Layered multipoint network defense and security policy enforcement

Stephen D. Wolthusen

Abstract— **This paper discusses the enhancement of security in general purpose operating systems, especially related to threats caused by internetworking, using extensions to operating systems. Such mechanisms have a significantly larger basis for reaching security policy decisions than older host-level security mechanisms and firewalls. By layering defensive mechanisms yet enforcing a consistent security policy across the security layers, goals such as workload distribution, vulnerability compartmentalization, and hierarchical refinement of security policies can be achieved.**

Keywords— **Security Policy, Operating System Extension, Firewalling, Access Control, Auditing**

I. Introduction

Internetworking computers and virtually any electronic device has become a largely unquestioned trend. Together with an ever-increasing reliance on such networks concomitant with a rise in the sensitivity of the data and applications used on them, this results in a sizable need for enforcing security and integrity requirements.

However, current COTS systems from operating systems to network firewalls do not fulfill these requirements or have been slow to react to a changing threat environment. In addition, while many applications now are network-aware, security is in many cases not integrated effectively.

While it would be desirable to perform a thorough review of security requirements in light of this changed enviroment and to create a set of mechanisms embedded into the entire infrastructure to meet these, such an approach is not viable mainly due to the large investments already in place in existing IT systems and the need for a homogeneous environment. This is reflected in the limited appeal of such attempts performed thus far [1].

What is apparently called for is a pragmatic mechanism to meet or at least partially address these requirements while retaining compatibility with existing systems. This can be achieved by retrofitting the necessary mechanisms to COTS software systems if one is willing to accept limitations in terms of overall assurance.

The main challenges here are that any such mechanism must be workable across node and system platform boundaries to realize a comprehensive security mechanism in an internetworked heterogeneous system, and that the security mechanisms must not necessitate changes to application programs by breaking the existing expected system behavior in an egregious way other than where explicitly required for security reasons that are to be specified in a separately defined policy. Finally, such a mechanism should also be as transparent to users as possible as long as they do not violate the security policy in effect.

S. Wolthusen is with the Security Technology Department, Fraunhofer-IGD, Darmstadt, Germany. E-mail: wolt@igd.fhg.de.

The first item is an obvious requirement; any non-trivial network will be heterogeneous in nature. The second requirement derives from the observation that it is very difficult to modify any application program typically because it is also a COTS product, or because any modification to custom applications would require major investments even if such a move were contemplated.

Another criterion in the design of the mechanisms described here is that they must be completely neutral regarding the policy that is to be enforced.

II. Threat Environment

Most networks today are segregated into internal levels of various trust levels and the outside world. Protection of the individual layers occurs using firewalls or guard mechanisms. These mechanisms rely on several assumptions, typically tacit, for their validity:

- The internal (protected) network is trustworthy
- There is a meaningful distinction between internal and external networks
- Attacks are initiated from external hosts directed at internal networks
- The syntax and semantics of the protocols used are known
- Code passing through a firewall is executed with the consent of a user.

While all of these assumptions were true at some point and are reflected in the design of firewalling mechanisms, most of them gradually lost their validity at some point in time; any new mechanism for countering the current threat environment may no longer make use of them. This section highlights some of the threats that need to be countered.

A. Mobile Devices and Remote Access

The topology of a network can no longer be used as the linchpin of its security [2]. Bellovin discussed this mainly in terms of performance [3], but network structures including mobile devices and remote access solutions render this if not obsolete then at least secondary. It must be assumed that a mobile device may, possibly using a transitive and therefore unanticipated network connection, contact sensitive network nodes. A similar scenario results if a user has established a remote access connection to a protected network and also has for some reason or accidentally established another network connection, possibly opening a routable connection to the Internet bypassing all protection and auditing mechanisms in the process.

B. Mobile Code

Many current applications include scripting mechanisms of varying levels of expressiveness; in addition, mechanisms such as Sun's Java or Microsoft's ActiveX permit the transmission and execution of full applications. While most of these mechanisms contain security mechanisms, the track record in using these is not encouraging if only because deactivating the macro language mechanisms would render some documents unusable. A threat assessment must therefore assume that malicious code can reach a sensitive network node.

Trojan horses of various levels of intricacy are another threat that can be classified as mobile code; given that the execution of such code is frequently caused by a social factor, technical means probably cannot fully prevent such infiltration of sensitive nodes.

