SECURE COMMUNICATION CHANNEL MECHANISMS FOR ISOLATED NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÖKDENİZ KARADAĞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

DECEMBER 2009

Approval of the thesis:

SECURE COMMUNICATION CHANNEL MECHANISMS FOR ISOLATED NETWORKS

submitted by **GÖKDENİZ KARADAĞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering** _____

Dr. Attila Özgit
Supervisor, **Computer Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. M. Ufuk Çağlayan
Computer Engineering Dept., Boğaziçi University _____

Dr. Attila Özgit
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU _____

Dr. Onur Tolga Şehitoğlu
Computer Engineering Dept., METU _____

Dr. Cevat Şener
Computer Engineering Dept., METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: GÖKDENİZ KARADAĞ

Signature :

ABSTRACT

SECURE COMMUNICATION CHANNEL MECHANISMS FOR ISOLATED NETWORKS

Karadağ, Gökdeniz
M.S., Department of Computer Engineering
Supervisor : Dr. Attila Özgit

December 2009, 54 pages

Current network security solutions are consisted of a single host, with network interfaces of the host connected to protected and external networks at the same time. This design ensures security by restricting traffic flow to a single point, where it can be examined and acted on by a set of rules. However, this design also has a flaw and a single point of failure, that being the vulnerabilities in the security device itself. An adversary would have unhindered access to protected networks if a vulnerability in the security device itself leads to its compromise. To prevent this possibility, high-security networks are completely isolated from external networks, by prohibiting any network connection and constituting a so-called air gap in between. But, data transfer needs do arise between external networks and high-security networks, and in current technology this problem does not have a solution without human intervention. In this theses, we propose a set of mechanisms that allows near-realtime data transfers between high-security network and external networks, without requiring any human intervention. The design consists of two hosts connected via a shared storage, transferring only application layer data between networks. This prevents attacks targeting network stacks of the security device's OS, and confines a compromised security device to the network that it is already connected to. In case of a compromise the amount of possible unwanted traffic to and from the high-security

network is vastly reduced.

Keywords: network, security, isolation, secure communication

ÖZ

YALITILMIŞ AĞLAR ARASINDAKİ GÜVENLİ İLETİŞİM KANALI MEKANİZMALARI

Karadağ, Gökdeniz
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Dr. Attila Özgit

Aralık 2009, 54 sayfa

Mevcut Ağ güvenliği çözümleri, aynı anda hem korunan ağlara hem harici ağlara bağlı ağ arayüzlerine sahip tek bir sunucudan oluşmaktadır. Bu tasarımda trafiğin akışı, incelenip üzerlerinde bir komutlar kümesi uygulanabilmesi amacıyla belirli noktalara sınırlanarak güvenlik sağlanır. Ancak bu tasarımın zayıf noktası, güvenlik cihazının kendisinde ortaya çıkabilecek güvenlik zafiyetleridir. Bu zafiyetlerden yararlanarak cihazı ele geçiren bir saldırgan, korunan ağlara hiç bir engellemeyle karşılaşmadan erişebilir. Bu olasılığın önüne geçmek için yüksek güvenlikli ağlar, herhangi bir ağ bağlantısına izin verilmeyerek yani bir "hava yastığı" oluşturularak harici ağlardan tamamıyla ayrı tutulmaktadır. Ancak harici ağlar ile yüksek güvenlikli ağlar arasında veri aktarımı ihtiyacı duyulmaktadır ve günümüzde kullanılan güvenlik teknolojilerinde insan müdahalesi gerektirmeyen bir çözüm mevcut değildir. Bu tezde, yüksek güvenlikli ağlarla harici ağlar arasında, insan müdahalesi gerektirmeksizin gerçek zamanlıya yakın veri aktarımına izin veren bir grup mekanizma önerilmektedir. Tasarım, paylaşılan bir veri depolama aygıtıyla birbirine bağlı iki sunucudan oluşmaktadır. Bu sunucular sadece uygulama katmanındaki verileri aktarmaktadır. Bu sayede, güvenlik cihazının işletim sistemini hedef alan saldırıların önüne geçilmekte, ele geçirilen güvenlik cihazları halihazırda bağlı bulundukları ağla sınırlanmaktadır. Başarıya ulaşan bir saldırı durumunda,

yüksek güvenlikli ağ ile harici ağlar arasında oluşabilecek istenmeyen trafik büyük oranda azaltılmaktadır.

Anahtar Kelimeler: ağ, güvenlik, yalıtım, güvenli iletişim

to my family

ACKNOWLEDGMENTS

I would like to thank to Kerem Hadımlı, Korhan Gürler, Can Eroğul, Ahmet Ketenci and Serdar Dalgıç for their help with the implementation,

to Attila Özgit and Onur Tolga Şehitoğlu for their invaluable guidance,

to Özgür Kaya and Can Eroğul for being wonderful roommates,

to Computer Engineering Dept. staff for their great company,

to Deniz Sevimli for easing life's burdens, and providing new perspectives on every single thing,

to my family, Gülhan & Naci Karadağ, for their immense support, patience and for all their efforts on me.

This study was supported by Graduate Study Scholarship of The Scientific & Technological Research Council of Turkey (TÜBİTAK), and by Invicta Research and Development Ltd.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

Introduction

Security is amongst the most important aspect of managing computer networks. Security should have a priority in network requirements analysis phase, so that the design and implementation of a network or sub-network is completed with necessary and foreseen security measures in place.

As with other computer security subtopics, network security is mostly dependent on the owner organization's policies. Policies are typically not technical in nature; they define what the organization sees as valuable assets, what threats those assets can be subject to, and risk analysis of possible threats. Based on the policies, procedures list the precautions against threats, any emergency responses to an incident, and recovery after the incident.

Some organizations (like military, finance, high-tech research, etc.) have security policies so strict, and risks of possible compromises so high, thus no connection to external networks is acceptable from within the organization's high-security internal network(s), or between internal networks themselves. Even then, some data transfer and/or online interaction needs arise between the isolated high-security network and external networks. Employing a regular router and/or firewall among those networks are not acceptable, manual data transfer is performed via physical media. The process is similar to these steps; as an operator or a robot must connect the media to one network, copy relevant data, disconnect it from the network, connect it to the other network, and complete the data transfer. This imposes very long delays to the data transfer compared to inter-network data transfer speed.

The aim of this study is to design and develop a secure communication channel solution by providing trust mechanisms and a device protection mechanism to construct a complete end-to-end trust chain. The channel is to be used when an organization needs both the security

provided by the network isolation and non-delayed data transfer between networks.

1.1 Current Network Security Facilities

Current technologies for securing computer networks that are somewhat limited in alternatives, are given below.

- Firewall software on router hardware
- Firewall software on *off-the-shelf* computer router
- Hardware firewalls
- Specialized application layer firewalls (*XML firewalls, etc*)
- Ethernet Bridges
- Intrusion Detection Systems & Intrusion Prevention Systems

For a network that requires only intermediate level of security, use of a firewall implementation on a chosen router can be satisfactorily enough. Application layer filtering can be performed on routers, in addition to stateless and stateful packet based filtering.

The router itself can be a hardware router, including optimized and specialized electronic circuits for routing network packets. They often run proprietary operating systems designed by the router manufacturer, and thus they are limited to the capabilities provided by the manufacturer.

A similar alternative would be to use an off-the-shelf computer with multiple network interfaces as a router. A proprietary or *free and open source* operating system can be installed and configured as a router. Common operating systems typically include firewall software; *Linux* is bundled with *Netfilter/iptables*, *Solaris* is bundled with *IPFilter*, *FreeBSD* is bundled with *IPFilter, PF and IPFIREWALL (IPFW)*, *OpenBSD* is bundled with *PF*, *Windows* is bundled with *Windows Firewall*.

These are stateful packet inspection firewalls, with limited application layer filtering capabilities.

Application layer firewalls are focused on one or more application layer protocols, and provide validation of data, and implement signature based attack detection facilities.

Ethernet bridges are deployed as “invisible” firewalls, without introducing any changes into the network topology. An ethernet bridge has two interfaces connected to the *same* network, and operate similar to a network switch, storing and forwarding messages. When located in a central location and configured for filtering the traffic that passes through it, an ethernet bridge can introduce and enforce security policies transparently [1].

Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) provide far greater control on the network traffic. These systems check the traffic for a number of signatures, and if a match is found they act on the traffic, which may range from rejecting traffic to modifying the application layer content.

1.1.1 Shortcomings of Current Solutions

Currently used network security solutions all share a common design; the firewall has two or more network interfaces directly connected to external and internal networks that are being protected. A compromise of the firewall leads to complete exposure of protected networks, the adversary or adversaries can access to all hosts within.

Even if the attackers achieve unprivileged code execution capabilities from a lesser security vulnerability, it is sufficient to access other hosts within protected network. This can lead to further exploitation of possible vulnerabilities on those hosts. Having firewalls activated on all internal hosts may protect against these attacks; but this practice is not common because of setting up firewalls on every host and keeping the configuration updated for user needs is labor intensive. Also, the vulnerability that led to the compromise of the firewall may be present on the internal hosts themselves, as well.

This weakness results in strict policies that prohibit any connection between the high-security network and external networks, even in cases where a non-delayed connection is needed.

1.2 Objectives

Objective of this study is to design a set of mechanisms to provide high security networks with as much isolation as possible, while allowing near real-time network traffic to and from the protected network. Security mechanisms providing trust between various system components and a mechanism for protecting and controlling access to hardware devices is proposed.

Secure Communication Channel Mechanisms for Isolated Networks (SCCM) that are proposed in this theses has been incorporated into a research prototype architecture, named as *"Virtual Air Gap (VAG)"*. There has been national and international patents issued for VAG [2].

VAG is a system including two hosts and a shared storage hardware. Protected network is isolated from external networks by stripping out all physical, data link, network and transport layer headers and trailers; transferring only the filtered application layer data over the system. The operating system of the hosts are hardened using widely applied security measures. Application layer data is checked and filtered to ensure any allowed traffic conforms to security policies. Encryption and obfuscation are utilized to hinder any attempts at infiltrating, reverse engineering and bypassing the security provided by the system.

1.3 Organization of Theses

This manuscript is organized as follows. Previous studies related to thesis matter is presented in Chapter 2. Chapter 3 elaborates on the general design of the system. In Chapter 4 the implementation and deployment details of the system are introduced, then various evaluations of the system are presented. Finally, Chapter 5 gives a summary of the theses and discusses future directions.

CHAPTER 2

Related Work and Literature Survey

As computer networking become ubiquitous, security implications of the computer networks became much more evident. Expanding of Internet to most commercial companies, governmental organizations and households resulted in a large volume of communications and transactions occurring on-line. Security weaknesses of network protocols and services became evident not so long after their use [3]. Protecting sensitive information from leaking, preventing monetary thefts and fraud, guarding the networked software from attacks became some of the basic aims of a whole network security industry.

First products for protecting computer networks were packet filters. They provided a decision mechanism based on source and destination addresses and ports, transport layer protocols, flags and other network packet properties. Their shortcomings and possible improvements were discussed and improvements have been suggested [4].

Packet filters provided some level of security, but for critical networks more security coverage was needed. Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) provided far more traffic inspection capabilities, They also introduced wider possibilities of action to be applied on the network traffic [5] [6].

Packet filters, IDS/IPS technology and application layer firewalls share a common design flaw of being connected to both networks at the same time, as discussed in Section 1.1.1 (Page 3). For high security networks where security is of paramount importance, the only ultimate solution was cutting off all network connections. Without a network connection, remote attacks are completely prevented, and local attacks can be prevented by physical security measures. This solution does not allow non-delayed data transfer to and from the network.

There had been a security device named “e-gap” in the industry, which provided physical isolation between networks while allowing data transfer [7] [8]. The product has been discontinued, there is no peer reviewed research done on the subject and viability of the solution. e-gap is designed to run on specially designed proprietary hardware.

Providing trust between system components have been achieved by various means. One method is to use cryptographically signed executable files[9] [10] [11] [12]. The ELF executable file format that is used on Unix systems has various fields that are used for describing details of the executable. Signed executables contain a cryptographic signature in a custom field of the ELF format. The operating system verifies the validity of the executable and the signature before execution. Common operating systems have a facility named *IOCTL* that provides applications the ability to directly communicate with device drivers. This facility has previously been used to provide a channel for transferring cryptographic information to OS kernel modules [13].

Asymmetric key cryptography [14] is a widely used cryptographic approach, where the communicating parties do not need to share a known secret. This approach eliminates the requirement of trusting the information exchange medium, or the corresponding party. Asymmetric key cryptography is extensively used in Public-key Infrastructure (PKI) [15].

