Security in Network Computing

José Miguel-Alonso

Summary—**Security in a network computing systems is a multifaceted problem, because risks are multiple: malicious code, intrusion by non-authorized users, sensitive information traversing a non-protected network, etc. In this paper we analyze security risks of such systems and propose some solutions. We use Purdue's PUNCH as the target system to illustrate the proposals.**
Keywords—**Network computing, PUNCH, security**

I. Introduction: network computing and security

These days nobody doubts that the World Wide Web has become a real, worldwide network of information resources, the place to search for all kinds of data. The WWW has been the "killer application" that has pushed ahead the growth of the Internet. Now that this infrastructure is available, new uses for it are possible. One of those uses is to perform computations over the data. The Internet or, in general, any computer network, can be seen as a heterogeneous computing system, full of available and, in many cases, underutilized resources that can be harnessed in a more productive way.

A NCS (Network Computing System) can be described as a virtual system on top of the Internet infrastructure, in the same way the WWW is a virtual system. Any person with Internet connection can be a potential user of the NCS, and any computer connected to the Internet can be part of the NCS, and allow CPU cycles (in addition to stored information) to be accessed by the users. Considering the current size of the Internet, it is possible to envision a system with millions of users and computing servers.

Size is not the only challenge of a NCS. Others are heterogeneity, and the dynamic nature of the system: computers are continuously added to or removed from the network, can be down for maintenance, or can be up but unavailable for external users. Also, sharing computing resources is not as simple as sharing just data by means of a WWW server. An organization willing to add a computing node to a NCS should have control of how this node is used, and by whom. That is, it will define a usage policy (that includes a *security* policy) and it is the responsibility of the NCS infrastructure to enforce that policy. A large scale NCS would comprise a collection of *domains* with disparate policies.

Departamento de Arquitectura y Tecnología de Computadores, Universidad del País Vasco (UPV/EHU). Apdo. 649, 20080 San Sebastián. Correo electrónico: miguel@si.ehu.es.
Trabajo realizado con financiación de la CICYT (TIC98-1162-C02-02). La estancia en Purdue University durante la cual se ha realizado este trabajo ha sido posible gracias al Prof. José A.B. Fortes (Departament of Electrical and Computer Engineering, Purdue University) y a la Secretaría de Estado de Universidades, Investigación y Desarrollo.

Users of a NCS have to consider the risks of using external resources to perform their computations. Data and programs will be put in someone else's hands, and will traverse an insecure network such as the current Internet. A NCS must include security mechanisms to protect the confidentiality and integrity of the users information.

The previous discussion has been purposefully general. There are currently many initiatives to build NCS. All of them are designed to use distributed computing resources, but this generic objective is tackled in many different ways. Here is not our intention to be exhaustive, so we will mention just a few, and then we will concentrate in one of them: PUNCH [KFF00]. The focus of some NCS such as Globus or Legion is in allowing programmers to use the full potential of the available, distributed computing power, by spreading the computation along many, heterogeneous nodes. Others such as VNC or Citrix Winframe are designed to allow a user to connect to a remote computing server while running a *remote display* in a local machine. PUNCH fits somewhere in the middle. Its main purpose is to allow a user to run programs that are available *somewhere* in the network, programs that do not need to be specifically designed to run in a NCS. These approaches are not really *alternatives*. In fact, they complement each other very well. A program can be designed using Globus, and then accessed by means of PUNCH. VNC is the enabling technology that allows PUNCH to give users access to graphically rich tools.

This document discusses security aspects of PUNCH, the security problems that may appear when running it in a large-scale setup, and the solutions that have been implemented or, in some cases, just proposed. The basic security problems that will be considered are:

- Authentication as the key for authorization.
- Data confidentiality and integrity while traversing the network.
- Protection of user's data, while stored and while being processed.
- Protection of underlying resources: computing nodes, network.