C. Tunneling Mechanisms

A number of application programs can transmit data and instructions across transmission channels designed for different purposes. In some cases, this is done surreptitiously by application programs, in other cases the tunneling occurs deliberately. The most notable recent example for such a mechanism is the SOAP[4], following similar earlier developments in the CORBA area. In both cases exposed application interfaces are called without intervening security mechanisms. The entire responsibility for security is placed on the developers of the thus exposed applications and their protocols. Even advanced application-level firewalls can not necessarily detect such mechanisms since the tunneled data conforms to the syntax of the "host" protocol except by resorting to ad-hoc detection schemes.

D. Application Complexity

Even a simple WWW application today typically involves a complex set of protocols and applications ranging from advanced HTML parsers and display engines – which themselves may be vulnerable – to JavaScript and may involve implicit operations and code execution on a client. Very little of this is perceived consciously by a user or can even be controlled at any meaningful level of granularity. Distinguishing deliberate actions on the part of the user from implicit operations due to application behavior or malicious operations resulting from mobile code (see section II-B) represents a considerable difficulty. This limits the effectiveness of most reactive intrusion detection systems [5], [6]. A related problem occurs when trying to specify permitted application behavior for intrusion detection mechanisms. Full specifications of generic COTS or even reasonably complex custom applications are very hard to obtain. Linking permitted operations on the part of either a user or of the application to observed behavior by the application will therefore frequently have to resort to heuristics or imprecise specifications. In doing so one is confronted with missing significant events or having to accept a considerable false-positive frequency [7].

III. Centralized Security Policy

Even though several of the threats outlined in section II are distributed in nature, any meaningful security policy must at least define a baseline for an organization that is valid for all nodes in such a network. When referring to the term security policy we here restrict ourselves to a definition that includes only rules that may be enforced by technical means. Items covered by such rules include any operations performed by users, applications acting on behalf of users, or of the node itself. Wherever possible, a security mechanism must attempt to tie such operations to the highest-level principals that can be detected, ideally individual users.

While this paper considers mainly the networking aspects of such an approach, it is obvious that the mechanism is general in applicability. We do not describe a specific security policy in this paper; the mechanisms described here are agnostic in this regard. However, we propose to use a role-based access control mechanism extended with additional rules regarding behavior of principals and operations on objects for this purpose. This allows the modeling of both simple discretionary policies as well as more stringent lattice-based models.

A. Decentralized Enforcement

The principle of separating security policy from its enforcement is now generally accepted [8]. While originally intended for use inside an individual node, we propose to use the same principle across a distributed system. Enforcement is to be performed at the operating system level of each individual node, based on decisions either related to the enforcement subsystem directly from a node distributing policy data and decisions or as a result of a delegated derived security policy.

This separation of concern between end nodes enforcing security policy and nodes controlling policy data can be performed using externally controlled reference monitors (ECRM). Using a proper balance between centralized decisions and locally delegated security policies, the overall network load – which is limited by latency, not bandwidth – can be kept at an acceptable level.

The ECRM mechanism is based on the segregation of policy decisions from enforcement by splitting the reference monitor into local node components (ECRM) and external policy repositories or external reference monitors (ERM), both of which can be confined to a secure coprocessor containing sensitive mechanisms. The ERM nodes contain security policy data for which they are either authoritative or acting as a cache instance. Using a suitable policy reconciliation mechanism, the handling of operations involing subjects and objects from multiple security policy domains can be resolved. The communication between an ECRM and an ERM may involve individual decisions such as granting read access to a data object which is valid only for the single instance it was requested or it may result in temporary local delegation of a derived subset of the security policy depending on the type of access and the capabilities identified (see section IV-B).

Not all decisions and enforcement mechanisms can be implemented effectively inside the ECRM, particularly when the mechanism was retrofitted onto an existing operating system; in other instances it is necessary for performance reasons to further delegate policy decisions (still derived from the central policy and interpreted by the ECRM) to individual modules inside the system. While this affects the assurance that can be obtained it does not, in our opinion, invalidate the ECRM mechanism in itself.

Details on the externally controlled reference monitor mechanism can be found in [9].

B. Hierarchical Refinement

An organization's baseline policy may be refined hierarchically where subordinate policies may only place additional restrictions onto the rule base.

Where principals and objects of different organizations overlap, a policy reconciliation mechanism is typically required to resolve the conflicting individual rules applied to these principals and objects. The reconciliation policy must be defined by the respective security administrators involved.