Key hopping [16] schemes provide protection against system compromises, by changing the keys within reasonably small time intervals. Any attacker successful at recovering a key can use the key only for the duration of the key hop. To completely compromise a key hopping scheme, an attacker must completely identify the algorithm used for key hops.

The Secure Communication Channel Mechanisms (SCCM) for Isolated Networks is implemented on a Linux system. Linux is an open source and free software Unix-like operating system. Its main design structure and development process has been discussed in various publications [17] [18] [19].

The Virtual Air Gap project, which incorporates The Secure Communication Channel Mechanisms for Isolated Networks proposed in this study, is designed to provide a security capability resembling complete isolation to computer networks, without introducing significant delays. It is designed and developed to accommodate security requirements of military networks. The approach is different from most other security devices and methods, while carrying some si-

milarity to “e-gap” commercial product. Previous academic literature does not define a similar approach. Virtual Air Gap aims to be a complete security solution, addressing the needs of high security organizations with minimized management and deployment costs.

2.1 Linux Security Modules

The Linux kernel implements various security hooks, to allow development of various security related functionalities, which could implement a variety of computer security models. *Linux Security Modules*, or *LSM*, is a generic security framework [20], which is mainly used for implementing Role Based Access Control and Mandatory Access Control. LSM provides *security hooks* within most kernel level operations; e.g., performing a system call, reading or writing a file, spawning a child process. Before performing an operation, the operating system’s kernel checks the security hooks, as depicted in Figure 2.1. The security hooks are called after all other checks performed by the kernel, e.g. existence of a file, hardware errors, and permission checks.

Checking of a security hook depends on its use by LSM modules. If a function is registered for a security hook by a security module, that function is called before performing the kernel level operation, the function gets relevant kernel structures as parameters to enable a security decision. The security function analyzes the request, and then decides on an allow/deny result. The decision is taken according to the details of the security module and the security policy implemented by it.

The return value of the security function denotes the outcome of the kernel level operation. If the security function allows the operation by returning zero, the operation is performed normally and its result is returned to the requester of the operation. The security function can prevent the execution of the operation by returning a non-zero error value.

From the process point of view, the security decision and the presence of the security function does not affect the actual execution of the kernel level operation. As the security module is consulted at the last step before execution of the requested operation, regular sanity and error checking operations are not affected. If the kernel operation would result in an error, caused by another external factor, the requester of the operation will get the error. If the operation completes successfully, the requester will be notified of the success. From the viewpoint of

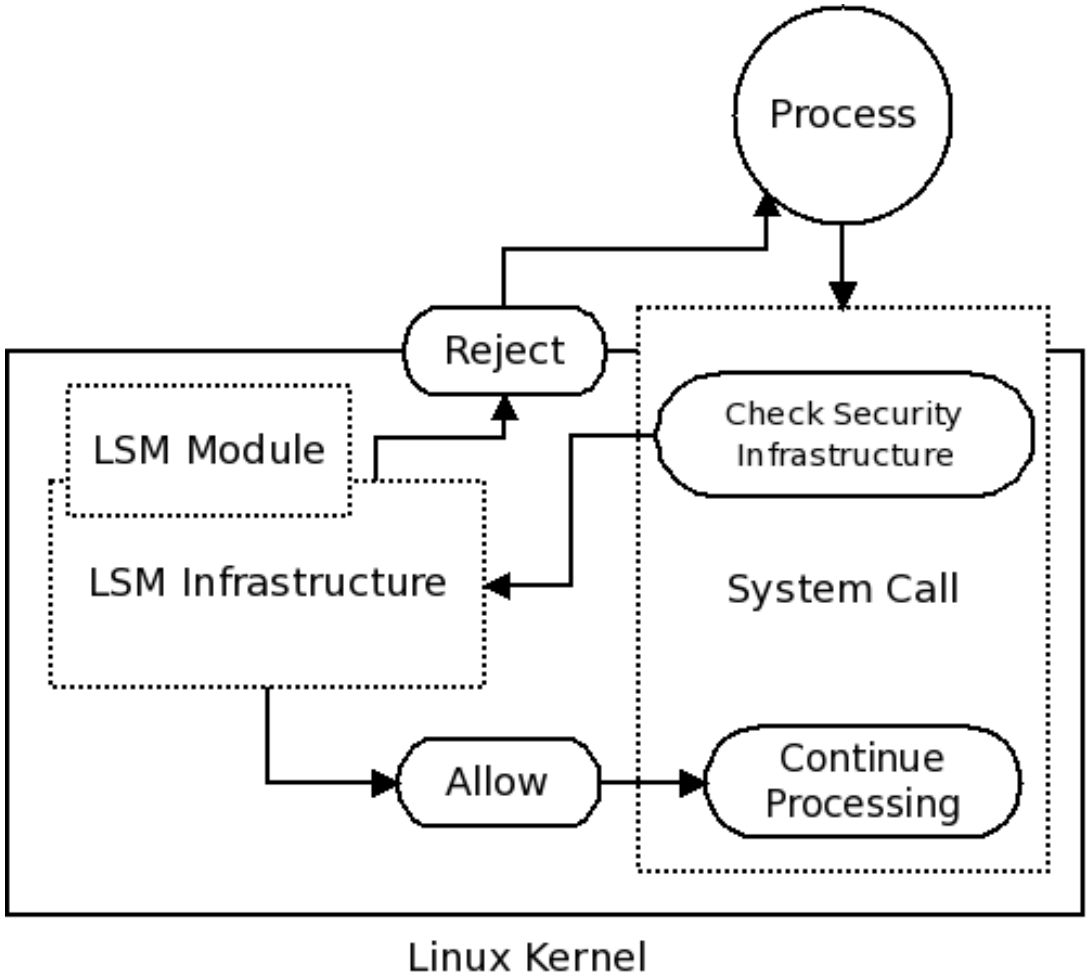

Figure 2.1: Overview of LSM Functionality

the requester of the operation, the total process of the kernel operation is unmodified, if the security module does not deny the operation. In that viewpoint, the operation has successfully completed with all required side effects and return values, only the overhead of consulting the security module is added to the whole process.

2.2 Security File System

The Linux Kernel includes various pseudo filesystem implementations. These pseudo filesystems do not correspond to data storage hardware, but they are used as pseudo interfaces to various kernel functions and structures. “proc” and “sys” filesystems are widely recognized examples of pseudo filesystems, which provide access to information about processes and

various kernel level settings.

The "securityfs" file system is such a pseudo file system designed to provide access to Linux Security Modules components, from user space processes [21]. The securityfs is generally mounted at the path */sys/kernel/security*. Each LSM can register new files and directories under the securityfs, using an API. The securityfs API is more flexible than other pseudo file system APIs. The API is simple, with only three available function calls, which are summarized in Table 2.1.

Table 2.1: Securityfs API functions

Function Name	Parameters	Description
securityfs_create_dir	const char *name, struct dentry *parent	This function creates a directory in the securityfs. Parent directory is defined by "`parent`" parameter. To create a directory directly under the root of the securityfs, NULL is passed as "`parent`".
securityfs_create_file	const char *name, mode_t mode, struct dentry *parent, void *data, const struct file_operations *fops	This function creates a file in the securityfs under the directory pointed by "`parent`". The file operations are defined by the functions pointed from inside the "`fops`" structure.
securityfs_remove	struct dentry *dentry	This function removes a previously created securityfs entity, supplied in the "dentry" parameter.

The bulk of the securityfs interface of the LSM is handled by the functions passed to the `securityfs_create_file` function inside the "`fops`" of the type `struct file_operations`. The structure contains pointers to functions for various file operations. The available file operations are all operations that the Linux kernel allows on files, including device special files. The LSM can register handlers for all operations, including but not limited to; read, write, ioctl, fsync, llseek operations. Figure 2.2 provides a general view on securityfs.

There are no restrictions for implementation of file operations in the LSM, thus the files on the securityfs have maximum flexibility, and can be freely implemented to match particular LSM design requirements.

Various LSM modules included in the Linux kernel can use the securityfs interface for security policy setting and auditing. The user level processes responsible with the security policy configuration can configure the LSM kernel module by accessing and modifying the files

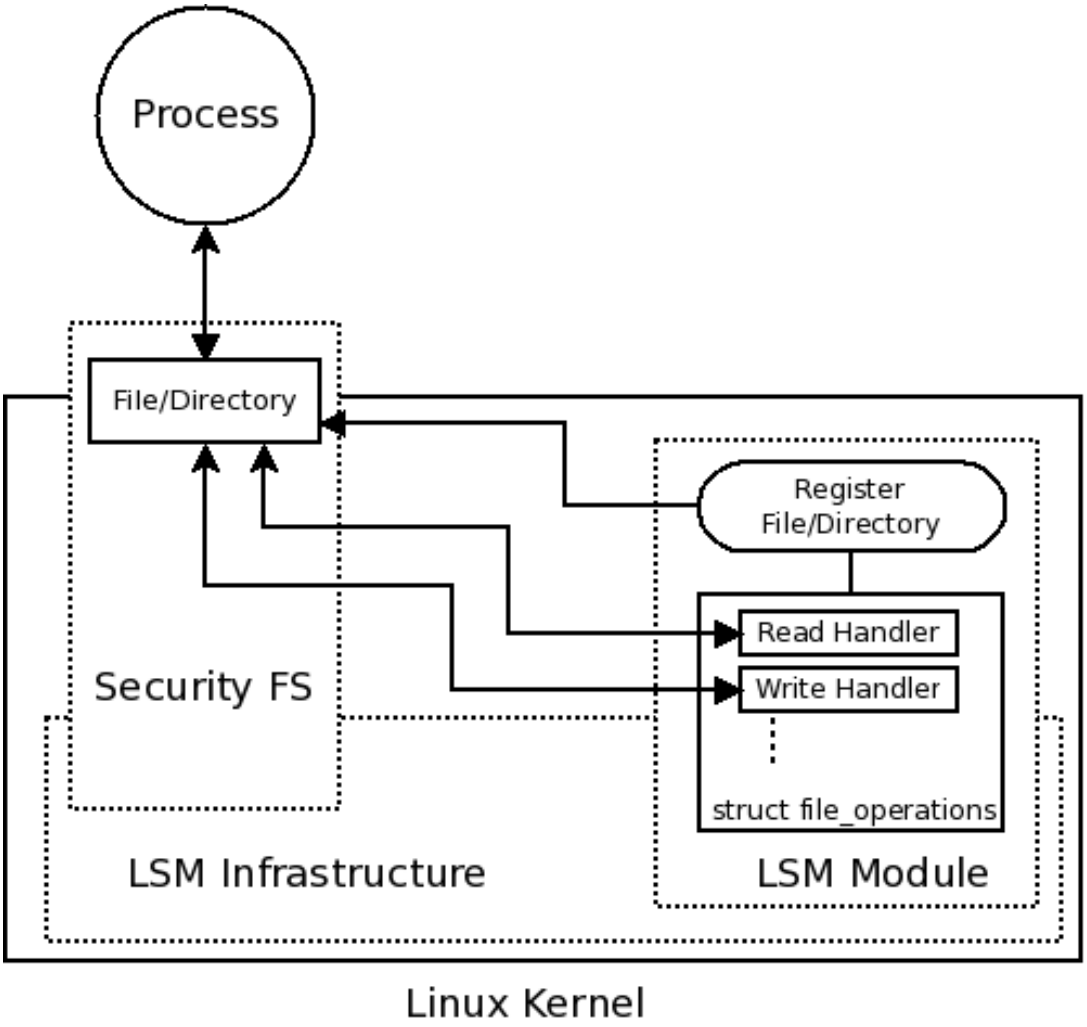

Figure 2.2: Overview of securityfs

under securityfs.

As of December 2009 there are three LSM modules in Linux Kernel; Selinux, Smack and Tomoyo [22]. There are also security modules not included in main Linux Kernel, like AppArmor [23]. These LSM modules implement security and access control functionality, using LSM infrastructure, each taking a different approach. Generally, the modules *label* the resources and entities in the Linux system; and at the time of hook checking, verify that the entity requesting the resource has a compatible security label. Actual implementation details vary among these modules.

Out of three LSM modules in Linux kernel, only Tomoyo uses securityfs. Selinux and Smack

have their own file system implementations, named *selinuxfs* and *smackfs*, respectively. AppArmor, an LSM module not included in Linux kernel, uses the securityfs for policy setting and auditing. As securityfs is generic and flexible enough, it can be assumed that modules currently implementing their own file systems would migrate to securityfs over time.

There are also users of securityfs other than LSM modules. Runtime Integrity Measurement Architecture (IMA) that is developed by The Trusted Computing Group (TCG) and included in Linux kernel, provides integrity checking. It does not use the LSM infrastructure, but it uses securityfs for input of policy changes and output of integrity reports.

