The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments

Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor,
S. Jeff Turner, John F. Farrell
tos@epoch.ncsc.mil
National Security Agency

Abstract

Although public awareness of the need for security in computing systems is growing rapidly, current efforts to provide security are unlikely to succeed. Current security efforts suffer from the flawed assumption that adequate security can be provided in applications with the existing security mechanisms of mainstream operating systems. In reality, the need for secure operating systems is growing in today's computing environment due to substantial increases in connectivity and data sharing. The goal of this paper is to motivate a renewed interest in secure operating systems so that future security efforts may build on a solid foundation. This paper identifies several secure operating system features which are lacking in mainstream operating systems, argues that these features are necessary to adequately protect general application-space security mechanisms, and provides concrete examples of how current security solutions are critically dependent on these features.

Keywords: secure operating systems, mandatory security, trusted path, Java, Kerberos, IPSEC, SSL, firewalls.

1 Introduction

Public awareness of the need for security in computing systems is growing as critical services are becoming increasingly dependent on interconnected computing systems. National infrastructure components such as the electric power, telecommunication and transportation systems can no longer function without networks of computers [50]. The advent of the World Wide Web has especially increased public concern for security. Security is the primary concern of businesses which want to use the Internet for commerce and maintaining business relationships [24].

The increased awareness of the need for security has resulted in an increase of efforts to add security to computing environments. However, these efforts suffer from the flawed assumption that security can adequately be provided in application space without certain security features in the operating system. In reality, operating system security mechanisms play a critical role in supporting security at higher levels. This has been well understood for at least twenty five years [2][54][39], and continues to be reaffirmed in the literature [1][35]. Yet today, debate in the research community as to what role operating systems should play in secure systems persists [11]. The computer industry has not accepted the critical role of the operating system to security, as evidenced by the inadequacies of the basic protection mechanisms provided by current mainstream operating systems.

The necessity of operating system security to overall system security is undeniable; the underlying operating system is responsible for protecting application-space mechanisms against tampering, bypassing, and spoofing attacks. If it fails to meet this responsibility, system-wide vulnerabilities will result.

The need for secure operating systems is especially crucial in today's computing environment. Substantial increases in connectivity and data sharing have increased the risk to systems such that even a careful and knowledgeable user running on a single-user system is no longer safe from the threat of malicious code. Because the distinction between data and code is vanishing, malicious code may be introduced, without a conscious decision on the part of a user to install executable code, whenever data is imported into the system. For example, malicious code could be introduced with a Java applet or by viewing apparently benign data that, in actuality, contains executable code [32][62]. More so than ever, secure operating systems are needed to protect against this threat.

The goal of this paper is to motivate a renewed interest in secure operating systems. By consolidating a number of well-documented examples from the literature, it argues that the threats posed by the modern computing environment cannot be addressed without support from secure operating systems and, as was stated in [8], that any security effort which ignores this fact can only result in a "fortress built upon sand." Section 2 describes a set of secure operating system features which are typically lacking in mainstream operating systems but are crucial to information security. The need for these features is high-

lighted in section 3, which examines how application-space access control and cryptography cannot provide meaningful security without a secure operating system. Section 4 provides concrete examples of how security efforts rely on these operating system security features. Section 5 discusses the role of operating system security with respect to overall system security.

2 The Missing Link

This section identifies some features of secure operating systems which are necessary to protect application-space security mechanisms yet are lacking in mainstream operating systems. They form the "missing link" of security. Although this section only deals with features, it is important to note that features alone are inadequate. Assurance evidence must be provided to demonstrate that the features meet the desired system security properties and to demonstrate that the features are implemented correctly. Assurance is the ultimate missing link; although approaches to providing assurance may be controversial, the importance of assurance is undeniable.

The list of features in this section is not intended to be exhaustive; instead it is merely a small set of critical features that demonstrate the value of secure operating systems. A more complete discussion on secure operating systems, including discussions of assurance, can be found in [25], [59] or [20]. Subsequent sections argue the necessity of these features by describing how application-space security mechanisms and current security efforts employing them are vulnerable in their absence.

Mandatory security

The TCSEC [20] provides a narrow definition of *mandatory security* which is tightly coupled to the multi-level security policy of the Department of Defense. This has become the commonly understood definition for mandatory security. However, this definition is insufficient to meet the needs of either the Department of Defense or private industry as it ignores critical properties such as intransitivity and dynamic separation of duty [12][22]. This paper instead uses the more general notion of mandatory security defined in [59], in which a mandatory security policy 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. Mandatory security can implement organization-wide security policies. Others have referred to this same concept as *non-discretionary security* in the context of role-based access control [22] and type enforcement [39][7][13].[1]

Likewise, as defined in [59], this paper uses a more general notion of *discretionary security* in which a discretionary security policy is considered to be any security policy where ordinary users may be involved in the definition of the policy functions and/or the assignment of security attributes. Here discretionary security is not synonymous with identity based access control; IBAC, like any other security policy, may be either mandatory or discretionary[58].

An operating system's mandatory security policy may be divided into several kinds of policies, such as an access control policy, an authentication usage policy, and a cryptographic usage policy. A mandatory access control policy specifies how subjects may access objects under the control of the operating system. A mandatory authentication usage policy specifies what authentication mechanisms must be used to authenticate a principal to the system. A mandatory cryptographic usage policy specifies what cryptographic mechanisms must be used to protect data. Additionally, various subsystems of the operating system may have their own mechanism usage policies. These subsystem-specific usage policies may be dependent on the cryptographic usage policy. For example, a network usage policy for a router might specify that sensitive network traffic should be protected using IPSEC ESP [4] in tunneling mode prior to being sent to an external network. The selection of a cryptographic algorithm for IPSEC ESP may be deferred to the cryptographic usage policy.

