SAND REPORT

SAND2001-3213

Unlimited Release

Printed October 2001

Final Report for the Intrusion Detection for Asynchronous Transfer Mode (ATM) Networks Laboratory Directed Research and Development Project

Thomas D. Tarman, Edward L. Witzke, Brian R. Kellogg, Keith C. Bauer, and
William F. Young

Sumit Ghosh - Stevens Institute of Technology

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401

Facsimile: (865)576-5728

E-Mail: reports@adonis.osti.gov

Online ordering: <http://www.doe.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847

Facsimile: (703)605-6900

E-Mail: orders@ntis.fedworld.gov

Online order: <http://www.ntis.gov/ordering.htm>

SAND2001-3213
Unlimited Release
Printed October 2001

Final Report for the Intrusion Detection for Asynchronous Transfer Mode (ATM) Networks Laboratory Directed Research and Development Project

Thomas D. Tarman, Edward L. Witzke, and Brian R. Kellogg
Advanced Networking Integration Department

Keith C. Bauer
Advanced Decision Support Applications Department

William F. Young
Networked Systems Survivability and Assurance Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Sumit Ghosh
Stevens Institute of Technology
Department of Electrical & Computer Engineering
Hoboken, NJ 07030

Abstract

Asynchronous Transfer Mode (ATM) is widely used as a backbone for corporate and public telephone and data networks, and its availability is paramount to the correct operation of the Internet Protocol (IP) and higher-layer applications. Key to the correct operation of ATM is its set of control plane protocols, which perform functions such as virtual circuit signaling and routing. If these protocols are attacked, then denial of service can result. Therefore, a system is required to monitor these control plane protocols, and respond if protocol events indicate a possible attack. Before this project, no system existed that performed Layer 2 (e.g., ATM, switched Ethernet, etc.) intrusion or misuse detection.

ATM is a switched network technology, and other protocols (notably IP) are also being implemented in a switched fashion. Therefore, the general problem of intrusion detection in a switched network is receiving more attention. This paper describes in detail the Switched Network Intrusion DETection (SNIDE) system that was developed under a Sandia Laboratory-Directed Research and Development project. This system was designed to perform control plane intrusion detection on a general switched network, and implemented to perform intrusion detection specifically on ATM networks. This system includes components that implement intrusion sensing, assessment, and response, along with management tools and graphical user interfaces to configure system components. Initial results have shown that this system operates as expected, and vendors have expressed interest in possible licensing of components from the SNIDE system.

Acknowledgments

The authors wish to thank Bob Hutchinson for his ideas and help with concept development during the proposal stage and early in the project. We would also like to thank Tim Draelos for his work early in the project, and for his ideas in adaptive intrusion detection. We are grateful to Andy McFarland of the U.S. DoD for his support and for providing us with a different application scenario. Finally, we would like to thank Marcus Epperson for his help in the review of this report.

“SSH” is a registered trademark of SSH Communications Security

All other trademarks belong to their respective owners.

Contents

Acronyms.....	6
Introduction	8
Example ATM Attack – Peer Group Leader Take-Over Attack	8
SNIDE Architecture and Components.....	10
Assessment Engine	11
Responses	13
Rule Editor	13
Software-Based Direct Sensors.....	14
UNI Sensor Implementation	16
FilterTool Implementation	18
Software-Based Indirect Sensors.....	21
Overview of SNMP	22
Use within ATM Intrusion Detection System	22
Hardware-Based Sensors	25
Testbed Experiences.....	28
Scaling Considerations.....	30
Applicability to Other Switched Network Technologies	34
Introduction to Multiprotocol Label Switching (MPLS).....	34
Applying SNIDE to MPLS.....	35
Conclusions	38
References	39
Appendix A: Sensor – Assessment Protocol Message Format.....	41
UNI Protocol Events	41
Setup Message Received	41
Connect Message Received	42
PNNI Protocol Events.....	42

New PNNI Node	42
New PNNI Link.....	42
Deleted Horizontal PNNI Link	42
New PNNI Peer Group Leader.....	43
Change PNNI Peer Group Leader	43
SNMP Events.....	43
UNI Signaling Down	43
UNI Signaling Up.....	44
ATM Switch Link Up	44
Appendix B: LDRD Data.....	45
FY00 Annual Report Summary	45
FY01 Annual Report Summary	49
Appendix C: Timeline of Sandia's ATM Forum Impact.....	53

Figures

Figure 1: Example ATM PNNI configuration – pre-attack	8
Figure 2: Peer group leader takeover.....	9
Figure 3: ATM intrusion detection architecture	10
Figure 4: Graphical rule editor.....	13
Figure 5: Active direct, PNNI sensor	15
Figure 6: Passive direct, UNI sensor.....	16
Figure 7: UNI sensor operation	17
Figure 8: Initial FilterTool window	19
Figure 9: Opening a local filter file	19
Figure 10: Filter specification dialog for UNI SETUP message.....	20
Figure 11: Example UNI sensor filter specification.....	21
Figure 12: SNMP sensor	23

Figure 13: Hardware-based sensors in network core	26
Figure 14: Simplified architecture of a hardware-based sensor.....	27
Figure 15: General architecture of a hardware-based sensor	27
Figure 16: ATM intrusion detection demonstration	28
Figure 17: Distributed intrusion detection architecture for ATM networks	31
Figure 18: A representative ATM network topology	33
Figure 19: Label switching example	35
Figure 20: Timeline of Sandia's impact in the ATM Forum	53

Acronyms

AAL – ATM adaptation layer

AE – assessment engine

ASIC – application-specific integrated circuit

ATM – Asynchronous Transfer Mode

CR-LDP – constraint routed LDP

DNS – Domain Name Service

IETF – Internet Engineering Task Force

IP – Internet Protocol

IPC – interprocess communication

FEC – forwarding equivalence class

GUI – graphical user interface

LDP – Label Distribution Protocol

LGN – logical group node

LSP – label switched path

LSR – label switching router

MAC – media access control

MPLS – MultiProtocol Label Switching

NIC – network interface card

OSPF – Open Shortest Path First

PGL – peer group leader

PLD – programmable logic device

PNNI – Private Network-Network Interface

QoS – quality of service

RSVP – Resource Reservation Protocol

RSVP-TE – RSVP with traffic engineering extensions

SNIDE – Switched Network Intrusion DEtection

SN – sensor notification

SNMP – Simple Network Management Protocol

TCL – Tool Command Language

UNI – User-Network Interface

VC – virtual circuit

VCI – virtual channel identifier

VPI – virtual path identifier

Introduction

Asynchronous Transfer Mode (ATM) networks currently form the foundation for many corporate networks, private and public telephone networks, and for much of the Internet backbone (current estimates by the IETF and ATM Forum network standards bodies state that 75% of today's Internet traffic traverses ATM networks). Due to ATM ubiquitous (albeit invisible) deployment, many applications and services that use IP often rely on the correct functioning of an ATM network at some point in the end-to-end path. If the core ATM network were taken down due to a misconfiguration or attack, the effect would be manifested as a network outage, and could be noticed by many users.

Many mechanisms exist today to monitor IP protocol flows to detect IP infrastructure attacks (such as SYN flooding) and precursors to IP attacks (such as ping sweeps). While these mechanisms serve a useful purpose in detecting a class of attacks, they do not address the entire picture of possible network infrastructure attacks. That is, attacks on the ATM control plane protocols (e.g., the virtual circuit establishment and routing protocols) are invisible to today's IP intrusion detection systems. An example of an ATM protocol attack is presented next.

Example ATM Attack – Peer Group Leader Take-Over Attack

The ATM protocol that is responsible for routing virtual circuits is the Private Network to Network Interface (PNNI) protocol. PNNI, which is based on the Open Shortest Path First (OSPF) protocol, is a *link state* protocol. That is, it tracks the network's topological and quality of service (QoS) state to determine how to best route a connection based on its QoS and connectivity requirements. However, the number of state parameters that must be tracked grows very quickly with the size of the network. Therefore, PNNI includes the concept of *summarization*, where ATM switches organize into peer groups, select a peer group leader, and the peer group leader speaks for the peer group when advertising topology and QoS link state information to other peer groups, as shown in Figure 1.

Figure 1: Example ATM PNNI configuration – pre-attack

In this example, switches A.1, A.2, and A.3 organize themselves into a peer group (because they have matching address prefixes), select switch A.1 as the peer group leader (PGL), and appear to the rest of the network as a logical group node (LGN) with address A. Likewise, switches B.1, B.2, and B.3 organize into a peer group, select B.2 as the PGL, and form LGN B. Therefore, in the next level of the hierarchy, two nodes appear – A and B, which reduces the complexity of the network topology and minimizes the required size of the switches' routing tables.

Because the peer group leaders represent the link state within their peer groups, their correct operation is important to the correct routing of virtual circuits in an ATM network. This leads to an example of an ATM protocol attack – the *peer group leader takeover* attack. This is an attack where a rogue node takes over leadership of an ATM PNNI Peer Group. This attack can be executed by an insider penetrating an existing PGL, or by a node outside of the peer group joining the group and getting himself elected PGL, as shown in Figure 2. At that point the corrupt PGL can falsify topology information or adjust QoS parameters and disseminate the modified information to other LGNs, effectively cutting off communication to portions of the network. This type of attack is described by Smith, Hill, and Robinson in [19].

Figure 2: Peer group leader takeover

For demonstrating this attack against the ATM network intrusion detection system, we will focus on the outsider forcing a PGL election. For this attack to be successful, we should detect three events. First, we should see a member join the group. Next, we should detect a PGL election. The third event is the announcement of a new PGL. (This might only be two detectable events – “New member joins group” and “PGL election held” – if the announcement of the PGL election includes the election results.) If these events occur (possibly within a specified time) and the newly elected PGL is the same as the new node who recently joined the peer group, then the PGL take-over attack is assumed to have occurred.

