Global Systems and Security

David Bentley, Alexandra Boldyreva, Holly Dail, Andreas Yannakopoulos

May 17, 1999

Abstract: A global or metacomputing operating system is a collection of geographically distributed resources (people, computers, devices, databases) connected by one or more high-speed networks and potentially spanning multiple administrative domains. Security is an essential part in the design of such a distributed system. A broad range of security needs, such as authentication, authorization, communication privacy and integrity, autonomy, intrusion detection and integration of existing security standards should be satisfied. In the following survey we examine how these needs are met (or not met) by several global operating systems.

1 Introduction

Global or metacomputing operating systems are platforms for combining very large collections of independently administered machines into a single, coherent environment. Like traditional operating systems, global systems build on a diverse set of lower level resources to provide convenient user abstractions, services, and policy enforcement mechanisms. Unlike traditional operating systems, in global systems resources may consist of thousands of heterogeneous processors, storage systems, databases, legacy codes, and user objects, all distributed over wide-area networks spanning multiple administrative domains. Global systems provide the means to pull these scattered components together into a single system that accommodates high degrees of flexibility and site autonomy.

A fundamental capability of global systems is scalability and the ability to incorporate resources from multiple trust domains. As such, the typical operating system paradigm of a central kernel or trusted code base that can monitor and control all interactions between users and resources is not applicable. For the same reasons, there can be no dependence on the concept of a superuser as no single person or entity can control all the resources in such a system. Further, security implementations in a metasystem must maintain the original goals of the system; in other words providing security must not significantly impact the usability or performance of a metasystem. The security requirements should therefore be considered as part of the metasystem design process. Unfortunately, attempting to incorporate security as an add-on late in the implementation process has been problematic in a number of first-generation metacomputing systems.

To satisfy the broad range of security needs, a well-designed metacomputing architecture must facilitate implementation of a number of different security features. The main requirements can be categorized as follows:

1. **Authentication** The ability to assert and confirm identity is essential for access control, intrusion detection, and secure communications.

2. **Authorization** Authorization is the process of determining whether a client should have access to an object or resource. Authorization mechanisms are required in metasystems to protect resources from access by unauthorized users.

3. **Communication integrity and privacy** Communication in metasystems often requires an insecure network to bind the metasystem components together. Privacy ensures that communication contents are protected while integrity specifies that communications are unmodified during transit. Encryption mechanisms are often required to ensure privacy and integrity of these communications.

4. **Autonomy** Metasystems must allow organizations to control local security policies but must still maintain the overall security of the system.

5. **Intrusion detection and recovery** A metacomputing environment should support mechanisms for detecting intrusion and misuse of resources, and for recovering after a security breach. This feature is particularly important in global systems, where we must generally assume that at least some of the underlying hosts have been compromised or may even be malicious.

6. **Integration of existing standards** Existing security standards, such as Kerberos and SSL, need to be integrated into the metacomputing environment

to satisfy local administrative policy and to handle legacy software.

At minimum, the secure function of a metasystem requires robust mechanisms for **authentication**, **authorization** and **communication integrity**. Depending on the needs of a system some security features such as communication privacy may be omitted. A well-designed system should implement an appropriate balance between security needs and other system requirements such as performance.

The focus of this paper is on treatment of the above security needs in modern metasystems. We begin in Section 2 with a general introduction to several modern metasystems. We continue in Section 3 by providing an introduction to the underlying security mechanisms currently available as enabling mechanisms for metasystem design. Section 4 is a detailed presentation and comparison of how modern metasystems address the important security issues. In section 5 we examine the CORBA approach to the global security issues. Finally in section 6 we evaluate the main existing Global Systems. Section 7 provides our view of future directions in this field.