A secure system must provide a framework for defining the operating system's mandatory security policy and translating it to a form interpretable by the underlying mandatory security mechanisms of the operating system. Without such a framework, there can be no real confidence that the mandatory security mechanisms will provide the desired security properties.

An operating system which provides mandatory security may nonetheless suffer from the presence of high bandwidth covert channels. This is an issue whenever the mandatory security policy is concerned with confidentiality. This should not, however, be a reason to ignore mandatory security. Even with covert channels, an operating system with basic mandatory controls improves security by increasing the required sophistica-

1. Actually, long ago, the term *non-discretionary controls* was used for multi-level security as well [39].

tion of the adversary. Once systems with basic mandatory controls become mainstream, covert channel exploitation will become more common and public awareness of the need to address covert channels in computing systems will increase[57].

In any system which supports mandatory security, some applications require special privileges in the mandatory policy in order to perform some security-relevant function. Such applications are frequently called trusted applications because they are trusted to correctly perform some security-related function and because they are trusted to not misuse privileges required in order to perform that function. If the mandatory security mechanisms of a secure operating system only support coarse-grained privileges, then the security of the overall system may devolve to the security of the trusted applications on the system. To reduce the dependency on trusted applications, the mandatory security mechanisms of an operating system should be designed to support the principle of least privilege. Type enforcement is an example of a mandatory security mechanism which may be used both to limit trusted applications to the minimal set of privileges required for their function and to confine the damage caused by any misuse of these privileges [48][28].

The mandatory security mechanisms of an operating system may be used to support security-related functionality in applications by rigorously ensuring that subsystems are unbypassable and tamperproof. For example, type enforcement may be used to implement assured pipelines to provide these properties. An assured pipeline ensures that data flowing from a designated source to a designated destination must pass through a security-related subsystem and ensures the integrity of the subsystem. Many of the security requirements of these applications may be ensured by the underlying mandatory security mechanisms of the operating system. [48]

Operating system mandatory security mechanisms may also be used to rigorously confine an application to a unique security domain that is strongly separated from other domains in the system. Applications may still misbehave, but the resulting damage can now be restricted to within a single security domain. This confinement property is critical to controlling data flows in support of a system security policy [33]. In addition to supporting the safe execution of untrustworthy software, confinement may support functional requirements, such as an isolated testing environment or an insulated development environment [48]. For example both the Sidewinder firewall and the DTE firewall use type enforcement for confinement [6][12].

Although one could attempt to enforce a mandatory security policy through discretionary security mechanisms, such mechanisms can not defend against careless or malicious users. Since discretionary security mechanisms place the burden for security on the individual users, carelessness by any one user at any point in time may lead to a violation of the mandatory policy. In contrast, mandatory security mechanisms limit the burden to the system security policy administrator. With only discretionary mechanisms, a malicious user with access to sensitive data and applications may directly release sensitive information in violation of the mandatory policy. Although that same user may also be able to leak sensitive information in ways that do not involve the computing system, the ability to leak the information through the computing system may increase the bandwidth of the leak and may decrease its traceability. In contrast, with mandatory security mechanisms, he may only leak sensitive information through covert channels, which limits the bandwidth and increases accountability, if covert channels are audited.

Furthermore, even with users who are benign and careful, the mandatory security policy may still be subverted by flawed or malicious applications when only discretionary mechanisms are used to enforce it.[2] The distinction between flawed and malicious software is not particularly important in this paper. In either case, an application may fail to apply security mechanisms required by the mandatory policy or may use security mechanisms in a way that is inconsistent with the user's intent. Mandatory security mechanisms may be used to ensure that security mechanisms are applied as required and can protect the user against inadvertent execution of untrustworthy applications. Although the user may have carefully defined the discretionary policy to properly implement the mandatory policy, an application may change the discretionary policy without the user's approval or knowledge. In contrast, the mandatory policy may only be changed by the system security policy administrator.

In the case of personal computing systems, where the user may be the system security policy administrator, mandatory security mechanisms are still helpful in

2. A discussion of the formal limitations of discretionary security mechanisms appears in [29].

protecting against flawed or malicious software. In the simplest case, where there is only a distinction between the user's ordinary role and the user's role as system security policy administrator, the mandatory security mechanisms can protect the user against unintentional execution of untrustworthy software. With a further subdivision of the user's ordinary role into various roles based on function, mandatory security mechanisms can confine the damage that may be caused by flawed or malicious software.

Although there are a number of commercial operating systems with support for mandatory security, none of these systems have become mainstream. These systems have suffered from a fixed notion of mandatory security, thereby limiting their market appeal. Furthermore, these systems typically lack adequate support for constraining trusted applications. In order to reach a wider market, operating systems must support a more general notion of mandatory security and must support flexible configuration of mandatory policies.

Mainstream commercial operating systems rarely support the principle of least privilege even in their discretionary access control architecture. Many operating systems only provide a distinction between a completely privileged security domain and a completely unprivileged security domain. Even in Microsoft Windows NT, the privilege mechanism fails to adequately protect against malicious programs because it does not limit the privileges that a program inherits from the invoking process based on the trustworthiness of the program [65].

