This paper appeared in the *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems*, September 12-14, 2005.

Towards a Security Service Integration Framework for Distributed Real-Time Systems

Tao Xie Xiao Qin
Department of Computer Science
New Mexico Institute of Mining and Technology
801 Leroy Place, Socorro, New Mexico 87801-4796
{xietao, xqin}@cs.nmt.edu

Abstract

Real-time applications with security requirements are increasingly emerging in parallel and distributed systems. However, the complexities and specialities of diverse security mechanisms dissuade users from employing existing security services for their applications. To effectively tackle this problem, in this paper we propose a security middleware (SMW) framework from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance the trustworthy executions of the applications. A quality of security control manager (QSCM), a centerpiece of the framework, is capable of achieving a flexible trade-off between overheads caused by security services and system performance, especially under high workload conditions where available resources are dynamically changing and insufficient. A security-aware scheduling mechanism, which plays a key role in QSCM, is able to maximize quality of security for real-time applications running on parallel and distributed systems like clusters and Grids.

Keywords security middleware, security control manager, security-aware scheduling, real-time applications, distributed systems.

1. Introduction

An increasing number of real-time systems have timing and security constraints because sensitive data and processing require special safeguards against unauthorized access [9] [11]. In particular, a variety of military real-time applications running on parallel and distributed systems like clusters require security protections to completely fulfil their security-critical needs. Unfortunately, conventional wisdom on the design of real time systems is inadequate for security-sensitive real-time applications because it did not factor in the applications' security needs.

To tackle the aforementioned problem, we propose a security middleware framework, which allows real-time applications to invoke various underlying security services through specific application programming interfaces (APIs) to satisfy their security needs. Employing the security services, however, requires extra overhead in terms of CPU time, network and disk bandwidth. Thus, real-time scheduling algorithms need to consider the overhead to make efficient schedules for tasks submitted. Consequently, applications or users are able to receive satisfactory service from real-time systems, which achieve high performance with respect to quality of security and schedulability. The security middleware framework can benefit both applications and the real-time systems. With the framework in place, applications or users are allowed to formally describe their security requirements using security services specifications, e.g., security-related APIs. These APIs then invoke an array of high-level security services provided by the fabric of the SMW framework (see Fig. 1). From a real-time system standpoint, it can leverage the framework to glean global information pertinent to the applications' security needs. Additionally, the framework makes it possible for the real-time system to measure the applications' security overhead. In doing so, the framework is able to make an effort to guarantee timing constraints and security requirements. In a security-critical real-time system, a task will be rejected by the system if the task's minimal security requirements cannot be met. This process is essential because running tasks without guaranteeing their security requirements tends to make the system vulnerable to attack. In short, the framework is intended to seamlessly integrate security into real-time scheduling for applications running in parallel and distributed systems.

The contributions of this paper are three-fold. First, a security middleware (SMW) framework is proposed. Second, a security-aware real-time scheduling mechanism is implemented. Finally, a case study illustrates the performance of the security-aware real-time scheduling mechanism in the light of the security middleware (SMW) framework. In the case study, we use trace-driven simulation to compare the performance of our mechanism against existing mechanisms in clusters where security-aware real-time scheduling algorithms are not employed. Our simulator combines performance and security overhead estimates using the security overhead model based on the three most commonly used security services, i.e., authentication, integrity, and confidentiality. We have used real world traces from a supercomputing centre to drive our simulations. Our empirical results demonstrate that the proposed framework, in which the scheduling mechanism is the centrepiece, is capable of achieving high quality of security while guaranteeing timing constraints of real-time applications.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces the architecture of our security middleware (SMW) framework. Section 4 concludes the paper with some comments on future work.

