

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

DYNAMIC PARAMETERIZATION OF IPSEC

by

Christopher D. Agar

December 2001

Thesis Advisor:
Second Reader:

Cynthia E. Irvine
Timothy E. Levin

Approved for public release; distribution is unlimited

Report Documentation Page		
Report Date 19Dec2001	Report Type N/A	Dates Covered (from... to) -
Title and Subtitle Dynamic Parameterization of IPSEC	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Christopher Agar	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Naval Postgraduate School Monterey, California	Performing Organization Report Number	
Sponsoring/Monitoring Agency Name(s) and Address(es)	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes		
Abstract		
Subject Terms		
Report Classification unclassified	Classification of this page unclassified	
Classification of Abstract unclassified	Limitation of Abstract UU	
Number of Pages 334		

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Dynamic Parameterization of IPsec			5. FUNDING NUMBERS	
6. AUTHOR(S) Christopher D. Agar				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The Internet has become the medium of choice for communications between most Government and Military organizations. Unfortunately the key Internet protocols were not designed to provide security and their security vulnerabilities have become apparent. IPsec was developed to provide users with a range of security services, for both confidentiality and integrity, enabling them to securely pass information across networks. Automated security mechanisms are typically designed and/or calibrated to meet an organization's security policy. However, once the mechanism is in operation the implemented policy is in a static state, and cannot be adjusted according to dynamic environmental conditions. This means that security mechanisms fail to reflect the policy that is appropriate for the changing contexts. Dynamic parameterization enables security mechanisms to adjust the level of security service "on-the-fly" to respond to changing conditions (i.e. INFOCON, THREATCON). This work includes the extension of the attributes encoded by the KeyNote Trust Management System and modification of the IPsec mechanism to incorporate dynamic parameters into the security service selection mechanism, and the construction of a graphical user interface, for demonstrating "proof-of-concept" of Dynamic Parameterization of OpenBSD 2.8 IPsec.				
14. SUBJECT TERMS KeyNote, ISAKMP, IKE, IPsec, Graphical User Interface, Security Association (SA), Security Policy Database (SPD), Security Association Database (SAD), Security Proposal			15. NUMBER OF PAGES 334	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

DYNAMIC PARAMETERIZATION OF IPSEC

Christopher D. Agar
Lieutenant, United States Navy
B.S., University of Florida, 1992

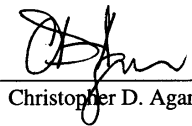
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

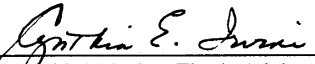
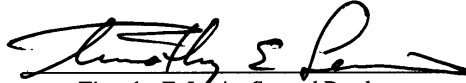
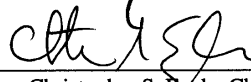
from the

**NAVAL POSTGRADUATE SCHOOL
December 2001**

Author:


Christopher D. Agar

Approved by:


Cynthia E. Irvine, Thesis Advisor
Timothy E. Levin, Second Reader
Christopher S. Egle, Chairman
Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Internet has become the medium of choice for communications between most Government and Military organizations. Unfortunately the key Internet protocols were not designed to provide security and their security vulnerabilities have become apparent. IPsec was developed to provide users with a range of security services, for both confidentiality and integrity, enabling them to securely pass information across networks. Automated security mechanisms are typically designed and/or calibrated to meet an organization's security policy. However, once the mechanism is in operation the implemented policy is in a static state, and cannot be adjusted according to dynamic environmental conditions. This means that security mechanisms fail to reflect the policy that is appropriate for the changing contexts. Dynamic parameterization enables security mechanisms to adjust the level of security service "on-the-fly" to respond to changing conditions (i.e. INFOCON, THREATCON). This work includes the extension of the attributes encoded by the KeyNote Trust Management System and modification of the IPsec mechanism to incorporate dynamic parameters into the security service selection mechanism, and the construction of a graphical user interface, for demonstrating "proof-of-concept" of Dynamic Parameterization of OpenBSD 2.8 IPsec.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	PURPOSE.....	1
B.	BACKGROUND	1
C.	EXPECTED BENFITS OF THE RESEARCH	2
D.	RESEARCH OBJECTIVES	3
E.	THE ROAD TO DYNAMIC PARAMETERIZATION	3
II.	QUALITY OF SECURITY SERVICE (QOSS).....	5
A.	INTRODUCTION.....	5
B.	QUALITY OF SECURITY SERVICE (QOS).....	5
C.	QUALITY OF SECURITY OF SERVICE (QOSS)	7
1.	Managing Quality of Security Service (QoSS)	9
D.	CONCLUSION	13
III.	SYSTEM ARCHITECTURE	15
A.	INTRODUCTION.....	15
B.	OVERVIEW OF IPSEC.....	15
C.	IPSEC ARCHITECTURE	18
1.	Security Services Provided by IPsec	18
D.	ANALYSIS	65
1.	IPsec (Re)Initialization.....	65
2.	IPsec Output Processing.....	65
3.	IPsec Input Processing.....	67
E.	CONCLUSION	69
IV.	DESIGN AND PROCESS	71
A.	INTRODUCTION.....	71
B.	PROVIDING GRANULARITY TO KEYNOTE	72
1.	Goal 72	
2.	Process Review.....	72
3.	Modification Phases.....	78
C.	PARAMETERIZING AND IMPROVING ISAKMPD.CONF – KEYNOTE PROPOSAL LOADING PROCESS	79
1.	Goal 79	
2.	Process Review.....	79
3.	Modifications Phases.....	82
4.	Testing 83	
D.	CONCLUSION	83
V.	IMPLEMENTATION	85
A.	INTRODUCTION.....	85
B.	PROVIDING GRANULARITY TO KEYNOTE	85
1.	Parameterization of KeyNote.....	85

C.	REPLACING ISAKMPD.CONF WITH KEYNOTE	93
1.	Current isakmpd.conf	93
2.	Process of Replacing isakmpd.conf with KeyNote	93
D.	CONCLUSION	129
VI.	GRAPHICAL USER INTERFACE (GUI) DEMONSTRATION	131
A.	INTRODUCTION	131
B.	COMMAND-LINE ENVIRONMENT	131
1.	IPsec System Flush	131
2.	Setting Up and Mounting the Security Policy Database	131
3.	IPsec Execution	133
4.	IPsec Connection Termination	134
5.	Display SPD 134	
6.	Display SAD 135	
7.	tcpdump 136	
C.	GRAPHICAL USER INTERFACE (GUI) DEMONSTRATION	136
1.	Goal 136	
2.	Mechanism of Demonstration	136
3.	Graphical Demonstration Components	139
D.	CONCLUSION	175
VII.	RESEARCH SUMMARY AND FUTURE WORK	177
A.	INTRODUCTION	177
B.	SUMMARY OF RESEARCH PERFORMED IN THIS THESIS	177
1.	Research Conclusions	179
C.	FUTURE DESIGN AND IMPLEMENTATION ON PARAMETERIZATION	180
1.	Ability to Handle all Possible Security Parameter Combinations	180
2.	Improving Dynamic Parameter Loading by Utilizing “Policy- Callback” Embedded Functionality	180
3.	Eliminate the Need for isakmpd.conf Entirely	181
4.	Develop a Parsing Mechanism to Retrieve the Initial Security Policy Database Entries	181
D.	HARNESSING OPENBSD’S IPSEC MECHANISM CAPABILITIES	181
1.	Behavior with all Possible Combinations of QoS and non- QoS Peers	181
2.	Per-User / Per-Application Relationship Capability	182
3.	Explore Proposal Caching Issues	182
4.	Security Policy Editor	182
5.	Additional Network Configurations	182
6.	IPV6 Addresses	183
7.	Distribution of KeyNote Policies	183
8.	KeyNote Protection	183
9.	Secure Dissemination & Storage of QoS Parameters' Values	183
10.	IPsec Costing Issues	183
11.	Graphical User Interface	183

E. CONCLUSION	184
APPENDIX A. CONF.C.....	185
APPENDIX B. IKE_QUICK_MODE.C	231
APPENDIX C. DEMO.JAVA.....	235
APPENDIX D. SPD.JAVA.....	245
APPENDIX E. DEMO_SUPPORT_FUNCTIONS.JAVA.....	251
APPENDIX F. SPFK.JAVA.....	265
APPENDIX G. DP_CONSOLE.JAVA	269
APPENDIX H. IPSECINFO.JAVA	281
APPENDIX I. TCPDUMP.JAVA.....	295
APPENDIX J. ISAKMPD.CONF FILE	299
APPENDIX K. ISAKMPD.POLICY FILE	301
APPENDIX L. KEYNOTEDNFFINAL.POLICY FILE.....	305
APPENDIX M. SECURITY PROPOSAL SUMMARY	309
LIST OF REFERENCES	311
INITIAL DISTRIBUTION LIST	315

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 2.1.	Quality of Service (QoS).....	7
Figure 2.2.	Quality of Security Service (QoSS).....	8
Figure 2.3.	Mapping Security Attributes to Security Policy.	12
Figure 2.4.	Costing Framework Model.	13
Figure 3.1.	IPsec vs. VPN Security Mechanisms.	16
Figure 3.2.	IPsec Architecture.	18
Figure 3.3.	ESP- Protected IP Packet.	20
Figure 3.4.	Encrypting and Decrypting using CBC and IV.	21
Figure 3.5.	AH-Protected IP Packet.	22
Figure 3.6.	IPsec Configurations.	24
Figure 3.7.	IPsec Transport and Tunnel Modes.	25
Figure 3.8.	Transport Adjacency.	27
Figure 3.9.	Iterated Tunneling.	27
Figure 3.10.	IPsec Security Policy.	29
Figure 3.11.	IPsec Security Policy Database (SPD) Populating Mechanisms.	30
Figure 3.12.	Security Policy Database (SPD).	32
Figure 3.13.	SAD: Security Association Database.....	34
Figure 3.14.	IPsec Input Module.	37
Figure 3.15.	IPsec Output Module.	38
Figure 3.16.	Controlling Events for IKE.	41
Figure 3.17.	IPsec Security Association Process Defined by ISAKMPD.	42
Figure 3.18.	IKE Phase I – Main Mode.....	43
Figure 3.19.	IKE Phase I Aggressive Mode.	44
Figure 3.20.	IKE Phase II – Quick Mode.	46
Figure 3.21.	isakmpd.conf Process.....	58
Figure 3.22.	KeyNote Process.	60
Figure 3.23.	IPsec Architecture.	67
Figure 3.24.	IPsec Architecture.	69
Figure 4.1.	Current KeyNote Process.....	74
Figure 4.2.	An Example of a Condition Statement inKeyNote.	76
Figure 4.3.	Modified KeyNote Process.	77
Figure 4.4.	Current (Re)Initialization Process.....	80
Figure 4.5.	Modified (Re)Initialization Process.....	81
Figure 5.1.	Current KeyNote Query Process.....	89
Figure 5.2.	Modified KeyNote Query Process.	92
Figure 5.3.	Logical Flow of Functions for Parsing KeyNote into isakmpd.conf Syntax.	116
Figure 5.4.	Logical Flow of Security Proposal Parsing and Loading Process with the Added Dynamic Parameter Interface.	122
Figure 5.6.	Security Proposal Default Loading Process.....	126
Figure 5.7.	Security Proposal Duplicate Checking Process.	129

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 3.1. Possible Selector Combination.	35
Table M.1. Security Proposal Summary	309

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to thank and acknowledge the following personnel:

- Dr Cythina Irvine, for her professional guidance and direction, inspiring me to go beyond my previously known limits and capabilities.

- Tim Levin, for his expert knowledge, guidance and patience.

- Evie Spyropoulou, for her patience, long hours of trouble-shooting, and expert system knowledge, without which the research would have been seriously hindered.

- Bruce Allen his vast knowledge of Java.

- David Schiffett for his mastery of C.

- Eliane Christian for very soothing patience during a very stressful time.

- Last but certainly not least, my wonderful and beautiful wife, Amy, who provided me with unlimited support and love during the performance of this research. Without her support, none of my accomplishments would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

Current implementations of IPsec and KeyNote do not provide the granularity or flexibility to adjust security controls to reflect real-world threat, information security levels and priority requirements for data delivery. The purpose of this thesis is to analyze, design and implement an interface that interacts with a Trust Management System (KeyNote) to dynamically modify the protection of IPsec communications in response to changing security threats, conditions and situations.

B. BACKGROUND

Information assurance and security have grown in concern and importance along with our dependence on network communications and our connectivity across the Internet. The DOD requires the ability to communicate securely within computer networks via the Internet. Unfortunately the key Internet protocols were not designed to provide security. As our dependence on the Internet has grown and its security vulnerabilities have become apparent.

Typically a “penetrate-and-patch” technique is used to attempt to locate and fix all system vulnerabilities when security is applied after development and implementation. The weakness in this approach is that to be 100% successful, you have to find all vulnerabilities and patch each one. This is very costly in man-hours and not guaranteed to be 100% effective. It is better to build security into a system from the ground up, incorporating secure methodology into the design and implementation phase. For example, formal methods may be used to mathematically determine the system’s assurance level.

One of the first successful attempts to secure network communication across an insecure medium (Internet), was the Virtual Private Network (VPN). This involved the establishment of a static secure connection between two peers, gateways or a combination of both. The limitation of a simple VPN is that it is not capable of applying a complex security policy in which different applications and different users would require different levels of security.

IPsec was developed to provide further granularity for the Internet Protocol (IP). IPsec extends the IP Protocol to enable security for TCP/IP communications. IPsec provides both secrecy and integrity services. A wide variety of choices are available when establishing protected communications across the network. The appropriate choice and combination of secrecy and integrity mechanisms will depend upon the “trust relationships” between the communicating entities. Those relationships are constrained by the policy of each entity. Negotiation of policy and mechanisms takes place in the context of the Internet Key Exchange (IKE) framework and the Internet Security Association and Key Management Protocol (ISAKMP). However, IKE and ISAKMP do not provide a general mechanism for managing and incorporating security policy. In order to ensure that IPSEC consistently meets local security policy needs of the user, a Trust Management System is used to encode policy and support communications security negotiation and management. (Thayer, R., Doraswamy, N., and Glenn, R., 1998)

A trust management system unifies the elements of security policy, credentials, access control, and authorization. Applications can use the Keynote trust management system to verify, through the compliance checker, whether a requested action is authorized. (Matt Blaze, John Ioannidis, and Angelos D. Keromytis, Feb 2000)

The concept of Quality of Security Service (QoSS) provides the foundation for implementation of security mechanisms, such as IPsec, that utilize a trust management system to manage security according to policy. QoSS provides a means to manage security services based on the requirements set by the user's requests, the system's security policy, the availability of system resources and the network environment. (Irvine, C.E. and Levin, T, September 2000)

Currently, IPSEC and Keynote do not have the flexibility to adjust security controls to adapt to changes in threat conditions, critical time transmissions, and network congestion/traffic. By providing more granularity through parameterization in IPsec, these and other dynamic security requirements can be represented and accurately trigger adjustments in security services.

C. EXPECTED BENEFITS OF THE RESEARCH

By providing dynamic parameterization to IPsec, government and military

security systems will be able to automate security service adjustments according to dynamic environmental parameter settings, such as INFOCON and THREATCON. Currently, when a dynamic environmental value changes, security systems must be stopped, reconfigured and executed to incorporate policy changes appropriately. With dynamic parameterization, the security services will adjust “on-the-fly” in accordance with local security policy. This research will provide a foundation for allowing IPsec mechanisms to be managed under dynamically changing network conditions. The additional granularity intended for the IPsec mechanism will allow it to reflect both Quality of Security Service requirements and highly granular security policies.

D. RESEARCH OBJECTIVES

The objectives of this research are three fold:

- Thoroughly study the current implementation of IPsec, specifically OpenBSD 2.8, to gain an understanding of the security mechanism and its components.
- Design and develop a dynamic parameterization module, providing an interface that will enable users to select values for “dynamic parameters,” and ultimately result in an IPsec reconfiguration according to established security policy.
- Design and develop a method of demonstrating the results of this research, specifically to users with limited knowledge of the OpenBSD operating environment.

E. THE ROAD TO DYNAMIC PARAMETERIZATION

The following chapters will be provided to describe the work required to achieve the objectives listed above: background, analysis, design, implementation, testing and demonstration.

- Chapter II **Quality of Security Service(QoS)** – a brief introduction to QoS.
- Chapter III **IPsec Architecture** – a review IPsec and its implementation in OpenBSD 2.8.
- Chapter IV **Design and Process** – an outline and description of the design and process phase of dynamic parameterization.
- Chapter V **Implementation** – a description of the methodology, and actual implementation used to achieve the objectives, including pseudo algorithms and source code.
- Chapter VI **Graphical User Interface Demonstration** – a description of the design and implementation of the GUI demonstration module.

- Chapter VII **Research Summary and Future Work** – a summary of the completed research and a discussion of future related work.

II. QUALITY OF SECURITY SERVICE (QOSS)

A. INTRODUCTION

The challenge of providing users with consistent and reliable access to resources in a distributed networking environment, such as across the Internet, has become a real and troublesome problem for System Administrators and Resource Managers. The former linear solution of providing resources by calculating and controlling the number of users and types of terminals cannot be used outside the local network environment. The Quality of Service (QoS) mechanism was developed to handle this management problem. It provides a metric to measure and manage computational resources in an effort to provide requested levels of service to customers. (Aurrecoechohea, C., Campbell, A., and Hauw, L. A., 1996) (Chatterjee, S., Sabata, B., Sydir, J., May 1998)

A similar challenge exists within the realm of network computer security. Here the objective is to be able to fulfill user security requests and adapt and adjust accordingly to environment security changes. This chapter discusses Quality of Security Service (QOSS), the foundation and methodology of Quality of Service, its importance, and costing mechanisms. The concept of dynamic parameters, Network Mode and Security Level, is introduced. These parameters allow environmental conditions to be mapped accordingly to security policy and ultimately determining the security parameters used by the system. To support security system management, a costing framework that will allow system administrators to properly orchestrate security resources is also discussed. (Spyropoulou, E., Levin, T., and Irvine, C.E., December 2000)

B. QUALITY OF SECURITY SERVICE (QOS)

The Quality of Service (QoS) mechanism, see Figure 2.1, provides the ability to manage network and computational resources in accordance with the user's service requests, availability of system resources, and the network environment. QoS in simple terms, is a mechanism that establishes a contract between users and resource managers. The service requests are based on the following abstract attributes: performance, accuracy and precision. A user will typically request a certain level of service and the resource manager will either approve the request or deny it, possibly offering another available

level of service. (Aurrecoechohea, C., Campbell, A., and Hauw, L. A., 1996)(Chatterjee, S., Sabata, B., Sydir, J., May 1998)

The QoS mechanism may also be required to adjust negotiated levels of service as the availability of resources increases or decreases. This will require the Quality of Service (QoS) mechanism to govern all services provided to all users according to overall system policy requirements. This allows the management system to control resources and services as a whole in the event of a shift of prioritization or a loss or gain of resources. Resource usage policies may vary, for example to allocate resources and services equally or perhaps to provide a prioritized range of security services. Ultimately, a user's level of service may be modulated to accommodate the system's policy and availability of resources. Of course, system policy level requirements will dominate individual level requirement of services and resources. (Aurrecoechohea, C., Campbell, A., and Hauw, L. A., 1996)(Chatterjee, S., Sabata, B., Sydir, J., May 1998)

A further level of granularity to service level negotiations can be provided by utilizing hard and soft requirements. Hard requirements must be met in order for the QoS mechanism to accept the user's request. Soft requirements permit requests to be satisfied by a range of acceptable services. The soft variables may be adjusted during service to accommodate other users or network environmental factors. (Aurrecoechohea, C., Campbell, A., and Hauw, L. A., 1996)(Chatterjee, S., Sabata, B., Sydir, J., May 1998)

Providing different ranges and levels of potential service enables a system manager to effectively orchestrate system resources dependent upon requests, network environment and available resources.

Quality of Service (QoS)

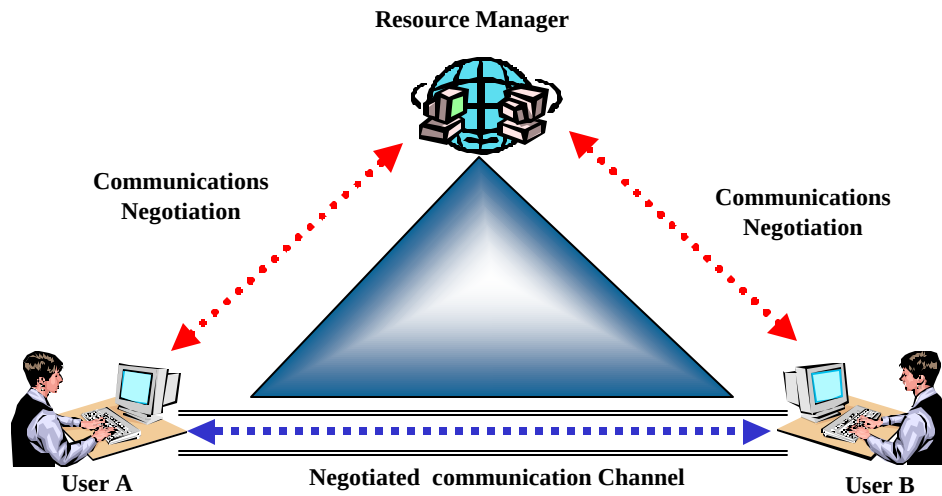


Figure 2.1. Quality of Service (QoS).

C. QUALITY OF SECURITY OF SERVICE (QOSS)

With the increased interest and implementation of computer security systems, specifically their effectiveness and impact on operational functionality, mechanisms to monitor and mediate security controls are required. In the QoS model, resource allocation is adjusted to meet user requests under changing network environment and resource availability conditions. Similarly, Quality of Security Service (QoSS), see Figure 2.2 provides a mechanism to manage security services to meet requirements set by the user's requests, system's security policy, availability of system resources and network environment. (Irvine, C.E. and Levin, T., September 2000)

Similar to the modulation of resources to support QoS, security services can be defined in terms of user and system requirements, network environment factors and available resources. Without a range of security services, a user is faced with the rigid and limited choice of "all or nothing": security or no security. Historically, security services have been provided in such a static manner. (Spryropoulou, Evdoxia, Agar, Chris, Levin, Timthoy, Irvine, Cynthia, January 2002) Quality of Security Service (QoSS) provides a more flexible solution to the provision of security services. The

security resource manager and/or the security system can adjust security service to meet user requirements, system security policy and network environment constraints. (Irvine, C.E. and Levin, T., September 2000)

Security systems and managers can maintain overall control of the security mechanism through QoSS “system security policies.” These policies dominate the individual “user security requirements.” Specifically, they define all authorized operations per user, system, application , etc.

QoSS has several mechanisms to handle security variances. A security variance exists when security policies may be enforced utilizing a specific range of attributes. Therefore, based on the policy parameters, the attributes used to enforce the security policy may differ according to selection criteria. Fixed requirements are used to set minimum level acceptable security standards. A range of security settings meeting or exceeding this minimum level can be provided. For example a system may utilize SHA as a minimum level authentication algorithm for all message handling. Further granularity in support of confidentiality could be applied to messages by users or applications by selecting an encryption algorithm from a provided range. Other examples of security attributes that may be used are: assurance level, key length or security attribute expiration date stamp. (Irvine, C.E. and Levin, T., September 2000)

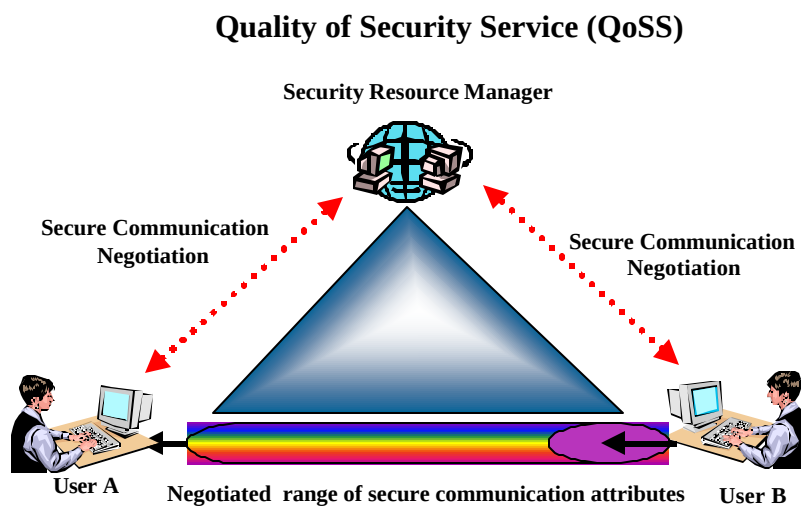


Figure 2.2. Quality of Security Service (QoSS).

1. Managing Quality of Security Service (QoSS)

Inevitably, security mechanisms result in a cost to the user, system and resources. Whether in the form bandwidth, algorithm processing time, overhead, or funds, the cost of security is a challenging concern to resource managers. A costing framework is required to map security service resource consumption to available resources, ultimately enabling a management system to efficiently and effectively handle security service costs.

Security services, as previously described, may utilize high level services and consume lower level resources in a system. High level services include, for example, non-repudiation, auditing, authentication, encryption, or intrusion detection. Low level resources include memory, bandwidth, or processor time. Further, each security service will require a governing policy, consisting of specific rules that determine how and when to use the service. Therefore each network task associated with QoSS can be mapped to a vector of security requirements directly associated with the security services the task requires. (NPS-CS-02-001, January 2002)

1.1 Dynamic Parameters

Government and DOD organizations utilize a variety of dynamic parameters to define a predefined response of specific actions according to policy. Examples include INFOCON and THREATCON levels. In order for a security mechanism to be fully functional within the DOD and Government infrastructure, it has to be able to incorporate the dynamic parameters into the security setting decision making process. A change in an INFOCON or THREATCON level should have an immediate effect on attributes and settings in a security mechanism. By introducing a dynamic mechanism, a system can modulate its security settings in response to these dynamic parameters. Security level and network mode, defined in the next section, have been chosen as two abstract dynamic parameters that govern changes to security attributes as defined in the an organization's security policy. (NPS-CS-02-001, January 2002)

By developing and implementing a security mechanism that can dynamically adjust in accordance with a change to network modes and/or security levels, the users and

managers do not have to be concerned with the fine granularity and low-level complexity involved in adjusting and selecting appropriate security attributes.

1.1.1 User Choices for Security Levels

Security levels are a common metric used in the government and DOD to distinguish authorization for classified information. Common levels include Top Secret, Secret, Classified and Unclassified. Each of these levels correspond to different governing policies and requirements associated with the threat to national security by the disclosure of information to adversaries. Likewise, security levels, as defined here for proof of concept, represent an increasing requirement for stronger security (e.g. encryption and authentication algorithms).

Network security policies will require a range of maximum and minimum security levels. Minimum security levels set the lowest acceptable security attributes and maximum security levels establish a ceiling on the use of available security resources. Intersections of policies require further granularity in security settings to satisfy all governing users and systems. A user may also desire to select a higher level of security than the predefined minimum. (NPS-CS-02-001, January 2002)

A user or application, however, may quickly become overwhelmed with the security setting details, potentially resulting in degraded security or performance. By developing security definitions that encompass detailed security settings required by users or applications, the complexity of the selection process for the security settings can be simplified to a reasonable level. One approach would involve the use of the following Network Security levels: high, medium and low. (NPS-CS-02-001, January 2002) High security level would utilize strong levels of security attributes, medium level, moderate level of security attributes, and low level, low to no security attributes.

By implementing this approach the system security resource manager or security engineer is responsible for presetting security variables and ranges in accordance with choices offered to users or applications. A mapping of allowable security settings to security levels providing a range of selection or specific values will be required to properly enforce the system security policy. (NPS-CS-02-001, January 2002)

1.1.2 The Notion of Network Modes

Networks exist in a variety of states, providing users and systems with varying levels of service. On one occasion the network may experience heavy levels of traffic resulting in a poor performance. At other times the network may be limited in the availability of resources due to maintenance, and at other times the network may be performing at its optimum level. To fully incorporate the performance and reliability of the network into a security mechanism, the notion of network modes is introduced.

There are numerous situations in which a network security policy will be required to dynamically change to properly address the current operational threats and needs, as well as the availability of resources and network performance. In the midst of a highly sensitive intelligence operation, a transmitted report will require the highest possible security to ensure the information and the source remain protected. In another scenario, a unit confronted with serious emergency will require the fastest possible transmission, and may not be concerned with transmission protection. (NPS-CS-02-001, January 2002) Therefore a requirement exists for a dynamic security mechanism that can appropriately adjust to meet the needs of the system, users or applications. One approach is to use the following network modes: normal, impacted, and crisis. Normal mode is defined as ordinary operating conditions with normal traffic load and no heightened threat conditions. Impacted mode may be defined when the network/system is experiencing high levels of traffic and therefore certain security selection may not be available due to efficiency constraints. Emergency mode may be defined as a situation that requires the highest level of security or the lowest level dependent on the situation and policy. (NPS-CS-02-001, January 2002)

1.2 Mapping Abstract Parameters to Security Mechanism

A mapping of abstract dynamic parameters to resident security mechanisms is required to properly enforce policy decisions. For example, network modes may be mapped to security level ranges and ultimately to security attributes and settings. The security resource manager and security engineer would define the network modes and security levels to provide the users and applications with appropriate security service as translated into QoS choices. Once defined, the complexity of the security mechanism

and security attribute selection is transparent to the user. (See Figure 2.3)

Mapping Security Attributes to Security Policy

Network Mode	Security Level	Security Attributes
Normal	Low	ENCRYPTION: NONE AUTHENTICATION: MD5
	Medium	ENCRYPTION: DES AUTHENTICATION: MD5
	High	ENCRYPTION: 3DES AUTHENTICATION: MD5
Crisis	Low	ENCRYPTION: NONE AUTHENTICATION: NONE
	Medium	ENCRYPTION: NONE AUTHENTICATION: NONE
	High	ENCRYPTION: DES AUTHENTICATION: MD5
Impacted	Low	ENCRYPTION: 3DES AUTHENTICATION: MD5
	Medium	ENCRYPTION: 3DES AUTHENTICATION: SHA
	High	ENCRYPTION: AES AUTHENTICATION: SHA

Broad **Granularity** **Fine**
 Users System Admin/
 Security Experts

Figure 2.3. Mapping Security Attributes to Security Policy.

1.3 Costing for QoS

As mentioned earlier, use of security services has a direct impact on systems, users and/or applications. The cost may differ depending upon the actual service provided in relation to funding, bandwidth, processing time, packet overhead, or memory requirements. Providing resource managers, users, or applications with a costing framework for security services will enable them to make appropriate security selection according to their needs and the availability of resources. In some cases a user may decide to forego a certain level of security as a result of the impact it may have on bandwidth availability. Likewise a user may decide to increase security after consulting the costing framework. (NPS-CS-02-001, January 2002)

A costing framework as described above, see Figure 2.4, will require a matrix of cost expressions relative to security services and security attributes. Further granularity will be required to incorporate system resources including existing loads on the

processor, memory and network, as well as any limitations pre-established by security levels and network modes. (NPS-CS-02-001, January 2002) A costing framework model can provide the user with general information about the impact of certain security settings (network modes/security levels). The complexity of the costing calculation and reference mapping would remain transparent to the user.

The development of a detailed costing framework is beyond the scope of this thesis but is described here to lay the groundwork for future efforts.

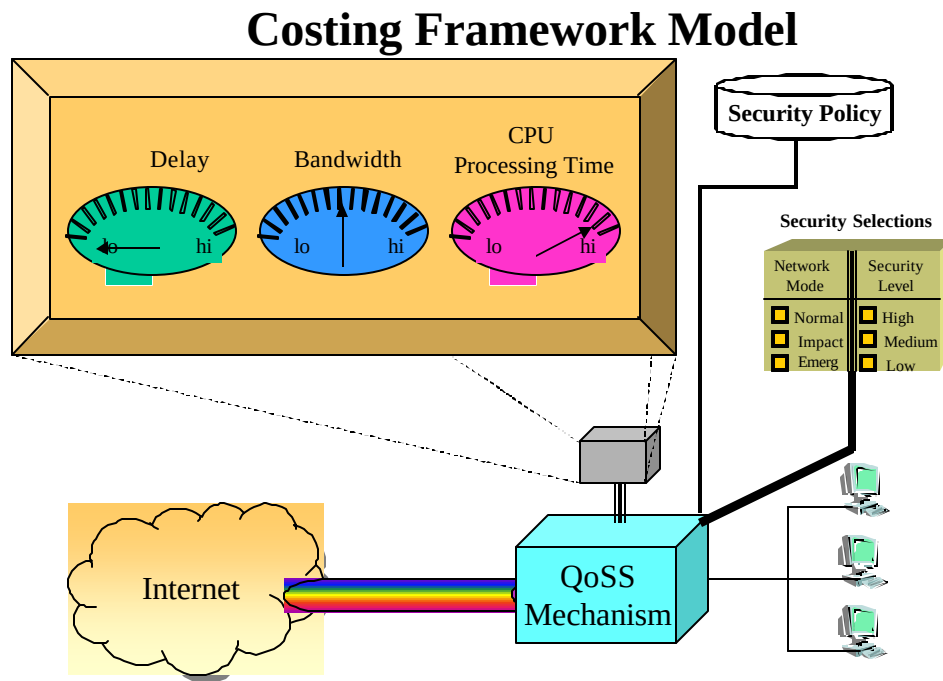


Figure 2.4. Costing Framework Model.

D. CONCLUSION

Quality of Security Service (QoSS) stems from the origins of Quality of Service (QoS), providing a mechanism to access security services in accordance with the user and the system requirements and the network environment. Security services can be defined in via various levels of granularity of integrity, confidentiality, non-repudiation, authentication, etc. To provide further integration of security mechanisms into the surrounding environment, dynamic parameters, i.e. environmental related variables with a defined range of values, can be mapped to adjustable levels of security protection in

accordance with policy. In order to manage the security mechanism's impact on resources and services, a costing framework is required to provide system and security managers with a tool for monitoring and adjusting security services.

Internet Protocol Security (IPsec), a security mechanism utilized to provide a range of security protection per packet at the network layer, is an ideal proof of concept platform to demonstrate how a specific security mechanism can be modulated to provide different levels of security response in accordance with QoSS. The following chapter will explain in detail the IPsec system architecture to provide the reader an understanding of the security mechanism.

III. SYSTEM ARCHITECTURE

A. INTRODUCTION

In this chapter I will discuss the origins and architecture of IPsec. The roots of IPsec spawned from the growth in popularity and dependency on network communications. It became quickly evident that all information sent across the Internet was susceptible to theft and modification. A means of protecting network packets soon became critical to most business and government organizations. The first attempt involved a simple “protect-all” approach to network security, Virtual Private Networks (VPN). However, with the added resource cost of network security on resources, this was not always the most efficient solution. IPsec introduced the ability to provide a range of security services through predefined set of security attributes ultimately defined by a security policy.

B. OVERVIEW OF IPSEC

The Internet Protocol (IP) has become the norm for network electronic communication in business, government and private life. However, the very definition of IP is fraught with vulnerabilities. IP packets have no inherent security. (Doraswamy, Naganand and Harkins, Dan, 1999, 41-55) Therefore, it is quite a trivial matter to spoof, modify and inspect an IP packet without authorization from the sender. IPsec was developed to address these problems by defining a security mechanism for sending data across an insecure medium.

Initial attempts at developing IP Security such as Virtual Private Networks (VPN), see Figure 3.1 resulted in very rigid security systems and a binary security choice: security or no security. There were no intermediate security choices. IPsec can provide users and systems with multiple selections of security attributes, which can enable, for example, the security mechanism to adapt to changing needs, all in accordance with the security policy.

IPsec vs. VPN Security Mechanisms

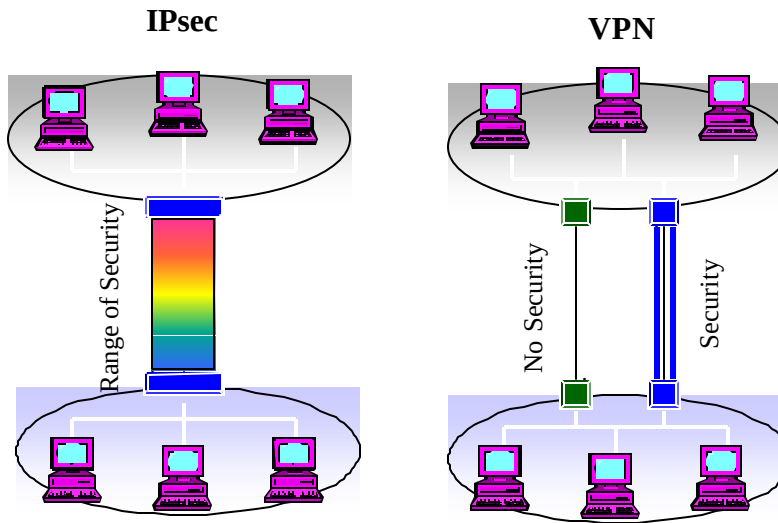


Figure 3.1. IPsec vs. VPN Security Mechanisms.

IPsec was designed to provide an efficient and effective cryptographic security mechanism for IP version 4 and IP version 6. Refer to Figure 3.2. The mechanism provides the following services: access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. These services are applied at the IP layer, providing security for IP and/or upper layer protocols. (Kent, S and Atkinson, R, 1998) The cryptographic algorithms are applied in accordance with system security policies that are defined within IPsec.

The IPsec implementation in OpenBSD version 2.8, researched in this thesis, consists of the following components (see Figure 3.2): Security Policy Database (SPD), Security Association Database (SAD), Internet Security Association and Key Management Protocol (ISAKMP), KeyNote, and Internet Key Exchange (IKE). A Security Association is defined as negotiated security terms for communications between two or more peers. The Security Policy Database (SPD) defines the authorized security associations. Security Association Database defines the security associations that have been established. The Internet Security Association and Key Management Protocol

(ISAKMP) define a framework for security association management and key negotiation. KeyNote is the trust management component that handles the mapping of policy to security attributes. Internet Key Exchange is the mechanism used to negotiate security associations with peers.

The IPsec as present in this research exists independent of ISAKMP, KeyNote and IKE (all described later) in our operating systems and environments. For the sake of consistency and relevancy to the research provided, I will cover only the IPsec version specific to OpenBSD version 2.8.

Changes to IPsec security variables and rules can be managed via Keynote. Local rules and policy are stored initially in the ISAKMP database (isakmpd.conf). During IPsec initialization, the information stored in isakmpd.conf is loaded into memory and cached in the SPD. Peer and service security associations and rules are stored initially in the Keynote Database and cached in the Security Association Database (SAD) as security associations (SA) once established through a successful peer negotiation (discussed later). (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79)

The IPsec process consists of two phases. Phase One involves the authentication among the peers and the negotiation of security parameters for Phase Two communications. The process of dynamically authenticating peers is managed according to the Internet Key Exchange (IKE) protocol. Phase Two is the actual authenticated and protect communication between peers. (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79)

IPsec can be used on a variety of system architecture models: host-to-host, gateway-to-gateway and gateway-to-host/host-to-gateway. (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79)

OpenBSD IPsec incorporates the concept of trust and security policy management by implementing KeyNote. The research performed in this thesis utilizes the OpenBSD IPsec mechanism as a model for discussion and implementation.

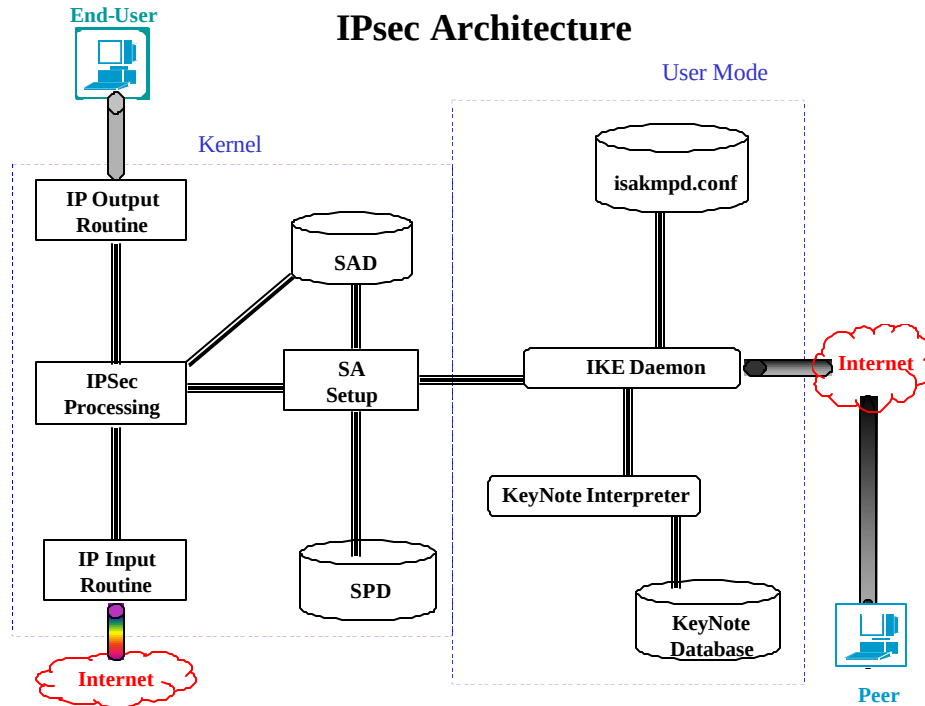


Figure 3.2. IPsec Architecture.
(After: Blaze, Matt, Ioannidis, John, and Kermoytis, Angelos D., February 2001)

C. IPSEC ARCHITECTURE

1. Security Services Provided by IPsec

The overall challenge of IPsec is to provide confidential, integrity, nonrepudiation and availability to network communications. IPsec provides the following security services that may be combined to meet these requirements: origin authentication, data integrity, data confidentiality, anti-replay protection and limited traffic confidentiality. It utilizes two methods to protect IP packets: Encapsulating Security Payload (ESP) providing data integrity, confidentiality and anti-replay protection; and Authentication Header (AH) providing only data integrity and anti-replay protection. In addition, IPsec can provide packet protection using transport or tunnel mode. Transport mode is used to protect upper level protocols. Tunneling mode is used to protect the entire packet (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79) Each mode can be used with either ESP or AH, so there are four possible IPsec packet formats, each of which will be explained in further detail below.

1.1 Encapsulated Security Protocol (ESP)

ESP provides data packets with confidentiality, integrity, source authentication and protection against replay attacks. This protection is accomplished by: (Doraswamy, Naganand and Harkins, Dan, 1999, 81-89)

- Inserting a new header after the IP Header but before the data of the packet,
- Appending a trailer to the packet
- Inserting IP protocol 50 to identify ESP. This protocol is used to enable Firewalls and routers to distinguish IPsec packets for processing and forwarding purposes.

Confidentiality and authentication are provided cryptographically. Encryption involves the conversion of data into unreadable form by using a reversible transformation (encryption algorithm i.e. DES, AES, IDE, etc.) Authentication is the process of cryptographic identity and integrity verification of transmitted data (authentication algorithms SHA MD5 and RIPEMD). [Cryptography and Network Security Principles and Practice] ESP can use any combination of encryption and authentication algorithms, as long as they are supported by the security mechanism. ESP can also be used without an encryption algorithm, or without an authentication algorithm. ESP without both encryption (NULL cipher) and authentication (NULL authenticator) algorithms, provides no security and results in a pointless drain of system resources, and therefore, typically is not supported. (Doraswamy, Naganand and Harkins, Dan, 1999, 81-89)

The ESP packet, ESP header, original packet, and ESP trailer are encrypted in the following manner: (see Figure 3.3) (Doraswamy, Naganand and Harkins, Dan, 1999, 81-89)

- The ESP header is not encrypted. This is done to allow for recipients and gateways (types of IPsec configurations discussed later) to process the packet according to IP Address (part of the Security Parameter Index, SPI).
- The original Data is encrypted
- The ESP Trailer is partially encrypted to allowing for data processing. The unencrypted portion on the trailer includes the Security Parameter Index (SPI) and destination IP address to enable SA identification, and the sequence number, authentication data and padding (if required). The justification for unencrypted portions will become clear in a moment.

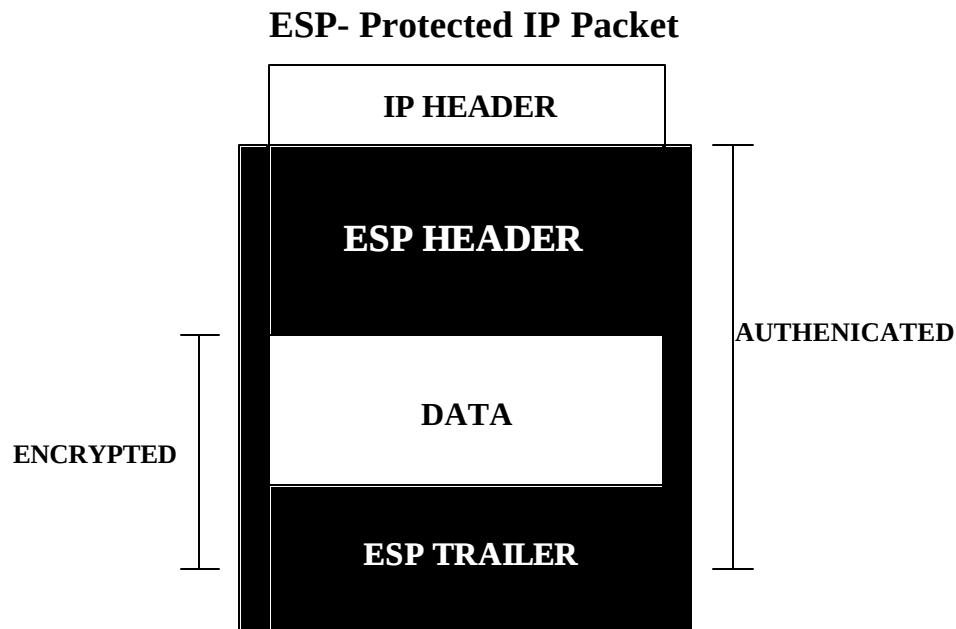


Figure 3.3. ESP- Protected IP Packet.
(After: Doraswamy, Naganand and Harkins, Dan, 1999, 49)

The order of processing for an ESP-protected packet on receipt is:

- Verify the sequence number
- Verify integrity
- Decrypt the data

ESP uses Cipher Block Chaining (CBC) and Initialization Vectors (IV) to strength its provided security. CBC is used to mask patterns of identical blocks within the same datagram. An Initialization Vector (IV) is a non-secret binary vector, which is different for every datagram, used as initialization input for encryption algorithms and to synchronize cryptographic equipment. By using CBC and IV, identical plaintext payloads will be encrypted to different cipher text payloads. The cipher text output is also random in appearance, a characteristic of a good cipher, and embeds the plaintext with the previous cipher ultimately strengthening the entropy of the following plaintext input to the cipher. See Figure 3.4. (Simpson, W. A., March 1999)

Encrypting and Decrypting using CBC and IV

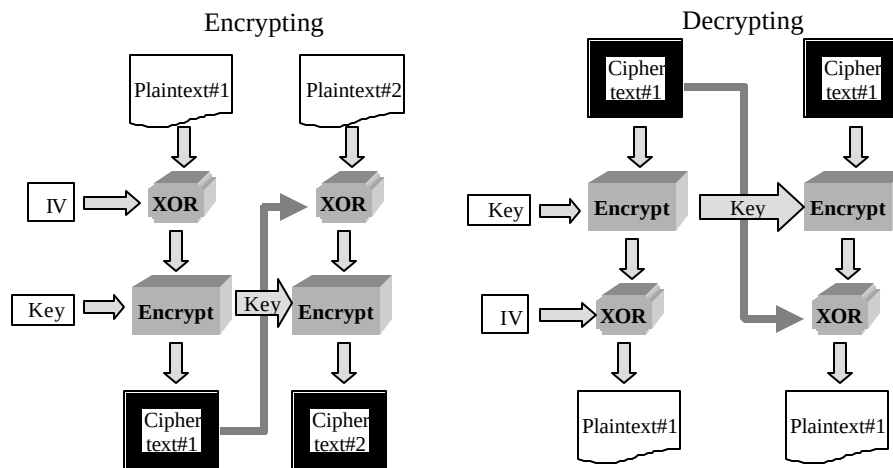


Figure 3.4. Encrypting and Decrypting using CBC and IV.
(After: Simpson, WA, March 1999)

A quick review of the CBC/IV encrypting and decrypting process follows.

The process of encrypting using CBC and IV is: (Simpson, W A, 1999, 1-3)

- An IV is used in an XOR operation with the first block of plaintext.
- The encryption key is used to encrypt the result of the XOR operation and generates a cipher text block.
- For the next successive blocks of plaintext, the previous ciphertext is XOR'd with the plaintext

Conversely, the process of decrypting using CBC and IV is simply the reverse:
(Simpson, W A, 1999, 1-3)

- The encryption key is used to decrypt the cipher text block. - An IV is used in an XOR operation with the first decrypt block of text.
- For the next successive blocks of cipher text, the previous cipher text is XOR'd with the decrypted block.

In transport mode, ESP protects the upper layer protocol by inserting an ESP header between the IP Header and the upper –layer protocol. The upper-layer protocol and data are then encrypted. (Doraswamy, Naganand and Harkins, Dan, 1999, 81-89) In

tunneling mode, ESP protects the entire data packet by embedding the packet in between the ESP header and trailer. A new IP header is then generated and added to the packet (Doraswamy, Naganand and Harkins, Dan, 1999, 81-89)

1.2 Authentication Header (AH)

The Authentication Header (AH) is utilized to maintain integrity of data, authenticated the sender, and provide (optionally) non-replay protection. However, it does not provide confidentiality. Structurally AH is simpler than ESP packets. It adds a header but no trailer to the data packet, and all information in the AH packet is unencrypted. There is no need for padding, a pad length indicator or an initialization vector. AH can be used alone within an SA or in conjunction with ESP in a separate SA.

AH packets are composed of the following: (see Figure 3.5) (Doraswamy, Naganand and Harkins, Dan, 1999, 91-98)

- Original IP Header
- AH Header. This contains the SPI which is used to locate the SA in the SADB during receipt processing, protocol field number 51, sequence number which is used to defend against replay attacks, and authentication data field (digest of keyed MAC).

AH- Protected IP Packet

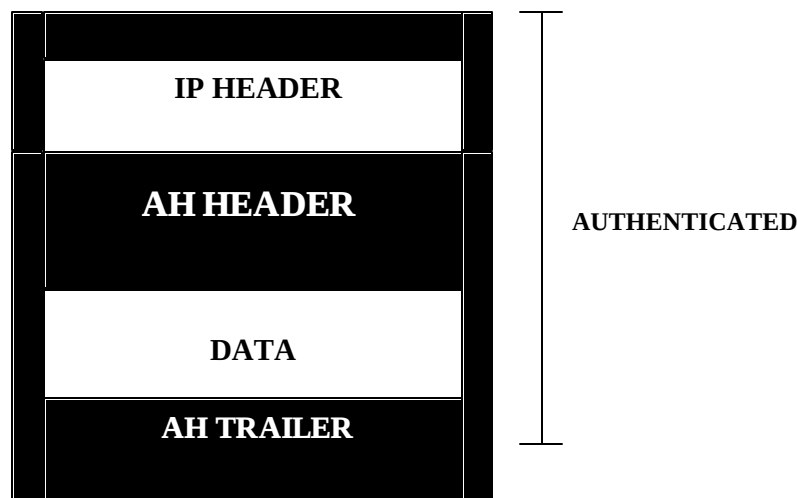


Figure 3.5. AH-Protected IP Packet.
(After: Doraswamy, Naganand and Harkins, Dan, 1999, 51)

In transport mode, AH protects end-to-end communications by inserting the AH header immediately following the IP Header and then authenticating the entire packet. By authenticating the entire packet, the recipient can be ensured that it came from the actual sender and was not captured, modified and resent. (Doraswamy, Naganand and Harkins, Dan, 1999, 91-98)

In tunneling mode, the AH encapsulates the protected datagram and adds an additional IP Header before the AH Header. (Doraswamy, Naganand and Harkins, Dan, 1999, 91-98)

1.3 IPsec Configurations

The material in this section is referenced from the following: (Doraswamy, Naganand and Harkins, Dan, 1999, 49-80)

IPsec can be utilized to support three network configurations as shown in Figure 3.6.

The peer-to-peer configuration is used to support secure communication between two IPsec end systems. In this situation, end systems would need a locally managed IPsec mechanism to manage the security mechanism and policy.

The peer-to-gateway configuration is used to support an IPsec standalone user communicating securely to an IPsec gateway or router. An IPsec gateway or router is an IPsec mechanism that supports multiple users or an internal network.

