Tempering Network Stacks

Dr.-Ing. Stephen D. Wolthusen
Fraunhofer-IGD
Security Technology Department
Fraunhoferstr. 5
64283 Darmstadt
Germany

E-Mail: wolt@igd.fhg.de

SUMMARY

This paper summarizes existing and describes ongoing work on securit ypolicy definition and particularly enforcement in heterogeneous distributed systems. Based on a formal model of operating systems and interactions among networked nodes in a distributed system axiomatizing relations among and abstractions in distributed systems, arbitrary security policies can be defined over the same model; automated reasoning techniques can be used to dynamically derive the compliance of operations with all applicable security policies. A key component for enforcing such security policies in operating system network stacks is described along with instrumentation techniques for the Microsoft Windows NT family of operating systems.

1.0 INTRODUCTION

Information assurance in distributed, heterogeneous systems frequently requires that formal and informal security policies be enforced by technical means. The expressiveness required by security models and, more generally, policies [21], however, frequently exceed the capabilities of the mechanisms available in currently deployed networking components and general purpose operating systems.

Extant (deployed) systems, particularly for network security policies are generally limited to simple access control lists and in some cases to elementary heuristics in the scope of their proactive security mechanisms both in case of operating system capabilities and add-on components.

Similar limitations also exist in the capabilities of the controls themselves since existing controls are typically limited in their design to enforce simple access control mechanisms, e.g.~in the form of basic or stateful packet filtering mechanisms integrated into operating systems.

A number of threats increasingly necessitate improvement of security policy mechanisms, controls, and the assurance provided by such controls even in nominally secured networks. Topologically oriented security mechanisms such as perimeter firewalls increasingly undermined through the use of a variety of wireless network interfaces including IEEE 802.11x, Bluetooth, and even IrDA/FIR frequently found on standard workstations that support ad hoc networking among peer nodes.

Threats inherent in this include denial of service, eavesdropping, and active penetrations, but more importantly also represent a vector through which malicious code can be inserted into a network to cause arbitrary damage. Frequently, such access is provided deliberately by end users, e.g.~when establishing a

piconet or point-to-point network interface connections for sharing materials written using popular office productivity applications with extensive macro capabilities and therefore also vulnerability surfaces for malware.

Moreover, even network traffic passing through perimeter security controls is increasingly opaque to such perimeter firewalls and network intrusion detection mechanisms. Reasons for this include the use of encrypted end-to-end channels or message formats that partially blind perimeter network security mechanisms nominally capable of scanning network traffic, but also the proliferation of protocols that are explicitly designed to circumvent network security controls such as the SOAP protocol [4,14] and additional layered protocols.

These observations lead to a number of desiderata for improving network stack security. Since information required for reaching decisions regarding conformity of network traffic and operations is increasingly available only at the end nodes themselves and end nodes may also be directly exposed to hostile traffic, network security controls must be integrated directly into the end nodes themselves [3].

2.0 SECURITY ARCHITECTURE BACKGROUND

General security policies within organizations, typically created only in informal prose, must be mapped onto available security models and ultimately security controls, potentially losing accuracy at each of these steps and also potentially incomplete because of limitations of the layer mapped onto in each step. Such mapping errors can be detrimental both in omitting controls that policies call for and in imposing overly restrictive controls that limit capabilities and effectiveness of the information system.

Moreover, demonstrating the correspondence of each mapping (e.g. document handling guidance onto technical access controls) is a resource-intensive effort and similarly prone to errors and oversights as the original mapping.

In modeling individual nodes (i.e. operating systems and the resources controlled by these systems) and interactions among nodes at a level of abstraction sufficient to capture operational semantics across multiple general-purpose operating systems through formal concept analysis using formal logic, bijective mappings onto specific instances of operating systems can be considered interpretations of such formal theories.

Arbitrary security policies can then be formulated within the same formal theory, interpreted as either permitted or required operations. Automated deduction mechanisms can therefore be used to derive additional statements and instances of the model.

By including abstraction relations over entities and operations within the axiomatization of the underlying system, the reasoning can, moreover, occur over multiple abstraction layers, such as deriving the permissibility of an read operation accessing an individual block on a fixed disk by a given process based on abstractions tracing these entities and operations onto e.g. personnel roles and documents; this can be achieved by mapping an operation onto the formal model in the form of a well-formed formula, a proof of the validity of such a hypothesis (derived e.g. via term rewriting or automated deduction mechanisms) is then considered permission to perform the operation.