2 Systems Overview

2.1 Legion

Legion [FKH+98] [WWK95], developed at University of Virginia, is an architectural model for the metacomputing environment, implemented in the Mentat Programming Language (MPL), an object-safe language. In the Legion model, all entities of interest within the system (processing resources, storage, users, etc) are represented by independent, active objects. Legion objects communicate via asynchronous method invocations, supported by an underlying message passing system. For the purpose of communication, every object is identified by a unique, location-independent Legion Object Identifier, or **LOID**. To facilitate message passing in Legion, LOIDs are bound to low-level object addresses (typically an IP address plus port number). Each Legion object belongs to a class of objects. The most important Legion object classes are **Host Objects**, **Vault Objects**, and **Class Manager Objects**. Host Objects represent processing resources, Vault Objects represent stable storage available within the system for containing the state of objects and Class Managers serve as location authorities. Currently, Legion is publicly available and widely deployed on hundreds of machines at dozens of sites spanning multiple trust domains.

2.2 WebOS

WebOS [VAD+98], developed at Berkeley, tries to simplify the development of wide-area applications while providing efficient global resource utilization. Some of the interesting mechanisms employed by WebOS include Smart Clients and WebFS. **Smart Clients** enable extensions of server functionality to be loaded onto the client machine, thus reducing server load. **WebFS**, a global cache-coherent file system, enables caching for performance improvements.

CRISIS [BVAD98] is the security subsystem of WebOS. It emphasizes a number of design principles for highly secure distributed system: **redundancy** to eliminate single points of attack, fine-grained **capabilities** to enable lightweight control of privilege, and **audit** to make each access decision. These principles are the basis for implementing the CRISIS security protocols. CRISIS is implemented only in Solaris, but designers plan to port it to other platforms in the near future.

2.3 Globus

Globus [FCTT98] [FKK+97], developed at USC, provides a set of services typically called the **Globus Toolkit** and application developers are meant to select from these services, making use only of those needed. The Globus Toolkit consists of: **GRAM** to provide resource allocation and process management, **Nexus** to provide unicast and multicast communication services, **MDS** to provide distributed access to structure and state information, **GSI** to provide authentication and related services, **HBM** to provide monitoring of the status of system components, **GASS** to provide remote access to data via sequential and parallel interfaces and **GEM** to provide executable management.

In addition, Globus attempts to provide mechanisms by which tools and applications can discover and control aspects of the underlying system; managing heterogeneity, instead of hiding it, is encouraged. In this highly heterogeneous environment, local authentication is assisted by a mapping of the user's Globus identity to local user identities. As of 1998 Globus has been deployed at 17 sites with more than 330 computers in four countries and the major components of the toolkit were either completely or partially in use.

2.4 Emerald

EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [PN97] is not a Global System, but an extremely successful project that focuses on the application of intrusion detection mechanisms to wide-area distributed systems. It introduces a highly distributed, building-block approach to network surveillance, attack isolation and automated response. EMERALD targets agents who attempt to subvert or bypass network interfaces and controls to gain unauthorized access to resources. It also provides a framework for applying the results from its distributed analysis to provide global detection of and response to network-wide coordinated attacks.

3 Security Mechanisms

Security implementations in metacomputing environments require a variety of services to deal with the security problems. A discussion of the commonly available security mechanisms, mostly based on [FK98] [SP99], is necessary before discussion of their application in metasystems. A reader familiar with modern cryptography and security systems may wish to skip this section.

3.1 Cryptography

Cryptographic methods are fundamental to the secure use of distributed systems. The open networks used for globally distributed systems introduce many security risks because anyone connected to the network may participate in communications by adding or removing information. In general, encryption employs an **encryption key** to scramble data to a form called **ciphertext** such that only the appropriate **decryption key** will allow decryption of the message into the original form, or the **plaintext**. Many algorithms exist for encoding plaintext, but two general classes can be identified.

3.2 Symmetric Cryptography

Symmetric or **private** key cryptography, also called standard or conventional cryptography, employs a single encryption key that is used by both the sender and the receiver of the encrypted data. The encryption key in this case is a parameter to an algorithm used to transform the data. A similar transformation using the same key is performed by the receiver of the ciphertext. Encryption and decryption using symmetric systems is generally relatively fast, but each pair of communicating entities needs a different key and both parties to the communication must maintain the key. This problem is often addressed by the use of a mutually trusted third party, usually called a **key-distribution center (KDC)**. A distribution key is required for each user of a KDC to provide secure transfer of secret keys. In response to a request for a secret key the KDC generates a session key that is sent in a secure fashion (by use of the distribution keys) to the communicating parties. An example of a symmetric cryptography scheme is **Kerberos**, a widely used authentication and key distribution protocol.