Current microkernel-based research operating systems have tended to focus on providing primitive protection mechanisms which may be used to flexibly construct a higher-level security architecture. Many of these systems, such as the Fluke microkernel [23] and the Exokernel [41], use kernel-managed capabilities as the underlying protection mechanism. However, as discussed in [59], typical capability architectures are inadequate for supporting mandatory access controls with a high degree of flexibility and assurance. L4 [38] provides some support for mandatory controls through its *clans and chiefs* mechanism and its IPC mechanism for identifying senders and receivers but still lacks a coherent framework for using these mechanisms to meet the requirements of a mandatory policy. Furthermore, L4 assumes that there will only be a small number of distinct security domains [38]. Flask [56], a variant of the Fluke microkernel, provides a mandatory security framework similar to that of DTOS [43], a variant of the Mach microkernel; both systems provide mechanisms for mandatory access control and a mandatory policy framework.

Trusted path

A trusted path is a mechanism by which a user may directly interact with trusted software, which can only be activated by either the user or the trusted software and may not be imitated by other software [20]. In the absence of a trusted path mechanism, malicious software may impersonate trusted software to the user or may impersonate the user to trusted software. Such malicious software could potentially obtain sensitive information, perform functions on behalf of the user in violation of the user's intent, or trick the user into believing that a function has been invoked without actually invoking it. In addition to supporting trusted software in the base system, the trusted path mechanism should be extensible to support the subsequent addition of trusted applications by a system security policy administrator [28].

The concept of a trusted path can be generalized to include interactions beyond just those between trusted software and users. The TNI introduces the concept of a trusted channel for communication between trusted software on different network components [44]. More generally, a mechanism that guarantees a mutually authenticated channel, or *protected path*, is necessary to ensure that critical system functions are not being spoofed. Although a protected path mechanism for local communications could be constructed in application space without direct authentication support in the operating system, it is preferable for an operating system to provide its own protected path mechanism since such a mechanism will be simpler to assure [59] and is likely to be more efficient.

Most mainstream commercial operating systems are utterly lacking in their support for either a trusted path mechanism or a protected path mechanism. Microsoft Windows NT does provide a trusted path for a small set of functions such as login authentication and password changing but lacks support for extending the trusted path mechanism to other trusted applications [65]. For local communications, NT does provide servers with the identity of their clients; however, it does not provide the server identity to the client.

3 General Examples

This section argues that without operating system support for mandatory security and trusted path, appli-

cation-space mechanisms for access control and cryptography cannot be implemented securely. These arguments will then be used to reinforce the discussion in section 4, which analyzes concrete examples.

3.1 Access Control

An application-space access control mechanism may be decomposed into an enforcer component and a decider component. When a subject attempts to access an object protected by the mechanism, the enforcer component must invoke the decider component, supplying it with the proper input parameters for the policy decision, and must enforce the returned decision. A common example of the required input parameters is the security attributes of the subject and the object. The decider component may also consult other external sources in order to make the policy decision. For example, it may use an external policy database and system information such as the current time.

If a malicious agent can tamper with any of the components in the access control mechanism or with any inputs to the decision, then the malicious agent can subvert the access control mechanism. Even if the components and all of the inputs are collocated within a single file, the operating system security mechanisms are still relied upon to protect the integrity of that file. As discussed in the prior section, only mandatory security mechanisms can rigorously provide such integrity guarantees.

Even with strong integrity guarantees for the policy decision inputs, if an authorized user invokes malicious software, the malicious software could change an object's security attributes or the policy database's rules without the user's knowledge or consent. The access control mechanism requires a trusted path mechanism in the operating system in order to ensure that arbitrary propagation of access cannot occur without explicit authorization by a user.

If a malicious agent can impersonate the decider component to the enforcer component, or if a malicious agent can impersonate any source of inputs to the decision, then the malicious agent can subvert the mechanism. If any of the components or external decision input sources are not collocated within a single application, then the access control mechanism requires a protected path mechanism.

If a malicious agent can bypass the enforcer component, then it may trivially subvert the access control mechanism. Mandatory security mechanisms in the operating system may be used to ensure that all accesses to the protected objects are mediated by the enforcer component.

3.2 Cryptography

An analysis of application-space cryptography may be decomposed into an analysis of the invocation of the cryptographic mechanism and an analysis of the cryptographic mechanism itself. The analysis of this section draws from the discussions in [51][15] [60][61][55][52].

As an initial basis for discussion, suppose that the cryptographic mechanism is a hardware token that implements the necessary cryptographic functions correctly and that there is a secure means by which the cryptographic keys are established in the token. Even in this simplified case, where the confidentiality and integrity of algorithms and keys is achieved without operating system support, this section will demonstrate that there are still vulnerabilities which may only be effectively addressed with the features of a secure operating system.

One vulnerability in this simplified case is that invocation of the token cannot be guaranteed. Any legitimate attempt to use the token might not result in a call to the token. The application that performs the cryptographic invocation might be bypassed or modified by malicious applications or malicious users. Malicious applications might impersonate the cryptographic token to the invoking application.

Mandatory security and protected path features in the operating system address this vulnerability. Mandatory security mechanisms may be used to ensure that the application that invokes the cryptographic token is unbypassable and tamperproof against both malicious software and malicious users. Unbypassability could also be achieved by using an *inline* cryptographic token, which is physically interposed between the sender of the data to be protected and the receiver of the protected data; however, this would be less flexible. A protected path mechanism may be used to ensure that malicious software cannot impersonate the cryptographic token to the invoking application.