In [FK99], C. Neuman identifies a longer list of requirements for security in a large-scale NCS, that includes those already mentioned, plus assurance, accounting, audit, and selection of security services. PUNCH also considers these points, although not all of them will be discussed in this document. For example, PUNCH performs accounting of used resources and stores extensive auditing information in the form of log files. Regarding assurance and selection of security services, PUNCH selects on behalf of its users the

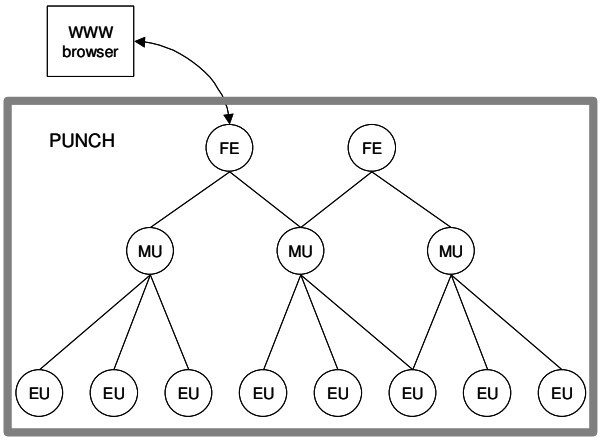
resources that best match their computational and security demands. As providing security is expensive, it tries to use the minimum level of security that guarantees users demands, without compromising the security of other users or of the infrastructure.

II. PUNCH

PUNCH (Purdue University Network Computing Hubs) is a network computing system currently deployed at Purdue University and other research and education centers. The main focus of PUNCH is to provide an infrastructure that allows people to run, from their desktops, a collection of tools already installed in the network nodes, with some basic premises:

- Location of the tool is transparent to the user
- Tools do not require to be modified in any way in order to be part of this infrastructure

The core system is designed around a three level hierarchically distributed architecture. Users access to the system via standard, unmodified web browsers, although the design allows for other kinds of accesses, for example, an enhanced shell that redirects some commands to PUNCH instead of executing them locally.



The network desktop interfaces, or front-ends, are the access points. They parse, interpret and act on user requests. This is the only level of the hierarchy that is visible to end-users. Management units act as scheduling engines for the hardware and software resources of the system. They also enforce resource-specific access control policies (e.g., licensing and ownership concerns). Functionally, a management unit analyzes the requirement of a given request and then matches those with appropriate resources from those available. Once the requirements are known, an applicable set of implementations (e.g. sequential, parallel) and platforms (a SUN workstation, a Condor pool, a vector processor) can be determined. At this point, the specific implementation and the execution unit to be used for the run can be forwarded to the selected execution unit.

The scalability of a NCS like PUNCH can be defined in terms of its ability to continue to perform well with an increasing number of users, software resources, hardware resources and administrative domains. In the case of PUNCH, scalability is achieved by replicating its components. At the top level, it is possible to offer several front-ends that present users with a consistent view of the resources, via shared or mirrored information.

Scalability with respect to users presents a logistical problem, because the management and execution units will generally be distributed across different geographical regions and administrative domains. It is impractical to expect all users to have physical accounts on all the resources available in the NCS. This problem has been addressed by:

- Partitioning the system so that the individual components of the NCS are treated as users with respect to each other, and
- Viewing an account as a logical entity rather than a physical one. With logical accounts, the allocation and partitioning of physical resources is done dynamically. With this approach, PUNCH is able to manage all of its users within a single physical account.

Scalability with respect to software resources is achieved by distributed them among an appropriate number of management units and, finally, scalability with respect to hardware resources and administrative domains can be achieved by allowing management units to forward requests to other management units. The hierarchy of management units also provides a mechanism that allows the NCS to be partitioned into independent cells.

A cell totally fits into an administrative domain, consists of at least a MU and a EU, and can be seen as a user by other cells. EUs only accept request from local MUs: a request from a remote cell has to pass through the local MU. A MU that accepts a request originating from a different cell authenticates the immediate sender and, optionally, the source of the request (the real user). Each cell can have its own set of usage and security policies, and these can be different for internal and for external users. In fact, the response to a request can be customized according to its source.