The overall policy set must be enforced consistently across the entire distributed system. When tying policy rules to high-level entities this implies that any entity performing such decisions or operations must be aware of the unified set of policy rules. If one accepts the possibility of propagation delays, this permits the keeping of both caches and localized security policy rule bases. The extent of propagation delays can be regulated by means of specifying a lifetime with each rule after which the policy element originator must be consulted.

IV. Node-Internal Layering

The defensive posture inside an individual node must should multi-layered. The term "layering" here refers mainly to a layering of abstraction layers to be protected; an actual implementation will usually involve several cooperating and intermeshing components at different levels of an operating system.

In designing such a mechanism, the issues outlined in section II need to be taken into consideration. To permit a basic protection mechanism, an outer layer of coarse granularity – in the interest of limiting minimum policy complexity – is necessary that surrounds the entire node. This implies all inputs and output that is handled by a node, including network traffic and file systems.

The general mechanism is to embed low-level security mechanisms that catch all I/O traffic and serve as basic protection. At other levels within the operating system, higher layer protocols are also instrumented. These layers must then cooperate for two purposes. First, the necessary information regarding the participating subjects and objects as well as the intended operations is typically dispersed through several data structures within an operating system and must be consolidated in any case before a decision regarding security policy rules can be made. Secondly, if a security policy rule exists for a higher-level protocol and a protocol data unit can be identified as belonging to such a transaction, the lower-level mechanisms must defer to such a decision.

The mechanism outlined above requires a security policy that is structured in such a way that lower-level protocols start with a default-deny stance and progressive relaxation occurs for specific instances that are identified as necessary operations involving authorized principals according to the risk analysis that preceded the definition of the security policy.

One implication of such a mechanism is that any security policy rule that is either not within the capabilities of a given node can therefore be covered by a lower-level protection layer. It is important to note, however, that the protection is provided only if the security policy itself is consistent with the construction rule outlined above.

A. State-Based Policy Decisions

The mechanisms described here are required to interact on an implementation level; additional benefits can also be derived from logically consolidating the information obtained from the various layers and using them as decision data points for a security policy rule.

To support this, the security policy specification mechanism must provide support for specifying sequences and decision trees.

The main difficulties with using such mechanisms are again in the overhead that is created by enforcing such rules and in the need for specifying meaningful rule sets in the first place. We therefore assume that the application of state-based policy decision mechanisms will be limited to a small number of well-defined critical application areas.

B. Layer Capabilities

We assume that the system described here will evolve over time by adding additional layers and protocols to its suite of supported mechanisms. The only requirement is that the most fundamental layers (described in sections IV-C, IV-D, and IV-E) are present.

Since the performance impact caused by having to consult a security policy for matching rules can be considerable if the rule base is too complex, a simple optimization for an ECRM is to provide the ERM with a capability vector in issuing a request for a local policy delegation (in case of an immediate policy decision request this optimization is irrelevant). The capability vector, issued by the trusted ECRM subsystem, identifies the protection layers that are implemented in the node issuing the request.

If supported by security policy specification language, the policy itself may also contain rules regarding specific capability implementations (or assurance levels) such as the presence of an ECRM implemented by means of a secure coprocessor and e.g. their influence on granting access to certain data.

C. Device Interface Layer

The lowest layer of protection mechanisms involve access control to the interfaces such as e.g. network interfaces, se-

rial communication lines, and SCSI interfaces. At least for some of these interfaces, current operating systems provide such services; they simply must be placed under the control of the centralized security policy enforcement. The level of control is typically at the level of the principal in the system security concept. We can extend this mechanism by employing a finer granularity access control mechanism identifying additional subjects such as application programs; this is elaborated below. Both application programs and users need not be aware of such an extension since the mechanisms for reporting access violations provided by the host operating system can be used to report violations of such fine-grained access violations.

A secondary protection mechanism beyond access control involves services for confidentiality, integrity, and authenticity including end-point authentication. This cannot be used at the most basic protection layer since the necessary features are not always present. An example of such a situation is a modem attached to a serial port. In this case, the only control mechanism at the interface level is the verification of compliance with the security policy if a set of subjects and a set of objects wishes to exchange data with such a device, resulting in access being simply either granted or denied at this level. If a protected and identified higher-level protocol is used, however, this can override the basic protection.

The distinction between the device interface layer and higher layers, particularly the network protocol layer is becoming less clear with plug-and-play interfaces that imply operations initiated by devices or hosts other than the local node. In such cases mechanisms analog to those discussed in the following section are required.

