An approach to security for world-wide applications

Ciarán Bryce and Winfried Kühnhauser
GMD - German National Research Centre for Information Technology
Schloss Birlinghoven, 53754 Sankt Augustin, Germany.

Abstract
There are two major security requirements for wide-area multi-media applications (WWAs). First, due to the large number of users and services, and to the presence of disparate legal requirements since WWAs operate in international environments, WWAs must support the enforcement of expressive user security policies. Second, for economic reasons, users require assurance guarantees that the WWA is capable of enforcing their security policies, and so engineering tools must be offered to help them design and verify these policies. This paper describes an approach to these requirements which is based on custodians. A custodian is a software black box which is linked to a WWA server and implements a security policy for objects under the server’s control. Using programmes to represent policies means that expressive security requirements are supported. In addition, we have developed tools to assist policy engineering. Custodians have been included in the design of an electronic market platform called CWASAR (Bryce et al. 1997).

Keywords
Security architecture, telematics and multi-media applications, custodians.

1 INTRODUCTION

Advances in multi-media technology are one reason for the large current interest in telematics applications. Such applications now provide a wide range of multi-media and commercial services to thousands of users spread over large areas, e.g., business-to-business transactions, mass consumer marketing, wide-area CAD applications. Telematics applications could thus be termed a class of world-wide applications (WWAs).

Despite the commercial success of telematics applications, many users still feel security to be a major problem. This conclusion was made clearer to us during the design phase of an electronic market platform called CWASAR; a market survey in Germany covering associations with over 200 000 member companies showed that security worries are widespread (Eco 1997). We be-
lieve that there are two core requirements to be addressed with respect to WWA security, which we elaborate upon in this paper.

The first of the key security requirements is that the WWA security architecture be capable of enforcing an expressive set of security policies. This is needed because users possess their own security requirements for their information. For instance, a typical user security requirement could be 'Access to my company documents is dependent on those clauses of an electronic contract which potential users of the information have signed'. Security requirements may also depend on the WWA service, e.g., access constraints to a hotel’s reservation server may differ for regular clients who are allowed to book without furnishing evidence of arrival.

Another reason for insisting on the ability of the architecture to enforce an expressive range of policies is that WWAs operate in an international context, where disparate legal guidelines are in place with respect to use of cryptography techniques and with respect to information comparison. To cater for more WWA services, the security architecture must be capable of implementing legal security constraints as precisely as possible.

The second of the key security requirements is that the WWA possess a support service for the security engineering of user and application security policies, that is, to help users and service providers to design and to verify their security policy requirements. There are two reasons for this. First, very few WWA users do know, want to know or should know the details of security policy implementation; this should be done for them. Second, companies are becoming increasingly dependent upon WWAs economically and so would appreciate a certified security quality of service (QoS) from the WWA provider.

Our goal in this paper is to describe the approach that we have taken to the dual objectives of an expressive security architecture and a security engineering framework in the Cwasar electronic market platform (Bryce et al. 1997). The security architecture is based on custodians (Kühnhauser 1995): encapsulated software programs which are bound to a WWA server and enforce the security of server objects under their control. As a custodian is a programmed representation of a policy, it allows for more expressiveness in the range of policies enforced. We discuss custodians, the security architecture and the Cwasar platform in section 2. In section 3 we overview the approach being adopted in Cwasar to security policy engineering. Our conclusions are presented in section 4.

2 A WWA SECURITY ARCHITECTURE

The WWA security architecture is influenced by the general architecture on which the WWA runs: that of an autonomous decentralised system (ADS). Thus, we first overview the main features of an ADS to see how its features are both exploited within and impose constraints upon the security architec-
ture. We subsequently introduce custodians as a mechanism for security and then terminate the section with a look at the implementation of the security architecture in the CWASAR platform.

![Architecture of an autonomous decentralised system](image)

**Figure 1** Architecture of an autonomous decentralised system.

### 2.1 Autonomous Decentralised Systems

As suggested by figure 1, an ADS is a networked connection of independently administered sites and domains. Since the network is an interconnection of several publicly and privately owned networks, WWA providers must be sensitive to the amount of network traffic which their application needs since users desire network costs to be as small as possible. Consequently, the security architecture must also keep extra communication costs to a minimum.

The second main feature of an ADS is the presence of multiple independent administrative domains. There is thus an absence of a global trusted authority, that is, there is no user equivalent to what the 'system administrator' is in a typical LAN distributed system. More importantly, the actions of users in different domains may be governed by different legal guidelines. An important example of this is the European law on encryption: in France, a government licence must be sought to make use of encryption whereas in Sweden, one is obliged to have all electronic information relating to an individual encrypted. This means that the security architecture must allow for the implementation of disparate security requirements within cooperating domains.