3.3 Asymmetric Cryptography

In **public** or asymmetric key cryptography the encryption and decryption processes employ different keys as parameters to their algorithms. The **public** key used for encryption is widely available, while the key used for decryption is called the **private** key and is kept only by the intended recipient of the ciphertext. Encryption and decryption is much slower using available asymmetric methods than available symmetric methods. However, key distribution and maintenance is much simpler as each server need only maintain a private key while any potential user can access the public key. Although, as in symmetric systems, a third party is often needed to certify public keys. Mechanisms for public key cryptography include RSA and Digital Signatures Algorithm (DSA), while deployed public key systems include Pretty Good Privacy (PGP) and Lotus Notes. The secure socket layer (SSL) protocol is also used in conjunction with server certificates to provide encrypted, one-way, authenticated communication paths.

3.4 Message Digests and Digital Signatures

A common application of the above systems in communications employs asymmetric encryption to encode a symmetric encryption key and a **checksum**. The symmetric key and checksum are used to protect the actual message data and are protected by asymmetric encryption. This combination provides the security of asymmetric encryption for the most vital data and the efficiency of symmetric encryption for the larger volume of message data. An alternative to the checksum is a **message digest**; a one-way function calculated on the message and designed such that given a message and a message digest it is computationally infeasible to find a different message that provides the same digest. A **digital signature** can also be created using public key cryptography; the sender signs the message using the private key and the receiver then verifies the sender's identity using the public key. An alternative mechanism for authentication involves a challenge/response protocol; one party chooses a random number and sends it to the other for encryption with a private key. The proper encryption can only be performed by the owner of the appropriate private key.

3.5 Certification

In order to maintain integrity in communications, one usually must associate an encryption key with an entity whose identity has been previously verified, called a **principal**. If this binding is not secure, an attacker could provide a false public key for which the attacker has the private key. A trusted third party, or **Certification Authority (CA)**, is usually employed to provide the binding between encryption key and principal. The Certification Authority issues **certificates**, objects that associate a principal with its characteristics. For public-key systems this includes the public key of the principal, but in general it may also include authorizations, group memberships, email addresses, or alternate names. The certificate is sent to the requester with the signature of the CA. The signature is verified with the CA's public key, generally obtained from a hierarchical structure of CA's. An important safety mechanism for certificates is the availability of certificate **revocation lists**, so that certificates can be revoked in case of security problems or changes in status. The most common certificate format is specified by the X.509 format. Commercial providers of key distribution services include VeriSign, CyberTrust, and CertCo.

3.6 Existing Standards

SDSI (Secure Distributed Security Infrastructure) [RL96] is a 'key-centric' approach; it treats all identities as having only local significance. Creators of messages sign their own certificates. SDSI certificates can be **name** certificates that attach names to the public keys of people you know, or **delegation** certificates that give others permission to do some of the things you are allowed to do.

Kerberos [FK98] is an authentication and key distribution protocol. In order to communicate with a service provider the user contacts the Kerberos authentication server which provides a session key and a ticket, each of which times-out. To authenticate itself the user sends forward the ticket encrypted with the session key to the server. The service provider then can verify the identity and validity of request. Kerberos has proven quite successful for local area networks.

SSL (Secure Sockets Layer) [FK98] is a widely used technology for confidentiality on the Internet. It also can be used in any distributed system for authentication of servers and clients if they have a certified public key, and for the protection of the communication in the general case.

4 Addressing the Security Problems

4.1 Authentication

Authentication is the process of verifying the identity of a participant in an operation or request and may be required for any entity in a metasystem, including users, servers, and other gateways to resources. Authentication in metasystems is sometimes complicated by the need for mutual authentication. Not only does a server need to authenticate a user's identity to assure that resources are not provided to non-authorized users, but a user also must authenticate a server to assure that data and resources received can be trusted. Authentication mechanisms are also complicated by the need to allow **delegation** of identity, or the granting of authority to assume the identity of a different principle for completion of some task.

