Access and Use Control using Externally Controlled Reference Monitors

Stephen D. Wolthusen
Security Technology Department, Fraunhofer-IGD, Darmstadt, Germany
wolt@igd.fhg.de

13th November 2001

1 Introduction

The reference monitor as a structuring mechanism for operating system design were proposed by Anderson [1] based on earlier work by Schell. It has since been used as a guiding principle for the design of secure operating systems or in adding security facilities to existing systems, arguably due to it being a simple yet powerful abstraction, but undoubtedly also with the aid of [17] which mandated the use of the reference monitor concept for systems conforming to the classes B2 and higher.
The mechanism described in [1] assumed that the processing of information would generally occur within the confines of a monolithic computer system that facilitated centralized control. Similarly, [17] provides only minimal considerations for the handling of information exchanged between multiple instances and then assumes that the sensitivity labeling of all instances obeys a single scheme.
This assumption cannot be maintained in current computing environments. Not only are a considerable number of computer systems internetworked even within a single area of control of an organization, a single computing device itself can and must be understood as a network of components whose components and interconnections may be exposed to both active and passive attacks.
Given the security model and policy of an organization or a set of organizations, it has therefore become necessary to ensure the consistent enforcement of these policies occurs on all components of the computing environment to which they apply. As has been argued before, this enforcement must encompass all resources on a given system and thus occur at the level of non-bypassable operating system mechanisms [8].
We propose a natural extension of the proven reference monitor abstraction that fulfills this requirement for distributed environments by externalizing the actual policy definition and distribution into a redundant network of external reference monitors (ERM) and providing only enforcement mechanisms at the level of the actual component systems, controlled by an externally controlled reference monitor (ECRM) subordinate to ERMs. In addition, the mechanism proposed here can serve to provide not only access control but also to exert behavioral restrictions and to provide distributed firewalling and intrusion detection facilities.
This extension could be part of a newly designed system; however, the primary focus here is on the capability for retrofitting the mechanism into existing operating systems. This implies that the mechanism must be implementable even for systems where there exists no such local facility.

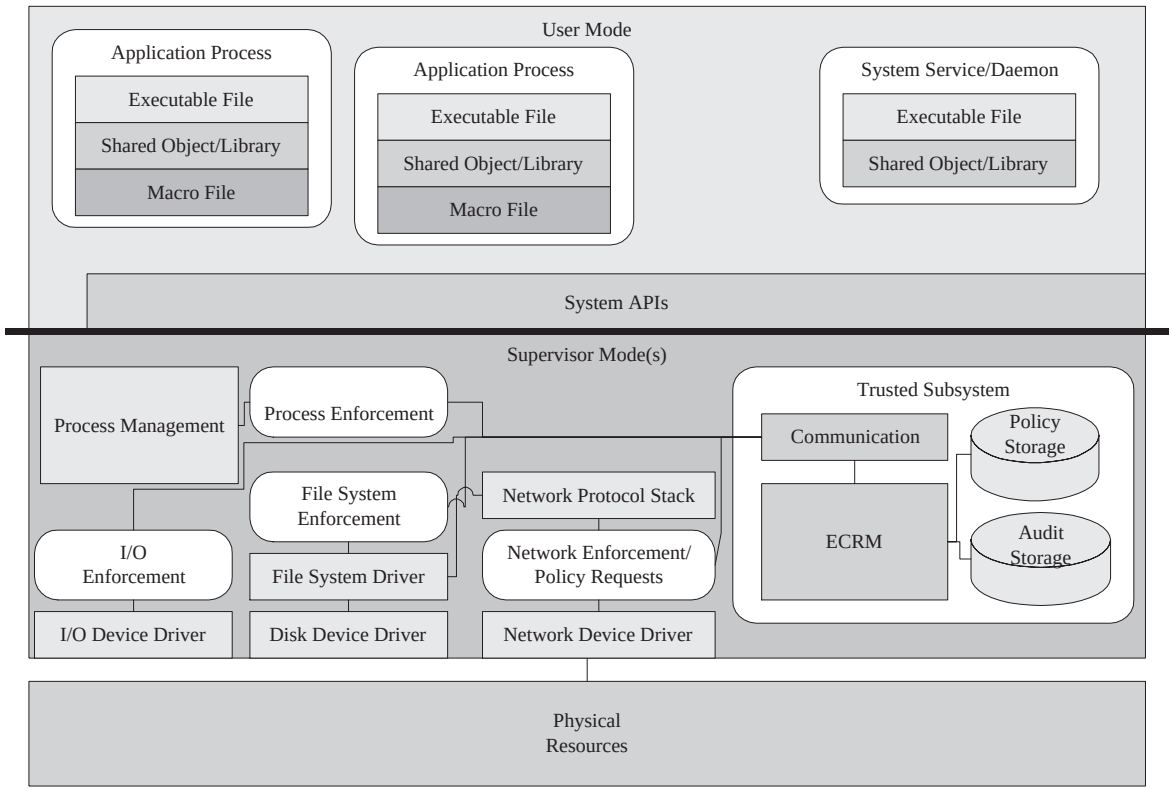

Figure 1: Components in a Controlled System

2 Architecture of Controlled Systems

The following discussion does not assume a pre-existing reference monitor system in the operating system to be equipped with an externally controlled reference monitor. In systems where this is the case the necessary alterations and simplifications to the description below should be obvious.