The gateway-to-gateway configuration is used to support secure communication between two or more IPsec gateways or routers.

IPsec Configurations

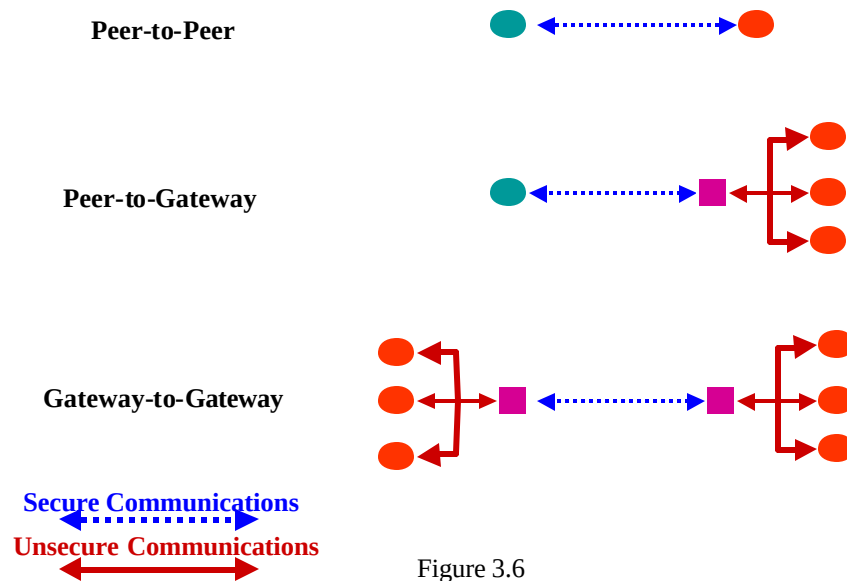


Figure 3.6

Figure 3.6. IPsec Configurations.

1.4 Use of Transport and Tunnel Modes

The material in this section is referenced for the following: (Doraswamy, Naganand and Harkins, Dan, 1999, 49-80)

As mentioned earlier, IPsec utilizes two modes for transport and tunneling. Refer to Figure 3.7.

Transport mode is used typically in the peer-to-peer configurations. There is no attempt to protect the identity of the sender and / or receiver when using this mode. The security emphasis is on the upper level protocols.

Tunnel mode is typically used with peer-to-gateway, gateway-to-peer, and gateway-to-gateway configurations. The security emphasis is on protecting (confidentiality and/or integrity) the entire packet including the sender/receiver identity. The original packet source and destination IP addresses are protected via encryption and/or authentication (by means of insuring integrity). An additional IP source/destination header is provided to the appropriate IPsec tunnel recipient. At the gateway, the packet is unwrapped and the inner packet is forwarded to the ultimate end

system recipient.

IPsec Transport and Tunnel Modes

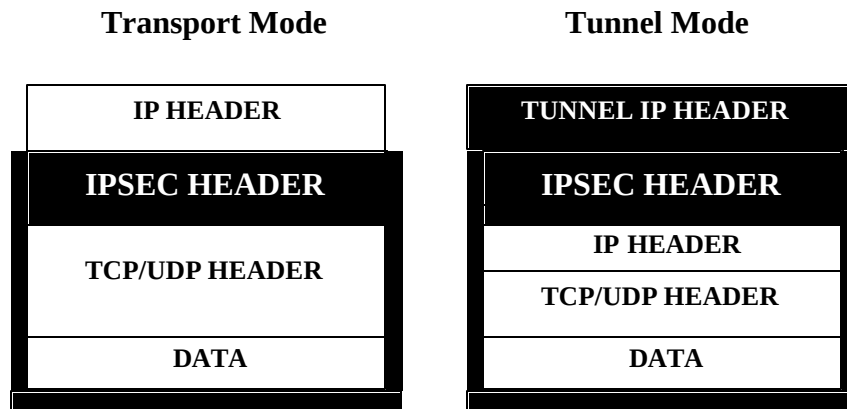


Figure 3.7. IPsec Transport and Tunnel Modes.
(After: Doraswamy, Naganand and Harkins, Dan, 1999, 57-80)

1.5 Security Associations (SA)

A Security Association (SA) can be referred to as a “simplex” connection path established to provide security services to IP packets. The “simplex” path requires the SA to represent either an AH or an ESP but not both, and only for one communication direction. If additional services are desired another SA will be required. Likewise other communications paths (such as a return path to the originator) will require a separate SA. As a result, SA’s are uniquely identified by the security parameter index (SPI), ESP or AH security protocol and the destination IP address. (Kent, S and Atkinson, R, November 1998) A secure communications path may require multiple SA’s to ensure all requirements are met.

IPsec requires that: (Kent, S and Atkinson, R, November 1998)

- A host or end system MUST support both transport and tunnel mode.
- A security gateway is required to support only tunnel mode.

1.6 Combining Security Associations

It is important to note that the IPsec mechanism only allows one security protocol, AH or ESP to be used with an SA. However, there may be times when a security policy calls for a combination of security services for a specific communications path that requires more security attributes than is possible with only one SA. Therefore multiple SAs are required to properly achieve the desired level of security as mandated by the security policy. The use of multiple SAs is referred as a *security association bundle* or *SA bundle*. The order of SA implementation sequence (ESP or AH first) is important and is defined by the security policy. (Leiseboer, John, 2001)

There are two ways to implement an SA bundle: transport adjacency and iterated tunneling.

Transport adjacency is the process of using more than one security protocol on the same IP datagram without utilizing tunneling. Only one level of combination is allowed. Additional nesting will not result in further security since all the IPsec processing will be performed at the same instance at the destination. Figure 3.8 provides an example of transport adjacency where first ESP provides confidentiality of IP data using encryption. Then AH is used to add authentication to the datagram. When using transport adjacency, the ordering of the applied SA bundle is important. As with the provided example, if AH and ESP are used in conjunction, AH should be used as the first header after IP after ESP security has been applied. This is justified by the fact that data integrity should be performed on as much packet data as possible to achieve the desired security effect of packet authentication. (Leiseboer, John, 2001)

Iterated tunneling is the process of applying multiple layers of security protocols through IP tunneling. This enables multiple levels of nesting. With each tunnel being able to originate or terminate at a different IPsec sites along the secure communications path. Figure 3.9 show an example of iterated tunneling, both security end points are different. This AH SA protects all traffic flowing between the two gateways. Host A and Host B are provided security via an ESP SA. With iterated tunneling, various orderings of AH and ESP are possible and sensible when bundling SAs. (Leiseboer, John, 2001)

Transport Adjacency

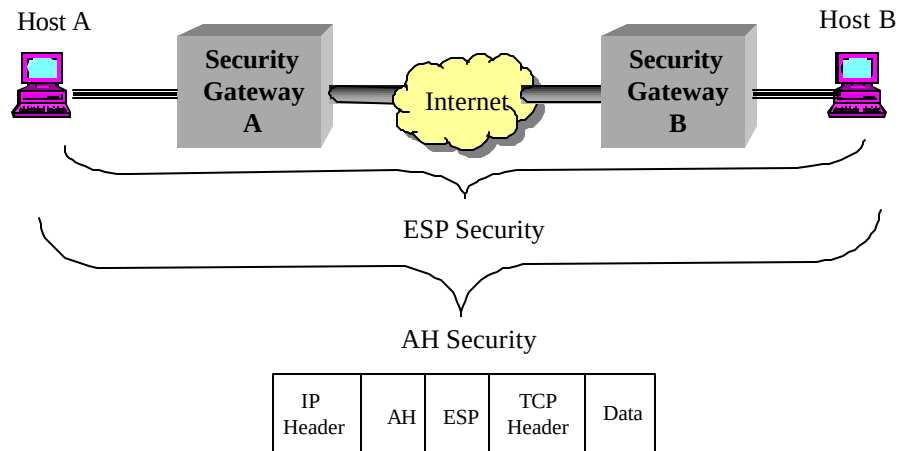


Figure 3.8. Transport Adjacency.
(After: Leiseboer, John, 2001)

Iterated Tunneling

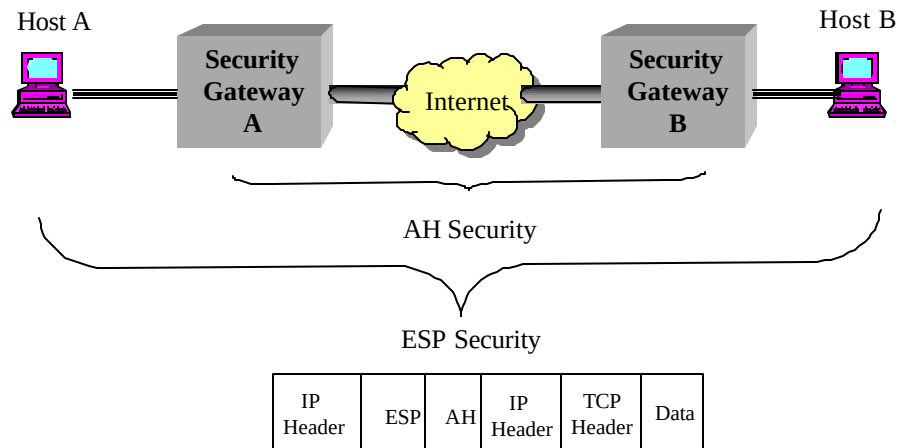


Figure 3.9. Iterated Tunneling.
(After: Leiseboer, John, 2001)

1.7 Security Policy

A security policy defines the rules and regulations of a system. Specifically what users are allowed to access to systems, applications, and data, what times during the day are they allowed access (security system administrator may define only working hours access to users), and what actions are users authorized to perform on the data (permissions: read, write and/or execute). The challenge is to present a seamless approach to controlling each user's actions, in regards to the security policy. The system security policy, for the purposes of this thesis, can be defined in terms of IPsec security parameters. (Blaze, Matt, Ioannidis, John, and Keromytis, Angelos D, February 2001)

Security Policy is represented in IPSec via the following: (see Figure 3.10)

- Application Layer: (Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D, February 2001)
 - <ISAKMPD.CONF> – initial peer security policy configuration used in IKE Phase I IKE & initial Phase II SA negotiation
 - < KeyNote/ISAKMPD.Policy > – Local security policy defined in Keynote semantics – queried during IPSec negotiation.
- Kernel Layer:
 - Security Policy Database (SPD)– cached security policy
 - Security Association Database (SAD) – cached valid security association (SA)

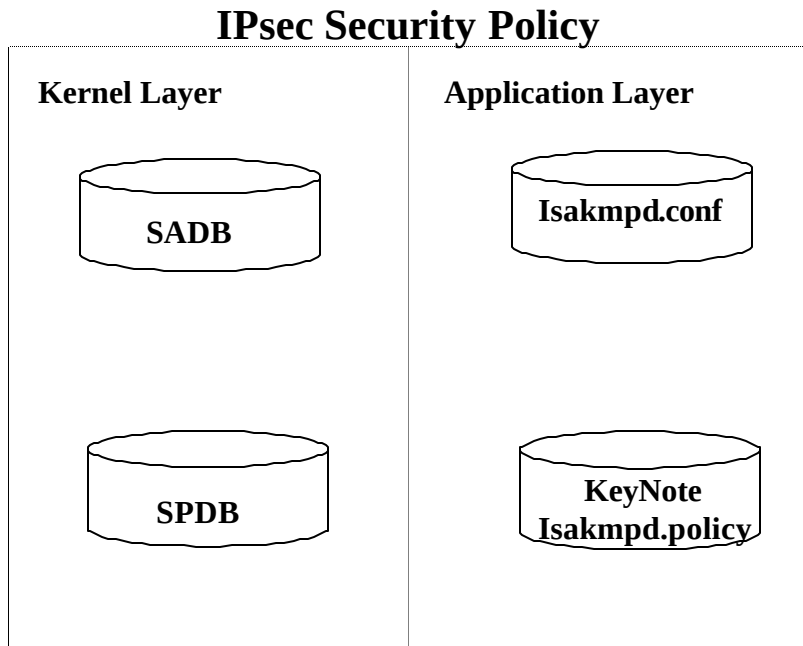


Figure 3.10. IPsec Security Policy.
(After Blaze, Matt, Ioannidis, John and Keromytis, Angelos D., February 2001)

1.8 Security Policy Database (SPD)

The IPsec mechanism manages security policy efficiently by implementing a Security Policy Database (SPD) in the Kernel to allow for quick reference by input and output processing modules (discussed later). The Security Policy Database (SPD) is implemented and maintained by a user or system administrator, or by an application constrained by pre-defined rules or policy. In general, packets are either afforded IPsec security services, discarded, or allowed to bypass IPsec, based on the applicable security policies found in the SPD. (Kent, S and Atkinson, R, November 1998) SPD is populated/updated by either manual keying or daemons (IKE/ISAKMPD/Photuris). See Figure 3.11.

IPsec Security Policy Database (SPD) Populating Mechanisms

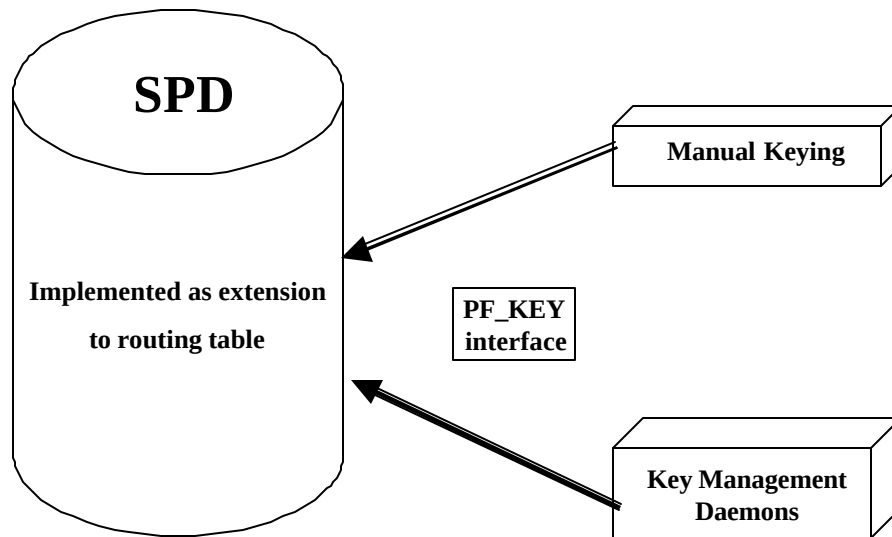


Figure 3.11. IPsec Security Policy Database (SPD) Populating Mechanisms.
(From: NPS-CS-02-003, January 2002)

Using one of these methods, the system's security policy is translated into database entries. The entries define the data traffic to be protected, what security mechanism should be used, and with whom the system is authorized to speak. For each packet entering or leaving the system, the SPD is queried to ensure the proper security parameters and measures are applied. (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79)

An SPD entry specifies one of three possible actions regarding all traffic that matches that entry:

- Discard, implying that the packet should not be allowed in or out.
- Bypass, implying that no security services should be applied nor should any security services be expected.
- Protect, implying security services are required for outbound and inbound packets.

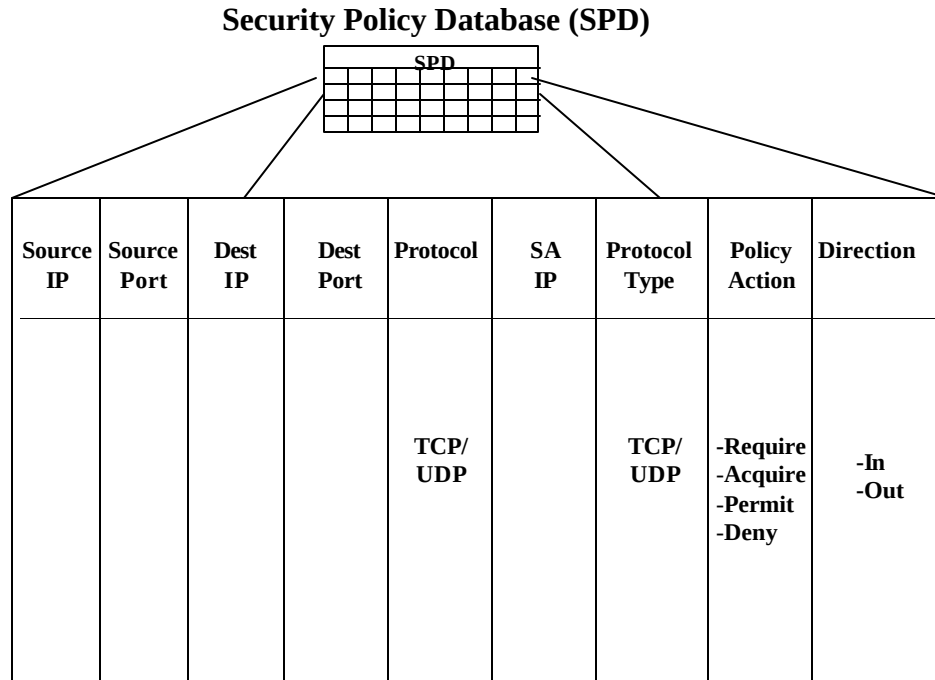
IPsec policy is mapped to IP traffic via "selectors". Information stored in SPD may be located by any of the following search selectors: (see figure 3.12) source IP address, destination IP address, Name (specific user or system), Protocol, and upper layer

ports. (Doraswamy, Naganand and Harkins, Dan, 1999, 57-79) Selectors can be either specific values, ranges of allowed values or wild cards. (Kent, S and Atkinson, R, November 1998)

The following are the in the SPD:

- Source IP – the source IP address in accordance with the security policy.
- Source Port – the source port to which the security policy applies.
- Destination IP – the destination IP address in accordance with the security policy.
- Destination Port – the destination port to which security policy applies.
- Name – used to bind a policy to a specific user or system.
- SA IP – pointer to active SAs
- Protocol Type – TCP/UDP
- Policy Action:
 - Require – Strict condition. If no SA exists, drop the packet.
 - Acquire – Set-up SA and continue communications without protection until IPSEC SA takes effect.
 - Permit – Bypass IPSEC process– exception for specific packet characteristics.
 - Deny – Drop the packet without further processing.

In general, the first qualifying SPD entry found will be used to determine the disposition of the current packet. The ability to use different selectors in the SPD allows IPsec systems to provide flexible policy mechanisms.



Figure

3.12. Security Policy Database (SPD).
(From: NPS-CS-02-003, January 2002)

1.9 Security Association Database (SAD)

After an SA is established for communication with a remote host, the SA is “cached” in the SAD. Each entry in the SAD represents one SA. Refer to Figure 3.13. The SAD is first consulted for both inbound and outbound traffic to determine processing requirements for the packets. If no SA is found (e.g. expired or non-existing SA in the SAD) Phase II IKE negotiations are initiated with the remote host. (Kent, S and Atkinson, R, November 1998)

Inbound packets are processed by indexing SA’s in the SAD with each of the following packet fields: (Kent, S and Atkinson, R, November 1998)

- Outer Header's Destination IP address: the IPv4 or IPv6 Destination address.
- IPsec Protocol: AH or ESP, specifies the IPsec protocol to be applied to the traffic on this SA.
- SPI: the 32-bit value used to distinguish among

SAs terminating at the same destination and using the same IPsec protocol.

Outbound packets are processed by using security parameter indexes (SPI), to index SA's in the SAD and the SPD. This is done in order to allow SA bundles, in which a policy entry in the SPD involves multiple SAs in a specific order. (Kent, S and Atkinson, R, November 1998)

An entry in the SAD will contain the value or values, which were created during peer negotiations. These fields can have the form of specific values, ranges, wildcards, or an "OPAQUE" (inaccessible due to fragmentation or encryption). The ESP and AH Protocol fields from an entry may contain NULL for one or the other but not for both. These values are used by the sending peer to determine the SA's required for packet processing. The receiving peer uses these values to check that the inbound packet adheres to the receiver's security policy. (Kent, S and Atkinson, R, November 1998)

The following SAD fields are used in IPsec processing:

- Sequence Number Counter: a 32-bit value used to generate the Sequence Number field in the AH or ESP headers.
- Sequence Counter Overflow: a flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent transmission of additional packets on the SA.
- Anti-Replay Window: a 32-bit counter and a bit-map used to determine whether an inbound AH or ESP packet is a replay.
- AH Authentication algorithm, keys, etc.
- ESP Encryption algorithm, keys, IV mode, IV, etc.
- ESP authentication algorithm, keys, etc. If the authentication service is not selected, this field will be null.
- Lifetime of this Security Association: a time interval after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur.
- IPsec protocol mode: tunnel, transport or wildcard.

- Path MTU: any observed path MTU and aging variables.

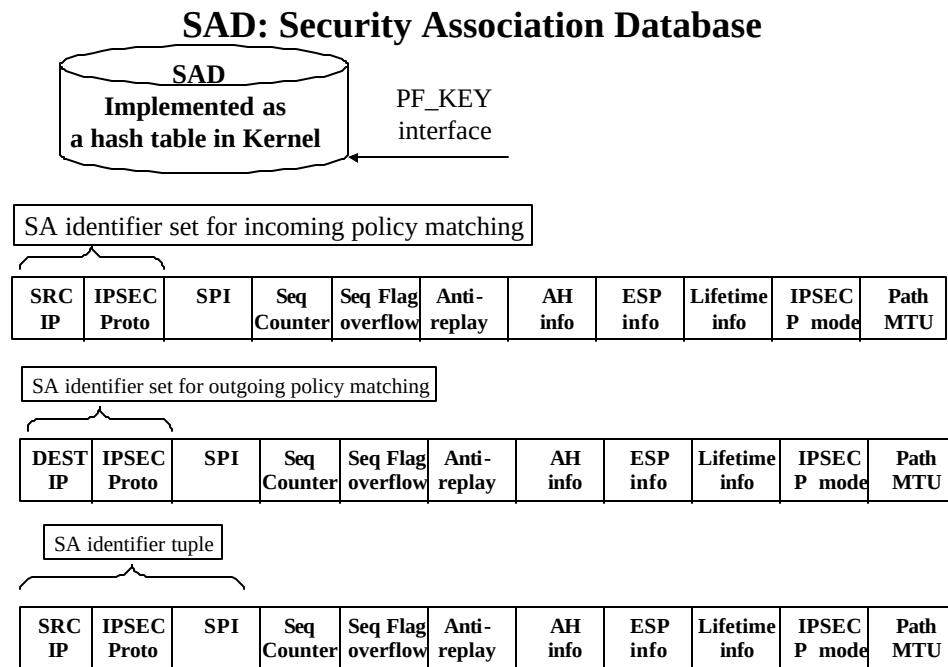


Figure 3.13. SAD: Security Association Database.
(From: NPS-CS-02-003, January 2002)

1.10 Selectors

“Selectors” are parameters that are used to locate or select a SA or SA bundle in the SPD and SAD. A selected SA (or SA bundle) may be very detailed or general, depending on the selectors used. Traffic between two peers/gateways may only require a single SA in each direction for a uniform set of services. Or traffic between peers/gateways may utilize a series of SA’s to handle different security services for different supported applications. The following are the IPsec selector parameters. (Kent, S and Atkinson, R, 1998)

- Source Address – can be an address range, network prefix, wild card or a specific address.
- Destination Address – can be an address range, network prefix, wild card or a specific address.
- Name – used to identify a policy associated with an authorized user or system.

- Protocol – the transport protocol.
- Upper-Layer Ports – may use individual UDP or TCP ports or wild cards
- Data Sensitivity Level

The following table describes the possible selector expression combinations in SAD and SPD: (Kent, S and Atkinson, R, November 1998)

Field	Traffic Value	SAD Entry	SPD Entry
src addr	single IPaddr	single range,wild	single range, wild
dst addr	single IPaddr	single range,wild	single range,wild
xpt protocol	Xpt protocol	single range,wild	Single, wildcard
src port	Single src port	single range,wild	Single, wildcard
dst port	Single dest port	single range,wild	Single, wildcard
user id	Single user id	single range,wild	Single, wildcard
sec. labels	Single value	single range,wild	Single, wildcard

Table 3.1. Possible Selector Combination.
(After: Kent, S and Atkinson, R, 1998)

1.11 IPsec Processing Modules

The IPsec mechanism uses input and output modules, a processing module and a SA set-up module to receive, send, and process IPsec packets, and setup SA as required, respectively. The following sections will describe modules.

1.11.1 IPsec Input Module

IPsec's input routine provides an interface and buffer to all incoming message traffic. The packets are filtered accordingly and are either (1) passed to the appropriate upper level process or internal user, (2) used to generate an IPsec negotiation with a peer or, (3) discarded. See Figure 3.14.

The following is a pseudo code representation of the IPsec input

routine module: (Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D., February 2001)

- Receives IP packets are received from external source.
 - Queries Security Policy Database to determine whether to forward the packet or discard: Query SPD(Packet's source IP, source port, destination IP, destination port, Protocol, IPSEC Protocol type):
 - If "Require" then set "Require" Flags and forward packet to IPSEC Processing module.
 - If "Acquire" then set Acquire flags and forward to IPSEC processing module.
 - If "Permit" (non-IPSEC packet) then forward to appropriate higher-layer protocol.
 - If "Deny" then discard packet.
 - If Null (no entry in SPD) and IPSEC Packet then discard.
 - If Null (no entry in SPD) and non-IPSEC Packet forward to appropriate higher-level protocol.

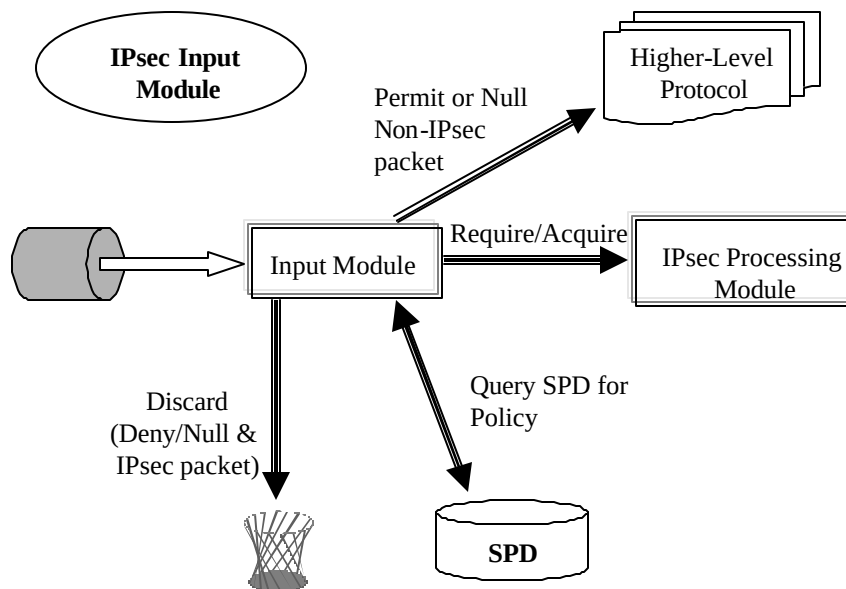


Figure 3.14. IPsec Input Module.

1.11.2 IPsec Output Routine

IPsec's output routine provides an interface and buffer for all outgoing message traffic. The packets are filtered and processed in one of three ways: provide IPsec protection, provide no protection, or discard. See Figure 3.15.

The following is a pseudo code representation of the IPsec input routine module: (Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D., February 2001)

- Packets are received from Higher/Upper Layer Applications.
- Queries Security Policy Database to determine whether IPsec packet protection required for packet - Query SPD(Packet's source IP, source port, destination IP, destination port, Protocol):
 - If "Require" then set "Require" Flags and forward packet to IPsec Processing module.
 - If "Acquire" then set Acquire flags and forward to IPsec processing module.
 - If "Permit" (non-IPsec packet) then forward to network for transmission.
 - If "Deny" then discard packet.
 - If Null (no entry in SPD) then forward to network for transmission.

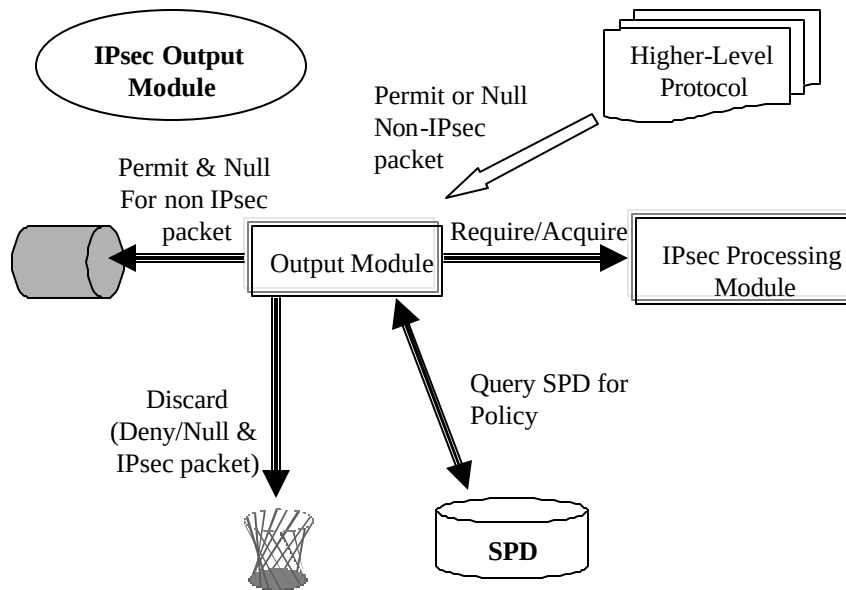


Figure 3.15. IPsec Output Module.

1.11.3 IPsec Processing Module

The IPsec processing module provides an interface for the input/output modules, the SAD and the SA set-up module. After IPsec packets are verified and approved by the SPD, they are forwarded to the IPsec processing module for further processing. For incoming packets, the SAD is queried to determine whether or not an SA exists in the SAD. If not, the IPsec peer negotiation phase is queued. If an SA exists the IPsec processing module utilizes the information provided by the existing SA to remove the IPsec protection from the packet for further processing. Similarly for outgoing packets, the SAD is queried, possibly triggering IPsec peer negotiation and finally using existing SAs to encapsulate the packet with the appropriate IPsec protection. (Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D, February 2001)

The following is pseudo code of the IPsec processing module:

- Receives IP packets from the Input/Output module.
- Query SAD (Packet's source/destination IP, Security Parameter Index (SPI), incoming/outgoing).
- If "SA Exists" and "incoming" then process IPsec packet resulting in a

de-capsulated IP Packet. Forward packet back to the IPsec Input module for further processing.

- If “SA Exists” and the packet is “outgoing” then process the IP packet resulting in an encapsulated/IPsec protected IP Packet. Forward IPsec packet to external network for routing.
- If “No SA Exists” then forward packet to SA set-up

1.11.4 SA SetUp Module

The IPsec SA set-up module provides an interface between the SAD, SPD, KeyNote DB, and IKE processing module. The SA set-up module is involved in the IPsec SA generation process. The SA setup module is triggered when it receives an IP packet from the IPsec processing module. (Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D, February 2001)

The following pseudo code represents the SA set-up module:

- If packet is “Incoming” then
 - Performs a double check routine and if SA exists for packet then packet is dropped (should have been determined earlier in process suspect situation)
 - Otherwise: IKE Daemon triggered.
- Awaits completion of IKE process.
- Updates SPD and SAD with peer communications security associations and policies.
 - Discards original unprotected packet.
- If packet “Outgoing” then
 - Queries SPD for relevant policy:
 - If policy is found then trigger the IKE Daemon.
 - Otherwise drop the packet
 - Awaits completion of the IKE process.
 - Updates the SPD and SAD with peer communications security associations and policies.
- Discards the original unprotected packet.

1.12 Internet Key Exchange (IKE)

The Internet Key Exchange provides IPsec with a means of performing automated SA creation for network peer communication. Specifically, IKE is an automated protocol for generating, negotiating, and creating Security Associations (SA) between network peers. (Savolainen, Sampo, 1999)

IKE is a hybrid protocol developed from the following protocols:
(Savolainen, Sampo, 1999)

- Oakley key exchange and
- Skeme key exchange
- Key Management Protocol (ISAKMP) framework.

Without IKE, IPsec would require costly manual SA generation seriously limiting the system ability to function at a dynamic level. (Savolainen, Sampo, 1999)

IKE can be triggered by the following specific events (See figure 3.16)
(Savolainen, Sampo, 1999)

- Remote peer negotiation initiation
- Timer scheduled events
- Kernel – PF_Key upcalls for new SA/ expired Sas. A PF_Key is key management kernel interface used to trigger an IKE/ISAKMP daemon.
- User – signals or first-in first out queuing (FIFO)

Controlling events for IKE

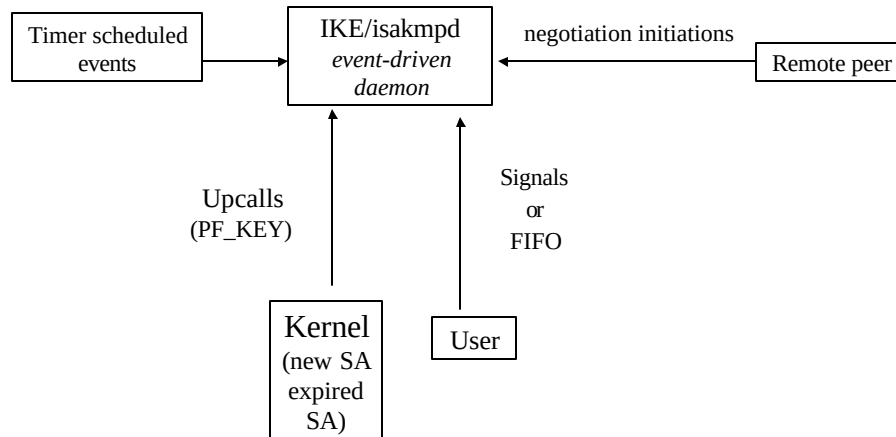


Figure 3.16. Controlling Events for IKE.
(From: NPS-CS-02-003, January 2002)

1.13 Internet Security Association And Key Management Protocol (ISAKMPD)

The following section is references (Maughan, D., Schertler, M., Schneider M., Turner J, November 1998).

The Internet Security Association and Key Management Protocol (ISAKMP) defines the required payload for exchanging key generation and authentication data between negotiating IPsec peers The ISAKMP daemon, ISAKMPD, defines the mechanics of implementing a key exchange protocol, and the negotiation of a security association.

ISAKMPD defines how peers:

- Communicate
- Construct their messages
- Establish state transitions required for secure communications.

In order for peers to communicate within the confines of the IPsec, they

must first negotiate on a Security Associations (SA). This negotiation is performed via the Internet Key Exchange (IKE). ISAKMPD defines the specifics and the syntax required to complete the negotiation.

There are two parts to IPsec's security processing as defined by ISAKMPD (See figure 3.17)

- Security Association Negotiations. Peers negotiate to agree on the security association that will define their secure communication.
- Security Association Processing. SAs are utilized for secure communication until they are deleted, modified, or expire.

IPsec Security Association Process Defined by ISAKMPD

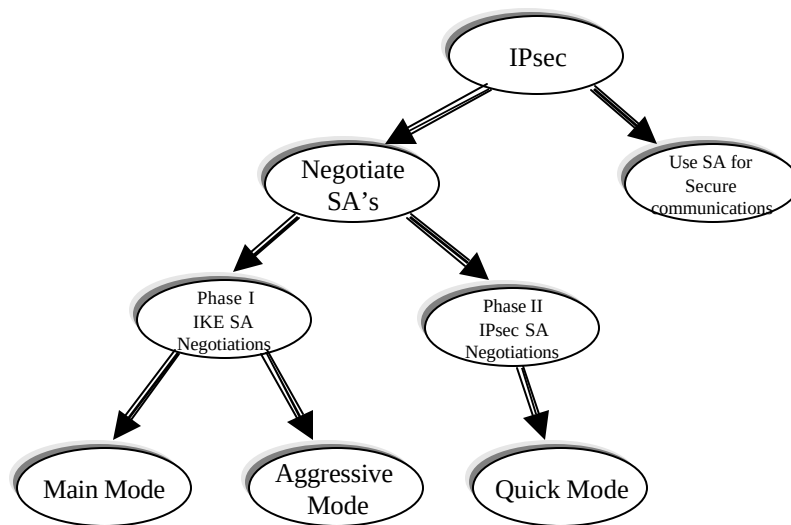


Figure 3.17. IPsec Security Association Process Defined by ISAKMPD.

Security association negotiations utilizes a two phases for security negotiations:

- **Phase I** – peers negotiate for IKE security associations (SA) (where none currently exists). There are two choices for Phase I: Main-Mode and Aggressive-Mode

- **Main-Mode** – protects the identity of peers by sending a sequence of

authentication information. (Refer to figure 3.18)

- Utilizes 6 messages:

- **Messages 1-2:** used for negotiating the security policy for the exchange. Sent in the clear.
- **Messages 3-4:** used for Diffie-Hellman keying material exchange.
- **Messages 5-6:** used for authenticating the peers with signatures or hashes and optional certificates. Sent encrypted.

IKE Phase I - Main Mode

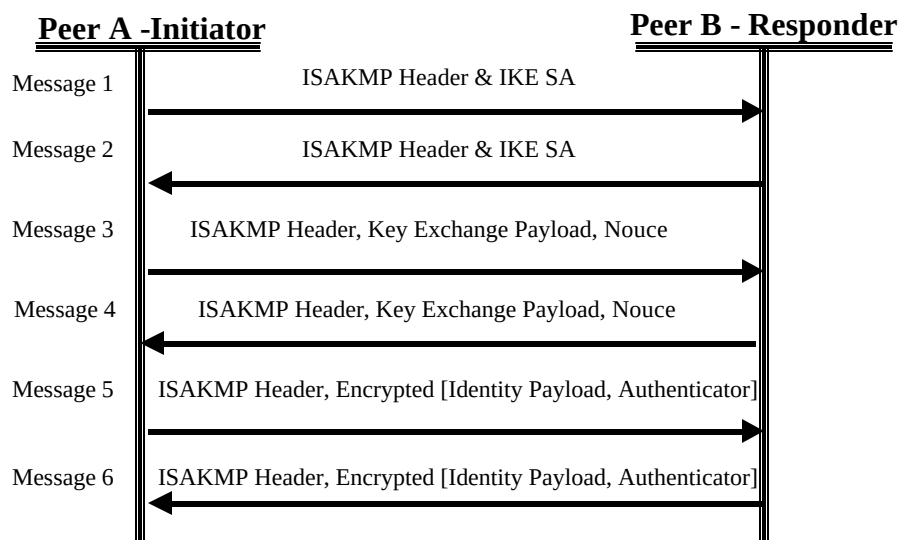


Figure 3.18. IKE Phase I – Main Mode.

(After: Maughan, D., Schertler, M., Schneider M., Turner J, November 1998) -

Aggressive Mode- does not protect the identity of peers and sends all authentication information at the same time. This mode is used when bandwidth is a concern. (Refer to Figure 3.19)- It utilizes three messages:

- **Message 1:** proposes the policy, and passes data for key-exchange, a nonce and some information for identification. Sent unencrypted.

- **Message 2:** a response, which authenticates the responder and concludes the policy and key-exchange. Sent in the clear.
- **Message 3:** used for authenticating the initiator and provides a proof of participation in the exchange. Encrypted.
- **Note:** The identity of the responder cannot be protected, but by encrypting the last message the identity of the initiator is protected.

IKE Phase I - Aggressive Mode

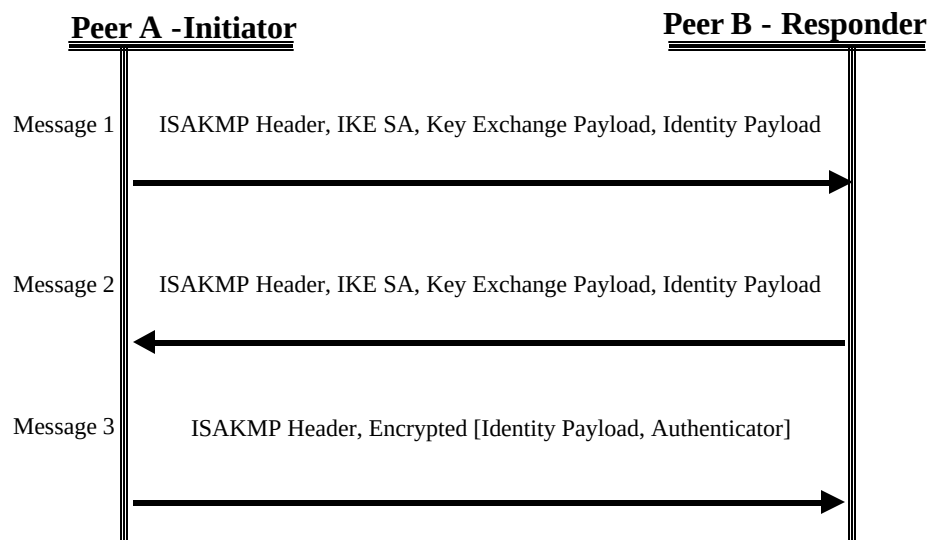


Figure 3.19. IKE Phase I Aggressive Mode.
(After: Maughan, D., Schertler, M., Schneider M., Turner J, November 1998)

- **Phase II** – peers negotiate for IPsec security associations (SA) or new keying material using the previously established IKE SA's for protection. There is only one mode for phase two: Quick Mode.

- Quick Mode is bound to Phase I in that it relies on Phase I to establish a valid IKE SA to protect IPsec SA negotiation. Quick mode is used to derive keying material and negotiate shared policy for non-ISAKMP SAs between peers. No peer authentication is required since Phase I establishes peer identities bound to security

associations. Exchange of keys to determine how the data between peers will be encrypted – establishing IPSec SAs. All the payloads except ISAKMP header are encrypted. A Diffie-Hellman key exchange may be done to achieve perfect forward secrecy (PFS). PFS means that an IPSec SA's key was not derived from any other secret. This ultimately strengthens the overall security of the exchange. Many SAs can be negotiated during one Quick Mode exchange. Either one of the parties might initiate the quick mode exchange regardless of who initiated the first phase.

- Quick Mode uses three messages:

- **Message 1:** Contains the ISAKMP header (unencrypted) and then includes proposed SA(s), identification & authentication information (for both sender and receiver) and nonce information (all encrypted)
- **Message 2:** Contains the ISAKMP header (unencrypted) and then includes responding proposed SA(s), identification & authentication information (for both sender and receiver) and nonce information (all encrypted)
- **Message 3:** Contains the ISAKMPD header (unencrypted) and a verifying Hash (encrypted).

IKE Phase II - Quick Mode

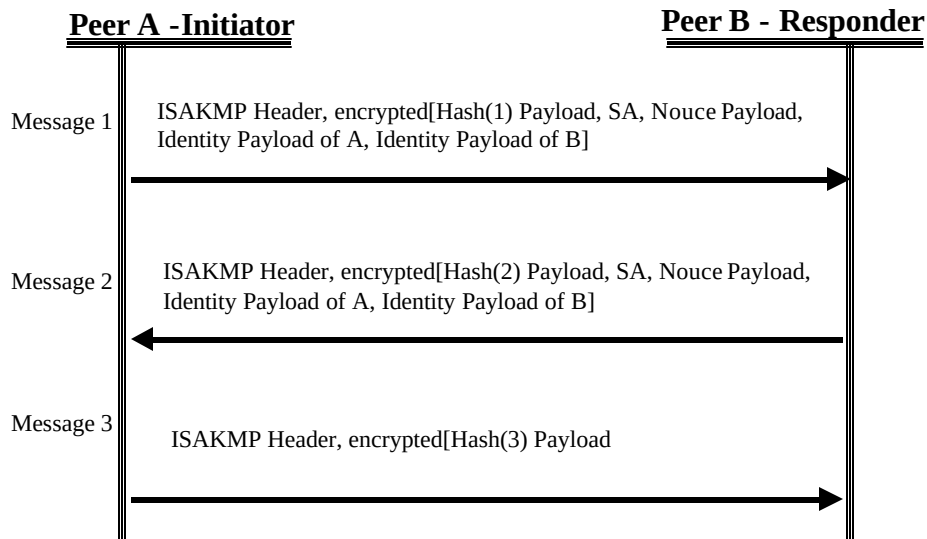


Figure 3.20. IKE Phase II – Quick Mode.
(After: Maughan, D., Schertler, M., Schneider M., Turner J, 1998)

1.14 isakmpd.conf

The following section contains information referenced from (ISAKMP.CON(5), OpenBSD Programmer's Manual, October 1998). **isakmpd.conf** is the configuration file for the **isakmpd** daemon. It provides the initial security associations and keys used for IPsec peer negotiations.

Once the IPsec process determines that Phase I negotiations are required, **isakmpd.conf** is queried to establish a secure communication channel to perform communication negotiations for Phase II. **Isakmpd.conf** provides the IKE Daemon with both Phase I and Phase II security proposals that are cached in memory until the IPsec process is restarted or reinitialized.

Isakmpd.conf utilizes the traditional **.ini** style file construct. The file is broken down into sections indicated by “[]”. Within each section parameters are defined using the <tag> = <tag value or range value>“. Tag values may consist of other section names. This results in a tree-like recursive structure. Unless a tag and tag value is a reserved **isakmpd.conf** word, it is defined by declaring the tag or tag value as a section later in the file. This continues until all tags and tag values have been defined in

terms of reserved words or values.

Many of the Tag fields have default values that are loaded from default structures in the **isakmpd.conf** code in the event that no value is present. However for completeness and portability it is recommended that implementers provide specific definitions to ensure functionality and policy enforcement.

1.14.1. isakmpd.conf Parameters

The following section contains information referenced from (ISAKMP.CON(5), OpenBSD Programmer's Manual, October 1998).

The following is a description of typical parameters found in the **isakmpd.conf** file- **General Section**

This section contains the global configuration parameters. The typical ones used are:

- *Policy-file*: Keynote policy file. Default is `"/etc/isakmpd/isakmpd.policy"`.
- *Default-Phase-2-Suites* - A list of Phase 2 suites that will be used when establishing dynamic SAs. Default is : QM-ESP-3DES-SHA-PFS-SUITE.
- *Retransmits*: Number of times a negotiation is retransmitted.
- *Check-interval*: Interval between "watchdog" checks of connections required to be up at all times.
- *Exchange-max-time*: Maximum time in seconds for an exchange to setup before aborting.
- *Listen-on*: IP-addresses to listen on.
- *Shared-SAD* Defined if multiple instances can be executed on top of one SAD
- *Pubkey-directory*: Path to directory holding public keys. The default directory is: `"/etc/isakmpd/pubkeys"`.

- Phase I Section

This section is used to establish ISAKMP negotiation SAs

. The following are typical tags used in this section:

- *IP-address*: The name of the ISAKMP peers at the given IP-address.

- *Default*: The name of the default ISAKMP peer.

Note: The name defined here will be used as a section name later on in the **isakmpd.conf** file.

- Phase II Section

This section is used to define the IPsec negotiation SAs.. The following are typical tags used in this section:

- *Connections*: List of IPsec "connection" names that should be established automatically during daemon startup.

Note: These names are section names where further parameter information is defined. See Section <IPsec-connection> below.

- *Passive-connections*: The list of IPsec "connection" names recognized and initialized.

- Keynote Section

This section is used to establish Keynote dependency parameters. The following are typical tags used in this section:

- *Credential-directory*: Directory, which contains directories for Ids. This in turn contains the files named ``credentials" and ``private_key".

- X509-Certificate Section

This section is used to establish Certificate dependency parameters. The following are typical tags used in this section:

- *CA-directory*: Directory for PEM certificates of

certification authorities which are trusted to sign other certificates.

- *Cert-directory*: Directory for PEM certificates that are trusted to be valid.
- *Accept-self-signed*: Defines certificates not originating from a trusted CA that will be accepted.
- *Private-key*: Private key match for certificate public key.

- Referred-to sections

- *_ISAKMP-peer_*: Negotiation parameters for ISAKMP peer
- *Phase*: The constant 1.
- *Transport*: Name of the transport protocol. The default is UDP.
- *Port*: Optional. If UDP used, the UDP port number. The default value is 500, which is the IANA-registered number for ISAKMP.
- *Local-address*: The local IP-address.
- *Address*: IP-address of the peer.
- *Configuration*: The name of the ISAKMP-configuration section. See <ISAKMP-configuration>.
- *Authentication*: If available, authentication data for this specific peer.
- *ID*: If available, the name of the section that describes the local client ID. If not available, the default value is the address of the local interface where packets are being sent to at the remote daemon. See <Phase1-ID>.
- *Remote-ID*: If available, section name that describes the remote client ID. If not available, it defaults to the address of the remote daemon. See <Phase1-ID>.
- *Flags*: List of flags specific to further handling of ISAKMP SA. Currently no specific ISAKMP SA flags are defined.

- PhaseI-ID

- *ID-type*: The ID type as defined by the RFCs. For Phase I: IPV4_ADDR, IPV4_ADDR_SUBNET, FQDN, USER_FQDN, or KEY_ID.
- *Address*: If the ID-type is IPV4_ADDR, then this tag should exist with an IP address.
- *Network*: If the ID-type is IPV4_ADDR_SUBNET this tag should exist with a network address.
- *Netmask*: If the ID-type is IPV4_ADDR_SUBNET this tag should exist with a network subnet mask.
- *Name*: If the ID-type is FQDN, USER_FQDN, or KEY_ID, this tag should exist with domain name, user@domain, or other identifying string.

- ISAKMP-configuration

DOI: The domain of interpretation as defined by the RFCs. Default is IPSEC.

EXCHANGE_TYPE: The exchange type as defined by the RFCs. ID_PROT is used for main mode and AGGRESSIVE is used for aggressive mode

Transforms: A list of proposed transforms to use for protecting the ISAKMP traffic.

- ISAKMP-transform

- *ENCRYPTION_ALGORITHM*: The encryption algorithm defined by RFCs or ANY to indicate that any encryption algorithm proposed will be accepted.
- *KEY_LENGTH*: Used for encryption algorithms with variable key length.
- *HASH_ALGORITHM*: The hash algorithm as defined by RFCs, or

ANY.

- *AUTHENTICATION_METHOD*: The authentication method as defined by RFCs, or ANY.

- *GROUP_DESCRIPTION*: Symbolic group names used for Diffie-Hellman exponentiations, or ANY. The names include:

MODP_768, MODP_1024, EC_155 and EC_185.

- *PRF*: The algorithm for the keyed pseudo-random, or ANY.

- *Life*: A list of lifetime descriptions, or ANY.

Lifetime

- *LIFE_TYPE*: SECONDS or KILOBYTES depending on the type of the duration. This field may NOT be set to ANY.

- *LIFE_DURATION*: An offer value, a minimum acceptable value, and a maximum acceptable value. Can also be set to ANY.

IPsec-connection

- *Phase*: The constant 2.

- *ISAKMP-peer*: ISAKMP-peer name used to establish a connection. The value is the name of an <ISAKMP-peer> section.

- *Configuration*: IPsec-configuration section name.

See <IPsec-configuration>.

- *Local-ID*: If present, the name of the section that describes the optional local client ID. See <IPsec-ID>.

- *Remote-ID*: If present, this is the name of the section that describes the optional remote client ID presented to the peer. See

<IPsec-ID>.

- *Flags*: A list of flags controlling the further handling of the IPsec SA. Currently only one flag is defined:

- *Active-only* If this flag is present and the <IPsec-connection> is part of the phase 2 connections, it will not automatically be used for accepting connections from the peer.

IPsec-configuration

- *DOI* : The domain of interpretation as defined by the RFCs. Normally IPSEC. If unspecified, defaults to IPSEC.
- *EXCHANGE_TYPE*: The exchange type as defined by the RFCs. For quick mode this is QUICK_MODE.
- *Suites*: A list of protection suites (bundles of protocols) available for protecting the IP traffic. Each of the list elements is a name of an <IPsec-suite> section.

IPsec-suite

- *Protocols*: List of the protocols included in the protection suite. Each of the list elements is a name of an <IPsec-protocol> section.

IPsec-protocol

- *PROTOCOL_ID*: The protocol as defined by the RFCs. Acceptable values include: IPSEC_AH and IPSEC_ESP.
- *Transforms*: List of transforms usable for implementing the protocol. Each of the elements is a name of an <IPsec-transform> section.
- *ReplayWindow*: The window size used for replay protection. Normally not adjusted.

IPsec-transform

- *TRANSFORM_ID*: The transform ID as defined by the RFCs.
- *ENCAPSULATION_MODE*: The encapsulation mode as defined by the RFCs. Normal values include: TRANSPORT or TUNNEL.
- *AUTHENTICATION_ALGORITHM*: The optional authentication algorithm for the ESP transform.

- *GROUP_DESCRIPTION*: An optional (provides PFS if present) Diffie-Hellman group description. The values are the same as GROUP_DESCRIPTION's in <ISAKMP-transform> sections described above.
- *Life*: List of lifetimes, each element is a <Life-time> section name.