A final feature of ADSs is the presence of *proximity* servers. Their role is generally to address the access cost issues which WWAs face, as well as performance overheads, by acting as a cache of information and services for client sites in their geographical vicinity. A proximity server has a trusted administrator who could be a telecom provider or a private company which charge for its usage. One could view the collection of proximity servers as
being to ADSs, what file servers are to LAN distributed systems. As we shall see, proximity servers are exploited by the security architecture.

The remaining host systems of an ADS range from standard PC/modems in someone's home or office, to ordinary LAN systems belonging to companies. Given the heterogeneity of client host systems, it is important that the security architecture support autonomy of mechanism. This means that we must allow users to participate in WWAIs irrespective of how open or insecure their sites are.

As an example of the need for autonomy of mechanism, consider some shortcomings with the traditional access control mechanisms of ACLs (Bever et al. 1993) and capabilities (Levy 1984). Possessors of capabilities naming objects owned by other domains are expected to protect them, and not to loosely transfer them to users who should not have them; otherwise the purpose of using capabilities is defeated. Some sites just cannot respect this constraint due to their open environment. Even in the ACL mechanism, a site employing an ACL to control access to its objects relies on other sites to similarly protect the user identity certificates needed for authentication of prospective accessors.

2.2 Custodians

The basis of the security architecture is to insert software black boxes, called custodians, into the WWA of the ADS. A custodian is bound to any number of WWA objects and enforces these objects' security (Kühnhauser 1995, Härtig et al. 1993). A custodian is trusted and self-contained in the sense that it contains most of the mechanisms needed by objects for their security. Moreover, a custodian enforces a security policy: it contains a programme which is evaluated each time that an object it is bound to receives a request. Based on the programmed rules, the access to the object may be permitted or refused, and supplementary actions such as logging in an audit file can also be taken.

The basic interface to a custodian is that of the object that the custodian is designed to secure, or of the server controlling these objects. The custodian is placed on the site of the object if that site is considered trustworthy enough, otherwise a proximity server is used. As will be seen, the interface to the custodians in CWASAR are those of the basic data functions defined for the system. The custodian also contains policy-specific operations in its interface. If, for instance the policy authorisation scheme were to be based on Unix ACLs, then the interface of the custodian would also contain the operations chown( ) and chmod( ), and the state of the custodian would contain the access lists.

Figure 2 illustrates with an example the principle of WWA security. An object O1 is retrieving information from object O2 of another administrative domain. The dotted arrows show the direction of the communications. The call starts from O1 on site 1 and is intercepted by custodian A; the call is
then sent to object O2 though is intercepted by the proximity server holding custodian B, the custodian of O2. This custodian may have been placed on the proximity server since the site 2 was considered not trustworthy enough to securely store the custodian. Object O2 has a replicate, denoted O2', and the availability policy of custodian B chooses to execute operations in parallel on both replica. The result is returned by following the dotted arrows in their reverse directions. In this example, O1 is authenticated by custodian A, custodian B is responsible for access authorisation of O2; both custodians implement communications security though it is assumed some prior agreement exists between the administrators of both domains.

As said in the opening of this section, LAN security mechanisms lack the expressiveness needed for today's security policies as well as sufficient autonomy of mechanism. We have argued that custodians help to offer expressiveness by encapsulating policies within custodians and by representing these policies as programme units. Custodians help to support autonomy of mechanism by permitting sites with varied support for security to participate in a secure WWA. For example, the custodian, as part of its policy, may choose sites that it trusts to store user certificates or session keys - belonging to user sites that lack the security to store them locally - by exploiting the proximity servers.

For any security architecture to work, we require that any custodian implementation abide by the following four properties:

1. **C1 - Dynamic Binding**: An object owner should be able to choose a custodian, and contained security policy, for his object, and then have this linked to the object. He may also decide to have this object linked to another custodian in the course of its lifetime, should his security policy for the object change.

2. **C2 - Total Mediation**: To ensure security, the custodian must intercept each WWA communication, so that the legality of the call may be verified.

3. **C3 - Encapsulation**: The custodian itself must be protected from tampering for security to be effective.

4. **C4 - Support for Simple Verifiable Design**: Since the security of an application depends on the correctness of the custodians' programming with respect to furnished requirements, it is useful to keep the custodian structure simple so that the QoS of an application’s security can be more
easily ascertained by its users. A further reason for this is to allow security violations and failures to be traced to responsible units.