In **Legion**, identity is based most naturally on **LOIDs** (unique low-level identifiers), since all entities of interest (including users) are represented as Legion objects. Each object is assigned a LOID by its Class Manager when it is created, though there are some exceptions, e.g. command-line programs create their own LOIDs. As a default Legion security practice, one of the LOID fields contains security information, including an RSA public key. Actually, the LOID security field is not just an RSA public key, but an X.509 certificate.

The X.509 certificate is signed by a certification authority (CA) that vouches for the association of the key with the object's identity. Once a certificate is signed by a CA, it is propagated with requests and method calls for authentication of the object. To cover the case where a recipient doesn't recognize the CA, the CA's own certificate can be chained onto the certificate, allowing the CA's CA to be the basis of authority. The chain can have multiple links, each link leading to a higher authority CA. A recipient validates a certificate by traversing the chain until it reaches a CA it recognizes and checks that all intervening certificates are properly signed. However Legion certificates do not time-out and are vulnerable to theft or abuse.

CRISIS, the security subsystem of WebOS, uses a similar authentication scheme to Legion. CRISIS users have their public keys signed by a CA. The signed certificates, called **identity** certificates, are attached to all future messages of the user. To determine whether a request for a particular operation should be authorized, the security manager of the recipient verifies that the message certificate is signed by a public key with an endorsement from a trusted CA. With this knowledge, the security manager reduces all certificates to the identity of single principals.

Unlike Legion certificates that do not time-out, CRI-

SIS certificates are valid only for a period of time. The CA identifies a locally trusted on-line agent (OLA) responsible for re-signing timed-out certificates. So the period of time when the certificate is vulnerable to attack is limited.

Globus uses a different approach for authentication, based on proxies. A special process called a **user proxy** is created by the user on his local Globus host and is given permission to act on behalf of the user for authentication purposes. The user proxy, rather than using the user certificate for access, creates a temporary certificate which contains the user's host and its duration. This time limit on the temporary certificate protects it against attack and, even if it is compromised, the user certificate remains protected. The authentication required to gain access to a resource is now simply carried out by the user proxy with the temporary certificate. To enable authentication on the resource side, a **resource proxy** maps global to local identities. The resource proxy is responsible for scheduling access to a resource and for beginning computation on that resource.

The sequence of events required to establish access to a new resource will help show how these proxies work within Globus. When a user process requires access to a new resource, it sends a request to the user proxy. Since the user proxy and requesting process may be in different domains they must authenticate each other using their certificates. The user proxy will then communicate the request to the resource proxy and will provide the user certificate for authentication purposes. The global user identity is mapped to a local authentication mechanism by the resource proxy. If the user process is authorized to obtain the requested resource, a certificate is signed by the resource proxy, sent to the user proxy, signed by the user proxy and passed back to the resource proxy. This double-signed certificate is then passed on to any processes started on that resource by the user. If these new processes require authentication in future operations they will use this certificate.

4.2 Authorization

Authorization is the process of determining whether a particular client may use or access a resource and is usually based on a match between the authenticated identity of the requester and some notion of who should have access to a resource. Traditionally, both **local access lists** and **security managers** have been used to describe the set of principals authorized to access a particular object. In metasystems, there are large numbers of users to create and maintain so separate accounts and access lists may be prohibitive. On the other hand, systems that depend on a security manager for access control provide an obvious target for a security attack.

In Global Systems, many applications can benefit from a mechanism that supports delegation of authority. **Delegation of authority** is a means by which a user or process authorized to perform an operation can grant that authority to another process. It is a more restricted form of delegation than delegation of identity in authentication.

Legion follows the local access list model. As Legion is an object-oriented system, authorization is defined as the ability to call a method on an object and each object is responsible for enforcing its own access control policy. Each method call received at an object passes through a **MayI layer** before being serviced. In this MayI layer, there are two access lists for each method, an allow and a deny. The LOIDs in the allow and deny access lists represent particular users, the object's Class Manager, groups or the object itself.