D. Network Protocol Layer

Even though there are some application areas where other protocols are still used and which need to be dealt with separately, the Internet Protocol is by far the dominant protocol and the focus of our considerations.

The implementation of the Network Protocol Layer is highly performance sensitive and requires a tight integration of all components to reduce critical code path lengths. However, for the purposes of the description here we identify individual functional components as an in itself layered architecture.

All components described here are subject to the global organizational security policy as described in section III-A.

Since we assume that a node may operate in an exposed environment and that the network protocol stack of a node's host operating system may not be able to deal with malformed protocol data units (PDU) or sequences of PDUs, a first layer of protection from external nodes (regardless of whether they are part of a protected environment) is required that performs integrity checking on incoming and outgoing data and ensures that any and all PDUs passed on to the host network stack can be handled by it.

Another role that must be assumed by the Network Protocol Layer is basic packet filtering and circuit-level control. This applies to both incoming and outgoing traffic and must be coordinated with other components within its own layer as well as with other layers. As many policy rules as possible must be formulated in a way that they can be dealt with at the component level.

In cases where the security policy requires confidentiality, integrity, and authenticity at the circuit level, this must be assured in this layer as well. The obvious choice of technology for such requirements is IPSEC [10] since it is part of IPv6 and provides the necessary features already either by itself or lends itself to straightforward extension; it is also an already widely accepted IETF standard. This also permits the use of such features when communicating with nodes that are not equipped with the mechanisms described here.

Actual benefits as compared to other security mechanisms besides enforcing a consistent security policy across all nodes belonging to a central authority result from the integration of information derived from other areas of the system.

By identifying the subject performing an operation involving the network layer (which, depending on the host operating system may be either derived from information available in the local layer or from the system call layer) we can enforce security policy rules at a much higher level of abstraction. Similary, the identity of the process performing the operation can be traced back with the help of other subsystems and can be related to the identity of an application or system service that is performing the operation.

This permits the specification of policy rules that identify which users may use which application programs to communicate; in addition, the set of nodes with which communication may be established can be restricted based on various factors including temporal restrictions. This mechanism is more coarse-grained than that proposed by Ko et al. [7], but also restricts the policy specification to a more manageable size.

We can achieve further benefits by employing a small modification to IPSEC, namely positive identification and authentication of peer nodes. For this to work, each node must be identified by a certificate. The certificate must be issued by a certification authority that is contained in a security policy that needs to decide the permissibility of a connection to a peer. Such a mechanism interacts closely with the general packet and circuit filtering discussed above; in cases where the peer also has an ECRM, the certificate should contain the same (node) subject name for the sake of simplifying policy rules; if the IPSEC implementation is not embedded within the ECRM, a second key pair for the IPSEC mechanism is required since the latter may be compromised by external manipulation of the node.

This also permits the enforcement of policy rules requiring certified peer nodes; if a peer node is compromised, the node's certificate can be placed on a certificate revocation list, limiting at least the potential exposure of other nodes after the detection and blacklisting of the compro-

mised node.

One disadvantage in requiring positive peer identification and using other features of IPSEC is that it makes the integration of proxies, network translation devices, and other intermediate nodes that modify data streams hard or impossible. This is intentional, but may require considerable changes to existing network infrastructures.

A final sublayer in the Network Protocol Layer can then optionally perform operations typically reserved for such intermediate nodes such as transformations of data before reaching an application (e.g. decryption of data that was stored in encrypted form on an external server) or performing virus scanning. This applies to both incoming and outgoing data streams, but is most relevant for incoming streams. Other than the earlier sublayers, this sublayer is not necessarily fully transparent since it may induce noticeable changes in e.g. the traffic pattern observed by an application.

E. File System Layer

The role of a file system level protection mechanism in a network security context may not be entirely obvious; however, when seen from the perspective of enforcing an overall security policy which will deal with data objects and applications, it becomes a focal point of policy enforcement.

This layer has two main purposes. One is to provide security mechanisms at the file system level in its own right, the other is as an information source for policy decisions that involve other layers. Here we are mainly concerned with the second role; however, the first one is a necessary element in overall security policy enforcement and therefore must also be dealt with.