IPsec-ID

- *ID-type*: The ID type as defined by the RFCs. The current value for IPsec is IPV4_ADDR or IPV4_ADDR_SUBNET.
- *Address*: If the ID-type is IPV4_ADDR, then an IP address should be listed.
- *Network*: If the ID-type is IPV4_ADDR_SUBNET, then an IP address should be listed.
- *Netmask*: If the ID-type is IPV4_ADDR_SUBNET, then a network subnet mask should be listed.
- *Protocol*: If the ID-type is IPV4_ADDR or IPV4_ADDR_SUBNET, then this tag indicates which transport protocol should be transmitted over the SA. If left unspecified, all transport protocols between the two address (ranges) will be sent (or permitted) over that SA.
- *Port*: If the ID-type is IPV4_ADDR or IPV4_ADDR_SUBNET, then this tag indicates which source or destination port is allowed to be transported over the SA (depending on whether this is a local or remote ID). If left unspecified, all ports of the given transport protocol will be transmitted (or permitted) over the SA. The Protocol tag must be specified in conjunction with this tag.

1.14.2. isakmpd.conf Example

The following section contains information referenced from

(ISAKMPD.CONF (5), OpenBSD Programmer's Manual, October 1998)

The following is an example **isakmpd.conf** file. The file provides for communication between two peers using both the ESP and AH protocols. Specifically it loads two security proposals. One for ESP using AES for an encryption algorithm, SHA for encryption authentication, and Tunnel mode, and enforcing Perfect Forward Security. The other for AH, using SHA for an authentication algorithm, tunnel for the transport mode and perfect forward security (PFS). Notice how after the required sections of [General] and [Phase 1], the rest are dependent on previous entries in the mandatory sections. Such as [Peer-131.120.8.95/131.120.8.91]

A configuration sample for the isakmpd ISAKMP/Oakley (aka IKE) daemon.

[General]

Listen-on= 10.1.0.2

Policy-file= /etc/isakmpd/isakmpd.policy

Retransmits= 3

Exchange-max-time= 120

[Phase 1]

10.1.0.1= ISAKMP-peer-west

[Phase 2]

Connections= IPsec-east-west

[ISAKMP-peer-west]

Phase= 1

Local-address= 10.1.0.2

Address= 10.1.0.1

Configuration= Default-main-mode

Authentication= mekmitasdigoat

[IPsec-east-west]

Phase= 2

ISAKMP-peer= ISAKMP-peer-west

Configuration= Default-quick-mode

Local-ID= Net-east

Remote-ID= Net-west

[Net-west]

ID-type= IPV4_ADDR_SUBNET

Network= 192.168.1.0

Netmask= 255.255.255.0

[Net-east]

ID-type= IPV4_ADDR_SUBNET

Network= 192.168.2.0

Netmask= 255.255.255.0

Main mode descriptions

[Default-main-mode]

EXCHANGE_TYPE= ID_PROT

Transforms= 3DES-SHA

Quick mode descriptions

[Default-quick-mode]

EXCHANGE_TYPE= QUICK_MODE

Suites= QM-ESP-AES-SHA-PFS-SUITE,QM-AH-SHA-PFS-SUITE

KeyNote credential storage

[KeyNote]

Credential-directory= /etc/isakmpd/keynote/

```

# Certificates stored in PEM format
[X509-certificates]
CA-directory=      /etc/isakmpd/ca/
Cert-directory=    /etc/isakmpd/certs/
Private-key=       /etc/isakmpd/private/local.key

# Main mode transforms
#####

[3DES-SHA]
ENCRYPTION_ALGORITHM= 3DES_CBC
HASH_ALGORITHM= SHA
AUTHENTICATION_METHOD= PRE_SHARED
GROUP_DESCRIPTION= MODP_1024
Life= LIFE_3600_SECS

# Quick mode protection suites
#####

# AES

[QM-ESP-AES-SHA-SUITE]
Protocols= QM-ESP-AES-SHA

# AH
[QM-AH-SHA-PFS-SUITE]
Protocols= QM-AH-SHA-PFS

# Quick mode protocols

# AES

```

```
[QM-ESP-AES-SHA-PFS]
PROTOCOL_ID= IPSEC_ESP
Transforms= QM-ESP-AES-SHA-PFS-XF
```

```
# SHA
```

```
[QM-AH-SHA-PFS]
PROTOCOL_ID=      IPSEC_AH
Transforms=      QM-AH-SHA-PFS-XF
```

```
# Quick mode transforms
```

```
# AES
```

```
[QM-ESP-AES-SHA-PFS-XF]
TRANSFORM_ID=  AES
ENCAPSULATION_MODE=  TUNNEL
AUTHENTICATION_ALGORITHM=  HMAC_SHA
GROUP_DESCRIPTION=  MODP_1024
Life= LIFE_3600_SECS
```

```
# AH
```

```
[QM-AH-MD5-PFS-XF]
TRANSFORM_ID=      SHA
ENCAPSULATION_MODE=  TUNNEL
GROUP_DESCRIPTION=  MODP_768
Life=  LIFE_3600_SECS
```

```
[LIFE_3600_SECS]
LIFE_TYPE=          SECONDS
LIFE_DURATION=      3600,1800:7200
```

1.14.3 isakmpd.conf Process

The IKE daemon reads the **isakmpd.conf** during IPsec initialization to load Phase I and Phase II security proposals, representing the local security policy, in memory. The IKE daemon is initialized during system start or when a “reinitialized” system call is executed (by either the kernel, upper-level process or a user). Once the security proposals are loaded in memory, the IPsec process can initialize the IKE Daemon to begin IPsec peer negotiations for Phase I and Phase II. (See figure 3.21

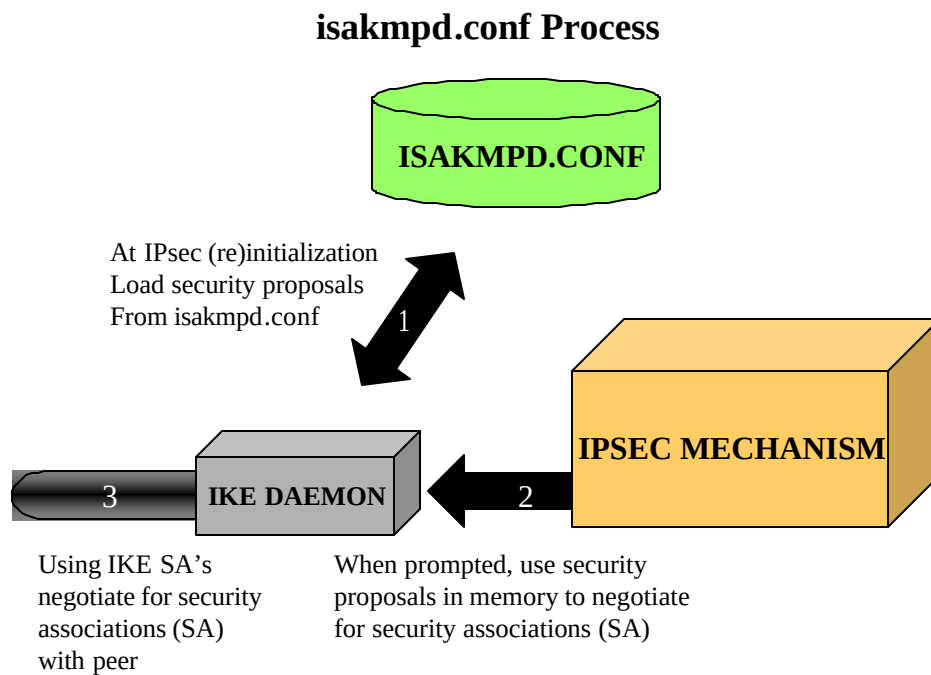


Figure 3.21. isakmpd.conf Process.

1.15 Security Policy – KeyNote

The following section contains information referenced from (Blaze, Matt, Feigenbaum, Joan and Keromytis, Angelos D., April 1998)

A security mechanism requires a policy to define the security requirements and establish rules and parameters. Security policies may have a variety of levels of interpretation, from human language descriptions to the fine granular specifications

describing encryption and authentication methods and key lengths. Of course, accurate mapping is required throughout the layers of the interpretation to ensure specific requirements are not inadvertently modified or disregarded during translation.

Previous versions of IPsec utilized static security policies. This implied that a security policy is invoked on the security mechanism prior to system initialization and then remains unchanged until system is taken off-line to make appropriate policy adjustments.

The OpenBSD version of IPsec utilizes a “trust management” infrastructure – Keynote. Specifically, KeyNote provides a mechanism for defining local policies used by IPsec in negotiating SA’s between peers. Keynote provides a straightforward syntax for defining both security credentials and local security policies. Credentials are a means of identifying specific network “principals” (users, hosts, etc.). The policies and credentials are combined to form “assertions”. The assertions define actions authorized by or for specified key holders. By signing the assertions, they may be safely transmitted across “untrusted” networks. The assertions are divided into three sections:

- Authorizer identity – may involve local policy or signed key for credentials.
- Key predicate – key(s) being authorized
- Action predicate – action being authorized

A respondent IPsec node having received a security proposal from an initiator IPsec node, would first perform a query on its local KeyNote using the syntax of an assertion to find out if the initiator’s proposed security proposal is allowed. KeyNote would compare the proposal to its stored policy assertions and credentials to determine whether the proposed actions are valid in accordance with the local policy. KeyNote would reply to the respondent IPsec node with a Boolean response. The respondent IPsec node would either establish a secure communications path with the initiator (Boolean response true) or not (Boolean response false). (see Figure 3.22)

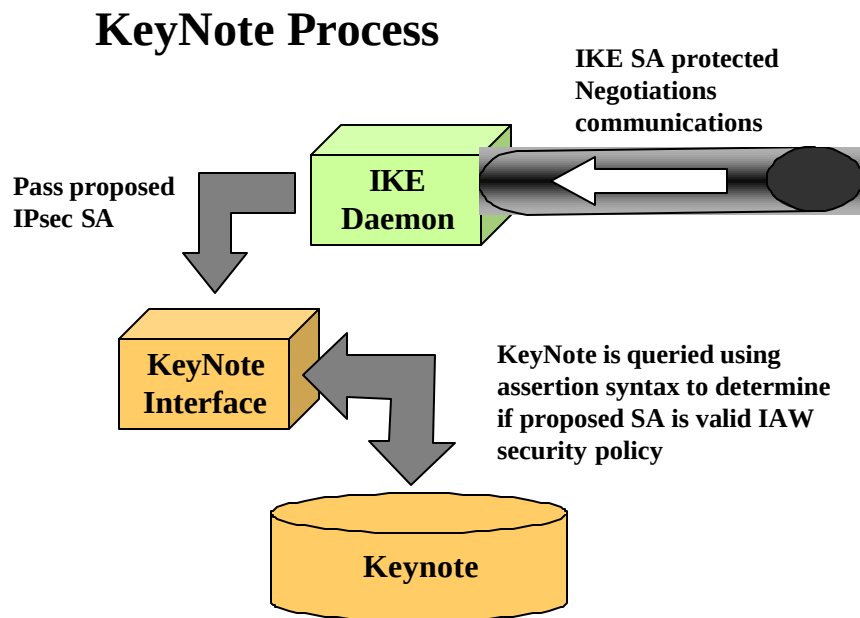


Figure 3.22. KeyNote Process.

1.15.1. isakmpd.policy

The following section contains information referenced from (KEYNOTE(5), OpenBSD Programmer's Manual Pages, October 1999)

KeyNote policy assertions in OpenBSD IPsec are defined in **isakmpd.policy**.

The assertions are divided into field sections using the tag: value syntax. The following are authorized field sections for KeyNote:

- Authorizer – only mandatory field. Identifies authorizer
- Comment
- KeyNote Version
- Licensees
- Local-Constraints
- Signatures

- Conditions – contains the action/policy predicates in accordance with the security policy.

1.15.2 Condition Attributes

The following section contains information referenced from (KEYNOTE(5), OpenBSD Programmer's Manual Pages, October 1999)

The following attributes are currently defined in the condition assertion section: (Note that in KeyNote/**iskamdpolicy** all values are in lowercase)

- **app_domain**: Always requires *IPsec policy*.
- **Domain of Interpretation (Doi)** : Always defined as *ipsec*.
- **Initiator**: *yes* if the local daemon is initiating the Phase II SA, *no* otherwise.
- **phase_1**: *aggressive* or *main* dependent on Phase I mode requirement.
- **PFS**: *yes* if a Diffie-Hellman exchange will be performed during phaseII/ Quick Mode, *no* otherwise.
- **ah_present, esp_present, comp_present**: *yes* if an AH, ESP, or compression proposal was received respectively, *no* otherwise.
- **ah_hash_alg**: *md5*, *sha*, *ripemd*, or *des*, based on the hash algorithm. Defines the generic transform to be used in the AH authentication.
- **esp_enc_alg**: *des*, *des-iv64*, *3des*, *rc4*, *idea*, *cast*, *blowfish*, *3idea-iv32*, *rc4*, *null*, or *aes*, based on the encryption algorithm specified in the ESP proposal.
- **comp_alg**: *oui*, *deflate*, *lzs*, or *v42bis*, based on the compression algorithm specified in the compression proposal.
- **ah_auth_alg**: *hmac-md5*, *hmac-sha*, *des-mac*, *kpdck*, or *hmac-ripemd*. Based on the authentication method specified in the AH proposal.

- **esp_auth_alg**: *hmac-md5, hmac-sha, des-mac, kpdk, or hmac-ripemd*. Based on the authentication method specified in the ESP proposal.
- **ah_life_seconds, esp_life_seconds, comp_life_seconds**: Lifetime of the AH, ESP, and compression proposal, in seconds. If none listed the corresponding attribute will be set to zero.
- **ah_life_kbytes, esp_life_kbytes, comp_life_Kbytes**: Lifetime of the AH, ESP, and compression proposal, in kbytes of traffic. If none listed the corresponding attribute will be set to zero.
- **ah_encapsulation, esp_encapsulation, comp_encapsulation**: *tunnel or transport*.
- **comp_dict_size**: log2 maximum size of the dictionary, according to the compression proposal.
- **comp_private_alg**: Integer specifying the private algorithm in use, according to the compression proposal.
- **ah_key_length, esp_key_length**: Number of key bits to be used by the authentication and encryption algorithms respectively
- **ah_key_rounds, esp_key length**: Number of rounds of the authentication and encryption algorithms respectively (for variable round algorithms).
- **ah_group_desc, esp_group_desc, comp_group_desc**: The Diffie-Hellman group identifier from the AH, ESP, and compression proposal, used for PFS during Quick Mode. Valid values are 1 (768-bit MODP), 2 (1024-bit MODP), 3 (155-bit EC), 4 (185-bit EC), and 5 (1536-bit MODP).
- **phase1_group_desc**: The Diffie-Hellman group identifier used in IKE Phase I.
- **remote_filter_type, local_filter_type, remote_id_type**:

IPv4 address, IPv4 range, IPv4 subnet, IPv6 address, IPv6

range, IPv6 subnet, FQDN, User FQDN, ASN1 DN, ASN1 GN, or Key ID, based on the Quick Mode Initiator ID, Quick Mode Responder ID, and Main Mode peer ID respectively

- **remote_filter_addr_upper, local_filter_addr_upper, remote_id_addr_upper:** For filter_type *IPv4 address* or *IPv6*

address, they contain the respective Ipv4 or Ipv6 address. For *IPv4 range* or *IPv6 range*, these contain the upper end of the address range. For *IPv4 subnet* or *IPv6 subnet*, they contain the highest address in the specified subnet

- **remote_filter_addr_lower, local_filter_addr_lower, remote_id_addr_lower:** For filter_type is *IPv4 address* or *IPv6*

address, these contain the respective address. For *IPv4 range* or *IPv6 range*, these contain the lower end of the address range. For *IPv4 subnet* or *IPv6 subnet*, these contain the lowest address in the specified subnet.

- **remote_filter, local_filter, remote_id:** For filter_type of an address range or subnet, these are set to the upper and lower part of the address space separated by a dash ('-') character (if the type specifies a single address, they are set to that address). For FQDN and User FQDN types, these are set to the respective string. If the Key ID payload contains non-printable characters then the hexadecimal representation of the associated byte string (lower-case letters) is used. Otherwise, they are set to the respective string. For ASN1 DN, these are set to the text encoding of the Distinguished Name in the payload sent or received. The format is the same as that used in the Licensees field.

- **remote_filter_port, local_filter_port, remote_id_port:** Transport protocol port.

- **remote_filter_proto, local_filter_proto, remote_id_proto:** *etherip, tcp, udp*, or the transport protocol number, depending on the transport protocol set in the IDci, IDcr, and Main Mode peer ID respectively.
- **remote_negotiation_address:** IPv4 address of the remote IKE daemon.
- **local_negotiation_address:** IPv4 address of the local interface used by the local IKE daemon for this exchange.
- **GMTTimeOfDay:** UTC date/time, in YYYYMMDDHHmmSS format.
- **LocalTimeOfDay:** local date/time, in YYYYMMDDHHmmSS format.

1.15.3 Condition Predicate Syntax

The condition section utilizes a form of predicate logic to state assertions for evaluating the security proposal. The following are specific syntax rules: (ISAKMPD.POLICY(5), OpenBSD Programmer's Manual, October 1998) (Blaze, Matt Ioannidis, J and Keromytis, Angelos, February 2001)

- No blank lines are allowed
- Assignment operator is “==”
- Logical And operator is “&&”
- Logical Or operator is “||”
- Parenthesis may be used to provide further granularity to statements.
- Assertion must be terminated with “=> true”

1.15.4 Example isakmpd.policy

The following is an example KeyNote **isakmpd.policy** file. Notice that the following operations are authorized:

- Telnet (port 23) using AES as the encryption algorithm, SHA as the

encryption authentication algorithm and “tunnel” as the transportation mode.

- Finger (port 79) using AH as the protection protocol, SHA as the authentication algorithm, and “tunnel” as the transportation mode.

KeyNote-Version: 2

Authorizer: "POLICY"

Licensees: "passphrase:mekmitasdigoat"

Conditions: app_domain == "IPsec policy" &&

((esp_present == "yes") && (esp_encapsulation == “tunnel”) &&

((local_filter_port == "23") ||

(remote_filter_port == "23")) &&

(esp_enc_alg == "aes")) ||

((ah_present == "yes") && (ah_encapsulation == “tunnel”) &&

((local_filter_port == "79") ||

(remote_filter_port == "79")) &&

(ah_auth_alg == "hmac-sha"))

-> "true";

D. ANALYSIS

The following is a consolidated description of the IPsec processes presented in section B .

1. IPsec (Re)Initialization

As mentioned earlier in the **isakmpd.conf** section, only during IPsec startup and re-initialization is the **isakmpd.conf** read and are security proposals loaded into memory.

2. IPsec Output Processing

The following is the step-by-step processing of an outbound packet through an IPsec mechanism: (Blaze, Matt, Ioannidis, J and Angelos D. Keromytis, February 2001) Refer to Figure 3.23.

- 1) A packet arrives from a high-level protocol.
- 2) The SPD is consulted to determine if the packet requires IPSec protection.
- 3a) If protection is not required, the packet is forwarded to the external network.

- 3b) If IPSec protection is required the packet is forwarded to IPSec processing module where
- 4) The SAD is consulted for SA specifics for the packet.
- 5a) If an SA exists for the packet, the appropriate security transformation are applied to the packet and it is forwarded to the external network.
- 5b) If no SA exists the SA management module is triggered.
- 6) The SA set-up module consults the SAD to verify that no SA exists.
- 7a) If one does, the packet is dropped.
- 7b) The KeyNote Database is consulted using packet information via the KeyNote interpreter to determine if the packet should be accepted, dropped or needs IPSec protection. Note: This step does not occur in the OpenBSD version 2.8 of IPsec.
- 8) In the event of the requirement for IPSec protection, the IKE daemon is initiated.
- 9) IKE then negotiates security parameters with the distant peer. The distant peer will reply with a proposed setting of secure communication parameters.
- 10) The Keynote database will be consulted to ensure that distant peer parameters comply with local policy.
- 11) If all is compliant, SA may be created updating the SAD and SPD.
- 12) At completion the original packet is discarded.

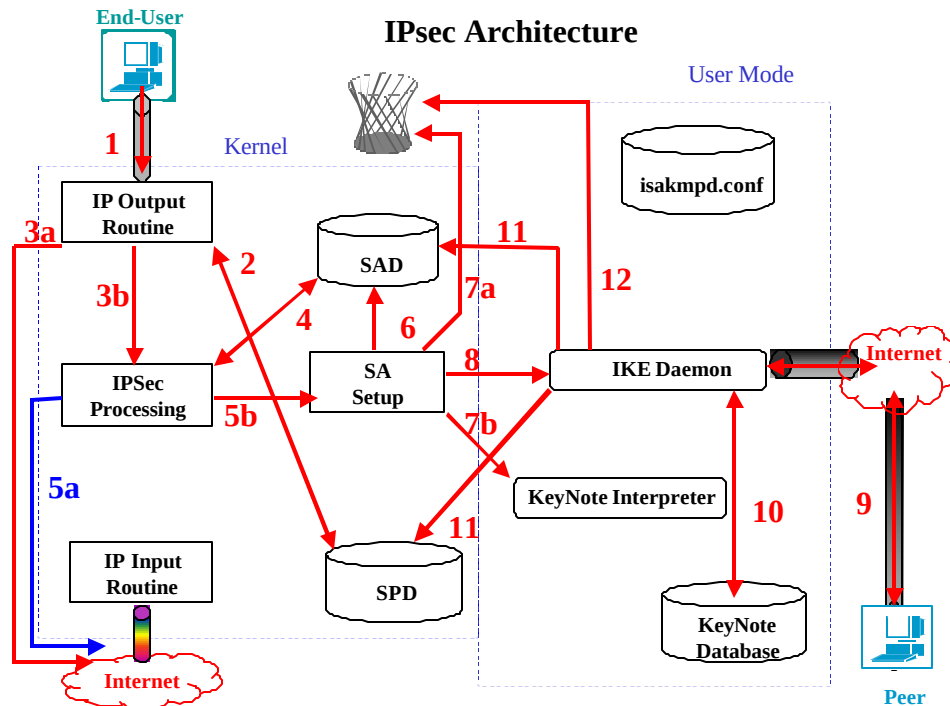


Figure 3.23. IPsec Architecture.

(After: Blaze, Matt, Ioannidis, John , and Keromytis, Angelos D, 2001

3. IPsec Input Processing

The following is a step-by-step process of outbound packet through an IPsec mechanism: (Blaze, Matt, Ioannidis, J and Angelos D. Keromytis, February 2001) [Refer to Figure 3.24

- 1) A packet arrives from an external network.
- 2) The SPD is consulted.
- 3a) If packet is an IPsec packet, forwarded to IPsec processing module
- 3b) If packet is an IPsec packet and SPD says to process packet, the packet is forwarded to internal system or upper-layer protocol/application.
- 3c) If SPD says to discard, the packet is discarded with no further processing.
- 4) The SAD is consulted for SA specifics for the packet.
- 5a) If an SA exists, the packet is dencapsulated and sent back to the IPsec input process module.
- 5b) If an SA does not exist, the SA SetUp module is triggered

- 6) The SPD is consulted.
- 7a) If the packet is authorized/valid, it is forwarded to internal system or upper-layer protocol/application.
- 7b) If SPD says to discard, the packet is discarded with no further processing.
- 8) The SA set-up module consults the SAD to verify that no SA exists.
- 9a) If one does, the packet is dropped.
- 9b) The KeyNote Database is consulted using packet information via the KeyNote interpreter to determine if the packet should be accepted, dropped or needs IPsec protection. Note: This step does not occur in OpenBSD version of IPsec.
- 10) In the event of the requirement for IPsec protection, IKE daemon is initiated.
- 11) IKE then negotiates security parameters with the distant peer. The distant peer will consult its own KeyNote Policy and reply with a proposed setting of secure communication parameters.
- 12) The Keynote database will be consulted to ensure distant peer parameters comply with local policy.
- 13) If all is compliant, SA may be created updating the SAD and SPD.
- 14) At completion the original packet is discarded.

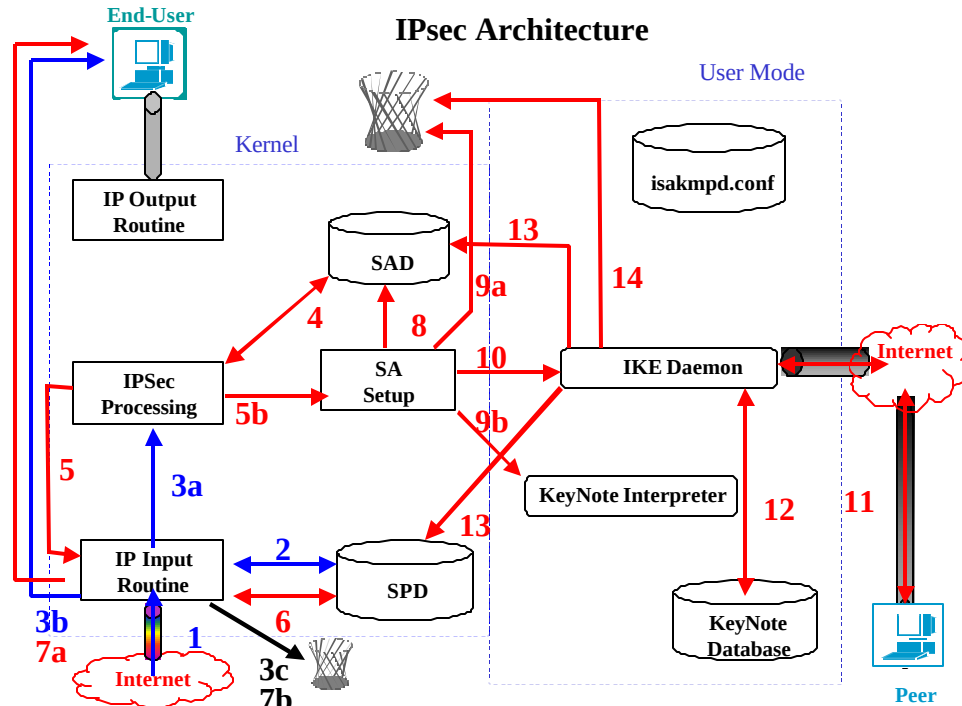


Figure 3.24. IPsec Architecture.

(After: Blaze. Matt, Ioannidis, John, and Keromytis, Angelos D, February 2001)

E. CONCLUSION

In this Chapter I reviewed the IPsec architecture, describing each of the components and the “incoming” and “outgoing” IP packet process. I introduced OpenBSD’s security policy implementation, KeyNote, explaining how it interfaces with the IPsec mechanism. I then concluded by providing a detailed step-by-step outline of IPsec’s input and output processing.

The next chapter, Design and Process, will detail the methodology involved in the parameterization of IPsec.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESIGN AND PROCESS

A. INTRODUCTION

In this chapter, I will outline and discuss the design and process for parameterizing IPsec. Specifically I will discuss two design goals: providing granularity to KeyNote, and parameterizing and improving the **isakmpd.conf** / **isakmpd.policy** (KeyNote) security proposal loading process. The goal of parameterizing IPsec is to enable IPsec to assume a dynamic functionality. This dynamism is based on new, external, environment parameters that can appropriately cause IPsec security attribute adjustments for protected communications.

In order to incorporate the environment/dynamic parameters into the IPsec mechanism, the security policy found in KeyNote must be extended appropriately. By adding dynamic parameters to KeyNote, the security mechanism can accurately perform Boolean assertion operations incorporating the specified value of the dynamic parameters. The dynamic parameters will also have to be properly loaded into the assertion query mechanism.

IPsec mechanisms negotiate security attributes between peers in the form of security proposals. A security proposal consists of a set of authorized security attributes that can be associated with a particular communication connection to meet with the local security policy. A security proposal set can be made up of one or more security proposals each with a set of security attributes.

Currently the IKE loading process retrieves security proposal ranges from **isakmpd.conf**. However, during peer negotiations, security proposals, which are loaded during (re)initialization from **isakmpd.conf**, are verified via **isakmpd.policy**. This requires security policy to be defined in multiple areas; namely **isakmpd.conf** and **isakmpd.policy**. For efficiency and ease of management, this process will be modified so that security proposal ranges are loaded directly from KeyNote/**isakmpd.policy**. In essence, my goal is to streamline the storage of security policy to only one location: KeyNote/**isakmpd.policy**.

Another modification is required to properly incorporate the dynamic parameter values into the proposal set loaded from **isakmpd.policy**. This will provide dynamic parameterization to the IPsec ensuring that only appropriate security proposals are loaded during (re)initialization.

Implementing these modifications to the IPsec mechanism will ultimately parameterize the IPsec (re)initialization and security proposal negotiation process.

B. PROVIDING GRANULARITY TO KEYNOTE

1. Goal

The process of providing granularity to KeyNote involves adding more attributes, both security- and non-security-related to KeyNote. This will result in a more complex security policy, increasing the overall potential parameter combinations possible in KeyNote/**isakmpd.policy**. To reflect dynamic, environmental parameters in IPsec, the security policy mechanism, KeyNote, will require modification. These modifications will enable the local security policy to change according to changes in the new dynamic external parameters.

The earlier examples of Network Mode and Security Level will be used as external parameters and will be added to KeyNote's condition assertion. The extended semantics of the security policy assertions will allow any dynamic external parameter and any number of security-related parameters to be represented. The KeyNote Query mechanism will then require the ability to import the current value of the dynamic external parameters to perform a query based on the new dynamic parameters.

2. Process Review

The following is a brief review of the current KeyNote structure and KeyNote query mechanism, and a description of the required modifications.

2.1 Current Keynote Process Review

Currently, KeyNote's structure is composed of IPsec specific parameters. Dynamic parameters will need to be added to the structure. Likewise the KeyNote query structure utilizes only IPsec specific parameters.

2.1.1 Keynote Structure

KeyNote's structure is made up of IPsec security parameters and

related application parameters that describe allowed interactions between different network peers. IPsec security parameters include encryption algorithms, authentication algorithms, transportation modes, key length, key lifetimes, identification and authentication certificates, and other security related variables. Related application parameters include local and remote ports utilized by applications.

2.1.2 Current Keynote Query Mechanism

The KeyNote query mechanism is handled by the KeyNote interface. To summarize the peer negotiation process, a brief description follows. The initiating IPsec peer prompts the IPsec mechanism by launching a security supported application (i.e. telnet or finger). The initiator's IPsec mechanism sends a security proposal to the responder peer to establish communication in support of the specified application. The responder peer uses the proposal received to perform a query on its KeyNote. If a security proposal is accepted, the responder returns the accepted security proposal. The initiator then performs a query on the local KeyNote to ensure that security proposal is authorized. If the query returns TRUE, the initiator returns a final message to the responder completing the negotiation handshake. The query performed on KeyNote verifies the security proposal's compliance with locally defined policy. The query mechanism loads the security proposals and performs the query on KeyNote receiving a Boolean response reflecting whether or not the security proposal is consistent with the local KeyNote policy.

2.1.3 Current KeyNote Process Flow

The following is a step-by-step review of the current KeyNote process. See Figure 4.1. For a more details refer to Chapter Three IPsec Architecture.

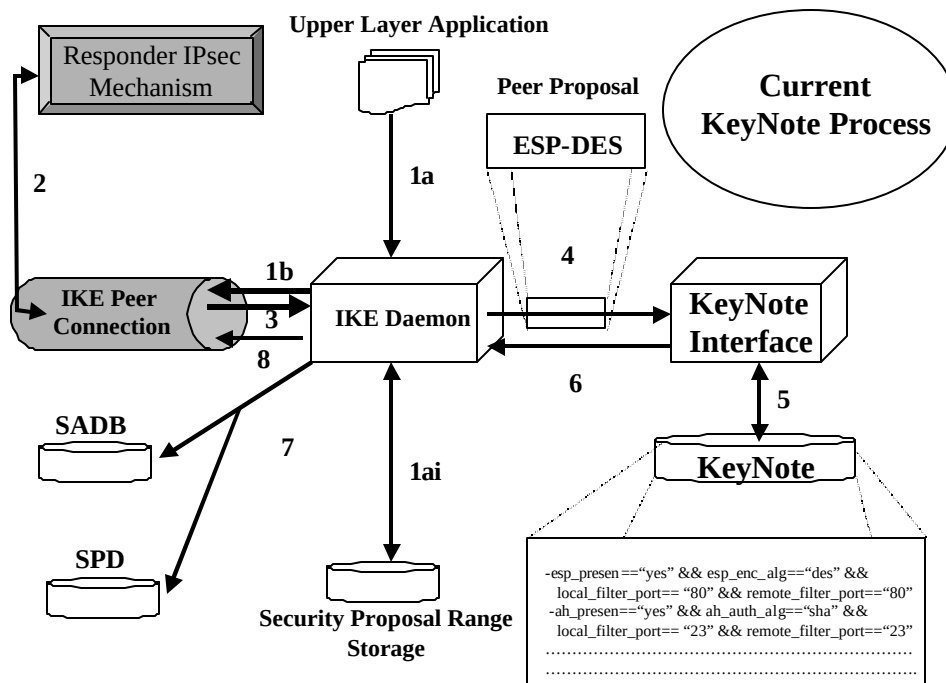


Figure 4.1. Current KeyNote Process.

- 1a - Upper Layer Application triggers Initiator's IKE Daemon and starts peer negotiation with responder peer.
- 1ai - Security proposal ranges are retrieved from initiator's memory
- 1b -. Initiator security proposal set is send to responder.
- 2 - Responder receives initiator's security proposal, processes the proposal and replies by sending an acceptable security proposal to the initiator peer. (Note: specifics on the responder's security mechanism are purposely obscure to demonstrate the importance of independent system architecture).
- 3 - Initiator peer receives the responder's security proposal.
- 4 - Responder Peer's proposal is submitted to the Initiator's KeyNote interface.

- 5 – Initiator’s KeyNote interface loads query mechanism and submits the query to KeyNote, receiving a Boolean response.
- 6 – Return to Initiator IKE Daemon.
- 7 – If a proposal is accepted then security associations are constructed and loaded into Initiator’s SAD and SPD.
- 8 – If the proposal is accepted, then the initiator sends an acknowledgement to the responder peer. Otherwise, the initiator notifies the responder peer of the refusal and the proposal is discarded.

2.2 Modified Keynote Structure and Query Mechanism

In order to incorporate dynamic parameters into the KeyNote structure and query mechanism, the following modifications will be required.

2.2.1 Modifications to Keynote Structure

The dynamic/environment parameters, Network Mode and Security Level, and their authorized local ranges, need to be represented in the condition section of the KeyNote structure. This will add further levels of granularity and complexity to KeyNote.

For example, the condition statement in KeyNote, shown in figure 4.2, authorizing telnet communication between two peers using 3DES as the encryption algorithm will have to be further defined for all ranges of network mode and security levels. At this point, the security policy can provide different levels of security attributes to the peer connection association by varying the encryption algorithm used in accordance with the security level and network mode values.

An Example of a Condition Statement in KeyNote

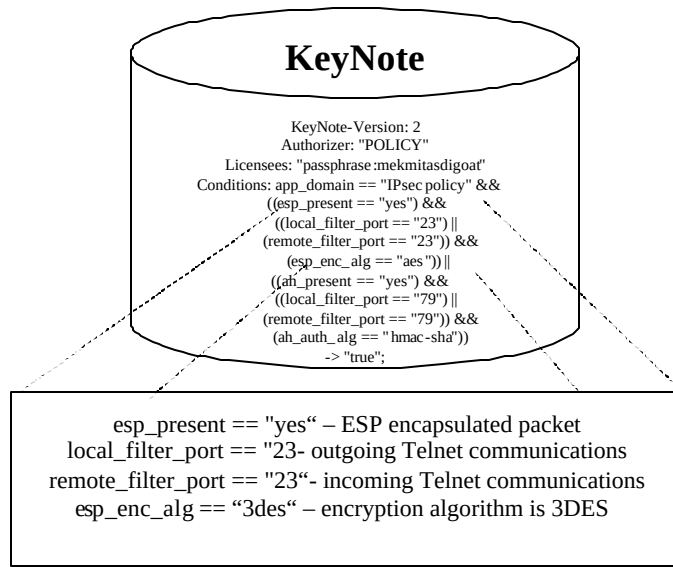


Figure 4.2. An Example of a Condition Statement inKeyNote.

It is important to ensure that the dynamic parameter modifications to KeyNote do not depend upon on IPsec peers having identical dynamic parameters. It is essential for compatibility that KeyNote incorporates the local dynamic parameters for local use only and that the query process remain independent of peer dynamic parameters.

2.2.2 Modifications to the KeyNote Query Mechanism.

The KeyNote query mechanism will require modification to allow for dynamic parameter value injection. This will enable the query to properly evaluate security proposals with respect to the current value of the dynamic parameters. The modifications will provide assurance that changes to dynamic parameters that might occur during the negotiation are reflected accordingly in KeyNote decisions.

2.2.3 Modified KeyNote Process Flow

The following is a step-by-step review of the modified KeyNote process. Refer to Figure 4.3.

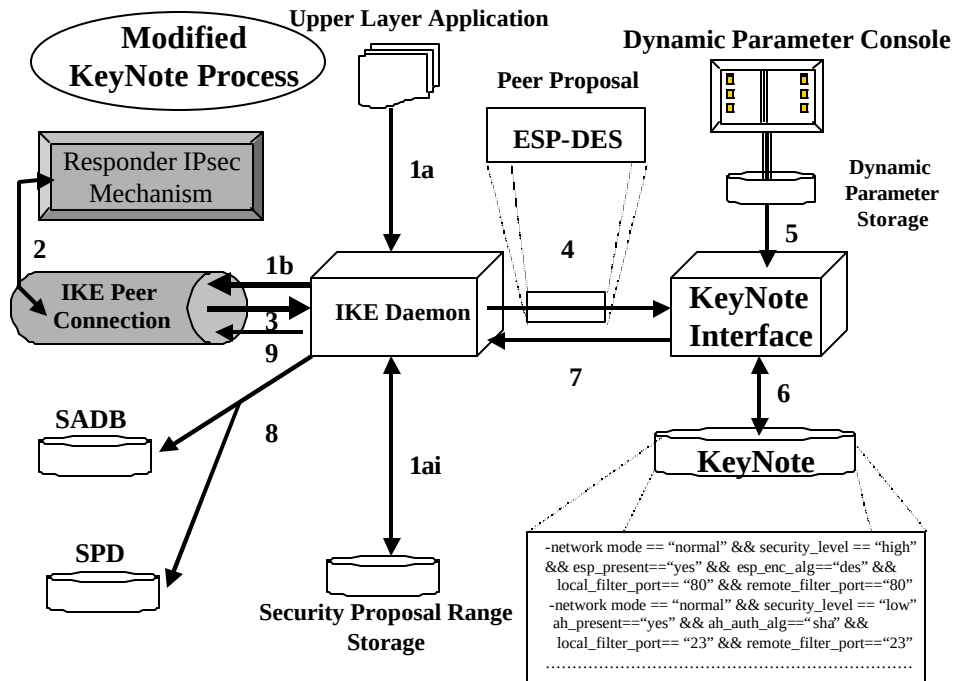


Figure 4.3. Modified KeyNote Process.

- 1a - Upper Layer Application triggers initiator's IKE Daemon and starts peer negotiation with responder peer.
- 1ai - Security proposal ranges are retrieved from initiator's memory
- 1b - Initiator security proposal is sent to responder.
- 2 - Responder receives initiator's security proposal, processes the proposal and replies by sending an acceptable security proposal to the initiator peer. (Note: specifics on the responders security mechanism are purposely obscure to demonstrate the importance of independent system architecture).
- 3 - Initiator peer receives the responder's security proposal.
- 4 - Responder's proposal is submitted to the initiator's KeyNote interface.
- 5 - Initiator's KeyNote injects current dynamic parameter

values into the query structure.

- 6 – Initiator’s KeyNote interface loads the query mechanism and submits the query to KeyNote, receiving a Boolean response.
- 7 – Return to initiator’s IKE Daemon.
- 8 – If the proposal is accepted then security associations are constructed and loaded into initiator’s SAD and SPD.
- 9 - If the proposal is accepted, the initiator sends an acknowledgement to the responder peer. Otherwise, the initiator peer notifies the responder of the refusal and the proposal is discarded.

3. Modification Phases

The modification of the KeyNote Structure and Query Mechanism will be performed in the following phases:

3.1 Add Dynamic Parameters to KeyNote

The first step in the process of adding granularity to KeyNote, is development of a method to add dynamic parameters and their ranges to KeyNote. It is essential that the parameters be added properly to the current security policy. This implies ensuring that all authorized values of the newly inserted parameters are correctly matched with other corresponding values reflecting the security policy. For example, if Security Level, with values High and Low, are added to KeyNote, the assertion would have to be rewritten to account for all security attributes authorized when security level is Low and High, respectively. This will add to the depth of the logical assertion as well as increases its complexity. The importance of understanding KeyNote’s structure and syntax is critical in performing the required modifications in achieving dynamic parameterization.

3.2 Develop a Method to Inject Dynamic Parameters into the KeyNote Query Mechanism.

The next step is to design and develop a method to inject the current values of the dynamic parameters into the KeyNote query mechanism. This process will involve understanding the current KeyNote query mechanism to determine how the

dynamic parameters values can be injected.

3.3 Develop a Console or Interface to Receive Dynamic Parameter Selection/Adjustment.

A method will be required to allow a system or a user to make changes to dynamic parameters used to select security attributes in accordance with the security policy. This mechanism must have the capability to effect immediate change on the dynamic parameters and may trigger adjustments to IPsec mechanism.

3.4 Testing Modifications

Once all modifications have been performed, a thorough testing phase will be required. Testing should involve at least two dynamic parameters (i.e. Network Mode and Security Level) with at least two ranges for each. The query mechanism should be tested to ensure that it is consistent and resistant to logical errors. All errors should be documented and corrected if possible. Any uncorrected errors should be listed in the Future Work Chapter Seven.

C. PARAMETERIZING AND IMPROVING ISAKMPD.CONF – KEYNOTE PROPOSAL LOADING PROCESS

1. Goal

The (re)initialization phase will require modification to incorporate the dynamic IPsec parameters. The value of these parameters will have a direct effect on which security proposals are loaded into memory for peer negotiations. Currently, local security policy is represented in two areas: KeyNote/**isakmpd.policy** and **isakmpd.conf**. This causes a problem in the area of security policy management. To provide for coherent policy management, the configuration process needs to be modified to utilize only KeyNote/**isakmpd.policy**. Ultimately, the security policy will be managed in only one database and will incorporate the granular dynamic parameters.

2. Process Review

The following is a brief review of the current configuration process of loading security proposal ranges from **isakmpd.conf**. It is followed by a description of a modified configuration process for loading security proposals from **isakmpd.policy**. This modified process includes dynamic parameterization.

2.1 Current Process Review

Currently, during the initialization phase, **isakmpd.conf** is read to retrieve valid security proposal ranges. The data stays in memory, unmodified, until needed or a re-initialization is triggered.

The following is a step-by-step review of the current process. Refer to Figure 4.4.

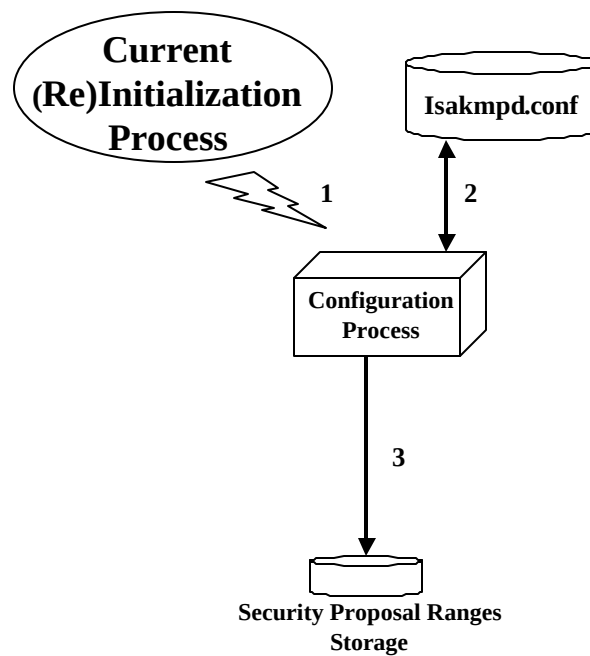


Figure 4.4. Current (Re)Initialization Process.

- 1 – (Re)Initialization triggers the IPsec configuration process.
- 2 – Read the **isakmpd.conf** for mechanism initialization parameters and valid security proposal ranges.
- 3 – Load valid security proposals into memory for later use.

2.2 Modified Process Review

To effectively parameterize the configuration phase by incorporating dynamic parameters, two modifications to the existing process are required. First, the current value of dynamic parameters must be retrieved. Second, valid security proposal ranges in accordance with the Network Mode and Security Level values must be retrieved from KeyNote/**isakmpd.policy** instead of **isakmpd.conf**.

The following is a step-by-step description of the modified process. Refer to Figure 4.5.

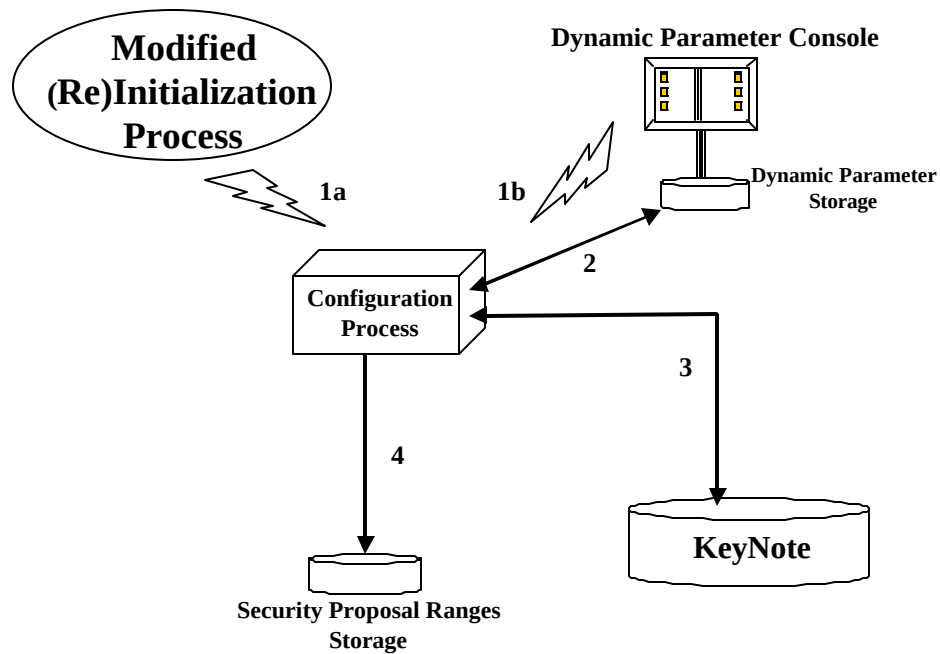


Figure 4.5

Figure 4.5. Modified (Re)Initialization Process

- 1a – (Re)Initialization triggers the IPsec configuration process, or
- 1b – Dynamic Parameter console triggers the IPsec configuration process.

- 2 – The Configuration Process retrieves the current values of the Dynamic Parameters.
- 3 The Modified Configuration Process retrieves valid security proposal parameters from KeyNote. Note that a reduced **iskampd.conf** will still be required to store non-security related mechanism initialization parameters.
- 4 – The Configuration Process loads valid security proposals into memory for later use.

3. Modifications Phases

The modification of KeyNote Structure and Query Mechanism will be performed in the following phases:

3.1 Determine Security Proposal Range Syntax

In order to retrieve valid security proposal ranges from KeyNote instead of **insakmpd.conf**, the proper loading syntax will need to be identified. If the KeyNote utilizes a different syntax for security proposals, a parsing method will be required to translate the KeyNote syntax into the syntax required by configuration process.

Initial review indicates that the syntaxes differ and that parsing will be required. After the valid security proposals are retrieved from KeyNote, they need to be translated into a form that is expected by the configuration process. Another potential solution would be to change the expected security proposal range syntax in the configuration process to one similar to that of KeyNote. (See Future Work Chapter Seven)

3.2 Develop a Method to Retrieve Current Dynamic Parameter Values

A method to determine the current dynamic environment variables will be required prior to retrieving the valid security proposals from KeyNote. The configuration process should be able to easily access the parameters values for these parameters

3.3 Develop Method to Retrieve Valid Security Proposal Ranges from KeyNote/iskampd.policy

Once the values of the dynamic parameters have been determined, a

method for retrieving the corresponding security proposals from KeyNote is required. This method will have to be able to traverse the KeyNote Condition assertion structure and retrieve the proposal ranges efficiently.

It is important to note that non-security related mechanism initialization parameters will remain in **isakmpd.conf**. For further discussion, refer to Future Work Chapter Seven.

4. Testing

Once all modifications have been performed, a thorough testing phase will be required. Testing should involve at least two dynamic parameters (ie Network Mode and Security Level) with at least two ranges for each. The query mechanism should be tested to ensure that it is consistent and free of logical errors. All errors should be documented and corrected if possible. Any uncorrected errors should be listed in the Future Work Chapter.

D. CONCLUSION

The design and method for parameterizing IPsec was outlined and discussed in this chapter. The design phase identified two specific modifications that will be required: providing granularity to KeyNote, and parameterizing and improving **isakmpd.conf** / **isakmpd.policy** (KeyNote) security proposal range loading process. Providing granularity to KeyNote will require an in-depth review of the KeyNote structure including the Boolean query mechanism. The goal will be to logically insert the dynamic parameter values according to policy definition, therefore, providing finer granularity to the Boolean query mechanism. Modifications to the configuration process to enable security proposal ranges to be retrieved from KeyNote, will require a syntax review of KeyNote and security proposal ranges expected by the configuration process. There is potential for a parsing requirement to translate proposals into the appropriate form.

Using the design and process provided, the implementation phase will follow with specifics on pseudo and source code structures, successes and challenges.

THIS PAGE INTENTIONALLY LEFT BLANK

V. IMPLEMENTATION

A. INTRODUCTION

The goal of this research is to modify the current implementation of OpenBSD IPsec to incorporate parameterization of dynamic parameters. After a thorough review of the current system architecture and an in-depth design phase, I was able to implement the modifications.

The modifications were performed in two phases: providing granularity to KeyNote, and stream-lining and incorporating parameterization to the **isakmpd.conf** / **isakmpd.policy** (KeyNote) security proposal set-loading process.

For each phase, I will discuss the implemented design and methodology, processing description, and pseudo code, including specific algorithms and code structure, assumptions, challenges, workarounds, and potential problems. Any implementation issues that could not be solved will be mentioned and further discussed in the Future Work Chapter Seven.

B. PROVIDING GRANULARITY TO KEYNOTE

1. Parameterization of KeyNote.

Keynote previously used static parameters set by the system administrator, defining system security information and authorized security associations. However, the challenge was to enable Keynote to handle dynamic parameters such as security level and network mode. These parameters would be set via an external module or device. Incorporating these changes enabled dynamic parameters to control security attribute setting adjustments in accordance with security policy.

The first step was to integrate dynamic parameter tag and value statements into the current condition assertion structure of KeyNote/**isakmpd.policy**. The next step was to analyze the current KeyNote query mechanism, specifically, the KeyNote query routine used with static security proposal sets had to be located. Modifications could then be made to the existing code to incorporate dynamic parameterization.

1.1 Inserting Dynamic Parameters into KeyNote’s Condition Assertion Structure.

Keynote as defined in Chapter III is made up of various sections. Security attributes reside in the condition section and are expressed in the form of logical assertions.