Misuse of the cryptographic token is a second vulnerability in the simplified case. Misuse may involve the use of a service, algorithm, session or key by an unauthorized application. Without operating system support for identifying callers, a cryptographic token can do little more than require that a user activate it, after which, any service, algorithm, session or key authorized for that

user may be used by any application on the system. In this case, the cryptographic token may be misused by applications operating on behalf of other users or may be misused by malicious software operating on behalf of the authorized user. Furthermore, unless the cryptographic token has a direct physical interface for user activation, malicious software can spoof the token to the user, obtain authentication information, and subsequently activate the cryptographic token without the user's knowledge or consent. Even with a direct physical interface to the user, it is impractical for the cryptographic token to require user confirmation for every cryptographic operation.

This second vulnerability may be addressed through mandatory security, trusted path and protected path features in the operating system. A trusted path mechanism obviates the need for a separate physical interface for activation. A protected path mechanism permits the cryptographic token to identify its callers and enforce fine-grained controls over the use of services, algorithms, sessions and keys. As an alternative to having the token deal with fine-grained controls over its usage, mandatory security mechanisms may also be used to provide such controls. For example, mandatory security mechanisms may be used to isolate the token for use only by applications executed by the user who activated the token. Furthermore, the mandatory security mechanisms can reduce the risk of malicious software being able to use the cryptographic token and may consequently limit the use of the trusted path mechanism to highly sensitive actions.

Hence, even in the simplest case, the features of a secure operating system are crucial to addressing the vulnerabilities of application-space cryptography. In the remainder of this section, the assumptions of the simplified case are removed, and the additional vulnerabilities are examined.

If the assumption that initial keys are securely established within the token is removed, then there is the additional vulnerability that the initial keys may be observed or modified by an unauthorized entity. Unless the initial keys are provided via a dedicated physical interface to the cryptographic token, the operating system must protect the path between the initial key source and the cryptographic token and may need to protect the initial key source itself. Mandatory security mechanisms may be used to rigorously protect the path and the key source. A trusted path may be required for initial keying.

If the assumption that the cryptographic mechanism is confined to a single hardware token is removed and implemented in software instead, the confidentiality and integrity of the cryptographic mechanism's code and data becomes dependent on the operating system, including both memory protection and file protection. Mandatory security is needed to rigorously ensure the mechanism's integrity and confidentiality. If any external inputs, such as input parameters to a random number generator, are used by the cryptographic mechanism, the input sources and the path between the input sources and the cryptographic mechanism must be protected with mandatory security mechanisms.

4 Concrete Examples

This section further demonstrates that secure operating systems are necessary by showing that some widely accepted security solutions critically rely on the features of secure operating systems. In particular, this section examines mobile code security efforts, the Kerberos network authentication system, firewalls and network security protocols.

4.1 Mobile Code

A number of independently-developed security solutions for the World Wide Web, each with its own protection model, have been developed to protect against the threats from malicious mobile code. However, systems relying on these security solutions are vulnerable because of a lack of operating system support for security. Primarily, this section will emphasize this point by focusing on efforts to secure Java [27], but other efforts will also be used to highlight issues.

The primary threat that these solutions attempt to address is the threat of hostile mobile code gaining unauthorized access to a user's files and resources in order to compromise confidentiality or integrity. The threat is not limited to interpreted applets loaded from the network by a web browser; both [26] and [30] extend this threat model to include helper applications which may have been actively installed by a user. There is little distinction between mobile code and what is traditionally considered data. For example, consider that Postscript documents are actually programs with potential access to the local filesystem. Consequently, helper applications which operate on untrustworthy data, such as Postscript viewers, must either be executed in a less flexible mode of operation, or must be carefully confined by the operating system.

The basic Java Security Model is based on the notion of "sandboxing." The system relies on the type-safety of the language in conjunction with the Java Security Manager to prevent unauthorized actions [27]. Efforts are currently underway to add additional security features to Java, such as capabilities, an expanded access control model, or additional controls over access to certain class libraries [70].

The fundamental limitation of these approaches is that none can be guaranteed to be tamperproof or unbypassable. For example, although the Java language is claimed to be secure, the Java Virtual Machine (JVM) will accept byte code which violates the language semantics and which can lead to security violations [32]. JVM implementation errors have led to violations of the language's semantics [19]. A significant portion of the Java system is currently in the form of native methods which are implemented as object code and are not subject to the JVM's type-safety checks. The JVM is not able to protect itself from tampering by other applications. Finally, the Java security model can offer no protection from the many other forms of malicious mobile code. In [30], the authors call for trusted systems to support a system-wide solution to address the threats presented by non-Java code.

Even if such problems with the JVM did not exist, these security solutions would still suffer from the fundamental limitation that they rely on application-space access control for security. They all depend on the local file system to preserve the integrity of the system code, including class files. All of the systems which store policy locally depend on file system access control to preserve the integrity of the policy files. Section 3.1 demonstrated the importance of secure operating system features for supporting application-space access control.

Another popular approach to "securing" mobile code is to require digitally signed applets and limit execution to those originating from trusted sources [27]. In fact, native ActiveX security is based entirely on digital signatures, as it has no form of access control [24][27]. The basic flaw with this approach is that it is an all-or-nothing proposition; the user cannot constrain a native ActiveX control to a limited security domain. Mandatory security mechanisms in the operating system may be used for this purpose, by confining the browser to a distinct security domain.