By definition, the reference monitor must be invoked on any access to resources controlled by the operating system. Examples for such resources are devices, files, memory segments, virtual circuits, or datagrams. We also assume that any access to resources not occurring in supervisor mode does not have direct access to the resources and use a well-defined set of entry points to the supervisor mode environment.

This implies a requirement to mediate each access to the relevant components; figure 1 shows the process management subsystem, generic I/O subsystem, the file system, and network subsystems as an example. This is accomplished by interposing a layer between the next-higher layer in the system hierarchy and the controlled component itself. These layers are referred to as “enforcement modules” in the following discussion.

The enforcement modules serve several purposes. First, any operation involving the controlled component is subject to mediation by the enforcement module which in turn relegates the access request to the ECRM. This extends to any operation under the control of enforcement modules even if the base operating system enforces security constraints only on first access. The ECRM can permit the operation to proceed as requested, or it can instruct the enforcement module to perform additional

operations as preconditions. An example for such a precondition is the use of cryptographic confidentiality and integrity protection. The second main purpose of the enforcement module is to gather information as instructed by the ECRM that is either a prerequisite for reaching a decision by the ECRM (i.e. by ERMs) for an operation request (which may be the same enforcement module as the one issuing the request) or as a result of an active ECRM query into the surrounding system based on an internal requirement of the ECRM.
The ECRM operates as an agent of one or more ERM dictating the security policies to be applied. It is contained in a – at least logically – separate trusted subsystem area whose components can be subjected to a rigorous design, verification, and validation. It also translates to and from a common format for characterizing operations in a suitable format for submission to ERMs over a communication channel providing integrity, confidentiality, and mutual authentication between the ECRM and any ERM contacted.
Assuming that the security policies for the subjects, objects, and operations involved permit this, the ECRM can act based on policy information that has been temporarily delegated by one or more ERM, mainly for reasons of performance and scalability.

3 Architecture of External Reference Monitors

The ERM itself is again a trusted subsystem and may also be combined with the mechanisms found in a controlled host as described above; however, it is generally sufficient to provide an execution environment for the trusted subsystem and a communication channel which can be controlled by this subsystem.
An ERM receives policy requests from ECRMs over a protected communication channel. These request (or hypotheses) are formulated as a 4-tuple $(\Sigma, \Omega, \xi, \lambda)$ where Σ is a vector of subjects σ_i and Ω is a vector of subjects ω_i (see section 5.1 for additional properties), ξ is an operation descriptor $\xi \in \mathbf{\Xi}$ where $\mathbf{\Xi}$ is the set of well formed operation descriptors, and λ is an operation lifetime $\lambda \in \mathbf{\Lambda}$. For all ξ there exists a predicate $\mathrm{operation}_{\xi}(\sigma_1, \ldots, \sigma_i, \omega_1, \ldots, \omega_j)$ where i and j depend on ξ. The set of valid operation lifetimes $\mathbf{\Lambda}$ is defined as ordered pairs (λ^s, λ^e) where λ^s and λ^e are elements of a set on which an order is defined for which $\lambda^s \leq \lambda^e$ holds.
The ERM applies the security policy or security policies which are stored under the control of the ERM according to the identities of the subjects and objects involved and may yield one or more replies; this is denoted with the symbol ∇, i.e. $\{(\Sigma^\nabla, \Omega^\nabla, \xi^\nabla, \lambda^\nabla)_k\} = \nabla(\Sigma, \Omega, \xi, \lambda)$. A negative reply tuple is denoted as $(\Sigma^\nabla, \Omega^\nabla, \xi^\perp, \lambda^\nabla)$. The application of any policy decision on a negative reply tuple is again the negative reply tuple: $\forall \nabla : \nabla(\Sigma^\nabla, \Omega^\nabla, \xi^\perp, \lambda^\nabla) = (\Sigma^\nabla, \Omega^\nabla, \xi^\perp, \lambda^\nabla)$. In addition, the ERM may generate one or more audit events for each request received.
In addition, the ERM may also issue commands to ECRMs; the syntactical constructs involved are the same as before although different or additional elements compared to those used for handling ECRM requests may be used. Such commands may include operations performed within the ECRM itself, but may also include commands which specify operations which affect the system in which the ECRM is embedded (e.g. termination of network connections, or specific surveillance of certain operations).
To ensure that the semantics of requests and replies are equivalent even if they are issued based on different platforms it is necessary to use a common set of primitives for the description of subjects, objects, and the operations to be performed. To ensure consistent application of the policy to subjects and objects regardless of which node of a distributed system they reside on or are otherwise connected to, it is also necessary to identify subjects and objects in a way that transcends individual nodes and