The condition section’s syntax is in the form of a logical statement, similar to a condition that might be found in an “if statement”. The section is usually broken into sub statements by using &&, ||, and parentheses to construct logical conditions. For example the following phrase describes two security proposals supporting telnet services (service_port= 23) using ESP with 3DES for encryption and finger services (service_port=79) using AH with SHA for authentication:

```
(local_filter_port == “23” &&
esp_present == "yes" &&
esp_enc_alg == "3des") ||
(local_filter_port == “79” &&
ah_present == "yes" &&
ah_auth_alg == "sha") -> “true”;
```

1.2 Inserting Dynamic Parameters into Keynote

Using the above example, the dynamic parameters, network mode and security level, are added to the condition statement. To properly insert the parameters, it was essential that the existing logical structure be maintained. This required the parameters to be added (and properly defined according to policy) to all the existing conditional expression combinations. Using the previous example with security levels “high” and “low” and network modes “normal” and “impacted”, the condition phrase is expanded. Notice that further granularity results in the use of different encryption and authentication algorithms for each network mode and security level.

```
(network_mode = “normal” &&
((security_level = “high” &&
((local_filter_port == “23” &&
esp_present == "yes" &&
```

```

        esp_enc_alg == "3des") ||
(local_filter_port == "79" &&
    ah_present == "yes" &&
    ah_auth_alg == "sha")))) ||
((security_level = "low" &&
    ((local_filter_port == "23" &&
        esp_present == "yes" &&
        esp_enc_alg == "des") ||
(local_filter_port == "79" &&
    ah_present == "yes" &&
    ah_auth_alg == "des-mac")))) ||
(network_mode = "impacted" &&
    ((security_level = "high" &&
        ((local_filter_port == "23" &&
            esp_present == "yes" &&
            esp_enc_alg == "aes") ||
(local_filter_port == "79" &&
    ah_present == "yes" &&
    ah_auth_alg == "sha")))) ||
((security_level = "low" &&
    ((local_filter_port == "23" &&
        esp_present == "yes" &&
        esp_enc_alg == "3des") ||
(local_filter_port == "79" &&
    ah_present == "yes" &&
    ah_auth_alg == "sha-md5")))) -> "true";

```

It becomes obvious that adding parameters to the keynote condition assertion greatly increases the complexity of each expression. In order to implement and manage a detailed and complex security policy, a *policy editor* would be required to translate the assertion's syntax into a representation that is easier for operators to manage. Otherwise, the potential for mistakes as a result of the complexity could be very high. See

Future Work Chapter Seven for more details on a security policy editor.

Variable precedence should also be considered. Which variable should be listed first and where is the precedence defined? Fortunately, we have proper logical expressions and the order and precedence does not matter so long as the combination is properly constructed and represents all authorized security associations. For example, the order of nested dynamic parameters, network mode and security level, is not relevant to the logical outcome of a Boolean query on the condition assertion.

1.3 KeyNote Query Functionality

The next step was to locate the existing KeyNote query calls. By reviewing the OPENBSD IPsec documentation, I was able to determine the following KeyNote interface mechanism:

[ike_quick_mode.c]

- **check_policy()**

- LK(kn_add_action()) – loads a tag and its value into the KeyNote query mechanism. If the same tag is loaded more than once, only the last instance will be used. Can be used to overwrite preloaded default values. If the process is successful it returns a ‘1’. Otherwise it returns a ‘0’. (*LK(kn_add_action())*, *OpenBSD Programmer’s Manual Pages*, 2000)

- LK(kn_do_query()) – performs the Boolean query on KeyNote on the information loaded into the query mechanism. If the proposal is accepted, (valid according to existing assertions in the condition section of KeyNote) a true value (‘1’) is returned. Otherwise a false value (‘0’) is returned. (*LK(kn_do_query())*, *OpenBSD Programmer’s Manual Pages*, 2000)

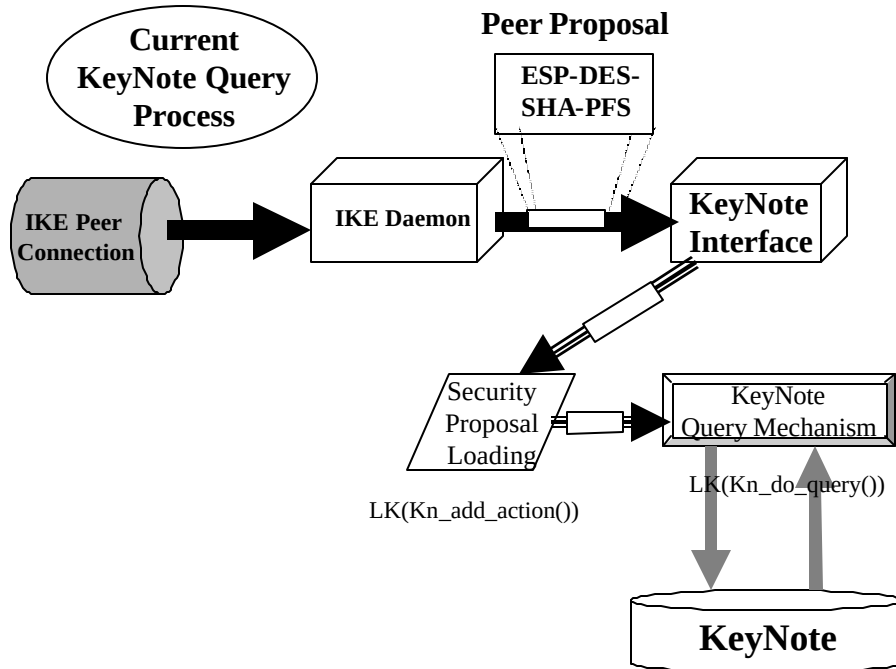


Figure 5.1. Current KeyNote Query Process.

1.4 Dynamic Parameter Input Module

The dynamic input module retrieves the current values of the dynamic parameters.

1.4.1 Design Approach

To provide for system parameter inputs, I implemented a file input/output (I/O) approach to incorporate the external parameter input simulation. The file I/O approach method is utilized throughout the current OpenBSD architecture allowing different processes to communicate with each other. Basically the file approach involves an interface component writing data or messages to a file and another component continually polling the file and responding accordingly. The following is the file location and an example of the syntax used to write and read the dynamic parameters:

```

[/usr/src/sbin/isakmpd/dynamic_parameters]
network_mode = normal
security_level = high

```

1.4.2 Processing Description

The following describes the algorithm utilized to develop my code:

- Declare file variables used to open and read from the file
- Define and declare a structure that can be used to hold the dynamic parameters. The structure should have the capability to grow dynamically as required to hold a variable amount of parameters.
- Read-in dynamic parameters from a file and load them into the structure accordingly.

1.4.3 Pseudo Code

The following code was inserted into **ipsec_quick_mode.c**:

- Structure : **dynamic_packet**. Contains the following variables:
 - char* title – character string used to hold the dynamic parameter title.
 - char* symbol – character string used to hold the symbol “=” “>” “<” “!=”.
 - char* value - character string used to hold the dynamic parameter value.
- Function: **struct dynamic_packet package_dynamic_parameters(int *package_counter)** - added to **ipsec_quick_mode.c** to retrieve the dynamic parameters from a file and load them into an array of dynamic_packet structure (defined above).
- Input:
 - int * package_counter - pointer to integer variable used for the number of dynamic parameters added to the structure. Pointer used to be able to return the value to the calling function.
- Output:
 - struct dynamic_packet - pointer to array of structures.
- Process:
 - Initialize the pointer to an array size of 10
 - Open file to read in dynamic parameters

- Check for errors in opening the file
- Do-while loop used to read in data until EOF reached or file reading error occurs.
 - Read in from the file expecting the following syntax:
 - title <string> symbol <string> value <string>
 - Create exactly enough new structure space for the dynamic parameters and copy values from temp variables to the structure variables.
 - Increment the package counter array of the structure.
 - Check to see if the array of dynamic parameters has reached max size. If so, resize array accordingly.
- Close file.
- Free temporary memory.
- Return array of structures.
- Code added to **check_policy()**:
 - Initialize dynamic parameter array structure
 - Call **package_dynamic_parameters()** function to read in the dynamic parameters and store them in an array of structures.

1.5 Inserting Dynamic Parameters Into The Keynote Query Functionality

Once dynamic parameters have been retrieved, they can be used in the KeyNote query process.

1.5.1 Design Approach

The current implementation of IPSEC in OPENBSD utilizes a **policy_callback** structure for loading the IPsec parameters into Keynote for a query. This enables fewer lines of code. I attempted to utilize this functionality for inserting security level and network mode. But I was unable to successfully load all dynamic parameters into KeyNote query mechanism at once. Instead, I proceed by loading each system variable individually into the KeyNote query structure. This is an area that will require future work to streamline the process. (See Chapter VII).

1.5.2. Processing Description and Pseudo Code

The following is the code added to **check_policy()**:

- Loop through the array of structures.
 - Load dynamic parameters individually using **LK(kn_add_action())** & **LK(kn_close())**
 - Check for loading errors.

The following (see figure 5.2) is a step-by-step review of the modified KeyNote Query Process:

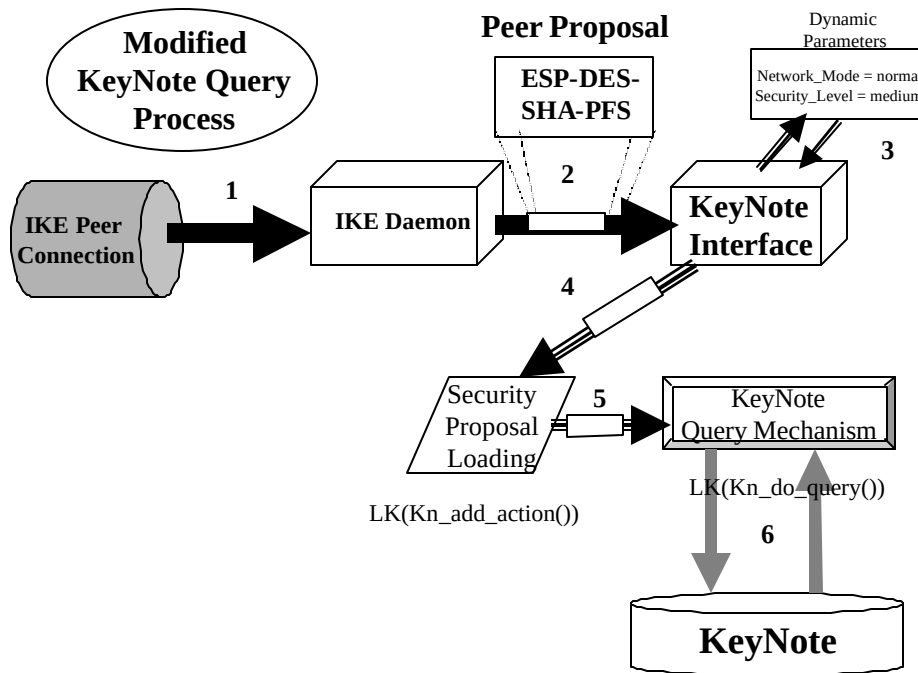


Figure 5.2. Modified KeyNote Query Process.

- 1 – Peer sends a security proposal to recipient peer during the security association negotiation phase.
- 2 – IKE Daemon forwards proposal to KeyNote interface.
- 3 – KeyNote interface retrieves the current values of the dynamic parameters.
- 4 – Mechanism prepares for the security proposal loading.
- 5 – Dynamic parameter values and security proposal are loaded into KeyNote query mechanism.

6 – Boolean query is performed on KeyNote.

C. **REPLACING ISAKMPD.CONF WITH KEYNOTE**

To incorporate a complete dynamic functionality into the IPsec mechanism, the loading process for a valid set of security proposals must be modified to allow for the injection of the dynamic parameter values. This will ultimately provide further granularity to the selection of security proposals. To further streamline the management of the security policy, all security proposals should be derived from **isakmpd.policy**/KeyNote. This will require the modification of the existing process of retrieving the valid set of security proposals from **isakmpd.conf**.

1. **Current isakmpd.conf**

isakmpd.conf is used as a configuration file for the **isakmpd** daemon during (re)initialization phase of the IPsec mechanism. It provides the initial security associations and keys used for Phase I of the ISAKMP daemon, and a set of valid security proposals used in Phase II for IPsec peer negotiation communication. (ISAKMPD.CONF(5), OpenBSD Programmer's Manual, October 1998) When the IPsec mechanism is (re)initialized, **isakmpd.conf** is read and all the information is loaded into memory

As previously discussed in Chapter III, IPsec Architecture, **isakmpd.conf** utilizes the traditional .ini style file structure and is segregated into sections indicated by “[]”. Within each section, parameters are defined using the “<tag> = <tag value or range value>” syntax. Tag values may consist of other section names. This results in a tree-like structure through **the isakmpd.conf** file. (ISAKMPD.CONF(5), OpenBSD Programmer's Manual, October 1998)

2. **Process of Replacing isakmpd.conf with KeyNote**

In order to properly modify the current process of loading the valid set of security proposals to incorporate dynamic parameters and reduce the security policy management scope to KeyNote, the following must be accomplished. First a method to retrieve the current values of the dynamic parameters and to inject them into the loading process must be implemented. Second, the loading process must be modified to allow for the valid set of security proposals to be retrieved from **isakmpd.policy**/KeyNote instead of **isakmpd.conf**.

2.1 Modification of the Valid Set of Security Proposal Loading Process

To successfully achieve the goal of modifying the current implementation of the loading process of the valid set of security proposals, the following steps had to be accomplished. First, a review of the current methodology and code was required to determine the appropriate location for the code modification. Second, an understanding of the **isakmpd.conf** configuration loading syntax was required so that security proposals retrieved from KeyNote could be appropriately translated. Third it was necessary to develop the parsing mechanism to retrieve valid security proposals from KeyNote and properly translate them, as required, into the form expected by the (re)initialization process. Last, the dynamic parameter values had to be incorporated into the security proposal retrieval process, to allow for further granularity in the selection process.

2.1.1. Review of the Current Implementation of the **isakmpd.conf** Loading Process

Currently **isakmpd.conf** is read during initialization of the IPsec mechanism and the re-initialization is triggered by a change of state.

2.1.1.1 Design Approach

Isakmpd.conf contains information on peers (IP addresses, Net Masks), IKE phase I parameters (IKE security associations: encryption, authentication and keys), and IPsec phase II security proposals. The IPsec mechanism reads the **isakmpd.conf** file from the **conf.c** file located in the **/usr/src/sbin/isakmpd/** directory. The file is read into a string buffer all at once. The buffer is then parsed, start to finish, using section tags as indicators of section existence. As information is retrieved it is stored in memory using a structure list/array.

2.1.1.2 Pseudo Code

The following is a brief step-by-step description of the process:

- Function: **conf_init()** – called to begin the initialize phase of the IPsec process.
- Storage structure for extracted information from **isakmpd.conf** is

prepared.

- **conf_reinit()** is called.

- Function **conf_reinit()** – called to begin the (re)initialization process.

- **file_secrecy ()** is called.

- size of file is returned (by reference to int).

- If **isakmpd.conf** exists.

- **isakmpd.conf** is opened. Error checking is performed in the event of I/O error.

- Entire contents of **isakmpd.conf** is read into a string buffer.

- File is closed.

- **conf_begin()** is called

- if first call, define static transaction number as 1 and returns.

- Otherwise increments transaction number and returns.

- **conf_parse()** called to parse string buffer.

- (See **conf_parse()** description below).

- Otherwise – **conf_begin()** is called.

- **conf_load_defaults()** is called. Pre-defined stored

values of security attributes are loaded into memory as

required. In the event of an incomplete load or loading error, the default values is used.

- As required, memory is freed.

- Function: **conf_parse()**

- While not at the end of buffer, parses the string buffer one line at a time.

- **conf_parse_line()** is called. (See description below).

- Function: **conf_parse_line()** – parse string buffer one line at a time

- Skips comments demoted by #

- Recognizes section headers.

- Parses sections by tags and values.

- **conf_set()** is called to load tags and values into memory structure.

(See description below)

- Function: **conf-set()** – receives transaction number, section name, tag name, value and other override/default flags. The information is then stored in a structure for later use.

2.1.2 Syntax Required by the conf.c Loading Process

Conf.c requires a specific syntax for the set of valid security proposals. Specifically, it requires that information be broken into sections, tags and tag values (as found in **isakmpd.conf**). This information is then loaded into the structure used to store the security proposal until required. As the information is parsed from the **isakmpd.conf**, it is sent to **conf_set()** function which loads it into memory.

In order to modify the loading process to allow all security proposal information to be retrieved from KeyNote/**isakmpd.policy**, a translation will be required from KeyNote/**isakmpd.policy** form to the **isakmpd.conf** form. Specific syntax differences between the two forms are as follows:

- Phase II suite (set of valid security proposals).

- **isakmpd.conf** – exists in the following syntax:

- Tags separated by dashes (-).

- <Phase Mode (QM)>-<ESP or AH>-<Encryption or Authentication used><other security related attributes>

- Additional security proposals are added via comma (,).
- KeyNote/**isakmpd.policy** – does not exist in simple form. Information exists within the condition assertion. In order to generate the proper syntax it must be derived from condition assertion.
 - Tag and Tag Value
 - **isakmpd.conf** – are a mixture of uppercase and lower case.
 - KeyNote/**isakmpd.policy** – all tag and tag values exist in lowercase and tag values are in quotes (“”). All possible tags and section names must be accounted for to ensure proper capitalization translation.
 - Assignment symbol
 - **isakmpd.conf** – utilized single equals sign for assignment (“=”).
 - KeyNote/**isakmpd.policy** – utilizes double equals signs for assignment (“= ”).
 - File structure.
 - **isakmpd.conf** – utilizes a sequential file structure with section tags to differentiate between sections.
 - KeyNote/**isakmpd.policy** – utilizes a logical predicate format. The information is embedded in a series of && and ||. A method of assertion translation will be required to retrieve the require information.

Some of the information found in **isakmpd.conf** was not security policy related. It dealt with other connection-oriented specifics. I decided to leave this information inside **isakmpd.conf**. In other words, only the security policy information was removed from **iskampd.conf**.

2.1.3. Parsing Mechanism to Retrieve Valid Security Proposals from KeyNote and Properly Translate as Required

A parsing mechanism is required to retrieve the security proposal information from the current form of KeyNote’s condition assertion and translate it into the expected format required by **conf.c**.

2.1.3.1 Design Approach

The first step in the parsing process was to develop an algorithm to properly evaluate KeyNote's condition assertions. A few attempts of direct parsing involving recursive calls and parenthesis counting proved to be too syntax dependent and unstable.

The next approach relied on the concurrent development of a DNF parser that would extract embedded security proposals from an assertion and reconstruct them in the following format (NPS-CS-02-002, January 2002):

```
(((<tag> == <tag value>) && (<tag>==<tag value>) && .... ))||  
(((<tag> == <tag value>) && (<tag>==<tag value>) && .... ))||
```

2.1.3.2 Processing Description

Given the previous DNF format, the following is the processing description of the implemented parser:

- Initialize and define a list of structures (linked list/array).
- Read in the DNF security proposal file.
- Parse the file according to security proposals.
 - Provide a new structure to store information for each security proposal.
- Translate the stored information into the expected **isakmpd.conf** syntax.
- Load memory with the set of valid security proposals.

2.1.3.3 Pseudo Code

Below is the detailed description of the parsing implementation. The function call to the KeyNote parsing (**conf_kn_parse()**) was inserted in the **conf_reinit()** in **conf.c**, after the **load_default()** function call. A description of the implementation follows:

- **Suite_Struct** was used to hold the extracted security proposal information retrieved from the DNF security proposal. The structure contained all possible entries (within the scope of the research. Note that potentially, there could be other undefined parameters. This is discussed further in the Future Work Chapter Seven). Memory for each char string was dynamically created for memory conservation. In other words memory was allocated as needed throughout the parsing routine.

```

struct suite_struct {
    char * suite_name;
    char * suite_protocol;
    char * suite_transform;
    char * protocol_id;
    char * transform_id;
    char * encapsulation_mode;
    char * group_description;
    char * authentication_algorithm;
    char * life;
    char * life_type;
    char * life_duration;
    char * network_mode;
    char * security_level;
    char * esp;
    char * ah;
    char * esp_enc_alg;
    char * esp_auth_alg;
    char * ah_auth_alg;
    char * pfs;
    char * key_length;
};

```

- Function: **conf_kn_parse()** – function called to activate the DNF security proposal parsing mechanism.

- input: int trans – used for the transaction number for sequential processing.

- output: void.

- process:

- Array of structures initialized for two security proposals. The number two was chosen because my testing example consist of two security proposals.

- DNF security proposal file is opened
 - Error checking is performed. If the error occurs, exit routine and return to **conf_reinit()**.
- Utilize technique taken from **file_secrecy ()** function
stat (<file name>, &<size variable>) to determine size of file to be read. This enabled a dynamic approach to allocating only enough memory as needed.
- Read in entire DNF file into string buffer.
- Close the file.
- While not at end of string buffer:
 - Search for the first security proposal expression.
The search is performed by searching for the first occurrence of “(“. An assumption is made that for every variation of DNF security proposal possible, they will all begin with “(“.
 - **DNF_parse()** function is called.
 - if 1 (true) is returned then a valid security proposal was found and the counter is incremented.
 - Check to see if any valid security proposals were found.
 - If so, call **send_to_conf_set()** function to load valid security proposals into memory.
 - Free temporary memory used.
- Function: **send_to_conf_set(int trans, char * suite_title, char * suite, struct suite_struct * suite_profile, int struct_size)** – this functions sends parsed information to **conf_set** in the correct syntax.
- Input:
 - int trans – transaction number.
 - char * suite_title – holds title for tag defined in previous function.
 - char * suite – holds set of security proposals

- struct suite_struct * suite_profile – points to the list/array of suite structures.
- int struct_size – holds the size of the list.
- Output: None.
- Process:
 - Performs initial **conf_set** call for the General section including the set of security proposals (suite).
 - Loops through the list/array of structures and executes conf_set in accordance with the information loaded in the structure.
 - No NULL check is performed. It is assumed that a NULL value will not have a negative effect on the loading process and that the loaded default will be used instead.
 - The following is the standard suite structure is expected by **conf_set()**:
 - [General] section with a Default-Phase-2-Suite tag and value (set of security proposals). Sent only once.
 - [<suite name>] section with suite protocol per structure.
 - [<suite protocol>] section with protocol id per structure.
 - [<suite transform>] section with
 - transform id
 - encapsulation mode
 - group description
 - authentication algorithm
 - lifetime
 - per structure.
 - [<suite lifetime definition>] section with

- lifetime type
- lifetime duration
- per structure.

- Function: **struct suite_struct* struct_initialization (struct suite_struct * suite_profile)** – used to initialize each suite structure.

- Input: - struct suite_struct * suite_profile – holds the pointer to suite structure.
- Output: - returns the newly initialized structure.
- Process:
 - In an effort to minimize wasted memory, all elements of the structure are initialized to NULL.
 - Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.

- Function: **int DNF_parse(char **suite, char *buff_temp, int * buff_temp_counter, int szkn, struct suite_struct *suite_profile, struct dynamic_packet * package, int package_size)** – This function parses each security proposal found.

- Input:
 - Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.
 - char **suite – holds the set of security proposals. Pointer to a pointer used to for dynamic memory creation.
 - char *buff_temp – string buffer holding the DNF file.
 - int* buff_temp_counter – location of parsing index.

Pointer to integer is used to allow for pass by reference.

- int szkn – size of file/string buffer.

- struct suite_struct *suite_profile – pointer to suite structure.

- Outputs:

- integer

- returns 1 (false) if parse routine successful.

- returns 0 (false) otherwise.

- Process:

- Note: The methodology used in the following parsing function is as follows. The parsing is performed on per character basis. If a character matches the first character of an expected key word, one of two functions are called:

- parse_ipsec_para_tag()** or **parse_ipsec_parameter()**.

- Parse_ipsec_para_tag()** (explained later in detail) is used for specific tags utilizing Boolean tag values, usually “yes” or “no.”

- Parse_ipsec_parameter()** (explained later in detail) is used for normal tag and tag value expressions.

- Initialize suite structure;

- While loop with Boolean flag

- The following parsing is performed on a character by character basis. If the first character matches and a further string comparison will not cause a buffer-overflow, then a follow-up routine is called to check for the full comparison. If a comparison match is found, parsing continues in the called function. If not, function returns and parsing in local function continues until Boolean flag is set.

- Check for ESP. Call **parse_ipsec_para_tag()**

- If ESP is found then dynamic memory is

created in the suite structure for the string
IPSEC_ESP.

- Check for ESP encryption algorithm by calling **parse_ipsec_parameter()**.

- If ESP encryption algorithm is found then dynamic memory is created in the suite structure for ESP encryption algorithm.

- Check for ESP authentication algorithm by calling **parse_ipsec_parameter()**.

- If ESP authentication algorithm is found then dynamic memory is created in the suite structure for the ESP authentication algorithm.

- Check for AH by calling **parse_ipsec_para_tag()**.

- If AH is found then dynamic memory is created in the suite structure for the string IPSEC_AH.

- Check for PFS by calling **parse_ipsec_para_tag()**.

- If PFS is found then dynamic memory is created in the suite structure for the string yes (as found in the Keynote).

- Check AH for authentication algorithm. Call **parse_ipsec_parameter()**.

- If AH authentication algorithm is found then dynamic memory is created in the suite structure for the algorithm. Note that HMAC is added to the end of the authentication algorithm. The end result is: HMAC_<Authentication Algorithm> (ie HMAC_SHA). This is done to conform to

the iskampd.conf syntax.

- Check for ESP Group Description by calling **parse_ipsec_parameter()**.

- If ESP Group Description is found then dynamic memory is created in the suite structure for the KeyNote version of the Group Description. The function **group_description_translation()** is then called to convert the string into the **isakmpd.conf** form required by the **conf_set()**.

- Check for AH Group Description by calling **parse_ipsec_parameter()**.

- If AH Group Description is found then dynamic memory is created in the suite structure for the KeyNote version of the Group Description. The function **group_description_translation()** is then called to convert the string into the **isakmpd.conf** form required by the **conf_set()**.

- Check for ESP Encapsulation by calling **parse_ipsec_parameter()**.

- If ESP Encapsulation is found then dynamic memory is created in the suite structure for the encapsulation mode.

- Check for AH Encapsulation by calling **parse_ipsec_parameter()**.

- If AH Encapsulation is found then dynamic memory is created in the suite

structure for the encapsulation mode.

- Check for ESP life seconds by calling **parse_ipsec_parameter()**.

- If ESP life seconds is found then dynamic memory is created in the suite structure and **life_seconds_translation()** is called to convert the string into the **iskampd.conf** format. The converted string is then copied into the newly created space in the suite structure.

- Check for AH life seconds by calling **parse_ipsec_parameter()**.

- If AH life seconds is found then dynamic memory is created in the suite structure and **life_seconds_translation()** is called to convert the string into the **iskampd.conf** format. The converted string is then copied into the newly created space in the suite structure.

- Check for ESP life kilobytes by calling **parse_ipsec_parameter()**.

- If ESP life kilobytes is found then dynamic memory is created in the suite structure and **life_kilobytes_translation()** is called to convert the string into the **iskampd.conf** format. The converted string is then copied into the newly created space in the suite structure.

- Check for AH life kilobytes by calling **parse_ipsec_parameter()**.

- If AH life kilobytes is found then dynamic

memory is created in the suite structure and **life_kilobytes_translation()** is called to convert the string into the **iskampd.conf** format. The converted string is then copied into the newly created space in the suite structure.

- Check for the end of a DNF suite set by checking for “|”.

- If found then the Boolean flag is set to exit while loop.

- **add_para_values()** is called to properly configure the suite.

- Check for end of file. If not found advance file character index to the next character.

- Return int 1 (true) to indicate successful parsing iteration.

- Function: **void life_kilobytes_translation(char ** life, char **life_type, char **life_duration)** – this function is used to convert lifetime in kilobytes from the **KeyNote/isakmpd.policy** format to the **isakmpd.conf** format.

- Inputs:

- Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.

- char ** life – holds the initial life time input. Pointer to a pointer used for dynamic memory allocation.

- char ** life_type – holds the life time type string KILOBYTES. Pointer to a pointer used for dynamic memory allocation.

- char ** life_duration – holds the life time duration string .
Pointer to a pointer used for dynamic memory allocation.

- Outputs:

- char ** life – used to return life time. Pointer to a pointer used for dynamic memory allocation.
- char ** life_type – used to return life time type string KILOBYTES. Pointer to a pointer used for dynamic memory allocation.
- char ** life_duration – used to return the life time duration string . Pointer to a pointer used for dynamic memory allocation.

- Process:

- This function takes the input value of life and compares it with predefined life constants. If the constant is found, the appropriate information is stored in life_type and life_duration. All memory is dynamically created in this function.
- Check if life equals 1000. If so:
 - Free memory used previously by life and dynamically create memory for “LIFE_1000_KB.”
 - Dynamically create memory for “KILOBYTES”.
 - Dynamically create memory for “1000,768:1356”.
- Check if life equals 32000. If so:
 - Free memory used previously by life and dynamically create memory for “LIFE_32_MB.”
 - Dynamically create memory for “KILOBYTES”.
 - Dynamically create memory for “32768,16384:65536”.
- Check if life equals 45000000. If so:
 - Free memory used previously by life and dynamically create memory for “LIFE_4.5_GB.”

- Dynamically create memory for “KILOBYTES”
- Dynamically create memory for “4608000,4096000:8192000”.
- If no match is found, the default load of 1000 is used.
- Function: **void life_seconds_translation(char ** life, char **life_type, char **life_duration)** – this function is used to convert lifetime in seconds from the KeyNote/**isakmpd.policy** syntax to the **isakmpd.conf** syntax.
- Inputs:
 - Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.
 - char ** life – holds the initial life time input. Pointer to a pointer used for dynamic memory allocation.
 - char ** life_type – holds the life time type string SECONDS. Pointer to a pointer used for dynamic memory allocation.
 - char ** life_duration – holds the life time duration string . Pointer to a pointer used for dynamic memory allocation.
- Outputs:
 - char ** life – used to return life time. Pointer to a pointer used for dynamic memory allocation.
 - char ** life_type – used to return life time type string SECONDS. Pointer to a pointer used for dynamic memory allocation.
 - char ** life_duration – used to return the life time duration string . Pointer to a pointer used for dynamic memory allocation.
- Process:

- This function takes the input value of life and compares it with predefined life constants. If the constant is found, the appropriate information is stored in `life_type` and `life_duration`. All memory is dynamically created in this function.
- Check if life equals 600. If so:
 - Free memory used previously by life and dynamically create memory for “LIFE_600_SECS.”
 - Dynamically create memory for “SECONDS”.
 - Dynamically create memory for “600,450:720”.
- Check if life equals 3600. If so:
 - Free memory used previously by life and dynamically create memory for “LIFE_3600_SECS.”
 - Dynamically create memory for “SECONDS”.
 - Dynamically create memory to for “3600,1800:7200”.
 - If no match is found, the default value of 3600 is used.
- Function: **void group_description_translation(char **group_description)** – this function is used to convert group description from the KeyNote/**isakmpd.policy** syntax to **the isakmpd.conf** syntax.
- Inputs:
 - Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.
 - `char     ** group_description`– holds the initial group description variable. Pointer to a pointer used for dynamic memory allocation.

- Outputs:

- char ** group_description – used to return translated group_description. Pointer to a pointer used for dynamic memory allocation.

- Process:

- This function takes the input value of the group description and compares it with predefined constants. If the constant is found, the appropriate information is stored in group_description. All memory is dynamically created in this function.

- Check if group_description equals 1. If so:

- Free memory used previously by group_description and dynamically create memory for “MOPD_768”

- Check if group_description equals 2. If so:

- Free memory used previously by life and dynamically create memory for “MODP_1024”

- Check if group_description equals 3. If so:

- Free memory used previously by life and dynamically create memory for “MODP_155”

- Check if group_description equals 4. If so:

- Free memory used previously by life and dynamically create memory for “MODP_185”

- Check if group_description equals 5. If so:

- Free memory used previously by life and dynamically create memory for “MODP_1536”

- If no match is found, the default value of group_description 1 is used.

- Function: **void add_para_values(char ** suite, struct suite_struct ** suite_profile)**- this function generates the security proposal format required by the configuration process.

- Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.
- Inputs:
 - char ** suite - holds the set of security proposals. Pointer to a pointer used to for dynamic memory creation.
 - struct suite_struct **suite_profile – pointer to suite structure.
- Outputs:
 - char ** suite - returns the modified set of security proposals. Pointer to a pointer used to for dynamic memory creation.
 - struct suite_struct **suite_profile – pointer to suite structure used to return the modified suite_profile structure.
- Process:
 - This function uses the parsed data and generates the suite/set of security proposals in the required **isakmpd.conf** syntax.
 - Dynamically creates memory for suite_name in suite_profile structure.
 - Checks if suite is large enough for further modification. If not, memory is dynamically reallocated for the suite.
 - If suite has not been allocated memory, default size of memory is allocated.
 - Performs basic error testing. If ESP and AH were not loaded during parsing routine, then checks to see what types of algorithms are used to determine if security proposal uses AH or ESP.
 - Check if suite structure parameters are null, if not, use the

value to construct the suite list.

- Function: **char * convert_to_uppercase(char * lowercase_string)** – converts a lower case string to an upper case string and returns the string.

- Input:

- char * lowercase – lower case string.

- Output:

- char * - returns uppercase string.

- Process:

- Loops through the input string capitalizing each character with the toupper function to convert the string to uppercase.

- Function: **void parse_ipsec_parameter(char *buff_temp, int *buff_temp_counter, char * sys_para_name, char ** temp_hold, int * success)** – verifies that the tag is the expected tag and then parses the tag and the tag value, storing information in input suite structure char string.

- Inputs:

- char *buff_temp – pointer to the file being parsed.

- int *buff_temp_counter – pointer to index of character in file being parsed.

- char * sys_para_name – pointer to the expected parameter tag name

- char ** temp_hold – pointer to a pointer (used for the purpose of dynamic memory allocation) of char string in suite structure.

- int *success – pointer to an integer used for the success flag.

-Outputs:

- int *buff_temp_counter – pointer to index of character in file being parsed is returned via pointer reference. Pointer may be advance in function.

- char * temp_hold - pointer to a pointer of a character string (used for the purpose of dynamic memory allocation)

in the suite structure returned via pointer reference.

- int *success – pointer to an integer used to hold success flag returned via reference.

- Process:

- Check to see if expected parameter string matches target string in string buffer being parsed. If so :

- Advance index pointer in buffer to tag value – determined by “” in KeyNote/isakmpd.policy syntax.

- Create dynamic memory in input suite structure string for the tag value from buffer.

- Copy tag value to input suite structure string.

- Set success flag to 1 (true).

- Convert tag value to upper case using **convert_to_uppercase()** function.

- If match not found, set success flag to 0 (false) and exit.

- Function: **void parse_ipsec_para_tag(char *buff_temp, int *buff_temp_counter, char * sys_para_name, char ** temp_hold, int sys_para_name_reduced, int * success)** - verifies that the tag is the expected tag and that tag value contains “yes”. If so, parses tag and stores its value in suite structure char string.

- Inputs:

- char *buff_temp – pointer to the file being parsed.

- int *buff_temp_counter – pointer to index of character in file being parsed.

- char * sys_para_name – pointer to the expected parameter tag name

- char ** temp_hold – pointer to a pointer (used for the purpose of dynamic memory allocation) of char string in suite structure.

- int sys_para_name_reduced – integer that holds the string

size of the tag. Either 2 or 3 used for AH, ESP or PFS tags.

- int *success – pointer to an integer used for the success flag.

- Outputs:

- int *buff_temp_counter – pointer to the index of character in file being parsed is returned via pointer reference. Pointer may be advanced in the function.

- char * temp_hold - pointer to a pointer of a character string (used for the purpose of dynamic memory allocation) in the suite structure returned via pointer reference.

- int *success – pointer to an integer used for the success flag returned via reference.

- Process:

- Check to see if expected parameter string matches target string in the string buffer being parsed. If so :

- Advance index pointer in buffer to tag value – determined by “” in KeyNote/**isakmpd.policy** syntax.

- Check to see if the tag value is “yes”. If so:

- Use sys_para_name_reduced to determine the appropriate size of the tag for dynamic memory creation and coping purposes.

- Create dynamic memory for input suite structure string to hold the tag from the buffer (i.e. AH, ESP or PFS). Note: the difference between this function and the previous one, is that the tag value is copied as opposed to the tag.

- Copy tag into input suite structure string.

- Set success flag to 1 (true).

- Convert tag value to upper case using

convert_to_uppercase() function.

- If match not found, set success flag to 0 (false) and exit.

Figure 5.3 provides a diagram of the process.

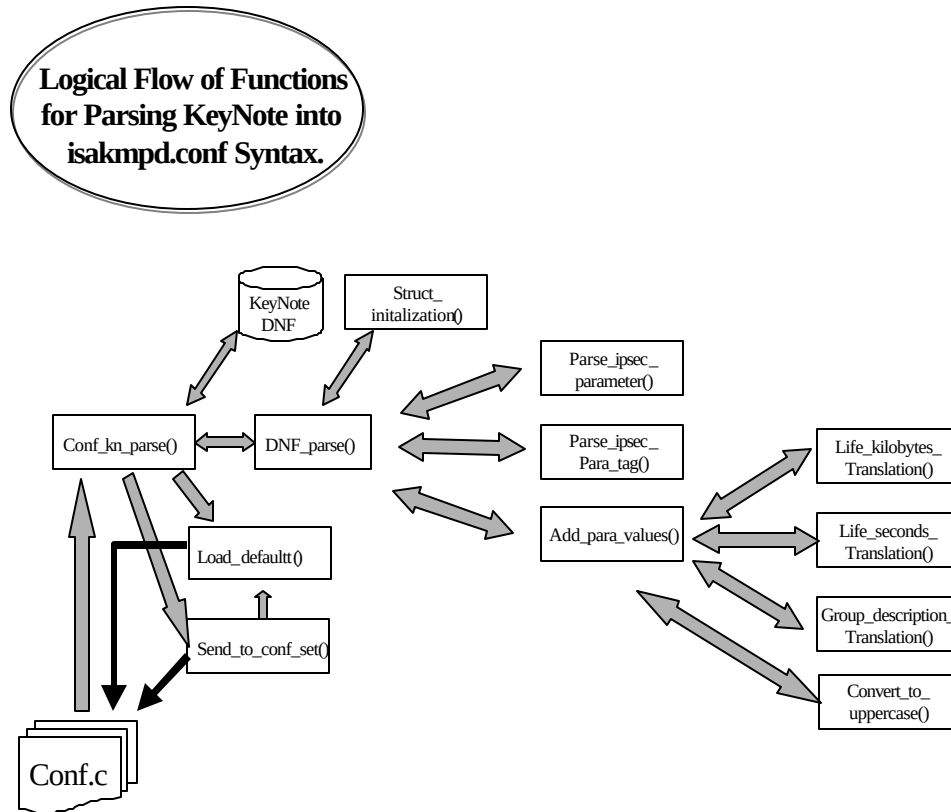


Figure 5.3. Logical Flow of Functions for Parsing KeyNote into isakmpd.conf Syntax.

2.2 Incorporating the Dynamic Parameters into the Security Proposal Loading Process

The last step required in the modification of the security proposal loading process is incorporating dynamic parameters. One mechanism is required to retrieve the current value of the dynamic parameters. Another mechanism will be required to utilize these values in the security proposal parsing process described earlier.

2.2.1 Design for Retrieving Value of Dynamic Parameters

Reusing the module implemented for the retrieval of the dynamic parameters in above section B.1.3, the process retrieves the current value of the dynamic

parameters from a file. An important assumption is made here: when the values of the dynamic parameters change, a process will initiate a reconfiguration to cause the triggering of the security proposal loading process (in order for the security proposal to properly reflect the new dynamic parameter values).

2.2.2 Design for Mechanism for Incorporating Dynamic Parameter Values into the Security Proposal Parsing Process

A mechanism is required to incorporate the values of the dynamic parameters during the security proposal parsing process. For efficiency the mechanism should first scan the DNF security assertion for valid dynamic parameters. If valid values exist then parsing continues. If not then there are no changes relevant to the query and the process skips to the next DNF security assertions.

2.2.2.1 Processing Description

The following is the processing description for the mechanism:

- Load current dynamic parameter values into a memory structure
- Prior to parsing each DNF security proposal assertion, perform a preliminary scan to check for matching dynamic parameter values. If a match exists, continue with the parsing process. If not, advance to the next DNF assertion and return to primary parsing loop to process next available DNF security proposal assertion.

2.2.2.2 Pseudo Code

The loading of dynamic parameter values into a memory structure utilizes the same routine discussed in section B1.3 in this chapter. The following functions are used to scan DNF security proposals for valid dynamic parameters.

- Function: **int dynamic_package_verification(char * buff_temp, int *buff_temp_counter, int buff_temp_end, struct dynamic_packet *package, int package_size, int szkn)** – used to check dynamic parameters of DNF security proposal assertions.

- Input:

- char * buff_temp – character string/buffer used to hold the isakmpd.conf/KeyNote file being parsed.
- int *buff_temp_counter – index used for parsing the buff_temp.
- int buff_temp_end – index to last character of buffer used to check for end-of-file (EOF) condition.
- struct dynamic_packet * package – structure that holds current value of the dynamic parameters.
- int package_size – size of array of dynamic_packet structure.
- int szkn – size of KeyNote file.

- Output:

- int – used as a Boolean flag to indicate if DNF security proposal assertion dynamic parameters match. Return 0 (false). Return 1 (true).

- Process:

- Loop through the array of the dynamic_packet structure:
- Set temp index pointer to the beginning of the DNF security proposal assertion.
- Loop through DNF security proposal assertion:
 - Check for the first character match of dynamic parameter tag. Avoid checking beyond the buffer by using szkn to check for buffer size limit.
 - Call **verify_parameter()** function to check rest of dynamic_parameter tag and tag value. Returns a flag 0-dynamic parameter does not match, 1-

dynamic parameter matches , 2- rest of tag does not match dynamic parameter tag.

- If 0 (dynamic parameter does not match), then advance to the next DNF security proposal assertion by calling **advance_to_end_DNF()** function. Return to calling function.

- If 1 (dynamic parameter matches) then set loop flag to exit DNF security proposal assertion loop and check for other dynamic parameter value matches.

- If 2 (rest of tag does not match dynamic parameter tag) then continue searching through DNF security proposal assertion loop.

- Check for end of DNF security proposal assertion (check for "|"). If found, set exit flag from loop searching through DNF security proposal assertion.

- Check for end-of-file (EOF) using szkn parameter. If found, set exit flag from loop searching through DNF security proposal assertion.

- Advance buff_temp index counter.

- Check to ensure that all dynamic parameter matches were found by comparing match counter to number of dynamic parameters in the array structure.

- If match counter and array size equal, return 1 (true) to calling function.

- Else advance buffer index to the next DNF security proposal and return 0 (false).

- Function: **verify_parameter(char *buff_temp, int**

***buff_temp_counter, char * sys_para_name, char * sys_para_value, int buff_temp_end)** – checks input dynamic parameter tag value and if valid, compares tag value with given value. Returns three possible flag values.

- Input:

- char *buff_temp – string buffer used for KeyNote file.
- int *buff_temp_counter - index of pointer in buff_temp string buffer.
- char * sys_para_name – Dynamic parameter tag
- char * sys_para_value – Dynamic parameter tag value
- int buff_temp_end – index of end-of-file (EOF) in buff_temp.

- Output:

- int – flag with the following three values:
 - 0 (dynamic parameter tag value does not match)
 - 1 (dynamic parameter tag value matches)
 - 2 (dynamic tag does not match).

- Process:

- Compare dynamic parameter tags:
 - If match found :
 - Advance index pointer to the start of the tag value for further comparison.
 - Compare dynamic parameter tag values:
 - If match found, set return flag to 1.

- Else (match not found) set

return flag to 0.

- Else set return flag to 2.

- Return flag.

- Function: **DNF_parse()** – previously described above in section B.2.2.3.3, required additional modifications to fully incorporate the dynamic parameter functionality. The modifications are listed below:

- Prior to perform parsing, a call to

- dynamic_package_verification()** function is made

- to determine if DNF security proposal assertion is

- valid in accordance to current dynamic parameters.

- If so parsing process continues. If not function

- skips parsing process and returns to calling

- function (which will advance to the next DNF security proposal assertion if EOF is not reached).

Figure 5.4 provides a diagram of the process.

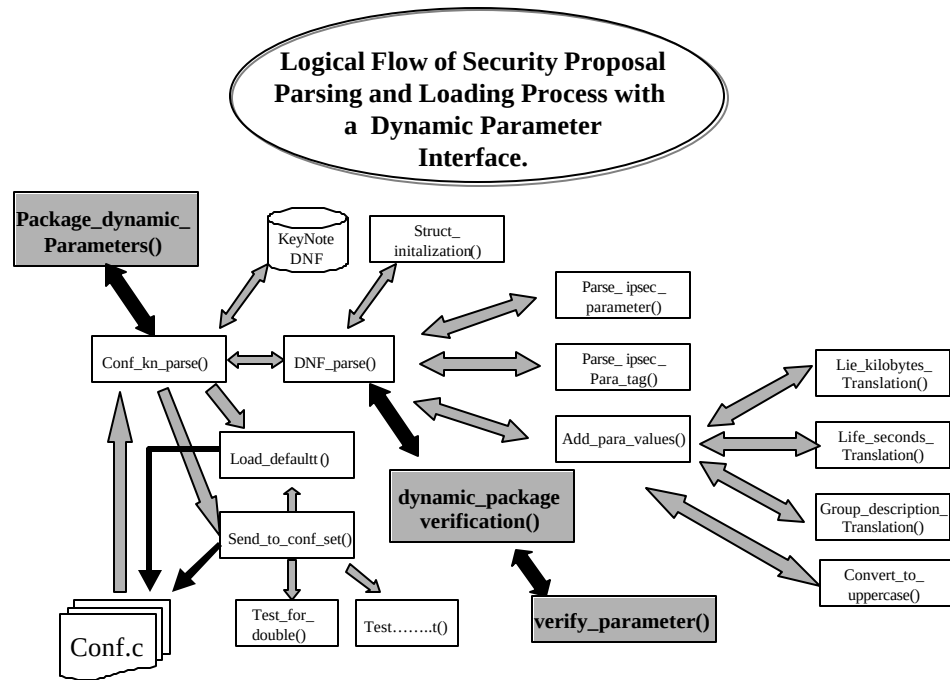


Figure 5.4. Logical Flow of Security Proposal Parsing and Loading Process with the Added Dynamic Parameter Interface.

2.3 Additional Modifications to Further Fine-tune Security Proposal Loading Process

To ensure the security proposal loading process runs smoothly, further fine tuning mechanisms are required. First a default value loading mechanism must be inserted to ensure that proposals that have incomplete fields are filled with default values. Second, the possibility of having more than one identical security proposal must be appropriately handled. Duplicates may result from the added granularity in the security proposal definitions caused by the inclusion of dynamic parameter field attributes into KeyNote.

2.3.1. Default Value Loading Mechanism

In the event of an incomplete security proposal, default values are required to fill the holes in the proposal. This will help to avoid a runtime system crash and/or unstable secure communications. To accomplish this, a default policy must be determined to handle all cases. The mechanism can then either utilize the whole default policy (in cases where insufficient policy is successfully parsed) or portions of the default policy (in cases where security policy is only partially defined).

The following security policy was determined to be default policy. Note: the default setting is set to the least upper bound of all possible proposals providing the highest level of security.

- ESP protected IPsec protocol
- Encryption algorithm: AES
- Authentication Algorithm: SHA
- Utilizing perfect Forward Security
- Transportation Mode: Tunnel
- Life: 3600 seconds
- Group description: MODP_1024

2.3.1.1 Processing Description

The following is the processing description used to perform default loading:

- After all parsing has been performed, check to see if any valid security proposals were created:
- If so call **send_to_conf_set()**.
- Perform a validity check to ensure that enough security parameters exist for the security proposal to be valid:
- If enough parameters exist, continue loading parameters with **conf_set()**.
- If not, abort and call **load_default_sa()** to load default security proposal.
- While sending the security proposal parameters to **conf_set()**, check for empty required parameters. If any are found, fill them with the appropriate values from the default security proposal structure.
- Else load full default security proposal structure.

2.3.1.2 Pseudo Code

The following is a description of the pseudo code:

- Function: **struct suite_struct * initialize_default_suite_profile(struct suite_struct *temp_ss)** – initialize a default suite structure.

- Input:
 - struct suite_struct *temp_ss – pointer to structure to be initialized and loaded with default parameters.
- Output:
 - struct suite_struct * - pointer to structure to be returned.
- Process:
 - create just enough memory as required and copy default string values into structure character strings.
 - return the pointer to the structure.
- Function: **void load_default_sa(int trans, char * section, char * title, struct suite_struct* default_suite_profile)** – loads default security proposal into conf_set().
 - Input:
 - int trans – transaction number required for conf_set()
 - char * section – character string defined in calling function
 - char * title – character string defined in calling function
 - struct suite_struct* default_suite_profile – default suite structure for the default security proposal parameters.
 - Output:
 - void
 - Process:
 - Call conf_set() using the default values from the default suite structure to load default security proposal into memory.
- Modification to Function: **send_to_conf_set(..., struct suite_struct * default_suite_profile)** – modifications to send_to_conf previously described will be required to handle default loading of parameters for incomplete security

proposals.

- Input:

- struct suite_struct * default_suite_profile – structure for default security proposal.

- Output: void.

- Process:

- Test if no suites were defined. If so, abort and call

load_default_sa()

- Call **test_suite_structure()** to verify that parsed security proposal(s) contains sufficient parameters to be used:

- If 0 (false) is returned, abort and call load_default_sa()

- If 1 (true) is returned, continue.

- Loop through array of security proposals:

- Verify that each required parameter has a value.

- If not, use the default structure to load appropriate default value.

- If so, continue.

- Call **conf_set()** as needed to load security proposal.

- Function: **int test_suite_structure(struct suite_struct * suite_profile, int struct_size)** – used to verify that suite structures contain sufficient parameter to be valid.

- Input:

- struct suite_struct * suite_profile – pointer to list of security proposal structures.

- int struct_size – size of array list

- Output:

- int – Boolean return flag: 0 (false), 1 (true).

- Process:

- Loop through the list of security proposal structures.

- Test if suite_name is NULL. If so, abort and return false.

- Test if protocol_id is NULL. If so, abort and return false.
- Test if transform_id is NULL. If so, abort and return false.
- Return true.

Figure 5.6 provides a diagram of the process.

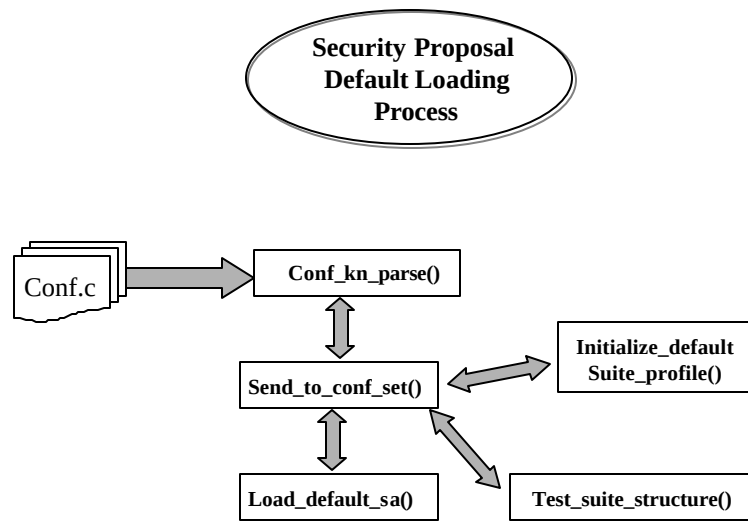


Figure 5.6. Security Proposal Default Loading Process.

2.3.2 Mechanism To Handle Duplicate Security Proposals

With the incorporation of dynamic parameters into the KeyNote structure, the possibility of duplicate security proposals being generated exists. To maintain an efficient security process, a mechanism must be implemented that ensures that duplicate security proposals are not generated. To efficiently manage this problem, the mechanism must be inserted into the parsing process so that it will skip ahead to the next DNF security proposal when a duplicate security proposal is recognized.

2.3.2.1. Processing Description

Once valid security proposals have been initially parsed from KeyNote, a function is required to eliminate duplicate security proposals.

The following is the processing description of the

procedure used to check for and handle duplicate security proposals:

- The process will utilize a list of parsed security proposals. As new security proposals are added, they must be compared to the current members of the list to check for duplication.
- In order to check for duplicates, the parsed security parameters must be checked. The following are the security parameters used for the proof of concept in this research:
 - ESP encryption algorithm
 - ESP authentication algorithm
 - AH authentication algorithm
 - encapsulation mode
 - group description
 - life time
 - Perfect Forward Security
 - Key Length

2.3.2.2. Pseudo Code

The following is a description of the pseudo code:

- Function: **int duplicate_sa (struct suite_struct * suite_list, int suite_count, struct suite_struct *suite_profile)** – compares the security proposals list with new security proposal for duplicates.

- Input:

- struct suite_struct * suite_list – pointer to a security proposal list/array.
- int suite_count – number of security proposal in the array.
- struct suite_struct *suite_profile – pointer to the new security proposal.

- Output:

- Int – used as Boolean flag. Returns 1 (true) if new security proposal is a duplicate. Returns false if new

security proposal is not a duplicate.

- Process:

- For loop is used to traverse the array of security proposals.

