically accelerator devices. In the legacy system they include a set of properties (reading, setting, control), and sometimes can be treated as an array, combined in composed structures, etc. The new model keeps all the elements in a hierarchical order and makes it possible to use the single object to get a device description from the database, to start a job, and to retrieve readings. Examples of the actual structures can be found in [5]. The server side converts

between DAQ and SCF data formats in both directions.

Besides getting data from the engines, the client applications need to have access to several important central databases, such as devices, parameter pages, and data logger items. SCF provides a flexible mechanism of pluggable modules each of which defines ad hoc requests to one database. These modules are deployed on the server side. On the clients, the Java Naming and Directory Interface (JNDI) is employed to obtain the data. Unlike in the conventional database access, the outcome of JNDI operations is pre-cooked Java objects rather than plain datasets.

The size of jar-files normally needed for a SCF client is around 300 kilobytes.

CONCLUSION

This paper has reviewed the design of the Secure Controls Framework, a new component of the Fermilab's Accelerator Control System meant to improve security for client applications. During development of the project, the concept of using an independent security infrastructure for the control system has been proven, and many practical details of such integration has been studied. The results of this work can be used in the design of new distributed control systems where reinforced data protection is required.

REFERENCES

[1] J. F. Patrick. ACNET Control System Overview.
http://beamdocs.fnal.gov/cgi-bin/public/DocDB/ShowDocument?docid=1762

[2] RFC 1510: The Kerberos Network Authentication Service (V5).
http://www.ietf.org/rfc/rfc1510.txt

[3] RFC 1964: The Kerberos Version 5 GSS-API Mechanism.
http://www.ietf.org/rfc/rfc1964.txt

[4] A. D. Petrov. Using Kerberos Authentication In Java.
http://beamdocs.fnal.gov/cgi-bin/public/DocDB/ShowDocument?docid=1502

[5] RFC 2712: Addition of Kerberos Cipher Suites to Transport Layer Security (TLS).
http://www.ietf.org/rfc/rfc2712.txt

[6] A. D. Petrov. Secure Controls Framework User Guide.
http://beamdocs.fnal.gov/cgi-bin/public/DocDB/ShowDocument?docid=1515