The axiomatization, based on embedding algebraic (e.g. lattice) structures within the formal theory provides critical efficiency gains not only in a priori providing proof structuring aids but also in permitting the re-use of decisions. By embedding lattices over both entity and operation types and over entity identities, policy decisions can be reached quickly by avoiding resolution to ground termns and re-used by simple rewriting in later decisions. Such derived (proven) formulae can be considered part of the policy set

with legitimate operations being described by the Lindenbaum operator over all policies with each such statement being assigned a lifetime providing implicit pruning and dynamism [31].

In the underlying architecture, there exist a number of nodes called external reference monitors (ERM) which are repositories for one or more security policies, each presumably derived from a security model. The system on which an ERM resides is called a Policy Controller Node (PCN). The other component of the framework consists of a number of nodes which are subject to the policies of one or more ERM [27,31].

The policies obtained from ERM are enforced through externally controlled reference monitors (ECRM) and its enforcement modules (EM); a system configured with a combination of ECRM and EMs is called a Policy Enforcing Node (PEN). As implied by the term reference monitor, each operation of the controlled nodes is mediated by the ECRM and may only proceed if it is found to be in compliance with all applicable policies. Applicable policies (and hence the ERMs to be consulted) are determined from the identity of subjects and objects involved which are uniquely identified by the conjunction of a subject identity and a subject type constant.

3.0 NETWORK ENFORCEMENT MECHANISMS

In addition to other system components such as device interfaces [32], file systems [28], and process management required for ensuring the completeness property for reference monitors, network interfaces constitute one of the minimum required controls for security policy enforcement.

The network enforcement module must satisfy a number of functional requirements, namely to control all in- and outbound data packets and circuit operations in such a way that data flows are presented to the host operating system only after having been validated; this is particularly relevant for operating systems where the network stack may not be capable of properly handling malformed data flows.

Moreover, the network enforcement module must provide transparent data object labeling to permit the identification of higher level entities in case of data flows among nodes within the security architecture.

To establish secure in-band communication with PCN nodes that may not be possible because of state space restrictions in using the host operating system network stack, the enforcement module must also provide a fully separated cryptographically secured communication channel.

This paper describes one such ongoing enforcement module implementation for the Microsoft Windows NT family[1] of operating systems.

However, the following sections concentrate mainly on the instrumentation mechanisms and omit more advanced object identification and security protocol elements.

3.1 Windows NT Family Network Protocol Stacks

Unlike the other components such as file system handling, the networking mechanisms provided by the Microsoft Windows NT operating system family do not share a common abstraction for all supported types of network communication.

Therefore, in addition to multiple environmental subsystems providing different access mechanisms to network communication subsystems, there exist several conceptually different networking application programming interfaces, namely

[1] including the 3.*x*, 4.0, 2000, XP, and 2003 versions at the time writing.

- WinSock
- Named Pipes
- Mailslots
- Remote Procedure Call
- NetBIOS
- Telephony

Other services such as DCOM [8] or the .NET framework [26,18] may be layered on top of these interfaces; while some of these interfaces have their own security and encryption mechanisms (such as RPC), others rely on the connection being assumed as secure and simply enforce access controls (e.g. named pipes and mail slots which are implemented as file systems and can use the access control mechanisms for file systems, see [23,24]).

Of these mechanisms, the telephony interfaces (TAPI) are in an unique class based on the mechanism used by user level programs to communicate with kernel-level components.

The TAPI user level component (`TAPISRV.DLL`) provides access to a number of TAPI service providers (TSP); while most of these map to networking subsystems discussed later in this section, this also includes direct access to device drivers for modem devices (which can themselves be used to establish arbitrary network connections including interfacing to other network protocols).

This particular component therefore requires specific enforcement mechanism support (e.g. in the form of device-level enforcement for modem-type devices) to avoid the introduction of unenforceable information flow paths.

In the general case, the network architecture of the Microsoft Windows NT family consists of a number of layers, depicted in figure 1.

