2 The Architectural-Level Patterns

2.1 Distrustful Decomposition

2.1.1 Intent

The intent of the Distrustful Decomposition secure design pattern is to move separate functions into mutually untrusting programs, thereby reducing the

- attack surface of the individual programs that make up the system
- functionality and data exposed to an attacker if one of the mutually untrusting programs is compromised

2.1.2 Also Known As

Privilege reduction

2.1.3 Motivation

Many attacks target vulnerable applications running with elevated permissions. This allows the attacker to access more information and/or allows the attacker to perform more damage after exploiting a security hole in the application than if the application had been running with more restrictive permissions. Some examples of this class of attack are

- various attacks in which Internet Explorer running in an account with administrator privileges is compromised
- security flaws in Norton AntiVirus 2005 that allow attackers to run arbitrary VBS scripts when running with administrator privileges
- a buffer overflow vulnerability in BSD-derived telnet daemons that allows an attacker to run arbitrary code as root

All of these attacks take advantage of security flaw(s) in an application running with elevated privileges (root under UNIX or administrator under Windows) to compromise the application and then use the application's elevated privileges and basic functionality to compromise other applications running on the computer or to access sensitive data. The Distrustful Decomposition pattern isolates security vulnerabilities to a small subset of a system such that compromising a single component of the system does not lead to the entire system being compromised. The attacker will only have the functionality and data of the single compromised component at their disposal for malicious activity, not the functionality and data of the entire application.

2.1.4 Applicability

This pattern applies to systems where files or user-supplied data must be handled in a number of different ways by programs running with varying privileges and responsibilities. A naive implementation of this system may allocate many disparate functions to the same program, forcing the program to be run at the privilege level required by the program function requiring the highest privilege level. This provides a large attack surface for attackers and leaves an attacker with access to a system with a high privilege level if the system is compromised.

A system can make use of the Distrustful Decomposition pattern if

- the system performs more than one high-level function
- the various functions of the system require different privilege levels

2.1.5 Structure

The general structure of this pattern breaks the system up into two or more programs that run as separate processes, with each process potentially having different privileges. Each process handles a small, well-defined subset of the system functionality. Communication between processes occurs using an inter-process communication mechanism such as RPC, sockets, SOAP, or shared files.

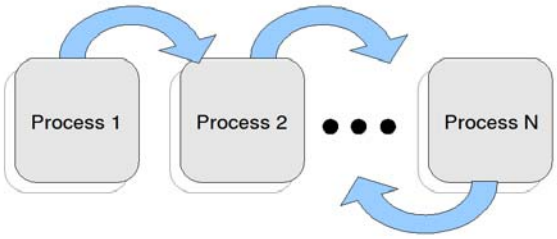

Figure 1: General Structure of the Distrustful Decomposition Secure Design Pattern

2.1.6 Participants

These are the participants in the Distrustful Decomposition pattern:

- a number of separate programs, each running in a separate process. For more complete separation, each process could have a unique user ID that does not share any privileges with the other user IDs.
- a local user or a remote system connecting over a network
- possibly the system's file system
- possibly an inter-process communication mechanism such as UNIX domain sockets, RPC, or SOAP

2.1.7 Consequences

Distrustful Decomposition prevents an attacker from compromising an entire system in the event that a single component program is successfully exploited because no other program trusts the results from the compromised one.

2.1.8 Implementation

This pattern employs nothing beyond the standard process/privilege model already existing in the operating system. Each program runs in its own process space with potentially separate user privileges. Communication between separate programs is either one-way or two-way.

- **One-way**. Only `fork()`/`exec()` (UNIX/Linux/etc.), `CreateProcess()` (Windows Vista), or some other OS-specific method of programmatic process creation is used to transfer control. One-way communication reduces the coupling between processes, making it more diffi-

cult for an attacker to compromise one system component from another, already compromised component.

- **Two-way**. A two-way inter-process communication mechanism like TCP or SOAP is used. Extra care must be taken when using a two-way communication mechanism because it is possible for one process involved in the two-way communication to be compromised and under the control of an attacker. As with the file system, two-way communication should not be inherently trusted.

The file system may be a means of interaction, but no component places any inherent trust in the contents of the file.

2.1.9 Sample Code

An excellent example system where this pattern is applied is the qmail mail system, which is a complex system with a large combination of interactions between systems, users, and software components.

The overall structure of the qmail system is shown in Figure 2 [Oppermann 1998].