SNIDE Architecture and Components

ATM intrusion detection, and intrusion detection on switched networks in general, presents a unique problem in that a single intrusion detection box is not sufficient to get a complete view of the traffic in a network or subnetwork. The Switched Network Intrusion DEtection (SNIDE) system developed under this project uses a distributed intrusion detection approach, where *sensors* are placed throughout the network (on the edges, in hosts, and in the core) and report ATM control plane protocol events to a central *assessment engine* for correlation and analysis. Sensors are configured to look for specific events that may be related to ATM network protocol attacks. Sensors send events to an assessment engine through an out of band communications channel (to prevent network attack from disabling intrusion detection system). At the assessment engine, the incoming events are correlated and matched against an attack template to determine if a response is required. If so, then it activates a response agent.

The obvious risk with distributed intrusion detection systems is that as the number of sensors per assessment engine increases, at some point the incoming events from the distributed sensors will overwhelm the assessment engine. Therefore, an assessment hierarchy is recommended for large networks, with the lowest-level assessment engines receiving events directly from the sensors (e.g., sensors belonging to the lowest-level ATM peer group), and higher-level assessment engines receiving events from the lower-level AE response agents, as shown in Figure 3.

Figure 3: ATM intrusion detection architecture

The purpose of this project is to develop the SNIDE architecture and components shown in Figure 3, including the assessment engine and sensors. The sensors fall into three categories – *active direct* sensors, *passive direct* sensors, and *indirect* sensors. The active direct sensors directly operate on the ATM protocols, and actively participate in the protocol operations. An example of this sensor is the PNNI sensor, where the sensor implements a portion of the

PNNI protocol, and joins an ATM peer group as if it were a switch. The second sensor is the passive direct sensor, which passively attaches to the network (e.g., using a fiber optic tap), and monitors ATM protocol messages. An example of this sensor is the UNI sensor, which monitors UNI messages, and if a message matches a filter template, it sends an event to assessment. The third sensor is the indirect sensor, which monitors the *effects* of ATM protocol operations. An example of an indirect sensor is the SNMP sensor, which takes switch SNMP traps (e.g., traps that indicate the creation of a new virtual circuit), filters them, and re-formats them into events for the AE.

Sensors may be implemented in software to facilitate wide distribution and handling of complex protocols. These sensors are typically deployed at the edges, where speed is not critical, but large-scale deployment is required. Sensors may also be implemented in hardware to facilitate high-speed filtering of simple protocol messages. These sensors are typically deployed in the core, where speed is critical, but traffic and protocol flows are aggregated, therefore not requiring a large number of sensors. These sensors must be simple; therefore, they typically implement the passive direct sensors (e.g., UNI sensors).]

Assessment Engine

The Assessment engine is the piece of the system that is responsible for receiving incoming messages from sensors, assimilating the information from the tripped sensors, and determining if the messages received constitute an intrusion, based on the rules specified by the system administrator. If it has been determined that a possible intrusion has occurred, the Assessment engine will send a message to one or more response components to perform the action specified in the rule.

The Assessment engine is rule driven, that is, its behavior is determined by specifying a set of text based rules. To provide flexibility, the definition of what constitutes an intrusion is left to the intrusion detection system administrator. A graphical rule editor, described below, is provided to allow the administrator to define what types of intrusions are to be monitored. The rules consist of combinations of sensors with boolean operators and filters. An example of a rule written out could be something like this: Watch for a PNNI New Node Sensor message from one switch AND a PNNI New Node Sensor message from another switch within a 20-second interval. If those events happen, within the time limit, then notify the administrator of the fact with an e-mail message.

Assessment Rules can be made arbitrarily complex by using Operator and Filters. Operators allow creation of logic constructs that combine incoming Sensor Notification (SN) events. When grouped together, they are referred to as solution sets. Each operator defines one logic operation that is satisfied when a complete solution set has been assembled. In the Assessment engine implementation, operators are organized in a tree-like hierarchy to allow complex logic to be specified, and can be nested to an arbitrary depth. Each operator may have one or more child operators and a single parent. The input to any operator is one or more solution sets from its child operators, and the output is one or more solution sets to its parent operator. When an operator "receives" a new solution set from one of its children, it tracks this as a partial solution set for its level. The new partial solution set is combined with existing partial solution sets to determine if new completed solution sets result that can be "sent" to its parent. As solution sets are combined, they are checked to make sure that they don't duplicate any existing sets by comparing the SN's that make up the sets. If a solution set is a duplicate, it is discarded. When an operator that has no parent generates a solution set, the intrusion signature has been satisfied and responses are generated.

The operators implemented in the Assessment engine are:

And - logical combination of two or more child operators; all child operators must be satisfied

Or - logical combination of two or more child operators; any child operator may be satisfied

Sequence – similar to the “and” operator, but sensor notifications from child operators must be in time-order

Count - a single child operator must be satisfied some specified number of times

One or more filters may be associated with each operator to restrict solution sets. Filters check the SN's in the solution set to make sure they have values that, by themselves or in combination with other SN's, are in accordance with the filter parameters. If a solution set does not pass a filter, it is discarded. In order to track the smallest number of solution sets, all filters in the hierarchy are applied any time a partial solution set is generated. This eliminates tracking solution sets at a particular operator level that would fail at some parent operator level.

The filters implemented in the Assessment engine are:

time - all of the sensor notifications in a solution set must occur within the specified delta time

maximum – all of the sensor notifications in a solution set must have a parameter of the specified type whose value is at most the specified value

minimum - all of the sensor notifications in a solution set must have a parameter of the specified type whose value is at least the specified value

select – all of the sensor notifications in a solution set must have a parameter of the specified type whose value is one contained in a specified list of values

same – all of the sensor notifications in a solution set must have a parameter of the specified type whose value matches all other sensor notifications

different - all of the sensor notifications in a solution set must have a parameter of the specified type and the values of the sensor notifications in each child operator must differ from the values in all other child operators

Assessment Process Design

Since rule definitions may be arbitrarily complex, there exists the possibility that the processing of a Sensor Notification through any single rule may take a lot of time. In order to allow other rules to process simultaneously, the Assessment component is split into separate processes. There is an Assessment Control process that is responsible for coordinating the inputs and outputs of one or more Assessment processes. Each Assessment process is responsible for the processing of one or more rules. They receive as input Sensor Notifications from the Assessment Control process, and output Intrusion Notifications to the Assessment Control process. The configuration of how many Assessment processes and which rules each execute is contained in the rules set definition file. The Assessment Control process has responsibility for starting and stopping Assessment processes.

Responses

The assessment engine design incorporates a modular architecture for implementing responses. Each response is implemented as a single module, and is triggered when an attack signature is detected. That is, when a rule is constructed, the operator can also specify zero, one, or more responses that are associated with the rule. Therefore, when the rule is satisfied, messages are sent to all of the specified response components. These messages, which are formatted the same as those in the sensor-assessment protocol, contain name-value pairs that provide parameters to the response modules, allowing them to tailor their responses appropriately. For example, if an SNMP sensor detects the connection of a new switch, it can send a message to the “disable port” response module that contains the IP address and port identifier that must be disabled. Other example responses [24] include: have a modem dial a telephone number to page the network administrator, generate an e-mail to the security officed, or change the sensitivity of a sensor.

Rule Editor

The assessment engine uses a misuse template, or rule file, to determine whether a sequence of events constitutes an attack, and if so, which response should be initiated. The rule file is an ASCII text file which follows a specific structure and syntax. For the casual user, this syntax can be cumbersome. Therefore, a graphical rule editor was developed, which automatically generates syntactically correct rule files.

The rule editor, which is described in more detail in [22], allows a user to specify event sequences in a graphical fashion, as shown in Figure 4. In this example, the rule “pnnidemo” is satisfied if a “PNNI New Node” event is received, AND either a “PNNI New PGL” OR a “PNNI PGL Change” event is received.

Figure 4: Graphical rule editor

The rule editor allows the user to specify filter criteria for the events (e.g., minimum and maximum values for various parameters that accompany event notifications). In addition, a graphical interface is also provided for selecting one or more responses for the rule, and the parameters that are provided in the message to the response module.

Once the rule and responses are specified, the user can check the rule for consistency and save the rule specification in the text-based format that the assessment engine can understand.

Software-Based Direct Sensors

This section describes two direct sensors – the PNNI sensor (active direct sensor) and the UNI sensor (passive direct sensor). Other sensors are also possible, but were not developed in this project. These sensors include sensors that monitor ATM/IP adaptation protocols (e.g., local area network emulation, classical IP over ATM), ATM infrastructure services (e.g., ATM name services), and IP services (e.g., MPLS label distribution, IP routing protocols, DNS).

The architecture of the PNNI sensor is shown in Figure 5. The PNNI protocol module interfaces to the ATM network through a set of ATM interface drivers that are specific to the operating system and network interface card (NIC) on the sensor host. The PNNI module implements a subset of the PNNI routing protocol, and therefore, causes the sensor host to appear to the rest of the ATM network as another ATM switch. (When monitoring the PNNI signaling protocol, the UNI sensor, which is described below, should be used.) PNNI routing-specific events (e.g., reception of PNNI topology state packets, protocol state transitions, etc.) are processed by a TCL script, which filters the events, and re-formats them into the format specified by the sensor – assessment protocol (these protocol messages are described in detail in Appendix A). The re-formatted messages are sent to the assessment engine using the TCL IPC shim, which makes calls to the IPC library, which implements the message passing protocol.

Figure 5: Active direct, PNNI sensor

The UNI sensor represents the class of passive direct sensors in that it directly implements a portion of the ATM UNI protocol and also the PNNI signaling protocol, both of which are based on the International Telecommunications Union's Q.2931 signaling protocol. This sensor observes protocol messages in a passive fashion. The general architecture of the UNI sensor is shown in Figure 6 (where it is attached between two switches to observe Q.2931 flows). At this level of abstraction, the UNI sensor is mostly identical to the PNNI sensor described above in that it implements a protocol processor, event filters, and sensor-assessment protocol handling routines. In fact, all modules except for the UNI protocol processor and event filters re-use the same code that is used for the PNNI sensor.

Figure 6: Passive direct, UNI sensor

However, rather than connecting directly to the network like the PNNI sensor, the UNI sensor attaches to the network using a fiber optic tap, and because the sensor architecture only allows one connection to the network, each UNI sensor can only view half of the duplex UNI protocol.