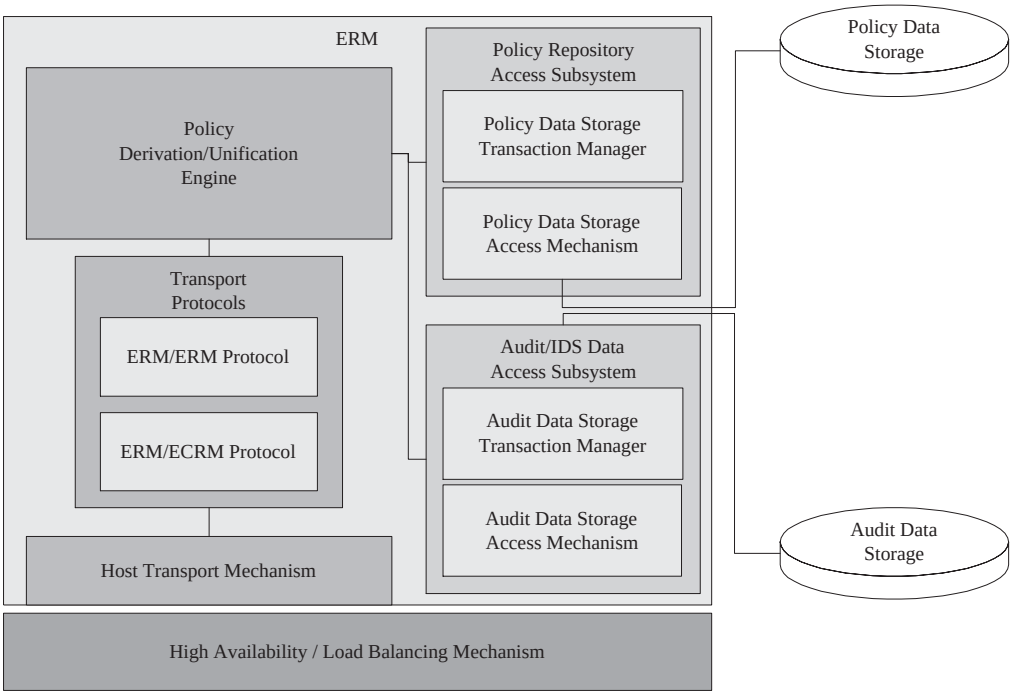

Figure 2: ERM Architecture

platforms. The ERM mechanism itself is independent of a specific security policy or model. The request mechanism ensures that all required information for e.g. access matrix, capability, or domain and type enforcement systems are transmitted from the ECRM to the ERM in referring a decision on an operation.
The overall architecture of an ERM is shown in figure 2 on page 4; both the storage and retrieval of polciy information and of audit information (also of relevance for intrusion detection) is externalized outside the trusted subsystem since it can be cryptographically protected.

4 Security Policy Mechanism

This section gives a brief outline of a security policy mechanism that can be implemented using the ECRM mechanism. While not specifically tied to it, it can serve as an illustration and also incorporates mechanisms necessary to provide interoperability between platforms; details can be found in [20].
The basis for all operations discussed is the use of formal logic, specifically of first order predicate logic. Subjects and objects in operating systems as well as other objects and meta-objects can be described as constants of a first order language. Subjects can also be grouped into categories using predicates such as $\mathrm{user}(S)$[1], $\mathrm{process}(S)$, $\mathrm{application}(S)$, $\mathrm{host}(S)$, $\mathrm{network}(S)$, $\mathrm{link\_node}(S)$, or $\mathrm{bus\_node}(S)$ etc.; objects are similarly grouped by predicates including but not limited to $\mathrm{file}(O)$,

[1]The use of the serif typeface (e.g. $\mathrm{user}(S)$) indicates meta-variables, sans serif typeface (e.g. $\mathsf{user}(S)$) indicates concrete instances (i.e. belonging to a specific interpretation), and fixed font usage (e.g. `Process`) indicates symbols belonging to a specific implementation