Security at the file system level is part of the basic security functionality provided by virtually all general-purpose operating systems. However, there are two deficiencies that need to be remedied. First is the lack of conformance to a consistent, centrally enforced security policy. Even in networked environments, such controls are honored only in largely homogeneous areas. Another issue is the need for protecting data once it has left the immediate control of a node's operating system. This can generally occur in two ways. One is by accessing the storage media locally with an operating system other than the one enforcing the security mechanisms (possibly just another instance of the same operating system, only configured differently), the other occurs when the storage media are exposed. While the latter threat has always existed in case of removable media or exposure of portable systems, this scenario is becoming more urgent as technologies such as network-attached storage and storage-area networks become more prevalent. The key problem in both cases is that the previous assumption that the storage medium and path to the node providing security enforcement is secure is no longer valid. What is therefore required is an encryption mechanism operating at the file system layer which transparently protects whatever storage medium or mechanism is employed. Transparent encryption at the file system layer also deals with the problem of accessing the storage medium from another operating system or an access method that does not honor the security policy.

As noted in section IV-D, the file system layer has access to other important information. This information correlates users and files they are using. We need to distinguish three types of files. The first type of file is the executable file as seen by the operating system. Such executables, which usually consist of several parts (a main file and a number of dynamically loaded shared objects or dynamically linked libraries), can be identified and matched against security policy rules containing approved applications. The second type of file is harder to identify when located at the file (operating) system layer and involves all scripting languages, i.e. mechanisms that involve files classified as non-executable by the operating system but executed by an intermediate application program. This class of applications includes macro languages found in many applications and has been the source of a large number of successful attacks. Here only heuristics and elaborate checks can attempt to identify and protect against malicious code. The third type of file consists of plain data objects. As outlined in [9] this requires a certain overhead in the form of a labeling mechanism that is transparent to the host operating system yet enforced across heterogeneous environments.

As indicated above, this information can be combined with other information collected at different layers. In particular, the integration of the file system layer permits the dynamic "sandboxing" of applications.

One example of such sandboxing in case of a MLS-like policy is the dynamic restriction of a process from making certain network connections once it has accessed a data object whose classification label does not match with the classification of a given network peer. The same mechanism obviously also is applicable to operations within the node local file system and can be used to implement a purely local MLS configuration. In most cases, however, caching and common networked file systems will require coordination of policy across node boundaries.

Finally, similar to the topmost sublayer described in section IV-D, it is also possible to embed additional transformations into the file system access mechanisms. One example of such a transform would be a virus scan (which can easily be extended to other types of undesirable files or contents; the main issues here are in determining what level of checking constitutes anacceptable performance impact, and that any such blacklisting is by necessity incomplete even when heuristics are used since we have to assume that a potential attacker is aware of the heuristics).

F. System Call Layer

Unlike in the design by Fraser et al. [11], the system call layer is merely an ancilliary layer instead of the focal point of embedding additional security.

Depending on the host operating system (mainly under Unix derivatives) it is necessary to insert code that triggers the other layers discussed above. In other instances it is desirable to block or modify the behavior of certain system

calls; but this should be a method of last resort since it violates the transparency sought after in the design of the entire mechanism and requires a level of complexity in the security policy specification that is difficult to administer. If and when they are employed as independent enforcement tools, they are subject to the same policy mechanisms discussed above and need to interact with other layers to obtain the necessary information for forming policy decision queries to the ECRM.

V. External Layering and Legacy Nodes

Besides layering abstractions, a secondary layering mechanism is required that may also be defined as a defense-in-depth posture.

Even though the mechanisms described here apply to most reasonably modern operating systems, there are some nodes that can not or may not be modified to include these mechanisms. These are typically legacy systems or single-purpose devices such as network printers.

Such nodes require the protection of a conventional firewall or a node equipped with the mechanisms described here acting as such. This should be an acceptable situation as both the limited number and protection requirements of such nodes permits protection by topologically oriented firewalls and a binary approach to filtering network traffic. In case such a node requires more elaborate granularity of control, an alternate approach to modifying the legacy node itself is to construct a wrapper consisting only of a node with the mechanisms described here acting as a gatekeeper to the legacy node. In this case, adequate physical protection of the thus generated composite must be assured.

Topological separation is by no means completely obsolete; wherever possible, a compartmentalization of a network should be performed along organizational lines as well as along the network topology. This provides protection against disasters in individual compartments as well as against resource exhaustion attacks targeted at the network infrastructure; obviously the latter protects only the operational characteristics of the internal node, external communication may be denied by such attacks.

One additional requirement that must be satisfied is that as a minimum communication from ECRM nodes to an ERM must be guaranteed in case policy updates or immediate policy decisions are required. To avoid denial of service attacks as much as possible, this necessitates a topological arrangement that ensures a route that is not subject to attacks is present for all ECRM-controlled nodes.