Now that the architecture of PUNCH has been described, the remaining sections will describe how security fits into the picture. The basic idea is that users want to protect their data and programs, and resource owners want to protect those resources against misuse.

This involves:

1. Security in the communication. How to protect data being accessed or altered while traversing the network. How to authenticate the partners involved in a communication.
2. Security in a computing node. How to protect resources from programs badly designed or intentionally malign. How to avoid resource usage by non-authorized users.

3. Security in the stored data. How to prevent users to access other user's data, intentionally or accidentally.

In this paper we will focus on point 1 (security in communications) and, partially, in point 2 (protecting resources).

III. Security in communications

The NCS can be seen as a collection of nodes that communicate through an insecure network, such as the Internet. Means are required to *authenticate* the communicating parties, and to ensure the *integrity* and the *confidentiality* of the transmitted data. Following the guidelines of [NASA99], we will consider already existing mechanisms to implement these security functions, instead of re-inventing the wheel. Quoting from this document, "This is generally more secure than creating a custom system, reduces implementation vulnerabilities and may facilitate scalability and compatibility". In [NASA99, BE00] a review some of them is performed: Kerberos, SSH and SSL. We do not consider them as "alternatives": sometimes it is required to combine several of them in order to accommodate different security policies, or to integrate legacy system into de NCS infrastructure.

A. User to NCS communication

The previous section described how users access PUNCH via web browsers, that is, using the HTTP protocol. In this environment, the standard approach to authentication, integrity and confidentiality is SSL, the Secure Socket Layer. SSL is a security framework in the sense that allows the use of a wide collection of cryptographic algorithms, which can be increased when necessary—so, it is not attached to a particular algorithm. There are several implementations available, commercial and public domain. Being an open specification, it has been exposed to public scrutiny—which provides evidence of suitability [WS96]. The most powerful reason to use it is that most popular web browsers already integrate SSL support. If an alternate method was to be used (such as an enhanced shell), SSL libraries are available to add the adequate support for this protocol.

A key piece of SSL is the use of X.509 certificates. A certificate is a piece of information that includes personal information (name, organization, e-mail address, etc.), cryptographic information (a public key) and some additional information (issue date, expiration date, etc.). A certificate may be signed by a Certification Authority (CA), which vouches for the association of the owner of the certificate and the information it contains. Trusting a CA means trusting the entities it endorses. Given the current size of the Internet, and its current growth rate, it is not realistic to think that a CA will be enough. Many of them already exist, some for organization's internal use; some others open to the general public. [BE00] provides some guidelines about how to cope with these complexities.

In an SSL connection through a web browser it is common to have the server authenticate to the client by sending the server certificate. If the client has a certificate, the server can request it to authenticate the client. We propose to use server authentication only. The PUNCH front-end could use a certificate obtained from a well-known Certification Authority (such as Verisign). This way, clients do not require installing a new CA certificate in their browsers and, thus, users could move more freely from machine to machine. Also, the risk of installed a non-safe CA certificate (opening the door to access non-trusted sites) is reduced.

Once a secure channel has been established between client and server, the client can authenticate by means of a <username, password> mechanism. One advantage of this approach is that it does not require an infrastructure to create and distribute client certificates. Another advantage is that allows a user (the person using the web browser) to move freely from machine to machine to use the system. Usually, certificates are stored in a browser's managed data structure, and if the user changes to another machine the process of transferring the certificate to the new one, although possible, is not immediate. If PUNCH is to be used only from an organization's intranet, and services such as floating profiles are provided, then the use of client certificates would be less of a burden. Additionally, users would not be required to type a username and a password to access the PUNCH infrastructure.

B. Internal communication

The modules that constitute the PUNCH infrastructure also need to communicate in a secure way. We start with the description of what is required in a given cell, which fits under a single administrative domain. Given this premise, and considering that users are not involved in this communication, there is no need to use a standard security mechanism. Any mechanism that fits into the local security policy, *and* fulfills the user's security demands, can be implemented and used. In fact, there could be more than one mechanism, and a given one could be selected as a function of the user's demands.