2. Related work

QoS-aware middleware has been extensively studied in the past both experimentally and theoretically [7][8][10]. Huang et al. proposed a middleware-oriented Global Resource Management System, or GRMS, which provides distributed applications with end-to-end QoS negotiation and adaptation [8]. Nahrstedt et al. designed a QoS-aware middleware that can offer a new generation QoS-sensitive applications such as media streaming and e-commerce with QoS support [10]. However, these two middleware systems are non-real-time in nature, meaning that they are inadequate for parallel and distributed real-time systems. Abdelzaher et al. presented a scheme for QoS negotiation in real-time applications. The scheme provides a generic way to express application-level semantics to control how application QoS is to be degraded under overload or failure conditions [7]. In addition, they demonstrated that their method enables QoS gracefully degradation under conditions in which traditional schedulability analysis and admission control schemes fail, while maintaining real-time guarantee. Although the above works addressed applications' QoS requirements in parallel and distributed systems, none of them paid attention to real-time applications' security requirements, which are increasingly becoming critical in real-time systems. Our work is orthogonal and complementary to the above approaches in the sense that the security middleware framework centered around underlying security services, is focused on the security needs of real-time applications.

The security middleware framework (SMW) provides a way of explicitly specifying the security requirements of real-time applications running on a parallel and distributed computing platform. It is indispensable for the framework to be aware of extra resource overhead incurred by applications' security requirements because the framework has to achieve an optimized trade-off between system security and performance. To the best of our knowledge, the way of calculating costs of security service has received little attention. Irvine et al. proposed a model of computing costs for quality of security service [1]. In their approach application's security requirements are specified by a security vector, which is composed of an array of sub-vectors with each sub-vector being a particular security service used [1]. Each sub-vector could consist of two components, namely, the name of the security service, and a security level range of the particular security service demanded by the application. Wang et al. presented a security measurement framework, which is based on theory and practice of formal measurements [13]. Their work provided us an insightful view of a future direction of security measurement. In our previous work [12], we proposed a practical security overhead model to estimate the CPU time overhead of some commonly used security services like authentication and integrity.

Real-time schedulers play an important role in parallel and distributed real-time systems. The schedulers are responsible for matching and dispatching real-time tasks onto system resources. The issue of scheduling for real-time applications was previously reported in the literature. Conventional real-time scheduling algorithms such as Rate Monotonic (RM) algorithm [2], Earliest Deadline First (EDF) [5], and Spring scheduling algorithm [4] were successfully applied in real-time systems. We proposed both static and dynamic scheduling schemes for real-time systems [3]. Unfortunately, none of the above real-time scheduling algorithms can be directly applied to real-time systems, which have security requirements.

Most recently, we proposed a family of dynamic security-aware scheduling algorithms for a single machine [6], a cluster [12], and a Grid [14]. We conducted simulations to show that compared with six heuristic algorithms, the proposed algorithms can consistently improve overall system performance in terms of quality of security and system schedulability under a wide range of workload conditions.

3. Security Middleware Framework (SMW)

Middleware is software that links two otherwise separate applications or separate products that serve as the glue between two applications. It is used to solve computer clients' heterogeneity and distribution issues by offering distributed system services that have standard programming interface and protocols [15]. We refer to these system services as *middleware services*, because they reside in a layer between networking, operating system software and specific applications. In this section we propose a security middleware (SMW) framework, which aims at meeting security requirements of a variety of applications and improving performance of distributed real-time systems. Section 3.1 presents an overview of the architecture for the SMW framework. Detailed functional descriptions of each component of the SMW framework can be found in Section 3.2. Section 3.3 illustrates how to specify applications' security requirements. Section 3.4 provides an analytical model for the local schedulability analyser.

3.1. Architecture of the SMW framework

The SMW framework consists of a user interface, a

framework, low-level security service APIs, a quality of security control manager, and security middleware services (see Fig. 1).