CHAPTER 3

Design of Secure Communication Channel Mechanisms for Isolated Networks

As can be inferred from previous discussion, providing non-delayed access to and from high-security protected networks is a major challenge. Some proposed solutions are either not scalable or does not cover all requirements of organizations like isolation and non-delayed access at the same time.

The proposed mechanisms an the overall design of VAG overcomes aforementioned shortcomings by employing a distinct host for both networks to be connected, and using a shared storage in between, as the **application layer** data transfer medium.

The Secure Communication Channel Mechanisms (*SCCM*) proposed in this thesis provide high security networks with complete network layer isolation from external networks. The design only allows application layer data to cross network boundaries, all data link layer and network layer data is stripped from the traffic thus shielding the protected network from any unauthorized access that may result from attacks targeting the security device and/or the operating system.

The usage of a shared data storage, with a unique data format and access mechanism increases security by preventing an attacker to compromise hardware aspect of the SCCM. Having the shared storage hardware as the sole data transfer medium, the SCCM requires minimal permissive changes in the existing security policies of an organization.

As the data is transferred without requiring any human intervention, the system does not impose large delays on network traffic, and provides near real-time communication between the protected network and the external network.

Industry standard security measures is applied to both hosts of the system, increasing overall system security. Various trust facilities between system components help preventing a possible host compromise from reaching further into the system and the communication channel.

In a network where mandatory access control principles are required, data can be transferred only from a lower security level to a higher security level; preventing *any* traffic to a lower security level network to a higher security one guarantees that the mandatory access control principles are always in effect. While this provides maximum level of compliance with the policies, preventing all egress traffic from the protected network may not be feasible. The client who initiates the connection will get no reply indicating the success of the data transfer. Having the capability of bidirectional communication, the SCCM can also be configured as unirectional channel, which allows replies and acknowledgments from reaching the client. In this configuration, The SCCM would still prevent connections originating from the protected network to a lower security network. In other scenarios where the protected network can transfer data to external networks, a bidirectional configuration of the SCCM is possible. Traffic in both directions is transferred, allowing connections being initiated by clients within both the external and protected networks.

The modular design of the system allows implementation of various application layer protocols, thus enabling transfer of virtually unlimited number of application layer protocols over the implemented secure channel.

The design is assumed to be running on a Linux[1] kernel, and the implementation is completed on Linux. But the design concepts are generic enough to allow implementation on different systems without modifying the design heavily.

In the following sections design of the system is discussed in detail.

3.1 General Design of the Mechanisms

The proposed mechanism consists of two hosts, both of which are connected to a shared storage. The shared storage may be any storage solution that allows two simultaneous connections over two distinct channels. Choosing a device with multiple disks allows the implementation to parallelize the traffic load and may improve performance. But a device with a

[1] http://kernel.org

single disk is equally usable with a small performance penalty.

The hosts must have access to shared storage over a well-known physical interface *(e.g. SCSI HBA, Fibre Channel)*. The hosts are not required to have multiple network interfaces; a single interface connected to relevant network is sufficient.

The host connected to the high-security network is called *internal* host, and the host connected to the external network is called *external* host. Throughout this manuscript we will use the terms *internal* and *external* to refer to this basic distinction of the hosts. During the communication, the two hosts perform most operations symmetrically, but in reverse order. While one hosts strips OS network communication headers, and adds SCCM headers needed for data format on the shared storage, the other host strips SCCM headers and adds OS network communication headers.

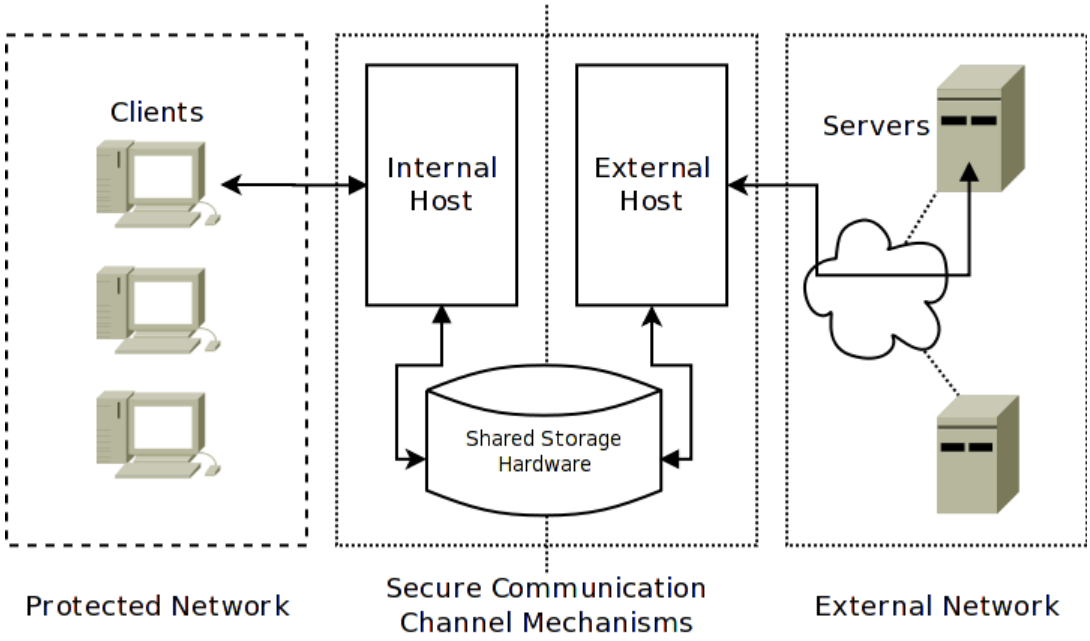

Figure 3.1: General system architecture

The SCCM can be configured as a unidirectional or bidirectional communication channel. In the unidirectional configuration, the protected network may contain clients that send requests to servers in the external network, or the protected network may contain servers that receive

requests from clients in the external network. Figure 3.1 shows the general architecture of the system, when it is configured as a unidirectional channel with clients in the protected network.

The shared storage access is encapsulated in a kernel module, to which the higher level components connect using standard system calls and specially designed IOCTL calls. Integrating the code into kernel increases the security of the system, by hiding the cryptographic keys in kernel memory. Cryptographic keys are retrieved from external hardware (specialized hardware or external disks) at system boot, and stored in random locations of kernel memory. The external hardware is then manually removed and stored in a safe location, the system disables any further access to external hardware.

This design choice requires an attacker to compromise an account with elevated privileges, commonly known as *root* or *administrator* account, to be able to access kernel memory and begin searching for the encryption keys. This reduces the visibility of the details of the communication mechanisms against external adversaries, in case of a possible compromise.

The message exchange algorithm does not transfer the data over standard file systems. A circular queue implementation uses all space allocated on the storage to queue up packets. The internal structure on the disk is not published, as a result an attacker cannot trivially access and read the messages on the disk, or inject messages into the disk without corrupting the messages present on the disk. This protection is enhanced by the encryption and signing of all messages on the storage, discussed at Section 3.5.2.

The designed security precautions can be considered *security by obscurity* precaution, which are not sufficient to guarantee the security of the system when singled out. But the SCCM integrates various detection mechanisms to detect and identify malicious activity. As soon as malicious activity is detected, alerts are dispatched and communication over the secure channel is suspended until further investigations. Presence of a maximally possible amount of security precautions increases the time and effort the attacker must spend on the system, thus increasing the probability of those actions triggering a detection mechanism. When a detection occurs before the attacker identifies and circumvents all of the security precautions, the SCCM changes its state to *inactive* until necessary investigations reveal the source of the compromise is found and patched. As a result no malicious data is transferred into the protected network.

The system includes a management interface, allowing the system administrators to configure and monitor the whole SCCM. It is located at the internal host, which is connected to internal protected network, to protect it from external access. The interface provides configuration options enabling the administrator to configure components like the packet filter, application layer filter, intrusion detection system, application layer modules, etc. The configurations can be backed up and at any time the system settings can be restored from backed up configurations. The interface also incorporates a monitoring system; displaying the status of various components of the SCCM, having log management capability and alerting the system administrators of any abnormal behavior anywhere within the SCCM.

In the following sections, prominent design aspects of the SCCM are discussed.

3.2 Main Components of The SCCM System

The SCCM system consists of components handling a variety of operational and management tasks. The parts of the SCCM that are active in data transfer paths form three independent and distinct layers; Application Layer, Message Layer and Device Layer. Each layer is connected to the one below and above it, making use of various connection facilities.

While there are other distinct components of the SCCM system, its core functionality of data transfer between isolated networks can be reduced to these three layers. Other components exist either to further increase the security of the SCCM or to provide additional functionalities, they also depend on the existence and operation of the main layers.

In this section, main layers of the SCCM; Application Layer, Message Layer and Device Layer will be elaborated on.

3.2.1 Application Layer

The connection of the SCCM to the other hosts is provided by the Application Layer. The Application Layers of the SCCM hosts handle the protocol level connections between the clients and corresponding servers. The Layer works essentially as a proxy, complying with the application level protocol, it enables transfer of protocol level data through itself. In a traditional proxy, the connection would be routed over the proxy software and the payload

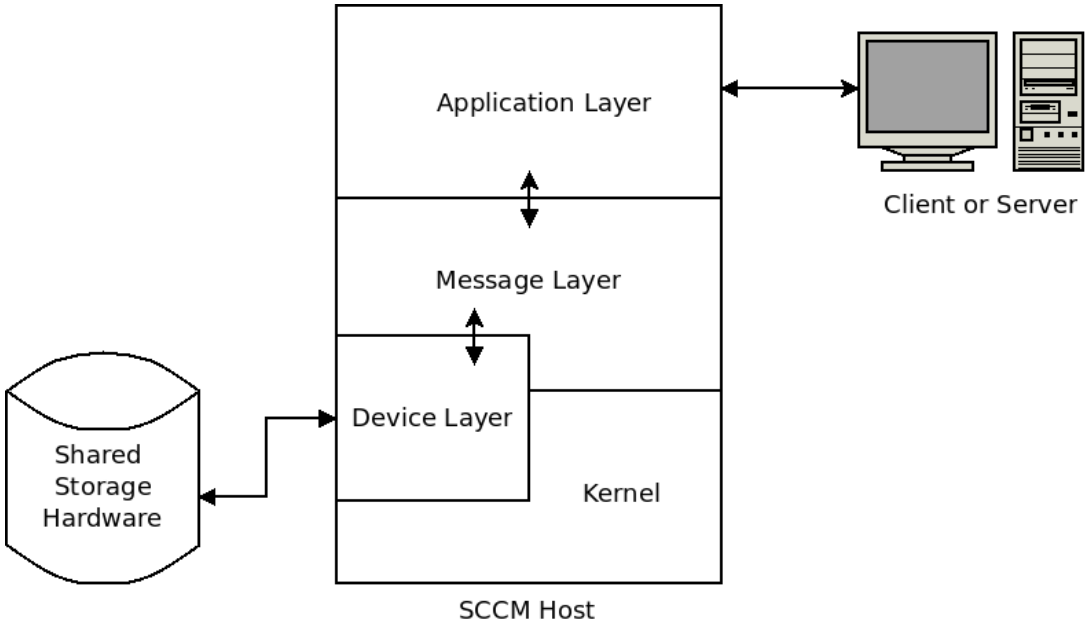

Figure 3.2: View of SCCM Layer structure on a single host

would reach its destination without going through any further device or software.

In case of the SCCM Application Layer, traditional proxy behavior is not followed. It reroutes the connection through the Message Layer to allow further processing and transfer through Device Layer and shared storage hardware.

The Application Layer extracts the data payload from the connection, and transfers only the payload through the secure channel. This feature completely isolates the networking stacks of the SCCM hosts. Various headers and data related to the underlying network protocols are handled at one host, and not passed to the other through the secure channel.

Each network protocol has different syntax, and complies to different sets of standards and specifications. As such, for each network protocol that is to be transferred through the SCCM, a separate Application Layer component would be necessary.

Ideally the application layer must validate the standards compliance of the data traffic passing through it, and should not allow non-compliant requests and replies. Many network software does not implement the standards fully or correctly, thus network proxy software must handle many discrepancies in software behavior. This is also true for the SCCM Application Layer, which may be used with various non-compliant server or client software. As a result, an ideal

filtering is not possible; but the SCCM design compensates for this shortcoming by including an Intrusion Detection System (IDS) as discussed in Chapter 3, Section 3.8.3.

3.2.2 Message Layer

The Message Layer of the SCCM design manages the connections transferred through the secure channel. The Message Layer is an arbitrator between various Application Layer components of the system, allowing each one to send and receive messages through the lower layers of the SCCM and shared storage hardware.

Message Layer labels each protocol level connection with a unique protocol and connection identifier, to guarantee correct delivery of application traffic to its destination in the Application Layer. The connection identifiers are synchronized between internal and external Message Layers, by using "control messages", carrying information about the main data stream.