In the current implementation of PUNCH, front-end and management unit are not actually separated into different entities. For this reason, there are not communication links to be protected. The design allows for separation in different processes and different machines and, if this were the case, a secure channel would be required. Management unit and execution units are actually separated, and in most cases run in different machines. Communications are currently protected through SSH, which provides strong authentication and encryption. As an alternative, SSL with authentication of both partners could be used here.

When there are more cells, spanning through several administrative domains, solutions have to be more standard and scalable. The proposal here is to use SSL. Top-level management units in each cell must have an X.509 certificate from a well-known CA. Inter-cell communication can, thus, be authenticated (the two

MUs involved in the communication can authenticate to each other) and encrypted.

The basic idea behind these proposals is that only those elements that interact with users or external cells require certificates validated by an accepted certificate authority. Ideally, there should be just one CA for the whole NCS. This is unrealistic, because a site can obtain a certificate from many places. Mechanisms like those implemented in Globus [BE00] are required to deal with multiple CAs.

IV. Security in the computing servers

Most of the nodes of a NCS are there to run programs on behalf of authorized users. In principle, an arbitrary program can be run through the NCS—including programs that attempt unauthorized operations against other user's resources, against the host machine, or against other nodes. Local protection mechanisms (no home directory, a very restrictive path, no group membership, etc.) would be used to its full extent, trying to limit the range of possible attacks. However, OS provided mechanisms are not always enough.

In this section we will discuss how to protect processes that are running in a computing server from interference from other processes, and how to protect the underlying resources from malicious (or badly designed) software. Some requirements for the proposed solutions are that they need to be implemented in user space, without requiring PUNCH to obtain root privileges, and without modification of the host OS.

A basic starting point is to use to their fullest extent the security characteristics of the host operating systems. Store the user's data in such a way that file permissions are as restrictive as possible, run the programs with the minimum set of privileges. Some authors claim that this is not an adequate solution, because currently available operating systems are not secure enough [LS98]. However, the purpose of PUNCH is not to build a complete system for scratch, but to reuse existing software (applications, operating systems) and networks.

The execution units of PUNCH are the ones actually in charge of running tools on behalf of the users. A EU is a piece of software that can run on any TCP/IP reachable computer, plus a companion collection of scratch accounts that will be used to actually run the tools. The decision of using scratch accounts under the supervision of a single, "punch" user was made to reduce the administration nightmare of opening and maintaining accounts for each user in each available machine.

In its current status, PUNCH only runs those programs installed under the supervision of the PUNCH administrators. However, in a near future, users will be allowed to install their own tools, and to edit, compile and run their own programs inside the PUNCH infrastructure. This will increase the risk of running dangerous software that may try to access unauthorized resources, such as data from other users (that may be part of PUNCH, or may be normal users of the host machine) and system resources (passwords files, network connections, etc.)

PUNCH implements a series of mechanisms to access user's files from the scratch accounts. How this is achieved is not discussed in this document. Independently of the chosen mechanism (FTP transfer, NFS access, etc.) it is strongly recommended to use a secure underlying transport mechanism.

A running process may also try to access other resources, in addition to files. For example, it could try to open a telnet session to other host and initiate any kind of cracking activity. This misbehavior should not be allowed. In fact, a tool should do just what it is expected to do, and use just the minimum set of resources required to complete its function—that can be part of the tool's specification. The PUNCH system has to carefully monitor the tool's usage of resources.

What is required is a mechanism to confine untrusted applications into a restrictive "sandbox". This is not a need specific for NCS. It has been proposed for other scenarios, such as mobile code (Java applets), mobile agents, helper applications (plug-ins for browsers or MIME mail agents) and CGI scripts. We propose to integrate a system like this into the EUs of PUNCH. The sandbox has to be tailored to each application, although a default, very restrictive sandbox can be used for most tools. In [GW96] Janus, a sandboxing system to execute helper applications is proposed and implemented in the Solaris operating system. It does not require modifications in those applications, and that fits in the goals of PUNCH. However, it is not available for all the operating systems that currently form part of PUNCH, so an important implementation effort should be done before fully deploying Janus.