At the lowest level is the physical device. Access to individual devices is regulated by the hardware abstraction layer (HAL). Network device drivers are generally realized as NDIS (Network Driver Interface Specification) modules consisting of the generic NDIS library and the device-specific NDIS miniport drivers; the library fully encapsulates the miniport drivers.

Accessing the NDIS library is the TDI (Transport Driver Interface) mechanism. This itself consists of transports (or protocol drivers), supporting the various transport mechanisms such as NetBEUI (NetBIOS Extended User Interface) and TCP/IP, and TDI clients which provide services for sockets and NetBIOS calls. None of these modules can be called directly from applications since they are protected kernel mode interfaces.

Upper-level APIs (application programming interfaces) such as NetBIOS and Windows Sockets are subsequently implemented at the user level and must use the aforementioned interface layers.

The Windows Sockets API (or WinSock) is modeled after the original BSD Unix sockets API [20] and has undergone significant revisions under various platforms before arriving in its current form [1,2]. It is available for both the NT-based and DOS-based operating systems from Microsoft Corporation.

The Windows Sockets API is itself composed of several modules. From an application's perspective the sockets API consists of the exposed API DLL; this DLL (dynamically linked library) communicates with the SPI (Service Provider Interface) layer.

This layer is controlled by the transport service provider DLL which in turn calls on a number of transport helper DLLs and namespace helper DLLs to perform its operations. Moreover, the transport service provider DLL forwards the thus generated calls to the System Support Library DLL that represents the interface to the abovementioned kernel components.

Since the Microsoft Windows NT design is predicated on a file system model and represents sockets as file handles, a translation mechanism is required. This service is performed by an Ancillary Function Driver (AFD).

Of particular interest in this is the ability to stack several of the transport helper DLLs so as to provide additional services at each level (there is no layering mechanism for namespace helper DLLs). WinSock here distinguishes between “base protocols” and “layered protocols”. The former are protocols capable of performing actual communication with a remote endpoint, the latter must rely on base protocols for actual communication and only provide added value.

Provided that all elements of such a stack are conforming to the interface specifications set forth in [1,2], it is possible to implement several stacked layers of such layered protocols.

At an abstraction level below the user level API mechanisms, the protocol driver layer accepts requests from API-level mechanisms and translates these into respective network protocol elements. The number and type of protocol drivers are variable among nodes and may include but are not limited to TCP/IP, NetBEUI, IPX/SPX (provided in a single protocol driver instance), and AppleTalk. Typically, each protocol driver supports all protocols of a protocol family (e.g. IP, ARP, RARP, ICMP, IGMP, UDP, and TCP in case of the TCP/IP protocol driver, `TCPIP.SYS`).

All protocol drivers communicate with API-level components (as well as other components such as the previously noted Windows Sockets ancillary function driver) using part of the TDI; which is specified in the form of IRP classes.

For connection-oriented protocols, `TdiDispatchCreate` creates a file object (also referred to as an address object) by through the use of an `IRP_MJ_CREATE` IRP which represents the node-local connection endpoint. This subsequently must be associated with an opened file object representing an address, referred to as an connection object.

Depending on the initiating node, subsequent IRP messages must then transition the connection object into listening or connecting state, which is then transitioned into an accepting state on the part of the listening node, which occurs using the IRP creation mechanisms `TdiDispatchDeviceControl`, `TdiDispatchFastDeviceControl`, and `TdiDispatchInternalDeviceControl`, respectively.

After a connection object has been discarded, `TdiDispatchClose` is used to discard the address object after `TdiDispatchCleanup` has ensured that no pending IRP messages exist for the address object; connectionless protocols omit the listening and connecting phase of this control flow.

TDI also permits the use of callback mechanisms and the intermediate caching of network protocol data units for efficiency purposes; this requires the registration of events with TDI client interfaces. Typically, this results in TDI clients generating `TDI_SEND` IRP messages and reacting to `TDI_EVENT_CHAINED_RECEIVE` and the `TDI_EVENT_RECEIVE*` family of IRP messages.

For communication with the device drivers controlling the network interface adapters, the TDI protocol drivers communicate by way of a library encapsulating device-specific properties.

This library, NDIS, provides a procedural interface for the TDI as well as for the actual device drivers (miniport drivers), which communicate to the remainder of the operating system only through the NDIS library.