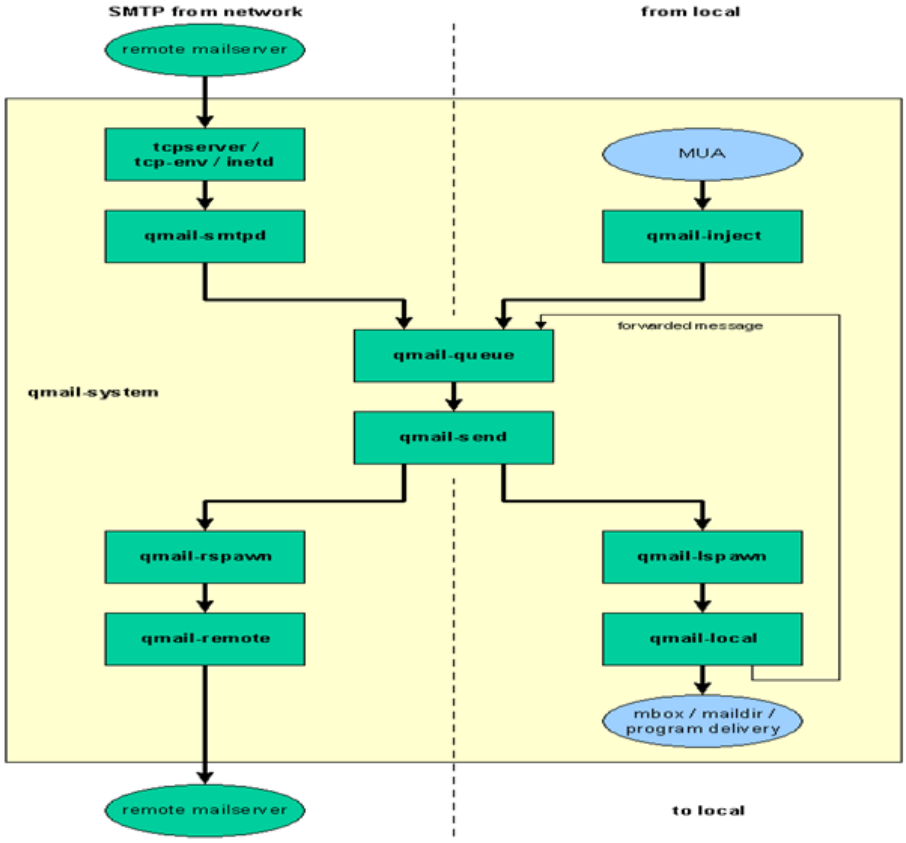

Figure 2: Structure of the Qmail Mail System[1]

1 Source: http://www.nrg4u.com/qmail/the-big-qmail-picture-103-p1.gif. Used with permission from the author.

The actual source code for the qmail system is omitted here; see the qmail website [Bernstein 2008] for examples.

2.1.10 Known Uses

- The qmail mail system [Bernstein 2008].
- The Postfix mail system uses a similar pattern [Postfix].
- Microsoft mentions this general pattern when discussing how to run applications with administrator privileges [MSDN 2009b].
- Distrustful decomposition for Windows Vista applications using user account control (UAC) is explicitly addressed in [Massa 2008].

2.2 PrivSep (Privilege Separation)

2.2.1 Intent

The intent of the PrivSep pattern is to reduce the amount of code that runs with special privilege without affecting or limiting the functionality of the program. The PrivSep pattern is a more specific instance of the Distrustful Decomposition pattern.

2.2.2 Motivation

In many applications, a small set of simple operations require elevated privileges, while a much larger set of complex and security error-prone operations can run in the context of a normal unprivileged user. For a more detailed discussion of the motivation for using this pattern, please see the motivation for the more general Distrustful Decomposition pattern.

Figure 3 provides a detailed view of a system where the PrivSep pattern could be applied and the security problems that can occur if the PrivSep pattern is not used [Provos 2003]. An implementation of `ftpd` is used as an example.

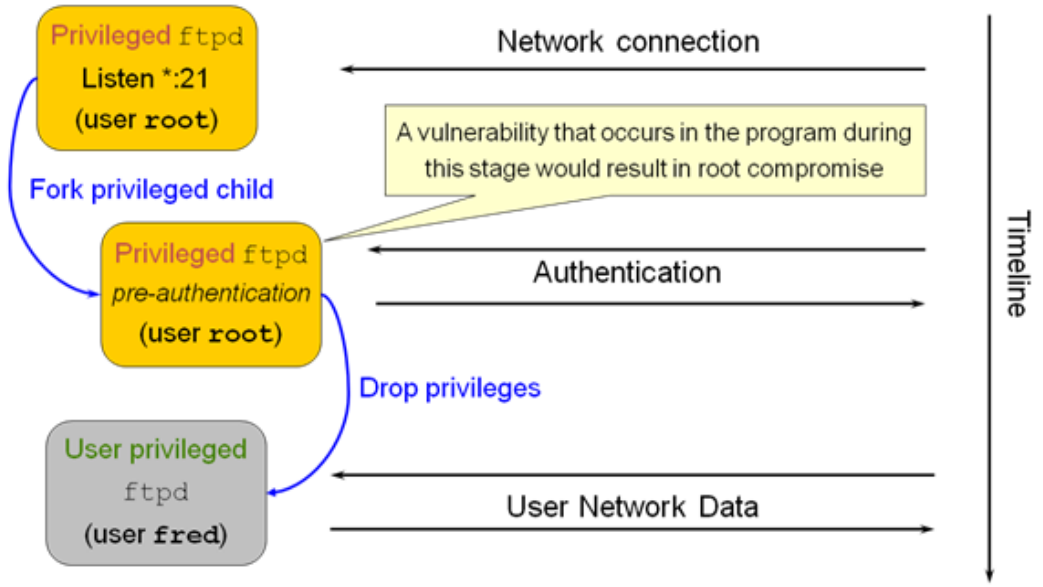

Figure 3: Vulnerable `ftpd` Program