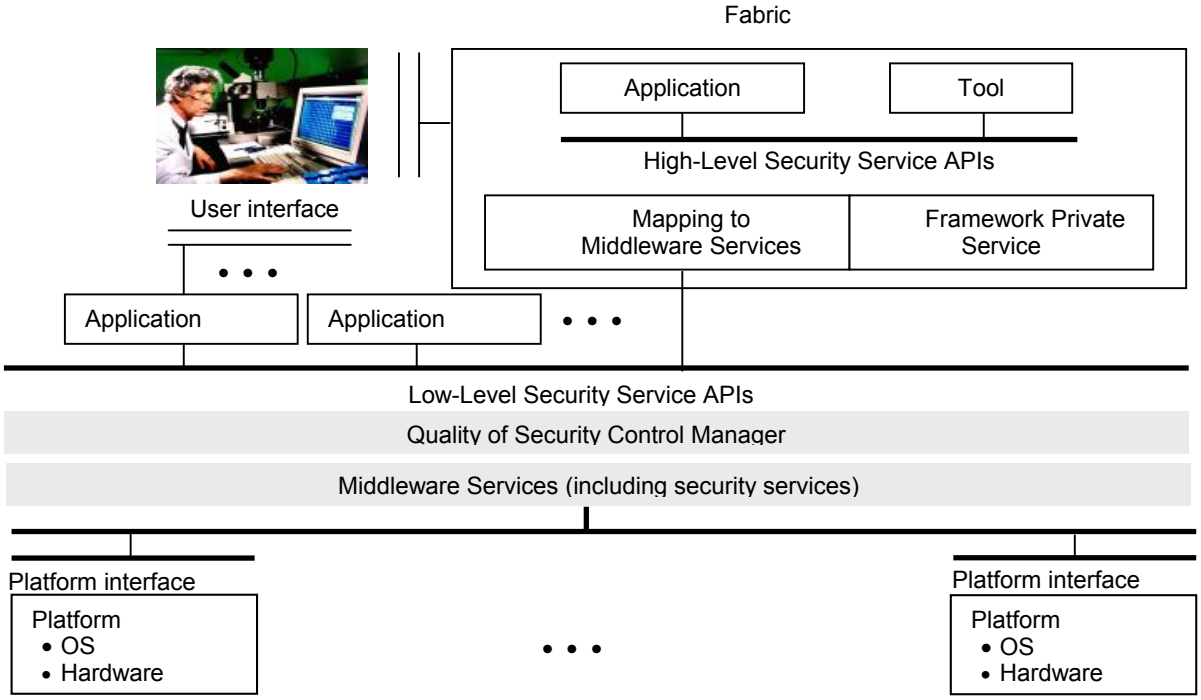

Figure 1. Security middleware architecture

The SMW framework provides two different types of user interfaces, namely, a professional user interface and a normal user interface. The professional user interface is an interface between developers (e.g., programmers) and applications being developed. An editor, a compiler and a debugger are essential components of the professional user interface. Programmers are allowed to directly access the low-level security service APIs, thereby efficiently constructing applications with various security functions. A normal user interface sits between a normal user and the framework. By using the normal user interface, usually an IDE (integrated development environment), a normal user such as a system administrator can leverage the framework to readily create his applications with security requirements.

A *framework* is a software environment that is designed to simplify application development and system management for a specialized application domain [15]. The framework illustrated in Fig. 1 is composed of a set of high-level security service APIs, an array of tools, a security middleware services mapping module, and framework-private middleware services. The functionality of the framework is two-fold. First, it provides developers an efficient computing environment in which security-aware applications can be rapidly developed. Second, the framework makes it possible for users to manipulate security-related system parameters. As a result, there is no need for developers and users to directly access low-level security service APIs, which are, in most cases, complicated to use. The high-level security service APIs may be (1) an abstraction of low-level security service APIs for the underlying security middleware services, or (2) a new set of APIs that encapsulate the low-level security service APIs. When the high-level APIs are different from their low-level peers, they may add value by specializing the user interface, simplifying the low-level APIs, or import framework-private middleware services. The applications within the framework are administration applications from which the users (including administrators and programmers) can manage and configure multiple security services by employing the high-level APIs with the assistance of some tools. The objective of the tools in the framework is to simplify the use of the high-level APIs. For example, a security service virtualization tool offers users a visible table that demonstrates all currently available security services and their corresponding costs (see Table 1. in Section 3.3). The security middleware services mapping module is responsible for translating the high-level security service APIs into their corresponding low-level counterparts. Framework-private services provide specific functions in addition to the underlying middleware services to meet framework's own needs.

The low-level security service APIs are programming interfaces through which underlying security services

included in the middleware services can be invoked. We can implement our low-level security service APIs based on the Generic Security Service API described in [16], which allows a calling application to authenticate principle identity associated with a peer application, to delegate rights to a peer application, and to exploit security services such as confidentiality and integrity on a per-message basis [16]. A sample API routine could be *gss_verify_mic()*, which can check a message integrity code (MIC) against a message to verify integrity of a received message. Another example routine is *gss_indicate_mechs()* that determines available underlying authentication mechanism.