The basic principle behind Janus is that processes perform all dangerous operations through system calls: user code is safe. The sandbox is enforced by means of intercepting system calls. A configuration method tailors the level of restriction imposed to a process by the confining mechanism: some system calls can be totally disabled; some others can be authorized only after a careful checking of parameters. As an example, it could be possible to prevent a program to have access to the network. Or it could be allowed to contact only with a given machine. A specification file will state, with specific rules, the set of files accessible by a tool, and the set of operations the process can do. For example, socket access could be disabled (default rule), or enabled to a fixed set of machines/ports.

V. Current status

Currently, neither SSL nor Janus is integrated into PUNCH, although both technologies have been tested outside of it, and we do not expect any significant challenge to appear when attempting the integration.

VI. Related work

A. Globus

GSI (Grid Security Infrastructure) is the part of the Globus toolkit in charge of security [BE00, FK99]. Based upon standards such as X.509 and SSL, it provides some security services, such as authentication, authorization and single sign-on. It does not provide yet data confidentiality, due to the multinational nature of computational grids and the exportability issues that arise. One of the main features of GSI is that it is able to interact with local security policies and mechanisms, performing the appropriate translations (mappings) between Globus identifications and credentials and local ones.

Reliable authentication is the key to successfully implement any authorization mechanism. Globus authorization is based on checking the user's certificates, verifying that an approved certification authority has signed those. Access rights can be delegated to a proxy agent that acts on behalf of the user

Other aspects of security are not currently considered part of GSI. However most of the security aspects considered for a network computing system (protection of data, programs, and system resources) can only be achieved with a well-implemented authorization mechanism.

B. Legion

(See full description of Legion's security in [FK99]). Legion uses asymmetric cryptography as the basic mechanism to provide security. A public key (in fact, an X.509 certificate) is an integral part of an object's name (good: no directory access; bad: no way of changing it, that is a change of name). Private keys are sometimes transmitted (when a parent object creates a child object), but always through encrypted channels.

Authorization is managed by credentials, which are signed by a proxy. The proxy has a credential directly signed by the user, and can generate new credentials. In most cases credentials are of short life, to reduce security breaches. A Refresh Object can be set up to refresh expired credentials.

Access control is performed by each object, in a decentralized way. A "May I" layer is part of the protocol stack used to invoke a method of an object. This layer implements an access security policy, keeping access control lists and checking the invokers' credentials against that list.

Communication between objects can be performed without security, in private mode, or in protected mode. The private mode provides encryption of all fields in a message, including the body. The protected mode only encrypts certain fields, including transmitted credentials, and provides data integrity by means of (encrypted) message digests. The encryption mechanism is RSA, combined with DES in private mode.

Regarding the issue of protecting a system from Legion and vice-versa, Legion limits itself to use the host's operating system mechanisms. To isolate users from one another, each user's job run in separate accounts. If a legion user has an account in a host system, that one is used. If that is not the case, an account from a pool is used (with minimal permissions: no home directory, no group membership). The usage of the accounts is monitored/accounted for, and operations logged with the user's X.509 information. Some operations require root access, managed by a Process Control Daemon.

VII. Conclusions

We have presented a security architecture for PUNCH, a network computing infrastructure. The architecture tries to be comprehensive, in the sense that it attempts to cover all the security concerns that may appear in a large scale, network-computing environment. It is based on already developed systems and standards, not only to reduce implementation costs, but also because these systems have already been subject of public scrutiny (which increments our confidence in the appropriateness of the solution), and may facilitate future compatibility.

Classic security topics in networked environments, such as authentication, data integrity and data confidentiality, are solved through the use of SSL. This reduces security risks while data is in transit. Deploying a sandbox where running processes are confined performs protection of user's data and programs, and of the PUNCH infrastructure. This sandbox drastically limits the operations a process can perform, but it can be tailored to the specific needs of the task in hand, allowing it to perform exactly what it is required, but no more. Although the proposal fits the particular requirements of PUNCH, most of the ideas are applicable to network computing systems in general.