$\mathrm{data\_file}(O)$, $\mathrm{executable\_file}(O)$, $\mathrm{virtual\_circuit}(O)$, $\mathrm{memory\_segment}(O)$, $\mathrm{datagram}(O)$, $\mathrm{physical\_link}(O)$, and $\mathrm{connection}(O)$.
Individual subjects and objects are identified by means of constants which uniquely represent an entity throughout a distributed system.
Predicates such as $\mathsf{write\_device\_control}(S, O)$ and $\mathsf{receive\_ipc}(S_1, S_2, O)$ indicating permitted operations are also defined; these represent a common set of abstractions found in operating systems relevant to control over subject behavior.
For each of these predicates, a definition of the semantics is given and for each of the systems on which the mechanisms discussed here are to apply, an interpretation of the predicates must be given. Similarly, bijective functions must be defined to map the constants against interpretation- and ultimately implementation-specific entities.
As an example, the definition of the predicate $\mathrm{write\_device\_control}(S, O)$ is:

> An interpretation should map this primitive to an operation which transfers information under the control of the entity identified by the metavariable S for which the predicate $\mathrm{subject}(S)$ holds to the resource identified by the metavariable O for which the predicate $\mathrm{device}(O)$ holds and which is mutually disjoint with information for which the operation identified by the predicate $\mathrm{write\_device\_data}(S, O)$ holds.

The corresponding definition for the Microsoft Windows 2000 operating system of the interpretation $\mathsf{write\_device\_control}(S, O)$ (which occurs at the level of the partially undocumented executive native application programming interface [14, 5, 11, 10] to ensure that all calls to the executive are mapped properly) is:

> This primitive is mapped to four native functions. The first function is `ZwSetEaFile` where the object O is identified as represented by the `File` executive object handle passed in to the `FileHandle` parameter which must refer to a `Device` executive object. The second function is `ZwSetInformationFile` where the object O is identified as represented by the `File` executive object handle passed in to the `FileHandle` parameter which must refer to a `Device` executive object. The third function is `ZwDeviceIoControlFile` where the identity of O is provided by the handle to the `File` executive object in the parameter `FileHandle` and must represent a `Device` executive object. The parameter `IoControlCode` must contain the flags `FILE_WRITE_ACCESS` or `FILE_ANY_ACCESS`. The fourth function is `ZwFsControlFile` where the identity of O is provided by the handle to the `File` executive object in the parameter `FileHandle` and must represent a `Device` executive object. The parameter `IoControlCode` must contain the flags `FILE_WRITE_ACCESS` or `FILE_ANY_ACCESS`.

The set of predicates, function symbols, and possibly axioms together form a security policy. The theory thus defined must be consistent; intuitively this means that only positive statments are permitted.
A policy request is formulated as a hypothesis in the formal logic and submitted to one or more ERM. An ERM may then attempt to prove that, based on its policy, the operation is granted. This can take several forms; in the simplest case a term substitution (which may be identity) can be applied. Even then, the use of lattices and returning the least upper bound (lub) of elements of the hypothesis and the policy element can yield a reply that is significantly more powerful than the hypothesis and which can then be re-used during the operation lifetime by the ECRM without the need for additional

queries. The reply may contain several elements, in that case the ECRM can consult (analogous to the mechanism described here for the ERM) these replies.
The expressiveness of the logic-based enforcement mechanism becomes relevant if a theorem prover is applied. This approach permits the use of several abstraction layers with specifications occurring at the highest feasible abstraction layer in the system and the policy logic deriving behavioral rules for subordinate abstraction layers. Additional details on this mechanism can be found in [21].
The structure of the hypotheses permit submission to several ERMs, possibly working on the same policy rule set. If any one of the ERMs return a positive reply, the ECRM can proceed. This permits the implementation of concurrent resolution with a simple term substitution mechanism and more elaborate provers. If after a certain resource limit has been reached the theorem prover has not arrived at a unification, a negative reply is returned; this does not violate security properties.
The application of multiple policies (typically from several ERMs) is denoted as $\{(\Sigma, \Omega, \xi, \lambda)'_k\} = \nabla_1 \circ \nabla_2 \circ \cdots \circ \nabla_n(\Sigma, \Omega, \xi, \lambda)$; the set of hypotheses in each application contains the original hypothesis. As each policy application yields additional hypotheses, the result of the chain of applications depends on the sequence of policy application at least in the additional hypotheses generated. The consultation of ERMs by an ECRM must therefore follow a fixed but arbitrary sequence; here we use the hierarchy defined by the ERM topology (see section 5).
The restriction to positive statements is, while found in other mechanisms as well, an impediment to natural formulation of rules. However, permitting specifications similar to natural language with rules and exceptions requires the use of non-monotonic logic. Permitting this would incur significant performance penalties and would also require a strict order on policies that must be processed in sequence without exceptions for each case; therefore we believe that the mechanism described here is an adequate compromise.
In addition to controlling the complexity of policies which must be defined by human security administrators, the logic-based mechanism fulfills another goal in that it permits specification of platform- and system-independent security policies. This is achieved since the formal logic requires the provision of a platform-dependent interpretation representing a model of the theory; by imposing a soundness constraint on the formal theory used we can guarantee that the policy will be enforced consistently.
Due to the flexibility and economy of mechanism the same general framework can also be used (substituting or adding predicates, constants, and functions as appropriate) for additional tasks such as specifying intrusion detection and data fusion mechanisms. Based on the properties of any first-order theory it is obvious that it can be used to represent any security model which can be expressed in an automated procedure.
One example of an augmented policy is the requirement for the mandatory encryption of all file objects meeting certain criteria on leaving the domain of control of an ECRM system (e.g. upon writing to storage media regardless of the mode of attachment).