Internally, however, the Microsoft Windows NT implementation of NDIS itself uses IRP-based messaging for control flow. The NDIS library provides services for both connectionless (e.g. IP) and connection-oriented (e.g. ATM) protocols as well as a number of other services [24].

NDIS also provides several other security-relevant services that need to be addressed, such as the ability to forward datagrams from one network interface to another without processing by the remaining operating system network protocol stack or the offloading of certain network processing (specifically TCP/IP-related operations) to the network interface device and hence the NDIS level.

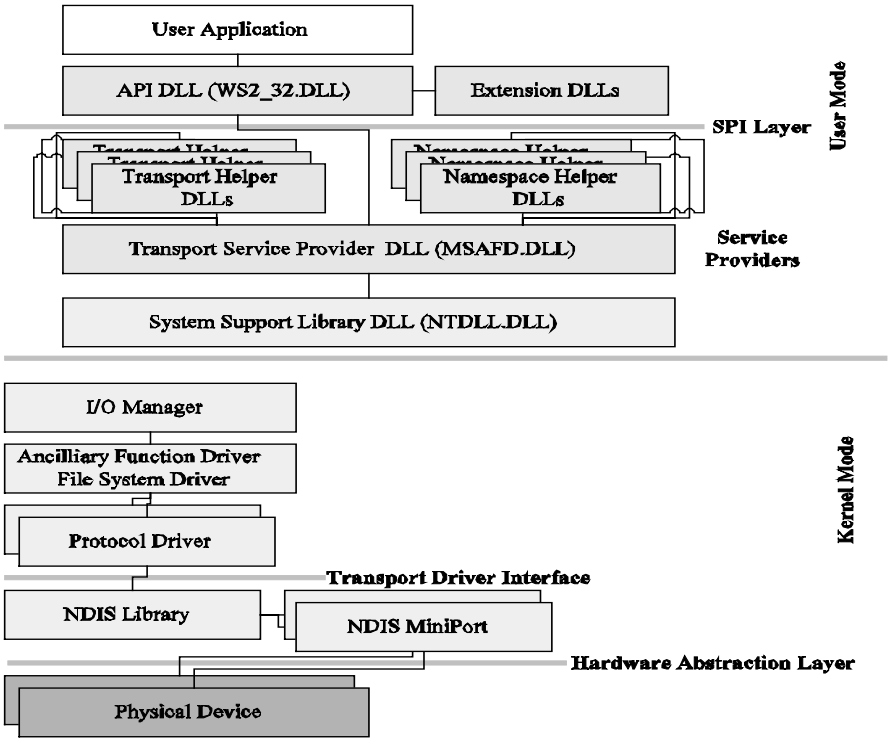

Figure 1: Components involved in networking mechanisms in the Microsoft Windows NT operating system family.

3.2 Protocol Stack Integration

The provision of the semantics appropriate for a network enforcement layer requires the insertion of instrumentation at least at two of the protocol layers described in the preceding section, namely at the NDIS and TDI layers.

3.2.1 NDIS Instrumentation

With the exception of parts of TAPI discussed above, all network traffic within the Microsoft Windows NT family is transmitted by way of NDIS devices, regardless of the API and protocol used; it is also possible for a user level process to directly communicate with the NDIS layer (again, TAPI is an example of this behavior).

It is therefore imperative for the provision of the required interpretation semantics to intercept and instrument the processing at the NDIS level. For this purpose, several implementation alternatives exist, two of which are of sufficient generality for the purposes discussed here.

One possible approach is the use of an NDIS Intermediate Driver, which permits the interpositioning of code between miniport drivers and the remainder of the NDIS library. While appealing and providing a well-defined interface for interposition, this approach does not provide the most general mechanism since NDISWAN miniport drivers are not supported in the NDIS revision (version 5.0 and 5.1, respectively) used by Microsoft Windows 2000, XP, and 2003.

This would require the mandatory use of backwards-compatible NDIS version 4.0 mechanisms, which for obvious reasons is highly undesirable given the improvements and features added in NDIS version 5.*x*.

The alternative to intermediate drivers providing the most general coverage of mechanisms supported is in the manual interception of control flows destined for and within NDIS.