The Message Layer can accommodate infinitely many simultaneous connections, whereas the restrictions of hardware, kernel parameters and programming language facilities would limit the number of maximum simultaneous connections during operation.

In addition to main data transfer channels, identified by connection identifiers, the message layer can utilize other specialized channels. The control messages are transported over such a specialized and dedicated channel. The Management Component of the SCCM needs to transfer data between hosts through the secure channel. The Message Layer provides a facility for this requirement by dedicating a data channel for the use of Management Component. The Message Layer can accommodate for other special channel needs by reserving more connection identifiers for internal use.

3.2.3 Device Layer

The Device Layer of the SCCM is the layer managing the shared storage hardware. The Device Layer provides controlled access to the hardware for the layers and components above it. Maintaining the on-disk structure of the messages on the shared storage hardware is also the responsibility of the Device Layer. The data on shared storage is stored in a uniquely designed structure, so it is not possible to reconnect the hardware to a non-SCCM host and

easily access the traffic data on the hardware.

The Message Layer connects to the Device Layer for the transfer of the messages that it manages. The Device Layer is is neutral to the message types handled by the Message Layer. The data coming through the message layer is encapsulated in headers specific the Device Layer by chunks, and written on the shared storage hardware. The corresponding Device Layer extracts each message, and the data contained within the message is moved to the Message Layer for further handling.

The Device Layer is flexible enough that it is not limited to shared storage disks. If the particular implementation requires data transfer over a different medium, after necessary modifications, the Device Layer can handle the medium and transfer data over it.

3.3 Network Isolation

The system provides network isolation by removing all physical layer, data link layer, network layer and transport layer headers and trailers from the transferred data packets. The application layer payload of the traffic is inspected, then transferred to the other host via shared storage. This process is akin to copying a file to a removable disk attached to a host in the external network, detaching the removable disk and reattaching it to a host in the protected network, then sending the file to its destination in the protected network.

This isolation denies, without fail, all possible attacks targeting the networking stack of the kernels of the hosts in the protected network. The crafted attack packets are either stripped off from their headers and trailers by the external host and only the application layer payload is transferred, or the attack succeeds at the external host. In both cases, the hosts in the high security protected network are shielded from the networking stack attack by the Secure Communication Channel Mechanisms (SCCM). The thesis proposes additional facilities within the SCCM to limit the impact of the successful attack to the external host.

When a non-malicious packet arrives, its payload is transferred to the internal host over the shared storage. The application layer payload is wrapped in fresh transport layer, network layer, data link layer and physical layer headers and footers by various components of the secure communication channel and the operating system. After the encapsulation of the application

layer payload with necessary networking header and footers, the packet is sent to its destination in the protected network. At this stage, it is no different than a packet originating from the protected network. Before reaching its destination, the packet can pass through routers, packet filters, application layer filters and possibly other SCCM deployments.

A possible reply to the packet will follow the same course in reverse direction. The packet is received by the internal host, which strips all headers and trailers leaving only the application layer payload. The payload is transferred to the external host via shared storage, which wraps the application layer data in headers and footers of other network layers. The newly formed packet is sent to the client in the external network which initiated the connection.

3.4 Root of Trust Concept

The SCCM system is designed to operate under the assumption that the components of the system are not modified. This includes the operating system kernel, programs, and the other components of the SCCM.

The design is built upon the assumption that the system kernel is not previously modified by a malicious entity, thus the system calls and the device layer in the kernel is assumed to be operating as designed, without any external interference.

Other components of the SCCM are designed upon the same assumption that the components they communicate with are not modified by malicious entities and operate as designed.

The design aims to establish a trust relation between components of the SCCM. The trust on the operating system is strengthened by the use of various security features incorporated into the design; use of Host and Network IDS, alerting the administrators upon suspicious activity. Other possible methods for improving the root of trust concept are discussed at Chapter 5.

3.5 Encryption and Trust Facilities

Secure Communication Channel Mechanisms for Isolated Networks includes various facilities providing trust between system components. In general, trust is provided by cryptographically verifiable signatures. The system uses the signatures and encryption to ensure correct

implementation of various crucial assumptions.

- Components of the system are not modified
- Peer is the expected system component and not a malicious entity
- Traffic cannot be easily eavesdropped or modified while being transferred over the secure channel
- Traffic cannot be easily injected into the secure channel
- Malicious activity can be detected and operators can be alerted

Performing encryption in the kernel space increases the security by further obfuscating the encryption keys, and increase performance via optimized and hardware tailored cryptography framework in Linux kernel.

In the design of the system, there is an inherent assumption that the whole host is *secure and trusted* initially after boot. The trust mechanisms, for the most part, rely on the fact that the system is composed of unmodified components at initialization. The very first processes that are run as the part of the SCCM begin establishing a trust chain by utilizing various methods; including cryptography and process tree structure of the operating system. A change in the child processes are detected via OS signals and acted upon accordingly. The security assessment of this aspect of the design is discussed in Chapter 4.

Some security measures of the system rely on this *"secure at boot"* assumption. But the SCCM is not completely reliant on this assumption; there are precautions in the form of cryptographic operations within the components, to verify the identity and integrity of the system component that is being communicated with. In case of a substitution of a system component with a malicious replacement; an adversary , to successfully impersonate a valid system component, must have obtained the cryptographic keys beforehand as well.

In this section, details of the various trust facilities between different components will be discussed.

3.5.1 Process–Device Trust

The storage device layer is comprised of a kernel module. The module provides the device interface to the shared storage for packets destined to the other side of the channel. The Message Layer componens of the system connects to the device provided by the kernel module, and sends the validated, encrypted and signed data. The protocol between kernel module and the higher level components incorporate a trust facility. The facility is implemented via special *ioctl* calls, transferring cryptographically signed messages to the device and kernel module. As a result of this facility the kernel module assures that the connected component has necessary cryptographic keys. This prevents trivially injecting messages into the secure channel, in case of a host compromise.

IOCTL is a mechanism for providing out-of-band communication with the device driver. It is complementary to *read* and *write* system calls. Main use of IOCTLs is to set device driver parameters, possibly modifying the underlying hardware's settings. IOCTLs are achieved by performing the *iotcl* system call, passing the type of the ioctl as a parameter to the device driver. Also, pointer to an arbitrary length data can be included as an argument to the ioctl call. The cryptographic data, authenticating the process is included in this argument part.

3.5.1.1 Design Alternative 1: Stand-alone Processes

The kernel module component of the SCCM, which controls and manages the communication over the shared storage hardware, is configured to accept a single ioctl call. The call carries a data argument part, which is encrypted and signed by the calling process. The encrypted data argument contains two commands targeting the storage layer component. These commands are *initialize* and *terminate*, which enable and disable the data transfer capabilities of the storage layer.

The *initialize* and *terminate* commands can also be implemented as two distinct ioctl calls. This design choice would also require the caller to encrypt and sign a data argument, which may be designated as a known string constant.

Before an *initialize* command is received by the kernel module, any read or write access to and from the device fails. In addition, the kernel module assumes that connection to the device

before a proper initialization message is possibly caused by an adversary, trying to bypass security mechanisms. Thus, an alert is raised and dispatched to the management interface at the internal host. Also, any further communication between the two hosts is inhibited to prevent a possibly ongoing attack on the secure communication channel.

When a component of the system executes the ioctl call properly, the process which has performed the ioctl is granted access to the device driver. The process can communicate with the storage device layer via read and write system calls. It is possible to use higher level wrappers around the rather primitive read and write system calls. The communication is expected to be encrypted and signed for protection of the data, and verification of the identity and integrity of the system component.

Any other process must perform the same initialization steps to be granted access to the device driver and the underlying data channel over shared storage hardware. After a properly encrypted and signed initialization is verified and decrypted successfully, the caller process is allowed access to the storage layer, as depicted in Figure 3.3

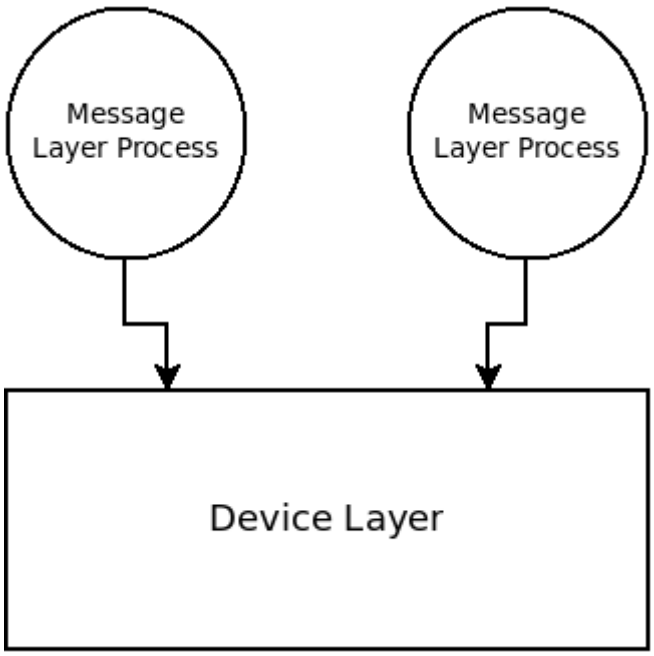

Figure 3.3: Overview of Design Alternative: Stand-alone Processes

Any unauthorized access attempt at any phase of the SCCM operation results in an alert to warn the system operators of a possible security breach. When an alert condition is detected at the storage layer, all communication to and from the shared storage is stopped to prevent

adversaries from reaching the internal host, or injecting unauthorized or malicious data to the communication channel. The only accepted traffic is a message from the internal host prompting that proper operation is to be restored.

After receiving the alert, which warns of a possible security breach, the internal host also stops accepting and sending any message from and to the shared storage device, until proper operation is restored by the administrators. The only accepted traffic is outbound messages, signaling the external host to restore proper operation.

3.5.1.2 Design Alternative 2: Central Management Process

This alternative design is similar to *Stand-alone Processes* design alternative for the most parts. In this design, a central management process manages all other processes of the system, and all communication is multiplexed and distributed by this central management process.

The management process is run as early in the boot sequence of the host as possible. This takes advantage of the *"secure at boot"* assumption discussed at Chapter 3, Section 3.5. All the other components of the system are spawned from this central management process, thus forming a tree of processes with the management process at the root. Any unexpected process termination, which may be indicative of an adversary's attempt at replacing a system component is detected via signal mechanism of the operating system. This would result in the management process to take necessary precautions against possible further compromise attempts, and to raise an alert, warning the human operators.

The central management process handles all communication to and from the storage layer. It initializes the device driver at the storage layer as described at the *Design alternative: Stand-alone Processes* section above. The device driver trusts the management process, and in turn, its child processes which form various components of the SCCM. The initialization operation on the device driver establishes this trust, and cryptographic keys inside the device driver are associated with the management process which always keeps the device open. All communication coming from the userspace components via the now-trusted management process is signed and encrypted before being written to the shared storage device, to provide device layer–device layer trust mechanism defined at Section 3.5.2.

An attempt of replacing the central management process would be detected by the kernel

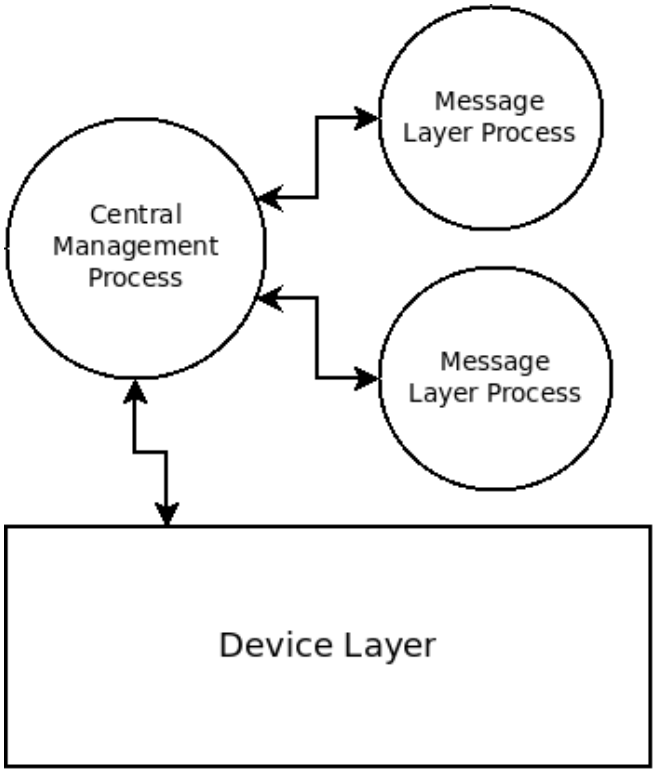

Figure 3.4: Overview of Design Alternative: Central Management Process

module at the storage layer of the SCCM, as soon as the device is closed by the terminated management process. The device driver is designed to raise the alert and prevent further communication with the user space processes, when an attempt at replacing the management process is detected.