Quality of security control manager (QSCM) is a module needed for optimizing applications' security requirements based on available system resources. Conceptually, it is an engine for security-critical real-time system to achieve a high system performance in terms of quality of security and schedulibility. Detailed description of QSCM will be explained in Section 3.2.

A middleware service is a generic service that operated between platforms and applications (see Fig. 1). The middleware service, which is defined by the APIs and supported protocols [15], has several features that differ general-purpose applications or platform-oriented services from the middleware service. Specifically, the middleware service is distributed, capable of running on multiple platforms, and supporting standard interfaces and protocols. Among the middleware services, authentication service, auditing service, encryption service and access controller are commonly used security services in a distributed system. For instance, authentication service provides functions to an application related to establishing, verifying, and transferring a person or a process. These security middleware services furnish a set of standard APIs (e.g., low-level security service APIs), which can be invoked in applications. The services are in forms of standard routines, which can be implemented using programming languages such as C and Java. For example, a Java Security Service Module is a commercial product that facilitates the above services implemented as classes [17].

3.2. Quality of Security Control Manager

QSCM (Fig. 2) is a centerpiece of the SMW framework because it can optimize the quality of security services requested by applications while maintaining a high-level system performance in terms of schedulability.

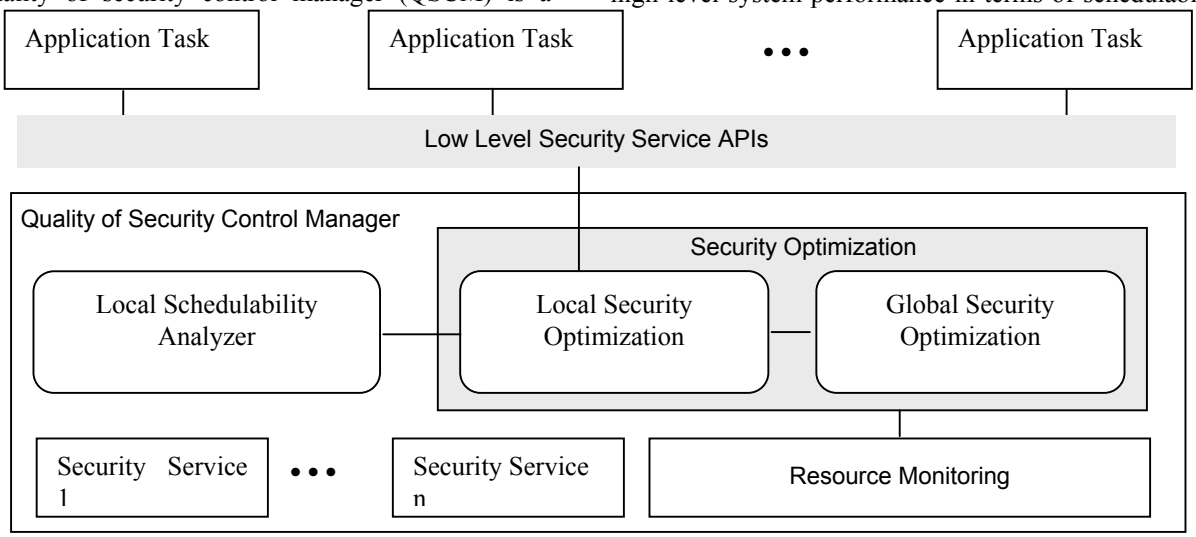

Figure 2. Quality of security control manager

The input of the QSCM module is a security service attribute-value vector specified by users, and the output is an array of selective values for each required security service. The most important abstraction in our QSCM module is *security level*, which is used to indicate the strength or safety degree of a particular security service.