VI. Implementation

The segregation of policy enforcement at the various interface layers from decision processes on end nodes is discussed in [9]; the ERM/ECRM core itself is platform--independent. The focus of this discussion is on the enforcement mechanisms themselves.

The first challenge in implementing the enforcement modules lies in identifying suitable interfaces within the host operating systems that are present – if possible – in all operating system families, and that do not undergo significant changes between revisions of the various host operating systems.

Another requirement is that such interfaces must be modular. Source code to targeted host operating systems is not always available; in addition, embedding modifications at the source level implies additional complexity in specification and testing of the host operating system. Even if source code is available, this mechanism allows embedding of security features transparent to the host operating system and is therefore in itself desirable.

We have so far concentrated on the Unix family of operating systems, in particular on derivatives of Unix System V Release 4, as well as on the Microsoft Windows NT family which includes the Windows 2000 and XP releases.

Cursory examinations have shown that similar mechanisms can also be implemented on other Unix-derived families such as BSD Unix, Mach, and Linux as well as other operating system families such as OpenVMS.

A. Microsoft Windows NT

The kernel and device driver level in the Windows NT operating system family is based on an asynchronous processing model and, at least in this regard, has a passing semblance to OpenVMS [12], [13], [14].

The basic units for individual operations are I/O request packets (IRPs). They contain the necessary information for the processing of an operation (which need not be related to I/O) and are passed between the modules inside the kernel.

The kernel is structured in a way that an I/O request is not handled by a monolithic module but rather by a sequence of drivers that assume responsibility for certain aspects of a request. There are explicit mechanisms for the insertion of additional kernel modules, called filter drivers, into this stack which can change, augment, or replace existing behavior.

A.1 Device Interface Layer

In many cases Windows NT separates the handling of devices into several distinct drivers. At the lowest level there are bus drivers that handle bus (e.g. PCI, USB) operations; these are of limited interest. Layered on top of these are function drivers that deal with individual devices, typically accessing the device hardware in the process. It is on top of these function drivers that a filter driver can be placed since this permits abstraction from any specific hardware. Examples of checks performed at this level would be the blocking of adding new devices to an USB or IEEE1394 bus that is not part of the approved system configuration. Beyond access decisions the role of this layer is limited by the amount of semantic information available.

A.2 Network Protocol Layer

Windows NT implements a number of networking protocols; handling of these protocols is dispersed through several layers which must be instrumented to obtain all

necessary information. Most operations, including the insertion of IPSEC and firewalling, can occur at the transport driver interface (TDI) layer. This needs to be performed for all network protocols that have to be supported (e.g. IP, DECNet). However, not all information necessary for higher-level processing is present at the TDI transport level. For this it is necessary to insert another mechanism at the WinSock level, namely a WinSock layered service provider. Details on this mechanism are discussed in [15].

A.3 File System Layer

The file system under Windows NT is in itself layered; it is possible to interpose functionality at several different layers. We have chosen to emplace a file system filter driver immediately between the I/O manager and the various file systems. This has several benefits the price of increased complexity. One of these is the ability to intercept at the individual file level, performing object label manipulations and selective operations such as encryption, regardless of whether the file system is local or remote. Similarly, it permits the operation under any type of file system supported by Windows NT since the file system filter driver operates at the generic function level that must be supported by all file systems.

B. System Call Layer

Unlike other operating systems, Windows NT does not have a simple mechanism for replacing or changing parts of system call interfaces. In addition, there are typically several code paths to achieve the same goal due in part to NT's multiple OS personalities. As a result, the most general mechanism is to replace system dynamic link libraries with "shell" libraries that take the place of the original library, forward most calls unmodified to the original library, and perform necessary manipulations and callouts both down- and upstream. This mechanism is general in nature, but implies a level of complexity and version tracking that makes it impractical from a pragmatic viewpoint.

C. Unix System V Release 4 Derivatives

Unix has been the subject of significant research into security mechanisms due to its extensible design and source availability. We have concentrated on implementing the mechanisms described here first in System V Release 4 derivaties due to their maturity and commercial significance. These systems, like all modern Unices, also have a unified virtual memory and caching architecture, but unlike Windows NT are largely synchronous in terms of the way kernel operations are performed, reducing complexity in the process.