5 External Reference Monitor Network Topology

As noted above it is necessary to impose an ordering or hierarchy on ERMs to ensure that ECRMs can obtain idempotent results.
One possible ordering is to use the same directed acyclical graph obtained by the Domain Name System (DNS) mechanism for name resolution. Since the system described here is intended for use in distributed environments and there is generally a correspondence between organizational specializa-

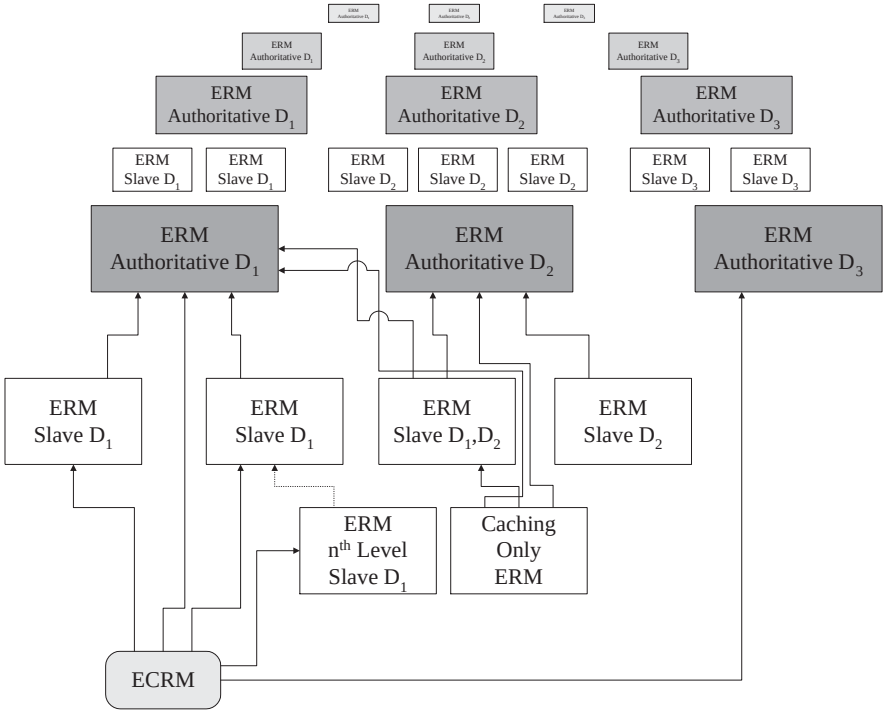

Figure 3: ERM Hierarchy

tion relations and the paths obtained by DNS names, this represents a workable solution which has the additional benefit of permitting implicit routing information for ECRMs wishing to contact an ERM, i.e. there is no need for forwarding requests since the ECRM implicitly knows the destination address for each request.

Also in analogy to DNS, ERM nodes can harbor and cache several policies; provided that sufficient trust and assurance exists this can be used to derive responses to requests that would otherwise have to be posed to other ERMs. Even if the trust required for a caching is not present, it is still possible to cache responses themselves by archiving the cryptographically protected tuples consisting of a hypothesis and the replies within the lifetime of the reply. The layout with several possible request paths is shown in figure 3 on page 7.

5.1 Type and Identity for Subjects and Objects

As described above, a subject is a constant denoted as σ. For each subject σ^i, there exist suitable constants $C^T_{\sigma^i}$, $C^I_{\sigma^i}$ and two predicates $\mathrm{subject\_type}(\sigma^i, C^T_{\sigma^i})$ and $\mathrm{subject\_identity}(\sigma^i, C^I_{\sigma^i})$ and such that an individual subject is uniquely identified by $\mathrm{subject\_type}(\sigma^i, C^T_{\sigma^i}) \wedge \mathrm{subject\_identity}(\sigma^i, C^I_{\sigma^i})$. Similarly, an object is a constant denoted as ω. For each object ω^i, there exist suitable constants $C^T_{\omega^i}$, $C^I_{\omega^i}$ and two predicates $\mathrm{object\_type}(\omega^i, C^T_{\omega^i})$ and $\mathrm{object\_identity}(\omega^i, C^I_{\omega^i})$ such that an individual object is uniquely identified by $\mathrm{object\_type}(\omega^i, C^T_{\omega^i}) \wedge \mathrm{object\_identity}(\omega^i, C^I_{\omega^i})$. The set of all objects is denoted as $\mathbf{\Omega}$, that of all subjects $\mathbf{\Sigma}$.