References

[KFF00] . Nirav H. Kapadia, Renato J. Figueiredo, and Jose' A. B. Fortes. *PUNCH: Web Portal for Running Tools*. IEEE Micro. May-June 2000.

[GW96] I. Goldberg, D. Wagner, R. Thomas and E. Brewer. *A Secure Environment for Untrusted Helper Applications (Confining the Wily Hacker).* Sixth USENIX UNIX Security Symposium. San Jose, California, July 1996.

[WS96] D. Wagner, B. Schneier. *Analysis of the SSL 3.0 Protocol.* Second USENIX Workshop on Electronic Commerce, November 1996.

[LS98] P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, J.F. Farrell. *The Inevitability of Failure: the Flawed Assumption of Security in Modern Computing Environment.* 21st National Information Systems Security Conference, Arlington VA, October 1998

[FK99] I. Foster and C. Kesselman (Eds). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[BE00] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch. *A National-Scale Authentication Infrastructure*. IEEE Computer, 33(12):60-66, 2000. Available at http://www.globus.org/documentation/incoming/butler.pdf

[NASA99] NASA Research and Education Network. *Bridging the Gap Workshop Report*. HPNAT/NRT Workshop, Aug. 10-11 1999. (www.nren.nasa.gov/img/BtG_FINAL_REPORT.pdf)

[FK99] A. Ferrari, F. Knabe, M. Humphrey, S. Chapin, A. Grimshaw. *A flexible Security System for Metacomputing Environments*. 7th Int. Conf. On High-Performance Computing and Networking Europe (HPCN'99).

APPENDIX: SSH AND SSL

SSH stands for Secure Shell. There are currently two versions of SSH: a free one (version 1, www.openssh.com) and a commercial one (version 2, www.ssh.com). The commercial version is available, free of cost, for university use. As SSH2 is in the IETF standards track, totally free versions of this protocol will be eventually available (the people behind the OpenSSH effort are already working on it).

The main features of SSH are:

- Strong encryption (3DES, Blowfish)
- X11 forwarding (encrypt X-Window traffic)
- Port forwarding (encrypted channels for legacy protocols)
- Strong authentication (public key, one-time password and Kerberos authentication)
- Agent forwarding (single-sign-on)
- Kerberos and AFS ticket passing
- Data compression

An SSH implementation includes these programs:

- ssh, a secure replacement for rlogin and telnet.
- scp, a secure replacement for rcp and ftp.
- sshd, the server side of SSH.
- Utilities such as ssh-add, ssh-agent and ssh-keygen.

SSH is quite effective in providing strong encryption and authentication, and it is widely deployed. However, credentials have to be managed manually: in its current implementation, it is not integrated with a PKI based on X.509 certificates. This is an important limitation for a wide-area system comprising a large set of organizations.

SSL/TLS is a protocol to provide security on top or TCP connections. It provides a framework to add security services such as authentication, data integrity and privacy, but it does not specify a particular set of algorithms to use to implement these functions. Common choices are RSA or DSA for authentication, SHA or MD5 for integrity, and 3DES or RC4 for encryption.

SSL depends on the existence of a PKI to work. It uses X.509 certificates as user's credentials. This fact makes deployment cumbersome, because requires the use of a Certification Authority, but also makes it more scalable because credentials are not managed manually.

OpenSSL (www.openssl.org) is a free implementation of SSL/TLS. It provides:

- A command to perform cryptographic functions from the shell
- A library of SSL functions, to deal with SSL sessions and certificates
- A library of cryptographic functions

This is just a basic toolkit to build secure programs. In fact, OpenSSH has been built using the crypto library of OpenSSL. Other programs built on top of OpenSSL are secure versions of the Apache web server, SSL-aware telnet and ftp replacements, and SSL-tunneling systems designed to wrap legacy protocols. GSI, the part of the Globus toolkit dealing with security, has been implemented using SSLeay, a predecessor of OpenSSL.