In order to support delegation of authority, Legion maintains two primary types of credentials: **delegated** and **bearer**. A delegated credential specifies exactly who is granted the listed rights, whereas possession of a bearer credential grants the rights listed within it. Bearer credentials are especially prone to security attacks, as they grant resource access to their holder. To prevent such an attack, bearer credentials time-out and expired ones can be refreshed by contacting special **refresh** objects at the issuer.

CRISIS uses the security manager approach. As all programs execute in the context of a security domain, each node runs a security manager responsible for mediating access to all local resources. A process requests access to an object by contacting the object's security manager. If the request passes this test, it is then checked against the contents of the object's local access list and is granted authorization if a match is found. CRISIS uses a mechanism similar to Legion for delegation of authority. A transfer certificate is used to transfer a subset of a principal's privileges to another principal. This transfer certificate corresponds to Legion's delegated credential.

Globus follows a hybrid approach. It provides authorization at the system access level, but local authorization is provided by the security system of each administrative domain. Access to the local security system of the resource requires a series of steps between a user process, the user proxy and the resource proxy, as described in the **authentication** section.

4.3 Communication Integrity and Privacy

Proper function of a metasystem requires that communications remain unaltered during transfer between nodes. Encryption of the entire message can be very expensive so the required **integrity** of communications is typically realized by the calculation of a message digest, which is then encrypted and transferred with the message. If the data consumed or produced by remote tasks is sensitive, **privacy** of the communication will also be necessary.

Legion provides encryption and integrity services at the level of messages. A message may be sent with no security, in private mode, or in protected mode. Private mode provides for encryption of the message body using RSA encryption and is used for the case in which privacy of the message body is required. Protected mode employs a message digest to guarantee message integrity.

Though integrity of communication is assured by message digests, the attacker can still attempt to replay a message. Replay of messages can be dangerous because without additional control they will be accepted as valid requests and can be used to compromise services. To prevent replay of messages, Legion includes a large random number and a timestamp in each message. The timestamp is used to determine if the message is outdated (based on the recipient's time-checking window). When a new message arrives, the recipient logs the random number for a time period based on the message's timestamp and the recipient's time-checking window. If another message arrives with the same random number during a so-called log period, it is rejected. In essence, the message timestamps allow coarse-grained detection of replays, while the random numbers provide the fine grain.

To achieve secure communication **CRISIS** also employs encryption. CRISIS designers chose public key over private key scheme, both because of the synchronous communication usually required by private key systems and because of the overhead associated with third party communication. Public key systems also require trusted third parties to produce certificates identifying principals with their public keys. However, as described in the **authentication** section, these certificates remove the need for synchronous communication with a third party to set up a communication channel.

Globus uses a modified version of the **Nexus** library for its communication [FKK+97]. The basic idea is that for metasystems a detailed security policy cannot be decided in advance and a single communication protection scheme is not going to be sufficient. For example, communication between nodes of the same computer remains internal and so generally need not be encrypted. On the other hand, if the communication were to be sent to a node of another computer, possibly located anywhere in the world, the message would need to be encrypted to guarantee security. Truly secure encryption schemes are expensive and should be used only when necessary. So Globus leaves the choice of security implementation flexible and provides instead a consistent mechanism for applying the chosen security implementation.

Specifically, in Globus a communication link is formed by binding two points of communication, a startpoint and an endpoint. In order to perform transformations on the data sent along in a remote service request, a set of transform functions called a **transform** module may be associated with each communication link. The startpoint transform function is called before data is sent along a communication link. It may modify the data being sent as well as the header information or the security state associated with each link. The endpoint transform function is invoked on data received at an endpoint and can also modify header or message information. Associated with each pair of points is a security context. Security contexts consist of two parts, the **configuration** and the **state**. The configuration describes the security mechanism chosen for the communication link, while the state is any values needed to implement the security policy laid out in the security configuration, e.g. keys for encryption algorithms. These facilities determine exactly how each possible communication path will handle protection and allow a user-defined protection mechanism to be associated with each communication link created.