The implementation of the UNI sensor is described in more detail in the next section.

UNI Sensor Implementation

The operation of the UNI sensor (configured to examine signaling messages on a switch-switch link) is depicted in Figure 7.

Figure 7: UNI sensor operation

On the FilterTool host the FilterTool GUI (described below) creates the filter specification file “default.ff” and pushes it through a secure shell tunnel using the “scp” command to the UNI filter host. The default.ff file is a human-readable (possibly manually generated) ASCII text file. FilterTool allows a user who does not know the filter specification language to generate a default.ff file that is syntactically correct. The FilterTool host then executes the parser on the remote UNI sensor using the “ssh” command. The use of the secure shell utilities permits strong public key authentication to restrict filter loading to authorized users only, and encrypts the transfer and remote execution commands to prevent eavesdropping of sensor management operations.

On the UNI sensor host (a RedHat version 7.1 Linux system, with kernel 2.4.2), the parser (implemented in flex and bison) separates the default.ff filter specification into two sub-specifications – a hardware specification (in hw.spec) and a software specification (in sw.spec). The filter itself is split into two components – a hardware component and a software component. For the purposes of this project, the hardware component is simulated in software (a design for an actual hardware filter is described later). When the simulated hardware filter receives the UNI message, it reads the hw.spec file. If the message passes the filter, then it is forwarded to the software filter. The software filter reads sw.spec, and if the message passes this second filter, an event is constructed and sent to the assessment engine. If new hw.spec or sw.spec files are generated, then their rules take effect on receipt of the next message.

The UNI sensor is composed of a TCL script, which implements the main functionality, and a series of C files that implement the “back end” processing functions. The q2931.tcl script performs the following functions:

1. calls the appropriate C routine to get a message that passes all of the filters (hardware and software)

2. reformats the returned message into the appropriate format for the sensor – assessment protocol
3. sends the message to the assessment engine

The functions in the C files (q2931.c, hw.c, and aal5.c) perform the following operations:

1. connects the C routine GetMsg() to the TCL command “q2931_getmsg”
2. initializes the ATM interface to receive signaling messages from the fiber optic tap
3. reads AAL 5 message from the signaling VC
4. performs the “hardware filter” function (currently based on the UNI message type) using the hw.spec file for the filter rules
5. performs the “software filter” function using the sw.spec file for the filter rules
6. decodes the resulting message
7. passes decoded result to calling TCL script (q2931.tcl)

FilterTool Implementation

FilterTool is a program designed to help users create/modify filter rules for UNI messages. FilterTool also lets the user load these filter rules to a remote system (using ssh) that does the actual filtering. The remote system will communicate the filter rules to the hardware sensor (hardware filtering may be simulated or not implemented) that will do a limited amount of filtering. Whatever passes the hardware sensor filters will be forwarded on to the remote host that will do the rest of the filtering in software. FilterTool supports filtering for four types of UNI messages: SETUP, CONNECT, RELEASE, and RELEASE COMPLETE. These were chosen because they are common UNI messages and could adequately demonstrate our UNI filters work properly.

FilterTool is written in C++ and uses Qt 3.0 beta 2 (www.trolltech.com) widgets for the user interface. It was written and tested on the RedHat Linux 7.1 operating system and should work on most Linux environments (not tested). Before FilterTool is started, the user needs to be authenticated with ssh. This is done by opening a terminal window and typing “ssh-agent -s xterm” at the prompt. Then in the new window, type “ssh-add” and launch FilterTool. Downloading and uploading filters uses scp to transfer files which requires the process mentioned above. If the user just wants to create, alter, or view filters that are on their own machine, ssh is not necessary. Using ssh and scp allows the data to be transferred securely. When the program is first started, all fields are empty as shown in Figure 8.

Figure 8: Initial FilterTool window

The user can proceed by either downloading the filter rules from a UNI sensor, opening a local filter file (with a .ff extension as shown in Figure 9), or start from scratch by adding new filters.

Figure 9: Opening a local filter file

In each case, filter rules will appear in the large list box in the center of the window. Each individual filter on the screen can be selected by clicking on the item. If the user wants to edit the selected item, they can either double-click that item or click the “Edit Filter” button. To remove a filter rule, the user can select the rule and press the “Remove Filter” button. When the “Add Filter” button is pressed, a dialog box appears allowing the user to select one of the four UNI message types supported: SETUP, CONNECT, RELEASE, and RELEASE

COMPLETE. The user will select one of these and a different dialog box will appear depending on the message type selected. For example, if the user chooses SETUP, a large dialog box appears that allows the user to specify the called party number, calling party number, ATM traffic descriptors, and AAL parameters (SETUP dialog box is shown in Figure 10).

Filter Specification

Fill in the fields to filter in a specific message. All of the fields need not be specified. Leave all fields blank to filter in all Setup messages.

Called Party Number:

Type: ATM End System Address

Value (hex string):

Mask (0-100):

Calling Party Number:

Type: ATM End System Address

Value (hex string):

Mask (0-100):

ATM Traffic Descriptor:

Peak Cell Rate (downward or backward): Greater than or equal to

Sustained Cell Rate (downward or backward): Greater than or equal to

Maximum Burst Size (forward or backward): Greater than or equal to

AAL Parameters:

☐ AAL 1 ☐ AAL 2 ☐ AAL 5 ☐ Any

Cancel OK

Figure 10: Filter specification dialog for UNI SETUP message

Alternatively, if they choose RELEASE, a small dialog box will appear that allows the user to select which cause or causes they want to filter.

Filter rules are actually just strings written in a specific order so that they can be parsed by the UNI sensor. By default, all messages will be filtered in (*filtered in* means that the message is kept and forwarded on to the assessment engine, *filtered out* means that it is thrown away). If the user wants all SETUP messages to be filtered in, they just need to make sure there are no SETUP rules or that the only SETUP rule on the screen is simply "SETUP". If a SETUP or CONNECT dialog is opened, and then accepted, without any fields being entered, the rule "SETUP" or "CONNECT" will be added to the screen showing that it will filter in all of that message type. For RELEASE and RELEASE COMPLETE, the "Filter in all causes" option should be added to filter in all of those message types. This is not necessary, though, because as long as there are no filter rules of that message type, all of those messages will be filtered in.

To filter out messages, the user needs to decide what messages they want to filter in. For example, if the user only wants to see RELEASE messages that are caused by “user busy”, they should press “Add Filter” and then choose RELEASE. In the Release Filter dialog box they can choose “user busy” and add that cause to the list. When the user presses the “OK” button, the string “release cause(val=5)” will be displayed in the main window. This means that all RELEASE messages that do not have the cause “user busy” will be filtered out.

After this is done to each message type (as needed) the user can save the filter specification (an example of which is shown in Figure 11) to a file. If this is what they want to configure the filter with, they can choose to upload it to a remote host. The user must specify the IP address/host name and the path to the directory of where the parser resides. The button “Upload Filters” is pressed and the user can choose the .ff file they want to upload. FilterTool will send this file to the remote UNI sensor and will run the parser on the file (thus configuring the sensor’s software/hardware filters). If there are any errors, a message will be displayed at the bottom of the screen saying to check the console for details. To verify that the correct file has been uploaded, the user can then push the “Download Filters” button and FilterTool will display all of the filters that are loaded on the remote host.

Figure 11: Example UNI sensor filter specification

Installation

This program requires Qt to run. To get Qt (it's free) go to www.trolltech.com and navigate to "Downloads". Then select "Qt/X11 Free Edition". Download the file and follow the instructions bundled with the download. After Qt is installed properly, FilterTool should be able to run. If there are any problems, go in to FilterTool's directory and enter "make clean" in the console. Then re-make the program by typing "make".

Software-Based Indirect Sensors

The SNMP Sensor is a series of Tcl scripts that interface SNMP operations and commands to the assessment engine. The SNMP Sensor is considered a supplementary element of the ATM Intrusion Detection effort and is not the primary focus of the project. However, SNMP

capabilities allow the inclusion of additional event information and collaborating data, and the ability to initiate responses on an ATM switch in the network. This section provides a brief overview of SNMP, discusses its use in the ATM intrusion detection context, describes the basic functionality of the implementation, and proposes how SNMP strengthens the extensibility and flexibility of the ATM intrusion detection approach.

Overview of SNMP

Simple Network Management Protocol (SNMP) is essentially a specialized request/reply protocol designed for the management of network devices. It supports client-server operations and allows a system administrator to monitor the status of various device parameters as well as perform configuration management from remote locations. A key component of the SNMP approach is the management information base (MIB), which specifies the pieces of information that can be retrieved from a network node. The syntax utilized for data storage by the MIB is the Abstract Syntax Notation One (ASN.1). In the request process, the SNMP client places the ASN.1 identifier for the desired MIB variable into the request message, and sends the message to the server. The server, upon receiving the request, maps this identifier into local variable, retrieves the current value stored in the variable, encodes that value using the ASN.1 syntax, and sends the response to the client. The two types of messages involved are an SNMP request and an SNMP reply. An unsolicited message, i.e. an SNMP trap, involves the SNMP server sending a message in the same format as previously described to a designated client(s), but without any initial request by the client(s). A more detailed explanation of SNMP operations and the key components can be found in "SNMP, SNMPv2, and RMON: Practical Network Management" by William Stallings [20]. It should be noted that SNMP was designed for the TCP/IP environment, and relies on IP as the data transmission protocol.

Use within ATM Intrusion Detection System

Within the ATM Intrusion Detection system, the SNMP agent on an ATM switch provides a secondary or indirect sensor for data collection, and allows the assessment engine to issue a response to the ATM switch. The type of trap events sent by the SNMP agent varies from statistical traffic data to basic connectivity establishment. Although the SNMP agent has a wide range of trap responses available, the only those directly relevant to an intrusion attempt received consideration by the SNMP Sensor.

The SNMP response capabilities also vary, but were limited to administratively disabling a port for this prototype. Use of responses must be carefully controlled so as not to allow the intrusion detection process to inadvertently shut down the network. Careful management of the response decision process reduces the potential for exploitation by an adversary to compromise the operation of the network.

