An operating system (OS) allows multiple applications to share the hardware resources of a physical

system, subject to a set of policies. A critical function of an OS is to protect applications against a

wide range of malicious attacks such as unauthorized access to privileged information, tempering with

executable code, and spoofing. Such attacks can now target even single-user systems such as personal

computers, tablets, or smartphones. Data brought into the system may contain malicious code; this

could occur via a Java applet, or data imported by a browser from a malicious Web site.

The mandatory security of an OS is considered to be “any security policy where the definition of

the policy logic and the assignment of security attributes is tightly controlled by a system security

policy administrator” [209]. Access control, authentication usage, and cryptographic usage policies are

all elements of mandatory OS security. The first policy specifies how the OS controls the access to

different system objects, the second defines the authentication mechanisms the OS uses to authenticate

a principal, and the last specifies the cryptographic mechanisms used to protect the data. A necessary

but not sufficient condition for security is that the subsystems tasked with performing security-related

functions are temper-proof and cannot be bypassed. The OS should confine an application to a unique

security domain.

Applications with special privileges that perform security-related functions are called trusted applications.

Such applications should only be allowed the lowest level of privileges required to perform

their functions. For example, type enforcement is a mandatory security mechanism that can be used to

restrict a trusted application to the lowest level of privileges.

Enforcing mandatory security through mechanisms left to the discretion of users could lead to a

breach of security due not only to malicious intent but also carelessness or lack of understanding.

Discretionary mechanisms place the burden of security on individual users. Moreover, an application

may change a carefully defined discretionary policy without the consent of the user, whereas a mandatory

policy can only be changed by a system administrator.

Unfortunately, commercial operating systems do not support multilayered security; such systems

only distinguish between a completely privileged security domain and a completely unprivileged one.

Some operating systems, such as Windows NT, allow a program to inherit all the privileges of the

program invoking it, regardless of the level of trust in that program.

The existence of trusted paths, mechanisms supporting user interactions with trusted software, is

critical to system security. If such mechanisms do not exist, malicious software can impersonate trusted

software. Some systems provide trust paths for a few functions such as login authentication and password

changing and allow servers to authenticate their clients.

The solution discussed in [209] is to decompose a complex mechanism into several components with

well-defined roles. For example, the access control mechanism for the application space could consist

of enforcer and decider components. To access a protected object, the enforcer will gather the required

information about the agent attempting the access and will pass this information to the decider, together

with the information about the object and the elements of the policy decision. Finally, it will carry out

the actions requested by the decider.

A trusted-path mechanism is required to prevent malicious software invoked by an authorized application

to tamper with the attributes of the object and/or with the policy rules. A trusted path is also

required to prevent an impostor from impersonating the decider agent. A similar solution is proposed

for cryptography usage, which should be decomposed into an analysis of the invocation mechanisms

and an analysis of the cryptographic mechanism.

Another question is how an OS can protect itself and the applications running under it from malicious

mobile code attempting to gain access to the data and the other resources and compromise system

confidentiality and/or integrity. Java Security Manager uses the type-safety attributes of Java to prevent

unauthorized actions of an application running in a “sandbox.” Yet, the Java Virtual Machine (JVM)

accepts byte code in violation of language semantics; moreover, it cannot protect itself from tampering

by other applications.

Even if all these security problems could be eliminated, good security relies on the ability of the file

system to preserve the integrity of Java class code. The approach to require digitally signed applets and

accept them only from trusted sources could fail due to the all-or-nothing security model. A solution to

securing mobile communication could be to confine a browser to a distinct security domain.

Specialized closed-box platforms such as the ones on some cellular phones, game consoles, and automated

teller machines (ATMs) could have embedded cryptographic keys that allow themselves to reveal

their true identity to remote systems and authenticate the software running on them. Such facilities are

not available to open-box platforms, the traditional hardware designed for commodity operating systems.

A highly secure operating system is necessary but not sufficient unto itself; application-specific

security is also necessary. Sometimes security implemented above the operating system is better. This

is the case for electronic commerce that requires a digital signature on each transaction.

We conclude that commodity operating systems offer low assurance. Indeed, an OS is a complex

software system consisting of millions of lines of code, and it is vulnerable to a wide range of malicious

attacks. An OS poorly isolates one application from another, and once an application is compromised,

the entire physical platform and all applications running on it can be affected. The platform security

level is thus reduced to the security level of the most vulnerable application running on the platform.

Operating systems provide only weak mechanisms for applications to authenticate to one another and

do not have a trusted path between users and applications. These shortcomings add to the challenges of

providing security in a distributed computing environment. For example, a financial application cannot

determine whether a request comes from an authorized user or from a malicious program; in turn, a

human user cannot distinguish a response from a malicious program impersonating the service from

the response provided by the service itself.