Note that, although not sufficient by themselves, digital signatures will play an important part in mobile code security, even on secure operating systems. They can reduce the risk of malicious code entering the system, provide some measure of trust that an applet will behave properly, and provide another piece of information to use in making an access control decision. However, as with the general application-space cryptography described in section 3.2, the digital signature verification mechanism depends on secure operating system features to guarantee invocation, to protect the integrity of the mechanism, and to protect the integrity of the locally cached public keys.

The need for an operating system trusted path mechanism was highlighted by [67] which demonstrates the ease with which a trojan horse applet can capture credit card numbers, PIN numbers or passwords by perfectly emulating a window system dialog box. The proposed solution was an ad hoc user-level trusted path mechanism which required a user to customize his dialog box with a complicated graphical pattern. This solution is not adequate as it only increases the sophistication required in the trojan horse.

Other systems attempt to provide alternative security solutions to the mobile code threat. The Janus system [26] interposes on Solaris system calls to constrain untrusted native applications, and Safe-Tcl [49] provides a "safe interpreter" which attempts to limit the command set available to untrusted code. However, like the Java security solutions, these systems are subject to the same vulnerabilities as any other application-space access control mechanism; consequently, they require secure operating system support.

Beyond enabling all of the mobile code systems mentioned above to function securely, a secure system could also simplify them. Rather than implementing their security primitives in application space where they are vulnerable, they could utilize the system security services to provide a better overall system. A properly designed secure system would provide a flexible, economic foundation with one consistent security model for all of the different virtual machine efforts to use.

4.2 Kerberos

Kerberos [31][47] is a network authentication service originally developed for Project Athena at MIT. In addition to providing an authentication service, Kerberos supports the establishment of session keys to support network confidentiality and integrity services. Derivatives of Kerberos have been used to provide authentication and key establishment services for AFS [64], DCE [53], and ONC RPC [21]. Kerberos and sys-

tems that rely on Kerberos have been suggested as a means of providing security for the World Wide Web [18][36][37].

Kerberos is based on symmetric cryptography with a trusted key distribution center (KDC) for each realm. The Kerberos KDC has access to the secret key of every principal in its realm. Consequently, a compromise of the KDC can be catastrophic. This is generally addressed by requiring that the KDC be both physically secure and dedicated solely to running the Kerberos authentication server [46].[3] A typical environment also uses physically-secure dedicated systems for the servers using Kerberos. Without these environmental assumptions, the Kerberos authentication service and the Kerberized server applications would require secure operating system features to rigorously ensure that they are tamperproof and unbypassable. For the sake of argument, the remainder of this section will consider these environmental assumptions to be true and focus only on the security of the client workstations.

Kerberos was designed for an environment where the client workstations and the network are assumed to be completely untrustworthy [10][45]. However, since the software on the client workstation mediates all interactions between its user and the Kerberized server applications, this assumption implies that the Kerberized server applications must view all client applications as potentially malicious software. Furthermore, a Kerberized server application has no means of establishing a trusted path to a user on a client workstation, since that would require trusted code on the client workstation. Thus, in a system that uses Kerberos, malicious software executed by a user is free to arbitrarily modify or leak a user's information, with no means of confinement; no distinctions between a user's legitimate requests and the requests of malicious software are possible. Given the increasing ease with which malicious software may be introduced into a system, the Kerberos environmental model seems untenable. As noted in [14], secure end-to-end transactions require trusted code at both end points.

As a basis of further discussion, suppose that there is a base set of trustworthy software on the client workstations which is protected against tampering, but that the client workstation operating system still lacks mechanisms for mandatory security and trusted path. Furthermore, suppose that the client workstation is a single-user system which does not export any services to other systems. In spite of these assumptions, a user is still vulnerable to attacks by malicious software, such as mobile code downloaded by the user.

If the malicious software could spoof the client-side authentication program to the user, then it may be able to obtain a user's password. Even with one-time passwords, this attack would permit the malicious software to act on behalf of the user during the login session. A trusted path mechanism in the client workstation's operating system can be used to prevent such an attack. Additionally, such a trusted path mechanism in combination with support for a network protected path can be used to provide a trusted path between users and server applications.

If the malicious software can read the files used by the Kerberos client software to store tickets and session keys, then the malicious software may directly impersonate the user to the corresponding Kerberized server applications. Even if the session keys are encapsulated within a hardware cryptographic token, the malicious software can invoke the cryptographic token on behalf of the user, exploiting the misuse vulnerability discussed in section 3.2. Mandatory security mechanisms can be used to rigorously protect either the file or the cryptographic token against access by malicious software.

4.3 Network Security Protocols

The IPSEC network security protocols [5][3][4] are used to provide authentication, integrity, and confidentiality services at the IP layer. Typical implementations of the IPSEC protocols rely on application-space key management servers to perform key exchanges and supply keys for security associations. The IPSEC module in the network stack communicates with the local key management server via upcalls to retrieve the necessary information.

SSL [69] is another network security protocol that provides authentication, integrity, and confidentiality services and a negotiation service for keys and cryptographic algorithms. SSL, however, is implemented entirely in application space and requires no kernel modifications. SSL has been implemented as a library that interposes on socket calls to incorporate the SSL protocol between the underlying transport protocol of the socket (e.g., TCP) and the application protocol (e.g., HTTP).

3. Variants of Kerberos have been proposed that use asymmetric cryptography either to reduce the cost incurred by a penetration of the KDC or to completely eliminate the need for the KDC [63] [66][42][18].