Each subject σ is embedded in a lattice $\mathcal{L}^I_\sigma$ imposed by $\mathrm{subject\_subset\_identity}$ on $\mathbf{\Sigma}$. For all σ^i there is a constant C and a predicate $\mathrm{subject\_subset\_identity}(\sigma^i, C)$ such that C is an element of

$\mathcal{L}^I_\sigma$ which contains σ^i. The function symbol $\mathrm{Subject\_subset\_identity}(A, B)$ yields the lub of the elements of $\mathcal{L}^I_\sigma$. Similarly, each object ω is embedded in a lattice $\mathcal{L}^T_\omega$ imposed by $\mathrm{object\_subset\_type}$ on $\mathbf{\Omega}$. For all ω^i there is a constant C and a predicate $\mathrm{object\_subset\_type}(\omega^i, C)$ such that C is an element of $\mathcal{L}^T_\omega$ which contains ω^i. The function symbol $\mathrm{Object\_subset\_type}(A, B)$ yields the lub of the elements of $\mathcal{L}^T_\omega$. In both cases the lattice is conveniently constructed by the power set $\mathcal{P}(S)$ and $\mathcal{P}(O)$, ordered by set inclusion.
This construction permits both the unique identification of subjects and objects across node and platform boundaries (assuming the corresponding mappings for the interpretations) and the specification of security policy rules on hierarchies of objects that lend themselves well to simple and efficient term substitution. Policy rules can affect individual objects or sets of objects identified by the respective lub of the set.
When aligned with the ERM topology and with the implicit definition of a dominance relation within the directed acyclical graph thus generated, this also permits the authentication of policy replies by following a chain of authentication elements (e.g. public key certificates).

6 Implementation Issues

The following section gives a brief overview of the issues involved in implementing an external reference monitor system; some of the design decisions are, as noted above, arbitrary and do not constitute an integral part of the ERM/ECRM mechanism.

6.1 External Reference Monitor

The ERM consists of a trusted subsystem which communicates with the outside world in the form of IPSec-authenticated messages. These messages are transmitted as UDP datagrams to ensure that – once a key exchange has occurred within a given timeframe – no extraneous transmissions increase the latency incurred by a request. UDP limits the maximum message size; however, this is not an issue since messages are considerably smaller than that. In addition, UDP is a best effort protocol. This is also no hindrance since the policy requests are idempotent and a request may either be repeated after a certain time has expired or be sent to several servers known to hold the required policy information. For this purpose the host IPSec implementation could be used; however, since it is desirable to exert a higher level of control over the permissibility of accepting packets (e.g. to avoid denial of service situations) and to have direct control over key management and exchange, a custom implementation is called for. On a System V Release 4 system, this is accomplished by using a STREAMS module inserted into the stream for the desired network interface; on a Microsoft Windows NT system this can occur by replacing all NDIS drivers with wrapper drivers that interpose themselves between the proper NDIS driver and the NDIS library[2].
The trusted subsystem proper (which is mainly concerned with deriving policy decisions, issuing commands to ECRMs, and performing audit data requests and replies) can be embedded into a cryptographic coprocessor and communicate with the host environment (including the ECRMs and other ERMs by extension) only using cryptographically protected messages. The integrity and confidentiality of hypotheses and replies are protected by a separate cryptographic protocol beyond the scope of

[2]The use of NDIS intermediate drivers would be desirable but is not feasible since the NDIS architecture permits bypassing of NDIS intermediate drivers. In any case, we do not anticipate the implementation of ERM on the Windows NT platform

this discussion; similarly, the ERM trusted subsystem must store and retrieve audit information and policy data from storage subsystems; these must also be protected by cryptographic means even if cryptographic coprocessors are employed since both can be of arbitrary size. Use of cryptographic means reduces the threat to policy and audit data to truncation unless the trusted subsystem is compromised.

6.2 Controlled Systems

The bulk of the implementation effort is required for ECRM embedding. This is particularly true in the case of the Windows NT (2000) operating system platform since the relevant interfaces are not all documented and source code is not generally available for review[3]. The following discussion concentrates on some aspects the Windows NT platform since comparably little literature exists in this area. It should be noted that all augmentations outlined here can be performed without access to source code by use of various device drivers and replacement of kernel dynamic libraries.

While Windows NT uses a reference monitor concept, this mechanism is applied only on first access; subsequent access is not verified. To achieve the possibility for controlling any operation on resources, interception points must be inserted at various points. The most prominent such location is the Object Manager. Intercepting any access to this component yields important information for later collation with outer information from different components.