4.4 Autonomy

Global operating systems must by definition incorporate the heterogeneous security policies and mechanisms of existing systems. It is critical to ensure that the global security mechanisms cannot be used to subvert the local security mechanisms. It is also critical that malicious processes are isolated and detected. The local system administrator is generally concerned with who can create processes on his system via the global system and what those processes can do. A design relying on deference to a global, centralized authority is not only less flexible, but less likely to be widely adopted.

In **Legion**, local protection is provided by treating each domain as a separate entity. Within each domain Legion protection mechanisms are used to ensure that critical resources are not passed out of the domain and that unauthorized callers will not be able to invoke methods on local critical resources. This intrusive approach to isolation may be unacceptable at some sites, because it uses the Legion mechanisms to control access to a host.

CRISIS is more intrusive than other metasystems and also provides a more comprehensive security implementation. It cannot accommodate local security mechanisms unless they follow the CRISIS model. But the existence of different local policies is encouraged **within this model**, by giving each domain the responsibility for determining the level of trust placed in any given remote domain. Also CRISIS does not support development of new policies for new security mechanisms as they are added to WebOS and existing security policies cannot be modified.

Within **Globus**, general state about users, security protocols, and other dynamic properties is maintained. When an autonomous local site must be accessed, this generic global policy is mapped to the local policy for the applicable site. These mappings are maintained by the local domain administrator. In addition, Globus maintains a global, distributed service, which provides site-specific information, such as hardware configurations and performance. This facilitates not only using the correct local policies, but also dynamically choosing the most efficient methods where choices are available. For example, if a particular site has hardware support for a specific encryption mechanism you could choose to use that method when interacting with that site. The user can therefore choose the encryption mechanism and can dynamically change that policy depending on where and when the information is being communicated and what information is being communicated.

4.5 Intrusion Detection and Recovery

A detection mechanism is useful for tracking breaches of security in the system. Usually, an **audit** function records operations that have been performed by a system, associating each action with the principal on whose behalf the operation was performed. The detection system will look at events generated by the function audit to find patterns of operations that fit the profile of a system intrusion or that do not fit the profiles of legitimate users. If such patterns are detected, an alert is generated. In traditional systems, the audit function is local to each server. However in global systems, the audit function should be **distributed** in order to detect network attacks.

Legion uses an event handler scheme to detect intrusion. The core objects managing the local site can be started and configured only by Admin, resulting in no external trust dependencies on outside systems. While invoking methods with bearer credentials, Admin must be sure that no messages are sent to off-site objects. If a message is inadvertently directed off-site that contains Admin's credentials, the message is blocked and the event handler raises an exception. So off-site objects will neither be able to generate nor steal Admin's credentials.

The detection of intrusion can be done in CRISIS by checking if a certificate has expired and notifying the resource security manager. The certificate may be well-formed and valid, but all CRISIS certificates time-out. Therefore if an attacker steals the certificate, he will rarely succeed in using it before it expires.

Globus protects the user by using temporary certificates for authentication. So in the event of a security breach, permanent user certificates are not compromised and, as in CRISIS, the temporary ones eventually time-out. In addition, local resources are protected against a security breach, by denying access to local resources for users without a local account.

Unlike the Legion event handler mechanism or the CRISIS and Globus time-out scheme, **EMERALD** proposes a complete system of independent monitors to detect intrusion attacks. Monitors are deployed throughout each domain to analyze the operation of the network and provide localized protection of key assets. Each monitor includes a set of response handlers that are invoked if it detects possible misuse. Finally, a system-wide analysis correlates the activity reports produced across the set of monitored domains. Thus, EMERALD not only detects local attacks, but also coordinated attacks such as distributed denials of service or repeated patterns of attack.

4.6 Integration of Existing Standards

In **Legion**, a site may use an existing standard and require its integration. For example, only users with local accounts may access a resource, and those users must be authenticated by a locally adopted authentication protocol, such as **Kerberos**. The implicit use of tickets is the basis for employing Kerberos as an authentication mechanism within Legion. The ticket is sent in the implicit parameters of an object creation request to the eventual Host Object. The Host Object uses the ticket it receives to request a new ticket from the local Kerberos server. The Host ensures that only accounts of users authenticated via Kerberos are used. Once authentication succeeds, though, normal Legion objects can be created.

