SECURITY CONSIDERATIONS IN DISTRIBUTED CONTROL SYSTEMS

Klemen Žagar, Jožef Stefan Institute and Cosylab, Ljubljana, Slovenia

Abstract

In recent years, a need appeared to distribute control systems of large experimental physical devices to participating laboratories across the globe [1]. Internet is the obvious choice for the communication medium that allows such distribution. However, given the Internet's public nature, one must ensure that access to multi-billion-Euro devices is appropriately secured. In this paper, security threats that endanger distributed control systems are analyzed. Then, guidelines are given on how, where and why to use technologies such as Secure Sockets Layer (SSL), Public Key Infrastructure (PKI), Digital Signature and Access Control Lists (ACL). Finally, we discuss security efforts we have invested in our products (in particular the *Advanced Control System* – ACS, and Abeans) and in our infrastructure (servers, web site, e-mail).

INTRODUCTION

During design of a control system, functionality that determines which devices can be controlled is of prime importance. Once this goal is found achievable, the non-functional requirements of performance and scalability are addressed. All of these are time consuming tasks, and once done, there is usually not enough budget left to identify and resolve security issues.

Security is often taken for granted. Under normal circumstances, people trust each other, and there seems to be no need for taking any security measures. In truth, security is a hinderance as it limits usability. Without it, entering a room is just as simple as turning the door knob. Once in place, key is needed to unlock the door, and security codes have to be punched into the alarm system.

These days computers are highly networked. Viruses and trojan horses are everywhere, and can spread across the globe within minutes, if not seconds. If there is a weakness that is not attended to, someone will exploit it.

Control systems for large experimental physics devices are *mission critical* in a sense that if they fail to do their job, a multi-billion Euro device is wasting its precious hours. Since in the future, when remote control will be not only *fancy*, but mandatory, the old way of securing the system (physically disconnecting it from the rest of the world) will no longer be satisfactory. Security issues will have to be dealt with differently.

ACCESS CONTROL AND AUTHENTICATION

From an abstract point of view, a system allows *users* to perform *operations* on certain *objects*. For example, an *operator* would *set* the *current of a power supply*.

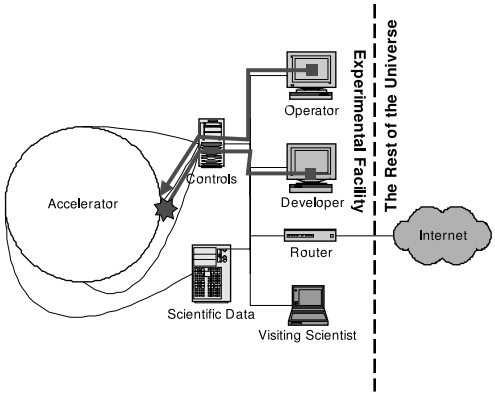

Figure 1: **Access control** is important: don't allow everyone to do anything! Thus, developers are usually not given access to the actual controls, as they might inadvertently cause serious damage.

However, not all such triplets should be allowed. In figure 1, the developer should be prevented from setting the current of an actual device, but should instead use a simulated or test device.

To implement adequate **access control**, every action that a user attempts to perform should be subjected to **authorization**. Two authorization schemes are particularly widespread:

- UNIX-style user/group/world read/write/execute privileges.
- Windows-style access control lists (ACL).

In many scenarios it turns out that UNIX-style authorization is not flexible enough. Fortunately, filesystems exist that overcome this difficulty on UNIX and Linux platforms (e.g., the SGI's XFS filesystem [2], and `ext3` filesystem with ACL support).

With authorization in place, the problem is reduced to identifying the *user* in the triplet user/operation/object (**authentication**). The users could be asked to identify themselves by (user)name, but then misrepresentation as someone else would be too easy. Typically, users identify themselves with passwords, which can be technically done in four flavors:

1. The system stores the password, and compares user's entry with what is stored. This method is very unsecure, and its use is not recommended.
2. The system stores a *hash code* of the password, and compares hash codes. This way, it is impossible (or

at least very hard) to obtain the user's password if the authentication database is accidentally disclosed.

3. The system stores the password (an encryption key), and asks the user to encrypt a message with this key (*challenge*). The *response*, once decrypted by the system, must match original message. This prevents the so-called **replay attacks**, where the attacker could replay the exact user's input during authentication, and steal his or her identity.

4. Same as method 3, except that the system stores the *public key* for the user's matching *private key*. This way, the authentication data in the system can be disclosed without harm.

THREAT CLASSIFICATION

Before securing a system, threats must be thouroughly understood. Failing to do so could lead to protecting too much (e.g., using security technologies for their own sake, or limiting usability), or too little.

Note that in the world of security, the *weakest link* principle applies: no matter how many Navy Seals are protecting your garage door, leaving a back window open will still make your possesion an easy prey.

In this section, a threat classification scheme called **STRIDE** [3] is presented. STRIDE is an acronym composed of the first letters of threat types.

Spoofing Identity

The threat where an attacker could represent himself as another user is called *spoofing identity*. Realization of this threat could have severe consequences, especially if the victimized user had many privileges.

To mitigate this security threat, consider the following:

- Don't store secrets such as passwords unprotected (e.g., in a world-readable file).
- Don't store private keys and certificates unprotected. Ideally, use *smart cards* and *hardware security modules*. Otherwise, protect these keys with *passphrases* (a password that serves as an encryption/decryption key for the private key, and is known only by the user).
- Use biometric authentication (fingerprints, photographs, voice, ...).
- Don't transmit secrets over the network unprotected. Avoid using POP3, IMAP, FTP, Telnet, and sending passwords and credit card numbers over HTTP. Use IMAPS, SFTP, SSH and HTTPS instead!