C.1 Device Interface Layer

Where access control mechanisms provided by the host operating system is insufficient, the mechanism to control the device interface layer is quite similar to the wrapper library discussed in section VI-B. In this case the functionality of the wrapper limited and well-defined and can be re-used for a number of devices since Unix device drivers fall into only a limited number of categories. The only instance where device-specific mechanisms are required is in handling IOCTLs.

C.2 Network Protocol Layer

The network subsystem of System V Release 4 Unix derivatives is highly structured; data traffic is ultimatlely passed through the asynchronous STREAMS stack even though it may have originated in other API layers. One can intercept or replace data at multiple sublayers from the link layer upward. Depending on the availability of a native IPSEC implementation, either augmentation or wrapping of the existing stack is called for; wrapping is the more general mechanism but implies a considerable performance impact. The STREAMS layer does, however, provide the necessary data structures to easily identify both the process, the executable data performing the operation, and the user, permitting to localize several policy decisions.

C.3 File System Layer

Most modern Unix derivatives use the modular virtual file system (VFS) architecture. This mechanism can be used to easily add new file systems without changing the kernel and core data structures itself; in addition, the Vnode abstraction has replaced the traditional Inode of older Unix versions. This architecture also permits the insertion of modular code that changes the behavior of existing file systems even though this use was not anticipated in the original design [16]. Another feature of the VFS/Vnode architecture permits maintenance of bookkeeping data within private data structures associated with each open file, greatly simplifying the required overhead operations.

D. System Call Layer

Since Unix traditionally provides a single system call mechanism that can be amended either by directly modifying the dispatch table or adding new system calls – the latter even dynamically in some newer systems – this has been a tempting area to add security functionality [17], [11]. While it is in fact possible to mimic some functionality such as file system access and use control or encryption at this level, this does not address issues such as the unified virtual memory architecture. Modifications at this level are therefore kept at a minimum.

VII. Related Work

The approach of modifying a COTS host operating system by inserting kernel modules has also been pursued with the SLIC system [17]; the focus there is on modifying the system call behavior of Unix derivatives.

The Generic Software Wrappers mechanism [11] similarly focuses on such modifications to system call behavior, although with application security as the main focus. [11] also defines a language for defining wrapper behavior. Wrapping individual applications regarding their network behavior is an approach that was proposed in [18].

Domain and Type Enforcement [19] represents another, more intrusive modification of an operating system for em-

bedding security mechanisms; the main drawback here is that the concept requires explicit labeling and cooperation in a networked environment; the DTE firewall concept is an all-or-nothing approach.

The latter aspect is dealt with in an extension to the Flask/Fluke [20] architecture [21]. This mechanism also permits the use of a fine-grained access control policy at the level of individual calls but requires the infrastructure from the Flask/Fluke environment and is therefore difficult to adapt to a COTS environment.

The approach of clearly segregating policy and enforcement at the level of a host operating system was articulated in the Synergy [8] project and its precursors [22] and can be found in most of the related work discussed above; the ERM/ECRM mechanism simply moves the already separated parts into different nodes on a network with distinct roles and permits a consistent enforcement of a given security policy.

VIII. Outlook and Future Work

Our group has been working on security enhancements to COTS operating systems since early 1998, at first focused mainly on enhancements to file system mechanisms. Both technological and threat developments have shown that the boundary between a firewalling mechanism and a more general host and network security system could no longer be maintained. Current research directed towards implementing the full breadth of mechanisms described in this paper and on identifying additional layering mechanisms where adding security mechanisms using higher-level semantics are beneficial. One example of such a mechanism would be control over printing operations which can, besides merely granting permission to submit print jobs, include the embedding of both visible (e.g. classification labels independent of applications) and invisible markings (digital watermarks [23]) into the print output.

Another major future research area is the reduction and analysis of the audit data gathered throughout the internetworked system and the interrelation of access and use patterns to relevant user or code behavior.

References

[1] E. J. Sebes, "Overview of the architecture of Distributed Trusted Mach," in *Proceedings of the USENIX Mach Symposium: November 20–22, 1991, Monterey, California, USA*, USENIX, Ed., Berkeley, CA, USA, 1991, pp. 251–262, USENIX Association.

[2] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman, *Building Internet Firewalls*, O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2nd edition, 2000.

[3] Steven M. Bellovin, "distributed firewalls," *;login: the USENIX Association newsletter*, vol. 19, no. Special Issue on Security, pp. 39–47, Nov. 1999.

[4] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1 ," Tech. Rep., W3C, may 2000, Status: W3C Note.