The Legion architecture is also flexible enough to accommodate **firewalls** with ease. Objects started on hosts behind a firewall automatically have a Proxy Object on the firewall host assigned to them by their Host Object. The Proxy Object then acts as a simple reflector forwarding any received messages to their intended destinations behind the firewall.

CRISIS integrates a number of other technologies. It uses Janus to 'sandbox' locally running applications that are not fully trusted. Janus runs at user-level, employs UNIX to intercept potentially dangerous system calls and to disallow accesses outside of each process's defined sandbox.

CRISIS also employs the SSL protocol to provide network privacy and integrity of data, using encryption and message authentication codes. SSL supports a wide variety of cryptographic algorithms and is being deployed in wide-area applications.

Globus security functions have been coded in terms of the Generic Security Services API (GSS-API). The use of the GSS-API allows modification of the underlying security mechanisms without requiring any changes to Globus code. For example, during initial testing and deployment Globus used a simple plaintext password system for authentication. As deployment became more widespread, the entire system was converted over to use the SSL protocol without ever changing a line of Globus code.

In addition, work is being done on a Kerberos-based security system. The flexibility is such that not only does Globus support different protection mechanisms, but it supports a negotiation protocol so that the most effective system can be determined at runtime, dynamically by agreement of the two parties involved.

5 The CORBA Approach

Many of the goals of the **CORBA** security model are similar to the goals of Global Systems, including simplicity, scalability, usability, and flexibility. The CORBA security model encompasses authentication, authorization, auditing, security of communication, non-repudiation, and security information administration. Typically, a vendor implements CORBA security using existing technology such as GSSAPI, Kerberos and SESAME. Also a recent extension to CORBA called **PARDIS** [KG97] adds the ability to handle multiple threads of execution running in separate address spaces as a single CORBA object. This allows the creation of a complex parallel service running on separate heterogeneous domains that is still capable of being invoked simply through a standard CORBA Object Request Broker. However, CORBA is not a metacomputing system; it does not construct an operating system-like environment using underlying distributed resources. Given this fundamental difference in target use, CORBA does not actually address the metacomputing security problem.

6 Evaluation of Global Systems

In terms of **design principles**, Legion and CRISIS are quite similar, but there are still differences. Legion focuses on object-oriented software approaches. This provides a clean and flexible interface, but requires the use of an object-oriented design methodology. However, Legion provides a rather high-level security model without describing architecture and specific protocols. CRISIS uses a similar model to Legion, but seems to be a more complete security system. It defines specific protocols, which can be used as one implementation of security for Legion objects as well. On the other hand, Globus is based on a set of library routines which can be used selectively by the programmer. This also allows applications to be made "grid aware" one step at a time, simplifying the porting of legacy software.

Another major difference between the systems is in their approach towards the security **autonomy** of local domains. Legion and Globus have as a primary goal the preservation of such autonomy. To that end both systems provide a mechanisms for transparently dealing with different, possibly incompatible, local security implementations. On the other hand, CRISIS, while still allowing local control over security policy, doesn't allow local administrators to choose the security mechanism used.

However CRISIS is superior to either Globus or Legion in terms of **robustness**. CRISIS has mechanisms for caching credentials for performance as well as timing-out identity certificates for security. In addition, it avoids the use of the Legion bearer certificates which, if compromised, give to the attacker all the rights of the original owner. Also CRISIS emphasizes attention to the robustness of the security system by locally replicating the CA.

Currently all three systems use similar **low-level protocols** to implement their security policies. These include X.509 certificates for authentication and the Secure Socket Layer (SSL) for message protection. However the way in which these standards are used is different. Both Legion and CRISIS hard-code those mechanisms into the security architecture, thus making incorporation of new standards difficult. By contrast, Globus codes all security functions in terms the GSS-API and by changing the underlying mechanism, you can easily change the Globus security system. The trade-off is that the GSS-API does not provide all the necessary security functions. Most notably absent are a secure group communication facility and a simple delegation facility.

