

# Three Design Patterns for Secure Distributed Systems

Alan H. Karp

HP Labs

## Introduction

Much of our thinking on the subject of computer security originated in the 1960s and 1970s, a time when computers were rarely networked, those that were connected largely trusted each other, and those connections rarely changed. That is not the situation today. Virtually all computers and an increasing number of other devices are connected to the Internet. Many of the machines out there are running software that attempts to harm others, either because their owners are malicious or because their owners are careless and have allowed malicious people take control of their machines. To make matters worse, the environment is constantly changing, with machines joining and leaving the system and even changing owners.

A significant result of these changes is that the mapping between *user* and *process* is far more tenuous than it used to be. We give privileges to people, but we enforce access control on processes. In the old days, this distinction was not much of a problem. Software was locally installed and there was little difference between the privileges granted to people and what those people wanted processes to do on their behalf. That is not the case today. When people run remote software on their machines, they may not want those processes to have all their privileges. When people run their processes on other machines, they may not want those processes to have all their privileges. In the former case, we want to protect the machine from the process; in the latter, we want to protect the process from the machine. The virus runs because we don't do the former; the person's bank account can be drained if we don't do the latter.

## Assumptions and Implications

The changes in the environment mean that builders of distributed systems need to re-examine their assumptions when designing infrastructure for the Internet. The work presented here is based on the following assumptions and implications.

**Large number of machines and users:** Because of the large number of machines, we can't rely on a centralized repository for identity management. Because of the time it takes changes to propagate through such a large system, we can't rely on global data consistency to manage privileges, particularly revocation of privileges. The problems of identity management are exacerbated because there are so many users.

**Dynamic:** Applications are very difficult to write if the underlying system is always changing. Relying on application code to reconnect to a service, or find an equivalent service is dangerous. Applications can be run by users with limited privileges, but security decision often need to be made in a context with more rights. Careless users can expose the entire machine, and even other machines, to attack. We need to shield to the greatest possible extent the applications from changes in its environment.

**Heterogeneous:** The environment is heterogeneous in device capabilities as well as in machine type and operating system. Some devices may have limited computation ability, communications bandwidth, or storage, which means we need to choose our security mechanisms carefully.

**Hostile:** We know that malicious people are trying to circumvent security, and we know that our mechanisms are not perfect. That means some attacks will succeed, and we need to design systems that will limit the damage that can be done when they do.

**Different Environments:** We need to recognize that there are different ways to achieve an end. Security mechanisms that work in the enterprise may be inappropriate in a small business or in a home. Our mechanisms must be flexible enough to support a wide variety of requirements.

What follows are three design patterns that address some of these concerns. None of these patterns can reduce the security exposures of the underlying operating system, but they will make it more difficult for attacks against applications to succeed.

## Use a Proxy

When your machine needs to talk to another machine, do it through separate process that acts as a proxy. There are a number of benefits. First of all, the proxy can enforce any policies set up for the machine. After all, you wouldn't want a careless user to inadvertently send confidential material to an insecure machine. In addition, the proxy can mitigate the damage done by some attacks. If the attacker crashes the process in which the proxy is running, the only effect is to cut off the attacker's access to your machine. If you detect an attack against or through the proxy, you only need to kill the process running the proxy to cut off the attacker.

In some operating systems, the proxy can run with a set of privileges appropriate for the remote user instead of the local one. In this case, the proxy can be extremely accommodating, trying to do everything it is asked to do. Since its permissions are controlled by the machine on which it is running, it won't be able to do anything that the remote user hasn't been granted permission to do. Privileges can be revoked by removing them from the proxy's set of permissions.

In other situations, the proxy can be made smarter, filtering requests that might cause problems if forwarded to the application. The advantage of putting these controls in the proxy is that they can protect applications written with no consideration for remote users, and updates to the proxy add protections to all applications on the machine.

## Mediate between Application and User

Add a level of indirection between the user and the application. If the user is remote, put the mediator between the proxy for the user and the application. The analogy is with an operating system. The user of a file doesn't talk directly to the file system; the requests pass through the operating system kernel. This mediator provides several useful functions. It can provide audit logs or generate use events that provide visibility needed to manage the system effectively. The mediator can translate between high-level policies, as expressed in an access control list, into low-level mechanism, such as

allowing access to a particular file. The mediator also acts as a trusted third party, providing a verifiable identity for both parties, among other things.

The alternative of putting these functions into libraries to be linked with the application executable doesn't work as well. There is the problem of upgrades to the libraries; making sure that all applications use the latest version and are tested with it is non-trivial. Also, there is no way to ensure that every application generates the necessary log entries and management events. The absence of a mediator also makes identity management an issue that needs to be addressed in every application.

## **Separate Granting of Rights from Access Control**

There is no reason why the mechanism used to grant privileges to a person or a process should be the same as the mechanism used to decide whether a particular request should be honored. Using this pattern simplifies the code needed to grant the rights, because it doesn't need to deal with the interface of the resource, and the code that decides whether to grant access, because it can be independent of any authentication.

This approach is not entirely new. When a Kerberos ticket is issued to a user who has presented certain credentials, that ticket can be used to access some resource. The access control decision is based on the ticket, not the authentication used to get the ticket. Unfortunately, in Kerberos and similar systems the form of the ticket is part of the architecture. This pattern suggests that the ticket be allowed to be something specific to the application. After all, the access decision may depend on some context not considered when the ticket granting mechanism was designed.

## **Status and Next Steps**

These three patterns were used in e-speak, an open source product offered by HP from 1999 until 2002. That product was discontinued, the team dispersed, and all traces of the product removed from HP's public web sites. Although no one is aware of a serious assault on the security of the system, it is also true that no one has reported a security breach in any of the existing applications. Since these include applications involving pride (HiTel), money, (mTicka) and proprietary information (SpinCircuit), where there is an incentive to find a flaw, there is a chance that the patterns described here helped.