Property C4 is a security engineering issue and we return to this in section 3. The first three issues must be dealt with explicitly by the security architecture. We now overview one such architecture: the Cwasar electronic commerce platform (Bryce et al. 1997).

2.3 The Cwasar Electronic Commerce Platform

Cwasar is a European Union (EU) sponsored project whose aim is to offer an electronic commerce infrastructure to European small and medium sized enterprises. The system consists of a large network connecting several central repositories of commercial data ('Digital Sites') with many independent commercial, administration, research and educational organizations of different sizes. The main services will include a continuous electronic fair for advertising, a service to conduct business transactions such as ordering, booking, contract negotiations as well as the secure transfer of documents such as text, hypertext, engineering data and multimedia documents. The key technical requirements for the system are: 1) low network access costs, 2) streamlining of services so that the information they furnish is as relevant as possible, and 3) security. By security, we especially mean the ability of information and service providers to specify and have enforced their own rules governing access to their information.

The first step in the design of the Cwasar architecture was to specify its service model. This defines how the system's information base is organised and the functions which electronic services use to exploit this base. The basic information base in Cwasar contains General Data. This is defined as the set of all up-to-date company related information in the system and is stored in a unique logical general database (GDB). All information needed by Cwasar's services is extracted from the GDB. Domain Data is data which is extracted from the general data for use by a particular electronic commerce service. Domain data is stored in a set of logical domain databases (DDBs). Moreover, the data of a particular domain is structured in a way which best suits the service using that domain. The basic idea here is that a service's DDB should be able to satisfy most (we say 80%) of the service's information requests in order to satisfy the 'relevant' Cwasar requirement (number 2 above). DDBs are located nearer to the end-user and his service front-end, so network access costs are reduced. General and Domain data is organised around a structured Document, corresponding to a top-level object class from which all other data classes are derived. The Document class has been defined to allow the representation of structured data, such as company profiles, as well as unstructured data, such as a picture of a company's product (Amouroux
et al. 1997]. A Document is the basic unit to be secured; each company that enters a Document into the system, also enters a custodian containing the security policy of that Document.

The basic functions of the service model are responsible for acquisition (the automatic update of the GDB from information stored on user sites), querying of the DDBs and GDB (through the GD-Query and DD-Query functions), as well as derivation (the automatic transformation of general data to domain data) (Amouroux et al. 1997).

![Cwasar server schema](image)

**Figure 3** Cwasar server schema.

The GDB is stored on one of several centralised and trusted Cwasar servers (figure 3). Each basic function has a software server implementing its functionality on the Cwasar server. The custodian server, whose role is principally to ensure that each access to a Document is mediated by the custodian of that Document, is implemented by the custodian server process. The custodians that the security function needs to access are stored alongside the Documents in a custodian cache on the Cwasar server.

In order to minimise network usage, clients will have access to Cwasar services and the information base via a proximity server placed in their locality. There will be a limited number of proximity servers, and therefore many clients might use the same proximity server. The architecture of the client site and proximity server is shown in the figure 4.

On the client side, users formulate their requests with an interface language and these requests are then sent to the nearest proximity server. The proximity server has a software proxy running for each data handling function; there is also by consequence a proxy for the security function. Proximity servers contact the server when they do not possess the information needed to satisfy a request. Also, GDB modification requests made through the acquisition function are forwarded to the Cwasar server, and then eventually to other proximity servers.

Proximity servers also cache custodians. More than this, proximity servers must be able to implement security constraints particular to the country in which the server is operating. For instance, a proximity server in France must
ensure that client-server communications are not encrypted, whereas a server in Sweden ensures that they are encrypted. To achieve this, each proximity server has custodian classes of its own programmed to model these requirements. Thus, the custodian cache of a proximity server does not just contain a (partial) copy of the CWASAR server’s custodian cache.

The custodian server and its proxies are responsible for the total mediation (C2) property. Dynamic binding (C1) occurs in the acquisition function - a custodian is entered along with the Document. CWASAR and proximity servers are trusted to protect the custodian caches for property C3. Finally, property C4, verifiable custodian design, is a security engineering problem, which we look at in section 3.

3 SECURITY CUSTODIAN ENGINEERING

The second WWA security issue is security policy engineering - the provision of methods to facilitate the design and verification of custodians and their policies. We give an outline of our work on this issue in the context of the CWASAR project here; more details are found in (Bryce 1997a, Bryce 1997b).