Since it relies on application-space cryptography, the key management server used by IPSEC is subject to the vulnerabilities described in section 3.2 and requires mandatory security mechanisms in the operating system for adequate protection. In turn, since the protection provided by IPSEC depends on the protection of the keys, mandatory security mechanisms in the operating system are also crucial to meeting the security requirements of IPSEC. Since the complete SSL implementation operates in application space, it is directly subject to the vulnerabilities described in section 3.2 and requires mandatory security mechanisms in the operating system for adequate protection.

Both IPSEC and SSL are intended to provide secure channels. However, as noted in [14], an end-to-end secure transaction requires a secure channel and secure end points. If an attacker can penetrate one of the end points and directly access the unprotected data, then the protection provided by IPSEC and SSL is only illusory.

4.4 Firewalls

A network firewall is a mechanism for enforcing a trust boundary between two networks. The analysis of this section is based on the discussions in [17][9][11][6]. Commonly, firewalls are used to maintain a separation between insiders and outsiders for an organization's computing resources. Internal firewalls may also be used to provide separation between different groups of insiders or to provide defense-in-depth against outsiders.

Modern firewall architectures typically involve the use of bastion hosts; in a screened subnet architecture, there may be an external bastion host on a perimeter network, which is highly exposed to outsiders, and an internal bastion host on the internal network, which is exposed to the external bastion host. The security of the bastion hosts is crucial to the security provided by the firewall. To reduce risk, bastion hosts are typically dedicated systems, only providing the minimal services required. Even with such minimal configuration, flaws in the proxy servers on the bastion host may permit penetration. However, mandatory security mechanisms in the operating systems of the bastion hosts may be used to confine proxy servers so that penetrations are narrowly limited. Similarly, the bastion host's mandatory security mechanisms may be used to protect proxy servers against tampering.

Firewalls provide no protection against malicious insiders. Typically, insiders can easily leak information through the firewall. Malicious insiders can construct tunnels to permit outsiders to perform inbound calls through the firewall or may provide ways of bypassing a firewall entirely. Additionally, malicious insiders can exploit data leaked between users within the firewall. Although internal firewalls may be used to partition insiders into multiple trust classes, the granularity of protection is quite limited in comparison to what can be provided by a secure operating system.

The ability of malicious insiders to leak data through the firewall can be confined by mandatory security mechanisms in the operating systems of the internal hosts. Likewise, mandatory security mechanisms in the operating systems of the internal hosts can confine outsiders who perform inbound calls through tunnels constructed by a malicious insider to the security domains in which the malicious insider is allowed to operate.

In addition to the threat of malicious insiders, a firewall is at risk from the threat of malicious software executed by benign insiders. Typically, firewalls do not require that insiders strongly authenticate themselves to the firewall in order to access external services through the firewall [40]. Hence, if a benign insider executes malicious software on an internal host, the malicious software may seek to subvert the protection of the firewall in the same fashion as a malicious insider. An example of using a malicious Java applet to enable outsiders to penetrate a firewall is given in [40]. Even if insiders are required to strongly authenticate themselves to the firewall, a benign insider may still execute a trojan horse whose overt purpose requires external access; in this case, the malicious software may still subvert the protection of the firewall.

Mandatory security mechanisms in the operating systems of the internal hosts may be used to protect users against execution of malicious software or to confine such software when it is executed. If strong authentication is required prior to accessing external services, mandatory security mechanisms could be used to ensure that only trustworthy software on the internal hosts can communicate with the strong authentication mechanism on the firewall. In any case, the mandatory security mechanisms would limit the ability of malicious software to leak information or support inbound calls.

Firewalls are also susceptible to malicious data attacks [62]. Some example malicious data attacks relevant to firewalls are described in [68][40][16]. As with malicious insiders and malicious software, mandatory security mechanisms in the operating systems of the

bastion hosts and the internal hosts may be used to confine malicious data attacks.

When inbound services are supported by a firewall, the firewall itself cannot protect the remote system against compromise. The remote system's operating system must protect against misuse of the allowed inbound services and must protect any information acquired through the inbound service against leakage. Mandatory security mechanisms in the remote system's operating system may be used to provide such protection. Additionally, mandatory security mechanisms in the internal host's operating system are needed to confine any attack from a penetrated remote system.

When a benign insider wishes secure access to a remote service, the firewall itself cannot provide complete protection for the use of the remote service. The internal host's operating system must protect against any attempts by the server to trick the client into misusing its privileges, as in the case where a browser executes a malicious applet provided by a server; mandatory security mechanisms in the internal host's operating system may be used to confine these client applications.

5 System Security

No single technical security solution can provide total system security; a proper balance of security mechanisms must be achieved. Each security mechanism provides specific security functions and should be designed to only provide those functions. It should rely on other mechanisms for support and for required security services. In a secure system, the entire set of mechanisms complement each other so that they collectively provide a complete security package. Systems that fail to achieve this balance will be vulnerable.

As has been shown throughout this paper, a secure operating system is an important and necessary piece to the total system security puzzle, but it is not the only piece. A highly secure operating system would be insufficient without application-specific security built upon it. Certain problems are actually better addressed by security implemented above the operating system. One such example is an electronic commerce system that requires a digital signature on each transaction. A application-space cryptographic mechanism in the transaction system protected by secure operating system features might offer the best system security solution.

No single security mechanism is likely to provide complete protection. Unsolved technical problems, implementation errors and flawed environmental assumptions will result in residual vulnerabilities. As an example, covert channels remain a serious technical challenge for secure operating system designers. These limitations must be understood, and suitable measures must be taken to deploy complementary mechanisms designed to compensate for such problems. In the covert channel example, auditing and detection mechanisms should be utilized to minimize the chances that known channels are exploited. In turn, these should depend on secure operating systems to protect their critical components, such as audit logs and intrusion sensors, because they are subject to the same types of vulnerabilities as those discussed throughout this paper.