Tampering with Data

Critical data should be protected against unauthorized modification (*tampering*). In control systems, the following should be tamper resistant:

- Configuration files. If modified by inexperienced or malicious personnel, the machine would become impossible to operate, or parts of it could even break down.
- Logs. An attacker, in an attempt to cover up his wrong-doings, should be prevented from modifying the logs and thus purging all traces behind him.

To tamper-protect the data, a *cryptographically secure hashing function* is applied over it, and both the data and the hash are recorded. To verify integrity, hash is recalculated and checked. The following technologies address data integrity issues:

- Hashing functions (the slightly insecure MD5, and SHA1).
- Message Authentication Codes (MAC), where encrypted message is hashed to also verify authenticity.
- Secure Sockets Layer (SSL) and protocols above it (HTTPS, SSL, ...).
- Digital signature (PKCS#11, XML Digital Signature, Pretty Good Privacy – PGP, Secure Multimedia Internet Extensions – S/MIME).

Repudiation

Repudiation refers to the ability of the data issuer to deny that he had produced the data. The feature of *non-repudiation* is particularly useful in legal proceedings, where the author of a message must be established *beyond reasonable doubt.*

The primary means of establishing non-repudiation is through *digital signatures.*

Beware! Implementing digital signatures properly is a complex issue, as it is not just a technical, but also a legal problem. Issues such as certificate expiration and revocation must be addressed, so that the digital signature remains valid, even though its issuing certificate had expired since the signing took place.

Information Disclosure

Information disclosure occurs when confidential information falls in the hands of someone, from whom it is supposed to have been kept secret. This can happen in two ways:

- Interception of data while in transit on the network.
- Acquisition of access to where the data is stored.

To counter the first case, *encrypt* the data while in transit. Using SSL or a protocol based on it (SFTP, HTTPS) solves the first issue well. For safe storage, encrypted file systems and databases should be used.

Denial of Service

When a system is subjected to a load that exceeds the one for which it has been designed, two things can happen:

- The system becomes unresponsive, and its *quality of service* is decreased significantly.
- The system breaks down.

Denial-of-Service attacks exploit this fact to render their targets useless. They are launched from several sources, which generate various kinds of requests on the victimized system (for example, `ping` or web page requests).

The best way to defend against these is to set up a firewall that filters inbound traffic. Also, most firewalls implement a technique called *throttling* where traffic from a given source is simply ignored if it is too frequent.

Elevation of Privilege

Elevation of privilege occurs when the attacker assumes high privileges on the victimized machine (e.g., those of the administrator/`root`). Such an attacker can do anything he likes – including realizing all the other threats listed above.

Elevation of privilege can be exploited if the administrator's password is compromised (i.e., through the *spoofing identity* threat). More commonly, vulnerabilities in installed networking services are taken advantage of. A typical such vulnerability is the **buffer overrun** where the attacker supplies a larger-than-expected input containing code that the service actually executes.

THREAT MODELING

Information systems, including control systems, are concerned with transmission and storage of data. Thus, the adversary's only means of attack is by manipulating the data, either while in transit, or when stored. To better understand the paths of data, a *data flow diagram* should be produced.

In a distributed control system, the following resources have to be secured:

- **Output channels to the controlled devices.** Unauthorized users must not be allowed to manipulate devices, as they might disrupt the operation of the facility, or, in the worst case, damage the devices.
- **Input channels from the monitored devices.** Unauthorized users should not be allowed to acquire control data.
- **Archive integrity.** It should be impossible to remove or alter the entries in the archive. Otherwise, malicious users would be able to remove traces of their activities.
- **Network bandwidth and other resources.** Unauthorized users should not be able to generate traffic on the control system's network, as they might abuse this capability to overload the network and/or servers, and thus issue a Denial of Service (**DoS**) attack on the control system.

SECURITY IN ACS AND ABEANS

The *Advanced Control System* provides infrastructure for authorization and authentication. This is implemented by the Manager, which checks whether a client is authorized to access a component whenever it requests one. Currently, only basic username/password authentication is performed.

To avoid buffer overruns and other pitfalls, the Standard Template Library (STL) has been extensively used while developing the C++ parts of the ACS. The remaining parts of ACS are written in Java, where critical coding errors resulting in possible exploits much are much less frequent.

On the client side, the Abeans framework has a plug for Web Services based on *Simple Object Access Protocol* (SOAP). Since SOAP uses HTTP and/or HTTPS for transport of data, secure authentication is integral part of all Abeans-based applications.

In the future, we intend to add *security reviews* to our software development process with the goal of eliminating the majority of potential exploits and *weak links* before they are detected or taken advantage of. Apart from that, we are looking into adding encryption and digital signature to the middleware (using CORBA Security), logging service (for log non-repudiation) and the ACS's Configuration Database.

CONCLUSION

Networked computer systems are more vulnerable than ever, as attackers can take advantage of them at any time, from anywhere. This is particularly true if systems are distributed across the globe using the Internet. Therefore, security considerations must be taken into account when designing, building and deploying them [4, 5].

Security issues should be identified already in the design phase and mitigated. Technologies to achieve this are readily available, and some of them have been briefly outlined in this article.

REFERENCES

[1] R. Bacher, "The Global Accelerator Network - Globalisation of Accelerator Operation and Control", PCaPAC 2002, Frascati, October 2002

[2] Silicon Graphics, "XFS", http://oss.sgi.com/projects/xfs/

[3] M. Howard, D. LeBlanc, "Writing Secure Code", Microsoft Press, 2002

[4] H. Frese, "Security Planning from the Start" (talk), PCaPAC 2002, Frascati, October 2002

[5] D. Agarwal, "Supporting Secure and Scalable GAN Collaborations" (talk), Collaboration Tools for the Global Accelerator Network (GAN) Workshop, August 26, 2002