Basic Functionality of SNMP Sensor

A series of Tcl scripts that interface SNMP operations and commands to the Assessment Engine make up the SNMP sensor. The scripts developed in the project only demonstrate the basic functionally possible with SNMP operations, and are not intended to serve as a production level application. Figure 12 indicates the basic components of the SNMP Sensor, and description of the various components and the necessary interactions follows.

Figure 12: SNMP sensor

General operation and components of the SNMP Sensor are as follows. The first component, the SNMP agent or code residing on the ATM switch, generates the SNMP trap messages (1), and sends those messages to the SNMP trap daemon on a Linux host, (2). Tcl scripts filter the incoming traps, and reformat the trap messages (3), destined for the Assessment Engine (4). The incoming messages are processed with data from the direct sensors at the Assessment Engine. If a response is necessary, the Assessment Engine issues a response to the SNMP sensor via the Linux host. The response message is formatted in the appropriate ASN.1 notation (5), and sent to the SNMP agent on the ATM switch (6). Initialization of the SNMP Sensor occurs by 1) running the SNMP daemon, and 2) running the `snmpitf.tcl` script. Creation or initialization of all other files and scripts occurs automatically, assuming accurate configuration parameter and path settings. More detailed descriptions of the individual blocks and the Tcl scripts, `trapitf.tcl`, `snmpitf.tcl`, and `snmpcmditf.tcl`, follow below.

The trap message (1) format from the ATM switch adheres to ASN.1 notation, and includes a specific identifier associated with a particular condition on the switch. A physical link connection, the signaling status on a port, etc., may have changed state, which causes the issuing of a trap message. Before forwarding to the Assessment Engine, the Linux host filters the generated SNMP trap messages for relevancy and reformats the data.

To communicate with the Linux host, the SNMP daemon (2) must be running on the Linux host, and the IP address of the Linux host must be set as the destination address for trap messages generated on the ATM switch. The trap daemon must be configured to call the `trapitf.tcl` script, described in the next section. This prototype utilized the SNMP daemon version 4.1.2 from University of California at Davis¹, with the Red Hat Linux operating system, kernel version 2.2.10². To initiate the processing of trap messages on the Linux host

¹ <http://www.redhat.com/swr/i386/ucd-snmp-4.1.2-8.i386.html>

² <http://www.redhat.com>

(transition from block (2) to (3)), the SNMP trap daemon calls the trapitf.tcl. The SNMP trap daemon must be configured to call the trapitf.tcl, accomplished through parametric settings in the snmptrapd.conf file. The snmptrapd.conf directs the SNMP trap daemon to execute the trapitf.tcl upon receiving trap messages from an ATM switch and to pass all available variables. MAN pages that describe how to set parameters in the snmptrapd.conf files are included with the trap daemon code. The snmptrapd.conf file should reside in the appropriate directory specified in the MAN pages (see *man snmptrapd.conf*). The directory path to the trapitf.tcl script must be specified in the snmptrapd.conf in order to execute the trapitf.tcl script when an SNMP trap is received.

The trapitf.tcl script inputs variables from the SNMP trap daemon and passes relevant information to the snmpitf.tcl script. Input variables include IP addresses, trap identifiers, and other associated variables. Some initial processing occurs in the trapitf.tcl script, with the prototype processing being specific to a Marconi (FORE) ATM Switch. One aspect of the processing consists of trimming leading identifiers in the Object Identifier (OID) information from particular trap variables so that the minimum amount of unique information is stored and forwarded. For example, the following portion of the OID string is removed from the second variable in all trap messages captured in the prototype.

```
.iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjects.snmpTrap.  
snmpTrapOID.0 enterprises.326.2.2.
```

The indicated string is unique to Marconi ATM switches. Finally, the trapitf.tcl selects and orders the trap variables, and places them in a buffer file in order to pass them to the snmpitf.tcl script.

The snmpitf.tcl script (block (3)) provides the decoding and reformatting of the trap messages placed in a buffer file by the trapitf.tcl script. The current version handles only Marconi (FORE) specific traps. A table of these traps is located in Appendix A of the *ATM Switch Configuration Manual*, and the FORE mibs are available at the Marconi website³. General operation of the snmpitf.tcl script consists of periodically searching for the buffer file containing trap information generated by the trapitf.tcl script. If the file exists, the data is input and the trap identifier is converted to a text string. The text strings are searched for the desired traps, and upon finding one, a reformatted trap message is sent to the assessment engine.

Reformatting of trap messages allows interfacing to the Assessment Engine and further refinement in the selection as to which variables present useful information for the intrusion detection process. A typical reformatted message takes the form:

```
TYPE=SENSOR_NOTIFICATION:AGENT_TYPE=SNMP_TRAP_1.0:  
SN_TYPE=<trap_id>:IP_ADDRESS=<X.X.X.X>:  
SW_PORT=<sw port number>:DIAG<diagnostic data>
```

Parameters selected from the original trap message include:

³ <http://www.marconi.com/html/login/tacticsonline.htm>

<i>trap_id</i>	SNMP trap identifier
<i>X.X.X.X</i>	IP address of SNMP agent on the ATM switch
<i>sw port number</i>	Software port number on the ATM switch
<i>diagnostic data</i>	Other relevant diagnostic data contained in the original trap

Once a trap message is selected and reformatted, it is sent to the Assessment Engine (4), indicating the completion of the SNMP Sensor processing of a SNMP trap. The prototype version of the `snmpitf.tcl` script forwards three types of traps, UNI signaling down, UNI signaling up, and ATM Switch Link up. Other traps may be included if they provide meaningful data in determining an intrusion attempt. Additionally, the prototype utilized a searching frequency of 5 seconds, and flushed the buffer file upon every read by the `snmpitf.tcl` script. This prevents rereading outdated trap messages.

If the Assessment Engine needs to issue a configuration response, the SNMP daemon is notified on the Linux host, and utilizes the `snmpset` command to send appropriate control messages to an ATM switch. Responses from the Assessment Engine must also take the ASN.1 form to interface to the SNMP agent on the target switch. The Assessment Engine passes the appropriate IP address, ATM switch software port number, and community string to the Linux host, which generates the appropriate `set` command with the proper OID (6) using the `snmpcmditf.tcl` script (5). The appropriate community string must be included with the `set` command in order to communicate with the SNMP agent on the designated ATM switch. A typical response might be to administratively shutdown a port on an ATM switch.

All of the SNMP Sensor Tcl script files can be located in the same directory. The buffer file created by the `trapitf.tcl` can be created in that directory as well, and the `snmpitf.tcl` should specify the same location for reading the buffer file. The Tcl script creation in the SNMP Sensor development relied heavily on the text, "Tcl and the Tk Toolkit," by John K. Ousterhout [15]. Note that the method of retrieving and transmitting the community strings necessary for a configuration response by the Assessment Engine is not an inherently secure approach. Security considerations for managing this information would be necessary for any production level installation that included response capabilities.

Extendibility and Flexibility

The SNMP Sensor utilized in the prototype only focused on the proof-of-concept. Extensions to the sensor could include the processing of an increased number of trap messages, and the support of additional Assessment Engine request messages. Also, while `snmpget` commands were not integrated in the prototype, that functionality could be included. This would allow the Assessment Engine to query for additional information beyond what is initially forwarded in the SNMP trap messages.

Since the SNMP Sensor relies on the IP/TCP suite of protocols, the application to switched IP based networks occurs naturally. Substantial modification of the sensor for use in Gigabit Ethernet and MPLS based networks would not be necessary.

Hardware-Based Sensors

Sensors implemented in software are quite suitable for use at the edges of an ATM network, where high-speed performance is not required and due to the quantity of sensors deployed, low cost implementations are desirable. Because of performance scaling issues, hardware-

based sensors are the appropriate choice for monitoring control plane protocol messages in the core of the network. Hardware sensors are also appropriate for real-time monitoring of ATM user plane (data) and operations and management cells (which may also be used for certain classes of attacks). Because hardware-based sensors scale to much higher speeds, and although they are more expensive than software implementations, only a few fast sensors are needed on the core trunks. This is illustrated in Figure 13.

Figure 13: Hardware-based sensors in network core

As shown above, hardware sensors are intended for use between several switches, or between switches and user nodes to examine signaling and routing messages for anomalies. These anomalies are then reported for analysis and assessment.

Hardware sensors are essentially high-speed cell or packet filters that operate at the signaling or line rates, typically OC-3 (155 Mbps) or OC-48 (2.488 Gbps). They function in the passive direct [22] fashion, siphoning off a copy of the bit stream and examining it for certain bit patterns, such as a specific message type on a certain virtual circuit. When a hardware sensor finds a cell (or having performed reassembly, a packet) that matches the specified criteria, it will generate a message for transmission to another system. This message may go directly to an assessment engine or it may go to a system hosting multiple hardware sensors, for further processing, formatting, and eventual transmission to an assessment engine. A simplified architecture for this is shown in Figure 14. In the simplified architecture, the VPI and VCI would be hard-coded, while the message type would be passed to the sensor as a parameter. The general architecture for a hardware sensor is shown in Figure 15 and can receive a full filter specification.

Figure 14: Simplified architecture of a hardware-based sensor

Figure 15: General architecture of a hardware-based sensor

Hardware-based sensors can be implemented with application-specific integrated circuits (ASICs) or with programmable logic devices (PLDs). Each of these has certain advantages. ASICs have a security advantage in that once the design is completed and fabricated, it cannot be changed, infected or corrupted. On the other hand PLDs, being reconfigurable logic, can be modified in real time. This allows not just for changing the sensor to look for a different value in a parameter (by writing a different value into its memory), but to load new logic into the sensor, completely changing its function, in response to an attack or event. In all but extremely large quantities, PLDs cost less than ASICs containing logic of equivalent complexity.

The use of PLDs on a common or standardized circuit board also facilitates the development of new sensors and the upgrading of older ones as the network threat environment changes. Furthermore, a PLD implementation would facilitate incorporation of hardware filtering functions in existing vendors' switches, where similar filtering operations are already performed in programmable logic.