Since it is possible that the configuration of both protocols and network interfaces may change at any time during runtime (e.g. through the addition of an ad-hoc network interface), a general mechanism is required that supports not only bootstrapping mechanisms but also provides monitoring and dynamic interception of such configuration changes. For this purpose, an NDIS layer enforcement sub-module can be loaded and started prior to the initialization of the network subsystem.

Since the NDIS architecture differs significantly in initialization and particular I/O flow from normal device I/O unter the Windows NT platform, however, the interception cannot be effected by registering with the I/O manager and redirecting the flow of IRP messages, but must occur directly by redirecting function entry points to the enforcement sub-module itself and subsequent transfer of control flow to the NDIS library once the required operations have been performed on the part of the enforcement sub-module.

To ensure that policies can be enforced uniformly, all network interfaces on a node must be intercepted and brought under the control of the enforcement sub-module. This occurs by intercepting the NDIS functions `OpenAdapter` and `CloseAdapter` and tracking the activation and deactivation of any (virtual) network interface; the actual interception mechanism relies on modifying the addresses contained in the export table of the module providing the NDIS library upon loading of the NDIS module.

Similarly, to be able to track information and control flows — particularly for callback mechanisms — the enforcement sub-module must retain information on which protocol drivers are registered with (and hence may access) the NDIS layer. This is accomplished by intercepting the `NdisRegisterProtocol` functions for registration and, correspondingly for unloading and deregistration, the `NdisMRegisterUnloadHandler` and `NdisDeregisterProtocol` functions.

The information thus obtained permits the correlation of information and subsequent coordination with instrumentation provided by enforcement sub-modules at the protocol driver level discussed in the following section.

While NDIS is the proper location to capture all control and data flows pertaining to network traffic and therefore also to perform protocol-specific operations, the information available at the level of the NDIS library (and hence the interception mechanism) are severely limited. At the level of the NDIS library, it is not directly possible to identify the subject (i.e. process) a data flow is associated with since the data flow from a process directed towards NDIS is translated into IRP messages at the kernel level, thus obliterating the information on the subject.

Conversely, data flows directed towards subjects are not associated with processes directly, but only with protocol drivers. It is therefore necessary (as described in the following section) to correlate information regarding the subject association with a data flow by coordinating the information available at the NDIS level with information from higher abstraction levels to permit the employment of security controls available only at the lower NDIS layer.

A similar problem exists with regard to the payload of the individual data flows processed by NDIS. At the NDIS level, only data already processed into protocol data units (PDU) are presented, and NDIS is expected to operate opaquely on the data provided in either direction. In this case, the information as to which protocol is associated with a PDU is obtained indirectly through the information gathered on registration of protocols.

A list of known protocol drivers must be maintained (this can occur through known identifying characteristics within the protocol driver or indirectly through the file system enforcement mechanism providing an unique fingerprint for a given protocol driver by way of the ECRM), and protocol-specific operations must be invoked on the PDU based on the information thus obtained.

However, since PDU may be constrained either by the respective protocol or by the network interface, it is not always possible to transform PDU in place. Instead, a given PDU (regardless of inbound or outbound processing) may result in several PDU after processing by the protocol-specific enforcement sub-module and, moreover, the protocol-specific enforcement sub-module can withhold the processed PDU (and hence process additional PDU from the same data flow) prior to emitting one or more PDU for further processing by the NDIS layer.

The necessary information for identifying subjects, objects, and operations are transmitted by the TDI sub-module as discussed in the following section. Individually, the instrumentation provided by the NDIS layer enforcement sub-module can monitor the activation and deactivation of protocols and adapters as well as monitor in- and outbound data flows, including the elimination of inbound traffic as well as outbound[2].

Another operation that can be performed by the NDIS layer without interoperation with other sub-layers is data flow normalization, i.e. providing well-defined temporal characteristics for all or selected data streams such as inter-PDU time intervals. This, however, requires potentially large buffers in the absence of flow control mechanisms that can be applied transparently to the communicating parties proper.

3.2.2 TDI Instrumentation

While the enforcement mechanism proper is located at the NDIS level as described in the preceding section, the implementation of the Microsoft Windows NT operating system family necessitates the addition of a further enforcement sub-module at the protocol driver level.

[2] The `NdisCancelSendPackets` and `NdisGeneratePartialCancelId` mechanisms are, while not strictly necessary for this purpose, supported only from NDIS 5.1 onward.