[5] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," Tech. Rep., James P Anderson Co., Fort Washington, PA, Apr. 1980.

[6] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey, "A Real-Time Intrusion Detection Expert System (IDES) – Final Technical Report," Tech. Rep., SRI Computer Science Laboratory, SRI International, Menlo Park, CA, Feb. 1992.

[7] C. Ko, M. Ruschitzka, and K. Levitt, "Execution Monitoring of Security-Critical Programs in a Distributed Systems: A Specification-Based Approach," in *Proceedings of the 1997 Conference on Security and Privacy (S & P '97)*, Los Alamitos, May 4–7 1997, pp. 175–187, IEEE Press.

[8] O. S. Saydjari, S. J. Turner, D. E. Peele, J. F. Farrell, P. A. Loscocco, W. Kutz, and G. L. Bock, "Synergy: A distributed, microkernel-based security architecture," Tech. Rep. version 1.0, National Security Agency, Ft. George G. Meade, MD, Nov. 1993.

[9] Stephen Wolthusen, "Enforcing Security Policies using Externally Controlled Reference Monitors," Submitted for publication.

[10] R. Thayer, N. Doraswamy, and R. Glenn, "RFC 2411: IP security document roadmap," Nov. 1998, Status: INFORMATIONAL.

[11] Timothy Fraser, Lee Badger, and Mark Feldman, "Hardening COTS Software with Generic Software Wrappers," in *Proceedings of the 1999 Conference on Security and Privacy (S & P '99)*, Los Alamitos, CA, May 9–12 1999, pp. 2–16, IEEE Press.

[12] David Solomon, *Inside Windows NT*, Microsoft Press, Bellevue, WA, USA, 2nd edition, 1998.

[13] David Solomon and Mark Russinovich, *Inside Windows 2000*, Microsoft Press, Bellevue, WA, USA, 3rd edition, 2000.

[14] Ruth Goldenberg and Saro Saravanan, *Open VMS AXP Internals and Data Structures: Version 1.5*, Digital Press, Maynard, MA, USA, 1994.

[15] Ero Rademer and Stephen Wolthusen, "Transparent Access To Encrypted Data Using Operating System Network Stack Extensions," in *Proceedings of the Communications and Multimedia Security Conference 2001.* Joint working conference IFIP TC6 and TC11, May 2001, p. ?, Kluwer Academic Publishers, To appear.

[16] Erez Zadok, Ion Badulescu, and Alex Shender, "Extending File Systems Using Stackable Templates," in *Proceedings of the USENIX 1999 Annual Technical Conference*, Berkeley, CA, June 6–11 1999, pp. 57–70, USENIX Association.

[17] Douglas P. Ghormley, David Petrou, Steven H. Rodrigues, and Thomas E. Anderson, "SLIC: An Extensibility System for Commodity Operating Systems," in *Proceedings of the USENIX 1998 Annual Technical Conference*, Berkeley, USA, June 15–19 1998, pp. 39–52, USENIX Association.

[18] Wietse Venema, "TCP WRAPPER: Network Monitoring, Access Control and Booby Traps," in *UNIX Security III Symposium, Baltimore, MD*, Berkeley, USA, Sept. 14–17 1992, pp. 85–92, USENIX Association.

[19] D. L. Sherman, D. F. Sterne, L. Badger, S. L. Murphy, K. M. Walker, and S. A. Haghaighat, "Controlling network communication with domain and type enforcement," in *Proceedings of the 18th NIST-NCSC National Information Systems Security Conference*, 1995, pp. 211–220.

[20] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, and Jay Lepreau, "The Flask security architecture: System support for diverse security policies," in *8th USENIX Security Symposium*, Washington, D.C., USA, Aug. 1999, USENIX, pp. 123–139.

[21] Ajaya Chitturi, "Implementing Mandatory Network Security in a Policy-flexible System," M.S. thesis, Department of Computer Science, University of Utah, Salt Lake City, June 1998.

[22] O. S. Saydjari, J. M. Beckman, and J. R. Leaman, "Locking Computers Securely," in *Proc. 10th NIST-NCSC National Computer Security Conference*, 1987, pp. 129–141.

[23] Christoph Busch, Wolfgang Funk, and Stephen Wolthusen, "Digital Watermarking: From Concepts to Real-Time Video Applications," *IEEE Comput. Graph. Appl.*, vol. 19, no. 1, pp. 25–35, Jan./Feb. 1999.