This design alternative would increase the performance of the system by eliminating most cryptographic operations between various components of the system, but by still maintaining a trust between system components, relying on the *"secure at boot"* assumption and centralization at the management process.

3.5.2 Device Layer–Device Layer Trust

Each kernel module interfacing to the storage contains encryption keys in kernel memory, retrieved from external sources, that being special cryptographic hardware or external disk. Each message is encrypted and signed using the cryptographic keys, before being written to the shared storage. Corresponding kernel module in the other host verifies the signature of the

packets read from the shared storage, and decrypts the message contents. This process ensures that the other peer behind the shared storage is in possession of approved cryptographic keys. This validation prevents an attacker from directly accessing the shared storage and injecting messages into the communication channel.

A possible scenario may involve locating the internal and external host in different physical locations, with the shared storage cables being the only physical link between locations. Physical security weaknesses may lead to a security compromise. For further exploitation of the system the attacker may disconnect shared storage cables, e.g. for purposes of off-line analysis, inserting a malicious device between shared storage and hosts. When a cable disconnection is detected by the Device Layer, any trust of the other host and the hardware is revoked. Thus the system prevents injection of hostile traffic into the channel. If the cable disconnection is not caused by an adversary, the operators of the SCCM can choose to continue transferring data over the channel.

Another advantage of the device layer–device layer trust is that the messages written to the shared storage are signed and encrypted. In case of a physical compromise of the shared storage, custom designed and unpublished structure of the storage prevents and attacker to trivially access and inject messages to the shared storage. After an analysis of the storage, the attacker can reverse engineer the data structures and become capable of extracting and injecting messages. Having the data encrypted, the attacker must extract decryption keys from the storage kernel module which resides in the kernel memory to be able to read a message. To inject a message that would be correctly decrypted and verified by the other host, the attacker must encrypt and sign it with the cryptographic keys of the host that was compromised. As with reading of messages, injecting messages would also require finding and extracting the keys from kernel memory of the storage kernel module. During this process, probability of the attacker being detected by one of the security measures increases with each action and the time the attacker spends within the system.

3.5.3 Application Layer–Application Layer Trust

Similar to device layer–device layer trust discussed in Section 3.5.2, the application layer–application layer trust facility includes encryption and signing of messages at the application layer components of the SCCM. The corresponding application layer component of the host

at the other side of the channel, verifies the signature and decrypts packets, assuring that they were sent by a peer having valid cryptographic keys. Once again, this may correspond to a valid and unmodified component of the system, but it may correspond to an attacker who has obtained the cryptographic keys.

As the application layer component runs on user space, an adversary can obtain the keys used with relative ease, compared to the keys hidden and protected in kernel memory. A privileged account is not needed, compromising only the valid user account of the application layer component is enough to begin analyzing process memory for key extraction. This is a perceived security weakness in the design, it is amended by requiring the application layer component perform various IOCTL system calls to the storage kernel module. Complicating the path the attacker must follow, increasing the probability that detection mechanisms can identify the attack.

3.6 Dynamic Key Hopping

The storage layer of the SCCM changes cryptographic keys with a reasonable frequency to provide an extra layer of obfuscation and security. By changing keys frequently, the system limits the capabilities of an attacker who may have obtained the keys used in device layer. The keys obtained by the attacker are not used by the device layers of both hosts after a small time interval, leaving the attacker a small time window to decrypt or inject messages. For a complete compromise of the channel, the attacker must reverse engineer the key hopping algorithm used in the storage kernel modules; thus increasing the time the attacker needs to spend on the system, increasing the probability of detection like other components.

3.7 Alerts

Various components of the SCCM are designed to present an attacker with complicated defensive mechanisms, increasing the time the attacker must spend at analyzing and reverse engineering the system. This significantly increases the probability of the attacker being detected by various detection facilities within the SCCM. When a detection takes place, an alert is generated and sent through the secure communication channel to the management interface. According to the severity of the alert, SCCM facilities may interrupt all communications. The

alert is displayed prominently in the management console. After an investigation and analysis process by the administrators, it can be determined that if the alert is result of a successful or unsuccessful attack, or if it is a false positive. After taking necessary precaution, the alert condition is cleared and regular system operation is resumed.

There is an inherent assumption that alerts can reach from external host to internal host, so that system administrators can be warned about a possible attack. If an attacker can garble the data transfer through shared storage, the attacker can prevent alerts from reaching their destination. This would require flooding the shared storage with bogus data, or generating large amount of traffic. However, implementing quality of service and providing a high priority to alert messages can prevent a successful blocking of alerts by flooding.

The device layer at the internal host generates an alert upon receiving bogus data from shared storage, and shuts down all communication through the channel. This alert may indicate a hardware problem, an encryption key mismatch or an attacker trying to garble the secure communication channel to prevent alerts. All of these cases requires a human to intervene and analyze the possible causes before continuing. After ruling out hardware and software error possibilities, the administrators can conclude that the alert was caused by an attacker, and perform necessary security audits on the system.

3.8 Other Security Features

In addition to main issues providing various levels of security, the design of SCCM integrates industry standard practices for securing hosts. The operating system contains minimal number of software components, only the software that are required for a functional barebone system and the SCCM itself are installed.The systems run absolute minimum of services, reducing the area of attack on the host. Only ports that are open to the network are the absolute requirement of the application layer protocols that are being transferred.

Minimizing the network and security footprint of the system is a widely accepted practice of securing a host. An undiscovered vulnerability in software packages could transform into a security hazard; however, by keeping the minimum number of packages installed on the system, the risk of compromise is reduced.

In case of a limited system compromise, availability of only a minimal set of software components hinders abilities of the attacker. The attacker may not easily use many ready made attack programs, as ready made programs require various libraries, language interpreters or compilers. This property of the system forces the attacker to design and implement custom attacks, increasing the skill barrier and at the same time increasing the time required for a complete system compromise without being detected.

Having a minimal number of packages also eases management of the system, as unnecessary packages are not required to be configured and updated. This results in a better managed system, with less distractions on the system administrators, they are expected to be more receptive of possible anomalies in the system.

3.8.1 Packet Filter

The hosts comprising the Secure Communication Channel Mechanisms are protected using a stateful packet filter. The rules are maximally restrictive, all traffic is checked by a default-deny ruleset. Both inbound and outbound traffic is checked and restricted. Only the ports required for the operation of SCCM are allowed, and they are open only to selected hosts which require access according to security policies. Any unauthorized access attempt is logged and displayed on the management interface as alerts.

3.8.2 Application Layer Filter

By inspecting the traffic and taking necessary actions, components of the SCCM eliminate or hinder most attacks targeting the system programs, the operating system or low level networking layers. But application layer data is not checked or modified in lower layers of the SCCM. Still, many contemporary attacks could be targeting the application layer.

Also, while the SCCM protects the internal network from most attacks to some degree as discussed above, if the attacker targets the protected hosts at the application layer, security of the protected network can be compromised. As such, having strong protection measures at the application layer is mandatory to provide an in-depth defense.

The SCCM integrates application layer filter, with rulesets preventing most common attack

forms targeting the application layer. The rules are customizable by the administrator, allowing flexibility to design and implement rules tailored for the specific operational environment.

The application layer filter raises an alert upon detecting data that matches the filter rules. The alert is shown at the management interface to allow the system administrators further investigate the situation. The non-conforming packet is prevented from reaching lower layers of the SCCM and to the protected network via the secure communication channel.

3.8.3 Host and Network Intrusion Detection Systems

One of the main aims of the SCCM is to complicate the process of compromising whole of the secure communication channel, to allow detection of an attack by various facilities. Each component of the SCCM implements necessary precautions for attack detection. Also, general purpose Intrusion Detection System (IDS) is integrated to aid in attack detection.

The two main class of Intrusion Detection Systems are Host IDS and Network IDS, and both approaches are utilized in the SCCM. Host IDS is used to secure the internals of a computer system. This is achieved by analyzing behavior of software programs, applying checksum to system files and detecting any change of the files, detecting hidden processes and rootkits, analyzing system logs for any unusual behavior.

Network IDS is used to secure the system against threats from both networks. This is achieved by using signatures to detect network packets that are part of an intrusion or denial of service attack. Malicious data payload in application layer traffic, like shellcodes or SQL injection patterns, is also detected by signatures and various heuristics.

The Host and Network IDS can detect a large number of anomalous conditions, which are most likely indicative of an attack taking place. The alerts generated by the Host and Network IDS are treated as SCCM alerts, and reported to system administrators.

3.9 Modular Design

Design of the SCCM is modular, allowing many application layer protocols to be implemented as protocol components. Each protocol component contains a proxy service capable of understanding the application layer protocol. The component extracts the application layer payload from the traffic and performs bidirectional transfers between the protected network, via the shared storage hardware. The protocol component needs to connect to the message layer of the SCCM, thus is required to implement all necessary security measures for communication with storage kernel module through the message layer.

3.10 Management

The management component is distinct from the protocol components. It is possible to manage all different protocol and SCCM components from the management interface, in addition to the various configurable aspects of each protocol component.

The user interface subcomponent of the management component is distinct from the main management subcomponent responsible of handling actual management. This enables implementing a set of management capabilities in the main management component, by employing a modular design approach.

Actual user interface exposing the management capabilities can be chosen according to users' needs. A web based, command line based or desktop application based user interface can be employed. A combination of these approaches can also be employed, accommodating a wide range of user needs.

3.11 Other Design Considerations

As main goal of the SCCM is to provide secure communication between isolated networks, the components of the SCCM must be sufficiently secure. While the general design prevents or hinders many types of attacks, introducing SCCM into a network should not increase the possibility of a compromise in hosts or the network. Each component of the SCCM is designed to provide maximum reliability and security. Combined with an implementation that is

sufficiently secure, the proposed SCCM system should achieve its ultimate goals.

Introducing many layers of different security precautions is a design consideration in the SCCM. Accomplishing a successful and undetected attack against the SCCM must be highly improbable to protect the security of the protected network. Preventing analysis of the system is a priority to achieve this goal, which can make use of binary compression and obfuscation, trace protection by use of LSM infrastructure.

The SCCM aims to provide non-delayed network traffic, meaning that the system does not impose a delay unacceptable from a computer network. The usage of shared storage hardware imposes a certain delay in communications. The SCCM design aims for minimum possible delay in networking traffic, thus providing the users of the secure communication channel with fast communications.

The SCCM is designed to run on off-the-shelf *(OTS)* hardware, thus increasing platform choice and reducing the entry barrier of using the system. Backup and high-availability configurations can easily be provided because of the use of off-the-shelf hardware components in the design, instead of the use of proprietary hardware.

CHAPTER 4

Implementation, Evaluation and Security Assessment

The design of the system detailed in Chapter 3 was implemented and tested on live systems. This chapter elaborates on the implementation of the design, various deployment environments and security assessment of the system.

The implementation was run on Debian operating system, version *lenny*, with Linux kernel.

The choice of implementation language for both the message layer and the device layer was C programming language. The choice of programming language for the kernel module was dictated by the choice of Linux kernel as the platform. As the Linux kernel is written mostly in C and Assembly language, kernel modules have been developed also in C language. The choice of C language for the message layer had a few supporting points; ability to program in a lower level than most other languages, potential to improve performance by exploiting the advantages a lower level language provides, and the author's experience with the C language.

Various external libraries and programs were incorporated into the implementation. These include;

- *OpenSSL* cryptography library[1] was used for encryption, decryption, signing and validation operations at the message layer.
- Web based management interface was implemented using *qooxdoo* JavaScript library[2]
- The CGI backend for the management was implemented using jsoncpp[3] and cgicc[4] libraries.

[1] http://openssl.org
[2] http://qooxdoo.org
[3] http://jsoncpp.sourceforge.net/
[4] http://www.gnu.org/software/cgicc/

The implementation of the SCCM was accomplished as part of a research project work, as a team. The author of this study implemented the messaging layer and the kernel module providing access to the shared storage. The cryptography capabilities of the message layer and on-disk data management part of the device layer were implemented by other members of the team.

4.1 Message Layer

The message layer of the SCCM is responsible for managing the traffic flow over the Secure Communication Channel. It can be thought of as a layer between the Application Layer and the Device Layer.

In a communication scenario involving clients inside the protected network and servers in the external network, the Message Layer on the internal host collects requests coming from the clients via the Application Layer. The message layer inspects the incoming traffic to determine if there is a new connection or a new traffic packet in a previously established connection is received. If the connection is new, the message layer assigns a new connection identifier, creates internal sub-structures for accommodating the new connection and creates necessary processes and threads that will handle the new connection for the rest of its lifetime.