The need for this additional sub-module stems from the lack of information regarding the association of subjects (and potentially of operations) as well as of objects of higher abstraction levels from which a given object or PDU is derived at the NDIS layer.

However, as noted before, there are potentially multiple protocol drivers active within a given node, each of which requiring specific actions for deriving the requisite information for reaching policy decisions by the ECRM in conjunction with other sub-module information. For the purposes of this dissertation, the discussion concentrates without loss of generality on the TCP/IP protocol driver.

Interception of the protocol driver occurs analogous to the mechanism described for the NDIS library in the preceding section; entry points are dynamically redirected on initialization of the protocol driver and forwarded after processing. As with the NDIS layer, this facilitates dynamic addition and removal of protocol drivers at runtime provided that the proper enforcement sub-module for a given TDI protocol driver is available.

The main operation performed at the TDI enforcement sub-module is the collection of information regarding subjects, objects, and data flows (the latter information is available implicitly through the observation of calls to the TDI); subject (i.e. process information that can be correlated with other subject information at the ECRM) information is implicitly available through the calling mechanism. In case of an outbound data flow, the information thus gathered must be made available to the NDIS sub-module to permit proper processing.

While it would be conceivable to transmit this information out of band or to store it at the ECRM itself, both possible alternatives would require not only considerable storage, but also imply complex storage management since the processing order is not necessarily the same for data flows at the TDI and NDIS layer, and special cases such as canceled data flows would need to be taken into account to avoid stale storage.

To avoid these problems as well as performance issues arising from extraneous communication between sub-modules (typically in the form of IOCTL messages that require considerable processing overhead), data flows can be annotated in-band with the requisite information. The NDIS sub-module can extract this information[3] and continue processing as described in the preceding section.

Similarly, inbound data flows can be reverse-associated with the information regarding subjects, objects, and data flows. This requires one instance of communication between the NDIS and TDI sub-modules for each flow (in the worst case of connectionless protocols, this is once for each PDU, although heuristics and information from other sub-modules not discussed here can be established to identify virtual flows based on addressing information in the more general case of connectionless protocols).

4.0 DEVELOPMENTAL ASSURANCE ASPECTS

Overall assurance achievable by the security architecture discussed here is, to a first approximation, limited by the lowest level of assurance of any component.

The set of components first and foremost also includes the host operating system in case of retrofits of security mechanisms as described in this paper since defects therein can potentially compromise or bypass any additional security controls. However, since the requirement for using systems with such limited overall assurance exists — primarily because of application program availability — it is imperative to provide sufficient levels of developmental assurance within said confines.

[3] There exists a mechanism for this purpose in the `NDIS_PACKET_STACK` structure introduced in NDIS 5.1; prior NDIS versions require the allocation of a new, larger packet for the integration of this data.

This can be achieved by including the system to be instrumented in the process of formally modeling the enforcement module. By not relying (solely) on documentation but rather performing reverse engineering and evaluation of the network stack components and capturing both expected and observed behavior in the model (primarily using the Z notation [25,15]), certain types of flaws based on incorrect assumptions or documentation can be avoided or, if subsequent evaluation results in assumptions becoming invalidated, new information can be incorporated into the model rigorously.

5.0 DISCUSSION AND RELATED WORK

The mechanisms described here represent part of a larger security architecture that touches upon a number of fields; the discussion here is restricted to network security policy mechanisms and implementation strategies.

Based on observations on the use of mobile devices, remote access mechanisms, and the performance requirements for choke point firewalls resulting from increasing network performance, Bellovin proposed the migration of firewall enforcement to the nodes to be protected while retaining a central policy definition for network access control [3]. Instead of relying on topological information for obtaining statements on the identity of an entity, Bellovin proposed to use the mechanisms for the use of public key infrastructures inherent in IPSec which also addressed the issue of support for virtual private networks found in traditional firewalls [17], although this was done earlier by Chitturi [7] in 1998 within the Fluke project context [10].

Another approach to distributed firewalling, also derived from concepts introduced by Bellovin was pursued by Payne and Markham at SCC. Payne and Markham realized the embedding of the firewalling mechanisms (EFW) on a COTS network interface card with cryptographic and limited processing capabilities [19,22]. A similar mechanism for EFWs was also developed by Friedman and Nagle at Carnegie Mellon University [11,12].