- Each member of the array is compared with the new security proposal. The elements of the security proposal are compared via **strcmp()** utilizing the embedded if/else statements. For each **strcmp** test, a NULL test is performed first in the if statement to avoid a potential run-time error (caused by **strcmp(NULL)**). If NULL is found, the rest of the **strcmp** test is by-passed and the next if statement is executed. This continues until all security proposal conditions are checked and found to be identical or not. If the process successful passes all if nested statements, true (1) is returned to the calling function indicating a the new security proposal is a duplicate. Otherwise, the function continues checking all security proposals in the array. If the complete array has been checked without finding a duplicate, false (0) is returned to the calling function. The elements of the security proposal checked are:

- AH Authentication algorithm (ah_auth_alg).
- Encryption algorithm (esp_enc_alg).
- ESP Authentication algorithm (esp_auth_alg).
- Encapsulation mode (encapsulation_mode).
- Group description (group_desription).
- Life time title (life).
- Life time type (life_type).
- Life time duration (life_duration).
- Key length (key_length).

- Figure 5.7 provides a diagram of the process.

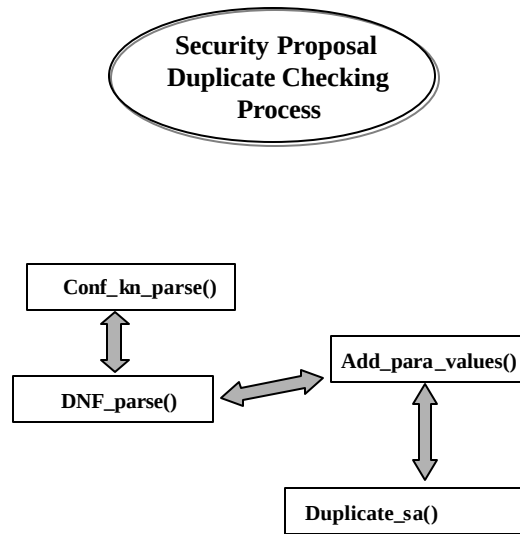


Figure 5.7. Security Proposal Duplicate Checking Process.

D. CONCLUSION

This chapter described the implementation of dynamic parameterization of IPsec, and includes the methodology, processing description, and pseudo code summaries for each component. The goal was to modify the current OpenBSD implementation of IPsec to incorporate additional fields representing dynamic parameters. The modification was performed in two phases: the first provided granularity to KeyNote/**isakmpd.policy**, and the second streamlined and incorporated parameterization into the **isakmpd.conf**-KeyNote security proposal set loading process. Additional error-checking functionality was added for security proposal duplicates, and for incomplete security proposals. These included the capability of loading default fields and proposals as required.

To demonstrate the work represented in this chapter, an interface was required. In the following chapter I will discuss the design and implementation of a Java graphical user interface that enables users to run IPsec while invoking dynamic parameters to adjust security parameters in accordance with the security policy.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. GRAPHICAL USER INTERFACE (GUI) DEMOSTRATION

A. INTRODUCTION

The current implementation of OpenBSD IPsec requires a complex sequence of commands via command-line prompts to establish a secure connection between peers. In the event of a dynamic parameter change requiring the reconfiguration of IPsec, a series of commands and scripts are used. This procedure is very time intensive and requires considerable operating system and environmental knowledge. Also, the process of demonstrating the results of this thesis was challenging without an available graphical user interface (GUI) representation. As a result, a more user-friendly graphical oriented interface was designed and developed to allow users, with limited knowledge of OpenBSD operating environment, to operate and observe the security mechanism and its dynamic parameterization.

This chapter will review the existing command line environment and will describe the Graphical User Interface (GUI) to simplify IPsec control and use.

B. COMMAND-LINE ENVIRONMENT

The OpenBSD operating environment, true to its UNIX roots, is controlled via the command-line for most operations. The typical commands used for operating the IPsec mechanism are explained below.

The following sections are referenced from

(<http://www.openbsd.org/faq/faq13.html>).

1. IPsec System Flush

Prior to initializing and starting the IPsec process, all previous security associations and security rules need to be removed from all databases and cached memory. This will allow new security associations to be generated. This clean-up and refresh process is called a “flush”. The syntax for the command is : **ipsecadm flush**.

2. Setting Up and Mounting the Security Policy Database

Prior to starting the IPsec process, the Security Policy Database (SPD) must be populated with a cached version of the peer connection and security attributes. Otherwise the IPsec will not properly secure network communications in accordance with the

defined security policy. In order to do so, two tasks must be performed: setup IPsec flows and mount the security policy database in the kernel (SPD). Note, that the SPD is updated periodically, with new security association (SA) security parameter index (SPI) references after successful peer negotiations are completed. The information found in the SPD can be derived from **iskampd.conf** (IPsec configuration file) and **iskampd.policy** (KeyNote). Future research should be performed to design a method of automating the population of the security policy data base based on entries found in **iskampd.conf** and **iskampd.policy**. (See Future Work Chapter Seven). Note that security associations (SA) may also be entered manually. However, this was not within in the scope of my research and, therefore, I did not include manual keying in the design or implementation.

To set up IPsec flows, the following information is required:

- Protocol type (ESP/AH).
- Destination IP address.
- Transport port (indicates the method of communication i.e. Finger or Telnet) .
 - Source IP Address.
 - Direction (In or Out).
 - Policy Action (Deny, Allow, Require, Acquire).
 - SPI (Security Parameter Index) – will be generated and inputted after SA's are created from peer negotiations.

The syntax to create IPsec flows is as follows:

ipsecadm flow -proto -dst -spi -transport -src -<direction> -<Policy Action>.

Flows can be generated manually via the command prompt calls or via a script. Below is the example of the IPsec flow script used in the testing phase of this thesis. The script is executed by the following syntax: **sh vpn_28_ah_a** (name of the saved file listed below).

```
#!/bin/sh
#Set-up flows for the two specific hosts
#Use for defining applications FINGER and TELNET
# ESP for TELNET
# AH for FINGER
#      -dport for egress traffic
#      -sport for ingress traffic

# Local and remote hosts
LOCAL_HOST=131.120.8.91
```

```

REMOTE_HOST=131.120.8.95

ipsecadm=/sbin/ipsecadm

#
# Create the host-to-host flow
#

#egress flow for finger
$ipsecadm flow -dst $REMOTE_HOST -proto ah \
               -addr $LOCAL_HOST 255.255.255.255 $REMOTE_HOST
255.255.255.255 \
               -transport tcp -dport 79 \
               -src $LOCAL_HOST -out -require

#ingress flow for finger
$ipsecadm flow -dst $REMOTE_HOST -proto ah \
               -addr $REMOTE_HOST 255.255.255.255 $LOCAL_HOST
255.255.255.255 \
               -transport tcp -sport 79 \
               -src $REMOTE_HOST -in -require

#egress flow for telnet
$ipsecadm flow -dst $REMOTE_HOST -proto esp \
               -addr $LOCAL_HOST 255.255.255.255 $REMOTE_HOST
255.255.255.255 \
               -transport tcp -dport 23 \
               -src $LOCAL_HOST -out -require

#ingress flow for telnet
$ipsecadm flow -dst $REMOTE_HOST -proto esp \
               -addr $REMOTE_HOST 255.255.255.255 $LOCAL_HOST
255.255.255.255 \
               -transport tcp -sport 23 \
               -src $REMOTE_HOST -in -require

exit 0

```

Note: When creating scripts within the OpenBSD environment, it is important to remember the following steps:

- **chmod 755 <script name>** is used to change the scripts default permissions and enable the script to be executed.
- **./<script name>** is the syntax for executing a script.

Once the IPsec flows have been created, they need to be mounted in the kernel (SPD). The following is the syntax required: **mount -t kernfs /kern /kern.**

3. IPsec Execution

To start the IPsec mechanism there are numerous syntaxes possible, depending on

the level of debugging desired. Since OpenBSD's IPsec is still considered to be under development, a debugging mode is very useful to examine functionality and operation. The following are typical syntaxes:

- **isakmpd** – starts the IPsec mechanism without any debugging functionality.
- **isakmpd -d -DA=99** – starts the IPsec mechanism with full debugging functionality. All levels of debugging messages will be displayed.
- **isakmpd -d -DA=99 -D1=<Debug Number>** - starts the IPsec mechanism with limited debugging functionality depending on the given debug number.

4. IPsec Connection Termination

There are numerous methods for terminating an existing IPsec connection. One way involves using **CTRL-C** in the shell to terminate the IPsec process and established connections. A similar result can be achieved by simply killing the IPsec process from the process list. This is performed by using **ps -al** to locate IPsec process id and then using **kill <id number>** to terminate the process.

A gentler approach allows the user to terminate security association connections individually. In case the user must track the SA connection number index. The typical progression of index numbers starts at zero and advances by one for both new connections and terminations. For example, the generation of an SA for telnet communication between two peers advances the index number to from zero to one. The generation of another SA for finger communication between two peers advances the index from one to two. The termination of an existing SA (either of the previously generated SAs) advances the index from two to three. The termination of the last SA advances the index from three to four. This process continues until the IPsec mechanism is halted, which resets the index to zero.

5. Display SPD

During development, it often becomes necessary to review the entries currently existing in the SPD. The command-line method to perform this is based on the UNIX **netstat** command, which displays routing tables. By refining the command using switches, it is possible to display only the routing information specific to the IPsec mechanism. The syntax is as follows: **netstat -rn -f encap**

The following is an example of **netstat -rn -f encap** between two peers for telnet

and finger communications:

Routing tables

Encap:	Source	Port	Destination	Port	Proto	SA(Address/Proto/Type/Direction)
	131.120.8.95/32	23	131.120.8.91/32	0	6	131.120.8.95/50/require/in
	131.120.8.95/32	79	131.120.8.91/32	0	6	131.120.8.95/51/require/in
	131.120.8.91/32	0	131.120.8.95/32	23	6	131.120.8.95/50/require/out
	131.120.8.91/32	0	131.120.8.95/32	79	6	131.120.8.95/51/require/out

6. Display SAD

To ensure that the IPsec mechanism is working properly, it is often important to view the existing security associations (SA's) stored in the SAD. The Open BSD IPsec mechanism stores the SPD in the following file: **/kern/ipsec**.

When no SA's exist in the file, the file contains the following entry:

Hashmask: 31, policy entries: 0.

When, for example, two SAs exist (one for telnet and the other for finger communication) the /kern/ipsec file contain the following:

```
Hashmask: 31, policy entries: 4
SPI = c322801f, Destination = 131.120.8.91, Sproto = 51
  Established 55 seconds ago
  Source = 131.120.8.95
  Flags (00001082) = <tunneling>
  Crypto ID: 4
  xform = <IPsec AH>
    Authentication = <HMAC-SHA1>
  577 bytes processed by this SA
  Expirations:
    Hard expiration(1) in 3545 seconds
    Soft expiration(1) in 3185 seconds
SPI = bd35c96d, Destination = 131.120.8.95, Sproto = 51
  Established 55 seconds ago
  Source = 131.120.8.91
  Flags (00001082) = <tunneling>
  Crypto ID: 3
  xform = <IPsec AH>
    Authentication = <HMAC-SHA1>
  446 bytes processed by this SA
  Expirations:
    Hard expiration(1) in 3545 seconds
    Soft expiration(1) in 3185 seconds
SPI = 9dde8de1, Destination = 131.120.8.91, Sproto = 50
  Established 66 seconds ago
  Source = 131.120.8.95
  Flags (00001082) = <tunneling>
```

```

Crypto ID: 2
xform = <IPsec ESP>
    Encryption = <3DES>
    Authentication = <HMAC-SHA1>
1248 bytes processed by this SA
Expirations:
    Hard expiration(1) in 1134 seconds
    Soft expiration(1) in 1014 seconds
SPI = 95bb697c, Destination = 131.120.8.95, Sproto = 50
Established 66 seconds ago
Source = 131.120.8.91
Flags (00001082) = <tunneling>
Crypto ID: 1
xform = <IPsec ESP>
    Encryption = <3DES>
    Authentication = <HMAC-SHA1>
1408 bytes processed by this SA
Expirations:
    Hard expiration(1) in 1134 seconds
    Soft expiration(1) in 1014 seconds

```

7. **tcpdump**

To verify packet security (either through encryption and/or authentication), a packet sniffer tool can be used to enable the user to view exchanged packets. **Tcpdump**, a packet sniffer utility, is available on OpenBSD. Syntax for the command is: **tcpdump – N host <peer A IP address> and <peer b IP address>**.

The **–N** switch is used to reduce Domain name qualification to make output more readable. The **host** switch to identify peer IP addresses.

C. **GRAPHICAL USER INTERFACE (GUI) DEMONSTRATION**

For many users, the above sequence of commands may be overwhelming and time consuming. A graphical user interface (GUI) would reduce the level of confusion and make the IPsec mechanism easier to control and use.

1. **Goal**

The goal of the GUI demonstration is to provide users with an ability to demonstrate and use the IPsec mechanism, and fully understand the successes in research and development that have been performed in this thesis.

2. **Mechanism of Demonstration**

Java was chosen as the demo’s development language because of its inherent graphical representation ability. OpenBSD utilizes Kaffe’s version of Java. In order to incorporate the full graphical capability of Java, a “Kaffe-friendly” SWING package was installed.

2.1 Run Time Execution of Shell Commands from Parent Java Process

The following section is referenced from (<http://java.sun.com/products/jdk/1.2/docs/api/>)

The demonstration module requires the ability to execute command-line calls. This is accomplished by using the following:

- Static declaration of an instance of RunTime: **Static Runtime rt**. Every Java application has a single instance of the RunTime class. This instance enables the application to execute command-line calls from within the application. It is important to note that an application cannot create its own instance of the RunTime class.
- Use of an array of strings to construct a shell script command to enable run time execution. In order for a command to be executed properly, the command must be incorporated into a string. The following example demonstrates a run time execution of IPsec:

```
private static Runtime rt;  
String[] s2 = new String[3];  
s2[0]= new String("/bin/sh");  
s2[1] = new String("-c");  
s2[2] = new String("ipsec");  
rt.exec(s2);
```

- Some commands require switches and additional data in order to be executed properly. In this case, there are two methods that are used in this research:
 - Command, switches and additional data are included in one string. An example of this is the use of the **ps** command (i.e. **ps -ax | grep isakmpd | grep -v grep**).
 - Command, switches and additional data are broken into separate string in the array. An example is the use of **tcpdump** command (i.e. **tcpdump -N 131.120.8.95 and 131.120.8.9**).
- If certain commands require immediate execution without the parent

routine proceeding in command execution, **waitFor** method can be added to the **exec(<string>)** to pause the process until the shell commands completes its processing. The following example demonstrates a run time execution of the IPsec flush routine, which would require the parent process to wait until shell's completion:

```
private static Runtime rt;
String[] s2 = new String[3];
s2[0]= new String("/bin/sh");
s2[1] = new String("-c");
s2[2] = new String("ipsecadm flush");
rt.exec(s2);
```

- All runtime calls require try-catch blocks.
- To receive the current instance of Runtime, the following is the syntax is used: **rt = Runtime.getRuntime();**

Execution of all command line calls must utilize the RunTime class and methods.

2.2 Graphical Demonstration Components

The demo consists of the following components:

- **Welcome Screen** – provides the user with a demo title screen with the ability to incorporate logos and credits as needed.
- **Main Menu Console** – provides the user with all available IPsec choices, and dynamic parameter console:
 - **Start IPsec** – initiates the IPsec mechanism.
 - **Display SADB** – displays current valid security associations.
 - **Display SPD** – displays contents of the SPD.
 - **Display TCPDUMP** – provides the user with a Java JFrame console to view the output of **tcpdump**.
 - **Dynamic Parameter console** – provides user with a console to make a selection on the dynamic parameters.
 - **Display Current Security Policy** – provides the user with a console to view the current security policy in

3. Graphical Demonstration Components

The following sections describe in some detail each of the above listed menu choices. The functionality of the demo is broken into the following java classes:

- **Demo.java** – creates the demo welcome screen and the menu choices in a Java JFrame console.
- **Demo_Support_Functions.java** – holds all functions used by other classes.
- **DP_Console.java** – creates dynamic parameter selection interface in a Java JFrame console.
- **Ipssec_Info.java** – creates a continuously updated output of the security association database (SAD) in a formatted Java JFrame console.
- **Tcpdump.java** – creates a continuously updated output of the network packets captured by **tcpdump** in Java JFrame console.
- **SPFK.java** – creates a Java JFrame console for formatting and displaying the security policy found in **iskampd.policy/KeyNote**.
- **SPD.java** - creates a Java JFrame console for formatting and displaying the security policy found in the security policy database (SPD).

3.1. demo.java

This class creates and handles the generation of the welcome screen and menu choices.

3.1.1 Design Approach

The initial design was to incorporate both the OpenBSD and NPS CISR logos on the welcome screen. However, Kaffe's java library did not properly support graphic logos (i.e. jpegs, bmp, .gif files). Therefore, the welcome screen consists of large font title and a **continue** button.

The menu screen consists of buttons allowing the user to make selections using a mouse. The menu also has an error checking display, which provides the user with feedback as necessary. Further details on error checking will follow.

3.1.2 Processing Description

The following is the processing description for the method:

- A welcome Java JFrame console is generated with a title and a **continue** button linked to an action handler.
- Once the user selects the **continue** button, the welcome screen is erased and the menu choice JFrame is painted.
- The menu JFrame provides the user with menu selections via buttons:
 - **Start IPsec** – starts the IPsec mechanism. If the IPsec mechanism is already started, an error message is displayed in the error message display panel.
 - **Display SAD** – starts the **ipsecinfo.java** thread to provide the user with a continuously updated display of the currently existing security associations (SA) via a JFrame.
 - **Display SPD** – creates a formatted display for the currently existing entries in the Security Policy Database (SPD) via a JFrame.
 - **Display Security Policy** – parses **iskampd.policy/KeyNote** and displays information in a JFrame.
 - **Display TCPDUMP** – starts the **tcpdump** mechanism and displays continuous output in a Java JFrame.
 - **Dynamic Parameters Interface** – provides the user with a console to select and submit dynamic parameters. If the user does not select two dynamic parameters prior to pressing the submit button, an error message is displayed in the error display panel.

3.1.3 Pseudo Code

The following is a description of the pseudo code used in

demo.java:

- Class: **demo** extends JFrame.
 - Variables used throughout the class, are declared as Global.

- Method: **demo** class constructor – initializes the welcome screen.

- Initializes title to JFrame
- Adds a window adapter to handle window exiting by calling **dispose()**.
- Creates welcome message and displays on JFrame.
- Creates the error display panel.
- Creates **continue** button with an action handler.
- Initialize menu choices JFrame title and buttons.

- Method: **initialize_connection_index_file()** – resets connection index counter to zero in the connection index counter file.

- Input: none.
- Output: none.
- Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer, file output stream and a print stream to **/root/demo/connection_number**
 - Writes the number “0” to the file.
 - Closes the file.

- Method: **load_dp_file(String nm, String sl)** – accepts network mode and security level inputs and writes them into the dynamic parameter file.

- Input:
 - String nm- network mode value.
 - String sl – security level value.
- Output: none.
- Process:
 - The following process is enclosed in a try-

catch block.

- Creates a file pointer, file output stream and a print stream to **/usr/src/sbin/isakmpd/dynamic_parameters**

- Writes the network mode value and security level value to the file in the following format:

network_mode = <network mode value>

security_level = <security level value>

- Closes the file.

- Class: **ContinueButtonHandler** implements ActionListener

- Method: **actionPerformed(ActionEvent e)** – action handler for the **continue** button on the welcome JFrame.

- Process:

- Erases existing components on the Welcome JFrame.

- Creates menu title, menu choices buttons and displays on menu choices JFrame.

- Adds an action handler to all buttons.

- Class: **SPButtonHandler** implements ItemListener

- Method: **actionPerformed(ActionEvent e)** – action handler for the **start IPsec** button.

- Process:

- Checks to see if IPsec is currently running by calling **demo_support_functions.daemon_running()**.

- If IPsec is already running then displays error message in error display panel and exit handler routine.

- Otherwise, proceeds.

- Loads default network mode and security level values using **load_dp_file()**.
- Flushes the IPsec mechanism of previous existing security associations (SA) by calling **demo_support_functions.flush_ipsec()**.
- Mounts the kernel by calling **demo_support_functions.mount_kern()**.
- Loads the SPD policy values by calling **demo_support_functions.load_spd()**.
- Class: **DSButtonHandler** implements ActionListener
 - Method: **actionPerformed(ActionEvent e)** – action handler for **Display SAD** button.
 - Process:
 - Instantiates and start **ipsecinfo.java** thread.
- Class: **SPDButtonHandler** implements ActionListener
 - Method: **actionPerformed(ActionEvent e)** – action handler for **Display SPD** button.
 - Process:
 - Instantiates **SPD.java** class to display the SPD.
- Class: **DSPButtonHandler** implements ActionListener
 - Method: **actionPerformed(ActionEvent e)** – action handler for **Display Security Policy** button.
 - Process:
 - Instantiates **SPFK.java** class to display the Security Policy.
- Class: **TCPButtonHandler** implements ActionListener
 - Method: **actionPerformed(ActionEvent e)** – action handler for **Display tcpdump** button.
 - Process:

- Instantiates and start **tcpdump.java** thread.
- Class: **DPButtonHandler** implements ActionListener
 - Method: **actionPerformed(ActionEvent e)** – action handler for **Dynamic Parameterization** button.
 - Process:
 - Instantiates **dp_console.java**.
- Class: **INFButtonHandler** implements ActionListener– action handler for **Stop IPsec** button.
 - Method: **actionPerformed(ActionEvent e)** – action handler for Exit menu.
 - Process:
 - Checks to see if IPsec is currently running by calling **demo_support_functions.daemon_running()**.
 - If so, then tears down existing connections by calling **demo_support_functions.Tear_down_connections()** and stopS the IPsec mechanism by calling **demo_support_functions.stop_ipsec()**.
- Class: **EXButtonHandler** implements ActionListener– action handler for **Exit** button.
 - Method: **actionPerformed(ActionEvent e)** – action handler for Exit menu.
 - Process:
 - Checks to see if IPsec is currently running by calling **demo_support_functions.daemon_running()**.
 - If so, then tears down existing connections by calling **demo_support_functions.Tear_down_connections()** and stopS the

IPsec mechanism by calling **demo_support_functions.stop_ipsec()**.

- Exits the program.

- Method: **static main(String args)** – the main program of the demo class.

- Process:

- Gets the current runtime instance by calling **getRuntime()**.

- Initiates the **demo.java** class.

3.2. demo_support_functions.java

This class holds commonly used functions by the other classes.

3.2.1 Design Approach

In accordance with proper software engineering techniques, commonly used functions should be shared among classes. This will reduce the code length and make code management easier. This module includes those functions.

3.2.2 Processing Description

No processing description is necessary since this class performs no tasks other than hold all functions used by other classes.

3.2.3. Pseudo Code

The following is a description of the pseudo code for **demo_support_functions.java**:

- Class: **demo_support_functions**.

- Variables used throughout this class, are declared as Global.

- Method: **demo_support_functions** constructor.

- Process:

- Gets the current instance of Runtime by calling **Runtime.getRuntime()** .

- Method: **flush_ipsec()** – generates the run time commands to flush the IPsec mechanism.

- Input: none.
- Output: none.
- Process:
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section B 2.1 in this chapter for more details) **of ipsecadm flush**.
 - Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.
- Method: **load_spd()** – generates the run time commands to load SPD with the security policy.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for more detail) of **sh vpn28_ah_a..**.
 - Note that **waitFor()** is not required since it is not necessary for the parent process to wait for its completion.
- Method: **mount_kern()** – generates the run time commands to mount the kernel.
 - Input: none.

- Output: none.
- Process:
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for more detail) of **sh /root/mount_kern**. This is the name of a script that contains the following code: **mount -t kernfs /kern /kern**.
 - Note that **waitFor()** is not required since the it is not necessary for the parent process to wait for its completion.
- Method: **start_ipsec()** – generates the run time commands to start the ipsec mechanism.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section B 2.1 in this chapter for more detail) of **sh /root/mount_kern**. This is the name of a script that contains the following code: **ipsec**.
 - Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.

- Method: **stop_ipsec()** – generates the run time commands to stop the IPsec mechanism.

- Input: none.

- Output: none.

- Process:

- The following process is enclosed in a try-catch block.

- Creates a file pointer and file input stream to **/var/run/isakmpd.pid**.

- Reads in the process id found in the file, which will be used to kill the IPsec process.

- Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for more detail) of **sh /root/mount_kern**. This is the name of a script that contains the following code: **kill <process id>**

- Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.

- Closes the file.

- Method **read_connection_index_file()**– reads in the current value of the index counter from a file.

- Input: none.

- Output: none.

- Process:

- The following process is enclosed in a try-catch block.

- Creates a file pointer and file input stream to **/root/demo/connection_number**.

- Reads in the current connection index counter and store in a global variable.
- Closes the file.
- Method: **write_connection_index_file()**– writes the current value of the index counter to a file.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer, file output stream, and print stream to **/root/demo/connection_number**.
 - Writes the current connection index counter and store in a global variable.
 - Closes the file.
- Method: **daemon_running()** – checks to see if the IPsec process is currently running.
 - Input: none.
 - Output:
 - Boolean result – true if IPsec is currently running and false otherwise.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for more detail) of **sh /root/mount_kern**. This is the name of a script that contains the following code: **ps -**

ax | grep iskmpd | grep -v grep > daemon_search. This command performs a ps list (active processes) and then retrieves only the entries that have the word **iskmpd** and pipes them into **daemon_search** file. The switch **-v grep** removes the call from the **ps** listing.

- Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.
- Creates a file pointer and file input stream to **/root/demo/daemon_search**
- Retrieves the current file size of **daemon_search**.
- If the file size is zero, the IPsec mechanism is not running. Returns false.
- If the file size is greater than zero, the IPsec mechanism is running. Returns true.
- Closes the file.
- Method: stop_tcpdump() – terminates tcpdump process.
 - Input: none.
 - Output: none.
 - Process:
 - While loop – until Boolean terminate flag is set.
 - The following process is enclosed in a try-catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for

more detail) of **ps -ax | grep tcpdump | grep -v grep | grep -v /bin/sh**. This command performs a ps list (active processes) that contain the tcpdump title.

- Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.
 - Creates a data stream pointer to the above created process stream.
 - Retrieves the process id of the running tcpdump process.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section 2.1 for more detail) of **kill + <tcpdump process>**. This command kills the tcpdump process.
 - Note: during testing it was discovered that when executing the tcpdump script two tcpdump processes are created. Therefore this routine loops until all tcpdump process are killed.
 - When no more tcpdump processes are found, set Boolean flag to exit while loop.
- Method: **tear_down_connections()** – tears down existing security association (SA) connections between peers.
- Input: none.

- Output: none.
- Process:
 - Calculates existing connections (security association) by calling **calc_connection()**.
 - Retrieves the current connection index counter by calling **read_connection_index_file()**.
 - The following process is enclosed in a try-catch block.
 - An assumption is made here that every SA consist of pair of two connections.
 - Divide number of current connections by two.
 - If the number of current connection equals zero, no security association currently exist. Exits method.
 - Otherwise, proceeds.
 - For-loop for the number of connections (divided by two).
 - Writes the connection number index to a file to terminate that connection by calling **write_to_fifo()**;
 - Checks to see if in the last iteration of the loop. If so, exit loop by using **break**.
 - Otherwise, increases connection index counter by one.
 - For information on connection counter index refer to section A.5 in this chapter.

- Creates a file pointer and file input stream to **/var/run/isakmpd.pid**
- Reads in the process id for **Psec** from **isakmpd.pid**.
- Closes the file.
- Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section B 2.1 for more detail) of **kill -HUP <process id>**. The **-HUB** switch causes a “hang-up” action to be performed on the connection and instructs it to reread the configuration files.
- Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.
- The current index counter is then written to the **connection_number** file by calling **write_connection_index_file()**.
- Method: **write_to_fifo()**– uses the current connection index to write teardown instructions to the IPsec mechanism in **/var/run/isakmpd.fifo** file.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer, file output stream, and print stream to **/var/run/isakmpd.fifo**.
 - Writes the current connection index counter and the tear down instruction by

writing **t Connection-+ <connection number index >**.

- Connection number is stored in a global variable.

- Closes the file.

- Method: **synchronized copy_kern_ipsec()** – copies the file **/kern/ipsec** (file containing the current security associations) to **/root/demo/tempipsec** (file used to parse security associations). This method is synchronized to avoid a deadlock when various threads, created by the demo, compete for this function.

- Input: none.

- Output: none.

- Process:

- The following process is enclosed in a try-catch block.

- Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above section B 2.1 in this chapter for more detail) of **cp /kern/ipsec /root/demo/tempipsec**.

- Note that **waitFor()** is required since it is necessary for the parent process to wait for its completion.

- Method: **calc_connection()**– calculates the number of existing security associations (SA) by reading **/root/demo/tempipsec** and parsing the information to count existing SAs.

- Input: none.

- Output: none.

- Process:

- The following process is enclosed in a try-catch block.
- Creates a file pointer and file input stream to **/root/demo/tempipsec**
- Performs a “thread sleep” operation for one second to allow time for other processes to use the **/root/demo/tempipsec**.
- Reads in the contents of the file into a StringTokenizer buffer.
- Closes the file.
- While loop until all tokens have been parsed.
 - Advances to next token.
 - Compares token with “SPI”. If match found, increment connection_counter (a global variable).

3.3 dp_console.java

This class generates the dynamic parameter selection interface with radio buttons for network mode and security level selection. The user must select a network mode and security level and press the **submit** button. The interface also has an **exit** button to close the window.

3.3.1 Design Approach

The dynamic parameter console provides the user with a selection mechanism for network mode and security level. Error checking is provided to ensure that the user selects one of each. To activate the selection, the user must press the **submit** button. Action handlers are provided to initiate the change in mode and level.

3.3.2. Processing Description

The following is the processing description used to design and develop **dp_console.java**:

- The dp_console is initialized as a JFrame with radio

buttons for network mode and security level selections.

- The user must select a network mode and a security level before pressing the submit button. If not, an error message is displayed.
- Once a valid selection has been made, the submit button generates a signal to the IPsec mechanism to reconfigure in accordance to the new network mode and security level.

3.3.3 Pseudo Code

The following is a description of the `dp_console` class pseudo code:

- Class: **dp_console** extends JFrame
 - Most variables used throughout this class are Global.
 - Method: **dp_console()** constructor – initializes the dynamic parameter selection interface.
 - Process:
 - Initializes interface JFrame.
 - Adds a window adapter to handle window exiting by calling **dispose()**.
 - Creates titles for network mode and security level.
 - Creates radio buttons with action handlers for network modes and security levels.
 - Creates **Submit** button with an action handler.
 - Creates an **Exit** button with an action handler to close the window by calling **dispose()**.
 - Initializes global network mode and security level variables.
 - Instantiates an instance of **demo_support_functions**.

- Gets the current Runtime instance by calling **getRuntime()**.
- Method: **start_dp_console()** – makes the **dp_console** visible.
 - Input: none.
 - Output: none.
 - Process:
 - Sets the JFrame to visible.
- Method: **reset_error_panel()** – clears error message panel.
 - Input: none.
 - Output: none.
 - Process:
 - Erases any existing text in the error message panel.
- Method: **set_dynamic_parameters()** – stores the value of the dynamic parameters in a file.
 - Input: none.
 - Output: none.
 - Process:
 - Ensures that that old file is deleted by calling **delete_file()**.
 - Writes the current value of the dynamic parameters in a file by calling **write_dynamic_parameters_file()**.
- Method: **write_dynamic_parameters_file** – writes the global current value of the dynamic parameters to **/usr/src/sbin/isakmpd/dynamic_parameters**.
 - Input: none.
 - Output: none.
 - Process:

- The following process is enclosed in a try-catch block.
- Creates a file pointer, file output stream, and print stream to **/usr/src/sbin/isakmpd/dynamic_parameters**.
- Writes the current value of the network mode and security level stored in global variables to the file.
- Closes the file.
- Method: **print_dynamic_parameters_file()** – displays the current value of network mode and security level to the system console for trouble shooting purposes.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer, and file input stream to **/usr/src/sbin/isakmpd/dynamic_parameters**.
 - Reads in the current value of the network mode and security level and displays the values via the system console.
 - Closes the file.
- Method: **read_dynamic_parameters_file()** – reads in the value of network mode and security level from file: **/usr/src/sbin/isakmpd/dynamic_parameters** and stores them in the class global variables respectively.
 - Input: none.

- Output: none.
- Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer, and file input stream to
/usr/src/sbin/isakmpd/dynamic_parameters.
 - Reads in the current value of the network mode and security level and stores them in their global variables respectively.
 - Closes the file.
- Method: **delete_file()** – deletes existing dynamic parameter file.
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer to
/usr/src/sbin/isakmpd/dynamic_parameters.
 - The file is deleted.
- Class: **SLRadioButtonHandler** implements ItemListener – security level radio button action handler
 - Method: **itemStateChanged(ItemEvent e)**
 - Process:
 - Stores the selected value in the global security level variable.
- Class: **NMRadioButtonHandler** implements ItemListener – network mode radio button action handler

- Method: **itemStateChanged(ItemEvent e)**
 - Process:
 - Stores the selected value in the global network mode variable.
- Class: **SubmitButtonHandler** implements ActionListener – submit button action handler.
 - Method: **actionPerformed(ActionEvent e)**
 - Process:
 - Verifies that a network mode and security level have been chosen.
 - If not, displays an error message and exit handler.
 - Otherwise continues.
 - Saves the selected values to the dynamic parameter file by calling **set_dynamic_parameters()**.
 - Verifies if IPsec mechanism is already running by calling **demo_support_functions.daemon_running()**.
 - If so, performs the following:
 - Copies the SAD values to a temp file for processing (used for connection calculations) by calling **demo_support_functions.copy_kernel_ipsec()**.
 - Tears down existing connections to reconfigure the IPsec mechanism with the new network mode and security level by calling **demo_support_functions.tear_down**

n_connection().

- Flushes existing security association values from the IPsec mechanism by calling **demo_support_functions.flush_ipsec()**.

- Loads the SPD for the reconfiguration phase by calling **demo_support_functions.load_spd()**;

- Otherwise, displays a message informing the user that the IPsec mechanism must be started first before the dynamic parameter can take effect.

- Class: **ExitButtonHandler** implements ActionListener – **Exit** button action handler.

- Method: **actionPerformed(ActionEvent e)**

- Process:

- Terminate the JFrame by calling **dispose()**.

3.4 ipsecinfo.java

The ipsecinfo class provides a display mechanism for the Security Association Database (SAD). The display needs to be updated constantly to reflect changes caused by the dynamic parameterization of the IPsec mechanism (i.e. shift in network mode and/or security level).

3.4.1. Design Approach

The ipsecinfo class will provide the user with a constantly updated display of existing security associations in the SAD. The mechanism once started must continue until terminated. To ensure that the user can easily read the output, the display should adjust in size in accordance with the number of SA's with the Security Association Database (SAD).

3.4.2. Processing Description

The following is the processing description used to design and develop the **ipsecinfo** class:

- Once initiated, the process runs independently (thread) until terminated.
- Reads in data from **/kern/ipsec** and parses it accordingly.
- To reduce wasted CPU processing time, the file is only read and parsed if it has been updated since the last parsing operation, and is in its complete form. To accomplish this, the class can utilize file date-time-stamps and comparison algorithms with previously displayed SA's. During the development phase, it was discovered that the IPsec mechanism writes to **/kern/ipsec** incrementally. Therefore it is important to make sure the whole file is present prior to parsing. An algorithm is required that will check the file for all mandatory fields prior to parsing.
- Also, a temporary file needs to be copied to avoid permission issues when the IPsec mechanism wants to write to the file.
- When the file is updated, the process of reading the file and parsing begins.
- To parse the file, the following tags are used:
 - SPI
 - Destination
 - Source
 - xform
 - encryption
 - authentication
- As the parsing occurs, text is added to the display

dynamically.

- At the completion of the parsing, the number of SA's is used to determine the size of the JFrame.
- The JFrame will terminate once the **exit** button is pressed.

3.4.3. Pseudo Code

The following is a description of the pseudo code for **ipseinfo.java**:

- Class: **ipseinfo** extends Thread
 - Most variables used throughout this class are Global.
 - Method: **ipseinfo()** – class constructor.
 - Process:
 - Initializes a JFrame.
 - Initializes all display mechanisms.
 - Gets the current Runtime instance by calling **getRuntime()**.
 - Instantiates an instance of **demo_support_functions**
 - Adds a window adapter to handle window exiting by calling **dispose()**.
 - Method: **frame_initialization()** – initializes the JFrame to repaint new SAs.
 - Input: none.
 - Output: none.
 - Process:
 - Removes existing components from the JFrame.
 - Reinitializes all components.
 - Method: **preliminary_test(StringTokenizer st)** – compares the current SA's in the JFrame with the SA's from the new file. This is performed to avoid unnecessary

painting and maintain good display resolution.

- Input:

- StringTokenizer st – contents of the file to be verified.

- Output:

- Boolean – True if string contains new SA's. False otherwise.

- Process:

- The following process is enclosed in a try-catch block.
- While loop through all string tokens.
 - Compares SPI's of existing SA's with new SA's.
 - If a new SA is found, returns true.
 - Otherwise, keep checking all SA's
 - Default return is false.

- Method: **String wait_for_full_copy(String record)**– verifies that all the required tags exist in the string prior to parsing. If not, file is reread and the string is verified until all the tags are found.

- Input:

- String record – contents of the file to be verified.

- Output:

- String – String that contains all required fields.

- Process:

- The following process is enclosed in a try-catch block.
- While loop through all string tokens.
 - Tests to see if the following tokens

exist in the string:

- SPI
- Destination
- Source
- xform
- If all tokens are found, the current string is returned.
- Otherwise, the file is reread and the new string is verified. This process continues until all required fields are found.
- Method: **parse(String record)** – parses the string into SA's to be displayed on the JFrame.
 - Input:
 - String record – contains the file to be parsed.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Verifies that the new string contains new SA's to paint by calling **prelimanary_test()**.
 - If new strings are found, continues parsing.
 - Otherwise, exits the method.
 - Verifies that all fields are present in the string. If not, waits until all fields are present. This is performed by calling **wait_for_full_copy()**.
 - If all fields are present, continues parsing.
 - Uses the StringTokenizer to efficiently breakup the string for parsing.

- While loop through the string tokens.
 - Parses the string based on the following fields:
 - SPI – indicates new SA.
Skip a line in JFrame.
Increment SA counter.
 - Destination – used to retrieve the destination IP address.
 - Source – used to retrieve the source IP address.
 - xform – used to determine the method of protection (AH or ESP)
 - Encryption – used to determine the encryption algorithm.
 - Authentication – used to determine the authentication algorithm.
 - Sizes the JFrame according to the number of SA's (SA counter).
- Method: run()
 - Input: none.
 - Output: none.
 - Process:
 - The following process is enclosed in a try-catch block.
 - Creates a file pointer to **/root/demo/tempipsec**.
 - While loop continuously until thread is

terminated.

- Compares old file date time stamp with new date time stamp.
- If they do not match, reads in the file.
- Creates a file input stream to **/root/demo/tempipsec**.
- Reads in the file into a string.
- Closes the file.
- Initiates the parsing routine by calling **parse()**.

3.5 **tcpdump.java**

A valuable tool in demonstrating the IPsec mechanism is **tcpdump** (described earlier in section B.8). It enables the user to view the actual packets being sent and received across the network. Specifically, it facilitates the demonstration of packets sent in the clear, encrypted and/or authenticated. This tool is typically used via the command prompt. The challenge was to filter the tool's terminal console output through a Java JFrame to maintain a consistent graphical user interface approach. Again the goal is to limit the user's required knowledge of the operating system and environment to utilize the security mechanism.

3.5.1 **Design Approach**

The goal of this class is to provide the user with graphical console display of **tcpdump**. By selecting the **Display TCPDUMP** option from the main menu, the **tcpdump** function starts and displays captured packet information in the generated Java JFrame. When the user closes the window, the **tcpdump** function terminates.

3.5.2. **Processing Description**

The following is the processing description for the **tcpdump.java**:

- Once instantiated, the **tcpdump** class launches, creating a JFrame Java console window.
- The JFrame should contain:
 - Console title.
 - Scrollable view pane (since a large amount of data

generated by tcpdump).

- **Exit** button to close the JFrame and terminate **tcpdump**.

3.5.3. Pseudo Code

The following is the pseudo code for **tcpdump.java**:

- Class: **tcpdump** extends Thread.
 - Variables used throughout the class, are declared as Global.
 - Method: **tcpdump()** – constructor for the class.
 - Process:
 - Instantiates **demo_support_functions** class.
 - Gets the current Runtime instance by calling **getRuntime()**.
 - Initializes JFrame with title.
 - Initializes scrollable text frame.
 - Initializes **exit** button panel.
 - Adds **exit** button action listener.
 - Terminates **tcpdump** by calling **demo_support_functions.stop_tcpdump()**.
 - Closes the JFrame by calling **dispose()**.
 - Adds a window listener that closes JFrame by calling **dispose()**.
 - Method: **start_tcpdump()** - executes **tcpdump** and creates a pipe to capture packet information.
 - Input: none.
 - Output: none.
 - Process:
 - The following code is enclosed in a try catch block.
 - Creates an array of strings to hold the

sequence of commands and tags required to perform a run time execution (see above section B 2.1 in this chapter for more detail) of **tcpdump -N 131.120.8.95 and 131.120.8.9**.

- Note1: that **wait()** is required since it is not necessary for the parent process to wait for its completion. If it did not wait, further processing in the class would be performed until the **tcpdump** process was terminated.

- Note2: With **tcpdump**, all tags and values have to be added to the string array separately in order to execute properly (unlike other commands where switches and tag values can be included in the string array as one string).

- Create an input stream to the **tcpdump** process.

- Method: **repaint_frame()** – this method generates the output from the piped stream and displays to the scrollable text area.

- Input: none.

- Output: none.

- Process:

- The following code is embedded in a try-catch block.

- Initializes the buffer reader to buffer the piped information from **tcpdump**.

- While loop until **JFrame** is closed (while (true)).

- Reads data from the buffered

- reader pipe.
 - Appends data to the scrollable pane.
 - Pauses to allow for printed changes to be viewed on the screen.
- Method: **run()** – run method for the thread.
 - Process:
 - Sets JFrame to visible.
 - Executes **tcpdump** by calling **start_tcpdump()**.
 - Activates the display of **tcpdump** by calling **repaint_frame()**.

3.6. SPD.java

The purpose of this class is to provide a display for the security policy database (SPD). Since the database has a specific format, title tags are used to display the information.

3.6.1. Design Approach

The goal of this class is to retrieve the data stored in the security policy database (SPD) and display on screen in an easy to read format. A static implementation is used.

3.6.2. Processing Description

The following is the processing description used to design and develop **SPD.java**:

- Initializes a JFrame with a view panel to display the SPD information and an **exit** button to terminate the window.
- Executes the command to generate an output of the SPD (**netstat -rn -f encap**).
- Creates a pipe to the process to retrieve the data.
- Displays the data on the screen in an organized easy to read manner.

3.6.3 Pseudo Code

The following is a description of the pseudo code used in

SPD.java:

- Class: **SPD**

- Variables used throughout the class, are declared as Global.

- Method: **SPD()** class constructor.

- Instantiates **demo_support_functions** class.
- Gets the current Runtime instance by calling **getRuntime()**.
- Initializes JFrame with a title.
- Initializes the view panel.
- Initialize the JPanels, JLabels and Text areas to be used in displaying SPD data.
- Initializes the **exit** button panel.
- Adds **exit** button action listener to close the JFrame by calling **dispose()**.
- Adds a window listener to close JFrame by calling **dispose()**.
- Retrieves data and display information in the JFrame by calling **setTextFields()**.

- Method: **String create_SPD_input()** – this method retrieves the SPD data, stores it in a string and returns it to the calling function.

- Input: none.

- Output:

- String – contains the retrieved SPD data.

- Process:

- The following code is enclosed in a try catch block.
 - Creates an array of strings to hold the sequence of commands and tags required to perform a run time execution (see above

section B 2.1 in this chapter for more detail)
of **netstat -rn -f encap**

- Note: that **waitFor()** is required since it is necessary for the parent process to wait for its completion.
 - Creates an input stream to the **netstat** process.
 - Retrieves the SPD data from the input stream by calling **InputStream.read()**.
 - Casts the information as a string and returns to calling method.
- Method: **setTextFields()** – this method parses the input string and displays the SPD data in the JFrame.
- Input: none.
 - Output: none.
 - Process:
 - Casts the string as a String Tokenizer to parse data using blank space as the token delimiter.
 - Skips the header titles.
 - While loop through all string tokens.
 - Parses the expected data appropriately into a display field.
 - Adjusts JFrame size according to number of rows in the SPD.

3.7 SPFK.java

The purpose of the **SPFK.java** class is to provide the user with a more readable display of the KeyNote file. From previous discussions and examples provided in this thesis, it should be clear that KeyNote can become quite complex and difficult to read. **SPFK.java** attempts to provide the user with an easier representation of the defined security policy.

3.7.1. Design Approach

The goal of this class is to translate KeyNote's complex assertion format into an easy to understand syntax and display it in a JFrame. By using the DNF parser (NPS-CS-02-002, January 2002), the class can take advantage of the translated DNF form to perform a parsing routine. The display should be equipped with a scroll bar for long policies (quite typical when dealing with numerous security attributes and dynamic parameters). It should also have an **exit** button to close down the JFrame.

3.7.2. Processing Description

The following is the processing description used to design and develop **SPDK.java**:

- Once instantiated, a Java JFrame console window is created.
- The JFrame contains:
 - Console title.
 - Scrollable view pane (since a large amount of policy data will typically be displayed).
 - **Exit** button to close the JFrame and terminate SPDK.

3.7.3. Pseudo Code

The following is a description of the pseudo code description for **SPDK.java**:

- Class: **SPFK()**
 - Variables used throughout the class, are declared as Global.
- Method: **SPFK()** – this method initializes the JFrame and reads in the security policy data from a file.
 - Input: none.
 - Output: none.
 - Process:
 - Initializes the JFrame with a title.
 - Initializes the scrollable text area.
 - Initializes the **exit** button and add an action

handler that will terminate the JFrame by calling **dispose()**.

- Retrieves the security policy from DNF converted KeyNote file **(/etc/isakmpd/KeynoteDNFFinal.policy)** by calling **demo_support_functions.read_file()** and storing data in a string.

- The following code is embedded within a try-catch block.

- Casts the string into a string tokenizer using '|' as the delimiter.

- While loop through all string tokens.

- Parses through the code retrieving security proposals as tokens.

- Further parses and appends KeyNote security attribute to the scrollable text area by calling **showPolicy()**.

- Method: **showPolicy(String policy)** – takes input string (security proposal), parses it into security attributes and dynamic parameters, and appends to the scrollable text area.

- Input:

- String – security proposal to be parsed.

- Output: none.

- Process:

- Casts input string as string tokenizer.

- While loop through all string tokens.

- Parses through the string, checking for expected tags and retrieving tag

values.

- Displays tags and tag values on the scrollable text area by calling `addText()`.

- Method: **`addText((String, String))`** – takes input strings and appends them to the scrollable text area.

- Input:

- String – tag.

- String – tag value.

- Output: none.

- Process:

- If tag value is not null, appends tag and tag value to the screen.

D. CONCLUSION

This chapter reviewed the process used to design and develop a graphical user interface demonstration to support the research performed in this thesis. Prior to this demo, all commands and processes were managed from the command prompt. This proved to very cumbersome and difficult to understand for most users. By researching the mechanics involved in the “command-line” commands required to operate the IPsec mechanism, I was able to capture the functionality in a GUI. The GUI demo is easy to use and understand, and ultimately provides an efficient method to perform a demonstration on the completed research from this thesis.

The following chapter is a future work discussion, outlining all the potential areas for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. RESEARCH SUMMARY AND FUTURE WORK

A. INTRODUCTION

The research presented in this thesis, provides a proof of concept for parameterizing the IPsec mechanism, specifically OpenBSD 2.8 IPsec. The focus of this work and development was on a limited number of security parameters and a peer-to-peer network configuration. Although the design was generalized to handle all security parameters and network configurations, further implementation will be required to broaden the functionality of IPsec parameterization. Additionally, there are other features that require research, development and implementation to harness OpenBSD's IPsec's capabilities.

In this chapter, the research performed in this thesis will be summarized and the future areas of research in OpenBSD IPsec will be discussed.

B. SUMMARY OF RESEARCH PERFORMED IN THIS THESIS

The goal of this thesis was to provide dynamic parameterization to security mechanisms, specifically OpenBSD 2.8 IPsec. To perform this, I studied the concepts of Quality of Security Service (QoSS), Dynamic Parameters (Network Mode and Security Level), and IPsec architecture. I then designed a dynamic parameterization implementation specific to OpenBSD IPsec version 2.8. To further illustrate this mechanism, I designed and implemented a graphical user interface that enabled the user to view and understand the dynamism of the modified IPsec mechanism. Below is a brief summary of all the above-mentioned areas.

Quality of Security Service (QoSS) provides a mechanism to access security services in accordance with the user and system requirements, as constrained by the network environment. Security services can be defined in terms of user and system requirements, network environment factors and available resources. Without a range of security options, a user is faced with the rigid and limited choice of "all or nothing": security or no security. Historically, security services have been static. Quality of Security Service (QoSS) provides a more flexible approach to security services. Users can define requirements with finer granularity through QoSS. The security resource

manager and/or the Security System can adjust security service accordingly to meet user requirements, system policy and network environment. The utilization of security services comes with a cost to the user, application, system and resources. Whether in the form bandwidth, algorithm processing time, overhead, or funds, the cost of security is a challenging concern to resource managers. A costing framework is required to map system resource impact to enable a management system to effectively handle security service requests.

Security services include non-repudiation, auditing, authentication, encryption, or intrusion detection. Each service will require a governing policy, consisting of specific rules of how and when to use the service. Therefore each network task associated with QoSS can be mapped to a vector of security requirements directly associated with the security services the task requires.

Static security parameters limit a security system's ability to adapt to changes. By introducing a dynamic mechanism, a system can modulate its security settings in accordance with changing conditions. To illustrate the ability to adapt dynamic parameters to a QoSS framework we utilize the Network Mode and Security Level abstractions.

Network Mode enables a network security policy to be classified according to environmental variables. Some example modes might be normal, impacted, and crisis. Normal mode could be defined as ordinary operating conditions with normal traffic load and no heightened threat conditions. Impacted mode may be defined when the network/system is experiencing high levels of traffic and therefore certain security selection may not be available due to efficiency constraints. Crisis mode may be defined as a situation that requires the highest level of security or the lowest level dependent on the situation and policy.

Security Levels are used to provide further granularity to Network Security Policies. By developing a security definition that encompasses general security settings required by users or applications, the complexity of security settings can be simplified to present to users selections such as High, Medium and Low. High would require the strongest security settings. Medium would require moderate security settings. Low would

require weak to no security settings. A mapping of security settings to security levels providing a range of selection or specific values will be required to properly enforce the system security policy.

As a proof of concept to demonstrate how a specific security mechanism can be modulated to provide different levels of security in accordance with QoSS, the OpenBSD IPsec security mechanism was used. OpenBSD IPsec utilizes the KeyNote Trust Management component, which enables a security policy to be mapped to appropriate security attributes. When an initiator, peer A, wants to establish secure communication with a responder, peer B, the peer A's IPsec mechanism proposes a set of security proposals to peer B. Peer B's IPsec mechanism verifies the proposal by performing a query against their security policy in KeyNote. If a suitable security proposal is found, the communication handshake is completed, and the peers establish secure communication channels defined by security associations (SA).

This process is static because when the network mode or security level changes and an adjustment in security policy is required, then the IPsec mechanism must be halted and manual adjustments are required. This is not practical or efficient. An automated technique is required to modulate IPsec's variant security attributes according to network mode and security level selections, enabling the security mechanism to dynamically adjust security parameters and settings in accordance with policy.

The current process also maintains security policy definitions in numerous areas, resulting in management challenges. During (re)configuration phase in IPsec, two separate files are read to load the security proposal set. Thus, another goal of this thesis was to streamline the security policy loading process by maintaining the security policy in only one area/file (KeyNote).

1. Research Conclusions

To accomplish the goals stated above, the following modifications to the existing IPsec implementation were required. The first modification was to provide granularity to KeyNote, and parameterize and improve **isakmpd.conf** / **isakmpd.policy** (KeyNote) security proposal range loading process. Providing granularity to KeyNote required an in-depth review of the KeyNote structure including the Boolean query mechanism. The

research proved to be successful in logically inserting the dynamic parameter values according to policy definition, therefore, providing finer granularity to Boolean query mechanism. Other modifications included the changes to the configuration process to enable security proposal ranges to be retrieved from KeyNote. All modifications were successful in providing the IPsec mechanism with the ability to adjust security attributes according to dynamic parameters and security policy.