A security service is implemented by a particular security mechanism. For example, encryption, a security mechanism, provides a means to implementing confidentiality, which is a security service. Thus, the strength of a security service is mainly decided by the robustness of the security mechanism that implemented it. Further, the strength of the security mechanism largely depends on (1) how rigorously the security algorithm is tested, (2) how long it has been used, and (3) how robust it is under attacks performed against it [13]. From a normal user's standpoint, a security level may be a subjective and qualitative value like "low", "medium", and "high". For a security professional, on the other hand, the security level could be a quantitatively measured value such as 0.3, a normalized value when setting the strongest security

mechanism as 1. In the latter case, security level is a relatively objective value obtained by some reasonable and practical measurement methods. In addition, security levels are represented in terms of security parameters whose semantics only need be known to the user and the service provider (e.g., security middleware service).

A *security range,* which is a scope, contains multiple distinct security levels for a particular security service. The lowest value in a security range indicates the minimal security strength mandated by the user, while the highest value implies the maximal security strength necessary for the user and all the values above should not be considered. We will discuss the security level specification in section 3.3.

The QSCM runs on top of middleware services and uses the Resource Monitoring module to monitor the underlying available resources. A user is enabled to submit a task T_i along with its security requirements expressed by a vector of security ranges, e.g., $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where T_i requires q security services. S_i^j is the security range of the jth requested security service.

The security optimization module, which plays a key role in QSCM, is responsible for choosing the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$. The objective of the security level selection is to maximize overall utility in terms of quality of security (see Section 4).

The local schedulability analyzer aims at checking whether or not the selected security level can be supported under current workload conditions. With the assistance of the local schedulability analyzer, the security optimization module performs admission control on arrival application tasks.

The scheduling mechanism has to make use of the schedulability analyzer and the security optimization module to measure the security benefits gained by each admitted task. In particular, the security benefit of task T_i is quantitatively modeled as the following security level function.

$$SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j \text{ ,where } 0 \le w_i^j \le 1, \sum_{j=1}^{q} w_i^j = 1, \quad (1)$$

where w_i^j is the weight of the jth security service. Note that it is programmers' responsibility to define the weights to reflect relative priorities given to the required security services.

Suppose X_i is all possible schedules for task T_i generated by the scheduling mechanism, and $x_i \in X_i$ is a scheduling decision. The schedulability analyzer considers x_i a feasible schedule if (1) the security requirements are satisfied, and (2) its deadline can be met. Given a real-time task T_i, the security benefit of T_i can be maximized by the security level controller (See Fig. 2) under the timing constraint:

$$SV(x) = \max_{x \in X} \left\{ \sum_{i=1}^{p} y_i SB(x_i) \right\} \quad (3)$$

where p is the number of submitted tasks, y_i is set to 1 is task T_i is accepted, and is set to 0 otherwise.

The local security optimization module is used to select security levels only for local clients based on local machine resources, while the global security optimization module is launched using a load-sharing algorithm, which can exploit the distributed system resources when the local resources is not enough to sustain client's service requests.

3.3. Security Service Requirements Specification

In this subsection we present two approaches to specifying users' security service requirements. One is for professional users, whereas the other is for system administrators. Irvine *et al.* proposed the notion of security range that consists of a set of security levels [18]. Users can define their security requirements for a particular security service by specifying a security range. To accomplish this goal, our SMW framework provides users with a *task submission description language* (TSDL), a vehicle that users can leverage to articulate their security needs upon the submissions of their tasks. Figure 3 illustrates an example of the task submission structure (TSS) described in TSDL.

A TSS is a highly flexible and extensible data model that can be utilized to represent multiple security services and constraints in a submitted task. It is a mapping from attribute names to expressions. For example, Processor_Num is an attribute and the number 5 is its corresponding expression. Expression might be integers, string constants, or more complicated expressions constructed with arithmetic and logical operators such as "0.3 <= Integrity <=0.8" (See Fig. 3).

After a user submits a TSS, the security service constraints it will be translated into the high-level security service APIs. Therefore, there is no need for users to directly deal with the APIs. To further alleviate users' burden, the framework of the SMW framework makes is possible for system administrators to specify security requirement expressions using a higher-level abstraction, e.g., security abstract table.