In terms of **implementation**, Globus has been implemented and is globally accessible. CRISIS has been implemented only on Solaris and its design principles and performance measurements have not be tested and proven by large-scale deployment. Performance measurements from both the CRISIS prototype and Globus indicate that protected communication can be done with little overhead from the security system. Unfortunately, Legion performance results are not provided, so we cannot compare it to the other systems.

7 Future Directions

Perhaps the most obvious direction for immediate future work is in further testing and deployment of the existing systems. Even in Global Systems, such as Globus, which have been fairly extensively installed, there are still many unanswered questions about scalability. Many potential bottlenecks and limitations of these systems cannot be discovered except through real-world experience. An excellent summary of more long-term directions for future research is provided in [FK98].

7.1 Group Communication

In large distributed computations, individual processes must communicate their results with other processes. This problem is complicated in metacomputing environments, as communications must cross multiple, individually administered security domains. In addition, travel across insecure networks will often be unavoidable. In these situations, data integrity and privacy must be ensured, but all existing mechanisms have major shortcomings.

Ideally, the data would be encrypted once, then sent to all processes which needed it. This could be done by providing a single secret key to each process. Unfortunately, either all processes must be trusted, or a separate pair of secret keys must be maintained for every pair of processes. The latter solution means that multicast can no longer be used and each message must be encrypted and sent individually. Alternatively, asymmetric cryptography allows the use of multicast, there are still two expensive encryption/decryption operations necessary for every message.

7.2 Resource Accounting

Another feature lacking in existing metasystems is the ability to globally track all resources used during a computation and bill them to a single account. Currently all accounting of resource usage is done by the local security domains at each site. This require the user to have a local account on each system at which he wishes to allocate resources.

A better strategy would be to maintain a global account for each user. Then a pool of temporary accounts at each site would be used to provide access to users of that resource. This would simplify administration of user accounts by centralizing all billing information, e.g. CPU cycles, and tracking it in a uniform way.

7.3 Group Administration

Authorization mechanisms in current generation metasystems are somewhat simplistic. Presently the most popular method of authorization is the maintenance of an access control list for a specific resource. This makes large scale modifications tedious because authorization must be changed for each user at each site. A higher-level access control abstraction such as group membership would greatly simplify administration of user rights.

References

[BVAD98] Eshwar Belani, Amin Vahdat, Thomas Anderson, and Michael Dahlin. The crisis wide area security architecture. In *Proceedings of USENIX Security Symposium*, 1998.

[FCTT98] I. Foster, C.Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security Conference*, 1998.

[FK98] Ian Foster and Carl Kesselman. The grid - blueprint for a new computing infrastructure. morgan kaufmann press, 1998. Book.

[FKH+98] Adam Ferrari, Frederick Knabe, Marty Humphrey, Steve Chapin, and Andrew Grimshaw. A flexible security system for metacomputing environments, 1998. Technical Report.

[FKK+97] I. Foster, N. Karonis, C. Kesselman, G. Koening, and S. Tuecke. A secure communication infrastructure for high performance distributed computing. In *6th IEEE Symposium on High-Performance Distributed Computing*, 1997.

[KG97] Katarzyna Keahey and Dennis Gannon. Pardis: Corba-based architecture for application-level parallel distributed computation. In *Proceedings of Supercomputing 1997*, 1997.

[PN97] Phillip A. Porras and Peter G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of National Information Systems Security Conference*, 1997.

[RL96] Ronald L. Rivest and Butler Lampson. Sdsi- a simple distributed security infrastructure, 1996. Working Document.

[SP99] Computer Science and Telecommunications Board. National Research Council. National Academy Press. Trust in cyberspace, 1999. Book.

[VAD+98] Amin Vahdat, Thomas Anderson, Michael Dahlin, Eshwar Belani, David Culler, Paul Eastham, and Chad Yoshikawa. Webos: Operating system services for wide area applications. In *Proceedings of High Performance Distributed Computing*, 1998.

[WWK95] Wm Wulf, Chenxi Wang, and Darell Kienzle. A new model of security for distributed systems, 1995. Technical Report.