6 Summary

This paper has argued that the threats posed by the modern computing environment cannot be addressed without secure operating systems. The critical operating system security features of mandatory security and trusted path have been explained and contrasted with the inadequate protection mechanisms of mainstream operating systems. This paper has identified the vulnerabilities that arise in application-space mechanisms for access control and cryptography and has demonstrated how mandatory security and trusted path mechanisms address these vulnerabilities. To provide a clear sense of the need for these operating system features, this paper has analyzed concrete examples of current approaches to security and has shown that the security provided by these approaches is inadequate in the absence of such features. Finally, the reader was given a perspective of system security where both secure operating systems and application-space security mechanisms must complement each other in order to provide the correct level of protection.

By arguing that secure operating systems are indispensable to system security, the authors hope to spawn a renewed interest in operating system security. If security practitioners were to more openly acknowledge their security solution's operating system dependencies and state these dependencies as requirements for future operating systems, then the increased demand for secure operating systems would lead to new research and development in the area and ultimately to commercially viable secure systems. In turn, the availability of secure operating systems would enable security practitioners to concentrate on security services that belong in their particular components rather than dooming them to try to address the total security problem with no hope of success.

7 References

[1] M. Abrams et al, *Information Security: An Integrated Collection of Essays*, IEEE Comp. 1995.

[2] J. Anderson, *Computer Security Technology Planning Study*, Air Force Elect. Systems Div., ESD-TR-73-51, October 1972.

[3] R. Atkinson. IP Authentication Header (AH). IETF RFC 1826, August 1995.

[4] R. Atkinson. IP Encapsulating Security Payload (ESP). IETF RFC 1827, August 1995.

[5] R. Atkinson. Security Architecture for the Internet Protocol. IETF RFC 1825, August 1995.

[6] Badger et al. DTE Firewalls, Initial Measurement and Evaluation Report. Trusted Information Systems Technical Report #0632R, March 1997.

[7] L. Badger et al. Practical Domain and Type Enforcement for UNIX. *Proceedings of IEEE Symposium on Security and Privacy*, May 1995.

[8] D. Baker. Fortresses Built Upon Sand. *Proceedings of the New Security Paradigms Workshop*, 1996.

[9] S. Bellovin and W. Cheswick. Network Firewalls. *IEEE Communications*, September 1994.

[10] S. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *Computer Communications Review 20(5)*, October 1990.

[11] B. Blakley. The Emperor's Old Armor. *Proceedings of the New Security Paradigms Workshop*, 1996.

[12] W. Boebert and R. Kain, A Further Note on the Confinement Problem. *Proceedings of the 30th IEEE International Carnahan Conference on Security Technology*, 1996.

[13] W. Boebert and R. Kain. A Practical Alternative to Hierarchical Integrity Policies. *Proceedings of the 8th National Computer Security Conference*, 1985.

[14] E. Brewer at al. Basic Flaws in Internet Security and Commerce. http://http.cs.berkeley.edu/~gauthier/endpoint-security.html, 1995.

[15] W. Brierley. Integrating Cryptography into Trusted Systems: A Criteria Approach. *Proceedings of the 8th IEEE Conference on Computer Security Applications*, 1992.

[16] Computer Emergency Response Team. Advisory 93:16.

[17] D. Chapman and E. Zwicky. *Building Internet Firewalls*. O'Reilly, 1995.

[18] D. Davis. Kerberos Plus RSA for World Wide Web Security. *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, July 1995.

[19] D. Dean et al. Java Security: From HotJava to Netscape and Beyond. *Proceedings of the IEEE Symposium on Security and Privacy*, 1996.

[20] DOD 5200.28-STD. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.

[21] M. Eisler et al. Security Mechanism Independence in ONC RPC. *Proceedings of the 6th USENIX UNIX Security Symposium*, July 1996.

[22] D. Ferraiolo and R. Kuhn. Role-Based Access Control. *Proceedings of the 15th National Computer Security Conference*, 1992.

[23] B. Ford et al. Microkernels Meet Recursive Virtual Machines. *Proceedings of 2nd USENIX Symposium on Operating Systems Design and Implementation*, October 1996.

[24] S. Garfinkel. *Web Security and Commerce*. O'Reilly & Associates, Cambridge, 1997.

[25] M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Company, New York, 1988.

[26] I. Goldberg et al. A Secure Environment for Untrusted Helper Applications. *Proceedings of 6th USENIX Unix Security Symposium*, July 1996.

[27] L. Gong. Java Security: Present and Near Future. *IEEE Micro*, May/June 1997.

[28] R. Graubart. Operating System Support for Trusted Applications. *Proceedings of the 15th National Computer Security Conference*, 1992.

[29] M. Harrison et al. Protection in Operating Systems. *Communications of the ACM 19(8)*, August 1976.

[30] T. Jaeger et al. Building Systems that Flexibly Control Downloaded Executable Content. *Proceedings of the 6th USENIX Security Symposium*, July 1996.

[31] J. Kohl and C. Neuman. The Kerberos Network Authentication Service V5. IETF RFC 1510, September 1993.

[32] M. Ladue. When Java Was One: Threats from Hostile Byte Code. *Proceedings of the 20th National Information Systems Security Conference*, 1997.

[33] B. Lampson. A Note on the Confinement Problem. *Communications of the ACM 16(10)*, 1973.