OpenBSD IPsec currently requires a complex sequence of commands via command-line commands to establish secure connection between peers. The process of establishing an IPsec connection with a peer requires considerable knowledge of the operating environment. The command-line procedure also provides challenges in demonstrating the effectiveness of the parameterization of IPsec mechanism. As a result, a user-friendly graphical oriented interface was design and developed to allow users, with limited knowledge of OpenBSD operating environment, to observe the security mechanism and its dynamic parameterization.

C. FUTURE DESIGN AND IMPLEMENTATION ON PARAMETERIZATION

To further develop the functionality of the parameterization of the OpenBSD IPsec mechanism, additional design and implementation is required. Listed below are the major items that will require future attention.

1. Ability to Handle all Possible Security Parameter Combinations

The research performed was limited in the range of possible security parameters. The goal instead was to achieve a proof of concept to pave the path for a fully functional parameterization of IPsec. To achieve full functionality, research will be required to ensure all possible combinations of security attributes are taken into consideration in the actual implementation. Examples of further implementation involve incorporating all possible encryption and authentication algorithms, algorithm key length, and time-of-day parameters. Additional development will be required to incorporate inequality definitions (<, >, !=) in the security policy management mechanism. For example, esp_enc_alg > DES which would imply 3DES and AES.

2. Improving Dynamic Parameter Loading by Utilizing “Policy-Callback” Embedded Functionality

The original implementation of OpenBSD IPsec utilizes a “policy_callback”

routine to load security proposals into a structure for KeyNote query operation. The procedure is very efficient because all parameters are loaded into the KeyNote query structure at once. During the implementation portion of this thesis, all attempts to utilize the existing “policy-callback” mechanism to load dynamic parameters were unsuccessful. The work-around was to load one dynamic parameter at a time.

3. Eliminate the Need for `isakmpd.conf` Entirely

This thesis involved removing `isakmpd.conf` security parameters specific to the ISAKMP Phase II quick mode. However, main/aggressive mode information and additional peer information (IP addresses, Net Mask, Gateway Address) still remain resident in `isakmpd.conf`. The challenge is that the current configuration of KeyNote does not support non-security specific parameters. Further research and development is required to evaluate the elimination of `isakmd.conf`.

4. Develop a Parsing Mechanism to Retrieve the Initial Security Policy Database Entries

A parsing routine similar to one developed in this research is required to load the appropriate security rules into the Security Policy Database (SPD). The challenge will be to provide an intelligent parsing routine that can read a security policy file (KeyNote/`isakmpd.policy`) and generate rules for the SPD. Currently, security policy relevant rules are loaded independent of the parameterization mechanism by a script that loads predefined policy rules into the SPD (see Chapter VI).

D. HARNESSING OPENBSD’S IPSEC MECHANISM CAPABILITIES

OpenBSD’s IPsec mechanism provides a wide range of flexible options involving network security. This research was limited due to the objectives and the required focus areas. Additional research is required to explore the other possible configurations and application of the OpenBSD IPsec.

1. Behavior with all Possible Combinations of QoS and non-QoS Peers

The research performed in this thesis was limited to a model involving a Quality Of Security Service (QoS) aware initiator and a non-QoS aware respondent. To account for the mechanism’s full capability and functionality, other combinations need to be researched and tested. Specifically, non-QoS aware initiator and a QoS responder, and both initiator and responder are QoS aware.

2. Per-User / Per-Application Relationship Capability

The testing and implementation performed within this research involved two specific applications: Telnet and Finger from a root access. In reality, an implemented IPsec mechanism would be required to support a broader range of applications and authorization levels of users. Some typical applications might include e-mail, video-conferencing, Internet Remote Chat (IRC), and Web Portals. Further design, development and testing needs to be performed, to support these other applications with the IPsec mechanism.

3. Explore Proposal Caching Issues

Further research and testing is required to determine the security mechanism's QoSS behavior concerning the change of non-QoSS parameters. An example is the expiration of a valid Security Association (SA) without a change to Network Mode/Security Level. Will the SA expiration trigger a reconfiguration? Other examples include testing behavior resulting from the non-QoSS initiators, QoSS aware initiators and responders (both capable of initiated changes to QoSS and non-QoSS parameters), and more frequent negotiations (resulting from shorter lifetimes).

4. Security Policy Editor

The current syntax and the potential complexity of QoSS and security parameter combinations in KeyNote pose a challenge in usability for the human user interface component used by security policy managers and implementers. A security policy editor that translates the KeyNote syntax into an "easy-to-read" format is required. This editor would enable an authorized user, to view and edit a security policy using a graphical user interface.

5. Additional Network Configurations

IPsec is designed to be able to handle numerous network configurations (discussed in the IPsec Architecture Chapter III section B.2.3). The research and development performed and discussed in this thesis is based on the peer-to-peer configuration. Therefore, further design, implementation, and testing will be required to ensure proper functionality of IPsec mechanism and the parameterization functionality of security parameters. Specifically additional research is required for Gateway-to-Gateway, Gateway to IPsec-enabled hosts, and hosts behind IPsec gateways.

6. IPV6 Addresses

In order to account for IPV6 addresses, further research, development and testing will be required.

7. Distribution of KeyNote Policies

The ability to securely distribute keying information to peers in a Public Key Infrastructure(PKI) environment is a crucial and challenging element of a secure network environment. Likewise, the ability to securely distribute KeyNote policy to participating peers or gateways is crucial to IPsec architecture. Research is required to explore potential mechanisms to enable this functionality.

8. KeyNote Protection

Ensuring that only authorized users have access to and the ability to modify KeyNote, is essential to the security policy mechanism of IPsec. An identification and authentication mechanism is required to ensure only authorized users have access to sensitive IPsec files and applications. A range of access may also be desired. For example, a user may have read-only permission to the security policy file, while the security manager will have read-write permissions.

9. Secure Dissemination & Storage of QoS Parameters' Values

To effectively employ QoS parameters, a method of secure storage and dissemination across a network is required. Vulnerabilities and threats posed by Denial of Service (DoS) attacks, and packet capturing and modification need to be examined.

10. IPsec Costing Issues

As with any security system, methods for calculating system resource costs related to IPsec choices and connections are required. To fully implement the QoS model, system managers need measurement tools to be able to gauge resource costs per security requirement to properly govern the IPsec mechanism.

11. Graphical User Interface

The current implementation of OpenBSD IPsec is command-line driven. To provide for easier human computer interaction , a graphical user interface (GUI) that is embedded into the mechanism would be helpful.

E. CONCLUSION

In this chapter, I summarized the research performed on the parametrization of IPsec in this thesis and discussed future areas on research in OpenBSD IPsec.

APPENDIX A. CONF.C

The following is code added to /usr/src/sbin/isakmpd/conf.c:

```
/*
 *
 *      Structure: suite_struct
 *
 *      This structure is used to stored security proposal
 *      information.
 */

struct suite_struct {
    char * suite_name;
    char * suite_protocol;
    char * suite_transform;
    char * protocol_id;
    char * transform_id;
    char * encapsulation_mode;
    char * group_description;
    char * authentication_algorithm;
    char * life;
    char * life_type;
    char * life_duration;
    char * esp;
    char * ah;
    char * esp_enc_alg;
    char * esp_auth_alg;
    char * ah_auth_alg;
    char * pfs;
    char * key_length;
    int copy_flag;
};

/*
 *
 *      Structure: dynamic_packet
 *
 *      This structure is used to stored dynamic parameter
 *      data, including dynamic parameter name, assignment symbol
 *      and value.
 */

struct dynamic_packet {
    char* title;
    char* symbol;
    char* value;
};
```

```
// The following are function prototypes added to conf.c to implement
// dynamic parameterization.

void conf_kn_parse(int);
int add_para_values(char**, struct suite_struct *, struct suite_struct *,
int);
char * convert_to_uppercase(char*);
void parse_selection_parameter(char*, int*, char*, char*, char*, int*,
int, int * );
void parse_ipsec_parameter(char*, int*, char*, char**, int*);
int DNF_parse(char **, char*, int*, int, struct suite_struct *, struct
dynamic_packet *, int, struct suite_struct*, int);
void parse_ipsec_para_tag(char*, int*, char*, char**, int, int*);
void life_seconds_translation(char**, char**, char**);
void life_kilobytes_translation(char**, char**, char**);
void group_description_translation(char**);
int test_suite_structure(struct suite_struct *, int );
void send_to_conf_set(int, char*, char*, struct suite_struct *, int, struct
suite_struct *);
void load_default_sa();
void test_print_suite_list(struct suite_struct *, int);
void test_conf_get_str();
int duplicate_sa (struct suite_struct *, int, struct suite_struct *);
void test_print_dynamic_packet(struct dynamic_packet *, int);
void test_print_suite_struct(struct suite_struct *);
int dynamic_package_verification(char *, int *, int, struct
dynamic_packet *, int, int);
struct suite_struct * struct_initialization(struct suite_struct *);
struct suite_struct * initialize_default_suite_profile(struct
suite_struct *);
struct dynamic_packet* package_dynamic_parameters(int *);
int verify_parameter(char *, int *, char *, char *, int);
void advance_to_end_DNF(char*, int *, int);
```

```
/*
*
* Function: package_dynamic_parameters() This function reads a file that
* contains the current inputs of dynamic parameters such as
* Network Mode and Security Level. Initializes an array of
* structures that will dynamically grow as required. Loads the
* file input into the array of struct.
*
* - Input:
*   - int * package_counter - used to determine the size of the
*     struct array
*
* - Output:
*   - struct dynamic_packet * - pointer to an array of
*     dynamic_packet structure
*
*/
```

```
struct dynamic_packet * package_dynamic_parameters( int
```

```

*package_counter) {

    int array_size = 10 ; // chosen as average case for struct size

    struct dynamic_packet * package; // pointer to array of struct

    // strings used to read from file
    char * title = (char*)malloc(20);
    char * symbol= (char*)malloc(20);
    char * value = (char*)malloc(20);

    // used for file IO
    FILE * pifp;
    int file_status=0;

    (*package_counter) = 0; // initialize the array size to zero

    // initialize the pointer to an array size of array_size

    package = (struct dynamic_packet *)malloc(sizeof(struct
dynamic_packet)*array_size);

    // open file to read in dynamic parameters
    // check for errors in opening the file

    if ( (pifp = fopen("/usr/src/sbin/isakmpd/dynamic_parameters",
"r"))== NULL) {
        // error opening the file

        LOG_DBG ((LOG_POLICY, 40, "ERROR OPENNING FILE %d",errno));
    }
    else { // file was successfully opened

        // do-while loop to read in data until EOF reached

        do {

            // read in from the file expecting the following syntax:
            //          title symbol value

            file_status = fscanf(pifp, "%s%s%s", title, symbol, value);

            LOG_DBG ((LOG_POLICY, 40, "Just read from file....",""));
            LOG_DBG ((LOG_POLICY, 40, "Title: %s Symbol: %s Value: %s",
title, symbol, value));

            // check for reading errors...

            if (file_status == 0) {

                // reading error occurred

                LOG_DBG ((LOG_POLICY, 40, "error reading from the
file....",""));
            }
        }
    }
}

```

```

// check for EOF...

else if (file_status == EOF) {

    // EOF reached

    LOG_DBG ((LOG_POLICY, 40, "end of file....", ""));
}

// otherwise... read operation successful

else {

    // create new struct space for dynamic parameters and
    // assign values from file to struct

    package[*package_counter].title = strdup(title);
    package[*package_counter].symbol = strdup(symbol);
    package[*package_counter].value = strdup(value);

    // increment package counter array of struct
    (*package_counter)++;

    // dynamic resize array if more memory is required

    if ((*package_counter) == array_size) {

        array_size+=10; // grow array in increments of 10

        package = (struct dynamic_packet
*)realloc(package, sizeof(struct dynamic_packet)*array_size);
    }
}

while ((file_status != 0) && (file_status != EOF));

fclose(pifp); // close file

test_print_dynamic_packet(package, *package_counter);

}

// free memory
free(title);
free(symbol);
free(value);
// free(pifp);

return (package);

}

/*****
*
*

```

```

* test_print_dynamic_packet()-This function displays all information in
* the array of dynamic_packet struct.
*
* - input:
*     - struct dynamic_packet * packet - array of struct
*     - int package_counter - used for the size of the struct
*       array
* - output: void.
*
*****/

void test_print_dynamic_packet(struct dynamic_packet *packet, int
packet_counter){

    int counter;

    for (counter = 0; counter < packet_counter; counter++) {

        LOG_DBG ((LOG_POLICY, 40, "Packet: %i",counter));
        LOG_DBG ((LOG_POLICY, 40, "Title: %s",packet[counter].title));
        LOG_DBG ((LOG_POLICY, 40, "Symbol: %s ",packet[counter].symbol));
        LOG_DBG ((LOG_POLICY, 40, "Value: %s",packet[counter].value));

    }

}

/*****
*
*
* Function: conf_kn_parse()function called to activate the DNF
security proposal
* parsing mechanism.
*
*
* - Input:
*     - int trans - used for the transaction number for
*       sequential processing.
* - Output: void.
*
*****/

void conf_kn_parse(int trans) {

    char *suite_title; // used for default phase 2 title
    char *suite;       // used to hold constructed suites
    char *namekn;      // used to hold name of keynote file
    char *buffer_kn;   // used to hold contents of keynote file
    char *buff_temp;   // used to traverse keynote file

    struct stat stkn;  // used to determine the size of the file
    off_t szkn;
    int fdkn;

```



```

int not_done=1;
int not_found=1;
char * section;
int buff_temp_counter=0;
int buff_temp_end;
int SA_counter = 0;
int suite_list_size = 2;

struct dynamic_packet *package;
int package_size=0;

struct suite_struct * suite_list;
struct suite_struct * default_suite_profile;

// Initialize the array of structures used to hold parsed security
// proposal information.

suite_list = (struct suite_struct *)malloc(sizeof(struct
suite_struct)*suite_list_size);

// Initalizing the default suite structure

default_suite_profile = (struct suite_struct*) malloc (sizeof(struct
suite_struct));

section = (char*) malloc (100);
suite_title = (char *) malloc(100);
suite = NULL; //(char *) malloc(100);
namekn = (char *) malloc(100);

default_suite_profile =
initialize_default_suite_profile(default_suite_profile);

suite_title = strcpy (suite_title,"Default-Phase-2-Suite");

namekn = strcpy (namekn,"/etc/isakmpd/keynotednffinal.policy");

// call function to retireve dynamic_parameters and load them into
struct

package = package_dynamic_parameters(&package_size);

if ( (fdkn = open(namekn, O_RDONLY, 0))== -1) {

    LOG_DBG ((LOG_POLICY, 40, "ERROR OPENNING FILE %d",errno));

}
else {

    LOG_DBG ((LOG_POLICY, 40, "OPENED KEYNOTE FILE",""));

}

if (stat (namekn, &stkn) == -1)
{
    LOG_DBG ((LOG_POLICY,40, "STAT FAILED %s", namekn));
}

```

```

    }

    // allocate memory for file buffer
    szkn = stkn.st_size;
    buffer_kn = (char *) malloc(szkn);
    buff_temp = (char*) malloc(szkn);

    // read in file into buffer
    if (read(fdkn, buffer_kn, szkn) !=szkn) {

        LOG_DBG ((LOG_POLICY, 40, "ERROR READING KEYNOTE FILE",""));
    }

    close(fdkn);

    buff_temp = buffer_kn;
    buff_temp_end = szkn-1;

    not_found=1;

    // search thru and find the next open parathesis
    while (buff_temp_counter <= buff_temp_end) {

        if (buff_temp[buff_temp_counter] == '(') {

            if (DNF_parse(&suite, buff_temp, &buff_temp_counter, szkn,
&suite_list[SA_counter], package,package_size, suite_list, SA_counter)
== 1) {

                SA_counter++;

                test_print_suite_list(suite_list,SA_counter);
            }

            else {
                not_found = 0;
            }// end else

        } // end of if

        buff_temp_counter++;

        if (buff_temp_counter <= buff_temp_end) {

            if (SA_counter == suite_list_size) {

                suite_list_size+=2; // grow array in increments of 2

                test_print_suite_list(suite_list,SA_counter);

                suite_list = (struct suite_struct

```

```

*)realloc(suite_list,sizeof(struct suite_struct)*suite_list_size);

    }
}

} // end while

// if no matches found...

if (SA_counter == 0) {

    LOG_DBG ((LOG_POLICY, 40,"No matches were found.... No SA's
Loaded from Keynote DNF",""));
    LOG_DBG((LOG_POLICY, 40, "Calling Load Default SA function.."));
    load_default_sa(default_suite_profile);
}

else { // matches found...

    LOG_DBG ((LOG_POLICY, 40,"%i Matches were found.... Matching SA's
Loaded from Keynote DNF",SA_counter));

    test_print_suite_list(suite_list,SA_counter);

    send_to_conf_set(trans,suite_title,suite,suite_list,SA_counter,
default_suite_profile);

    test_conf_get_str();
}

}

/*****
*
*      Function: test_print_suite_list()- displays all security
*      proposals that currently exist in the array of
*      structures. This function can be used for error checking
*      and debugging.
*      - Input:
*          - struct suite_struct *suite_list - pointer to the array
*          of security proposals.
*          - int struct_size - number of current structures in the
*          array.
*      - Output: void.
*
*****/

void test_print_suite_list(struct suite_struct *suite_list, int
struct_size) {

    int count = 0;

```

```

LOG_DBG((LOG_POLICY, 40,"TEST PRINT SUITE LIST",""));
LOG_DBG((LOG_POLICY, 40,"Struct Size:%i ",struct_size));

for (count = 0; count < struct_size; count++) {

    LOG_DBG((LOG_POLICY,40,"STRUCT NODE: %i",count));
    LOG_DBG ((LOG_POLICY, 40, "SUITE
NAME:%s",suite_list[count].suite_name));
    //LOG_DBG ((LOG_POLICY, 40, "SUITE NAME
size:%i",strlen(suite_list[count].suite_name));

    //LOG_DBG((LOG_POLICY,40,"Pointer address of suite_name:
%i",suite_list[count].suite_name));

    LOG_DBG ((LOG_POLICY, 40, "SUITE
PROTOCOL:%s",suite_list[count].suite_protocol));
    LOG_DBG ((LOG_POLICY, 40, "SUITE
TRANSFORM:%s",suite_list[count].suite_transform));
    LOG_DBG ((LOG_POLICY, 40, "SUITE TRANSFORM
ID:%s",suite_list[count].transform_id));
    LOG_DBG ((LOG_POLICY, 40, "SUITE ENCAPSULATION
MODE:%s",suite_list[count].encapsulation_mode));
    LOG_DBG ((LOG_POLICY, 40, "SUITE GROUP
DESCRIPTION:%s",suite_list[count].group_description));
    LOG_DBG ((LOG_POLICY, 40, "SUITE AUTHENTICATION
ALGORITHM:%s",suite_list[count].authentication_algorithm));
    LOG_DBG ((LOG_POLICY, 40, "SUITE LIFE:%s",suite_list[count].life));
    LOG_DBG ((LOG_POLICY, 40, "LIFE
TYPE:%s",suite_list[count].life_type));
    LOG_DBG ((LOG_POLICY, 40, "LIFE
DURATION:%s",suite_list[count].life_duration));
    LOG_DBG ((LOG_POLICY, 40, "COPY
FLAG:%i",suite_list[count].copy_flag));

}
}

/*****
*
*      Function: test_print_suite_struct()- displays security
*      proposal info in a suite structure. This function can be
*      used for error checking and debugging.
*
*      - Input:
*      - struct suite_struct *suite_list - pointer to the
*      security proposal structure.
*
*      - Output: void.
*
*****/

void test_print_suite_struct(struct suite_struct *suite) {

    LOG_DBG((LOG_POLICY, 40,"TEST PRINT SUITE STRUCT",""));
    LOG_DBG ((LOG_POLICY, 40, "SUITE NAME:%s",suite->suite_name));

```

```

    LOG_DBG ((LOG_POLICY, 40, "SUITE PROTOCOL:%s",suite-
>suite_protocol));
    LOG_DBG ((LOG_POLICY, 40, "SUITE TRANSFORM:%s",suite-
>suite_transform));
    LOG_DBG ((LOG_POLICY, 40, "SUITE TRANSFORM ID:%s",suite-
>transform_id));
    LOG_DBG ((LOG_POLICY, 40, "SUITE ENCAPSULATION MODE:%s",suite-
>encapsulation_mode));
    LOG_DBG ((LOG_POLICY, 40, "SUITE GROUP DESCRIPTION:%s",suite-
>group_description));
    LOG_DBG ((LOG_POLICY, 40, "SUITE AUTHENTICATION ALGORITHM:%s",suite-
>authentication_algorithm));
    LOG_DBG ((LOG_POLICY, 40, "SUITE LIFE:%s",suite->life));
    LOG_DBG ((LOG_POLICY, 40, "LIFE TYPE:%s",suite->life_type));
    LOG_DBG ((LOG_POLICY, 40, "LIFE DURATION:%s",suite->life_duration));
    LOG_DBG ((LOG_POLICY, 40, "COPY FLAG:%i",suite->copy_flag));

}

```

```

int test_suite_structure(struct suite_struct * suite_profile,int
struct_size) {
    int result = 1;
    int count;
    for (count = 0; count < struct_size; count++) {

        if (suite_profile[count].copy_flag == 1) {

            LOG_DBG ((LOG_POLICY, 40, "copy flag set...skipping suite
test",""));

        }

        else {

            LOG_DBG ((LOG_POLICY, 40, "copy flag not set...performing suit
test",""));

            if (suite_profile[count].suite_name == NULL) {

                LOG_DBG ((LOG_POLICY, 40, "suite_name is empty on struct #:
%i",count));
                LOG_DBG ((LOG_POLICY, 40, "entry required... suite_struct test
failed..",""));
                return(0);
            }

            if (suite_profile[count].protocol_id == NULL) {
                LOG_DBG ((LOG_POLICY, 40, "protocol_id is empty on struct #:
%i",count));
                LOG_DBG ((LOG_POLICY, 40, "entry required... suite_struct test

```

```

failed..", ""));
    return(0);

}

    if (suite_profile[count].transform_id == NULL) {
        LOG_DBG ((LOG_POLICY, 40, "transform_id is empty on struct #:
%i", count));
        LOG_DBG ((LOG_POLICY, 40, "entry required... suite_struct test
failed..", ""));
        return(0);

    }

}

}
}
return(result);

}

```

```

/*****
*
* Function: int duplicate_sa ()- compares the security proposals
* list with new security proposal for duplicates.
* - Input:
* - struct suite_struct * suite_list - pointer to a security
* proposal list/array.
* - int suite_count - number of security proposal in the
* array.
* - struct suite_struct *suite_profile - pointer to the new
* security proposal.
* - Output:
* - Int - used as Boolean flag. Returns 1 (true) if new
* security proposal is a duplicate. Returns false if new
* security proposal is not a duplicate.
*
*****/

int duplicate_sa (struct suite_struct * suite_list, int
suite_count, struct suite_struct *suite_profile) {
    int count1=0;
    int true = 1;
    int false = 0;

    for (count1 = 0; count1 < suite_count; count1++) {

        if (((suite_list[count1].ah_auth_alg==NULL) && (suite_profile-
>ah_auth_alg==NULL)) || (((suite_list[count1].ah_auth_alg!=NULL) &&
(suite_profile->ah_auth_alg!=NULL)) &&

```

```

(strcmp(suite_list[count1].ah_auth_alg,suite_profile->ah_auth_alg) ==
0))) {

    if (((suite_list[count1].esp_auth_alg==NULL) && (suite_profile-
>esp_auth_alg==NULL)) || (((suite_list[count1].esp_auth_alg!=NULL) &&
(suite_profile->esp_auth_alg!=NULL)) &&
(strcmp(suite_list[count1].esp_auth_alg,suite_profile->esp_auth_alg) ==
0))) {

        if (((suite_list[count1].esp_enc_alg==NULL) && (suite_profile-
>esp_enc_alg==NULL)) || (((suite_list[count1].esp_enc_alg!=NULL) &&
(suite_profile->esp_enc_alg!=NULL)) &&
(strcmp(suite_list[count1].esp_enc_alg,suite_profile->esp_enc_alg) ==
0))) {

            if (((suite_list[count1].encapsulation_mode==NULL) &&
(suite_profile->encapsulation_mode==NULL)) ||
(((suite_list[count1].encapsulation_mode!=NULL) && (suite_profile-
>encapsulation_mode!=NULL)) &&
(strcmp(suite_list[count1].encapsulation_mode,suite_profile-
>encapsulation_mode) == 0))) {

                if (((suite_list[count1].group_description==NULL) &&
(suite_profile->group_description==NULL)) ||
(((suite_list[count1].group_description!=NULL) && (suite_profile-
>group_description!=NULL)) &&
(strcmp(suite_list[count1].group_description,suite_profile-
>group_description) == 0))) {

                    if (((suite_list[count1].life==NULL) && (suite_profile-
>life==NULL)) || (((suite_list[count1].life!=NULL) && (suite_profile-
>life!=NULL)) && (strcmp(suite_list[count1].life,suite_profile->life)
== 0))) {

                        if (((suite_list[count1].life_type==NULL) &&
(suite_profile->life_type==NULL)) ||
(((suite_list[count1].life_type!=NULL) && (suite_profile-
>life_type!=NULL)) &&
(strcmp(suite_list[count1].life_type,suite_profile->life_type) == 0)))
{

                            if (((suite_list[count1].life_duration==NULL) &&
(suite_profile->life_duration==NULL)) ||
(((suite_list[count1].life_duration!=NULL) && (suite_profile-
>life_duration!=NULL)) &&
(strcmp(suite_list[count1].life_duration,suite_profile->life_duration)
== 0))) {

                                if (((suite_list[count1].pfs==NULL) && (suite_profile-
>pfs==NULL)) || (((suite_list[count1].pfs!=NULL) && (suite_profile-
>pfs!=NULL)) && (strcmp(suite_list[count1].pfs,suite_profile->pfs) ==

```

```

0))) {

        if (((suite_list[count1].key_length==NULL) &&
(suite_profile->key_length==NULL)) ||
(((suite_list[count1].key_length!=NULL) && (suite_profile->
key_length!=NULL)) &&
(strcmp(suite_list[count1].key_length,suite_profile->key_length) ==
0))) {

                suite_profile->copy_flag = 1;
                LOG_DBG ((LOG_POLICY, 40, "Match found setting
copy flag", ""));
                return (true);

        }
    }
}

return (false);
}

/*****
*
*   Function: send_to_conf_set() - this functions sends parsed
*   information to conf_set in the correct syntax.
*
*   - Input:
*       - int trans - transaction number.
*       - char * suite_title - holds title for tag defined in
*       previous function.
*       - char * suite - holds set of security proposals
*       - struct suite_struct * suite_profile - points to the
*       list/array of suite structures.
*       - int struct_size - holds the size of the list.
*   - Output: None.
*
*****/

void send_to_conf_set(int trans,char * suite_title, char * suite,
struct suite_struct * suite_profile, int struct_size, struct
suite_struct * default_suite_profile) {

    char * section = (char *) malloc(100);
    char * title = (char *) malloc(100);
    int count =0;

```



```

int tempsize = 0;

char * temp1, *temp2, *temp3 = malloc(100);

strcpy(section,"General");

if (suite == NULL) {
    LOG_DBG ((LOG_POLICY, 40, "loading error: suite is null",""));
    LOG_DBG ((LOG_POLICY, 40, "Aborting suite struct load",""));
    LOG_DBG ((LOG_POLICY, 40, "loading default default suite
structure",""));
    load_default_sa(trans,section,suite_title,default_suite_profile);
    return;
}

else {
    LOG_DBG ((LOG_POLICY, 40, "Suite Set complete....continuing",""));
    LOG_DBG ((LOG_POLICY, 40, "Testing suite structures for min
requirements",""));

    if (test_suite_structure(suite_profile, struct_size) == 1) {
        LOG_DBG ((LOG_POLICY, 40, "Suite Structure has min required
entries... continuing loading process",""));
    }
    else {
        LOG_DBG ((LOG_POLICY, 40, "Suite Structure does not have min
requirements...", ""));
        LOG_DBG ((LOG_POLICY,40, "Aborting suite struct load",""));
        LOG_DBG ((LOG_POLICY,40,"Loading default suite structure",""));
        load_default_sa(trans,section,suite_title,default_suite_profile);
        return;
    }
}

LOG_DBG ((LOG_POLICY, 40, "Loading suite set into conf_set...", ""));
conf_set(trans, section, suite_title, suite,0,0);

for (count = 0; count < struct_size; count++) {
    if (suite_profile[count].copy_flag == 0 ) {
        LOG_DBG ((LOG_POLICY, 40, "Loading suite#: %i into conf_set
...",count));
        LOG_DBG ((LOG_POLICY, 40, "Loading suite_protocol into conf_set
...", ""));
    }
}

```

```

        section = strcpy (section,suite_profile[count].suite_name);
        title = strcpy(title, "Protocols");

conf_set(trans,section,title,suite_profile[count].suite_protocol,0,0);

        LOG_DBG ((LOG_POLICY, 40, "Loading protocol_id into conf_set
...",""));

        section = strcpy (section,suite_profile[count].suite_protocol);
        title = strcpy(title, "PROTOCOL_ID");

        conf_set(trans,section,title,suite_profile[count].protocol_id,0,0);

        LOG_DBG ((LOG_POLICY, 40, "Loading suite_transform into conf_set
...",""));
        title = strcpy(title, "Transforms");

conf_set(trans,section,title,suite_profile[count].suite_transform,0,0);

        LOG_DBG ((LOG_POLICY, 40, "Loading transform_id into conf_set
...",""));
        section = strcpy (section,suite_profile[count].suite_transform);
        title = strcpy(title, "TRANSFORM_ID");

conf_set(trans,section,title,suite_profile[count].transform_id,0,0);

        LOG_DBG ((LOG_POLICY, 40, "Loading encapsulation_mode into conf_set
...",""));
        section = strcpy (section,suite_profile[count].suite_transform);
        title = strcpy(title, "ENCAPSULATION_MODE");

        if (suite_profile[count].encapsulation_mode == NULL) {

            LOG_DBG ((LOG_POLICY, 40, "loading error: encapsulation_mode is
null",""));

            LOG_DBG ((LOG_POLICY, 40, "loading default encapsulation
mode",""));
            LOG_DBG ((LOG_POLICY, 40, "default encapsulation mode:
%s",default_suite_profile->encapsulation_mode));
            conf_set(trans,section,title,default_suite_profile-
>encapsulation_mode,0,0);

        }

        else {

conf_set(trans,section,title,suite_profile[count].encapsulation_mode,0,
0);
        }

        LOG_DBG ((LOG_POLICY, 40, "Loading group_description into conf_set
...",""));
        section = strcpy (section,suite_profile[count].suite_transform);
        title = strcpy(title, "GROUP_DESCRIPTION");

```

```

        if (suite_profile[count].group_description == NULL) {

            LOG_DBG ((LOG_POLICY, 40, "loading error: group_description is
null", ""));

            LOG_DBG ((LOG_POLICY, 40, "loading default
group_description", ""));
            conf_set(trans, section, title, default_suite_profile-
>group_description, 0, 0);

        }

        else {

conf_set(trans, section, title, suite_profile[count].group_description, 0, 0
);
        }

        section = strcpy (section, suite_profile[count].suite_transform);
        title = strcpy(title, "AUTHENTICATION_ALGORITHM");

        if ((suite_profile[count].authentication_algorithm ==
NULL)&&(strcmp(suite_profile[count].protocol_id, "IPSEC_AH") == 0)) {

            LOG_DBG ((LOG_POLICY, 40, "loading error:
authentication_algorithm is null", ""));
            LOG_DBG ((LOG_POLICY, 40, "AH SA... using transform_id to load
authentication algorithm", ""));

            //LOG_DBG ((LOG_POLICY, 40, "loading default
authentication_algorithm", ""));

conf_set(trans, section, title, strcat("HMAC_", default_suite_profile-
>authentication_algorithm), 0, 0);

        }

        else if ((suite_profile[count].authentication_algorithm ==
NULL)&&(strcmp(suite_profile[count].protocol_id, "IPSEC_ESP") == 0)) {

            LOG_DBG ((LOG_POLICY, 40, "no authentication algorithm found for
ESP suite...", ""));
            LOG_DBG ((LOG_POLICY, 40, "Assuming no ESP authenctication
algorithm needed... no default loading..", ""));

        }

        else {
            LOG_DBG ((LOG_POLICY, 40, "Loading authentication
algorithm...", ""));

conf_set(trans, section, title, suite_profile[count].authentication_algori
thm, 0, 0);
        }

```

```

        section = strcpy (section,suite_profile[count].suite_transform);
        title = strcpy(title, "Life");

        if ((suite_profile[count].life ==
NULL)|| (suite_profile[count].life_type ==
NULL)|| (suite_profile[count].life_duration == NULL)) {

            LOG_DBG ((LOG_POLICY, 40, "loading error: 1 or more life time
parameters are null",""));

            LOG_DBG ((LOG_POLICY, 40, "loading default life
parameters",""));
            conf_set(trans,section,title,default_suite_profile->life,0,0);
            section = strcpy(section,default_suite_profile->life);
            title =strcpy(title, "LIFE_TYPE");
            conf_set(trans,section,title,default_suite_profile-
>life_type,0,0);
            title = strcpy(title,"LIFE_DURATION");
            conf_set(trans,section,title,default_suite_profile-
>life_duration,0,0);
        }

        else {
            LOG_DBG ((LOG_POLICY, 40, "loading life parameters...", ""));
            LOG_DBG ((LOG_POLICY, 40, "Life title...", ""));
            conf_set(trans,section,title,suite_profile[count].life,0,0);
            section = strcpy(section,suite_profile[count].life);
            title =strcpy(title, "LIFE_TYPE");
            LOG_DBG ((LOG_POLICY, 40, "Life_Type...", ""));
            conf_set(trans,section,title,suite_profile[count].life_type,0,0);
            title = strcpy(title,"LIFE_DURATION");
            LOG_DBG ((LOG_POLICY, 40, "Life_Duration...", ""));

            conf_set(trans,section,title,suite_profile[count].life_duration,0,0);

        }
        }
        else {
            LOG_DBG ((LOG_POLICY, 40, "Skipping a duplicate sa in
send_to_conf_set",""));
        }
    }
}

// Load conf_set with default value

/*****
*
* - Function: void load_default_sa()- loads default security proposal
*           into conf_set().
*
* - Input:
*   - int trans - transaction number required for conf_set()
*   - char * section - character string defined in calling
*****/

```

```

*          function

*          - char * title - character string defined in calling
*          function
*          - struct suite_struct* default_suite_profile - default
*          suite structure for the default security proposal
*          parameters.
*          - Output: void.
*
*****/

void load_default_sa(int trans,char * section, char * title,struct
suite_struct* default_suite_profile) {

    LOG_DBG ((LOG_POLICY, 40, "Loading default sa's...",""));

    conf_set(trans, section, title, default_suite_profile->
suite_name,0,0);

    section = strcpy (section,default_suite_profile->suite_name);
    title = strcpy(title, "Protocols");
    conf_set(trans,default_suite_profile->
suite_name,title,default_suite_profile->suite_protocol,0,0);

    section = strcpy (section,default_suite_profile->suite_protocol);
    title = strcpy(title, "PROTOCOL_ID");
    conf_set(trans,section,title,default_suite_profile->protocol_id,0,0);

    title = strcpy(title, "Transforms");
    conf_set(trans,section,title,default_suite_profile->
suite_transform,0,0);

    section = strcpy (section,default_suite_profile->suite_transform);
    title = strcpy(title, "TRANSFORM_ID");
    conf_set(trans,section,title,default_suite_profile->
transform_id,0,0);
    section = strcpy (section,default_suite_profile->suite_transform);
    title = strcpy(title, "ENCAPSULATION_MODE");
    conf_set(trans,section,title,default_suite_profile->
encapsulation_mode,0,0);
    section = strcpy (section,default_suite_profile->suite_transform);
    title = strcpy(title, "GROUP_DESCRIPTION");
    conf_set(trans,section,title,default_suite_profile->
group_description,0,0);

    section = strcpy (section,default_suite_profile->suite_transform);
    title = strcpy(title, "AUTHENTICATION_ALGORITHM");
    conf_set(trans,section,title,default_suite_profile->
authentication_algorithm,0,0);

    section = strcpy (section,default_suite_profile->suite_transform);
    title = strcpy(title, "Life");
    conf_set(trans,section,title,default_suite_profile->life,0,0);

    section = strcpy(section,default_suite_profile->life);

```

```

        title =strcpy(title, "LIFE_TYPE");
        conf_set(trans,section,title,default_suite_profile->life_type,0,0);

        title = strcpy(title,"LIFE_DURATION");
        conf_set(trans,section,title,default_suite_profile-
>life_duration,0,0);

    }

/*****
*
*   Function:   struct suite_struct* struct_initialization()- used to
*               initialize each suite structure.
*
*   - Input:   - struct suite_struct * suite_profile - holds the
*               pointer to suite structure.
*   - Output:  - returns the newly initialized structure.
*
*****/

struct suite_struct * struct_initialization(struct suite_struct *
suite_profile) {

    // initialize struct
    suite_profile->suite_name = NULL;
    suite_profile->suite_protocol= NULL;
    suite_profile->suite_transform= NULL;
    suite_profile->transform_id= NULL;
    suite_profile->protocol_id= NULL;
    suite_profile->encapsulation_mode= NULL;
    suite_profile->group_description= NULL;
    suite_profile->authentication_algorithm= NULL;
    suite_profile->life= NULL;
    suite_profile->life_type= NULL;
    suite_profile->life_duration= NULL;
    suite_profile->esp = NULL;
    suite_profile->ah = NULL;
    suite_profile->esp_enc_alg = NULL;
    suite_profile->esp_auth_alg = NULL;
    suite_profile->ah_auth_alg = NULL;
    suite_profile->pfs = NULL;
    suite_profile->key_length = NULL;
    suite_profile->copy_flag = 0;

    return (suite_profile);

}

/*****
*
*   Struct suite_struct * initialize_default_suite()
*
*   Initializes default suite in the event of failure to properly
*   load a security proposal.
*
*****/

```

```

struct suite_struct * initialize_default_suite_profile(struct
suite_struct *temp_ss) {

    // struct suite_struct temp_ss;
    // initialize struct

    LOG_DBG ((LOG_POLICY, 40, "in initialize_default_suite_profile", ""));
    temp_ss->suite_name = strdup("QM-ESP-AES-SHA-PFS-SUITE");
    LOG_DBG ((LOG_POLICY, 40, "test1", ""));
    temp_ss->suite_protocol= strdup("QM-ESP-AES-SHA-PFS");
    temp_ss->suite_transform= strdup("QM-ESP-AES-SHA-PFS-XF");
    temp_ss->transform_id= strdup("AES");
    LOG_DBG ((LOG_POLICY, 40, "test2", ""));
    temp_ss->protocol_id= strdup("IPSEC_ESP");
    temp_ss->encapsulation_mode= strdup("TUNNEL");
    temp_ss->group_description= strdup("MODP_1024");
    temp_ss->authentication_algorithm= strdup("HMAC_SHA");
    LOG_DBG ((LOG_POLICY, 40, "test3", ""));
    temp_ss->life= strdup("LIFE_3600_SECS");
    temp_ss->life_type= strdup("SECONDS");
    temp_ss->life_duration= strdup("3600,1800:7200");
    temp_ss->copy_flag=0;
    return (temp_ss);

}

/*****
*
*   Function: int dynamic_package_verification() - used to check
*   dynamic parameters of DNF security proposal assertions.
*   - Input:
*       - char * buff_temp - character string/buffer used to hold
*       the isakmpd.conf/KeyNote file being parsed.
*       - int *buff_temp_counter - index used for parsing the
*       buff_temp.
*       - int buff_temp_end - index to last character of buffer
*       used to check for end-of-file (EOF) condition.
*       - struct dynamic_packet * package - structure that holds
*       current value of the dynamic parameters.
*       - int package_size - size of array of dynamic_packet
*       structure.
*       - int szkn - size of KeyNote file.
*   - Output:
*       - int - used as a Boolean flag to indicate if DNF security
*       proposal assertion dynamic parameters match. Return 0
*       (false). Return 1 (true).
*
*****/

int dynamic_package_verification(char * buff_temp, int
*buff_temp_counter, int buff_temp_end, struct dynamic_packet * package,
int package_size, int szkn) {

    int temp_buff = *buff_temp_counter;

```

```

int loop_counter;

// used to verify that all packages parameters exist in expression
int package_counter_check = 0;

int not_done = 1;

int result = 0;

int dynamic_test_counter = 0; // used to count matches

char * testprint = (char*)malloc(buff_temp_end + 50);

//loop through dynamic package array to check for matching parameters

if ((buff_temp_end - (temp_buff)) > 0) {

    testprint =
strncpy(testprint,&buff_temp[*buff_temp_counter],(buff_temp_end -
temp_buff));

}
else LOG_DBG ((LOG_POLICY, 40, "Here is our problem",""));

for (loop_counter = 0; loop_counter < package_size; loop_counter++)
{
    // reseting temp pointer for scan

    temp_buff = *buff_temp_counter;

    not_done = 1; // reset for while loop

    while (not_done==1) {

        // check to see if first letter matching

        if ((buff_temp[temp_buff] == package[loop_counter].title[0])
&&((temp_buff + strlen(package[loop_counter].title)) < szkn)) {

            // if result is false not a match return false...

            result =
verify_parameter(buff_temp,&temp_buff,package[loop_counter].title,
package[loop_counter].value,buff_temp_end);

            // if dynamic parameter title found but value does not match
            //advance to next DNF expression and exit

            if (result == 0) {

advance_to_end_DNF(buff_temp,buff_temp_counter,buff_temp_end);

                return (0); // return false
            }

```



```

        // if dynamic parameter title found and value matches...MATCH!
        if (result == 1) {
            package_counter_check++;
            not_done = 0; // set flag to exit while loop
        }

        // title match not found keeping looking...
        if (result == 2) {
        }
    }

    if (buff_temp[temp_buff] == '|') {
        not_done = 0;
    }
    else {
        // Check to see if at end of file
        if (*buff_temp_counter >= szkn) {
            not_done = 0;
        }
        else {
            temp_buff++;
        }
    }
}

} // end of for loop

// if counter match all conditions met return true

if (package_counter_check == package_size) {
    LOG_DBG ((LOG_POLICY, 40, "All conditions met.... returning
true", ""));

    return (1); // return true;
}

else { // not all conditions met....

```

```

        LOG_DBG ((LOG_POLICY, 40, "Not all conditions met....advancing
book mark and return false...", ""));

        //advance pointer to next expression

        advance_to_end_DNF(buff_temp,&temp_buff,buff_temp_end);

        // set actual buffer place holder to advanced marker
        (*buff_temp_counter) = temp_buff;

        // return false;
        return (0);
    }
}

/*****
*
*
* Function: int DNF_parse()- This function parses each security
* proposal found.
*     - Input:
*         - Note: That in order to facilitate dynamic memory
*           use, pointer to pointer coding syntax at times
*           was required. By having a pointer to a pointer,
*           memory created in a function will still be
*           resident/within scope after returning from the
*           function.
*         - char **suite - holds the set of security proposals.
*           Pointer to a pointer used to for dynamic memory
*           creation.
*         - char *buff_temp - string buffer holding the DNF
*           file.
*         - int* buff_temp_counter - location of parsing index.
*           Pointer to integer is used to allow for pass by
*           reference.
*         - int szkn - size of file/string buffer.
*         - struct suite_struct *suite_profile - pointer to
*           suite structure.
*     - Outputs:
*         - integer
*           - returns 1 (false) if parse routine
*             successful.
*           - returns 0 (false) otherwise.
*
*****/

int DNF_parse(char **suite, char *buff_temp, int * buff_temp_counter,
int szkn, struct suite_struct *suite_profile, struct dynamic_packet *
package, int package_size, struct suite_struct *suite_list, int
SA_counter) {

    int success =0;
    char *temp_name, *temp_value;
    int not_done = 1;
    //int para_counter=0;
    int buff_temp_end = szkn -1;

```

```

int temp_size=0;
int completed = 1;// if dynamic parameters are met
temp_name = (char*)malloc(100);
temp_value =(char*)malloc(100);

suite_profile = struct_initialization(suite_profile);

LOG_DBG ((LOG_POLICY, 40, "ENTERING DNF PARSE",""));

    // check expression dynamic parameter matching

    // if false... return


    if (dynamic_package_verification(buff_temp, buff_temp_counter,
buff_temp_end, package, package_size,szkn)==0) {

        return (0) ;

    }

    else {

        LOG_DBG ((LOG_POLICY, 40, "Dynamic package conditions were
met!...preceding to perform parsing ...",""));

    }


while (not_done == 1) {

    // Check for ESP

    if ((buff_temp[*buff_temp_counter] == 'e')&&((*buff_temp_counter +
strlen("esp_present")) < szkn)) {

        temp_name = "esp_present";

        temp_size = 3;
        success=0;

parse_ipsec_para_tag(buff_temp,buff_temp_counter,temp_name,&suite_profi
le->esp,temp_size, &success);

        // if successful load esp protocol into structure

        if (success == 1) {

            suite_profile->protocol_id = strdup("IPSEC_ESP");

        }

    }
}

```

```

// Check for ESP ENC ALG

if ((buff_temp[*buff_temp_counter] == 'e') && (*buff_temp_counter +
strlen("esp_enc_alg")) < szkn)) {

    temp_name = "esp_enc_alg";
    success=0;

    LOG_DBG ((LOG_POLICY, 40, "CHECKING FOR ESP_ENC_ALG", ""));

    parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &(suite_pro
file->esp_enc_alg), &success);

    // if successful copy appropriate esp enc alg to structure
    if (success == 1) {

        suite_profile->transform_id = strdup(suite_profile->esp_enc_alg);

    }

}

// Check for ESP AUTH ALG

if ((buff_temp[*buff_temp_counter] == 'e') && (*buff_temp_counter +
strlen("esp_auth_alg") < szkn)) {

    temp_name = "esp_auth_alg";
    success=0;

    parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &suite_prof
ile->authentication_algorithm, &success);

    // if successful copy appropriate esp auth alg to structure
    if (success == 1) {
        char * temp_holder = (char*)malloc(20);
        char * temp_holder2 = NULL; //(char*) malloc(20);

        // check for HMAC- header on transform_id
        if (suite_profile->authentication_algorithm[0] == 'H') {

            temp_holder2 = strchr(suite_profile-
>authentication_algorithm, '-');
            //suite_profile->authentication_algorithm[4] == '_';
            (*temp_holder2) = '_';
            temp_holder2++;
            suite_profile->transform_id = strdup(temp_holder2);
            suite_profile->esp_auth_alg = strdup (temp_holder2);

        }
        else {

            temp_holder = strcpy(temp_holder, "HMAC_");
            suite_profile->esp_auth_alg = strdup(suite_profile-
>authentication_algorithm);
            suite_profile->transform_id = strdup(suite_profile-

```

```

>authentication_algorithm);
    temp_holder = strcat(temp_holder,suite_profile-
>authentication_algorithm);
    free(suite_profile->authentication_algorithm);

    suite_profile->authentication_algorithm = strdup(temp_holder);

}

free(temp_holder);
free(temp_holder2);
}

}

// Check for AH

    if ((buff_temp[*buff_temp_counter] == 'a')&&((*buff_temp_counter +
strlen("ah_present")) < szkn)) {

        temp_name = "ah_present";
        success=0;
        temp_size = 2;

parse_ipsec_para_tag(buff_temp,buff_temp_counter,temp_name,&suite_profi
le->ah, temp_size,&success);

        // if successful load ah protocol into structure

        if (success == 1) {

            suite_profile->protocol_id = strdup("IPSEC_AH");
        }

    }

// Check for PFS

    if ((buff_temp[*buff_temp_counter] == 'p')&&((*buff_temp_counter +
strlen("pfs")) < szkn)) {

        temp_name = "pfs";
        success=0;
        temp_size = 3;

parse_ipsec_para_tag(buff_temp,buff_temp_counter,temp_name,&suite_profi
le->pfs, temp_size,&success);

        if (success == 1) {
            LOG_DBG ((LOG_POLICY, 40, "pfs= %s",suite_profile->pfs));
        }
    }

```

```

    }

    // check for ah_auth_alg

    if ((buff_temp[*buff_temp_counter] == 'a') && ((*buff_temp_counter +
strlen("ah_auth_alg")) < szkn)) {

        temp_name = "ah_auth_alg";
        success=0;

        parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name,
            &suite_profile->authentication_algorithm, &success);

        // if successful copy appropriate ah auth alg to structure
        if (success == 1) {
            char * temp_holder = (char*)malloc(20);
            char * temp_holder2 = NULL; //(char*) malloc(20);

            // check for HMAC- header on transform_id
            if (suite_profile->authentication_algorithm[0] == 'H') {
                //suite_profile->authentication_algorithm[4] = '_';
                //temp_holder2 = strchr(suite_profile-
>authentication_algorithm, '_');
                //temp_holder2++;
                temp_holder2 = strchr(suite_profile-
>authentication_algorithm, '-');
                (*temp_holder2) = '_';
                temp_holder2++;
                suite_profile->transform_id = strdup(temp_holder2);

                suite_profile->ah_auth_alg = strdup (temp_holder2);

                LOG_DBG ((LOG_POLICY, 40, "Result of HMAC check/fix trans: %s
alg: %s", suite_profile->transform_id, suite_profile-
>authentication_algorithm));
            }
            else {

                temp_holder = strcpy(temp_holder, "HMAC_");
                suite_profile->ah_auth_alg = strdup(suite_profile-
>authentication_algorithm);
                suite_profile->transform_id = strdup(suite_profile-
>authentication_algorithm);
                temp_holder = strcat(temp_holder, suite_profile-
>authentication_algorithm);
                free(suite_profile->authentication_algorithm);

                suite_profile->authentication_algorithm = strdup(temp_holder);
            }

            free(temp_holder);
        }
    }
}

```

```

// check for esp_group_desc

if ((buff_temp[*buff_temp_counter] == 'e')&&((*buff_temp_counter +
strlen("esp_group_desc")) < szkn)) {

    char * temp_holder = NULL; //(char *) malloc(20);
    temp_name = "esp_group_desc";
    success=0;
    parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &temp_holder, &success);

    // if successful load esp_group_desc into structure

    if (success == 1) {

        suite_profile->group_description = strdup(temp_holder);
        group_description_translation(&suite_profile->group_description);
        free(temp_holder);
    }

}

// check for ah_group_desc

if ((buff_temp[*buff_temp_counter] == 'a')&&((*buff_temp_counter +
strlen("ah_group_desc")) < szkn)) {

    char * temp_holder=NULL; //(char*)malloc(100);
    temp_name = "ah_group_desc";
    success=0;

    parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &temp_holder, &success);

    // if successful load ah_group_desc into structure

    if (success == 1) {

        suite_profile->group_description = strdup(temp_holder);
        group_description_translation(&suite_profile->group_description);
        free(temp_holder);
    }

}

// check for esp_encapsulation

if ((buff_temp[*buff_temp_counter] == 'e')&&((*buff_temp_counter +
strlen("esp_encapsulation")) < szkn)) {

    temp_name = "esp_encapsulation";
    success=0;

```

```

parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &suite_profile->encapsulation_mode, &success);

    }

// check for ah_encapsulation

    if ((buff_temp[*buff_temp_counter] == 'a') && ((*buff_temp_counter +
strlen("ah_encapsulation")) < szkn)) {

        temp_name = "ah_encapsulation";
        success=0;

parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &suite_profile->encapsulation_mode, &success);

    }

// check for esp_life_seconds

    if ((buff_temp[*buff_temp_counter] == 'e') && ((*buff_temp_counter +
strlen("esp_encapsulation")) < szkn)) {

        temp_name = "esp_life_seconds";
        success=0;

parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &suite_profile->life, &success);

        // if success then do esp life in seconds translation

        if (success == 1) {

            life_seconds_translation(&suite_profile->life, &suite_profile->life_type, &suite_profile->life_duration);

        }

    }

// check for ah_life_seconds

    if ((buff_temp[*buff_temp_counter] == 'a') && ((*buff_temp_counter +
strlen("ah_life_seconds")) < szkn)) {

        temp_name = "ah_life_seconds";
        success=0;

parse_ipsec_parameter(buff_temp, buff_temp_counter, temp_name, &suite_profile->life, &success);

        // if success then do ah life in seconds translation

        if (success == 1) {

```



```

        life_seconds_translation(&suite_profile->life,&suite_profile-
>life_type,&suite_profile->life_duration);

    }

}

// check for esp life time in kilobytes

    if ((buff_temp[*buff_temp_counter] == 'e')&&((*buff_temp_counter +
strlen("esp_life_kilobytes")) < szkn)) {

        temp_name = "esp_life_kilobytes";
        success=0;

    parse_ipsec_parameter(buff_temp,buff_temp_counter,temp_name,&suite_prof
ile->life,&success);

        // if success then do esp life in kilobytes transalation

        if (success == 1) {

            life_kilobytes_translation(&suite_profile->life,&suite_profile-
>life_type,&suite_profile->life_duration);

        }

    }

// check for ah life time in kilobytes

    if ((buff_temp[*buff_temp_counter] == 'a')&&((*buff_temp_counter +
strlen("ah_life_kilobytes")) < szkn)) {

        temp_name = "ah_life_kilobytes";
        success=0;

    parse_ipsec_parameter(buff_temp,buff_temp_counter,temp_name,&suite_prof
ile->life,&success);

        // if success then do ah life in kilobytes transalation

        if (success == 1) {

            life_kilobytes_translation(&suite_profile->life,&suite_profile-
>life_type,&suite_profile->life_duration);

        }

    }

// check for end of DNF expression

    if (buff_temp[*buff_temp_counter] == '|') {

        LOG_DBG ((LOG_POLICY, 40, "END OF EXPRESSION",""));

```

```

        not_done = 0;

        if (add_para_values(suite, suite_profile, suite_list, SA_counter)
== 1) {

            completed =1;

        }
        else {
            completed = 0;
        }
    }

    else {

        // Check to see if at end of file
        if (*buff_temp_counter >= szkn) {

            not_done = 0;

        }

        else {

            (*buff_temp_counter)++;

        }

    }

}

return (completed);
}

```

```

/*****
*
*
*
* Function: void life_kilobytes_translation() - this function is
* used to convert lifetime in kilobytes from the
* KeyNote/isakmpd.policy format to the isakmpd.conf format.
*
*   - Inputs:
*       - Note: That in order to facilitate dynamic memory
*         use, pointer to pointer coding syntax at times was
*         required. By having a pointer to a pointer, memory
*         created in a function will still be resident/within
*         scope after returning from the function.
*       - char ** life - holds the initial life time input.
*         Pointer to a pointer used for dynamic memory
*         allocation.
*       - char ** life_type - holds the life time type string
*         KILobyTES. Pointer to a pointer used for dynamic
*         memory allocation.
*

```

```

*           - char ** life_duration - holds the life time
*           duration string . Pointer to a pointer used for
*           - dynamic memory allocation.
*       - Outputs:
*           - char ** life - used to return life time. Pointer to
*           a pointer used for dynamic memory allocation.
*           - char ** life_type - used to return life time type
*           string KILOBYTES. Pointer to a pointer used for
*           dynamic memory allocation.
*           - char ** life_duration - used to return the life
*           time duration string . Pointer to a pointer used for
*           dynamic memory allocation.
*
*****/

```

```

void life_kilobytes_translation(char ** life, char **life_type, char
**life_duration) {

```

```

    if (strcmp(*life, "1000")== 0) {

        free(*life);
        *life = strdup("LIFE_1000_KB");
        *life_type = strdup("KILOBYTES");
        *life_duration = strdup("1000,768:1536");

    }

    else if (strcmp(*life, "32000")== 0) {

        free(*life);
        *life = strdup("LIFE_32_MB");
        *life_type = strdup("KILOBYTES");
        *life_duration = strdup("32768,16384:65536");

    }

    else if (strcmp(*life, "45000000")== 0) {

        free(*life);
        *life = strdup("LIFE_4.5_GB");
        *life_type = strdup("KILOBYTES");
        *life_duration = strdup("4608000,4096000:8192000");

    }

    else {

        free(*life);
        *life = strdup("LIFE_1000_KB");
        *life_type = strdup("KILOBYTES");
        *life_duration = strdup("1000,768:1536");

    }
}

```

```

/*****
*
*
*
* Function: void life_seconds_translation() - this function is
* used to convert lifetime in kilobytes from the
* KeyNote/isakmpd.policy format to the isakmpd.conf format.
*
*   - Inputs:
*
*     - Note: That in order to facilitate dynamic memory
*       use, pointer to pointer coding syntax at times was
*       required. By having a pointer to a pointer, memory
*       created in a function will still be resident/within
*       scope after returning from the function.
*     - char ** life - holds the initial life time input.
*       Pointer to a pointer used for dynamic memory
*       allocation.
*     - char ** life_type - holds the life time type string
*       SECONDS. Pointer to a pointer used for dynamic
*       memory allocation.
*     - char ** life_duration - holds the life time
*       duration string . Pointer to a pointer used for
*       dynamic memory allocation.
*
*   - Outputs:
*     - char ** life - used to return life time. Pointer to
*       a pointer used for dynamic memory allocation.
*     - char ** life_type - used to return life time type
*       string SECONDS. Pointer to a pointer used for
*       dynamic memory allocation.
*     - char ** life_duration - used to return the life
*       time duration string . Pointer to a pointer used for
*       dynamic memory allocation.
*
*****/

```

```

void life_seconds_translation(char ** life, char**life_type, char
**life_duration) {

```

```

    if (strcmp(*life, "600")== 0) {

        free(*life);

        *life = strdup("LIFE_600_SECS");
        *life_type = strdup("SECONDS");
        *life_duration = strdup("600,450:720");

    }

    else if (strcmp(*life, "3600")== 0) {

        free(*life);
        *life = strdup("LIFE_3600_SECS");
        *life_type = strdup("SECONDS");
        *life_duration = strdup("3600,1800:7200");

    }

```

```

else {
    free(*life);
    *life = strdup("LIFE_3600_SECS");
    *life_type = strdup("SECONDS");
    *life_duration = strdup("3600,1800:7200");

}

}

```

/*****

Function: void group_description_translation- this function is used to convert group description from the KeyNote/**isakmpd.policy** syntax to the **isakmpd.conf** syntax.