After connection initialization operations, the Message Layers sends the requests further towards the shared storage. As a reply arrives to the Message Layer from the shared storage, the Message Layer delivers it to the Application Layer, which sends the reply to the client.

The Message Layer on the external host waits for requests arriving through the shared storage, when a new request arrives it initiates a new network connection to the target server via the Application Layer. When a reply from the target server is received through the Application Layer, the message layer sends it to the shared storage to follow the same path in reverse.

One directional operation of the SCCM dictates that legitimate connections are first encountered by the components at the internal host. This leads to a natural design decision that assigning connection identifiers are to be done at the internal host. The state of a connection is determined by “control messages” which are transferred over a separate control connection. The internal Message Layer sends control messages stating that a new connection is to

be handled, and the control message includes the connection identifier of the new connection. When the a control message informing of the new connection is received at the external host, the message layer at the external host replicates the operations of internal message layer. The operations are; initialization of accommodating structures, handler process and thread creation. Then the handler process and its threads handle the connection for the rest of its lifetime.

As there is only one side assigning connection identifiers to connections, the possibility of conflict is eliminated. The internal message layer keeps track of the connections flowing through the secure channel, and it does not assign duplicate connection identifiers to different connections. The external message layer uses the identifiers assigned by the internal side.

The Message Layer presents a socket interface to the Application Layer components. The application layer connects to the message layer through standard Unix domain sockets. During the life time of the connection between the client and the server, the application layers keep the connection open to message layers. This allows continued transfer of data traffic, keep-alive requests and other necessary facilities of the application layer protocol.

Message layer itself is an agnostic protocol, it can be used with many application layer components. This provides the SCCM with the ability to transfer a wide variety of application level protocols' data.

Communication between the Message Layer and the Device Layer can occur directly, but in the current implementation of the SCCM, there exists a small layer, called the "Message Layer – Device Layer Interface". This layer handles the multiplexing of multiple connections, and maintains the structure of data on shared storage hardware. In the absence of this layer, the Device Layer would handle both tasks.

While running, the message layer maintains a number of processes and threads. For each live connection, Message Layer maintains one process, consisting of two threads. One thread is responsible for the "downstream" data path, that is the flow of data starting from the application layer and continuing to the device layer. The other thread is responsible for the reverse path, i.e. the "upstream" path.

A limited communication facility betwen the upstream and downstream threads is present for informing the other thread of a connection termination. This makes use of the socket bet-

ween Application Layer and Message Layer, as well as the communication channel over the shared storage. No direct communication facility, like an Inter-Process Communication (IPC) mechanism, is required between the upstream and downstream threads of each connection process. This simplifies the implementation of the Message Layer and reduces the need for complex message passing, locking and synchronizing operations.

During the normal operation of the Message Layer, the upstream thread pushes data traffic from the shared storage towards the Application Layer, and downstream thread operates in the reverse direction.

When a communication is closed by the client or the server, the Application Layer component closes the socket of the Message Layer. The downstream thread of the Message Layer sends a "connection closed" message through shared storage, to the Message Layer on the other host. After sending the message, downstream thread exits, while the upstream thread is still running. When the Message Layer on the other host receives the "connection closed" message, its upstream thread shuts down the socket between Message Layer and Application Layer and then exits. The socket shut down event causes the Application Layer component to initiate proper routines for closing the connection between the component and its peer. This shut down event can be detected by the downstream thread. Before exiting itself, the downstream thread sends a "connection closed" message targeting the other host's still running the upstream thread. Exit conditions of both threads are detected by their parent process, by which they were spawned at the beginning of the connection; and the parent process exits. After final "connection closed" message is received by the corresponding upstream thread at the other host, it exists. This event causes the parent process of the connection to exit. This event sequence is designed such that expensive locks and other signaling mechanisms are not required, improving the reliability and performance of the implementation [24].

Under normal operating conditions, the described shutdown sequence is sufficient enough to clean up all processes and threads involved with the connection. It also does not leave any orphaned messages on the shared storage hardware. The upstream thread consumes all traffic data from the shared storage until "connection closed" message, coming from the corresponding Message Layer thread.

The Message Layer is designed in such a way such that it never discards data when the data path to transfer it is still available. For the thread that is connected to the peer that is closing

the connection, the Application Layer socket is unavailable for write operations. Thus the data that arrives between socket closure and receiving of the “connection closed” message is discarded. On the other side of the connection, the application level peer may still be waiting for data from the peer that closed the connection. The Message Layer of the SCCM sends any remaining data on the storage hardware channel, that was sent by the closer peer to the waiting peer. The transfer continues until the “connection closed” message is received, which was generated when the peer closed the connection. After this condition, there remains no data that the peer would wait for in case of an unfiltered, errorless connection. The Message Layer processes exit after having completed transferring entire connection data.

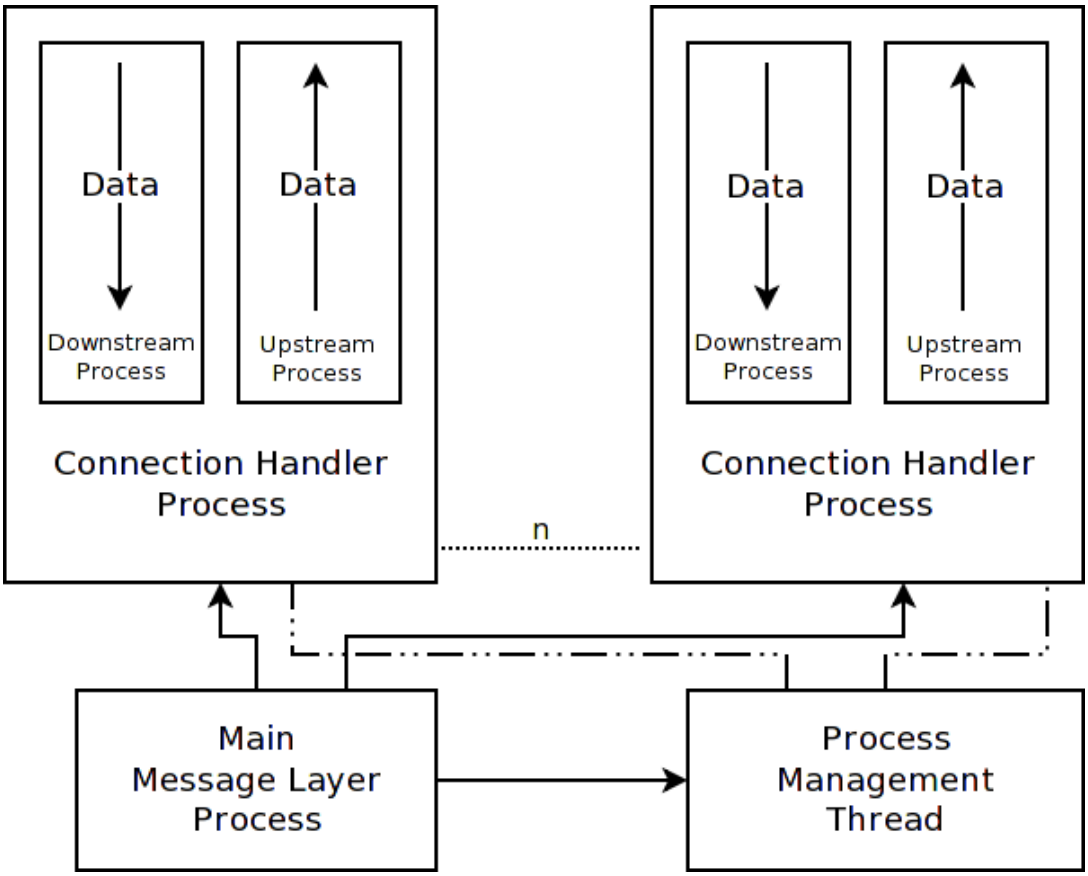

Figure 4.1: Overview of Message Layer processes and threads

The Message Layer maintains a dedicated channel for Management Component. It exposes a separate socket, bound to this dedicated channel, solely for the use of Management Component. This allows for possible prioritization of management commands over the ordinary data traffic passing through the secure channel.

The Message Layer implementation includes accommodating structures for keeping track of connections and processes/threads responsible for each connection. An independent process management thread is implemented to keep track of terminating connection processes, and to guarantee clean shutdown of connections in case of an unexpected operation. The connection closing mechanism is designed on a coordinated shutdown event among four threads on different machines. A problem with the operating system (e.g. Linux "Out of Memory Killer" kills an SCCM process after the system runs out of memory), or a coding error in the implementation may prevent occurence of a required event. These kinds of errors would prevent the sending of "close connection" message, and would leave orphaned data on the shared storage hardware. In case of a premature termination of a connection process, the process management thread takes control of the affected communication channel. It sends a "connection closed" message through the secure channel to start the shutdown procedure on the other host. The process management thread also ensures that all data packets belonging to the connection are cleaned from the storage hardware; it reads and discards data packets of the closed connection, until the "connection closed" message comes from the other host.

4.2 Device Layer

The Device Layer of the SCCM is responsible for providing controlled access to the shared storage hardware. In the design presented at Chapter 3, the Device Layer is a single entity handling the access control of the storage hardware as well as maintaining the structure of data traffic on the shared storage hardware.

In the implementation, this monolithic design was split to favor manageability. The access control responsibility is implemented as a kernel module, using the LSM infrastructure discussed at Chapter 2. The management of the on-hardware data structures are implemented in a separate user space program which is described at Chapter 4, Section 4.3.1. The rest of this section will elaborate on the LSM kernel module part of the Device Layer.

The kernel module has the ability of controlling access to any part of the system, through the hooks provided by the LSM infrastructure. Although in the scope of this study the access to a single entity, the shared storage hardware is controlled.

The module creates a directory under the securityfs pseudo file system, using the `securityfs_-`

`create_dir` API call. A single securityfs file named "checkpath" is created under that directory. The securityfs API call used for the operation is `securityfs_create_file`, and a `struct file_operations` is passed to the call, registering the file operation handlers for the "checkpath" file.

The checkpath file is used for communication the device path of the shared storage hardware. As the hosts of the SCCM system has internal disks for system use beside the shared storage hardware for communication channel, the kernel module cannot reliably determine the exact device path of shared storage hardware. The checkpath file on the securityfs is utilized by the user space components of the SCCM to inform the kernel module of the correct device path to protect. After the checkpath file is opened, and the correct device path is written into it, the LSM kernel module stores the path to use in further operations. The LSM can be configured to accept the path through "checkfile" only one time, preventing alterations of the protected device's path by possible adversary activity further into the operation of the SCCM. During implementation process, kernel module can be configured to accept the protection path as many times as desired.

After the device path to be protected is given to the LSM kernel module, the module resolves the path to an inode pointer. Inodes are structures representing files within the file systems subsystem of the Linux kernel, they mainly correspond to on-disk inode structures of various file systems. Having the inode of the protected device path ready, the kernel module can control access of a file operation with high performance.

The LSM infrastructure provides a pointer to a `struct file` structure to the function in the LSM module that is registered into the `file_permission` security hook. The access control function must perform all checking operations based on the provided `struct file` structure. Each file open operation results in a new file structure being created. While this structure includes many fields related to files, the LSM kernel module implementation uses only two fields within the structure. The inode of the file that is to be checked can be accessed through several indirections of C structures within the file structure. The exact C expression yielding the inode from a `struct file` pointer called `file` is; `file->f_path.dentry->d_inode`. The other field used inside the `struct file` structure is the `private_data`, which is used to store current authorization status of the open file.