Additional interception points are required within the network layer; as in the case of the ERM noted above this requires the insertion of a NDIS wrapper into the network stack. This location on one hand ensures that only well-formed data meeting the requirements of all active security policies and being within the limits imposed by the robustness of the host network stack are forwarded to the remainder of the operating system and vice versa and on the other hand permits the issuing and receiving of hypotheses and requests from ERMs without recursion and deadlock within a host operating system network stack.

An essential part is in the interception of the file system. The Windows NT operating system family permits the insertion of both high-level file system filter drivers as well as interception points at lower levels in the storage hierarchy[10]. High-level interception is of particular interest since it permits the embedding of additional semantics such as labeling for the identification of objects and transparent encryption and decryption for all file systems supported including both local and remote file systems. In conjunction with the object manager and controls over processes and the network interface this also permits constructs such as predicates permitting applications (as identified by a collection of executable resources from the file systems and a process) network access depending on the history of resources accessed previously by such a process. Details on this mechanism can be found in [22].

A final example for an interception point is the control over the printing subsystem. Some security policies may impose restrictions on hard copy output or the emplacement of sensitivity labels or the identity of the subject (user or system process) initiating a print job. By replacing a choke point component of the printing subsystem of the Windows NT family it is possible to accomplish this irrespective of the type of printer or printer driver used. Details on this mechanism can be found in [23].

As described in [21], this mechanism can be extended arbitrarily and can be layered in terms of abstraction layers covered by the interception points; the mechanisms discussed here are not sufficient to achieve full coverage.

[3] While source code can be obtained, intellectual property concerns dictated the use of other means

At least some of the mechanisms (e.g. policy derivation, audit trail generation, intrusion detection analysis, and auxiliary functions such as object en-/decryption) can be embedded in a cryptographic coprocessor. The interface to this coprocessor must as narrow as possible and permit well-defined communication exclusively under the control of the trusted subsystem. This is not always the case as was recently demonstrated [3]. If, however, the environmental considerations are also taken into account, the use of cryptographic coprocessors can significantly enhance both robustness against tampering and performance of the system provided that the considerable cost incurred by the use of such devices can be justified.

7 Related Work

The ECRM concept can be seen as a logical next step beyond the Network Reference Monitor introduced by Anderson [2].
Early work on segregating policy decisions from enforcement was performed at UCLA [18] and also in the LOCK project [12, 13]. The DTOS project also dealt with this concept [9] based on a Mach microkernel architecture.
The generally promising approach to the issue of improving assurance by means of a microkernel and enforcing security at critical component junctions is pursued by work on the Flask security architecture [15] based on the Fluke operating system architecture [6] and earlier work on DTOS; some of the more recent work of that group is focused on bringing the concepts of Fluke to the Linux kernel. The basic concept of externalizing policy temporarily decisions can also found there [4], although policy delegation and performing caching in trusted subsystems has apparently not been pursued, although the problems were recognized [9]. Similarly, enforcement of security in this system requires a homogeneous environment; using encryption as an enforcement tool addresses this issue to some extent. DTOS is also of interest in that it was designed with policy flexibility, although policy composition was not considered.
Several applications for secure coprocessors have been part of the LOCK project and also proposed by Tygar and Yee [16] as well as White [19] including use of encryption of critical features for enforcing software rental agreements; this can be considered similar to the access and use control by ECRM of described proposed here.
In contrast to these mechanisms as well as to [7], the mechanisms outlined above permit the specification of behavioral controls if so desired without requiring the detailed level of specification.

8 Conclusion

This paper has presented a mechanism for the specification and enforcement of an arbitrary set of security policies for access, use, and behavioral control on an extensible set of subjects, objects, and operations. The use of formal logic for the derivation of policy statements from either hypotheses posed by externally controlled reference monitors in response to an attempted operation or previously issued commands or from other policy statements permits a layering of abstraction models that alleviates the complexity problems faced in the specification of security policies.
By being independent of a specific operating system — and specifically, by being able to provide security as an add-on mechanism to widespread COTS systems such as Microsoft Windows 2000 and System V Release 4 Unix derivatives such as Sun Solaris without affecting application programs – and mapping the semantics onto system-specific constructs, a broad coverage essential for use in

distributed systems is achieved. The simultaneous application of multiple policies for subjects and objects involved also permits the modeling of complex interrelationships between entities of different organizations. The use of of cryptographic coprocessors for both controlling (ERM) and controlled (ECRM) instances particularly permits the distribution of caching ERM instances without a trust relationship being necessarily involved, providing the foundation for a high degree of scalability.
Results obtained so far indicate that the overhead required for ECRM/ERM communication are barely noticeable in a LAN environment for regular office productivity applications even in cases where each individual file access operation requires separate communication; this represents a rather pathological situation.