- Inputs:
- Note: That in order to facilitate dynamic memory use, pointer to pointer coding syntax at times was required. By having a pointer to a pointer, memory created in a function will still be resident/within scope after returning from the function.
- char ** group_description- holds the initial group description variable. Pointer to a pointer used for dynamic memory allocation.
- Outputs:
- char ** group_description - used to return translated group_description. Pointer to a pointer used for dynamic memory allocation.

*****/

```

void group_description_translation(char ** group_description) {

    if (strcmp(*group_description, "1")== 0) {

        free(*group_description);
        *group_description = strdup("MODP_768");
    }

    else if (strcmp(*group_description,"2")==0) {

        free(*group_description);
        *group_description = strdup("MODP_1024");
    }

    else if (strcmp(*group_description,"3")==0) {

        free(*group_description);
        *group_description = strdup("MODP_155");
    }

    else if (strcmp(*group_description,"4")==0) {

        free(*group_description);
    }
}

```

```

    *group_description = strdup("MODP_185");
}

else if (strcmp(*group_description,"5")==0) {

    free(*group_description);
    *group_description = strdup("MODP_1536");
}
else {

    free(*group_description);
    *group_description = strdup("MODP_768");
}

}

/*****

- Function: void add_para_values()- this function generates the
security proposal format required by the configuration process.

- Note: That in order to facilitate dynamic memory use,
pointer to pointer coding syntax at times was required. By having a
pointer to a pointer, memory created in a function will still be
resident/within scope after returning from the function.

- Inputs:
- char ** suite - holds the set of security proposals. Pointer to a
pointer used to for dynamic memory creation.
- struct suite_struct **suite_profile - pointer to suite structure.
- Outputs:
- char ** suite - returns the modified set of security proposals.
Pointer to a pointer used to for dynamic memory creation.
- struct suite_struct **suite_profile - pointer to suite structure
used to return the modified suite_profile structure.

*****/

```

```

*****/

int add_para_values(char ** suite, struct suite_struct *suite_profile,
struct suite_struct * suite_list, int SA_counter){

    int success = 0;
    char * temp_hold4;
    //int suite_current_length=0;
    int max_SA_suite_size = 50;

    // Checking for duplicate SA....

    if (duplicate_sa(suite_list,SA_counter,suite_profile) == 0 ) {

        success = 1;

        // allocated enough space for SA

```

```

suite_profile->suite_name=(char*)malloc(max_SA_suite_size);

if (*suite != NULL) {
    if ((*suite =realloc(*suite, (strlen(*suite) +
max_SA_suite_size))) == NULL) {
        LOG_DBG ((LOG_POLICY, 40, "Memory Reallocation error...", ""));
        return(0);
    }
    (*suite) = strncat(*suite, ",", strlen(","));
}
else { // suite equals NULL

    // Allocated max space needed for SA
    (*suite)=(char*)malloc(max_SA_suite_size);
}

// use length of string for coping of suite info at end of routine

suite_profile->suite_name = strcpy(suite_profile->suite_name,"QM");

    // check for missing ESP or AH but existing ESP alg or AH alg

if ((suite_profile->esp == NULL) && (suite_profile->ah == NULL)) {
    LOG_DBG ((LOG_POLICY, 40, "ESP & AH are empty....", ""));
    // check for existing ESP enc alg or ESP auth alg then add ESP
    if ((suite_profile->esp_enc_alg!= NULL)|| (suite_profile->
>esp_auth_alg!=NULL)) {
        suite_profile->suite_name = strncat(suite_profile->suite_name,"-
ESP",strlen("-ESP"));
    }
    // check for AH auth alg then add AH
    else if (suite_profile->ah_auth_alg != NULL) {
        suite_profile->suite_name = strncat(suite_profile->
>suite_name,"-AH", strlen("-AH"));
    }
}

// ESP or AH exist....

```

```

else {

    if (suite_profile->esp != NULL) {
        suite_profile->suite_name = strncat(suite_profile->suite_name, "-",
            strlen("-"));
        suite_profile->suite_name = strncat(suite_profile->suite_name,
            suite_profile->esp, strlen(suite_profile->esp));
    }

    else if (suite_profile->ah != NULL) {
        suite_profile->suite_name = strncat(suite_profile->suite_name, "-",
            strlen("-"));
        suite_profile->suite_name = strncat(suite_profile->suite_name,
            suite_profile->ah, strlen(suite_profile->ah ));
    }
}

if (suite_profile->esp_enc_alg != NULL) {

    suite_profile->suite_name = strncat(suite_profile->suite_name, "-",
        strlen("-"));
    suite_profile->suite_name = strncat(suite_profile->suite_name,
        suite_profile->esp_enc_alg, strlen(suite_profile->esp_enc_alg ));
}

if (suite_profile->esp_auth_alg != NULL) {

    suite_profile->suite_name = strncat(suite_profile->suite_name, "-",
        strlen("-"));
    suite_profile->suite_name = strncat(suite_profile->suite_name,
        suite_profile->esp_auth_alg, strlen(suite_profile->esp_auth_alg ));
}

if (suite_profile->ah_auth_alg != NULL) {

    suite_profile->suite_name = strncat(suite_profile->suite_name, "-",
        strlen("-"));
    suite_profile->suite_name = strncat(suite_profile->suite_name,
        suite_profile->ah_auth_alg, strlen(suite_profile->ah_auth_alg ));
}

// check for PFS .. if true add -PFS

// Check to make sure PFS is not NULL...

if (suite_profile->pfs != NULL) {

```



```

    if (strcmp(suite_profile->pfs,"PFS")==0) {

        suite_profile->suite_name = strncat(suite_profile->suite_name,"-
PFS",strlen("-PFS"));

    }

    else {

        LOG_DBG ((LOG_POLICY, 40, "PFS is NULL",""));
    }

    // dynamially creat memory for SA info and generate proper SA syntax

    suite_profile->suite_protocol = strdup(suite_profile->suite_name);

    suite_profile->suite_transform = (char*) malloc(strlen(suite_profile-
>suite_protocol) + 4);

    suite_profile->suite_transform = strcpy(suite_profile-
>suite_transform,suite_profile->suite_protocol);

    suite_profile->suite_transform = strncat( suite_profile-
>suite_transform,"-XF",strlen("-XF"));

    LOG_DBG ((LOG_POLICY, 40, "Protocol suite: AFTER:
%s",suite_profile->suite_protocol));
    LOG_DBG ((LOG_POLICY, 40, "Transform suite: AFTER: %s",suite_profile-
>suite_transform));

    LOG_DBG ((LOG_POLICY, 40, "Suite Name before: %s and size of:
%i",suite_profile->suite_name,strlen(suite_profile->suite_name) ));

    suite_profile->suite_name = strncat(suite_profile->suite_name,"-
SUITE",strlen("-SUITE"));

    LOG_DBG ((LOG_POLICY, 40, "Suite Name after: %s and size of:
%i",suite_profile->suite_name,strlen(suite_profile->suite_name) ));

    //dynamically assign space if necessary to suite
    if (*suite == NULL) {

        (*suite) = strdup(suite_profile->suite_name);

    }
    else {

        *suite = strncat(*suite, suite_profile->suite_name,
strlen(suite_profile->suite_name ));

    }

}

```

```

    else { // duplicate exists....
        success = 0;
    }

    return (success);
}

/*****

Function: char * convert_to_uppercase()- converts a lower case string
to an upper case string and returns the string.

```

- Input:
 - char * lowercase - lower case string.
- Output:
 - char * - returns uppercase string.

```

*****/

```

```

char * convert_to_uppercase(char * lowercase_string) {

    int string_size = strlen(lowercase_string);
    char * uppercase_string = (char *) malloc(100);
    char temp_char;
    int c;

    for ( c=0; c< string_size; c++) {

        temp_char = lowercase_string[c];
        temp_char = toupper(temp_char);

        uppercase_string = strcat(uppercase_string,&temp_char,1);
    }

    lowercase_string = strcpy(lowercase_string,uppercase_string);

    return (lowercase_string);
}

```

```

/*****

```

- Function: verify_parameter()- checks input dynamic parameter tag value and if valid, compares tag value with given value. Returns three possible flag values.

- Input:
 - char *buff_temp - string buffer used for KeyNote file.
 - int *buff_temp_counter - index of pointer in buff_temp string buffer.
 - char * sys_para_name - Dynamic parameter tag
 - char * sys_para_value - Dynamic parameter tag value

- int buff_temp_end - index of end-of-file (EOF) in buff_temp.

- Output:

- int - flag with the following three values:
 - 0 (dynamic parameter tag value does not match)
 - 1 (dynamic parameter tag value matches)
 - 2 (dynamic tag does not match)

*****/

```
int verify_parameter(char *buff_temp, int *buff_temp_counter, char *
sys_para_name, char * sys_para_value,int buff_temp_end) {
```

```
    int sys_para_name_size = strlen(sys_para_name);
    int sys_para_value_size = strlen(sys_para_value);
    char * temp_string = (char *) malloc(buff_temp_end);
    char * temp_hold = (char*)malloc(100);
    int success = 2;
    char * temp_str = (char *) malloc(100);
```

```
    temp_str = strncpy(temp_str,&buff_temp[*buff_temp_counter],25);
```

```
    temp_string= &buff_temp[*buff_temp_counter];
```

```
    // compare tag titles
```

```
    if (strncmp (&buff_temp[*buff_temp_counter], sys_para_name,
sys_para_name_size) == 0) {
```

```
        (*buff_temp_counter)+= sys_para_name_size;
```

```
        // advance pointer to "
```

```
        while (buff_temp[*buff_temp_counter] != '\\") {
```

```
            (*buff_temp_counter)++;
```

```
        }
```

```
        // advance to pass ending "
```

```
        (*buff_temp_counter)++;
```

```
        // compare tag value with current selection parameter
```

```
        // if valid save value and advance counter
```

```
        temp_str =
        strncpy(temp_str,&buff_temp[*buff_temp_counter],sys_para_value_size);
        temp_str[sys_para_value_size]='\0';
```

```
        if (strncmp(&buff_temp[*buff_temp_counter], sys_para_value,
sys_para_value_size) == 0) {
```

```
            LOG_DBG ((LOG_POLICY, 40, "Parameter value MATCH",""));
```

```
            temp_hold = strncpy(temp_hold,&buff_temp[*buff_temp_counter],
```

```

sys_para_value_size);

    temp_hold[sys_para_value_size]='\0';

    //set flag to true

    success = 1;

    // advance buffer counter to end of tag value
    (*buff_temp_counter)= (*buff_temp_counter) +
sys_para_value_size;

    // make temp_hold all UPPER_CASE

    temp_hold= convert_to_uppercase(temp_hold);

}

else {
    success = 0;

} // end else

}

return (success);

}

/*****
- Function: advance_to_end_DNF()- advances file pointer to the next DNF
security proposal.

- Input:
    - char *buff_temp - string buffer used for KeyNote
    file.
    - int *buff_temp_counter - index of pointer in buff_temp string
    buffer.
    - int buff_temp_end - index of end-of-file (EOF) in buff_temp.
- Output:
    - int *buff_temp_counter - index of pointer in buff_temp string
    buffer.

*****/

void advance_to_end_DNF(char *buff_temp, int *buff_temp_counter, int
buff_temp_end) {

    // advance buffer to end of expression

    while ((*buff_temp_counter <= buff_temp_end) &&

```

```

(buff_temp[*buff_temp_counter] != '|')) {
    (*buff_temp_counter)++;
}

}

```

```

/*****

```

Function: **void parse_ipsec_parameter(char *buff_temp, int *buff_temp_counter, char * sys_para_name, char ** temp_hold, int *success)** - verifies that the tag is the expected tag and then parses the tag and the tag value, storing information in input suite structure char string.

- Inputs:

- char *buff_temp - pointer to the file being parsed.
- int *buff_temp_counter - pointer to index of character in file being parsed.
- char * sys_para_name - pointer to the expected parameter tag name
- char ** temp_hold - pointer to a pointer (used for the purpose of dynamic memory allocation) of char string in suite structure.
- int *success - pointer to an integer used for the success flag.

- Outputs:

- int *buff_temp_counter - pointer to index of character in file being parsed is returned via pointer reference. Pointer may be advance in function.
- char * temp_hold - pointer to a pointer of a character string (used for the purpose of dynamic memory allocation) in the suite structure returned via pointer reference.
- int *success - pointer to an integer used to hold success flag returned via reference.

```

*****/

```

```

void parse_ipsec_parameter(char *buff_temp, int * buff_temp_counter,
char * sys_para_name, char ** temp_hold, int * success) {

```

```

    int temp_counter, temp_buffer;

```

```

    int sys_para_name_size = strlen(sys_para_name);
    int sys_para_value_size = 0;
    char * temp_string;
    temp_string = (char *) malloc(100);

```

```

    // CHECKING FOR TAG LABEL MATCH

```

```

    if (strcmp (&buff_temp[*buff_temp_counter], sys_para_name,
sys_para_name_size) == 0) {

```

```

        LOG_DBG ((LOG_POLICY, 40, "Match Found... copying label info",""));

```

```

*buff_temp_counter+= sys_para_name_size;

// ADVANCE TO THE START OF PARATHESISSES
while (buff_temp[*buff_temp_counter] != '"') {

    sys_para_value_size++;
    (*buff_temp_counter)++;

}

// advance passed opening "
(*buff_temp_counter)++;

// COUNT THE SIZE OF TAG VALUE - UNTIL CLOSE "

temp_counter = 0;
temp_buffer = *buff_temp_counter;

while (buff_temp[temp_buffer] != '"') {

    temp_buffer++;
    temp_counter++;

}

//COPYING TAG VALUE

// creating dynamic memory space for variable storage
(*temp_hold) = (char*)malloc(temp_counter+1);

*temp_hold = strncpy(*temp_hold, &buff_temp[*buff_temp_counter],
temp_counter);

(*temp_hold)[temp_counter]='\0';

(*buff_temp_counter)= temp_buffer + 1;

*success=1; // set success flag to true

// make temp_hold all UPPER_CASE

*temp_hold = convert_to_uppercase(*temp_hold);

}

else {
    LOG_DBG ((LOG_POLICY, 40, "Match Not Found...", ""));
}

}

```

```

/*****

- Function: void parse_ipsec_para_tag()- verifies that the tag is the
expected tag and that tag value contains "yes". If so, parses tag and
stores its value in suite structure char string.

- Inputs:
- char *buff_temp - pointer to the file being parsed.
- int *buff_temp_counter - pointer to index of character in file being
parsed.
- char * sys_para_name - pointer to the expected parameter tag name
- char ** temp_hold - pointer to a pointer (used for the purpose of
dynamic memory allocation) of char string in suite structure.
- int sys_para_name_reduced - integer that holds the string size of the
tag. Either 2 or 3 used for AH, ESP or PFS tags.
- int *success - pointer to an integer used for the success flag.

-Outputs:
- int *buff_temp_counter - pointer to the index of character in file
being parsed is returned via pointer reference. Pointer may be advanced
in the function.
- char * temp_hold - pointer to a pointer of a character string (used
for the purpose of dynamic memory allocation) in the suite structure
returned via pointer reference.
- int *success - pointer to an integer used for the success flag
returned via reference.

*****/

```

```

*****/

void parse_ipsec_para_tag(char *buff_temp, int * buff_temp_counter,
char * sys_para_name, char ** temp_hold, int sys_para_name_reduced, int
* success) {

    //int temp_counter;
    int temp_buffer;

    int sys_para_name_size = strlen(sys_para_name);
    //int sys_para_value_size = 0;
    char * bool_tag = "yes";
    int bool_tag_size = 3;
    char * temp_string;
    temp_string = (char*) malloc (100);

    // CHECKING FOR TAG LABEL MATCH

    if (strncmp (&buff_temp[*buff_temp_counter], sys_para_name,
sys_para_name_size) == 0) {
        *buff_temp_counter+= sys_para_name_size;

        temp_buffer = *buff_temp_counter;
    }
}

```

```

// ADVANCE TO THE START OF PARATHESISES
while (buff_temp[temp_buffer] != '') {
    temp_buffer++;
}

// advance passed opening "
temp_buffer++;

// Check to see if boo_tag equals "YES"
if (strcmp (&buff_temp[temp_buffer], boo_tag, boo_tag_size)
== 0) {
    *buff_temp_counter = temp_buffer;

    *temp_hold=(char*)malloc(sys_para_name_reduced+1);

    *temp_hold = strncpy(*temp_hold, sys_para_name,
sys_para_name_reduced);

    (*temp_hold)[sys_para_name_reduced]='\0';

    (*buff_temp_counter)+= boo_tag_size;

    *success = 1; // set success flag to true

    // make temp_hold all UPPER_CASE
    *temp_hold=convert_to_uppercase(*temp_hold);
}
else {
    LOG_DBG ((LOG_POLICY, 40, "Tag bool match not found...", ""));
}
}
else {
    LOG_DBG ((LOG_POLICY, 40, "Tag Name no match....", ""));
}
}

```

[In the function conf_reinit() the following line of code is added]

```

con_kn_parse(trans);

```


THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. IKE_QUICK_MODE.C

The following is the code added to /usr/src/sbin/isakmpd/ike_quick_mode.c:

```
/*
 *
 *      Structure: dynamic_packet
 *
 *      This structure is used to stored dynamic parameter
 *      data, including dynamic parameter name, assignment symbol
 *      and value.
 *
 */
*****

struct dynamic_packet {

    char* title;
    char* symbol;
    char* value;
};

struct dynamic_packet* package_dynamic_parameters(int *);
struct dynamic_packet * package_dynamic_parameters2( int *);

/*
 *
 *      Function: package_dynamic_parameters() This function reads a file that
 *      contains the current inputs of dynamic parameters such as
 *      Network Mode and Security Level. Initializes an array of
 *      structures that will dynamically grow as required. Loads the
 *      file input into the array of struct.
 *
 *      - Input:
 *          - int * package_counter - used to determine the size of the
 *            struct array
 *
 *      - Output:
 *          - struct dynamic_packet * - pointer to an array of
 *            dynamic_packet structure
 *
 */
*****/

struct dynamic_packet * package_dynamic_parameters( int
*package_counter) {

    int array_size = 10 ; // chosen as average case for struct size

    struct dynamic_packet * package; // pointer to array of struct

    // strings used to read from file
```

```

char * title = (char*)malloc(20);
char * symbol= (char*)malloc(20);
char * value = (char*)malloc(20);

// used for file IO
FILE * pifp;
int file_status=0;

(*package_counter) = 0; // initialize the array size to zero

// initialize the pointer to an array size of array_size

package = (struct dynamic_packet *)malloc(sizeof(struct
dynamic_packet)*array_size);

// open file to read in dynamic parameters
// check for errors in opening the file

if ( (pifp = fopen("/usr/src/sbin/isakmpd/dynamic_parameters",
"r"))== NULL) {
    // error opening the file

    LOG_DBG ((LOG_POLICY, 40, "ERROR OPENNING FILE %d",errno));
}
else { // file was successfully opened

    // do-while loop to read in data until EOF reached

    do {

        // read in from the file expecting the following syntax:
        //          title symbol value

        file_status = fscanf(pifp, "%s%s%s", title, symbol, value);

        LOG_DBG ((LOG_POLICY, 40, "Just read from file....",""));
        LOG_DBG ((LOG_POLICY, 40, "Title: %s Symbol: %s Value: %s",
            title, symbol, value));

        // check for reading errors...

        if (file_status == 0) {

            // reading error occurred

            LOG_DBG ((LOG_POLICY, 40, "error reading from the
file....",""));
        }

        // check for EOF...

        else if (file_status == EOF) {

            // EOF reached

            LOG_DBG ((LOG_POLICY, 40, "end of file....",""));

```

```

    }

    // otherwise... read operation successful

    else {

        // create new struct space for dynamic parameters and
        // assign values from file to struct

        package[*package_counter].title = strdup(title);
        package[*package_counter].symbol = strdup(symbol);
        package[*package_counter].value = strdup(value);

        // increment package counter array of struct
        (*package_counter)++;

        // dynamic resize array if more memory is required

        if ((*package_counter) == array_size) {

            array_size+=10; // grow array in increments of 10

            package = (struct dynamic_packet
*)realloc(package,sizeof(struct dynamic_packet)*array_size);
        }
    }
}
while ((file_status != 0) && (file_status !=EOF));

fclose(pifp); // close file

test_print_dynamic_packet(package,*package_counter);

}

// free memory
free(title);
free(symbol);
free(value);
// free(pifp);

return (package);
}

```

[In the check_policy() function the following lines of code are added to read int dynamic parameters and then load them in the KeyNote query mechanism]

```

package = package_dynamic_parameters2(&package_size);

// load Dynamic parameters into KeyNote

for (c= 0; c < package_size;c++) {

```

```

    if (LK (kn_add_action, (isakmp_sa->policy_id,
        package[c].title,package[c].value,0)) == -1)
    {
        log_print ("CHECK_POLICY: "
            "kn_add_action loading FAILED for title: %s value: %s",
            package[c].title,package[c].value);
        LK (kn_close, (isakmp_sa->policy_id));
        isakmp_sa->policy_id = -1;
        return 0;
    }
    else {

        LOG_DBG ((LOG_POLICY, 40, "CHECK_POLICY: load successful for
            title: %s value: %s",package[c].title,package[c].value));
    }
}

```

APPENDIX C. DEMO.JAVA

```
/******  
Class demo.java - This class creates and handles the generation of the  
welcome screen and menu choices.
```

```
*****/
```

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.Toolkit;  
import kaffe.awt.*;  
import java.awt.event.*;  
import java.io.*;  
import java.util.*;  
import dp_console;  
import demo_support_functions;  
import ipsecinfo;
```

```
public class demo extends JFrame {  
    private SPD spd;  
    private JPanel p, ptitle, pchoices;  
    private JButton enter;  
    Color bg_color = Color.white;  
  
    private JRadioButton sp_choice, dp_choice, ep_choice, ds_choice,  
b2_choice, b3_choice, b4_choice, ex_choice;  
    private JButton sp_button, dp_button, ep_button, ds_button,  
b2_button, b3_button, b4_button, ex_button;  
    private JLabel qoss_menu, status_label;  
    private dp_console dp;  
    private ipsecinfo ip;  
    private tcpdump tcp;  
    private SPFK spfk;  
    private static Runtime rt;  
    private demo_support_functions dsf;  
    // private Thread file_copy fc;  
  
    private Image i;
```

```
/******  
Method: demo - class constructor - constructor - initializes the  
welcome screen.
```

```
*****/
```

```
    public demo() {  
  
        super("QoSS IPsec");  
        dsf = new demo_support_functions();  
        dp = new dp_console();  
  
        dp.addWindowListener(  

```

```

new WindowAdapter() {
    public void windowClosing(WindowEvent e)
    {

        System.exit(0);
    }
});

p = new JPanel();
pchoices = new JPanel();
setContentPane(p);
p.setBackground(bg_color);
p.repaint();
pchoices.setBackground(bg_color);
pchoices.repaint();

p.setLayout(new FlowLayout() );
enter = new JButton("Continue");
ContinueButtonHandler ct_handler = new
ContinueButtonHandler();
enter.addActionListener(ct_handler);

JLabel label;
p.add(label = new JLabel("OpenBSD "));
label.setForeground(Color.black);
label.setFont(new Font("Serif",Font.BOLD,20));
p.add(label = new JLabel("IPsec "));
label.setForeground(Color.blue);
label.setFont(new Font("Serif",Font.BOLD,50));

p.add(label = new JLabel("Dynamic Parameterization"));
label.setForeground(Color.red);
label.setFont(new Font("Serif",Font.ITALIC,40));
p.add(label = new JLabel("
Chris Agar, LT USN
"));
label.setForeground(Color.black);
label.setFont(new Font("Serif",Font.BOLD,15));
p.add(enter);
p.repaint();
setSize(500,175);
setVisible(true);
status_label = new JLabel("");
qoss_menu = new JLabel("QoS IPsec Demonstration Selection
Menu");

sp_button = new JButton("Start IPsec");
dp_button = new JButton("Dynamic Parameterization");
ep_button = new JButton("Stop IPsec");
ds_button = new JButton("Display SAD");
b2_button = new JButton("Display TCPDUMP");
b3_button = new JButton("Display SPD");
b4_button = new JButton("Display Security Policy");

```

```

        ex_button = new JButton("Exit");

        SPButtonHandler spb_handler = new SPButtonHandler();
        DPButtonHandler dpb_handler = new DPButtonHandler();
        DSButtonHandler dsb_handler = new DSButtonHandler();
        EXButtonHandler exb_handler = new EXButtonHandler();
        TCPButtonHandler tcpb_handler = new TCPButtonHandler();
        SPDBButtonHandler spdb_handler = new SPDBButtonHandler();
        DSPButtonHandler dspb_handler = new DSPButtonHandler();
        INFButtonHandler infb_handler = new INFButtonHandler();

        sp_button.addActionListener(spb_handler);
        dp_button.addActionListener(dpb_handler);
        ex_button.addActionListener(exb_handler);
        ds_button.addActionListener(dsb_handler);
        b2_button.addActionListener(tcpb_handler);
        b3_button.addActionListener(spdb_handler);
        b4_button.addActionListener(dspb_handler);
        ep_button.addActionListener(infb_handler);

        // initialize connection_index
        initialize_connection_index_file();
        show();

    }

    /*****
    Method: initialize_connection_index_file() - resets connection index
    counter to zero in the connection index counter file.
    - Input: none.
    - Output: none.
    *****/

    public void initialize_connection_index_file() {

        try {
            int temp_string = 0;
            System.out.println("Writing connection# to the file.");
            File f = new File("/root/demo/connection_number");

            FileOutputStream fos = new FileOutputStream(f);

            PrintStream out = new PrintStream(fos);

            out.println("0");

            fos.close();
            out.close();

        }

        catch (Exception e) {

            System.out.println("Execption Thrown in Write_CF e: " +e);

        }
    }

```



```

    }

/*****
Method: reset_error_panel() - refreshes error panel and removes old
message.
- Input: none.
- Output: none.
*****/

    public void reset_error_panel() {

        status_label.setText(" ");

    }

/*****
Method: load_dp_file(String nm, String sl) - accepts network mode and
security level inputs and writes them into the dynamic parameter file.
-Input:
- String nm- network mode value.
- String sl - security level value.
- Output: none.
*****/

    public void load_dp_file(String nm, String sl) {

        try {
            File f = new File("/usr/src/sbin/isakmpd/dynamic_parameters");

            FileOutputStream fos = new FileOutputStream(f);

            PrintStream out = new PrintStream(fos);

            out.println("network_mode = " + nm+ "\n");
            out.println("security_level = " + sl + "\n");

            fos.close();
            out.close();

        }
        catch (Exception e) {
            System.out.println("Exception Thrown in Write_DP e: " +e);
        }
    }

/*****
Class: ContinueButtonHandler implements ActionListener

- Method: actionPerformed(ActionEvent e) - action handler for the
continue button on the welcome JFrame.
*****/

```

```

private class ContinueButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        try{

            System.out.println("Continue Button Selected");

            p.removeAll();

            p.setLayout(new FlowLayout() );
            p.add(qoss_menu);
            p.add(status_label);
            pchoices.setLayout(new GridLayout(4,2));
            pchoices.add(sp_button);
            pchoices.add(ds_button);
            pchoices.add(dp_button);
            pchoices.add(ep_button);

            pchoices.add(b2_button);
            pchoices.add(b3_button);
            pchoices.add(b4_button);
            pchoices.add(ex_button);
            p.add(pchoices);

            setSize(450,150);
            setVisible(true);

        }

        catch (Exception s) {
            System.out.println("Exception thrown in Continue Button
Handler.");
            System.out.println("Exception: "+s);
        }

    }

}

```

/*****

Class: **SPRadioButtonHandler** implements ItemListener

- Method: **itemStateChanged(ItemEvent e)** - action handler for the **start IPsec** radio button.

*****/
// Start IPsec Process

```

private class SPButtonHandler implements ActionListener {

```

```

public void actionPerformed(ActionEvent e)
{
    reset_error_panel();

    System.out.println("Start IPsec choice was selected.");

    // check to see if process is running
    if (dsf.daemon_running()) {

        status_label.setText("IPsec is already running!");

    }

    else {    // ipsec is not running

        // load default security level and network mode into
dp file

        load_dp_file("default","default");
        initialize_connection_index_file();

        // Flushing ipsec mechanism
        dsf.flush_ipsec();

        // Mount kernel
        dsf.mount_kern();

        // LOad SPD
        dsf.load_spd();

        //start ipsec mechanism
        dsf.start_ipsec();

    }

}

/*****

Class: DSRadioButtonHandler implements ItemListener

- Method: itemStateChanged(ItemEvent e) - action handler for Display SAD radio button.

*****/

private class DSButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();

        System.out.println("Display      Negotiated      SA's      was
selected.");

        ip = new ipsecinfo();
        ip.start();

```

```

    }
}

/*****

Class: INFButtonHandler implements ItemListener- item listener for stop
ipsec button.

- Method: actionPerformed(ActionEvent e) - action handler for Exit
menu.

*****/

private class INFButtonHandler implements ActionListener {

    public void actionPerformed (ActionEvent e)
    {

        reset_error_panel();
        System.out.println("Stop IPsec choice was selected.");

        if (dsf.daemon_running()) {

            System.out.println("IPsec is running....");
            dsf.tear_down_connection();
            dsf.stop_ipsec();
        }
        else System.out.println("IPsec is not running....");

    }
}

/*****

Class: TCPButtonHandler implements ActionListener

- Method: actionPerformed(ActionEvent e) - action handler for Display
tcpdump button.

*****/

private class TCPButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();
        System.out.println("TCP Dump start....");

        tcp = new tcpdump();
        tcp.start();

    }
}

```

```

/*****

Class: SPDButtonHandler implements ActionListener

- Method: actionPerformed(ActionEvent e) - action handler for Display SPD button.

*****/

private class SPDButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();
        System.out.println("Display SPD");

        spd = new SPD();

    }

}

/*****

Class: DSPButtonHandler implements ActionListener

- Method: actionPerformed(ActionEvent e) - action handler for Display Security Policy button.

*****/

private class DSPButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();
        System.out.println("Display Security Policy");

        spfk = new SPFK();

    }

}

/*****

Class: EXButtonHandler implements ActionListener- action handler for Exit button.

- Method: actionPerformed(ActionEvent e) - action handler for Exit menu.

```

```

*****/

private class EXButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();
        System.out.println("Exit IPsec choice was selected.");

        if (dsf.daemon_running()) {

            System.out.println("IPsec is running...");
            dsf.tear_down_connection();
            dsf.stop_ipsec();

        }

        System.exit(0);
    }
}

/*****

Class: DPButtonHandler implements ActionListener

- Method: actionPerformed(ActionEvent e) - action handler for Dynamic Parameterization button.

*****/

private class DPButtonHandler implements ActionListener {

    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();
        //dp_choice.setSelected(false);
        System.out.println("Dynamic Parameter selection choice was
selected.");
        dp.start_dp_console();
    }
}

/*****

Method: static main(String args) - the main program of the demo class.

*****/

public static void main (String args[])
{
    demo d = new demo();
    rt = Runtime.getRuntime();

```

```
d.addWindowListener(  
    new WindowAdapter() {  
        public void windowClosing(WindowEvent e)  
        {  
            System.exit(0);  
        }  
    });  
  
}  
  
}
```

APPENDIX D. SPD.JAVA

```

/*****
    Class SPD.java - This class creates and handles the
    generation of the Security Policy Database.
*****/

// SPD.java

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.text.Document;
import javax.swing.text.BadLocationException;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Color;
import java.awt.Font;
import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.*;
import java.awt.*;
import java.util.NoSuchElementException;
import java.util.StringTokenizer;
//import java.net.*;
import java.io.*;

public class SPD {
    JSeparator separator;
    demo_support_functions dsf;
    private static Runtime rt;
    InputStream in;
    Process p;
    FileInputStream fin;
    JPanel contentPane;
    JPanel panel, toolbar;
    JPanel [] subpanel1, subpanel2, subpanel3;
    JButton exitButton;
    JTextField []sourceIP;
    JTextField []sourcePort;
    JTextField []destIP;
    JTextField []destPort;
    JTextField []protocol;
    JTextField []sa;
    JFrame j;

/*****
    Method: SPD - class constructor - constructor - initializes the
    SPD JFrame.
*****/
}

```



```

public SPD() {
    j = new JFrame("Security Policy Database");

    j.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            doExit();
        }
    });

    rt = Runtime.getRuntime();
    dsf = new demo_support_functions();

    // build GUI

    contentPane = new JPanel(new BorderLayout());
    panel = new JPanel(new GridLayout(0,1));

    subpanel1 = new JPanel[12];
    subpanel2 = new JPanel[12];
    subpanel3 = new JPanel[12];
    sourceIP = new JTextField[12];
    sourcePort = new JTextField[12];
    destIP = new JTextField[12];
    destPort = new JTextField[12];
    protocol = new JTextField[12];
    sa = new JTextField[12];

    toolbar = new JPanel();

    for (int c=0;c<11;c++) {
        // source IP
        subpanel1[c] = new JPanel(new GridLayout(1,4));
        //panel.add(subpanel[0]);
        subpanel1[c].add(new JLabel("Source IP:  "));
        subpanel1[c].add(sourceIP[c] = new JTextField("DATA NOT
AVAILABLE"));
        sourceIP[c].setEditable(false);

        // source port

        subpanel1[c].add(new JLabel("Source Port:  "));
        subpanel1[c].add(sourcePort[c] = new JTextField(4));
        sourcePort[c].setEditable(false);

        // dest IP
        subpanel2[c] = new JPanel(new GridLayout(1,4));
        subpanel2[c].add(new JLabel("Destination IP:  "));
        subpanel2[c].add(destIP[c] = new JTextField(20));
        destIP[c].setEditable(false);

        // dest port
        subpanel2[c].add(new JLabel("Destination Port:  "));
        subpanel2[c].add(destPort[c] = new JTextField(4));
        destPort[c].setEditable(false);
    }
}

```

```

        // protocol
        subpanel3[c] = new JPanel(new GridLayout(1,4));
        subpanel3[c].add(new JLabel("Protocol:  "));
        subpanel3[c].add(protocol[c] = new JTextField(4));
        protocol[c].setEditable(false);

        // SA
        subpanel3[c].add(new JLabel("SA(Addr/Proto/Type/Dir):"));
        subpanel3[c].add(sa[c]= new JTextField(20));
        sa[c].setEditable(false);
    }

    // separator
    separator = new JSeparator(SwingConstants.HORIZONTAL);

    // panel for exit button

    toolbar.add(exitButton = new JButton("Exit"));
    exitButton.setToolTipText("Exit Security Policy Database");
    exitButton.setMnemonic ('x');
    exitButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            doExit();
        }
    });

    contentPane.add(panel, BorderLayout.NORTH);
    contentPane.add(toolbar, BorderLayout.SOUTH);
    j.setContentPane(contentPane);
    panel.setVisible(true);
    toolbar.setVisible(true);

    // now fill in actual values
    setTextFields();
}

// -----
// exit
// -----
void doExit() {
    j.dispose();
}

/*****

    Method: String create_SPD_input() - this method retrieves
    the SPD data, stores it in a string and returns it to the
    calling function.

- Input: none.
- Output:
    - String - contains the retrieved SPD data.

*****/

String create_SPD_input() {

```

```

String SPD_s = new String();

try {

    String[] s = new String[3];
    s[0]= new String("/bin/sh");
    s[1] = new String("-c");
    s[2] = new String("netstat -rn -f encap");

    p=rt.exec(s);
    p.waitFor();
    in = p.getInputStream();
    System.out.println("buffer size: "+in.available());
    byte[] buffer = new byte[in.available()];
    in.read(buffer);
    SPD_s = new String(buffer);
    System.out.println("spd buffer: \n"+SPD_s);

}
catch (Exception e) {
    System.out.println("Exception thrown in create spd input
"+e);
}

return (SPD_s);
}

```

```

/*****

    Method: setTextFields() - this method parses the input
    string and displays the SPD data in the JFrame.
- Input: none.
- Output: none

```

```

*****/

```

```

void setTextFields() {
    String s = create_SPD_input();
    int field_counter = -1;

    StringTokenizer st = new StringTokenizer(s);
    try {
        if (st.hasMoreTokens()) {

            System.out.println("SPD data...");
            // skip initial tokens
            for (int i=0; i<9; i++) {
                System.out.println("next spd token: "+st.nextToken());
            }

            // set text fields

            // Check for no entries in table
            if (st.hasMoreTokens()) {
                //panel.removeAll();

```

```

while (st.hasMoreTokens()) {
    field_counter= field_counter +1;

    String temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    sourceIP[field_counter].setText(temps);

    temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    sourcePort[field_counter].setText(temps);
    panel.add(subpanel1[field_counter]);
    subpanel1[field_counter].setVisible(true);

    temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    destIP[field_counter].setText(temps);

    temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    destPort[field_counter].setText(temps);
    panel.add(subpanel2[field_counter]);
    subpanel2[field_counter].setVisible(true);

    temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    protocol[field_counter].setText(temps);

    temps = st.nextToken();
    System.out.println("next spd token: "+temps);
    sa[field_counter].setText(temps);
    panel.add(subpanel3[field_counter]);
    subpanel3[field_counter].setVisible(true);

    panel.add(
JSeparator(SwingConstants.HORIZONTAL));
    }
    }
    else { // empty table
        panel.add(new JLabel("No Entries Exist in the SPD"));
    }
}
else {
    System.out.println("Empty String...no tokens to parse.");
}
} catch (NoSuchElementException e) {
    System.out.println("SPD tokenizer error in string <"
        + s + ">\n" + e);
}

j.pack();
j.setVisible(true);
}

```

}

APPENDIX E. DEMO_SUPPORT_FUNCTIONS.JAVA

```
/******
```

Class: **demo_support_functions** – contains support functions for other classes.

```
*****/
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
```

```
public class demo_support_functions {
```

```
    private static Runtime rt;
    private int connection_count = 0;
    private int connection_index = 0;
    private Process p;
    private InputStream in;
```

```
/******
```

Method: **demo_support_functions** constructor.

```
*****/
```

```
public demo_support_functions() {
```

```
    try {
        connection_count = 0;
        connection_index = 0;
        rt = Runtime.getRuntime();
    }
```

```
    catch (Exception e) {
```

```
        System.out.println("Exception thrown in dsf constructor :
"+e);
    }
}
```

```
/******
```

Method: **flush_ipsec()** – generates the run time commands to flush the IPsec mechanism.

- Input: none.
- Output: none.

*****/

```
public void flush_ipsec() {
    try {
        System.out.println("performing ipsecadm flush...");
        String[] s = new String[3];
        s[0]= new String("/bin/sh");
        s[1] = new String("-c");
        s[2] = new String("ipsecadm flush");
        rt.exec(s).waitFor();
    }
    catch (Exception e) {
        System.out.println("Excpetion thrown in flush_ipsec : "+e);
    }
}
```

/*****/

Method: **load_spd()** - generates the run time commands to load SPD with the security policy.

- Input: none.
- Output: none.

*****/

```
public void load_spd() {
    try {
        System.out.println("performing sh vpn28_ah_a - loading SPD with defaults");
        String[] s2 = new String[3];
        s2[0]= new String("/bin/sh");
        s2[1] = new String("-c");
        s2[2] = new String("sh /root/vpn28_ah_a");
        rt.exec(s2);
    }
    catch (Exception e) {
        System.out.println("Excpetion thrown in load_spd : "+e);
    }
}
```

```

/*****

        Method: mount_kern() - generates the run time commands to
        mount the kernel.

- Input: none.
- Output: none.

*****/

```

```

public void mount_kern() {
    try {
        System.out.println("Mounting Kern");

        String[] s3 = new String[3];
        s3[0]= new String("/bin/sh");
        s3[1] = new String("-c");
        s3[2] = new String("sh /root/mount_kern");
        rt.exec(s3);
    }
    catch (Exception e) {
        System.out.println("Excpetion thrown in mount_kern : "+e);
    }
}

```

```

/*****

        Method: start_ipsec() - generates the run time commands to
        start the ipsec mechanism.

- Input: none.
- Output: none.

*****/

```

```

public void start_ipsec() {
    try {
        System.out.println("Executing isakmpd/ipsec");

        String[] s3 = new String[3];
        s3[0]= new String("/bin/sh");
        s3[1] = new String("-c");
        s3[2] = new String("isakmpd");
        rt.exec(s3).waitFor();
    }
}

```



```

    }
    catch (Exception e) {
        System.out.println("Excpetion thrown in start_ipsec : "+e);
    }
}

/*****

    Method: stop_ipsec() - generates the run time commands to
    stop the IPsec mechanism.
    - Input: none.
    - Output: none.

*****/

public void stop_ipsec() {
    try {
        System.out.println("Stopping IPsec...");
        File f = new File("/var/run/isakmpd.pid");
        if (f.exists()) {
            FileInputStream fis = new FileInputStream(f);
            byte[] buffer = new byte[fis.available()];
            fis.read(buffer);

            String tempString = new String();
            String kill_value = new String(buffer);

            System.out.println("Killing the ipsec process id: " +
kill_value);
            String[] s3 = new String[3];
            s3[0]= new String("/bin/sh");
            s3[1] = new String("-c");
            s3[2] = new String("kill " + kill_value);

            rt.exec(s3).waitFor();

            fis.close();
        }
    }
    catch (Exception e) {
        System.out.println("Excpetion thrown in stop_ipsec : "+e);
    }
}

```

```

}

/*****

        Method read_connection_index_file()- reads in the current
        value of the index counter from a file.
- Input: none.
  - Output: none.

*****/

public void read_connection_index_file() {
    System.out.println("Reading connection file");

    int test =5;

    try {
        File f = new File ("connection_number");

        FileInputStream fis = new FileInputStream(f);
        int size = fis.available();

        byte[] buffer = new byte[size];

        String ts = new String();

        fis.read(buffer);

        String tempString = new String(buffer);
        StringTokenizer st = new StringTokenizer(tempString);
        while (st.hasMoreTokens()) {
            try {
                ts = st.nextToken();
                System.out.println("ts token:"+ts+":");
                connection_index = Integer.parseInt(ts);
                System.out.println("ci:"+connection_index);
            }
            catch (Exception ef) {System.out.println("error dude:
"+ef);}
        }

        fis.close();

    }
    catch (Exception e) { System.out.println("Exception Thrown in
read_connection file: "+e + "connection: "+test);}
}

/*****

```

Method: **write_connection_index_file()**- writes the current value of the index counter to a file.

- Input: none.
- Output: none.

*****/

```
public void write_connection_index_file() {  
    try {  
        System.out.println("Writing connection# to the file.");  
        File f = new File("/root/demo/connection_number");  
        FileOutputStream("/root/demo/connection_number");  
        FileOutputStream fos = new FileOutputStream(f);  
        PrintStream out = new PrintStream(fos);  
        out.println(connection_index);  
  
        fos.close();  
        out.close();  
  
    }  
  
    catch (Exception e) {  
  
        System.out.println("Exception Thrown in Write_CF e: " +e);  
    }  
}
```

/*****/

Method: **daemon_running()** - checks to see if the IPsec process is currently running.

- Input: none.
- Output:
 - Boolean result - true if IPsec is currently running and false otherwise.

*****/

```
public boolean daemon_running() {  
    boolean result = false;  
  
    try {  
  
        System.out.println ("Testing to see if daemon is running...");  

```

```

        String[] s = new String[3];
        s[0]= new String("/bin/sh");
        s[1] = new String("-c");
        s[2] = new String("ps -ax | grep isakmpd | grep -v grep >
daemon_search");
        rt.exec(s).waitFor();

        File f = new File("daemon_search");

        if (f.exists()) {
            System.out.println("File daemon_search exists..");}
        else { System.out.println("file daemon_search does not exist!");}
            long test = f.length();

        FileInputStream in = new FileInputStream(f);

        int size_before = in.available();

        byte[] buffer = new byte[size_before];

        in.read(buffer);
        String tempString = new String();

        String record = new String(buffer);
        in.close();

        if (size_before > 0) {

            System.out.println("daemon is running.");

            // write to file to stop daemon...

            result = true;

        }

        else {
            System.out.println("daemon is not running.");
            result = false;
        }

    }
    catch (Exception e) {
        System.out.println("Exception Throw is_daemon_running: " +e);
    }

    return (result);

}

/*****

```

Method: pause() – sleeps for a given number of seconds.

- Input:

int – number of seconds.

- Output: void

*****/

```
public void pause(int t) {
    try {
        long pauseTime = (long) (t*1000);
        Thread.sleep(pauseTime);
    }
    catch (Exception etp) {
        System.out.print("Run Error: " + etp);
    }
}
```

*****/

Method: stop_tcpdump() - terminates tcpdump process.

- Input: void

- Output: void

*****/

```
public void stop_tcpdump() {
    boolean notDone = true;
    while (notDone) {
        try {
            System.out.println("in stop tcp_dump");
            String[] s = new String[3];
            s[0]= new String("/bin/sh");
            s[1] = new String("-c");
            s[2] = new String("ps -ax | grep tcpdump | grep -v grep |
grep -v /bin/sh");
            p = rt.exec(s);

            p.waitFor();

            in = p.getInputStream();
            System.out.println("performed rt.");
        }
        catch (Exception e) {
            System.out.println("Exception thrown in stop_tcp_dump"+e);
        }

        String line = null;

        try {
            BufferedReader br = new BufferedReader(new
InputStreamReader(in));
```

```

        line = br.readLine();

        if (line != null) {

            System.out.println("tcpdump found running... process output
#: "+line);

            pause(3);

            String s = new String(line);

            StringTokenizer st = new StringTokenizer(s);

            String kill_value = st.nextToken();

            String[] s3 = new String[3];
            s3[0]= new String("/bin/sh");
            s3[1] = new String("-c");
            s3[2] = new String("kill " + kill_value);

            rt.exec(s3).waitFor();

            System.out.println("Killing      the      following      process:
"+kill_value);

        }
        else {
            System.out.println("tcpdump is not running");
            notDone = false;
        }
    }
    catch (Exception e){
        System.out.println("Error thrown while trying to read tcpdump
kill process id");
        notDone = false;
    }
}
}

```

```

/*****
    Method: read_file() - reads in data from the input file.
    - Input:
        String - file name.
    - Output:
        String - data from the file.
*****/

```

```

*****/

```

```

public String read_file(String file_name) {

    String s = new String();
    int file_size =0;
    byte[] buffer;

```

```

try {
    File f = new File(file_name);
    FileInputStream fis = new FileInputStream(f);
    file_size = fis.available();
    buffer = new byte[file_size];
    fis.read(buffer);
    s = new String(buffer);
    System.out.println("String read: /n <"+s+">");
}
catch (Exception e) {
    System.out.println("Error reading from file: " + file_name +
"/n"+ " Exception thrown: "+e);
}
return (s);
}

```

/*****

Method: **tear_down_connections()** - tears down existing
security association (SA) connections between peers.

- Input: none.
- Output: none.

*****/

```

public void tear_down_connection () {
    System.out.println("Tear_Down_Connection");
    calc_connection();
    read_connection_index_file();
    try {
        // assumption made that for every SA there are a pair
        int counter_temp = connection_count/2;

        if (connection_count == 0) {
            System.out.println("No current SAs exist");
        }
        else {
            for (int count = 1; count <=counter_temp; count++) {

```

```

        write_to_fifo();
        if (count != counter_temp) {
            connection_index++;
        }
        //write_to_fifo();
    }

}

File f = new File("/var/run/isakmpd.pid");

if (f.exists()) {

    FileInputStream fis = new FileInputStream(f);

    byte[] buffer = new byte[fis.available()];

    fis.read(buffer);

    fis.close();

    String tempString = new String();

    String kill_value = new String(buffer);

    String[] s3 = new String[3];
    s3[0]= new String("/bin/sh");
    s3[1] = new String("-c");
    s3[2] = new String("kill -HUP " + kill_value);

    rt.exec(s3).waitFor();

}

}

catch (Exception e) {
    System.out.println("Error in SPI count.");
    System.out.println("Exception thrown in tear_down_connection e:
"+e);
}

write_connection_index_file();

}

/*****

```

Method: write_to_fifo()- uses the current connection index to write teardown instructions to the IPsec mechanism in **/var/run/isakmpd.fifo** file.

- Input: none.
- Output: none.