Testbed Experiences

Halfway through the project, a demonstration of the prototype capability was given that showed the use of a PNNI sensor, assessment engine, and responses. In the demonstration scenario (shown in Figure 16), an ATM switch was configured to represent an ATM network, and a PNNI sensor was attached to this switch to monitor PNNI activity for the lowest-level peer group. The sensor was also configured to send event notifications to an assessment engine over an “out of band” IP/Ethernet network.

Figure 16: ATM intrusion detection demonstration

To simulate a peer group leader takeover attack, a “rogue switch” was configured with a high peer group leader priority to ensure that when the switch is attached to the “ATM network”, it would force a peer group leader election and win the election. The PNNI sensor was configured to send “PNNI new node” and “PNNI peer group leader change” events to the assessment engine when these PNNI events are detected. The assessment engine was configured with a rule file (developed using the graphical rule editor tool) that looked for a new node followed by a PGL change event, and if this rule was satisfied, it would send an email response to the system administrator and also issue an SNMP command that would disable the port on the switch to which the rogue switch attached.

When the demonstration was performed, the system worked as planned.

Of course, the “port disable” response was a drastic response, and during development of the demonstration system, normal network events that happened to match the attack rules configured in the assessment engine would result in network reconfiguration and disabled switch ports. On several occasions, the group was surprised to find switch ports that were turned off unexpectedly as a result of normal system functioning. Therefore, in an operational situation, the rule file would need to be specified in more detail to filter out events that are part of the normal function of the ATM network. Alternatively, less drastic responses (e.g., email alerts) may also be required.

In addition, the demonstration system was installed on ATDnet in Washington, DC. ATDnet is a large network comprised of at least 5 organizations. When the system was installed, the PNNI sensor was configured as before – to detect PNNI new node and PGL change events. However, when the PNNI sensor was brought on-line on ATDnet, it would report PNNI new node events for every switch in ATDnet (because, to the sensor, every switch was a *new* switch). These events were generated very rapidly, and were reported to the assessment engine. Nevertheless, despite the fact that up to 50 new node events were reported in a brief time span, the assessment engine was able to queue the event notifications and process them without any evidence of missed events or other ill effects. Therefore, the scalability of the assessment engine for medium-sized networks was verified.

Scaling Considerations

This section describes ways to scale the SNIDE architecture described earlier to large networks, while providing rapid response. The issue of intrusion detection and mitigation in ATM networks poses an important and unique challenge to which Sandia National Laboratories [22] has committed itself. Like any networking architecture, ATM networks consist of a number of ATM nodes distributed over a wide geographical area. While the sheer number of nodes and the non-trivial inter-node distances pose a challenge to centralized decision-making, with respect to detecting and mitigating intrusion, the relative high speed at which cells are transported across an ATM network complicates the problem immensely. Tarman, Witzke, Bauer, Kellogg, and Young [22] describe an architecture where PNNI and UNI sensors are deployed at the edges of a network to enable an assessment engine detect attacks, by matching anomalous network behavior against standard templates gathered from known attacks, and then deploying effective countermeasures via the response agents. While the paper correctly observes that, for a small-scale ATM network, it is adequate to execute the sensors, assessment engines, and response agents in software, it motivates the development of a practical, scalable intrusion detection architecture for large-scale ATM networks.

To address the problem comprehensively, the causes and origins of the key challenges are enumerated as follows. First, intrusion attacks directed against the PNNI protocol including the call admission control (CAC) and operation and management (OAM) functions, generally manifest at speeds of milliseconds to seconds which, for small-scale ATM networks, may be adequately addressed in software. The precise definition of what constitutes a small-scale ATM network, i.e. the maximum number of nodes for which the above claim is true, requires experimental analysis and is under study at Stevens Institute of Technology in collaboration with Sandia. Second, for modest- to large-scale ATM networks, organized in the form of a collection of peer groups, the combined weight of the decision-making for the increased number of ATM nodes, may render a straightforward software solution lacking. Third, given that the processes in the ATM switch fabric operate at microseconds or faster, intrusion attacks directed at ATM cell transport may defy a pure software solution. While a hardware dominated approach may constitute a logical response to issues 2 and 3, this research is driven by the following concern. As ATM switching speeds continue to increase in the future, even a pure hardware solution may not be adequate. The aim here is to pursue a new thinking, a novel architecture for ATM intrusion architecture.

In developing the new architecture, following constitute the key issues and guiding principles. First, clearly, the architecture cannot be centralized. Second, while it is logical to pursue a purely decentralized [12][13] approach, i.e. one where the overall task of intrusion detection and mitigation is distributed uniformly among all constituent ATM nodes, such an approach may not correspond exactly to the hierarchically distributed organization of the ATM network. Third, recall that an ATM network consists of a very fast cell switching process at the switch fabric level, and a relatively slower process, that includes CAC, OAM, signaling, etc. Fourth, in general, detecting and mitigating an intrusion at the switch fabric level would require an ultra fast response from the architecture while for attacks at the CAC, OAM, and signaling level, the architecture may be permitted to generate a relatively slower response. Fifth, according to the principles of computational intelligence, to generate an ultra fast response, the associated decision-making must be simple, i.e. the number of decisions must be small and the complexity of each decision, low. Using similar reasoning, when a slower response is tolerable, the complexity of the decision-making process, may be high. Sixth and final, although the degree of complexity, associated with conceiving and launching an intrusion, either at the switch fabric or CAC level, may range anywhere from low to high,

any manifestation of the intrusion at the switch fabric level must be addressed by the architecture very quickly while that at the CAC level may be addressed relatively slowly. An architecture that emerges from careful consideration of the six issues is presented in Figure 17 and explained as follows.

Figure 17: Distributed intrusion detection architecture for ATM networks

Figure 17 presents an ATM network consisting of two peer groups – A and B, with ATM nodes A1 through A4 and B1 through B4 respectively. The connectivity between the nodes of A and B, is not shown here. Associated with each node, say A2, is a tactical decision-making unit, A2-T, termed tactical ATM sentinel, that is realized primarily in hardware and whose function is six-fold. First, its sensors continuously monitor the behavior of the switch fabric of A2 and it responds quickly to any anomaly. Second, A2-T's response is limited to the ports and architectural elements of A2 and user traffic connected to the UNI of A2. Third, anomalous behavior of the call processor of A2 are reported to the strategic ATM sentinel for the entire peer group A. Fourth, a copy of A2-T's response is propagated to the strategic ATM sentinel. Fifth, the strategic ATM sentinel's response to anomaly in the call processor, is propagated to A2-T where it is executed. Sixth, where necessary, the behavior of A2-T is amenable to reprogramming by the strategic ATM sentinel. The strategic ATM sentinel is a

hardware/software entity, distinct from the nodes of the peer group, whose function is to analyze every anomaly occurring within the peer group, its own past response to earlier intrusions, and compute long-term decisions, a subset of which is then employed to reprogram the tactical ATM sentinels. The level 1 strategic ATM sentinels of peer groups A and B, in turn, report their activities to the level 2 strategic ATM sentinel which takes an even broader and longer-term view of the anomalies and computes responses that are manifest through modifying the response behaviors of level 1 strategic ATM sentinels.

Analysis reveals that, in real-world distributed systems, a true picture of a system-wide behavior does not reside in any one of the constituent entities. A higher-level entity, such as a strategic ATM sentinel, must first acquire data from the entities, clearly a slow process, and only then determine the big picture through intelligent computation.

Thus, the underlying philosophy of the intrusion detection architecture may be described as follows. While tactical sentinels are deployed to respond to low level intrusions on fast ATM subprocesses in a timely manner, these sentinels' behaviors are governed by the strategic ATM sentinels that examine anomalous network behavior over a wide time scale and compute long-term decisions. In essence, the architecture is hierarchically distributed, scalable, and practical.

For a comprehensive study, we plan to identify intrusions/attacks that are already known and ones that are conceptually feasible. As a first step, in addition to the types of intrusions enumerated in [22], the issues of detecting and mitigating the following attacks/intrusions will be addressed in this effort.

- Excessive, intensive traffic from one or more users at the UNI, exceeding the negotiated contract. A user's traffic may either exceed the negotiated SCR (sustained cell rate) or PCR (peak cell rate). If the UNI were to tolerate minor violations of SCR and PCR, under specific scenarios, their impact at the switch level could be far worse, especially where the bandwidths are high.
- Too many open connections at a node, with either little or normal traffic, may constitute an attack.
- Analysis shows that a few calls with excessive high bandwidth traffic may cause serious QoS problems and may be exploited as a surreptitious attack.
- A specific attack had been developed in which, if a call is already established between any two nodes in the network via an intermediate node, already under an attacker's control, and if the source and destination nodes are on the attacker list of "nodes to be attacked," then all of the traffic cells arriving on the correct channel are redirected to an arbitrarily and periodically selected channel, also already established. Following extensive study of the attack through simulation, it appears that the switch fabrics of one or more of the ATM nodes in the network may encounter incoming cells with VPI/VCI pairs for which there is no corresponding entry in the routing table. The problem can affect both traffic as well as control cells.

Our proposed modeling and simulation based investigation will employ a representative ATM network topology, shown in Figure 18.

Applicability to Other Switched Network Technologies

At the inception of this project, ATM enjoyed widespread deployment and vendor support. However, as the project progressed, ATM sales declined while vendor and user interest in IP switching technologies increased. One standard for IP switching is MultiProtocol Label Switching, or MPLS. As the market realities became apparent, the project team decided to examine the ATM intrusion detection developments in light of IP switching, and the results of this study are documented in [26].

As described earlier in this report, the general architecture that was adopted for ATM intrusion detection can be readily viewed as an intrusion detection architecture for switched networks in general. That is, an architecture for switched network intrusion detection would necessarily be distributed, with a single assessment engine for the entire network, or a hierarchy of assessment engines performing strategic and tactical assessment, as described in the previous section. The main differences in the architecture are in the details related to the MPLS protocols for routing and label distribution. That is, protocol processors in the sensors would need to be modified, the assessment rules would need to be modified, and the sensor-assessment protocol would need to be modified.