LSM infrastructure provides the `file_ioctl` hook for controlling ioctl operations. The De-

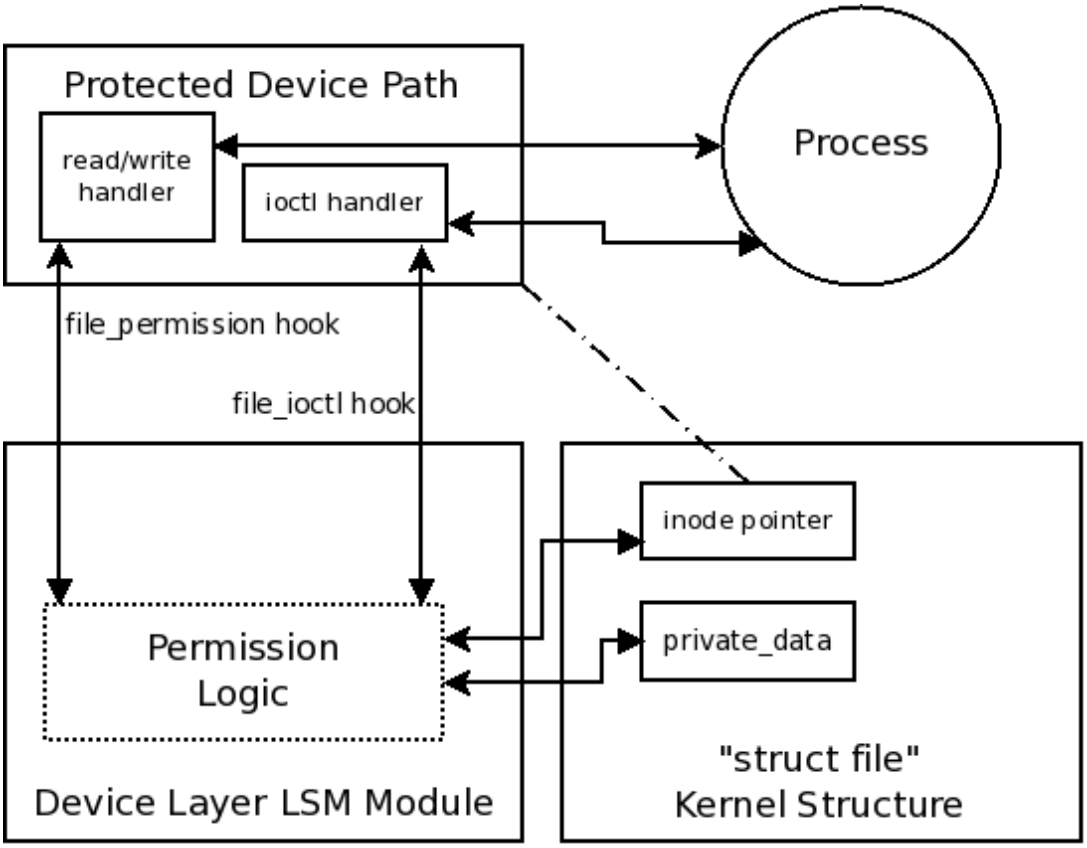

Figure 4.2: Overview of Device Layer LSM Module Operation

vice Layer kernel module registers a function for this hook. The LSM infrastructure provides the ioctl command as an integer, and ioctl argument as a void pointer to user space memory, as well as the file structure pointer for the target device of the ioctl operation. When an ioctl operation to the protected device, the kernel module checks the ioctl command and its arguments. In the SCCM design, these checks include cryptographical operations with key pairs, further strengthening the identity of the serviced SCCM component. In the implemented version of the device layer, cryptographic functions are not called. Access control is based upon known shared secrets. After a legitimate and expected ioctl call, the kernel module authenticates the caller process, storing the authenticated status inside the `private_data` field of the file structure. The `private_data` field cannot be modified by user space applications in any way, only kernel level code can manipulate the field. As the system is based on a trust rooted concept, all the kernel code is assumed to be valid and not malicious, so under this assumption storing authentication data within the file structure's `private_data` field is feasible.

For checking file operations, the `file_permission` hook handler function checks the contents of `private_data` structure, if it matches the authenticated status data used by the `file_ioctl` hook handler, file access is allowed. This corresponds to the case where the process opened the device, performed a valid ioctl on it and then attempted read and write operations.

If read and write operations are performed before a valid ioctl call, the operations are denied access to the device. In this case alerts can be sent through the Management Component which continuously monitors kernel log files for possible alerts.

This implementation satisfies the general design requirements detailed in Chapter 3.

4.3 Other Components

The finished SCCM system incorporates various other components, which are completed by other members of the project team. As a whole, the system provides most of the capabilities proposed in the design at Chapter 3.

4.3.1 Message Layer – Device Layer Interface

An interface between message layer and device layer enqueues and dequeues the messages, and manages the internal structure on the shared storage hardware. It implements a behavior similar to a circular queue, with necessary extra information in headers. Also synchronization of the messages on disk are handled by this layer, which checks a special area for updates and locations of newly inserted messages, then retrieves the messages and uses the message layer to deliver the traffic to relevant application layer component. This interface was also implemented in C language.

4.3.2 Web Based Management Interface

A cross-platform and cross-browser Web based management interface was implemented in *JavaScript* language. The management interface exposes various configurable aspects of the system to the administrators, provides access to alerts generated by both hosts, and allows

facilities to stop all traffic on the SCCM in case of an emergency. In figure 4.3, a screenshot from the web based management interface can be seen.

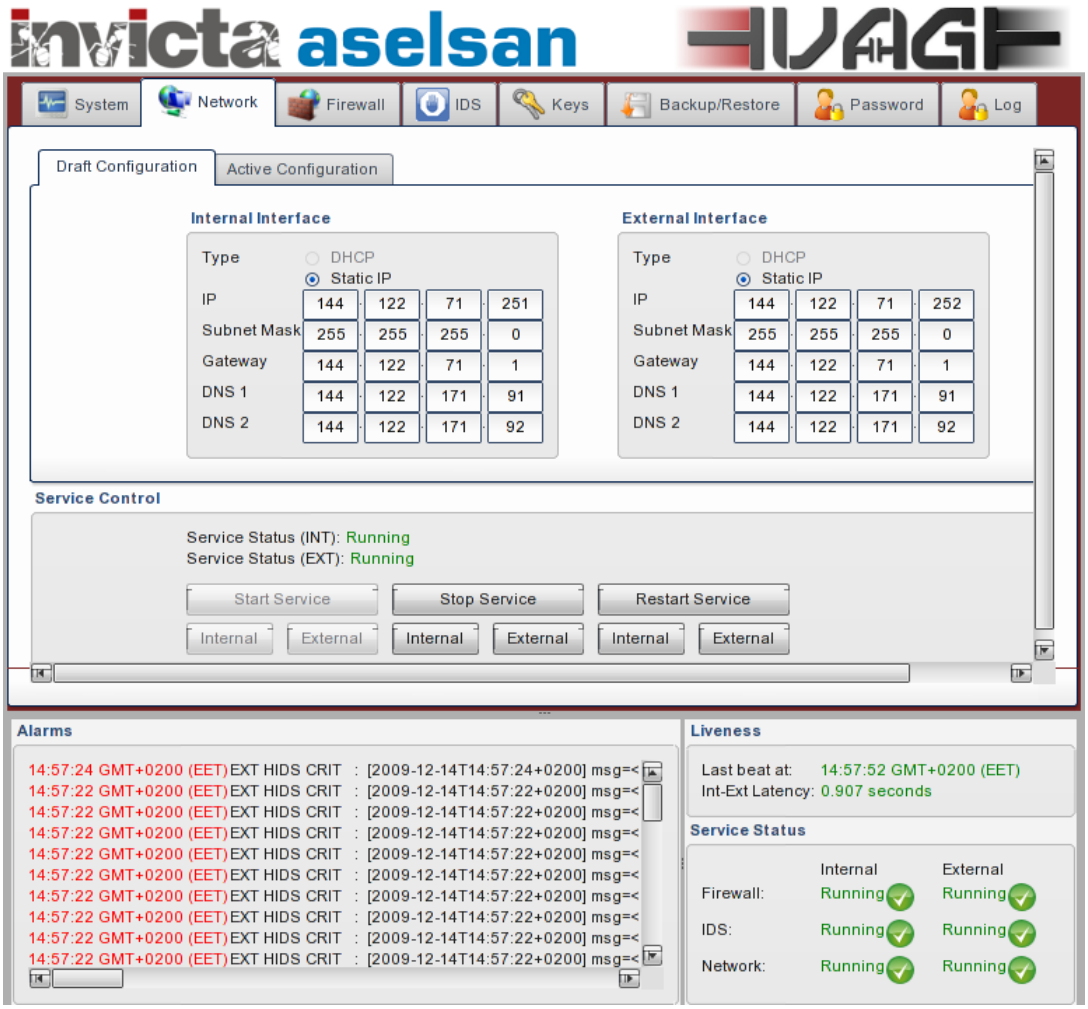

Figure 4.3: Web Based Management Interface

4.3.3 Management Component

Web based management interface relies on two distinct components, the Management Component and the CGI interface between Web based management interface and the management component.

The management component handles all management tasks of the system. It can modify manageable parts of the whole SCCM. It is responsible for collecting alerts from various components of the SCCM and the system log files, and transferring all alerts to the internal

management component, where they can be seen by system administrators.

The main management component exposes a socket based communication channel for management commands. It is possible to manage the system using a command line program connecting to this socket. In this scenario, the administrator must type management commands into the command line.

This separation between the main management component and the web based interface allows possible future implementation of other management interfaces, which may run on the administrators' computers and connect to the main management component directly. Because of this separation a Common Gateway Interface (CGI) backend for the Web based management interface was implemented.

CGI backend provides two-directional communication between the web interface and the main management component. It does not perform any significant operation on the management traffic. It collects the data received from the web based interface via a web server, then passes it to the main management component over the socket. When management component sends data over the socket, the CGI backend reads the data, encapsulates it in a JSON response, and sends it to the web interface via the web server.

4.3.4 Application Layer Component

The design of the SCCM allows for various application layer protocols to be transferred over the SCCM system. The current implementation supports transferring HTTP traffic over the secure communication channel.

The implementation incorporated a modified version of the tinyproxy[5] proxy. The proxy is utilized in two roles within the SCCM. First role is for parsing the HTTP requests and managing connections between the HTTP clients and the SCCM. Then the modifications on it send the parsed contents to the messaging layer via its socket. Second role is for initiating a connection to a HTTP server. The modifications allow the proxy to listen a socket for incoming connections from the message layer, then the contents of the traffic is encapsulated in standards-compliant HTTP headers and sent to the target web server. The proxy manages the connection between HTTP servers and the SCCM.

[5] http://tinyproxy.banu.com/

4.3.5 USB Key Component

The system requires the presence of a specially prepared USB key before starting operation. When instructed by the USB Key Component, the key must be connected to one of the USB ports of both hosts at boot time. This guarantees that an authorized person carrying the key is physically present by the internal and external hosts. The USB key can be used to securely store the cryptographic keys used in various components. After the key is removed, only remaining copy of the cryptographic keys would be the ones in process, this would enhance the secrecy of keys, as an adversary will not have immediate access to the keys and will have to extract it from process memory. This operation requires elevated privileges, and also requires a long analysis of the process memory, which result in an alert to be triggered as a consequence of adversary activities, resulting in detection of the adversary.

4.3.6 Packaging and Installation

The SCCM components are packaged into *.deb* packages that can be installed on Debian Linux systems. The packages contain the binaries of system components, configuration files and other necessary files for component operations.

4.4 Deployments

The system was deployed on two disjoint sets of hardware, their details can be seen in Table 4.1.

Table 4.1: Systems that VAG have been successfully deployed on

System	Host Hardware	Storage Interface on Hosts	Shared Storage System
1	Intel Core 2 Quad CPU, 1 GB RAM	Adaptec AHA-2940U/UW/D / AIC-7881U SCSI HBA	IBM EXP 400 storage enclosure with 6×148 GB U320 SCSI disks
2	Intel Core 2 Quad CPU, 2 GB RAM	QLogic 2500 Fiber Channel HBA	EMC storage with 12×300 GB SAS disks

This shows that the proposed mechanisms are able to run on a variety of hardware configura-

tions, using off-the-shelf components.

4.5 Performance

As the system aims for near real time operation, performance of the system was a key concern during implementation. All code involving payload data transfer was written in low level C language, while keeping the overhead imposed by the system at a minimum. Use of external libraries is also limited to prevent performance implications from external sources.

4.5.1 Performance Measurements

The following measurements were obtained on systems listed on Table 4.1, using a dedicated disk for each direction of traffic. A total of 10 test runs for each configuration was performed and averaged. The test runs consisted of downloading of a 5 MB file and a 100 MB file over the network, using *wget* HTTP client program.

To compare the performance penalty of deploying the SCCM systems, the download speeds on the same network, without any SCCM components, are included.

Tests results can be found at Table 4.2. The results point out that there is a significant performance loss when SCCM is deployed, this result is understandable as the speed of the data transfer is limited to speed of shared storage used in the SCCM. Also, the results show that the performance of the system is highly dependent on the choice of storage system, disks and storage interface.

Table 4.2: Performance results of test runs on two distinct systems

Configuration	Average speed of 10 runs (5MB file)	Average speed of 10 runs (100MB file)
Unfiltered Network	7.91 MB/s	7.72 MB/s
System 1	133 KB/s	129 KB/s
System 2	2210 KB/s	2100 KB/s

4.6 Security Assessment

Compete compromise of external host does not bring grave security implications to high-security network. Thanks to the setup of the deployed example system, if the attacker finds out the architecture of the system after analyzing, the attacker can only begin sending application level queries to the high security network.

Exploiting application level vulnerabilities is a possibility. As the protection of the application level traffic is achieved by integrating industry standard IDS software, weak points of the software may allow some application layer attacks into the protected network.