```

*****/

public void write_to_fifo() {
    try {
        System.out.println("writing to fifo");
        File f = new File("/var/run/isakmpd.fifo");
        FileOutputStream fout = new FileOutputStream(f);
        PrintStream out = new PrintStream(fout);

        out.println("t Connection-"+connection_index);

        out.flush();

        out.close();
        fout.close();

    }

    catch (Exception e) {System.out.println("Exception thrown in
write_to_FiFO e: "+e);}

}

/*****

```

Method: **synchronized copy_kern_ipsec()** - copies the file /kern/ipsec (file containing the current security associations) to /root/demo/tempipsec (file used to parse security associations). This method is synchronized to avoid a deadlock when various threads, created by the demo, compete for this function.

- Input: none.
- Output: none.

```

*****/

```

```

public synchronized void copy_kern_ipsec() {
    try {
        pause(1);

```

```

        String[] s = new String[3];
        s[0]= new String("/bin/sh");
        s[1] = new String("-c");
        s[2] = new String("cp /kern/ipsec /root/demo/tempipsec");
        rt.exec(s).waitFor();
    }
    catch (Exception e) {
        System.out.println("in demo-support Exception thrown in
copying file e: "+e);
    }
}

```

/*****

Method: **calc_connection()**- calculates the number of existing security associations (SA) by reading **/root/demo/tempipsec** and parsing the information to count existing SAs.

- Input: none.
- Output: none.

*****/

```

public void calc_connection() {
    connection_count=0;

    try {
        copy_kern_ipsec();
        File f = new File("/root/demo/tempipsec");

        pause(1);

        if (f.exists()) {

            FileInputStream fis = new FileInputStream(f);

            byte[] buffer = new byte[fis.available()];

            fis.read(buffer);
            String tempString = new String();

            String record = new String(buffer);

            StringTokenizer st = new StringTokenizer(record);
            while (st.hasMoreTokens()) {

                tempString =st.nextToken();

                if (tempString.equals("SPI")) {

```

```

        // advance connection_counter
        connection_count++;
    }
} //
    fis.close();
}
else {
    System.out.println("/kern/ipsec File does not exist...");
}
}
catch (Exception e) { System.out.println("In Calc_Connections
Exception Thrown: "+e);}
}
}

```

APPENDIX F. SPFK.JAVA

```

/*****

    Class: SPFK.java - The goal of this class is to translate
    KeyNote's complex assertion format into an easy to
    understand syntax and display it in a JFrame.

*****/

* SPFK.java
*/

import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;

public class SPFK {
    private JFrame frame;
    private JTextArea textArea;
    private demo_support_functions dsf;
    private JButton exitButton;
    private JPanel contentPane, toolbar;

/*****

    Method: SPFK() - this method initializes the JFrame and
    reads in the security policy data from a file.

    - Input: none.
    - Output: none.

*****/

    public SPFK() {

        dsf = new demo_support_functions();

        textArea = new JTextArea(15, 40);
        JScrollPane scrollPane = new JScrollPane(textArea);
        contentPane = new JPanel(new BorderLayout());
        toolbar = new JPanel();
        toolbar.setSize(100,75);

        frame = new JFrame("Security Policy File / Keynote");
        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLL
        BAR_AS_NEEDED);

```

```

exitButton = new JButton("Exit");
exitButton.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        frame.dispose();
    }
});

toolbar.add(exitButton);
contentPane.add(scrollPane, BorderLayout.NORTH);
contentPane.add(toolbar, BorderLayout.SOUTH);
frame.getContentPane().add(contentPane);
frame.pack();
frame.setVisible(true);

String bigString = new String();

bigString = dsf.read_file("/etc/isakmpd/keynotednffinal.policy");

try {
    StringTokenizer policyST = new StringTokenizer(bigString,
    "|");
    while (policyST.hasMoreTokens()) {
        String policy = policyST.nextToken();
        showPolicy(policy);
    }
} catch (NoSuchElementException e) {
    System.out.println("SPFK.Error: " + e);
}
}

```

/*****

Method: showPolicy(String policy) - takes input string
 (security proposal), parses it into security attributes
 and dynamic parameters, and appends to the scrollable text
 area.

- Input:
 - String - security proposal to be parsed.
- Output: none.

*****/

```

private void showPolicy(String policy) throws
NoSuchElementException {

    String app_domain = null;
    String esp_auth_alg = null;
    String esp_enc_alg = null;
    String esp_present = null;
    String ah_auth_alg = null;

```

```

String ah_present = null;
String local_filter_port = null;
String network_mode = null;
String remote_filter_port = null;
String security_level = null;

StringTokenizer st = new StringTokenizer(policy);
// note that the string tokenizer cannot use hasMoreTokens()
// because the token delimiter changes during parsing.

System.out.println("Policy passed: "+policy);

while (st.countTokens() != 1) {
    String key = st.nextToken("\\()" & \\r\\n");
    String equal = st.nextToken("\\=");
    String value = st.nextToken("\\)");

        if (key.equals("app_domain"))                app_domain =
value;
        if (key.equals("esp_auth_alg"))                esp_auth_alg =
value;
        else if (key.equals("esp_enc_alg"))                esp_enc_alg =
value;
        else if (key.equals("esp_present"))                esp_present =
value;
        else if (key.equals("app_domain"))                app_domain =
value;
        else if (key.equals("local_filter_port"))
local_filter_port = value;
        else if (key.equals("network_mode"))                network_mode =
value;
        else if (key.equals("remote_filter_port"))
remote_filter_port = value;
        else if (key.equals("security_level"))                security_level
= value;
        else if (key.equals("ah_present"))                ah_present = value;
        else if (key.equals("ah_auth_alg"))                ah_auth_alg =
value;

        else System.out.println("SPFK.Unrecognized key: " + key);
    }

    textArea.append("-----
\\n");
    addText("\\n network_mode: ", network_mode);
    addText("\\n security_level: ", security_level);
    addText("\\n esp_present: ", esp_present);
    addText("\\n ah_present: ", ah_present);

    addText("\\n esp_enc_alg: ", esp_enc_alg);
    addText("\\n esp_auth_alg: ", esp_auth_alg);
    addText("\\n ah_auth_alg: ", ah_auth_alg);

    addText("\\n local_filter_port: ", local_filter_port);

    addText("\\n remote_filter_port: ", remote_filter_port);

```

```

        textArea.append("\n\n");
    }

/*****

        Method: addText((String, String) - takes input strings and
        appends them to the scrollable text area.
    - Input:
        - String - tag.
        - String - tag value.
    - Output: none.

*****/

    private void addText(String description, String value) {
        if (value != null) {
            textArea.append(description + value);
        }
    }
}

```

APPENDIX G. DP_CONSOLE.JAVA

```
/******  
  
    Class: dp_console.java - The dynamic parameter console  
    provides the user with a selection mechanism for network  
    mode and security level.  
  
*****/  
  
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
import java.io.*;  
import java.util.*;  
import java.lang.*;  
import demo_support_functions;  
  
public class dp_console extends JFrame {  
  
    private JFrame actionframe;  
  
    private JButton submit, exit;  
    private ButtonGroup networkmode, securitylevel;  
    private JRadioButton nm_normal, nm_crisis, nm_impact, sl_high,  
sl_medium, sl_low;  
    private JPanel panel, p, pdpbuttons,  
p2, ptitle, pcomments, pactionbuttons;  
    private JLabel sl_title, nm_title, comment, blank1, blank2, blank3;  
  
    String current_network_mode, current_security_level;  
    private static Runtime rt;  
    private demo_support_functions dsf;  
  
/******  
  
    Method: dp_console() constructor - initializes the dynamic  
    parameter selection interface.  
  
*****/  
  
public dp_console() {  
  
    super ("Dynamic Parameter Selection Window");
```



```

addWindowListener(
    new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {
            //if (input != null)
            //    closeFile();

            dispose();
        }
    }
);

setDefaultCloseOperation(DISPOSE_ON_CLOSE);

p = new JPanel();
p2 = new JPanel();
ptitle = new JPanel();
pcomments = new JPanel();
pdpbuttons = new JPanel();
pactionbuttons = new JPanel();
p.setLayout(new FlowLayout());
ptitle.setLayout(new GridLayout(1,2));
pcomments.setLayout(new GridLayout(0,1));
pdpbuttons.setLayout(new GridLayout(4,2));
actionframe = new JFrame();
pactionbuttons.setLayout(new FlowLayout());
setContentPane(p);

current_network_mode = null;
current_security_level = null;

comment = new JLabel("

");

blank1 = new JLabel();
blank2 = new JLabel();
blank3 = new JLabel();

sl_low = new JRadioButton ("Low", false);
sl_medium = new JRadioButton ("Medium", false);
sl_high = new JRadioButton ("High", false);

nm_normal = new JRadioButton ("Normal", false);
nm_crisis = new JRadioButton ("Crisis", false);
nm_impact = new JRadioButton ("Impacted", false);

nm_title = new JLabel("Network Modes Selection");
sl_title = new JLabel("Security Levels Selection");

networkmode = new ButtonGroup();
securitylevel = new ButtonGroup();

submit = new JButton("Submit");
exit = new JButton("Exit");

networkmode.add(nm_normal);

```

```

networkmode.add(nm_crisis);
networkmode.add(nm_impact);

securitylevel.add(sl_low);
securitylevel.add(sl_medium);
securitylevel.add(sl_high);

NMRadioButtonHandler nm_handler = new NMRadioButtonHandler();
SLRadioButtonHandler sl_handler = new SLRadioButtonHandler();
SubmitButtonHandler sb_handler = new SubmitButtonHandler();
ExitButtonHandler eb_handler = new ExitButtonHandler();

submit.addActionListener(sb_handler);
exit.addActionListener(eb_handler);

nm_normal.addItemListener(nm_handler);
nm_crisis.addItemListener(nm_handler);
nm_impact.addItemListener(nm_handler);

sl_high.addItemListener(sl_handler);
sl_medium.addItemListener(sl_handler);
sl_low.addItemListener(sl_handler);

submit.setVisible(true);
exit.setVisible(true);

blank1.setVisible(false);
blank2.setVisible(false);
blank3.setVisible(false);

pdpbuttons.add(nm_title);
pdpbuttons.add(sl_title);

pdpbuttons.add(nm_normal);
pdpbuttons.add(sl_low);
pdpbuttons.add(nm_crisis);
pdpbuttons.add(sl_medium);
pdpbuttons.add(nm_impact);
pdpbuttons.add(sl_high);
p.add(pdpbuttons);
pcomments.add(comment);
p.add(pcomments);

pactionbuttons.add(blank1);
pactionbuttons.add(submit);
pactionbuttons.add(blank2);
pactionbuttons.add(exit);
pactionbuttons.add(blank3);
p.add(pactionbuttons);

setSize(350, 150);
setResizable(true);
setVisible (false);

dsf = new demo_support_functions();
rt = Runtime.getRuntime();
}

```

```
/******
```

Method: **start_dp_console()** - makes the **dp_console** visible.

- Input: none.
- Output: none.

```
*****/
```

```
public void start_dp_console() {
```

```
    setVisible (true);
```

```
}
```

```
/******
```

Method: **reset_error_panel()** - clears error message panel.

- Input: none.
- Output: none.

```
*****/
```

```
public void reset_error_panel() {
```

```
    comment.setText("");
```

```
}
```

```
/******
```

Method: **set_dynamic_parameters()** - stores the value of the dynamic parameters in a file.

- Input: none.
- Output: none.

```
*****/
```

```
public void set_dynamic_parameters() {
```

```
    delete_file();  
    write_dynamic_parameters_file();
```

```
}
```

```

/*****

    Method: write_dynamic_parameters_file - writes the global
    current value of the dynamic parameters to
    /usr/src/sbin/isakmpd/dynamic_parameters.
    - Input: none.
    - Output: none.

*****/

```

```

public void write_dynamic_parameters_file() {
    try {
        File f = new
File("/usr/src/sbin/isakmpd/dynamic_parameters");
        FileOutputStream fos = new FileOutputStream(f);
        PrintStream out = new PrintStream(fos);
        out.println("network_mode = " + current_network_mode+ "\n");
        out.println("security_level = " + current_security_level +
"\n");
        fos.close();
        out.close();
    }
    catch (Exception e) {
        System.out.println("Execption Thrown in Write_DP e: " +e);
    }
}

```

```

/*****

    Method: print_dynamic_parameters_file() - displays the
    current value of network mode and security level to the
    system console for trouble shooting purposes.
    - Input: none.
    - Output: none.

*****/

```

```

public void print_dynamic_parameters_file() {

    try {
        File f = new File("/usr/src/sbin/isakmpd/dynamic_parameters");

        FileInputStream fis = new FileInputStream(f);

        int size_before = fis.available();

        System.out.println("file size: " + size_before);

        byte[] buffer = new byte[fis.available()];

        fis.read(buffer);

        String tempString = new String();

        String record = new String(buffer);

        System.out.println("File : " + record);

        fis.close();

    }
    catch (Exception e) {
        System.out.println("Exception throwm e: "+e);}
}

/*****

    Method: read_dynamic_parameters_file() - reads in the
    value of network mode and security level from file:
    /usr/src/sbin/isakmpd/dynamic_parameters and stores them in
    the class global variables respectively.
    - Input: none.
    - Output: none.

*****/

```

```

*****/

public void read_dynamic_parameters_file() {

    System.out.println("Reading dynamic parameters");

    try {

        File f = new File("/usr/src/sbin/isakmpd/dynamic_parameters");

        if (f.exists()) {

```

```

        System.out.println("dynamic_parameters File exists...");
        FileInputStream fis = new FileInputStream(f);
        byte[] buffer = new byte[fis.available()];
        fis.read(buffer);
        String tempString = new String();
        String record = new String(buffer);
        StringTokenizer st = new StringTokenizer(record);
        while (st.hasMoreTokens()) {
            tempString =st.nextToken();
            if (tempString.equals("network_mode")) {
                // skip equals sign
                st.nextToken();
                current_network_mode = st.nextToken();
            }

            else if (tempString.equals("security_level")) {
                // skip equals sign
                st.nextToken();
                current_security_level = st.nextToken();
            }
        }
        fis.close();
    }
    else {
        System.out.println("dynamic parameters File does not
exist...");
    }
}
catch (Exception e) { System.out.println("Exception Thrown reading
dp file: "+e);}
}

```

/*****

Method: **delete_file()** - deletes existing dynamic parameter file.

- Input: none.
- Output: none.

```

*****/

public void delete_file() {

    try {

        File f = new
File("/usr/src/sbin/isakmpd/dynamic_parameters");
        f.delete();
        System.out.println("deleting dynamic_parameters file.");
    }
    catch (Exception e) {
        System.out.println("Exception thrown while trying to delete
dynamic_parameters file. exception: "+e);}

    }
}

```

```

/*****

```

Class: SLRadioButtonHandler implements ItemListener -
security level radio button action handler

```

*****/

```

```

private class SLRadioButtonHandler implements ItemListener {

    public void itemStateChanged(ItemEvent e)
    {

        reset_error_panel();

        if (current_security_level == null) {

            current_security_level = new String();
        }

        System.out.println("in SL Radio Button item listener");
        if (e.getSource() == sl_high) {

            System.out.println("Security Level High selected.");
            current_security_level = "high";

        }

        else if (e.getSource() == sl_medium) {

            System.out.println("Security Level medium selected.");
            current_security_level= "medium";

        }

        else if (e.getSource() == sl_low) {

```

```

        System.out.println("Security Level low selected.");
        current_security_level = "low";
    }
}

/*****

        Class: SubmitButtonHandler implements ActionListener -
        submit button action handler.

*****/

private class SubmitButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e)
    {
        try {
            reset_error_panel();
            System.out.println("Submit Button selected");

            if
((current_network_mode==null)&&(current_security_level==null)){
                comment.setText("You must select a network mode and
security level");
            }

            else if (current_network_mode == null){
                comment.setText("You must select a network mode");
            }
            else if (current_security_level== null) {
                comment.setText("You must select a security level.");
            }
            else {
                comment.setText("

");

                set_dynamic_parameters();
                print_dynamic_parameters_file();

                if (dsf.daemon_running()) {
                    System.out.println("daemon is running.");

                    dsf.copy_kern_ipsec();
                    dsf.tear_down_connection();

```



```

        dsf.flush_ipsec();
        dsf.load_spd();
    }
    else { // daemon not running....

        comment.setText("Please start IPsec mechanism from
Qoss menu");
    }

}
}
catch (Exception e2) {
    System.out.println("Exception Thrown in Submit Button
Handler.");
    System.out.println("Exception: "+e2);
}
}
}

```

/*****

Class: **ExitButtonHandler** implements ActionListener - **Exit**
 button action handler.

*****/

```

private class ExitButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e)
    {
        reset_error_panel();

        System.out.println("Close Button Selected");

        dispose();
    }
}

```

/*****

Class: **NMRadioButtonHandler** implements ItemListener -
 network mode radio button action handler

*****/

```

private class NMRadioButtonHandler implements ItemListener {

```

```

public void itemStateChanged(ItemEvent e)
{
    reset_error_panel();
    if (current_network_mode == null) {

        current_network_mode = new String();
    }

    if (e.getSource() == nm_normal){

        System.out.println("Network Mode Normal selected.");
        current_network_mode = "normal";

    }

    else if (e.getSource() == nm_crisis){

        System.out.println("Network Mode Crisis selected.");
        current_network_mode = "crisis";

    }

    else if (e.getSource() == nm_impact){

        System.out.println("Network Mode Impacted selected.");
        current_network_mode = "impacted";

    }
}
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. IPSECINFO.JAVA

```
/******  
  
    Class: ipsecinfo.java - provides a display mechanism for  
    the Security Association Database (SAD).  
  
*****/  
  
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
import java.io.*;  
import java.util.*;  
  
public class ipsecinfo extends Thread {  
    File f2 = new File("/root/demo/tempipsec");  
    private static Runtime rt;  
    private JFrame j;  
    private JTextField[] connected_label, unconnected_label ;  
    public JTextField[] destination, source, protocol, enc_alg,  
enc_auth, auth_alg;  
    private JTextField[] title, dest_label, src_label, protocol_label,  
enc_alg_label, enc_auth_label, auth_alg_label, status, blank1, blank2;  
    private JTextField blank_field;  
    //private Container c;  
    private ObjectInputStream input;  
    private BorderLayout layout;  
    private GridLayout gl;  
    private JPanel[] jp1, jp2, jp3, jp4, jp5, jp6, jp7, blank;  
    private JPanel p;  
    private String spi[];  
    private JPanel toolbar;  
    private JPanel[] pvec;  
    private JButton exit;  
    private int SA_number =0;  
    private JPanel contentPane;  
    boolean first_time = true;  
    boolean test_catch = false;  
    demo_support_functions dsf = new demo_support_functions();  
    private boolean kill_thread = false;  
  
/******  
  
    Method: ipsecinfo() - class constructor. Initializes JFrame.  
  
*****/  
  
public ipsecinfo() {
```

```

System.out.println("Test1..");

rt = Runtime.getRuntime();

j = new JFrame("OpenBSD IPsec Status Window");

j.addWindowListener(
    new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {
            j.dispose();
            kill_thread = true;
        }
    }
);

toolbar = new JPanel();
exit = new JButton("Exit");
exit.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {

        j.dispose();
        kill_thread = true;

    }
});
toolbar.add(exit);
p = new JPanel();
p.setLayout(new GridLayout(0, 1));
JPanel contentPane = new JPanel();
contentPane.setLayout(new BorderLayout());
contentPane.add(p, BorderLayout.NORTH);
contentPane.add(toolbar, BorderLayout.SOUTH);
j.setContentPane(contentPane);

System.out.println("Test2..");

JPanel emptyPanel = new JPanel();

JTextField emptyText = new JTextField("No current security
association (SA) established");
emptyText.setEditable(false);
emptyPanel.add(emptyText);
p.add(emptyPanel);
j.pack();
j.setResizable(true);
j.setVisible(true);

}

/*****

```

- Method: **frame_initialization()** - initializes the JFrame to repaint new SAs.

- Input: none.
- Output: none.

*****/

```
public void frame_initialization() {

    int number = 10;
    // JFrame Initiization

    pvec = new JPanel[number];
    jp1 = new JPanel[number];
    jp2 = new JPanel[number];
    jp3 = new JPanel[number];
    jp4 = new JPanel[number];
    jp5 = new JPanel[number];
    jp6 = new JPanel[number];
    jp7 = new JPanel[number];
    blank = new JPanel[number];
    spi = new String[number];

    dest_label = new JTextField[number];
    src_label= new JTextField[number];
    protocol_label = new JTextField[number];
    enc_alg_label = new JTextField[number];
    enc_auth_label = new JTextField[number];
    auth_alg_label = new JTextField[number];
    status = new JTextField[number];
    connected_label= new JTextField[number];
    unconnected_label = new JTextField[number];

    destination = new JTextField[number];
    source = new JTextField[number];
    protocol = new JTextField[number];
    enc_alg = new JTextField[number];
    enc_auth = new JTextField[number];
    auth_alg = new JTextField[number];
    blank1 = new JTextField[number];
    blank2 = new JTextField[number];

    SA_number=0;

    p.removeAll();

    p.setLayout(new GridLayout(0, 1));

}
```

/*****

Method: preliminary_test(StringTokenizer st) - compares the current SA's in the JFrame with the SA's from the new file.

This is performed to avoid unnecessary painting and maintain good display resolution.

- Input:

- StringTokenizer st - contents of the file to be verified.

- Output:

- Boolean - True if string contains new SA's.
False otherwise.

*****/

```
boolean preliminarary_test(StringTokenizer st) {
```

```
    boolean found_match = false;
    boolean repaint = false;
    String tempString = new String();
    StringTokenizer s = st;
    int spi_count = 0;
```

```
    while (s.hasMoreTokens()) {
        tempString = s.nextToken();
        if (tempString.equals("SPI"))
            spi_count++;
    }
    if (spi_count != SA_number) {
        repaint = true;
    }
    else {
```

```
        while (st.hasMoreTokens()) {
```

```
            tempString = st.nextToken();
            if (tempString.equals("SPI")) {
                found_match = false;
```

```
                // advance token passed assignment
                tempString = st.nextToken();
                tempString = st.nextToken();
                for (int c=0; c<SA_number; c++) {
```

```
                    if (spi[c].equals(tempString)) {
                        found_match = true;
```

```
                        break;
```

```
                    }
                }
```

```
                // if match not found after searching array...repaint
```

```
needed.
```

```

        if (!found_match) {
            return(true);
        }

    }

}

return (repaint);

}

/*****

    Method: String wait_for_full_copy(String record)- verifies
    that all the required tags exist in the string prior to
    parsing. If not, file is reread and the string is verified
    until all the tags are found.
    - Input:
        - String record - contents of the file to be
          verified.
    - Output:
        - String - String that contains all required fields.

*****/

public String wait_for_full_copy(String record) {

    StringTokenizer st = new StringTokenizer(record);
    String tempString = new String();
    boolean redo = false;
    boolean keep_looking = true;
    boolean not_found = true;
    boolean check1 = false;
    boolean check2 = false;
    boolean check3 = false;
    FileInputStream in2;
    byte[] buffer;
    int length;

    try {

        while (st.hasMoreTokens() && keep_looking) {

            tempString =st.nextToken();

            if (tempString.equals("SPI")) {

                not_found = true;
                check1 = false;

```



```

check2 = false;
check3 = false;

while (st.hasMoreTokens() && not_found) {
    tempString = st.nextToken();
    if (tempString.equals("Destination")) {
        check1 = true;
    }
    else if (tempString.equals("Source")){
        check2 = true;
    }

    else if (tempString.equals("xform")) {
        check3 = true;
    }

    if (check1 && check2 && check3)
        not_found = false;
    if (tempString.equals("SPI")){
        not_found = false;
        redo = true;
    }
}

if (redo) {
    try {

        long pauseTime = (long) (1*1000);
        Thread.sleep(pauseTime);
    }
    catch (Exception etp) {
        System.out.print("Run Error: " + etp);
    }

    System.out.println("in wait...");
    dsf.copy_kern_ipsec();

    try {

        in2 = new FileInputStream(f2);
        buffer = new byte[in2.available()];
        in2.read(buffer,0,buffer.length);
        record = new String(buffer);
        st = new StringTokenizer(record);

        in2.close();

    }

    catch (Exception e) {
        System.out.print("in ipsecinfo wait for full copy Run
Error: " + e);

```

```

        }
        redo = false;
    }

}

}
catch (Exception e) {
    System.out.print("Run Error: " + e);
}
return (record);
}

```

/******

Method: **parse(String record)** - parses the string into SA's
to be displayed on the JFrame.

- Input:

- String record - contains the file to be parsed.

- Output: none.

*****/

```

public void parse(String record)
{
    String tempString = new String();

    boolean esp = false;
    boolean ah = false;
    boolean prelim_test = false;
    int count = -1;
    int file_contents_check = 0;

    try {

        boolean no_SA_flag = true;

        record = wait_for_full_copy(record);
        StringTokenizer st = new StringTokenizer(record);
        StringTokenizer temp_st = new StringTokenizer(record);

        // if preliminary test passed.... continue parsing...

        if (preliminary_test(temp_st)) {

            frame_initialization();
            test_catch= true;

```

```

while (st.hasMoreTokens()) {

    tempString =st.nextToken();

    if (tempString.equals("SPI")) {
        //skip =
        st.nextToken();
        no_SA_flag=false;

        if (count != -1) {

            //setup blank line
            blank[count] = new JPanel(new GridLayout(1,2));
            blank1[count] = new JTextField(" ");
            blank2[count] = new JTextField(" ");
            blank1[count].setEditable(false);
            blank2[count].setEditable(false);
            blank[count].add(blank1[count]);
            blank[count].add(blank2[count]);
            blank[count].setVisible(true);

            p.add(blank[count]);
        }

        count = count + 1;

        spi[count]= (String)st.nextToken();

        esp = false;
        ah = false;
    }

    else if (tempString.equals("Destination")) {

        st.nextToken();

        // setup Destination panel
        jp1[count] = new JPanel(new GridLayout(1,2));

        dest_label[count] = new JTextField ("Destination: ");
        destination[count] = new JTextField();
        dest_label[count].setEditable(false);
        destination[count].setEditable(false);
        jp1[count].add(dest_label[count]);
        jp1[count].add(destination[count]);
        destination[count].setText((String) st.nextToken());
        jp1[count].setVisible(true);
        p.add(jp1[count]);

    }

    else if (tempString.equals("Source")) {

        st.nextToken();

        // setup source panel

```

```

        jp2[count] = new JPanel(new GridLayout(1,2));
        src_label[count]= new JTextField ("Source: ");
        source[count] = new JTextField();
        src_label[count].setEditable(false);
        source[count].setEditable(false);
        jp2[count].add(src_label[count]);
        jp2[count].add(source[count]);

        source[count].setText((String) st.nextToken());
        jp2[count].setVisible(true);
        p.add(jp2[count]);

        //setup status panel
        jp3[count] = new JPanel(new GridLayout(1,2));
        status[count] = new JTextField ("Connection Status:
");
        connected_label[count] = new JTextField ("IPsec
connection established");
        status[count].setEditable(false);
        connected_label[count].setEditable(false);
        jp3[count].add(status[count]);
        jp3[count].add(connected_label[count]);
        jp3[count].setVisible(true);
        p.add(jp3[count]);

    }

    else if (tempString.equals("xform")) {
        st.nextToken();
        tempString =(((String) st.nextToken()) + " " +
((String) st.nextToken()));

        //setup Protocolpanel
        jp4[count] = new JPanel(new GridLayout(1,2));
        protocol_label[count] = new JTextField ("Protocol:
");

        protocol[count] = new JTextField();
        protocol_label[count].setEditable(false);
        protocol[count].setEditable(false);
        jp4[count].add(protocol_label[count]);
        jp4[count].add(protocol[count]);

        protocol[count].setText(tempString);
        jp4[count].setVisible(true);
        p.add(jp4[count]);

        if (tempString.equals("<IPsec ESP>")) {
            esp = true;
        }
        else { System.out.println("Protocol: "+tempString);}
        if (tempString.equals("<IPsec AH>")) {
            System.out.println("IPsec AH....");
            ah = true;
        }
    }
}

```

```

else if (tempString.equals("Encryption")) {

    st.nextToken();
    tempString = st.nextToken();

    //setup encryption algo panel
    jp5[count] = new JPanel(new GridLayout(1,2));
    enc_alg_label[count] = new JTextField ("Encryption
Algorithm: ");
    enc_alg[count] = new JTextField();
    enc_alg_label[count].setEditable(false);
    enc_alg[count].setEditable(false);
    jp5[count].add(enc_alg_label[count]);
    jp5[count].add(enc_alg[count]);

    enc_alg[count].setText(tempString);
    jp5[count].setVisible(true);
    p.add(jp5[count]);

}

else if (tempString.equals("Authentication")) {

    st.nextToken();
    tempString = st.nextToken();

    if (esp) {
        //setup esp authentication panel
        jp6[count] = new JPanel(new GridLayout(1,2));
        enc_auth_label[count] = new JTextField
("Encryption Authentication Algorithm: ");
        enc_auth[count] = new JTextField();
        enc_auth_label[count].setEditable(false);
        enc_auth[count].setEditable(false);
        jp6[count].add(enc_auth_label[count]);
        jp6[count].add(enc_auth[count]);

        enc_auth[count].setText(tempString);
        jp6[count].setVisible(true);
        p.add(jp6[count]);
        //esp = false;

    }
    else if (ah) {

        //setup ah authentication panel
        jp7[count] = new JPanel(new GridLayout(1,2));
        auth_alg_label[count] = new JTextField
("Authentication Algorithm: ");
        auth_alg[count] = new JTextField();
        auth_alg_label[count].setEditable(false);
        auth_alg[count].setEditable(false);
        jp7[count].add(auth_alg_label[count]);
        jp7[count].add(auth_alg[count]);
        auth_alg[count].setText(tempString);
        jp7[count].setVisible(true);
        p.add(jp7[count]);
    }
}

```

```

        //ah = false;

    }

}

}

//Check for no SA's

if (no_SA_flag) {
    JPanel emptyPanel = new JPanel();
    JTextField emptyText = new JTextField("No current security
association (SA) established");
    emptyText.setEditable(false);
    emptyPanel.add(emptyText);
    p.add(emptyPanel);
    j.pack();
}
else {
    // increment by 1 from array adjustment
    count = count +1;
    // setup JFrame size
    if (count < 2) j.setSize(300, 300);
    else if (count < 3) j.setSize(400,300);
    else if (count < 4) j.setSize(400,400);
    else if (count < 5) j.setSize(400,500);
    else j.setSize(600,600);
}

test_catch = true;
SA_number = count;

}
else {
    System.out.println("Preliminary      Test      failed...skipping
parsing..");}

}

catch (Exception e) {System.out.println("Erooring in parsing: "
+e);}

}

}

/*****

```

- Method: run() – continually checks files date-time-stamp and if file is updated copies file and reads data into a string.

- Input: none.
- Output: none.

*****/

```

public void run()
{
    int current_file_size=0;
    int new_file_size= 0;
    long old_date_time_stamp = 0;
    long new_date_time_stamp = 0;
    int file_size=0;

    FileInputStream in1,in2;
    byte[] buffer;
    byte[] buffertemp;
    String record;
    File f1; //,f2;
    try {
        f1 = new File("/kern/ipsec");
    }
    catch (Exception e) { System.out.println("Error: "+e);}

    while (!kill_thread) {

        try {

            long pauseTime = (long) (1*1000);
            Thread.sleep(pauseTime);
        }
        catch (Exception etp) {
            System.out.print("Run Error: " + etp);
        }

        try {

            if (f1.exists()) {

                in1 = new FileInputStream(f1);
                new_file_size = in1.available();

                new_date_time_stamp = f1.lastModified();

                if (new_date_time_stamp != old_date_time_stamp) {

                    in2 = new FileInputStream(f2);
                    //copy file
                    dsf.copy_kern_ipsec();
                    file_size = in2.available();

                    buffer = new byte[in2.available()];
                    buffertemp = new byte[in2.available()];

                    in2.read(buffer,0,buffer.length);

```

```

        old_date_time_stamp = new_date_time_stamp;
        record = new String(buffer);
        in2.close();
        parse(record);
    }
    in1.close();
}

}

catch (Exception e) {
    System.out.print("Run Error: " + e);
}

}
}

```


THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. TCPDUMP.JAVA

```

/*****

        Class: tcpdump.java - provide the user with graphical
        console display of tcpdump.

*****/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

public class tcpdump extends Thread {

    private static Runtime rt;
    private JFrame j;
    private JScrollPane jsp;
    private JLabel jl2;
    private JTextArea jta;
    private JPanel toolbar;
    private JPanel contentPane;
    private ObjectInputStream input;
    private BorderLayout layout;
    private GridLayout gl;
    private JButton exit;
    private demo_support_functions dsf;
    private InputStream in;
    private Process p;
    private boolean stop = false;

/*****

Method: tcpdump() - constructor for the class.

*****/

public tcpdump() {
    dsf = new demo_support_functions();

    System.out.println("in constructo");

    rt = Runtime.getRuntime();

    jl2 = new JLabel();
    jta = new JTextArea(10,70);

```

```

jta.setLineWrap(true);
jta.setRows(10);
contentPane = new JPanel();
toolbar = new JPanel();

exit = new JButton("Exit");
exit.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        stop = true;
        dsf.stop_tcpdump();
        j.dispose();
    }
});

j = new JFrame("OpenBSD IPsec TCPDUMP View Window");

j.addWindowListener(
    new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {
            stop = true;
            dsf.stop_tcpdump();
            j.dispose();
        }
    }
);

toolbar.add(exit);
jsp = new JScrollPane(jta);
jsp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_
NEEDED);
contentPane.setLayout(new BorderLayout());
//contentPane.setPreferredSize(new Dimension(600,250));
contentPane.add(jsp, BorderLayout.NORTH);
contentPane.add(toolbar, BorderLayout.SOUTH);
toolbar.setSize(600,50);
j.setContentPane(contentPane);
j.pack();
j.setVisible(true);
}

```

/*****

Method: **start_tcpdump()** - executes **tcpdump** and creates a pipe to capture packet information.

- Input: none.
- Output: none.

*****/

```

public void start_tcpdump() {

```

```

    try {
        System.out.println("in try for rt...");
        String[] s = new String[3];
        s[0]= new String("/bin/csh");
        s[1] = new String("-c");
        s[2] = new String("tcpdump -N -v");
        p=rt.exec(s);
        in = p.getInputStream();
        System.out.println("performed rt.");
    }
    catch (Exception e) {
        System.out.println("Exception thrown in tcpdump"+e);
    }
}

}

/*****

    Method: repaint_frame() - this method generates the output
    from the piped stream and displays to the scrollable text
    area.
        - Input: none.
    - Output: none.

*****/

public void  repaint_frame() {
    String line;

    try {
        System.out.println("in repaint_frame");
        BufferedReader br = new BufferedReader(new
InputStreamReader(in));

        jta.setText("TCPDUMP Output....");

        while (stop == false) {
            try {

                line = br.readLine();

                StringTokenizer token = new StringTokenizer(line, ">
:.\n\r");

                boolean found = false;
                while (token.hasMoreTokens()) {
                    String tok = token.nextToken();
                    if (tok.equals("athina") || tok.equals("mshn3")) {
                        found = true;
                    }
                }
            }
        }
    }
}

```

```

        if (found) {
            System.out.println("here: "+line);
            jta.append("..ok..\n");
            jta.append(line);
        }
    }
    catch (Exception e2) {
        System.out.println("Exception thrown try to buffer read
for tcpdump: "+e2);}

    }
}
catch (Exception e) {
    System.out.println("Error reading tcpdump file: "+e);
}

}

/*****

```

Method: run() - run method for the thread.

```

*****/

public void run()
{
    start_tcpdump();
    repaint_frame();
}

}

```

APPENDIX J. ISAKMPD.CONF FILE

The following is the **isakmpd.conf** file used in the testing phase.

Note: only fields required after modifications to IPsec/IKE/ISAKMPD mechanism were those related to network connections (IP Addresses and Net masks), ISAKMP Phase I and certificates. All ISAKMP Phase II was derived from **isakmpd.policy/KeyNote**.

```
# A configuration sample for the isakmpd ISAKMP/Oakley (aka IKE)
daemon.

[General]
Listen-on=          131.120.8.91
Shared-SADB=        Defined
Retransmits=        5
Exchange-max-time=  120

[Phase 1]
#set-up to work specifically with athina with new configuration style
131.120.8.95=        Peer-131.120.8.95/131.120.8.91

#set-up to work specifically with athina with new configuration style
[Peer-131.120.8.95/131.120.8.91]
Phase=              1
Address=             131.120.8.95
Local-address=       131.120.8.91
Transport=           udp
Configuration=       Default-main-mode
Authentication=       mekmitasdigoat

#Must stay - by CDA
[Default-main-mode]
DOI=                 IPSEC
EXCHANGE_TYPE=       ID_PROT
Transforms=          3DES-SHA

# Certificates stored in PEM format
[X509-certificates]
CA-directory=        /etc/isakmpd/ca/
Cert-directory=       /etc/isakmpd/certs/
#Accept-self-signed= defined
Private-key=         /etc/isakmpd/private/MSHN3.key
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K. ISAKMPD.POLICY FILE

The following is the **isakmpd.policy** file used in the testing phase.

Note: Refer to Appendix M Table M.1 for summary of security proposals.

```
KeyNote-Version: 2
Comment: Policy file for Network Modes and Security Levels
Authorizer: "POLICY"
Licensees: "passphrase:mekmitasdigoat"
Conditions: ( (app_domain == "IPsec policy") &&
  (
    ( (network_mode == "normal") &&
      (
        ( (security_level == "low") &&
          (
            ( (esp_present == "yes") &&
              ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
                (esp_enc_alg == "des") &&
                (esp_auth_alg == "hmac-md5")
            )
            ||
            ( (ah_present == "yes") &&
              ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
                (ah_auth_alg == "hmac-md5")
            )
          )
        )
      )
    )
    ||
    ( (security_level == "medium") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
                (esp_enc_alg == "cast") &&
                (esp_auth_alg == "hmac-sha")
            )
            ||
            ( (ah_present == "yes") &&
              ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
                (ah_auth_alg == "hmac-md5")
            )
          )
        )
      )
    )
    ||
    ( (security_level == "high") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
                (esp_enc_alg == "3des") &&
                (esp_auth_alg == "hmac-sha")
            )
          )
        )
      )
    )
  )
)
```



```

        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
            (ah_auth_alg == "hmac-sha")
          )
        )
      )
    )
  )
  ||
  ( (network_mode == "impacted") &&
    (
      ( (security_level == "low") &&
        (
          ( (esp_present == "yes") &&
            ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
              (esp_enc_alg == "des") &&
              (esp_auth_alg == "hmac-md5")
            )
          ||
          ( (ah_present == "yes") &&
            ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
              (ah_auth_alg == "hmac-md5")
            )
          )
        )
      )
    )
    ||
    ( (security_level == "medium") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
            (esp_enc_alg == "des") &&
            (esp_auth_alg == "hmac-md5")
          )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
            (ah_auth_alg == "hmac-md5")
          )
        )
      )
    )
    ||
    ( (security_level == "high") &&
      (
        ( (esp_present == "yes") &&
          ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
            (esp_enc_alg == "3des") &&
            (esp_auth_alg == "hmac-md5")
          )
        )
      )
    )
  )

```

```

        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-sha")
        )
      )
    )
  )
||
  ( (network_mode == "crisis") &&
    (
      ( (security_level == "low") &&
        (
          ( (esp_present == "yes") &&
            ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
            (esp_enc_alg == "3des") &&
            (esp_auth_alg == "hmac-sha")
          )
        )
        ||
        ( (ah_present == "yes") &&
          ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
          (ah_auth_alg == "hmac-sha")
        )
      )
    )
  )
||
  ( (security_level == "medium") &&
    (
      ( (esp_present == "yes") &&
        ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
        (esp_enc_alg == "3des") &&
        (esp_auth_alg == "hmac-sha")
      )
      ||
      ( (ah_present == "yes") &&
        ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
        (ah_auth_alg == "hmac-sha")
      )
    )
  )
||
  ( (security_level == "high") &&
    (
      ( (esp_present == "yes") &&
        ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
        (esp_enc_alg == "aes") &&
        (esp_auth_alg == "hmac-sha")
      )
      ||
      ( (ah_present == "yes") &&
        ( (local_filter_port == "79") ||

```

```
(remote_filter_port == "79") ) &&
    (ah_auth_alg == "hmac-sha")
    )
    )
    )
)
||
( (network_mode == "default") &&
  (security_level == "default") &&
  (
    ( esp_present == "yes") &&
    ( (local_filter_port == "23") ||
(remote_filter_port == "23") ) &&
      (esp_enc_alg == "des") &&
      (esp_auth_alg == "hmac-md5")
    )
    ||
    ( (ah_present == "yes") &&
      ( (local_filter_port == "79") ||
(remote_filter_port == "79") ) &&
        (ah_auth_alg == "hmac-md5")
      )
    )
  )
)
)
)
-> "true";
```

APPENDIX L. KEYNOTEDNFFINAL.POLICY FILE

The following file is the **keynotednffinal.policy** file used in the testing phase. The file is generated by the DNF module which converts the condition assertion found in **inisakmpd.policy** into a DNF form.

Note: Refer to Appendix M Table M.1 for summary of security proposals.

```
(((((local_filter_port == "23") && (esp_auth_alg == "hmac-sha")) && (esp_enc_alg == "3des")) && (esp_present == "yes")) && (security_level == "high")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "23") && (esp_auth_alg == "hmac-sha")) && (esp_enc_alg == "3des")) && (esp_present == "yes")) && (security_level == "high")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "79") && (ah_auth_alg == "hmac-sha")) && (ah_present == "yes")) && (security_level == "high")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "79") && (ah_auth_alg == "hmac-sha")) && (ah_present == "yes")) && (security_level == "high")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "23") && (esp_auth_alg == "hmac-sha")) && (esp_enc_alg == "cast")) && (esp_present == "yes")) && (security_level == "medium")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "23") && (esp_auth_alg == "hmac-sha")) && (esp_enc_alg == "cast")) && (esp_present == "yes")) && (security_level == "medium")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "79") && (ah_auth_alg == "hmac-md5")) && (ah_present == "yes")) && (security_level == "medium")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "79") && (ah_auth_alg == "hmac-md5")) && (ah_present == "yes")) && (security_level == "medium")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "23") && (esp_auth_alg == "hmac-md5")) && (esp_enc_alg == "des")) && (esp_present == "yes")) && (security_level == "low")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "23") && (esp_auth_alg == "hmac-md5")) && (esp_enc_alg == "des")) && (esp_present == "yes")) && (security_level == "low")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "79") && (ah_auth_alg == "hmac-md5")) && (ah_present == "yes")) && (security_level == "low")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((remote_filter_port == "79") && (ah_auth_alg == "hmac-md5")) && (ah_present == "yes")) && (security_level == "low")) && (network_mode == "normal")) && (app_domain == "IPsec policy")) || (((local_filter_port == "23") && (esp_present == "yes")) && (esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) && (network_mode == "default")) && (security_level == "default")) || (((remote_filter_port == "23") && (esp_present == "yes")) && (esp_enc_alg == "des")) && (esp_auth_alg == "hmac-md5")) && (network_mode == "default")) && (security_level == "default")) ||
```

[illegible]

```
(network_mode == "crisis"))))) || (((((((local_filter_port == "23") &&
(esp_auth_alg == "hmac-sha")) && (esp_enc_alg == "3des")) &&
(esp_present == "yes")) && (security_level == "low")) && (network_mode
== "crisis")) || (((((((remote_filter_port == "23") && (esp_auth_alg ==
"hmac-sha")) && (esp_enc_alg == "3des")) && (esp_present == "yes")) &&
(security_level == "low")) && (network_mode == "crisis"))) ||
((((((local_filter_port == "79") && (ah_auth_alg == "hmac-ripemd")) &&
(ah_present == "yes")) && (security_level == "low")) && (network_mode
== "crisis")) || (((((((remote_filter_port == "79") && (ah_auth_alg ==
"hmac-ripemd")) && (ah_present == "yes")) && (security_level == "low"))
&& (network_mode == "crisis")))))
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX M. SECURITY PROPOSAL SUMMARY

The following table is a summary of the security proposals found in **isakmpd.policy** (Appendix K) and **keynotednf.policy** (Appendix L).

			SECURITY LEVEL		
			Low	Medium	High
			Encryption/Authentication Algorithm		
N E M T O W D O E R K	Normal	Telnet	DES MD5	CAST SHA	3DES SHA
		Finger	MD5	MD5	SHA
	Crisis	Telnet	3DES SHA	3DES SHA	AES SHA
		Finger	SHA	SHA	SHA
	Impact	Telnet	DES MD5	DES MD5	3DES MD5
		Finger	MD5	MD5	MD5
	Default Setting	Telnet	DES MD5		
		Finger	MD5		

Table M.1. Security Proposal Summary

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Angelos D. Keromytis, John Ioannidis, and Jonathan M. Smith, *Implementing IPsec*, In Proceedings of the IEEE *Global Internet (GlobeCom) 1997*, pp. 1948 - 1952. November 1997, Phoenix, AZ.

Aurrecoechohea, C., Campbell, A., and Hauw, L. A., *A Survey of Quality of Service Architectures*, Multimedia Systems Journal, Special Issue on QoS Architectures, 1996.

Blaze Matt, Feigenbaum, Joan, Ioannidis, John, and Keromytis, Angelos D, *The KeyNote Trust Management System Version 2, (RFC 2704*, Network Working Group, September 1999, <http://www.ietf.org/rfc/rfc2404.txt>

Blaze, Matt, Feigenbaum, Joan, and Keromytis, Angelos D., *KeyNote: Trust Management for Public-Key Infrastructures*, In Proceedings of the *1998 Security Protocols International Workshop*, Springer LNCS vol. 1550, pp. 59 - 63. April 1998, Cambridge, England. Also *AT&T Technical Report 98.11.1*.

Blaze, Matt, Ioannidis, John and Keromytis, Angelos D. *Trust Management and Network Security Protocols*, In Proceedings of the *1999 Security Protocols International Workshop*, April 1999, Cambridge, England.

Blaze, Matt, Ioannidis, John and Keromytis, Angelos D., *Trust Management for IPsec*, In Proceedings of the Internet Society *Symposium on Network and Distributed Systems Security (SNDSS) 2001*, pp. 139 - 151. February 2001, San Diego, CA.

Chatterjee, S., Sabata, B., Sydir, J. *ERDOS Qos Architecture*, SRI Technical Report ITAD-1667-TR-98-075, Menlo Park, CA, May 1998.

Doraswamy, Naganand and Harkins, Dan, *IPsec The New Security Standard for the Internet, Intranets, and the Virtual Private Networks*, Prentice-Hall, Inc., 1999.

Harkins, D. and Carrel, D., *The Internet Key Exchange (IKE)*, RFC 2409, Network Working Group, November 1998, <http://www.ietf.org/rfc/rfc2409.txt>

Irvine, C.E. and Levin, T., *Quality of Security Service*, Proceedings of the New Security Paradigms Workshop, Cork, Ireland, September 2000

ISAKMPD.CONF(5), OpenBSD Programmer's Manual, <http://www.openbsd.org/cgi-bin/man.cgi>, October 1998.

ISAKMPD.POLICY(5), OpenBSD Programmer's Manual, <http://www.openbsd.org/cgi-bin/man.cgi>, October 1998.

Java 2 Standard Edition, V1.2.2 API Specification,
<http://java.sun.com/products/jdk/1.2/docs/api/>, Sun Microsystems, Inc., 1999.

Kent, S and Atkinson, R, *Security Architecture* for the Internet Protocol, RFC2401, Network Working Group, November 1998, <http://www.ietf.org/rfc/rfc2401.txt>

KEYNOTE(5), *OpenBSD Programmer's Manual*, <http://www.openbsd.org/cgi-bin/man.cgi>, October 1999.

Leiseboer, John, *IPSEC – Security Architecture for IP, Part 2: Security Association*, <http://www.chipcenter.com/eexpert/jleiseboer/jleiseboer036.html>, ChipCenter: The Web's Definitive Electronics Resource, Modified 12/05/2001.

(LK(kn_add_action())), *OpenBSD Programmer's Manual Pages*, OpenBSD Operating System Version 2.8, 2000.

LK(kn_do_query()), *OpenBSD Programmer's Manual Pages*, OpenBSD Operating System Version 2.8, 2000.

Maughan, D., Schertler, M., Schneider M., Turner J., *Internet Security Association and Key Management Protocol (ISAKMP)*, RFC 2408, Network Working Group, November 1998, <http://www.ietf.org/rfc/rfc2409.txt>

Naval Postgraduate School, NPS-CS-02-001, *IPsec Modulation for the Quality of Security Service*, Spyropoulou, Evdoxia., Agar, Christopher D., Levin, Timothy, and Irvine, Cynthia, January 2002.

Naval Postgraduate School, NPS-CS-02-002, *KeyNote Policy Files and Conversion to Disjunctive Normal Form for Use in IPsec*, Spyropoulou, Evdoxia., Levin, Timothy, and Irvine, Cynthia, January 2002.

Naval Postgraduate School, NPS-CS-02-003, *Demonstration of Quality of Security Service Awareness for IPsec*, Spyropoulou, Evdoxia., Levin, Timothy, and Irvine, Cynthia, January 2002.

Savolainen, Sampo, *Internet Key Exchange (IKE)*, Department of Electrical and Communications Engineering, Helsinki University of Technology, 1999.

Simpson, W A, *Cipher Block Chaining (CBC)*, Internet Draft, Network Working Group, July 1998, <http://www.ietf.org/proceedings/99mar/I-D/draft-simpson-cbc-01.txt>

Spyropoulou, E., Levin, T., and Irvine, C.E., *Calculating Costs for Quality of Security Service*, Proceedings of the 16th Annual Computer Security Applications Conference, New Orleans, LA, December 2000.

Thayer, R., Doraswamy, N., and Glenn, R., *IP Security Document Roadmap*, RFC 2411, Network Working Group, November 1998, <http://www.ietf.org/rfc/rfc2411.txt>

Using IPsec (Internet Security Protocol), <http://www.openbsd.org/faq/faq13.html>,
October 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Carl Siel
Space and Naval Warfare Systems Command, PMW 161
San Diego, California
4. Commander, Naval Security Group Command
Naval Security Group Headquarters
Fort Meade, Maryland
5. Ms. Deborah M. Cooper
Deborah M. Cooper Company
Arlington, Virginia
6. Ms. Louise Davidson
N643, Presidential Tower 1
Arlington, Virginia
7. Mr. William Dawson
Community CIO Office
Washington DC
8. Ms. Deborah Phillips, Community Management Staff
Community CIO Office
Washington DC
9. Capt. James Newman
N64
Presidential Tower 1
Arlington, Virginia
10. Major Dan Morris
HQMC, C4IA Branch
TO: Navy Annex
Washington, DC

11. Mr. Richard Hale
Defense Information Systems Agency, Suite 400
Falls Church, Virginia
12. Ms. Barbara Flemming
Defense Information Systems Agency, Suite 400
Falls Church, Virginia
13. Mr. Michael Green, Director
Public Key Infrastructure Program Management Office
National Security Agency
Ft. Meade, Maryland
14. Dr. Cynthia E. Irvine
Computer Science Department, Code CS/IC
Naval Postgraduate School
Monterey, California
15. Mr. Timothy Levin
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, California