Introduction to Multiprotocol Label Switching (MPLS)

Multi-Protocol Label Switching (MPLS) is a method for controlling the flow of IP packets (datagrams) and quickly switching them through a network. When standard routers forward IP datagrams, they make routing decisions on a per-packet basis using most of the contents provided by the IP header. However, datagram switching is much faster than routing because it uses a path setup protocol to establish a path through the network. Once the path is established, a small flow identifier is added to the packet, which allows datagram forwarding by switching hardware rather than route lookups.

MPLS borrows many concepts from ATM, including the ideas of path switching, dynamic flow setup, and support for traffic engineering. The following sections describe these architectural concepts in more detail.

MPLS is a set of Internet specifications that describes interoperable mechanisms for implementing IP switching. An example of using MPLS for IP switching is shown in Figure 19. MPLS Label Switching Routers (LSRs) switch IP datagrams over a Label Switched Path (LSP) that traverses an MPLS Domain. The LSP is established manually via management access or via signaling.

Figure 19: Label switching example

Switching on an LSP is supported by a label affixed to the IP packet by an MPLS ingress LSR (LSR1 in Figure 19), and the label is removed by the egress LSR (LSR4 in Figure 19) when the datagram leaves the MPLS domain. Whereas conventional IP routing looks at the IP header to make forwarding decisions, MPLS forwarding decisions are based on a small label affixed to the packet in a “shim header” between the IP and the link layer protocol (e.g., Ethernet MAC) headers. This label is assigned to the packet’s Forwarding Equivalence Class (FEC), which in practice is a network prefix. Therefore, all packets belonging to the same FEC (i.e., having the same destination address prefix) are assigned the same label, and are switched along the same LSP. Labels may be unique to the LSR interface (interface label space), or unique to all interfaces on the LSR (platform label space).

LSPs may be configured manually or via signaling, and like ATM virtual circuits or virtual paths, LSPs allow traffic engineering. This permits a network operator to override the automatic path that a packet would take (i.e., one determined by normal IP routing), and send the packet through one that is configured via management. Standard IP routed networks also support traffic engineering, but it is based on source routing, which has security and efficiency issues. Therefore, MPLS is considered a better solution for engineering traffic flows in a backbone network.

When LSPs are configured via signaling, a label distribution protocol is required to allow LSRs to inform other LSRs of bindings between FECs and labels. The binding between an FEC and a label is normally determined by the downstream LSR and communicated to the upstream LSR via signaling. Once communicated, the signaling protocol is used to maintain the binding and tear it down when it is no longer needed. As described later in this paper, multiple protocols for label distribution are defined by the Internet Engineering Task Force, including the Label Distribution Protocol with extensions for Constraint Routing (CR-LDP) and the Resource Reservation Protocol with extensions for Traffic Engineering (RSVP-TE).

Applying SNIDE to MPLS

The Switched Network Intrusion DEtection system (SNIDE) was originally developed to monitor protocol flows in ATM networks to detect ATM infrastructure attacks. The SNIDE architecture, which is shown in Figure 3, uses various types of sensors that report to a

central assessment facility via IP. These sensors are the passive-direct, active-direct, and indirect sensors, which have been described earlier.

As described earlier in this section, there are many architectural similarities between MPLS and ATM. Both protocols can be described in terms of the three-plane protocol reference model, which consists of the user plane, control plane, and management plane. With both approaches, user plane data is switched along a virtual path using a small amount of switching information contained in the packet header. In order to switch data along this path, the tables in the switches must be configured *a priori* to understand how to handle data containing specific flow identification values in the headers.

The role of the ATM and MPLS control planes is to dynamically configure switch tables to configure a new virtual path when it is needed. While there are vast differences in the details between the ATM and MPLS control plane protocols, on a more general level, these protocols are similar. Whereas ATM uses the UNI and PNNI signaling protocols to signal a new virtual circuit, MPLS uses CR-LDP or RSVP-TE to establish an LSP. Likewise, ATM uses the PNNI routing protocol to track current network state to make routing decisions, where IP routers and LSRs use one of a number of routing protocols (including Open Shortest Path First, from which PNNI was derived) to perform the same function.

These similarities between MPLS and ATM allow the SNIDE system to be applied to MPLS networks with little modification to the assessment engine and the sensor-to-assessment communications libraries. Since the assessment engine was designed to be very flexible, it can be easily enhanced with new MPLS-specific message types and new rules that describe MPLS attacks (including those described earlier).

However, the sensors, which are concerned with the protocol details, need a number of modifications to support MPLS. These modifications are:

- Passive-direct sensors – these sensors will be placed on inter-LSR links, LSRs themselves, or on boarder routers, and will implement event monitoring of the MPLS label distribution protocols. The existing sensors will need to be modified to decode fields in protocol messages and track protocol state. Also, unlike the UNI sensors, the MPLS sensors will need to be configurable to support multiple protocols (CR-LDP, RSVP-TE, and BGP messages carrying label information).
- Active-direct sensors – these sensors will be placed within an MPLS domain, and will implement the IP routing protocols and participate actively in protocol exchanges with the LSRs or routers in the domain. As with the passive-direct sensors, the active-direct sensors will need to decode messages and support procedures for multiple protocols (e.g., BGP and OSPF).
- Indirect sensors – these sensors translate between SNMP and the SNIDE reporting protocol. Since IP LSRs use SNMP for configuration and trap reporting, these sensors will require little modification. The only modifications required would be to translate the additional, MPLS-specific traps that may be reported.

Attack rules in the assessment engine will need to be developed to identify sensor event sequences that may indicate an MPLS attack. Some possible attack rules are defined in [26].

As one can see, the fact that SNIDE was designed for intrusion detection in switched networks allows it to be readily applied to detect misuse of the MPLS protocols. The bulk of

the development that is required to perform modification is largely limited to the sensors themselves, which are concerned with the details of the protocols they are monitoring.

Conclusions

Asynchronous Transfer Mode (ATM) is widely used as a backbone for corporate and public telephone and data networks, and its availability is paramount to the correct operation of the Internet Protocol (IP) and higher-layer applications. Key to the correct operation of ATM is its set of control plane protocols, which perform functions such as virtual circuit signaling and routing. If these protocols are attacked, then denial of service can result. Therefore, a system is required to monitor these control plane protocols, and respond if protocol events indicate a possible attack.

ATM is a switched network technology, and other protocols (notably IP) are also being implemented in a switched fashion. Therefore, the general problem of intrusion detection in a switched network is receiving more attention. This paper describes in detail the Switched Network Intrusion DEtection (SNIDE) system that was developed under a Sandia Laboratory-Directed Research and Development project. This system was designed to perform control plane intrusion detection on a general switched network, and implemented to perform intrusion detection specifically on ATM networks. This system includes components that implement intrusion sensing, assessment, and response, along with management tools and graphical user interfaces to configure system components.

Initial results have shown that this system operates as expected, and vendors have expressed interest in possible licensing of components from the SNIDE system. Specifically, we have talked with vendors about the UNI sensor and assessment software, and the possible implementation of sensor functions in ATM switches.

References

1. Edward Amoroso, *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*, Intrusion.Net Books, Sparta, NJ, 1999.
2. ATM Forum Technical Committee, *ATM Security Specification Version 1.1*, af-sec-0100.002, March, 2001.
3. ATM Forum Technical Committee, *Private Network-Network Interface Version 1.0*, af-pnni-0055.000, The ATM Forum, Mountain View, CA, 1996.
4. ATM Forum Technical Committee, *User to Network Signaling Specification Version 4.0*, af-sig-0061.000, The ATM Forum, Mountain View, CA, 1996.
5. Rebecca Gurley Bace, *Intrusion Detection*, Macmillan Technical Publishing, Indianapolis, IN, 2000.
6. Tim Bass, "Intrusion Detection Systems and Multisensor Data Fusion," in *Communications of the ACM*, vol. 43, no. 4, pp. 99-105, 2000.
7. R. Bettati, W. Zhao, and D. Teodor, "Real-time Intrusion Detection and Suppression in ATM Networks," in *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, held in Santa Clara, CA, April 9-12, 1999, USENIX, 1999.
8. Daniel Bilar and Daniel Burroughs, "Introduction to State-of-the-Art Intrusion Detection Systems," in *Proceedings of SPIE, Vol. 4232, Enabling Technologies for Law Enforcement and Security*, held in Boston, MA, November 5-8, 2000. Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, 2001.
9. Sumit Ghosh and Pete Robinson, "A Framework for Investigating Security Attacks in ATM Networks," in *IEEE Military Communications Conference (MILCOM 1999) Proceedings*, held in Atlantic City, NJ, October 31-November 3, 1999, IEEE, Piscataway, NJ, 1999.
10. A. Goldman, C. Wilcox, and R. McFarland, "ATDnet – A Gigabit ATM Network Testbed," in *Proceedings, Gigabit Networking Workshop GBN '97*, held in Kobe, Japan, April, 1997.
11. Stephen Hofmeyr, Stephanie Forrest, and Anil Somayaji, "Intrusion Detection using Sequences of System Calls," in *Journal of Computer Security*, vol. 6, pp. 151-180, 1998.
12. Tony Lee and Sumit Ghosh, "Simulating Asynchronous, Decentralized Military Command and Control," in *IEEE Computational Science and Engineering*, vol. 3, no. 4, pp. 69-79, 1996.
13. Tony Lee and Sumit Ghosh, "A Novel Approach to Asynchronous, Decentralized Decision-Making in Military Command and Control," in *Military Review*, No. 6, pp. 77-81.
14. Teresa F. Lunt, "A Survey of Intrusion Detection Techniques," in *Computers & Security*, vol. 12, pp. 405-418, 1993.