4. Summary and Future Work

In this paper, we presented a novel security middleware (SMW) framework from which a security-sensitive real-time application can exploit a variety of security services to enhance the safety of its execution on

distributed systems.

```
DEFINE Task : flight_control
{
    Input = (altitude: 1230, heading: 35, …);
    Output = (takeoff_distance, climb_rate);
    Type = "Real Time";
    Period = 80;
    Completion_Time = 0;
    Owner = "sqin";
    Cmd = "flight_con";
    Processor_num= 5;
    Data_secured=250;
    Priority = 3;
    Constraint
        • Arch == "INTEL";
        • OS == "UNIX";
        • Disk >= 480;
            • Memory >=128;
        • Deadline = 80;
            • 0.3 <= Authentication <=0.6;
            • 0.4 <= Integrity <= 0.8;
            • 0.5 <= Confidentiality <= 0.9
}
```

Figure 3. Task submission structure

Future studies in this research can be performed in the following directions: (1) Extend our SMW framework to multi-dimensional computing resources. For now, we simply consider CPU time, which is only one of the computing resources consumed by the security services. Memory, network bandwidth and storage capacities should be considered in the future; (2) Accommodate more security services into our SMW framework. Besides the three security services discussed, we plan to include authorization and auditing services into consideration; (3) Consider parallel applications that have dependencies upon each other. In this paper, we assume that all applications are independent. To make our SAREG strategy more practical, we need to extend it to general parallel applications.

References

[1] C. Irvine and T. Levin, "Towards a taxonomy and costing method for security services," Proc. 15th Annual Computer Security Applications Conference, 1999.

[2] L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, Vol.20, No.1, pp. 46-61, 1973.

[3] X. Qin, H. Jiang, D. R. Swanson, "An Efficient Fault-tolerant Scheduling Algorithm for Real-time Tasks with Precedence Constraints in Heterogeneous Systems," Proc. 31st Int'l Conf. Parallel Processing (ICPP 2002), pp.360-368. 2002.

[4] K. Ramamritham, J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time system," IEEE Software, Vol. 1, No. 3, July 1984.

[5] J. A. Stankovic, M. Spuri, K. Ramamritham, G.C. buttazzo, "Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms," Kluwer Academic Publishers, 1998.

[6] T. Xie, A. Sung, and X. Qin, "Dynamic Task Scheduling with Security Awareness in Real-Time Systems", Proc. Int'l Symp. Parallel and Distributed Processing, the 4th PMEO Workshop, IEEE/ACM, April 2005.

[7] T. F. Abdelzaher, E.M. Atkins and K. Shin, "QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control," *IEEE Transactions on Computers*, Vol. 49, No. 11, November 2000.

[8] J. Huang, Y. Wang and F. Cao, "On Developing Distributed Middleware Services for QoS- and Criticality-Based Resource Negotiation and Adaptation," Real-Time Systems 16(2): 187-221; May 1999.

[9] G. Donoho, "Building a Web Service to Provide Real-Time Stock Quotes," MCAD.Net, February, 2004.

[10] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-aware middleware for ubiquitous computing," *IEEE Communications Magazine*, 39(11):140--148, November 2001.

[11] E. Durant, "Embedded Real-Time System Considerations," EECS Department of Milwaukee School of Engineering, April 23, 1998.

[12] T. Xie, X. Qin, and A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters," *Proceedings of the 34th International Conference on Parallel Processing (ICPP 2005)*, Norway, June 14-17, 2005.

[13] C. Wang, and W.A. Wulf, "Towards a Framework for Security Measurement," Proceedings of the Twentieth National Information Systems Security Conference, Baltimore, MD, pp. 522-533, October, 1997.

[14] T. Xie and X. Qin, "Integrating Security Requirements into Scheduling for Real-Time Applications in Grid Computing," Pthe 2005 International Conference on Grid Computing and Applications (GCA'05), Las Vegas, June 20-23, 2005.

[15] P.A. Bernstein, "Middleware: A Model for Distributed System Services," Communication of ACM 39(2): pp. 86-98, 1996.

[16] J. Wray, RFC2744- Generic Security Service API Version 2: C-bindings, http://www.faqs.org/rfcs/rfc2744.html, January, 2000.

[17] Programming Security for Java Applications, http://e-docs.bea.com/wles/docs42/programmersguide/

[18] C. Irvine and T. Levin, "Quality of Security Service", Proc. of New Security Paradigms Workshop2000, Cork, Ireland, September, 2000.