References

[1] ANDERSON, J. P. Computer Security Technology Planning Study. Tech. Rep. ESD-TR-73-51, Air Force Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, MA, Oct. 1972. AD-758 206, ESD/AFSC.

[2] ANDERSON, J. P. A Unification of Computer and Network Security Concepts. In *Proc. IEEE Symposium on Security and Privacy* (1985), pp. 77–87.

[3] BOND, M. Attacks on Cryptoprocessor Transaction Sets. In *Cryptographic Hardware and Embedded Systems – CHES 2001. Proceedings of the Third International Workshop, May 14–16 2001, Paris, France* (Heidelberg, Germany, May 2001), Ç. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 220–234.

[4] CHITTURI, A. Implementing mandatory network security in a policy-flexible system. Master's thesis, Department of Computer Science, University of Utah, Salt Lake City, June 1998.

[5] CUSTER, H. *Inside Microsoft Windows NT File System*. Microsoft Press, Redmond, WA, USA, 1994.

[6] FORD, B., HIBLER, M., LEPREAU, J., MCGRATH, R., AND TULLMANN, P. Interface and execution models in the fluke kernel. In *Proceedings of the 3rd Symposium on Operating Systans Design and Implementation (OSDI-99)* (Berkeley, CA, Feb. 22–25 1999), Usenix Association, pp. 101–116.

[7] FRASER, T., BADGER, L., AND FELDMAN, M. 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), IEEE Press, pp. 2–16.

[8] LOSCOCCO, P. A., SMALLEY, S. D., MUCKELBAUER, P. A., TAYLOR, R. C., TURNER, S. J., AND FARRELL, J. F. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *Proceedings of the 21st National Information Systems Security Conference* (1998), National Computer Security Center, pp. 303–314.

[9] MINEAR, S. E. Providing policy control over object operations in a Mach based system. In *5th USENIX UNIX Security Symposium, June 5–7, 1995. Salt Lake City, UT* (Berkeley, CA, USA, June 1995), USENIX, Ed., USENIX, pp. 141–155.

[10] NAGAR, R. *Windows NT File System Internals*. O'Reilly & Associates, Sebastopol, CA, USA, 1997.

[11] NEBBETT, G. *Windows NT/2000 Native API Reference*. Macmillan Technical Publishing, Indianapolis, IN, USA, 2000.

[12] SAYDJARI, O. S., BECKMAN, J. M., AND LEAMAN, J. R. LOCK trek: Navigating uncharted space. In *Proc. IEEE Symposium on Security and Privacy* (1989), pp. 167–175.

[13] SMITH, R. E. Cost Profile of a Highly Assured, Secure Generating System. *ACM Transactions on Information and System Security 4*, 1 (Feb. 2001), 72–101.

[14] SOLOMON, D. A., AND RUSSINOVICH, M. E. *Inside Microsoft Windows 2000*, 3rd ed. Microsoft Press, Redmond, WA, USA, 2000.

[15] SPENCER, R., SMALLEY, S., LOSCOCCO, P., HIBLER, M., ANDERSEN, D., AND LEPREAU, J. 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.

[16] TYGAR, J. D., AND YEE, B. Dyad: a system using physically secure coprocessors. In *Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment* (MIT, Program on Digital Open High-Resolution Systems, Jan. 1994), The Journal of the Interactive Multimedia Association Intellectual Property Project, Coalition for Networked Information, Interactive Multimedia Association, John F. Kennedy School of Government, pp. 121–152.

[17] UNITED STATES DEPARTMENT OF DEFENSE. *DoD 5200.28-STD: Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)*, 1985.

[18] WALKER, B. J., KEMMERER, R. A., AND POPEK, G. J. Specification and Verification of the UCLA Unix Security Kernel. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)* (1979), pp. 64–65.

[19] WHITE, S. R., AND COMERFORD, L. ABYSS: A trusted architecture for software protection. In *Proc. IEEE Symposium on Security and Privacy* (1987), pp. 38–52.

[20] WOLTHUSEN, S. Formal Logic for Security Policy Definition and Enforcement. Under preparation., 2001.

[21] WOLTHUSEN, S. Layered multipoint network defense and security policy enforcement. In *Proceedings from the Second Annual IEEE SMC Information Assurance Workshop, United States Military Academy, West Point, NY* (June 2001), pp. 100–108.

[22] WOLTHUSEN, S. Security Policy Enforcement at the File System Level in the Windows NT Operating System Family. In *Proceedings 17th Annual Computer Security Applications Conference (ACSAC'01), New Orleans, LA* (Dec. 2001). To appear.

[23] WOLTHUSEN, S. Sensitivity Labels and Invisible Identification Markings in Human-Readable Output. In *Proceedings of Electronic Imaging 2002, San Jose, CA* (Jan. 2002), The International Society for Optical Engineering (SPIE). To appear.