Policy-based network security has been the subject of intensive research; Burns *et al.* describe a network security policy architecture based on security models at moderate abstraction levels [5]; for an earlier survey of such mechanisms see [6].

The problem of insufficient instrumentation for security purposes has also been addressed by other researchers; Keromytis describes a data flow tagging architecture similar to the one described here for the OpenBSD environment, these are also used in OpenBSD to record buffer- (packet-) specific information such as security data that is not retained in normal data and control flows [16]; more specialized interposition mechanisms include, among others, work on Exokernels at MIT by Kaashoek *et al.* [9] and SLIC by Ghormley *et al.* [13].

6.0 CONCLUSIONS AND FUTURE WORK

This paper has described an instrumentation mechanism for enforcing flexible and dynamic security policies imposed by a distributed security policy mechanism and a specific implementation thereof for retrofitting an instrumentation mechanism onto a COTS (commercial off the shelf) operating system without access to source code. While the modeling of the host operating system under such conditions requires significant resources, it also ensures that major discrepancies between intended (documented) and actual behavior are discovered.

Uses of the network policy mechanisms include dynamic distributed firewalling [27] and intrusion detection and response [29,30] as described in earlier papers.

Ongoing extensions to the instrumentation include the implementation of an in-line (*bump-in-the-stack*) IPSec mechanism for the Internet protocol family and further refinements of the mirror network stack mechanism itself.

In addition, measurements and tuning are a major focus of ongoing work as the latency imposed by the mirror stack and policy enforcement can become pronounced for TCP/IP connections when used rapid circuit establishment and teardown occur over high speed network interfaces.

7.0 REFERENCES

[1] ANDERSEN, D. B. Windows Sockets 2 Application Provider Interface. Techn. Rep. Intel Corp., May 1997, Version 2.2.1

[2] ANDERSEN, D. B. Windows Sockets 2 Service Provider Interface. Techn. Rep. Intel Corp., May 1997, Version 2.2.1

[3] BELLOVIN, S. M. Distributed Firewalls. *;login. The USENIX Association Newsletter 19*, Special issue on security (Nov. 1999), 39-47.

[4] BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSOHN, N.,NIELSEN, H.F., THATTE, S., AND WINER, D. Simple Object Access Protocol (SOAP) 1.1. Techn. Rep. W3C, May 2000. Status: W3C Note.

[5] BURNS, J. CHENG, A., GURUNG, P., RAJAGOPALAN, S., RAO, P., ROSENBLUTH, D. SURENDRAN, A., AND MARTIN, JR. D. M. Automatic Management of Network Security Policy. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX II)* (Anaheim, CA, USA, June 2001), IEEE Press, pp.1012-1026.

[6] CARNEY, M., AND LOE, B. A Comparison of Methods for Implementing Adaptive Security Policies. In *Proceedings of the 7th USENIX Security Symposium* (San Antonio, TX, USA, Jan. 1998), USENIX, pp.1-14

[7] CHITTURI, A. Implementing Mandatory Network Security in a Policy-flexible System. Master's thesis, University of Utah Department of Computer Science, Salt Lake City, UT, USA, June 1998.

[8] EDDON, G., AND EDDON, H. *Inside Distributed COM*. Microsoft Press, Redmond, WA, USA, 1998.

[9] ENGLER, D.R., AND KAASHOEK, M.F., AND O'TOOLE, JR., J. Exokernel: An Operating System Architecture for Application-Level Resource Management. *ACM Operating Systems Reviewd 29,5* (Dec. 1995), 251-266. Proceedings of the 15th Symposium on Operating Systems Principles (15th SOSP '95).

[10] 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 Systems Design and Implementation (OSDI '99)* (New Orleans, LA, USA, Feb. 1999), ACM Press, pp. 101–116. Published as an ACM Operating Systems Review Special Issue.

[11] FRIEDMAN, D., AND NAGLE, D. F. Building Scalable Firewalls with Intelligent Network Interface Cards. Tech. Rep. CMU-CS-00-173, Carnegie Mellon University School of Computer Science, Pittsburgh, PA, USA, Dec. 2000.

[12] GANGER, G. R., AND NAGLE, D. F. Enabling Dynamic Security Management of Networked Systems via Device-Embedded Security. Tech. Rep. CMU-CS-00-174, Carnegie Mellon University School of Computer Science, Pittsburgh, PA, USA, Dec. 2000.