Our approach to custodian engineering is based on software development reuse, through the use of an object-oriented style library that contains security policy units. These units include state of the art communication protocols, audit logs, developed custodian access policies, etc. The units are mainly specified as C++ classes so that design reuse via inheritance may be exploited (Dixon et al. 1987). As seen from figure 6, the library currently has compartments for access control policies, audit policies, as well as for the SecuDE source code (Schneider 1993). SecuDE is a Unix tool kit containing cryptographic tools for secure communications; this package will be used in CWASAR.

An example of the library’s use of C++ inheritance is illustrated in figure 5, which shows a custodian Mailer developed for a CWASAR message exchange service, a custodian Closed Groups developed to enforce the access abstraction of a closed virtual network for a group of users, and a combined custodian,
Cwasar Mailer, designed to include features of both former custodians. These are in fact custodians of policies developed using the facilities of the security library (Bryce 1997a). Further, the custodian is the mandatory policy being used by a Cwasar service known as the Business Yellow Pages (Bryce et al. 1997). Regarding the custodian figure, it suffices to say here that Mailer has an audit component and that its access control component centres on the use of a lattice and access matrix; the reason for the presence of the lattice is explained below. The role of sys and its super-classes in figure 5 is to house mechanisms needed to implement the custodian policy components in a Unix based system (another custodian implementation); this class contains the code to implement properties C1 to C3. (We could use this implementation on the Cwasar server; tamper-proofness (TP) and total mediation (TM) are got by booting the custodian as a system administrator process and where user socket spaces are controlled (Bryce 1997a).)

![Diagram of C++ inheritance hierarchies.](image)

**Figure 5** Examples of the custodian C++ inheritance hierarchies.

Policy verification in practice involves much standard program verification - reasoning about the pre and post conditions of policy operations e.g., that audit_store() does log the communication, or that am_revoke() does remove the right from the policy’s access matrix. Admittedly, C++ is not the best choice of language for conducting these kinds of proofs, so the idea is that the essential policy components such as access matrix and audit functionality be declared using a high-level type definition language, and then pre-compiled to a C++ class. We possess such a language for access control policies, an example declaration being given below of a multi-level security policy MLS (DoD 1985). Access control types are stored in their own compartment of the library, as shown in figure 6.

The access control policy example declaration is taken from classic multi-level security. It defines the policy state and the operations which update this state (e.g., exchange access rights among the entities governed by the policy). A policy’s state is represented by an access matrix \( m \) and a classification hierarchy \( cl \) modelling a lattice. We chose policy types to be operational in specification in order to facilitate the building of the custodian code generator; type declarations employ set and function operators, e.g., \( \oplus, \cup, \setminus \). For
instance, given a partial function \( F \) of type \( \text{Nat} \to \text{Nat} \) with value \( \{1 \mapsto 2, 3 \mapsto 4\} \); \( F \cup \{3 \mapsto 6\} \) is \( \{1 \mapsto 2, 3 \mapsto 4, 5 \mapsto 6\} \); \( F \uplus \{3 \mapsto 5\} \) is \( \{1 \mapsto 2, 3 \mapsto 5\} \); and \( F \setminus \{3 \mapsto F(3)\} \) is \( \{1 \mapsto 2\} \). Below, read and write are application operations, that is, operations of the application which the policy is trying to control; transfer-read is a policy specific operation, its goal being to alter the access state.

**Policy** \( \text{MLS} = \text{Policy-Sets} \): /* define variables of policy */

\[
\text{level} = \{\text{level0}, \text{level1}, \text{level2}, \text{level3}\};
\]

**Classification**: /* define classification ordering */

\[
\text{level} : \{\text{level0} \leq \text{level1}, \text{level1} \leq \text{level2}, \text{level2} \leq \text{level3}\}
\]

**Oper**: /* process’ level must dominate object’s */

\[
\text{read}(s, o) : \text{cl}(s) \leq \text{cl}(o) \land \text{read} \in m(s, o)
\]

**write**(s, o): /* process’ level must be dominated by object’s */

\[
\text{write}(s, o) : \text{cl}(s) \leq \text{cl}(o) \land \text{write} \in m(s, o)
\]

**transfer-read**(s1, o, s2): /* s1 transfers read right for o to s2 */

/* since this is a policy operation, its semantics must be defined */

**where**:

\[
m' = m \uplus \{(s1, o) \mapsto m(s1, o) \setminus \text{read} \} \cup \{(s2, o) \mapsto m(s2, o) \cup \text{read}\}
\]