15. John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994.
16. Martin de Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN*, Ellis Horwood Ltd., New York, 1993
17. Abhijit S. Pandya and Ercan Sen, *ATM Technology for Broadband Telecommunications Networks*, CRC Press, Boca Raton, FL, 1999.
18. Perry J. Robertson, Robert L. Hutchinson, Lyndon G. Pierson, Thomas D. Tarman, and Edward L. Witzke, *Final Report and Documentation for the PLD11 Multipurpose Programmable Logic VME Board Design*, SAND99-0914. Sandia National Laboratories, Albuquerque, NM, April 1999.
19. Robert N. Smith, Douglas W. Hill, and N. P. Robinson, "ATM Peer Group Leader Attack and Mitigation," in *IEEE Military Communications Conference (MILCOM 1999) Proceedings*, held in Atlantic City, NJ, October 31-November 3, 1999, IEEE, Piscataway, NJ, 1999.
20. William Stallings, *SNMP, SNMPv2, and RMON: Practical Network Management*, Addison-Wesley, Reading, MA, 1996.
21. Altera Corporation, *System-on-a-Programmable-Chip Development Board User Guide ver. 1.1*, Altera Corporation, San Jose, CA, July 2000.
22. Thomas D. Tarman, Edward L. Witzke, Keith C. Bauer, Brian R. Kellogg, and William F. Young, "Asynchronous Transfer Mode (ATM) Intrusion Detection," Sandia National Laboratories report SAND2001-0561C (to be published in *MILCOM 2001 Proceedings*).
23. Thomas D. Tarman and Edward L. Witzke, *Hardware and Software Based Implementations of Passive-Direct Sensors for ATM Intrusion Detection*, SD-6771, October 2000.
24. Thomas D. Tarman and Edward L. Witzke, *Implementing Security for ATM Networks*, Artech House Publishers, Boston, MA, 2002.
25. Thomas D. Tarman and Edward L. Witzke, "Intrusion Detection Considerations for Switched Networks," in *Proceedings of SPIE, Vol. 4232, Enabling Technologies for Law Enforcement and Security*, held in Boston, MA, November 5-8, 2000. Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, 2001.
26. Thomas D. Tarman, "Security and Intrusion Detection in Multiprotocol Label Switching (MPLS) Networks," internal report SAND2001-1589P, May 4, 2001.

Appendix A: Sensor – Assessment Protocol Message Format

ATM intrusion detection messages shall be null-terminated ASCII text strings composed of a number of fields, which are separated by the delimiter character ':'. The formats of messages are dependent on the message TYPE.

TYPE
<i>Message-specific data</i>

Each field is a name/value pair in the form <field name>=<value>. Values for the message TYPE field include:

- TYPE=SENSOR NOTIFICATION
- TYPE=LOAD RULES
- TYPE=SENSOR STARTUP

Sensor notifications messages have the following format:

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=<sensor type_version>
SN_TYPE=<protocol> <protocol event>
<event data>
DIAG=<diagnostic data>

with the delimiter character ':' separating each field.

UNI Protocol Events

Setup Message Received

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDEu_1.0
SN_TYPE=UNI SETUP
CALLING_NSAP=<calling NSAP> (Note)
CALLED_NSAP=<called NSAP>
DIAG=<diagnostic data>

Note: NSAP addresses are 20-byte hex values, and hence, encode to 40 character ASCII strings.

Connect Message Received

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDE _u 1.0
SN_TYPE=UNI_CONNECT
CALLING_NSAP=<calling NSAP>
CALLED_NSAP=<called NSAP>
DIAG=<diagnostic data>

PNNI Protocol Events

New PNNI Node

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDE _p 1.0
SN_TYPE=PNNI_NEW_NODE
NODE_ID=<node ID>
(Note)
DIAG=<diagnostic data>

Note: Node IDs are 22-byte hex values, and hence, encode to 44 character ASCII strings. They are of the form <prefix length><address length><NSAP address>.

New PNNI Link

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDE _p 1.0
SN_TYPE=PNNI_NEW_LINK
SRC_NODE_ID=<node ID>
DEST_NODE_ID=<node ID>
DIAG=<diagnostic data>

Deleted Horizontal PNNI Link

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDE _p 1.0
SN_TYPE=PNNI_DELETE_HORIZ_LINK

SRC_NODE_ID=<node ID>
DEST_NODE_ID=<node ID>
DIAG=<diagnostic data>

New PNNI Peer Group Leader

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDep_1.0
SN_TYPE=PNNI_NEW_PGL
PEER_GROUP=<peer group ID> (Note 1)
PGL_ID=<peer group leader ID> (Note 2)
DIAG=<diagnostic data>

Note 1: Peer group ID is a 21-byte hex value of the form <prefix length><NSAP address>.

Note 2: Peer group leader ID is a 22-byte hex value of the form <prefix length><address length><NSAP address>.

Change PNNI Peer Group Leader

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNIDep_1.0
SN_TYPE=PNNI_PGL_CHANGE
PEER_GROUP=<peer group ID> (Note 1)
PGL_ID=<peer group leader ID> (Note 2)
DIAG=<diagnostic data>

Note 1: Peer group ID is a 21-byte hex value of the form <prefix length><NSAP address>.

Note 2: Peer group leader ID is a 22-byte hex value of the form <prefix length><address length><NSAP address>.

SNMP Events

UNI Signaling Down

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNMP_TRAP_1.0
SN_TYPE=asxQ2931Down
IP_ADDRESS=X.X.X.X (Note 1)
SW_PORT=<sw port number > (Note 2)

Note 1: This is the IP address of the qaaX interface on the ATM switch.

Note 2. This is an integer value representing the software port associated with the Q2931 signaling.

UNI Signaling Up

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNMP_TRAP_1.0
SN_TYPE=asxQ2931Up
IP_ADDRESS=X.X.X.X (Note 1)
SW_PORT=<sw port number > (Note 2)

Note 1: This is the IP address of the qaaX interface on the ATM switch.

Note 2. This is an integer value representing the software port associated with the Q2931 signaling.

ATM Switch Link Up

TYPE=SENSOR NOTIFICATION
AGENT_TYPE=SNMP_TRAP_1.0
SN_TYPE=asxSwLinkUp
IP_ADDRESS=X.X.X.X (Note 1)
SW_PORT=<sw port number > (Note 2)

Note 1: This is the IP address of the qaaX interface on the ATM switch.

Note 2. This is an integer value representing the software port associated with the link that comes “up”.

Appendix B: LDRD Data

FY00 Annual Report Summary

Status: Submitted September 2000

Project Number: 10767

Title: Intrusion Detection for Asynchronous Transfer Mode (ATM) Networks

Project Manager: VAHLE, MICHAEL O.

Project Investigator: TARMAN, THOMAS D.

Team Members: HUTCHINSON, ROBERT L., BAUER, KEITH C., YOUNG, WILLIAM F., WITZKE, EDWARD L.

Number of Patent Disclosures: 0

Number of Staff Hired: 0

Number of Applications: 0

Number of Post Docs: 0

Number of Patents: 0

Number of Students: 1

Number of Copyrights: 0

Number of Awards: 1

Names of Awards: "ATM Forum Spotlight Award," awarded July 2000 by the Asynchronous Transfer Mode (ATM) Forum for editing and technical work on ATM security specifications.

Quantitative Assessment: 100%

Qualitative Assessment: Goals met, Hypothesis remains unchanged

Abstract

Currently, network intrusion-detection (NID) systems are in place in networks at Sandia and its customers. NID systems allow site security personnel to monitor network devices and hosts for activities that indicate a possible attack, misuse, or operational anomaly in the information system. Since NID systems protect against attacks from inside and outside users, they provide a complementary service to devices that protect internal networks against outside attacks (e.g., encryptors and firewalls). However, current NID systems only work down to the Internet Protocol (IP) level. Data and control flows below IP are not covered by the current technology. This represents a risk to Asynchronous Transfer Mode (ATM) network backbones, which are found inside many enterprises (including Sandia), public carriers, and the Internet. This makes ATM networks vulnerable to a variety of denial-of-service attacks, routing infrastructure attacks, and other attacks.

This project is developing an ATM intrusion-detection system that monitors the signaling flows associated with a variety of ATM protocols (including virtual circuit setup, call routing, local area network [LAN] emulation, etc.) to determine if insider or outsider attacks are being conducted against the ATM network. This project will research and document issues such as analysis of ATM protocol attacks, sensor performance on high-bandwidth links and switch fabrics, and scalability. We will develop prototype ATM intrusion-detection systems that incorporate methods that address these specific concerns.

Sandia is uniquely qualified to make significant contributions to this area due to its recognized leadership in ATM security and high-speed encryption, and in-depth experience with ATM protocols and IP NID. Results from this work will lead to key technologies that can be incorporated into the ATM intrusion-detection devices for Sandia and DOE technologies. This work will also enhance Sandia's leadership in ATM security.

Accomplishments

We made much progress in the development of a proof-of-concept software-based ATM intrusion-detection system.

We selected and documented an architecture for the intrusion detection system. This architecture uses a number of ATM-specific sensors that monitor routing message flows, virtual circuit signaling flows, and Simple Network Management Protocol (SNMP) trap information. To accommodate this variety of sensors, we developed a common communication protocol, sensor message formats, and communication library functions. This protocol will allow the addition of new sensor types (e.g., hardware-based sensors) in the future.

We selected an assessment engine to correlate intrusion sensor events and to determine whether an attack is occurring in the network. This assessment engine provides a flexible rule specification language, which allows the end user to specify custom attack templates and responses to verified attacks. These rules are specified in a text file, which is generated manually or via a JAVA graphical user interface (GUI).

We developed a template for a simple but interesting attack scenario. While researching attack scenarios, we met with a researcher from Arizona State University to discuss his findings. We used this attack scenario to show successful detection of an ATM network attack. The prototype intrusion-detection system used software-based network event sensors, along with the assessment engine and rule generator described above.

Publications

Type: Other Publication, Miscellaneous publications

Authors: Tarman, T. D.

Title: Asynchronous Transfer Mode Intrusion Detection and Prevention

Location Published:

Detail: Invited talk to the Banking Industry Technial Symposium, Santa Fe, NM, 9 August 2000

Report Number: SAND99-0518C

Type: Other Publication, Presented but NOT Published in Proceedings

Authors: Tarman, T. D.

Title: Sandia's Straw Ballot Comments on the Security Version 1.1 Specification

Conference: Presentation to the ATM Forum Technical Committee

Date: May 2000

Location: San Francisco, CA

Type: Refereed Publication, Accepted for Publication

Authors: Tarman, T. D., and Witzke, E. L.