[13] GHORMLEY, D. P., RODRIGUES, S. H., PETROU, D., AND ANDERSON, T. E. Interposition as an Operating System Extension Mechanism. Tech. Rep. CSD-96-920, University of California at Berkeley, Berkeley, CA, USA, Apr. 1997.

[14] GUDGIN, M., HADLEY, M., MOREAU, J.-J., AND NIELSEN, H. F. Simple Object Access Protocol (SOAP) 1.2, Dec. 2001. W3C Working Draft, consists of three parts.

[15] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND INTERNATIONAL ELECTROTECHNICAL COMMITTEE. *Information Technology — Z Formal Specification Notation — Syntax, Type System and Semantics. International Standard 13568*, 2002.

[16] KEROMYTIS, A. D. Tagging Data in the Network Stack: mbuf tags. In *Proceedings of BSDCon '03* (San Mateo, CA, USA, Sept. 2003), USENIX, pp. 125–132.

[17] KEROMYTIS, A. D., AND WRIGHT, J. L. Transparent Network Security Policy Enforcement. In *Proceedings of the 2000 USENIX Annual Technical Conference* (San Diego, CA, USA, June 2000), USENIX, pp. 215–226.

[18] LAMACCHIA, B., LANGE, S., LYONS, M., MARTIN, R., AND PRICE, K. *.NET Framework Security*. Addison-Wesley, Reading, MA, USA, 2002.

[19] MARKHAM, T., AND PAYNE, C. Security at the Network Edge: A Distributed Firewall Architecture. In *Proceedings of the DARPA Information Survivability Conference (DISCEX II)* (Anaheim, CA, USA, June 2001), pp. 18–27.

[20] MCKUSICK, M. K., BOSTIC, K., KARELS, M. J., AND QUARTERMAN, J. S. *The Design and Implementation of the 4.4 BSD Operating System*. Addison Wesley, Reading, MA, USA, 1996.

[21] MCLEAN, J. Security Models. In *Encyclopaedia of Software Engineering*, J. J. Marciniak, Ed. John Wiley & Sons, Inc., New York, NY, USA, 1994, pp. 1136–1145.

[22] PAYNE, C., AND MARKHAM, T. Architecture and Applications for a Distributed Firewall. In *Proceedings 17th Annual Computer Security Applications Conference (ACSAC'01)* (New Orleans, LA, USA, Dec. 2001), IEEE Computer Society Press, pp. 329–336.

[23] SOLOMON, D. A., AND CUSTER, H. *Inside Microsoft Windows NT*, 2nd ed. Microsoft Press, Redmond, WA, USA, 1998.

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

[25] SPIVEY, J. M. *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall International Series in Computer Science. Prentice Hall, London, UK, 1992.

[26] THAI, T., AND LAM, H. Q. *.NET Framework Essentials*. O'Reilly & Associates, Sebastopol, CA, USA, 2002.

[27] 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, USA, June 2001), IEEE Press, pp. 100–108.

[28] 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, USA, Dec. 2001), IEEE Press, pp. 55–63.

[29] WOLTHUSEN, S. Distributed Intrusion Detection for Policy-Controlled Heterogeneous Environments. In *Proceedings from the Third Annual IEEE SMC Information Assurance Workshop*, United States Military Academy (West Point, NY, USA, June 2002), IEEE Press, pp. 255–262.

[30] WOLTHUSEN, S. Embedding Policy-Controlled ID Sensors within Host Operating System Security Enforcement Components for Real Time Monitoring. In *Proceedings of the NATO RTO Symposium on Real Time Intrusion Detection Symposium (RTID)* (Estoril, Portugal, May 2002), NATO Research and Technology Organization. Publication RTO-MP-101.

[31] WOLTHUSEN, S. *A Model-Independent Security Architecture for Distributed Heterogeneous Systems*. Logos Verlag, Berlin, Germany, 2003.

[32] WOLTHUSEN, S. Goalkeeper: Close-In Interface Protection. In *Proceedings 19th Annual Computer Security Applications Conference (ACSAC'03)* (Las Vegas, NV, USA, Dec. 2003), IEEE Press, pp. 334– 341.