It is a possibility that internal host can be compromised by exploiting vulnerabilities in the code of various modules of the secure communication mechanisms. The implementation was completed with giving security utmost importance. Standard software engineering practices of code reviews, thorough bound and error checking were followed. Security audits were performed on the code paths involved with network traffic. As a result of these activities the mechanisms were found to be sufficiently secure, allowing deployment in military networks.

The currently deployed configuration (*"System 2"* in Table 4.1) is tested for possible security issues by a team at National Research Institute of Electronics and Cryptology (UEKAE), an affiliate of the Scientific and Technological Research Council of Turkey (TÜBİTAK). As a result of this test, set of mechanisms proposed in this theses are found to be secure to a degree that allows them to be deployed on military networks.

4.6.1 Data Flow Through the SCCM

The complete SCCM system consists of a number of components. The data flow path that is followed throughout the system is listed at Table 4.3.

The data flow path represents flow of data from one network to another. It begins by receiving of the packet on one hosts interface, follows the SCCM components until the data is written to shared storage. Then, the path of the data at the other host is followed, until the complete network packet leaves the host's network interface.

Various protection mechanisms that are previously discussed are related to the points at the

data flow, where the protection is applied.

Table 4.3: Data flow path through the SCCM

Event	Protection provided by SCCM
Network packet is received at the network interface	Firewall and NIDS detect and block known and possibly unknown attacks.
Data arrives at application layer listener	All lower layers are stripped, leaving only application layer data
Message layer handles and multiplexes messages, then dispatches to Device Layer	Message Layer – Device Layer trust via IOCTL provides protection of Device Layer
Device layer writes the data to shared storage	Data is encrypted & signed before being written to prevent unauthorized access
The data crosses shared storage hardware, reaches other host	–
Device layer reads the data from shared storage	The data is decrypted and its signature is validated to ensure operation over a non-compromised channel
Message layer fetches data from Device layer, and after multiplexing sends it to application layer component	Message Layer – Device Layer trust via IOCTL provides protection of Device Layer
Application Layer component encapsulates data with protocol headers & trailers.	–
Network packet is sent through the network interface	Firewall and NIDS detect and block known and possibly unknown attacks.

4.6.2 Security Comparison

The security benefits of SCCM have been discussed throughout this study. In this section, comparison of SCCM system with a basic, and an advanced security device will be conducted.

The basic security device is defined as a firewall product integrated into the operating system, which does not contain advanced security features. The host running the firewall is assumed to be running some network services that are to be protected from the external network.

The advanced security device is defined as a stand alone security hardware, including application level filtering, Intrusion Detection Systems and other advanced security features.

While the actual security levels provided by a security system depends heavily on the configuration of the device, for the following comparison it is assumed that each system is configured optimally to provide maximum security.

The comparison is detailed in Table 4.4. As can be seen from the table, SCCM has distinct advantages that are not provided by other current systems.

Table 4.4: Comparison of security features provided by three different systems

Security Consideration	Basic Security Device	Advanced Security Device	The SCCM System
Protection from unauthorized incoming connections	+	+	+
Protection from unauthorized outgoing connections	+	+	+
Protection from application layer attacks	-	+	+
Providing alerts for security incidents	-	+	+
Minimum network performance overhead	+	-	-
Prevent direct compromise of network services in case of a security device compromise	-	+	+
Continue protecting internal network in case of a security device compromise	-	-	+
Complete networking stack isolation between networks	-	-	+
Complies with network seperation policies of high security organizations	-	-	+

4.7 Other Deployment Schemes

The design of the mechanisms is flexible enough that deployment schemes other than the one described in this chapter are possible. According to needs, requirements and restrictions of the organization that intends to use the system, the components that constitute the secure communication channel mechanisms can be realized and deployed in different variations.

If cost is a limiting factor in a deployment, it is possible to run the proposed set of mechanisms on a single host, using virtualization technologies. In this configuration, internal and external hosts manifest as virtual machines. Two distinct network interfaces on the physical machine are each dedicated to one and only one virtual host. The shared storage is simulated by a single virtual disk simultaneously connected to both virtual machines.

Obviously, use of virtual machines leads to a different consideration in security, namely the security of the virtual machine engine used. In case of a compromise of a *–now virtual–* host, possible security vulnerabilities in virtual machine engine are open to exploitation by

adversaries. It can be said that the virtual machines, and as a corollary, the networks are isolated as much as the virtual machine engine succeeds in isolating them.

Other deployment schemes may involve replacing the shared disk storage system. To achieve a faster data transfer rate the storage may be replaced with a single connection that transfers the application level payload directly. This reduces the *psychological network isolation effect* that is elicited by the revolving disks. Various data transfer technologies can be employed, with their speeds listed at Table 4.5 [25] [26] [27] [28].

Table 4.5: Comparison of speeds of various data transfer technologies

Technology	Data transfer speed
USB 2.0	60 MB/s
Ultra-320 SCSI	320 MB/s
Fibre Channel 8GFC	1600 MB/s
Infiniband 4X QDR	4000 MB/s

CHAPTER 5

Conclusion and Future Directions

In this theses work, a set of mechanisms that allow network traffic to and from high security networks are designed and implemented. The mechanisms do not require human intervention for data transfer. The design is proved to be realizable on a variety of off-the-shelf hardware. The security implications of the mechanisms are elaborated on. The proposed mechanisms provide an alternative to complete network isolation vs. single control point for network traffic.

Having the networks completely isolated is the most secure method, but if external communication without human intervention is a requirement, the proposed communication mechanism can be considered more secure than traditional security devices (router and firewall combination). Also the mechanisms can be regarded as an improvement over current network security devices and software.

Compete compromise of external host does not bring grave security implications to high-security network. Thanks to the setup of the deployed example system, if the attacker finds out the architecture of the system after analyzing, the attacker can only send application level queries to the high security network.

The complete set of mechanisms proposed in this theses introduces a long chain of security measures against attacks. Some of the protection measures introduced require a large time frame to circumvent. By increasing the time required before a successful penetration, probability to detect the attack is increased.

Possible usage areas of these secure communication mechanisms are numerous; military networks where security is of utmost importance, companies and governmental organizations

in need of high-security networks for classified data. Also, mission-critical systems on ships, planes, space vessels would benefit from the isolation provided by the proposed mechanisms.

Deployments that are being used in real world scenarios will provide valuable feedback on future directions of the SCCM.

Implementation of other application layer protocols as modules can increase the real world application areas of the mechanisms.

Providing Quality of Service (*QOS*) facilities can allow prioritizing some application layer protocols, some connections between selected ends or alerts. This capability may be implemented in software as a whole, or a dedicated set of disks may be reserved for different QOS levels.

The root of trust concept can be strengthened further. One approach would be to store the critical SCCM system binaries in the USB Key, which is required to be present at boot time. This would reduce the risks of executing a maliciously modified binary file instead of the genuine binary file. The USB Key component would extract the binary from USB Key, which contains the binary in an obfuscated form, then execute the binary. After the disconnection of the USB key, a malicious attacker would not be able to modify SCCM system binaries directly. A method of attack at this scheme would involve the modification of USB Key Component first, changing it such that it modifies the binary in the time frame between reading from USB key and executing it.

Another approach for strengthening the trusted root concept would be introducing networked boot servers. Two boot servers would be connected to SCCM hosts, one for each hosts. The connections would be on dedicated network interfaces, with a single cable, such that no external access is present between SCCM host and the boot server. The SCCM host would be configured to boot from network, which is viable through Preboot Execution Environment *(PXE)* facilities of modern hardware. The boot server would include a Dynamic Host Configuration Protocol *(DHCP)* server for assigning an IP address to SCCM host, and a Trivial File Transfer Protocol *(TFTP)* server to transfer the operating system kernel to the SCCM host. Booting this server supplied kernel would increase confidence in executing an unmodified kernel. To further enhance this approach, an organizational security policy can rule that the boot servers should be connected to SCCM hosts only during the system boot, and no

other network connection is allowed to and from boot servers. This approach would deny an attacker to replace the operating system kernel remotely.

The Linux Security Modules *(LSM)* infrastructure, which is used in the kernel layer, can be exploited further. As LSM is a flexible infrastructure, it exposes many security hooks within kernel operations. These hooks can be utilized to control access to and from various connection points between layers of the system, these include sockets and message queue facilities. The LSM facility at the kernel layer can also be used for disallowing the modification of SCCM binaries, strengthening the trusted root concept.

Another approach in using LSM facility for increased security would be to disallow analyzing of process execution and process memory. A probable weak point of the SCCM is, if the adversary gains elevated privileges on the system, cryptographic keys can be collected by analyzing process memory, details and inner workings of the system may be identified by tracing processes. As the system aims to hide internal details as much as possible, to increase the probability of an alert being generated after malicious activity, depriving the adversary of detailed knowledge about the system is valuable. The LSM infrastructure can be facilitated to prevent analysis of system components, mitigating a possible weak point in the system design.

REFERENCES

[1] A.D. Keromytis and J.L. Wright. Transparent network security policy enforcement. In *Proceedings of the Annual USENIX Technical Conference*, pages 215–226. Society Press, 2000.

[2] Attila Özgit. Virtual Air Gap, 2009. PCT/TR2008/000099, WO/2009/075656.

[3] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, 1989.

[4] D.B. Chapman. Network (in) security through IP packet filtering. In *Proceedings of the Third UNIX Security Symposium*, 1992.

[5] M. Roesch. Snort–lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*. USENIX, 1993.

[6] N. Desai. Intrusion prevention systems: The next step in the evolution of IDS. *Security Focus*, 27 Feb 2003 (accessed 30 August 2009). `http://securityfocus.com/printable/infocus/1670`.

[7] K. Gennuso. Disconnect from the internet – whale's e-gap in-depth, 13 September 2001 (accessed 20 August 2009). `http://www.sans.org/reading_room/whitepapers/firewalls/802.php`.

[8] M. Hurley. Network air gaps – drawbridge to the backend office, 4 April 2001 (accessed 27 August 2009). `http://www.sans.org/infosecFAQ/firewall/gaps.htm`.

[9] L. Catuogno and I. Visconti. A format-independent architecture for run-time integrity checking of executable code. *Lecture notes in computer science*, pages 219–233, 2003.

[10] A. Apvrille, D. Gordon, S. Hallyn, M. Pourzandi, and V. Roy. Digsig: Run-time authentication of binaries at kernel level. In *Proceedings of LISA*, 2004.

[11] R.G. Atkinson, J.W. Kelly Jr, B.W. Tuttle, R.M. Price, and R.P. Reichel. Embedding certifications in executable files for network transmission, April 2 2002. US Patent 6,367,012.

[12] G.I. Davida, Y.G. Desmedt, and B.J. Matt. Defending systems against viruses through cryptographic authentication. In *1989 IEEE Symposium on Security and Privacy, 1989. Proceedings.*, pages 312–318, 1989.

[13] E. Zadok, I. Badulescu, and A. Shender. Cryptfs: A stackable vnode level encryption file system. Technical report, Citeseer, 1998.

[14] RL Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. 1978.

[15] C. Adams and S. Lloyd. *Understanding public-key infrastructure: concepts, standards, and deployment considerations*. Macmillan Technical Publishing, 1999.

[16] K. Srinivasan and S. Mitchell. State Based Key Hop (SBKH) Protocol. In *Wireless 2004 Conference*, jul 2004.

[17] M. Beck, H. Bohme, U. Kunitz, R. Magnus, M. Dziadzka, and D. Verworner. *Linux kernel internals*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1996.

[18] D. Bovet and M. Cesati. *Understanding the Linux kernel*. Oreilly & Associates Inc, 2005.

[19] E.S. Raymond. *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2001.

[20] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *Proceedings of the 11th USENIX Security Symposium*, volume 2. Citeseer, 2002.

[21] corbet. securityfs. *LWN.net*, 27 September 2005. `http://lwn.net/Articles/153366/`.

[22] Linus Torvalds. Linux kernel source tree, 13 December 2009 (accessed 13 Dec 2009). `http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=tree;f=security;h=234133d00e15a1de77834b3750c5d48888001125;hb=HEAD`.

[23] M. Bauer. An Introduction to Novell AppArmor. *Linux Journal*, (148), 2006.

[24] Babkin S. File descriptors and multithreaded programs, 07 November 2008 (accessed 06 Dec 2009). `http://www.ddj.com/hpc-high-performance-computing/212001285`.

[25] USB Implementers Forum. Universal serial bus revision 2.0 specification, 27 April 2000 (accessed 11 Sep 2009).

[26] L.P. Hewlett-Packard Development Company. Ultra320 SCSI: The Next Generation of SCSI Technology, March 2003.

[27] Fibre Channel Industry Association. Roadmap, accessed 11 Sep 2009. `http://www.fibrechannel.org/roadmaps`.

[28] InfiniBand Trade Association. Infiniband architecture specification, October 24 2000. Release 1.0.