Title: Intrusion-Detection Considerations for Switched Networks

Publication Name: Proc. SPIE Photonics East 2000, accepted

Type: Other Publication, Presented but NOT Published in Proceedings

Authors: Witzke, E. L., and Tarman, T. D.

Title: Proposed Foreword to the ATM Security Specification, Version 1.1

Conference: ATM Forum Technical Committee

Date: May 2000

Location: San Francisco, CA

Type: Other Publication, Presented but NOT Published in Proceedings

Authors: Tarman, T. D., and M. Laurent (ENST, Bretagne, France).

Title: Security Services Negotiation Through Operations and Management (OAM) Cells

Conference: ATM Forum Technical Committee

Date: May 2000

Location: San Francisco, CA

Type: Other Publication, Miscellaneous publications

Authors: Kellogg, B. R.

Title: Intrusion-Detection Systems for Asynchronous Transfer Mode (ATM) Networks

Location Published:

Detail: Presentation to the Student Intern Program Symposium, 10 August 2000, Sandia National Laboratories, Albuquerque, NM

Report Number: SAND2000-1866

Non-LDRD Funding

Source: DoD

Amount: \$300K

FY01 Annual Report Summary

Status: Submitted September 2001

Project Number: 10767

Title: Intrusion Detection for Asynchronous Transfer Mode (ATM) Networks

Project Manager: STANS,LEONARD

Project Investigator: TARMAN,THOMAS D.

Team Members: BAUER,KEITH C., WITZKE,EDWARD L., YOUNG,WILLIAM F.,
KELLOGG,BRIAN R.

Number of Patent Disclosures: 1

Number of Staff Hired: 1

Number of Applications: 0

Number of Post Docs: 0

Number of Patents: 0

Number of Students: 0

Number of Copyrights: 0

Number of Awards: 0

Names of Awards:

Quantitative Assessment: 100%

Qualitative Assessment: Goals met, Hypothesis proved

Abstract

Currently, network intrusion detection (NID) systems are in place in networks at Sandia and its customers. NIDs allow site security personnel to monitor network devices and hosts for activities that indicate a possible attack, misuse, or operational anomaly in the information system. Since NIDs protect against attacks from inside and outside users, they provide a complementary service to devices that protect internal networks against outside attacks (e.g., encryptors and firewalls). However, current NID systems only work down to the Internet Protocol (IP) level. Data and control flows below IP are not covered by the current

technology. This represents a risk to Asynchronous Transfer Mode (ATM) network backbones, which are found inside many enterprises (including Sandia), public carriers, and the Internet. This makes ATM networks vulnerable to a variety of denial of service attacks, routing infrastructure attacks, and other attacks.

This project is developing an ATM intrusion detection system that monitors the signaling flows associated with a variety of ATM protocols (including virtual circuit setup, call routing, local area network emulation, etc.) to determine if insider or outsider attacks are being conducted against the ATM network. Issues such as analysis of ATM protocol attacks, sensor performance on high-bandwidth links and switch fabrics, and scalability issues will be researched and documented during this project. Prototype ATM intrusion detection systems will be developed that incorporate methods that address these specific concerns.

Sandia is uniquely qualified to make significant contributions to this area due to its recognized leadership in ATM security and high-speed encryption, and in-depth experience with ATM protocols and IP network intrusion detection. Results from this work will lead to key technologies that can be incorporated into the ATM intrusion detection devices for Sandia and DOE technologies. This work will also enhance Sandia's leadership in ATM security.

Accomplishments

During this year of research, we completed development of a software-based ATM intrusion detection system and demonstrated it in small laboratory environments at Sandia and at the Stevens Institute of Technology, and in a large unclassified test network for the DoD at one of their locations. In all of these demonstrations, the system successfully detected a mock "ATM peer group leader takeover attack" (described in a MILCOM 1999 paper by Smith and Robinson). This successful detection involved all of the subcomponents in our ATM intrusion detection architecture, including the active-direct sensors (for the Private Network to Network Interface, or PNNI, protocol), the passive-direct sensors (for the User to Network Interface, or UNI, protocol), the indirect sensors (for the Simple Network Management Protocol, or SNMP), the assessment engine, rule editor, and response components (both active and passive). In response to the attack, the assessment engine successfully responded by sending a notification to the administrator, and isolating the offending intruder by shutting down an appropriate port on the switch to which it attached. In addition, the DoD installation also showed the scalability of the basic, software-based architecture by successfully "sifting through the noise" of a large number of new-switch events to successfully detect the mock attack.

As part of our collaboration with Dr. Sumit Ghosh of the Stevens Institute of Technology (formerly with Arizona State University), we initiated a contract with him to examine scalability and response timeliness issues associated with our intrusion detection approach, and with distributed intrusion detection systems in general. We delivered a working ATM intrusion detection system to him in April, set it up in his lab, and demonstrated the system to him. His progress reports indicate the need to partition a distributed intrusion detection system into two components -- a strategic component which extracts parameters (sensor notifications), and a tactical component which is configured by the strategic part with network reconfiguration rules and responds according to these rules. A detailed description of Dr. Ghosh's results is found in the project's final report.

Another aspect of this work that was performed in FY01 was the continued development of the UNI sensor, which is a passive-direct sensor (that is, a sensor that directly monitors ATM protocol flows, but does so silently, without actively participating as a network device). This sensor was designed to facilitate separation of the sensor configuration user interface from the actual sensor device, which allows distributed management of multiple UNI sensors. The UNI sensors themselves were designed to be implemented on a Linux platform, with some functionality in hardware and the rest in software. A language for communicating filter specifications between the user interface and the sensor was developed, and a protocol for securely transferring filter specifications (based on secure shell, with public key cryptography) was also developed. Although all sensor components were implemented in software, progress was made in the design and implementation of the hardware components on an Altera System on a Programmable Chip (SOPC) platform.

Given the current network market conditions, vendor interest has now shifted away from ATM and toward IP-based flow switching techniques such as MultiProtocol Label Switching (MPLS). Therefore, this project also spent some time analyzing the MPLS protocols to determine how the ATM intrusion detection architecture can be applied to MPLS networks. The results of this analysis are documented in an internal white paper, and a copy has been sent to our collaborator at the DoD.

Publications

Type: Refereed Publication, Accepted for Publication

Authors: Thomas D. Tarman, Edward L. Witzke, Keith C. Bauer, Brian R. Kellogg, William F. Young

Title: Asynchronous transfer mode (ATM) intrusion detection

Publication Name: Proceedings MILCOM 2001

Type: Refereed Publication, Presented and Published in Proceedings

Authors: Thomas D. Tarman, Edward L. Witzke

Title: Intrusion detection considerations for switched networks

Publication Name: Proceedings of SPIE - Enabling Technologies for Law Enforcement and Security

Date: November 5-8, 2000

Volume: 4232

Pages: 85-92

Conference: SPIE Photonics East 2000

Date: November 5-8, 2000

Location: Boston, MA USA

Non-LDRD Funding

Source: DoD WFO

Amount: \$300K

Appendix C: Timeline of Sandia's ATM Forum Impact

Given the current market shift away from ATM, this project is probably the last Sandia LDRD project relating to ATM. Over the past nine years, Sandia LDRDs have had a key role Sandia's work in the ATM Forum (an ATM standards body) developing specifications for security, high-performance physical layer devices, and flow control protocols. As Figure 20 shows, the past nine years have been very productive.

Figure 20: Timeline of Sandia's impact in the ATM Forum

(sources: Helen Chen, Steve Gossage, Lyndon Pierson, and Tom Tarman)

DISTRIBUTION:

1		Andy McFarland 9800 Savage Rd. Ft. Meade, MD 20755 Attn: R23
5		Sumit Ghosh Director, Computer Engineering Program Dept. of Computer and Elect. Engineering Stevens Institute of Technology Burchard Building, Room 212 Hoboken, NJ 07030
1	M/S 0139	M.O. Vahle, 9900
1	M/S 0451	S.G. Varnado, 6500
1	M/S 0630	J.P. VanDevender, 9400
1	M/S 0785	R.L. Hutchinson, 6516
5	M/S 0785	W.F. Young, 6516
1	M/S 0801	A.L. Hale
1	M/S 0801	M.R. Sjulín, 9330
1	M/S 0801	W.F. Mason, 9320
1	M/S 0806	P.C. Jones, 9332
1	M/S 0806	D.F. Beck, 9332
1	M/S 0806	C.D. Brown, 9332
1	M/S 0806	L. Stans, 9336
1	M/S 0806	J.P. Brenkosh, 9336
1	M/S 0806	J.M. Eldridge, 9336
1	M/S 0806	M.J. Ernest, 9336
1	M/S 0806	S.A. Gossage, 9336
1	M/S 0806	J.R. Hamlet, 9336
1	M/S 0806	R.L. Hartley, 9336
1	M/S 0806	T.C. Hu, 9336
1	M/S 0806	J.A. Hudson, 9336
5	M/S 0806	B.R. Kellogg, 9336
1	M/S 0806	L.G. Martinez, 9336
1	M/S 0806	M.M. Miller, 9336
1	M/S 0806	J.H. Naegle, 9336
1	M/S 0806	L.G. Pierson, 9336
1	M/S 0806	T.J. Pratt, 9336
1	M/S 0806	J.A. Schutt, 9336
10	M/S 0806	T.D. Tarman, 9336
1	M/S 0806	L.F. Tolendino, 9336
5	M/S 0806	E.L. Witzke, 9336
1	M/S 0807	M.R. Epperson, 9338
1	M/S 0813	R.M. Cahoon, 9327
1	M/S 0813	R.A. Suppona, 9327
1	M/S 1137	J.L. Mitchiner, 6534
5	M/S 1137	K.C. Bauer, 6534
1	M/S 1206	J.V. Vonderheide, 5933

1	M/S 0161	Patent and Licensing Office, 11500
1	M/S 0188	LDRD Office (D. L. Chavez)
1	M/S 9018	Central Technical Files, 8945-1
2	M/S 0899	Technical Library, 9616
1	M/S 0612	Review and Approval Desk, 9612 For DOE/OSTI