[34] B. Lampson et al. Authentication in Distributed Systems: Theory and Practice. *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1992.

[35] J. Lepreau et al. The Persistent Relevance of the Local Operating System to Global Applications. *Proceedings of the 7th ACM SIGOPS European Workshop*, September 1996.

[36] S. Lewontin. The DCE-Web Toolkit. *Proceedings of the 3rd International World Wide Web Conference*, 1995.

[37] S. Lewontin and M. Zurko. The DCE Web Project: Providing Authorization and Other Distributed Services to the World Wide Web. *Proceedings of the 2nd International World Wide Web Conference*, 1994.

[38] J. Liedtke. L4 Reference Manual. Research Report RC 20549, IBM T. J. Watson Research Center, September 1996.

[39] T. Linden. Operating System Structures to Support Security and Reliable Software. *ACM Computing Surveys* 8(4), Dec. 1976.

[40] D. Martin et al. Blocking Java Applets at the Firewall. *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, 1997.

[41] D. Mazieres and M. Kaashoek. Secure Applications Need Flexible Operating Systems. *Proceedings of the 6th Workshop on Hot Topics in Operating Systems*, May 1997.

[42] A. Medvinsky et al. Public Key Utilizing Tickets for Application Servers. IETF Draft Jan 1997 expires July 1997.

[43] S. Minear. Providing Policy Control Over Object Operations in a Mach Based System. *Proceedings of the 5th USENIX Security Symposium*, April 1995.

[44] NCSC-TG-005. Version 1. *NCSC Trusted Network Interpretation*, July 1987.

[45] C. Neuman and J. Steiner. Authentication of Unknown Entities on an Insecure Network of Untrusted Workstations. *Proceedings of the Usenix Workshop on Workstation Security*, August 1988.

[46] C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, September 1994.

[47] C. Neuman et al. The Kerberos Network Authentication Service V5 R6. IETF Draft July 1997, expires Jan 1998.

[48] R. O'Brien and C. Rogers. Developing Applications on LOCK. *Proceedings of the 14th National Computer Security Conference*, 1991.

[49] J. Ousterhout et al. The Safe-Tcl Security Model. Sun Labs Technical Report TR-97-60, March 1997.

[50] President's Commission On Critical Infrastructure Protection. *Research and Development Recommendations for Protecting and Assuring Critical National Infrastructures*, September 1997

[51] M. Roe and T. Casey. Integrating Cryptography in the Trusted Computing Base. *Proceedings of the 6th IEEE Conference on Computer Security Applications*, 1990.

[52] RSA Laboratories. Public Key Cryptography Standard No. 11 - Cryptoki Version 2.0. RSA Laboratories, pp. 24-25, April 1997.

[53] R. Salz. DCE 1.2 Contents Overview. Open Group RFC 63.3, October 1996.

[54] J. Saltzer and M. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9), September 1975.

[55] B. Schneier. *Applied Cryptography, 2nd Edition*. John Wiley & Sons, New York, 1996. p. 169-187, 216-225.

[56] Secure Computing Corporation. *Assurance in the Fluke Microkernel:: Formal Security Policy Model*, Technical report MD A904-97-C-3047 CDRL A003, March 1998.

[57] Secure Computing Corporation. *DTOS Covert Channel Analysis Plan*, Technical report MD A904-93-C-4209 CDRL A017, May 1997.

[58] Secure Computing Corporation. *DTOS Generalized Security Policy Specification*, Technical report MD A904-93-C-4209 CDRL A019 June 1997. (http://www.securecomputing.com/randt/HTML/dtos.html)

[59] Secure Computing Corporation. *DTOS General System Security and Assurability Assessment Report*, Technical report MD A904-93-C-4209 CDRL A011 June 1997. (http://www.securecomputing.com/randt/HTML/dtos.html)

[60] Secure Computing Corporation. *LOCKed Workstation Cryptographic Services Study*, Technical Report MD A904-94-C-6045 CDRL A009, September 1995. .

[61] Secure Computing Corporation. *Security Requirements Specification and Requirements Rationale Report for the Technical Study Demonstrating the Feasibility of Software-Based Cryptography on INFOSEC Systems*, Technical report MDA904-91-C-7103 CDRL A011 and A012, May 1994.

[62] W. Sibert. Malicious Data and Computer Security. *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[63] M. Sirbu and J. Chuang. Distributed Authentication in Kerberos using Public Key Cryptography. *Proceedings of the Symposium on Network and Distributed System Security*, 1997.

[64] M. Spasojevic and M. Satyanarayanan. An Empirical Study of a Wide-Area Distributed System. *ACM Transactions on Computer Systems 14(2)*, May 1996.

[65] S. Sutton and S. Hinrichs. MISSI B-level Windows NT Feasibility Study Final Report. Technical Report, NSA MISSI Contract MDA904-95-C-4088, December 1996.

[66] B. Tung et al. Public Key Cryptography for Initial Authentication in Kerberos. IETF Draft expires Jan 1998.

[67] J. Tyger and A. Whitten. WWW Electronic Commerce and Java Trojan Horses. *Proceedings of the 2nd Usenix Workshop on Electronic Commerce*, November 1996.

[68] W. Venema. Murphy's Law and Computer Security. *Proceedings of the 6th USENIX Unix Security Symposium*, 1996.

[69] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, November, 1996.

[70] D. Wallach et al. Extensible Security Architectures for Java. Technical Report 546-97, Dept. of Computer Science, Princeton University, April 1997.