**pre**: \( \text{cl}(s1) \leq \text{cl}(s2) \land \text{read} \in m(s1, o)\)

**Inv**: /* define valid states */ /* true;**

**State**: /* initial state of policy */

/* entities governed by policy - bound to WWA objects */

\[
\text{cl}(e1) = \text{level1}, \text{cl}(e2) = \text{level3}, \text{cl}(e3) = \text{level0}, \text{cl}(e4) = \text{level2}, \text{cl}(e5) = \text{level0}, \text{cl}(e6) = \text{level1}, \text{cl}(e7) = \text{level2} /* classifications of initial entities */
\]

**m(e1, e6) = \{\text{read, write}\}, m(e4, e7) = \{\text{write}\} /* initial matrix values */

**End** MLS.

However, the main challenge with security policy verification is that many interesting security properties which we want to prove, e.g., safety and information flow security, need specific security models. Security models curtail or structure the policy requirements specification so that the set of theorems associated with the model may be (semi)-automatically proved for the policy.

Our goal with the current policy type declarations is to tackle information flow security. To this end, in (Bryce 1997b) we present an algorithm permitting policies with anti-symmetry, reflexivity and transitivity exceptions to be
modelling using lattices, and hence the specification of these policies using the lattice-oriented type definition language.

The most important aspect for security policy engineering is verification reuse. That is, we want to be able to replace custodians following updates to policy requirements, and also to reuse units from the library in new custodians (using inheritance), not just to save design time, but to reduce correctness proof requirements by reusing what we have already proven of the included custodian components. Combination is also needed for the assignment of a custodian to a DDB data item in Cwasar since one item may be derived from different GDB Documents; the security policy for the new data item is some combination of the Documents. Comparing and combining policies necessitates the assignment of values to policies, so that an interpretation can be given to policy replacement (‘how does this new policy differ from the old?’) and construction (‘when these two policies are added, what is the sum?’).

For the access control security model that we have been working on, the output of the verification reuse work is also described in (Bryce 1997b), and consists of an algebra of policies \( \text{POL, \ AND, \ OR, \ NOT, \ \equiv, \ \leq} \). \text{POL} is the set of policies that can be expressed using the type definition language. The AND of two policies, \( P_1 \text{ AND} P_2 \), is a policy permitting an application operation to execute in a state if both \( P_1 \) and \( P_2 \) policies permit it to execute. An OR of two policies permits an application operation if either policy component permits it. A homomorphism exists from this policy structure to Boolean algebra; e.g., \( P_1 \text{ AND} P_2 \equiv \text{NOT(} \text{NOT} P_1 \text{ OR} \text{ NOT} P_2 \text{)} \), thus giving a precise meaning to policy construction. Policy comparison through the ordering \( \leq \) and equality \( \equiv \) operators is a function of the set of access states of policies. Policies \( P_1 \) and \( P_2 \) are equal \( P_1 \equiv P_2 \) if both permit the same set of application operations to execute. \( P_1 \) is inferior to \( P_2 \) \( P_1 \leq P_2 \) if every access state of \( P_1 \) has a sister state of \( P_2 \) where the latter state permits at least those operations to execute that are permitted in \( P_1 \). The importance of the ordering operator \( \leq \) is that it tells us when we can safely use one policy from the library database in place of another (security reuse correctness criteria). That is, when \( P_1 \leq P_2 \), we may use \( P_1 \) in place of \( P_2 \) without any verification being necessary since theorems proved of \( P_2 \) (with respect to permitted accesses to information in objects) also hold for \( P_1 \).

The formal work described in this section is for one class of security model. But we believe in the need for this algebraic approach to policy development for other models so that a much wider class of policies may be engineered into custodians.

4 CONCLUSIONS

This paper has outlined a security framework for wide-area telematics applications. The key requirement for a framework is that it permit the development
An approach to security for world-wide applications

(engineering] and enforcement of an expressive range of security policies. Our architecture is based on custodians which are software black boxes inserted into the architecture. Policy engineering necessitated the design of a library with support for policy component reuse at the design and verification levels. The framework in being exploited in the Cw asar platform. In this paper we could only overview many aspects of our work: More information on the security architecture can be found in (Kühnhauser 1995), on security engineering in (Bryce 1997b, Bryce 1997a), and on Cw asar in (Bryce et al. 1997).

Acknowledgments This work is partially supported by the European Commission Telematics program under contract number 1005. Cw asar is a joint venture of Bull S.A. (France), CitiusNet (France), DerbI (France), ECO GmbH (Germany), Effort S.L. (Spain), and GMD - Forschungszentrum Informationstechnik GmbH (Germany).

REFERENCES